MACHINE LEARNING

(Wine Quality Prediction)

Summer Internship Report Submitted in partial fulfillment of the requirement for

undergraduate degree of

Bachelor of Technology

In

Electronics and Communication Engineering

By

M.Karthik Sai Phanindra

221710405015

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

DECLARATION

I submit this industrial training work entitled “Wine Quality Prediction” to GITAM(Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of “Bachelor of Technology” in “Electronics and Communication Engineering”. I declare that it was carried out independently by me under the guidance .

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:	**M.Karthik Sai Phanindra**
Hyderabad	**221710405015**
Date: 12.07.2020	

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated: **12.07.2020**

CERTIFICATE

This is to certify that the Industrial Training Report entitled "**Wine Quality Prediction**" is being submitted by M.Karthik Sai Phanindra (221710405015) in partial fulfilment of the requirement for the award of Bachelor of Technology in **Electronics & Communication Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2020-21.

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

	Dr.K.Manjunathachari
Assistant Professor	Professor and HOD
Department of ECE	Department of ECE

Certificate

ACKNOWLEDGEMENT

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

I would like to thank respected Dr. N. Siva Prasad, Pro Vice Chancellor, GITAM Hyderabad and Dr. N. Seetharamaiah, Principal, GITAM Hyderabad.

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

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

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

M.Karthik Sai Phanindra
221710405015

ABSTRACT

Modeling the complex human taste is an important focus in wine industries. The main purpose of this study was to predict wine quality based on physicochemical data. This study was also conducted to identify outlier or anomaly in sample wine set in order to detect adulteration in the quality of wine, without the use of sommelier as it involves lots of investment.

In this project, two large separate datasets are used, which contains 1, 599 instances for red wine and 4, 898 instances for white wine with 11 attributes of physicochemical data such as alcohol, PH, sulfates etc. Two classification algorithms, Decision tree and Random Forest are applied on the dataset and the performance of these two algorithms is compared. Results showed that Random Forest outperformed Decision Tree techniques particularly in red wine, which is the most common type. The study also showed that two attributes, alcohol and volatile-acidity contribute highly to wine quality. White wine is also more sensitive to changes in physicochemistry as opposed to red wine, hence higher level of handling care is necessary.

This concludes that classification approach will give rooms for corrective measure to be taken in effort to increase the quality of wine during production.

INDEX

FIGURE INDEX

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

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

1.2 IMPORTANCE OF MACHINE LEARNING:

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

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

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

The process flow depicted here represents how machine learning works

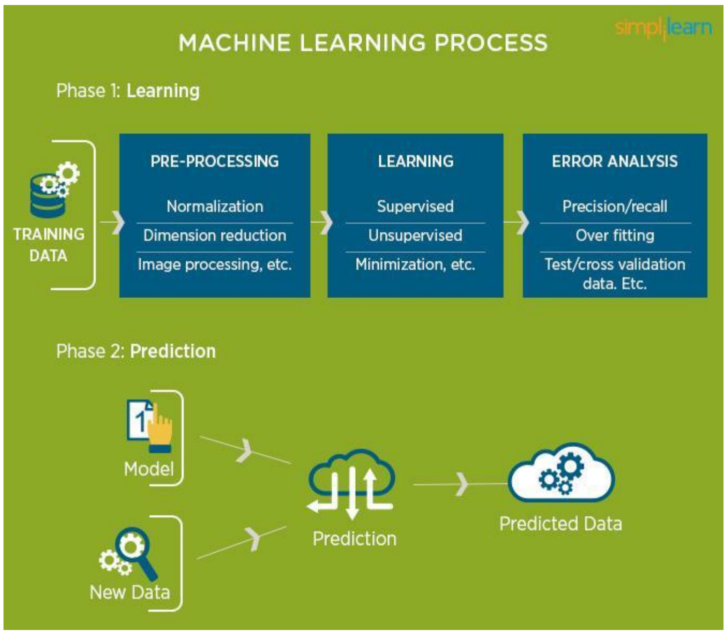


Figure 1.2.1: The Process Flow

1.3 USES OF MACHINE LEARNING:

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

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

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

1.4 TYPES OF LEARNING ALGORITHMS:

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

1.4.1 Supervised Learning:

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

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

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

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

1.4.2 Unsupervised Learning:

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

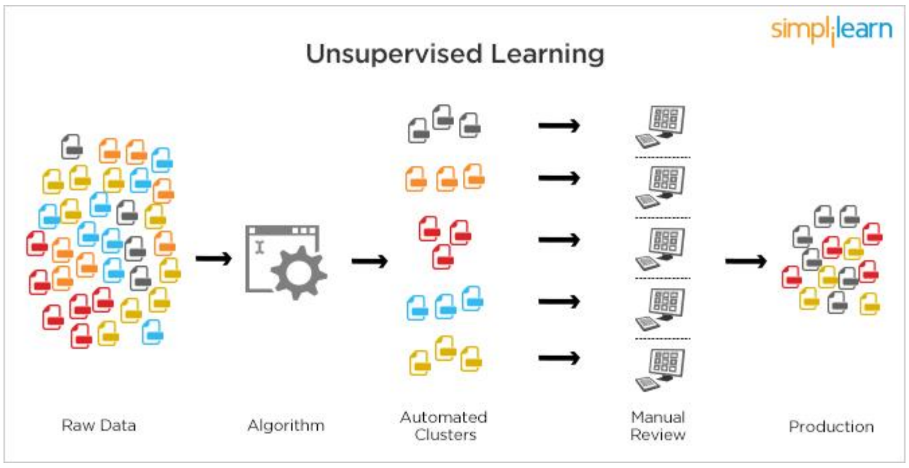


Figure 1.4.2.1: Unsupervised Learning

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

1.4.3 Semi Supervised Learning:

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

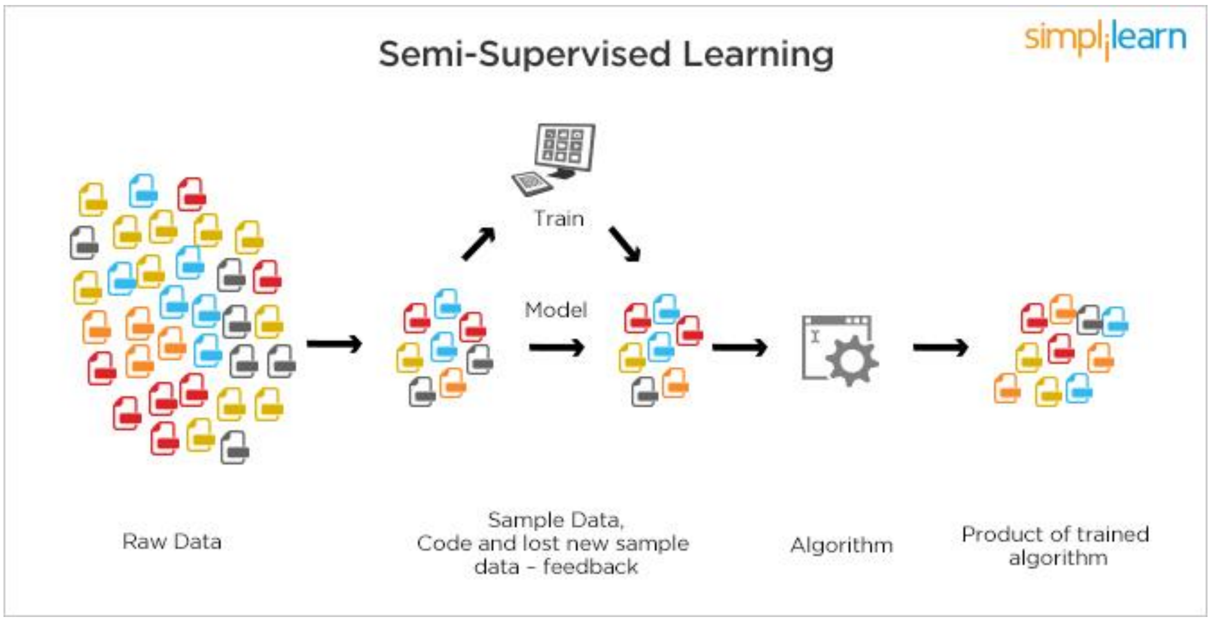

simpl;learn
Semi-Supervised Learning
Train
Model
Raw Data
Sample Data, Code and lost new sample data - feedback
Algorithm
Product of trained algorithm

Figure 1.4.3.1: Semi Supervised Learning

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

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

Deep learning, on the other hand, uses advanced computing power and special 5 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

DEEP LEARNING

2.1 INTRODUCTION:

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

2.2 IMPORTANCE OF DEEP LEARNING:

The ability to process large numbers of features makes deep learning very powerful when dealing with unstructured data. However, deep learning algorithms can be overkill for less complex problems because they require access to a vast amount of data to be effective. For instance, ImageNet, the common benchmark for training deep learning models for comprehensive image recognition, has access to over 14 million images.

If the data is too simple or incomplete, it is very easy for a deep learning model to become overfitted and fail to generalize well to new data. As a result, deep learning models are not as effective as other techniques (such as boosted decision trees or linear models) for most practical business problems such as understanding customer churn, detecting fraudulent transactions and other cases with smaller datasets and fewer features. In certain cases like multi classification, deep learning can work for smaller, structured datasets.

2.3USES OF DEEP LEARNING:

2.3.1 Self-Driving Cars:

Deep Learning is the force that is bringing autonomous driving to life. A million sets of data are fed to a system to build a model, to train the machines to learn, and then test the results in a safe environment. The Uber Artificial Intelligence Labs at Pittsburg is not only working on making driverless cars humdrum but also integrating several smart features such as food delivery options with the use of driverless cars. The major concern for autonomous car developers is handling unprecedented scenarios.

A regular cycle of testing and implementation typical to deep learning algorithms is ensuring safe driving with more and more exposure to millions of scenarios. Data from cameras, sensors, geo-mapping is helping create succinct and sophisticated models to navigate through traffic, identify paths, signage, pedestrian-only routes, and real-time elements like traffic volume and road blockages.

According to Forbes, MIT is developing a new system that will allow autonomous cars to navigate without a map as 3-D mapping is still limited to prime areas in the world and not as effective in avoiding mishaps. CSAIL graduate student Teddy Ort said, “The reason this kind of ‘map-less’ approach hasn’t really been done before is because it is generally much harder to reach the same accuracy and reliability as with detailed maps. A system like this that can navigate just with on-board sensors shows the potential of self-driving cars being able to actually handle roads beyond the small number that tech companies have mapped.”

2.3.2 Virtual Assistants:

The most popular application of deep learning is virtual assistants ranging from Alexa to Siri to Google Assistant. Each interaction with these assistants provides them with an opportunity to learn more about your voice and accent, thereby providing you a secondary human interaction experience. Virtual assistants use deep learning to know more about their subjects ranging from your dine-out preferences to your most visited spots or your favorite songs.

They learn to understand your commands by evaluating natural human language to execute them. Another capability virtual assistants are endowed with is to translate your speech to text, make notes for you, and book appointments. Virtual assistants are literally at your beck-and-call as they can do everything from running errands to auto-responding to your specific calls to coordinating tasks between you and your team members.

With deep learning applications such as text generation and document summarizations, virtual assistants can assist you in creating or sending appropriate email copy as well.

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

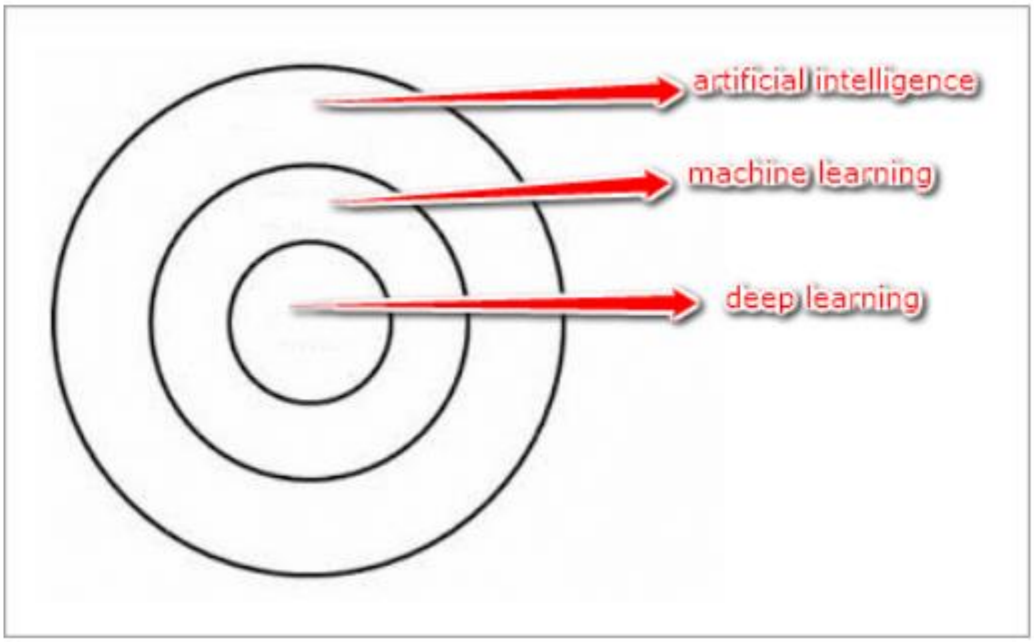


Figure 2.4.1: Relation

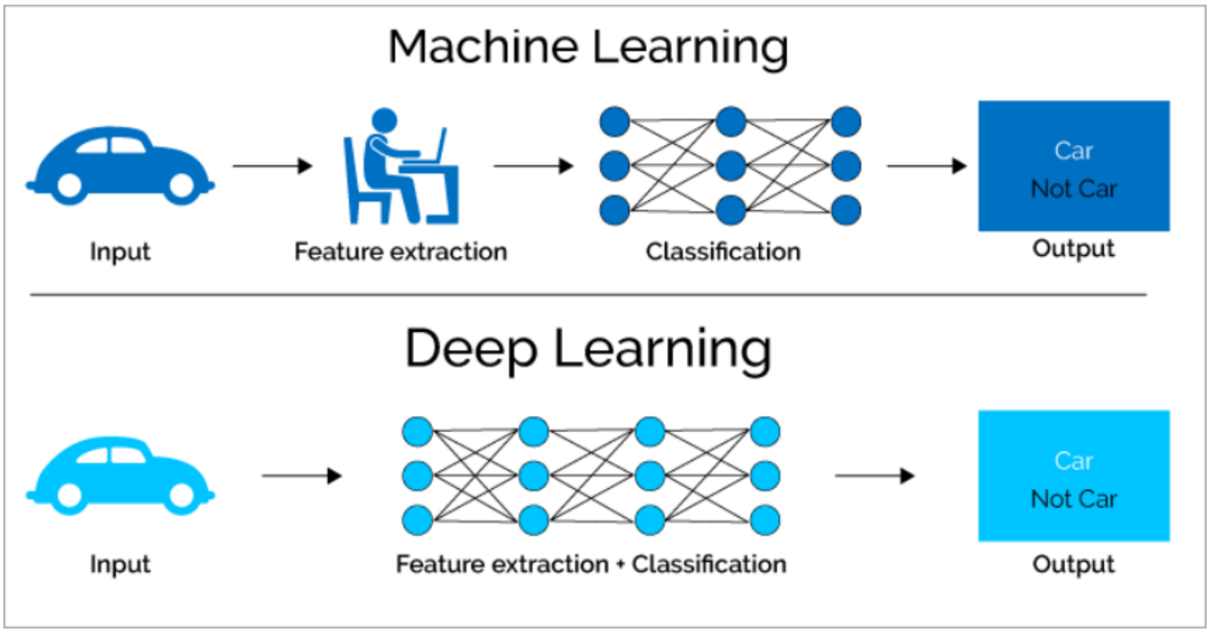


Figure 2.4.2: Machine Learning vs Deep Learning

Machine Learning comprises of the ability of the machine to learn from trained data set and predict the outcome automatically. It is a subset of artificial intelligence.

Deep Learning is a subset of machine learning. It works in the same way on the machine just like how the human brain processes information. Like a brain can identify the patterns by comparing it with previously memorized patterns, deep learning also uses this concept.

Deep learning can automatically find out the attributes from raw data while machine learning selects these features manually which further needs processing. It also employs artificial neural networks with many hidden layers, big data, and high computer resources.

Data Mining is a process of discovering hidden patterns and rules from the existing data. It uses relatively simple rules such as association, correlation rules for the decision-making process, etc. Deep Learning is used for complex problem processing such as voice recognition etc. It uses Artificial Neural Networks with many hidden layers for processing.

At times data mining also uses deep learning algorithms for processing the data.

CHAPTER 3

PYTHON

Basic programming language used for machine learning is: PYTHON

3.1 INTRODUCTION TO PYHTON:

● Python is a high-level, interpreted, interactive and object-oriented scripting language.

● Python is a general purpose programming language that is often applied in scripting roles

● Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.

● Python is Interactive: You can sit at a Python prompt and interact with the interpreter
directly to write your programs.

● Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

3.2 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

3.3 FEATURES OF PYTHON:

● 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's source code is fairly easy-to-maintaining.

● A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

● Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

● Extendable: You can add low-level modules to the Python interpreter. These enable programmers to add to or customize their tools to be more efficient.

● Databases: Python provides interfaces to all major commercial databases.

● GUI Programming: Python supports GUI applications that can be created and ported
to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

3.4 HOW TO SETUP PYTHON:

● Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

● The most up-to-date and current source code, binaries, documentation, news, etc., is
available on the official website of Python.

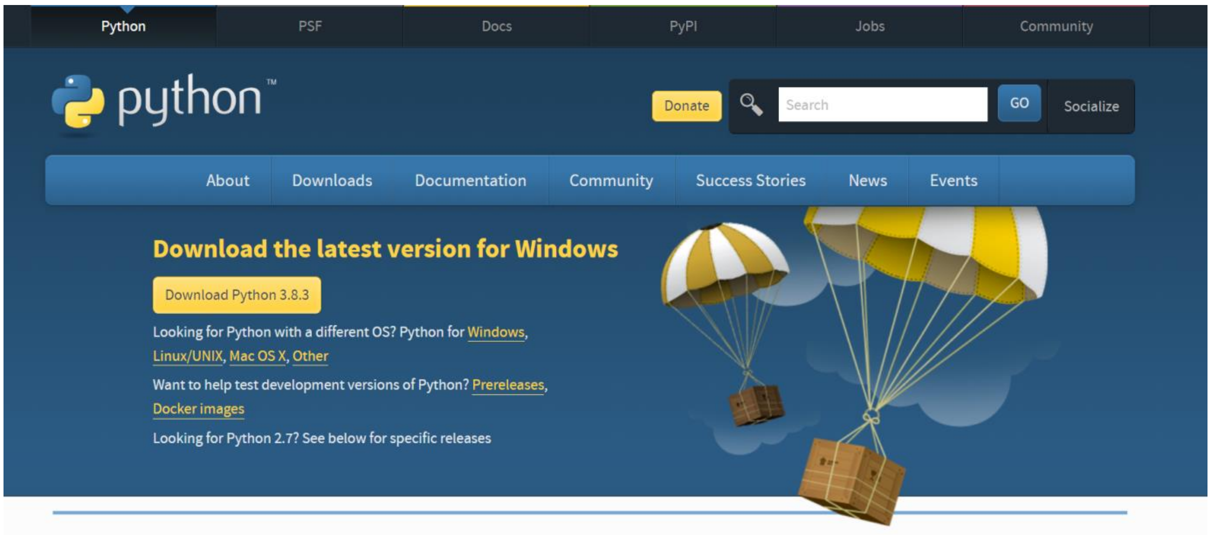
3.4.1 Installation(using python IDLE):

● 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.



Download the latest version for Windows

Download Python 3.8.3

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

Figure 3.4.1.1: Python download

3.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.

- Anaconda is a free open source distribution of python for large scale data processing,
  predictive analytics and scientific computing.

- Conda is a package manager quickly installs and manages packages.

- In WINDOWS:

- In windows

  - Step 1: Open Anaconda.com/downloads in web browser.

  - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)

  - Step 3: select installation type(all users)

  - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python
    3.4) next click install and next click finish

  - Step 5: Open jupyter notebook (it opens in default browser)

Figure 5 : Anaconda download
Figure 6 :Jupyter notebook

3.5 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 –

o Numbers

o Strings

o Lists

o Tuples

o Dictionary

3.5.1 Python 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).

3.5.2 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.

3.5.3 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 11 ([]).

● 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.

3.5.4 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.

3.5.5 Python Dictionary:

● Python's dictionaries are kind of hash table type. They work like associative arrays 12 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.

3.6 PYTHON FUNCTION:

3.6.1 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.

3.6.2 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.

3.7 PYTHON USING OOP's CONCEPTS:

3.7.1 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.

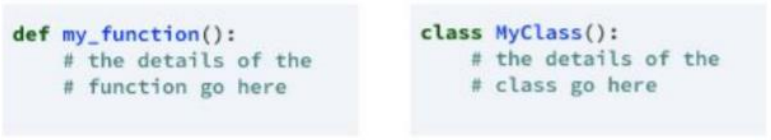
● Data member: A class variable or instance variable that holds data associated with a class and its objects.

● Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

● Defining a Class:

o We define a class in a very similar way how we define a function.

o Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.



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

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

Figure 3.7.1.1: Defining a Class

3.7.2 __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.

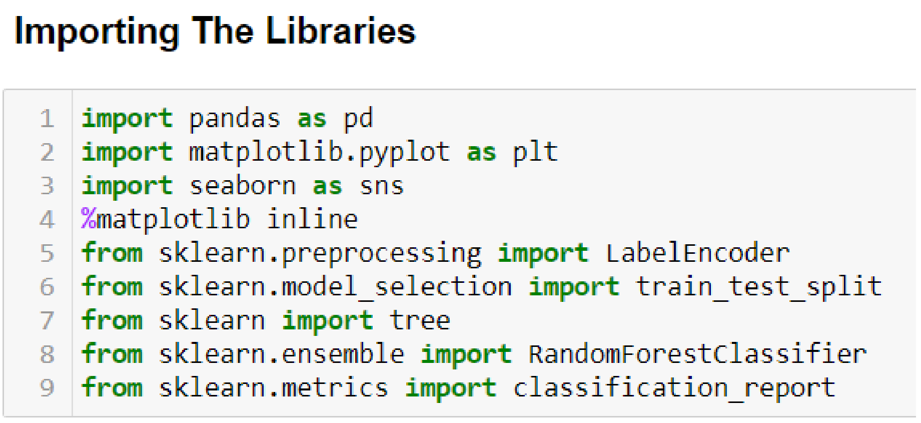
● The init method has a special name that starts and ends with two underscores:__init__().

CHAPTER 4

WINE QUALITY PREDICTION

4.1 PROJECT REQUIREMENTS:

4.1.1 Packages used:

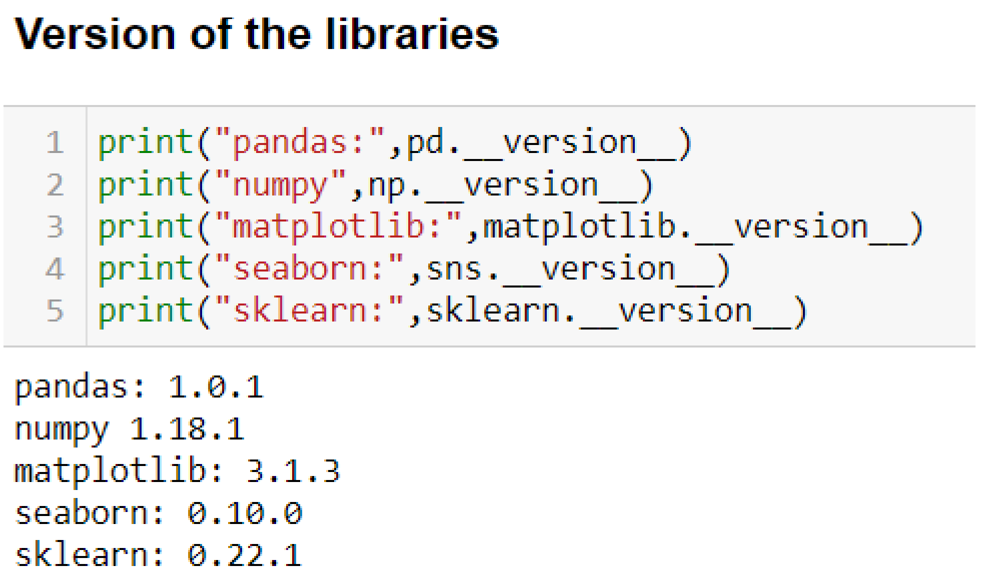


Importing The Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

Figure 4.1.1.1: Importing Libraries

4.1.2 Versions of the packages:



Version of the libraries

```
print("pandas:",pd.__version__)
print("numpy",np.__version__)
print("matplotlib:",matplotlib.__version__)
print("seaborn:",sns.__version__)
print("sklearn:",sklearn.__version__)
```

```
pandas: 1.0.1
numpy 1.18.1
matplotlib: 3.1.3
seaborn: 0.10.0
sklearn: 0.22.1
```

Figure 4.1.2.1: Version of the packages

4.1.3 Algorithms used:

4.1.3.1 Decision Tree Classifier:

Decision Tree Analysis is a general, predictive modelling tool that has applications spanning a number of different areas. In general, 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. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

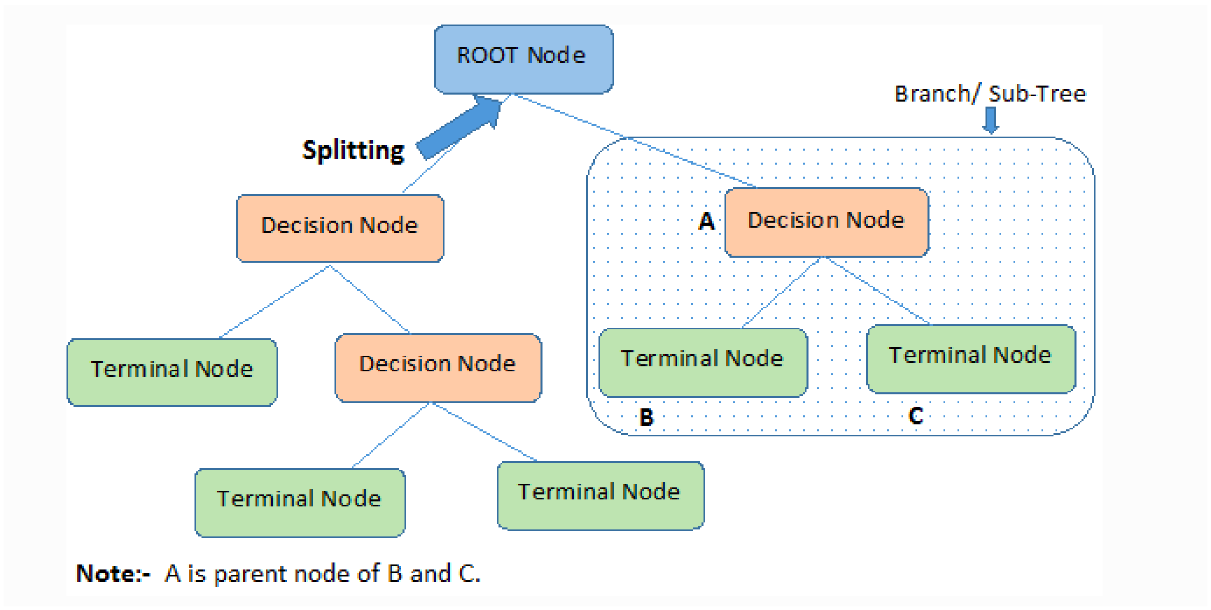


Figure 4.1.3.1.1: Decision Tree Classifier

4.1.3.2 Random forest Classifier:

Random forest is a supervised learning algorithm which is used for both classification as well as regression. it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

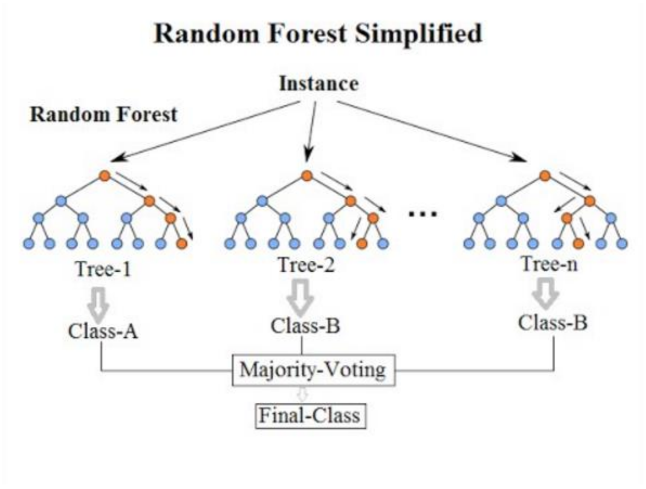


Figure 4.1.3.2: Random forest Classifier

4.2 PROBLEM STATEMENT:

To Predict the Quality of wine based on the various chemical compositions of the wine, using Machine Learning Algorithms**.**

4.3 DATA SET DESCRIPTION:

The given data set consists of the following parameters:

1. Fixed Acidity
2. Volatile Acidity
3. Citric Acid
4. Residual Sugar
5. Chlorides
6. Free Sulfur Dioxide
7. Total Sulfur Dioxide
8. Density
9. PH
10. Sulphates
11. Alcohol
12. Quality
13. Color

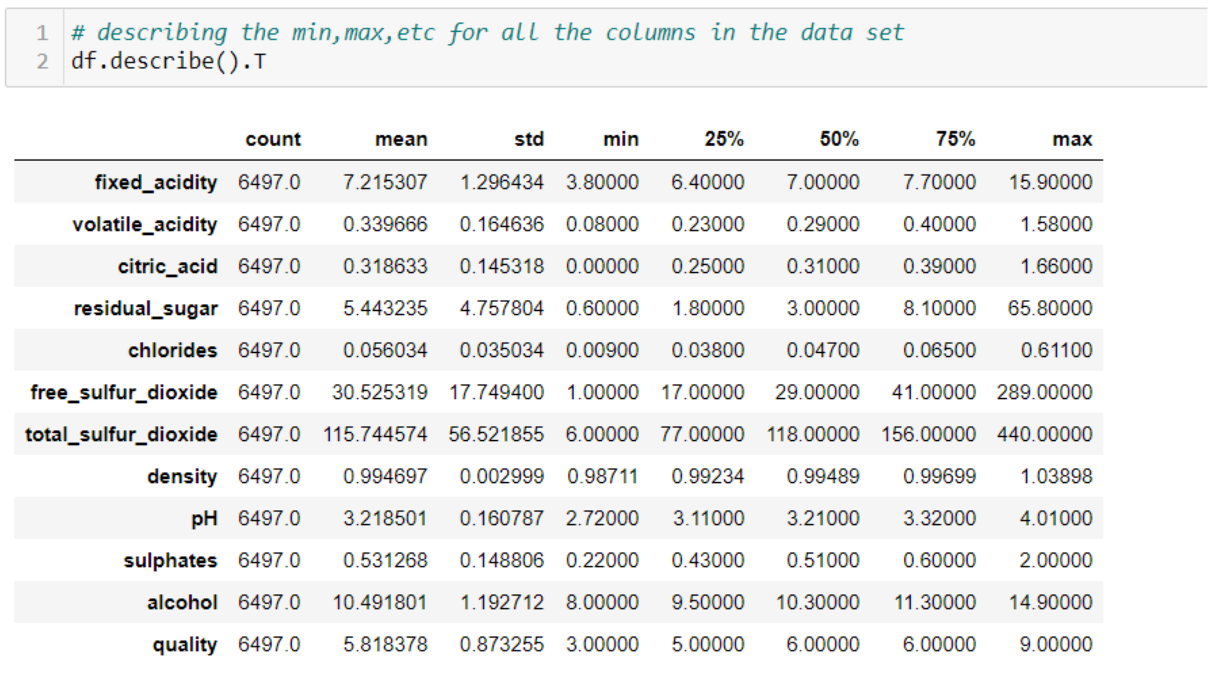
4.4 OBJECTIVE OF THE CASE STUDY:

To get a better understanding about the quality of the wine. The quality of the depends of the following factors like , Fixed Acidity , Volatile Acidity ,Citric Acid, Residual Sugar, Chlorides , Free Sulfur Dioxide , Total Sulfur Dioxide , Density , PH ,Sulphates , Alcohol, Color . Based on the above parameters that are mentioned we should be able to predict the quality of wine.

CHAPTER 5

DATA PREPROCESSING

5.1 STATISTICAL ANALYSIS:



```python
# describing the min,max,etc for all the columns in the data set
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
fixed_acidity	6497.0	7.215307	1.296434	3.80000	6.40000	7.00000	7.70000	15.90000
volatile_acidity	6497.0	0.339666	0.164636	0.08000	0.23000	0.29000	0.40000	1.58000
citric_acid	6497.0	0.318633	0.145318	0.00000	0.25000	0.31000	0.39000	1.66000
residual_sugar	6497.0	5.443235	4.757804	0.60000	1.80000	3.00000	8.10000	65.80000
chlorides	6497.0	0.056034	0.035034	0.00900	0.03800	0.04700	0.06500	0.61100
free_sulfur_dioxide	6497.0	30.525319	17.749400	1.00000	17.00000	29.00000	41.00000	289.00000
total_sulfur_dioxide	6497.0	115.744574	56.521855	6.00000	77.00000	118.00000	156.00000	440.00000
density	6497.0	0.994697	0.002999	0.98711	0.99234	0.99489	0.99699	1.03898
pH	6497.0	3.218501	0.160787	2.72000	3.11000	3.21000	3.32000	4.01000
sulphates	6497.0	0.531268	0.148806	0.22000	0.43000	0.51000	0.60000	2.00000
alcohol	6497.0	10.491801	1.192712	8.00000	9.50000	10.30000	11.30000	14.90000
quality	6497.0	5.818378	0.873255	3.00000	5.00000	6.00000	6.00000	9.00000

Figure 5.1.1: Describing the Mean , Min , Max, Std etc

5.2 GENERATING PLOTS:

Visualize the data between Target and the Features:

```
sns.barplot(x='category',y='fixed_acidity',data=df).
set_title("Fixed acidity with respect to calegory")
```

```
Text(0.5, 1.0, 'Fixed acidity with respect to calegory')
```

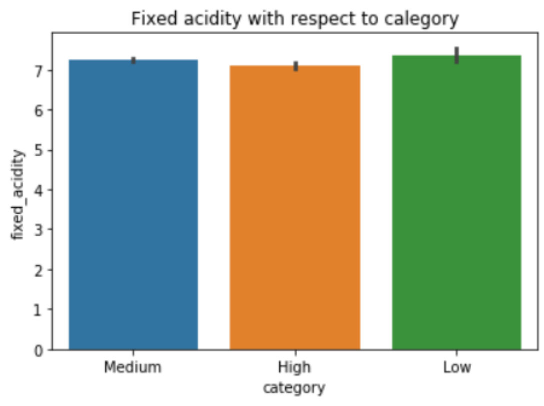


Figure 5.2.1: Variation of Fixed Acidity with respect to Category

Here in the above graph we can see that there is no large difference between the various wine quality. Fixed acidity of “low” is higher , in case of “medium” is moderate and in case of “high” its low .

```
sns.barplot(x='category',y='volatile_acidity',data=df).
set_title("Volatile acidity with respect to calegory")
```

```
Text(0.5, 1.0, 'Volatile acidity with respect to calegory')
```

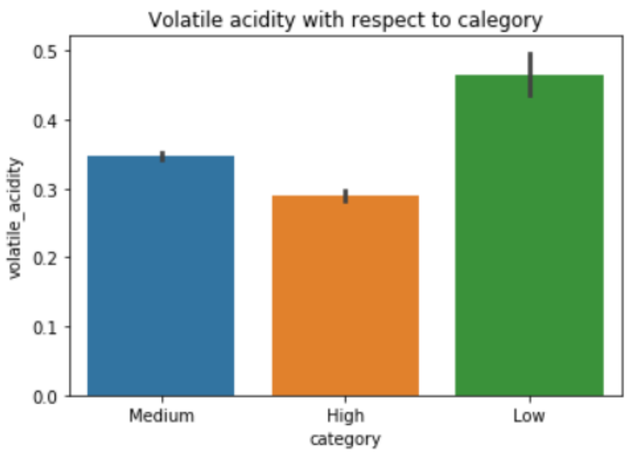


Figure 5.2.2: Variation of Volatile Acidity with respect to Category

Volatile acidity of “low” is higher than the rest , in case of “medium” is moderate and in case of “high” its low .

```
sns.barplot(x='category',y='citric_acid',data=df).
set_title("Citric acid with respect to calegory")
```

```
Text(0.5, 1.0, 'Citric acid with respect to calegory')
```

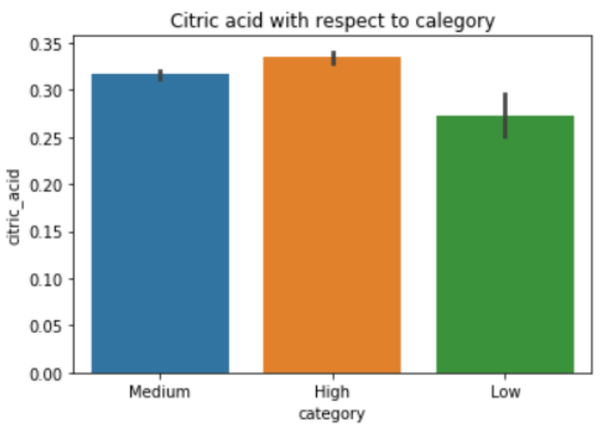


Figure 5.2.3: Variation of Citric Acid with respect to Category

Citric acid of “High” is higher than the rest , in case of “Medium” is moderate and in case of “low” its low .

```
sns.barplot(x='category',y='residual_sugar',data=df).
set_title("Residual sugar with respect to calegory")
```

```
Text(0.5, 1.0, 'Residual sugar with respect to calegory')
```

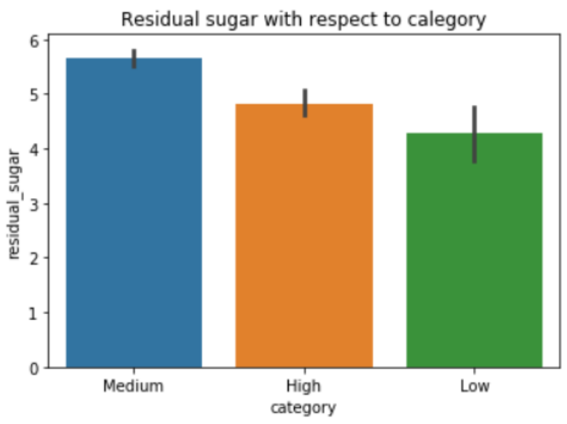


Figure 5.2.4: Variation of Residual Sugars with respect to Category

Residual Sugars of “Medium” is higher than the rest, in case of “High” is moderate and in case of “Low” its low .

```python
sns.barplot(x='category',y='chlorides',data=df).
set_title("Chlorides with respect to calegory")
```

```
Text(0.5, 1.0, 'Chlorides with respect to calegory')
```

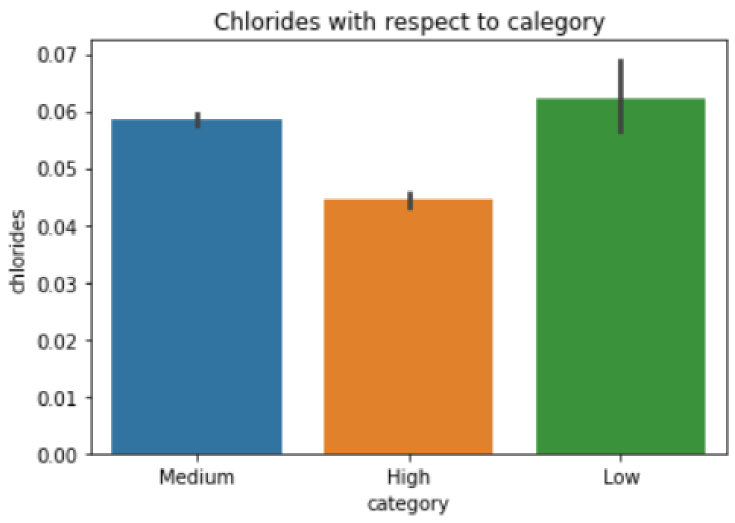


Figure 5.2.5: Variation of Chlorides with respect to Category

Chlorides of “Low” is higher than the rest , in case of “Medium” is moderate and in case of “High” its low .

```python
sns.barplot(x='category',y='free_sulfur_dioxide',data=df).
set_title("Free Sulful Dioxide with respect to calegory")
```

```
Text(0.5, 1.0, 'Free Sulful Dioxide with respect to calegory')
```

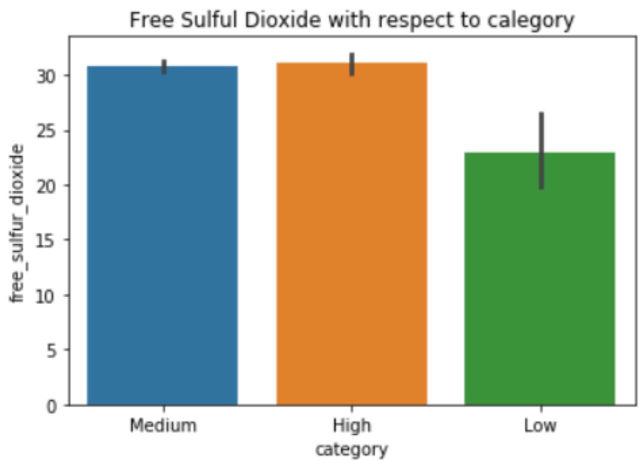


Figure 5.2.6: Variation of Free Sulfur Dioxide with respect to Category

Free Sulfur Dioxideof “High” is higher than the rest , in case of “Medium” is moderate and in case of “Low” its low .

```
sns.barplot(x='category',y='total_sulfur_dioxide',data=df).
set_title("Total Sulfur Dioxide with respect to calegory")
```

```
Text(0.5, 1.0, 'Total Sulfur Dioxide with respect to calegory')
```

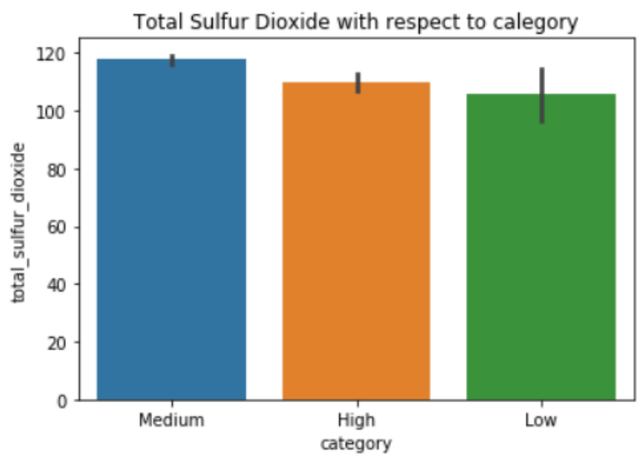


Figure 5.2.7: Variation of Total Sulfur Dioxide with respect to Category

Total Sulfur Dioxide of “Medium” is higher than the rest , in case of “High” is moderate and in case of “Low” its low .

```
sns.barplot(x='category',y='density',data=df).
set_title("Density with respect to calegory")
```

```
Text(0.5, 1.0, 'Density with respect to calegory')
```

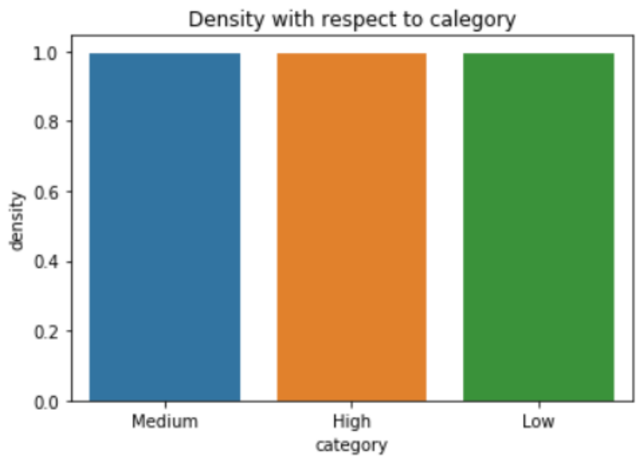


Figure 5.2.8: Variation of Density with respect to Category

The Density of different quality’s of wine are same.

```
sns.barplot(x='category',y='pH',data=df).
set_title("Ph with respect to calegory")
```

```
Text(0.5, 1.0, 'Ph with respect to calegory')
```

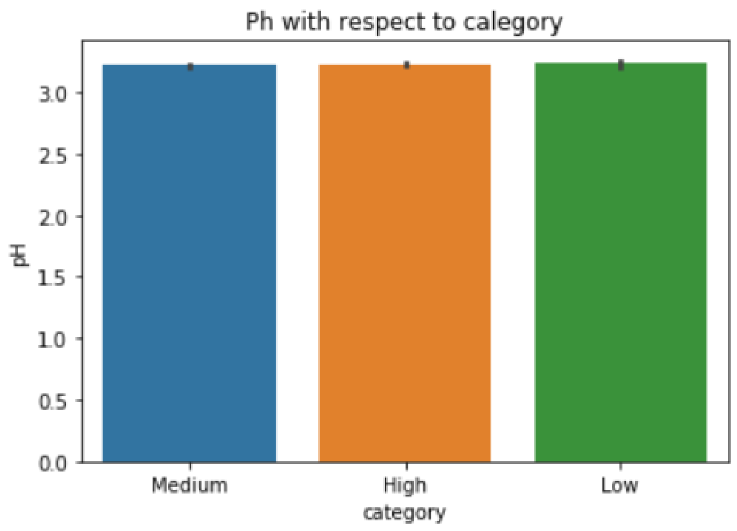


Figure 5.2.9: Variation of PH with respect to Category

The PH of different qualities of wine are same.

```
sns.barplot(x='category',y='sulphates',data=df).
set_title("Sulpahtes with respect to calegory")
```

```
Text(0.5, 1.0, 'Sulpahtes with respect to calegory')
```

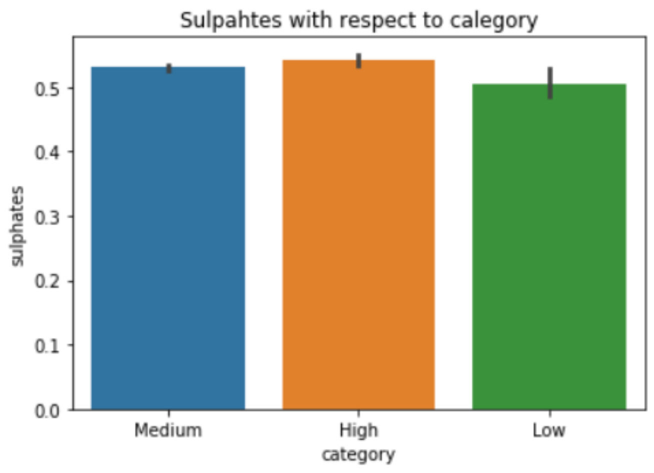


Figure 5.2.10: Variation of Sulphates with respect to Category

The amount of Sulphates is higher in case of "High", moderate in case of "Medium" and low in case of "Low".

```
sns.barplot(x='category',y='alcohol',data=df).
set_title("Alcohol with respect to calegory")
```

```
Text(0.5, 1.0, 'Alcohol with respect to calegory')
```

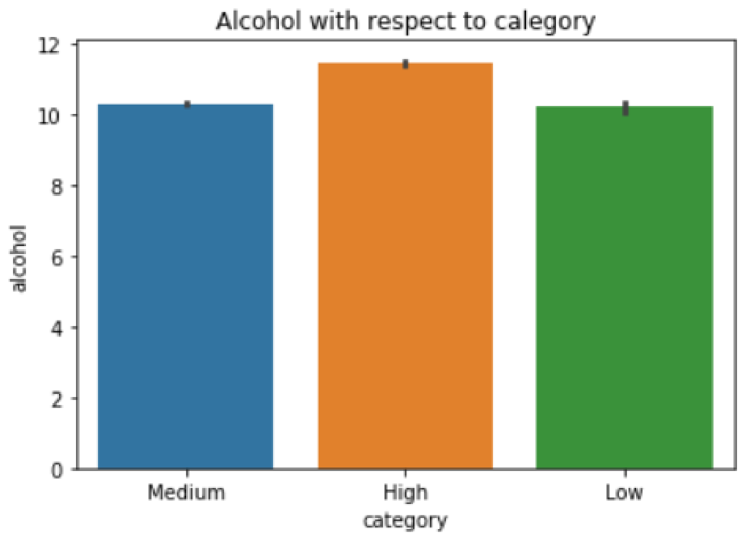


Figure 5.2.11: Variation of Alcohol with respect to Category

The amount of Alcohol is more in case of “High” quality wine and is relatively same in cases of “Low”& “Medium” quality wines.

```python
## visualizing the correlation
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(),annot=True,cmap="YlGnBu").set_title("Correlation heat map of all the features")
```

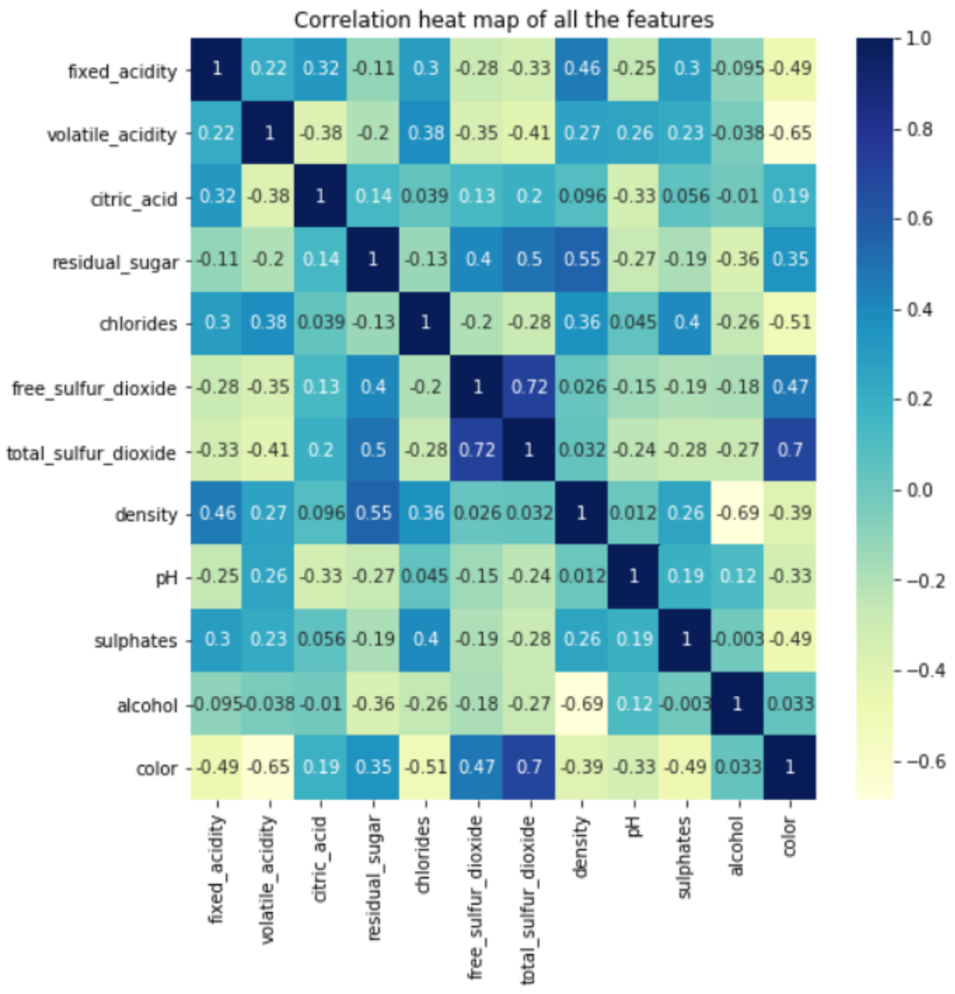


Figure 5.2.12: Correlation Heat map of all the features

```
plt.figure(figsize=(8,8))
sns.heatmap(df.corr()[df.corr()>0.7],annot=True,cmap="YlGnBu").set_title("High Correlation heat map of all the features")
```

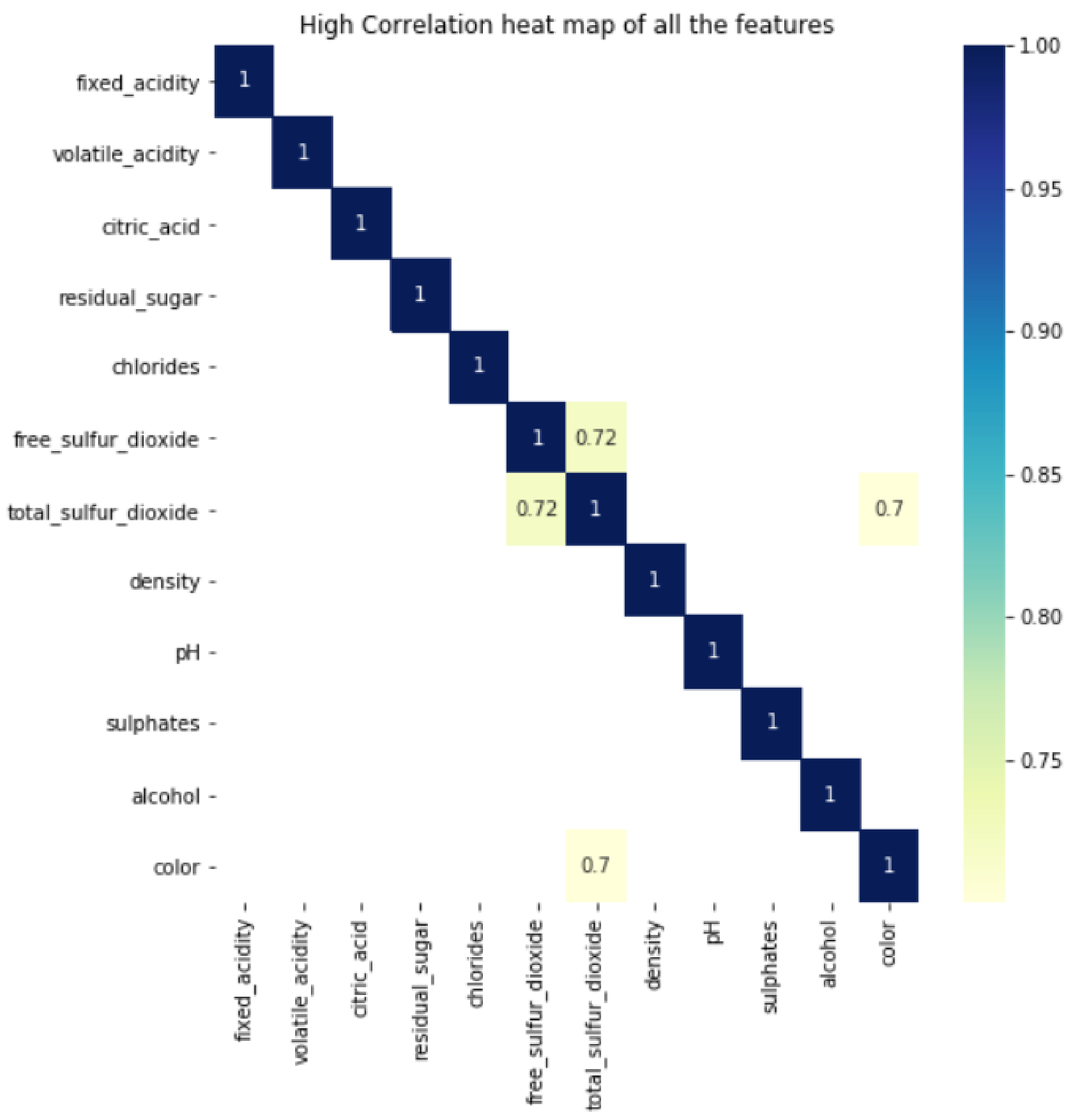
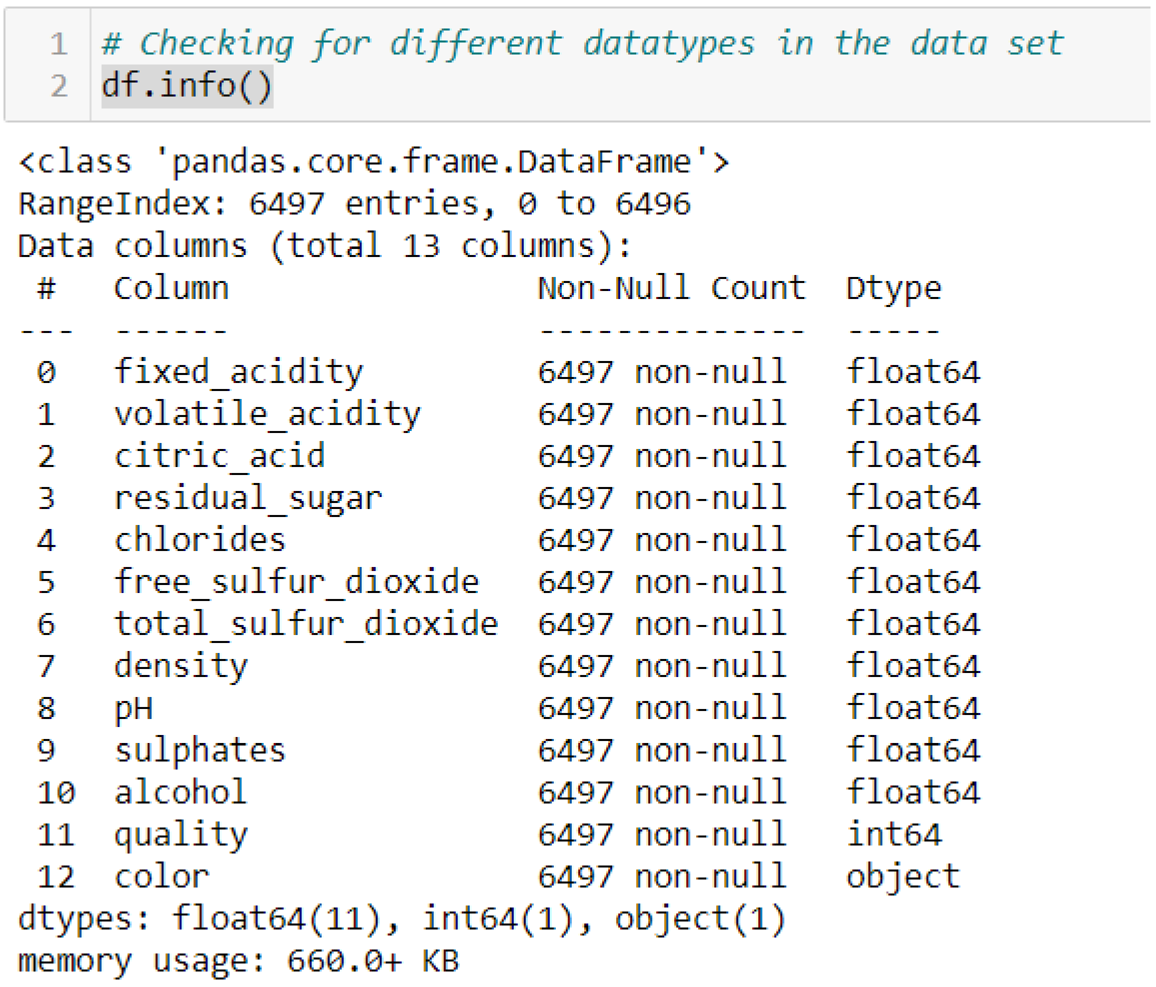


Figure 5.2.13: High Correlation Heat map of all the features

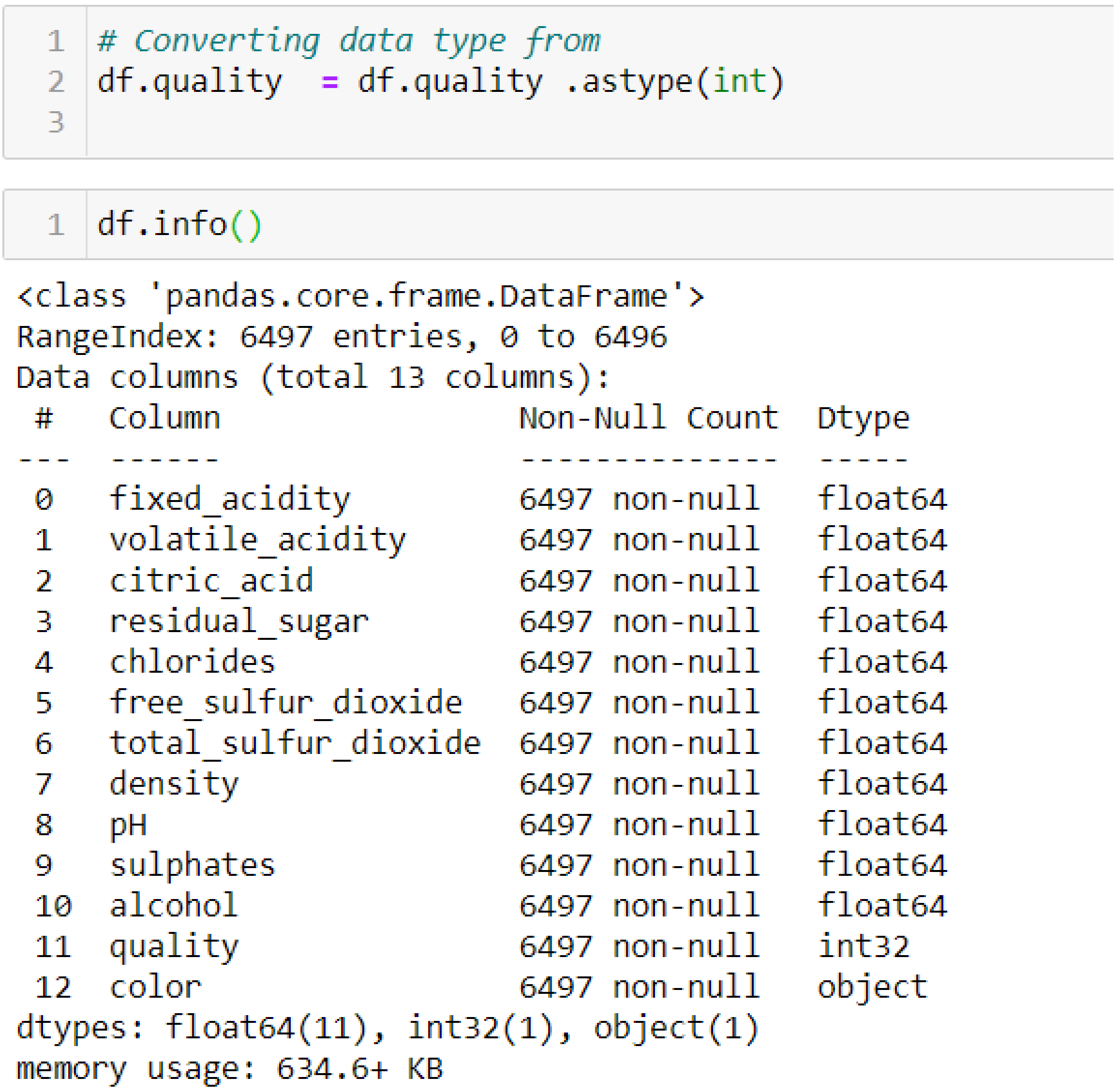
5.3 DATA TYPE CONVERSION:



```
# Checking for different datatypes in the data set
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  color                 6497 non-null   object 
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

Figure 5.3.1: Data Type before conversion

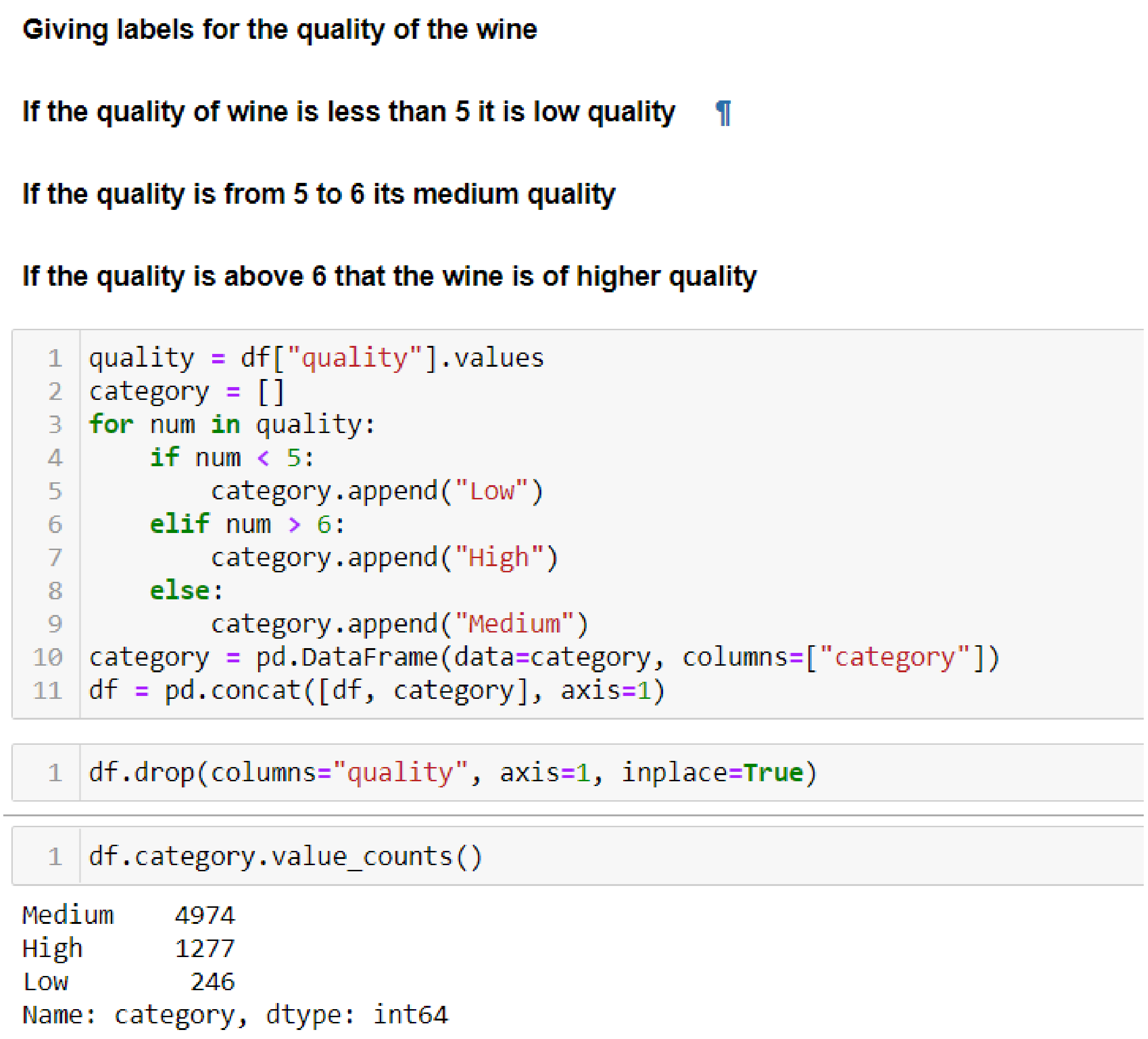


```python
# Converting data type from
df.quality  = df.quality .astype(int)

```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int32  
 12  color                 6497 non-null   object 
dtypes: float64(11), int32(1), object(1)
memory usage: 634.6+ KB
```

Figure 5.3.2: Data Type after conversion



Giving labels for the quality of the wine

If the quality of wine is less than 5 it is low quality ¶

If the quality is from 5 to 6 its medium quality

If the quality is above 6 that the wine is of higher quality

```python
quality = df["quality"].values
category = []
for num in quality:
    if num < 5:
        category.append("Low")
    elif num > 6:
        category.append("High")
    else:
        category.append("Medium")
category = pd.DataFrame(data=category, columns=["category"])
df = pd.concat([df, category], axis=1)
```

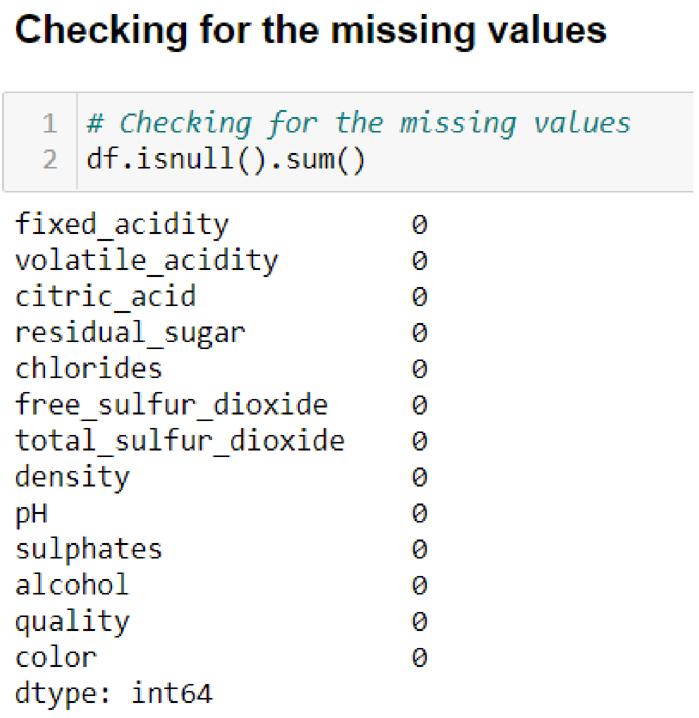
```python
df.drop(columns="quality", axis=1, inplace=True)
```

```python
df.category.value_counts()
```

```
Medium    4974
High      1277
Low        246
Name: category, dtype: int64
```

Figure 5.3.3: Conversion of Numerical Column of quality to Categorical Column

5.4 HANDLING MISSING VALUES:



Checking for the missing values

```python
# Checking for the missing values
df.isnull().sum()
```

```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
color                   0
dtype: int64
```

Figure 5.4.1: Checking for Missing values

```python
sns.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x19fbd383788>
```

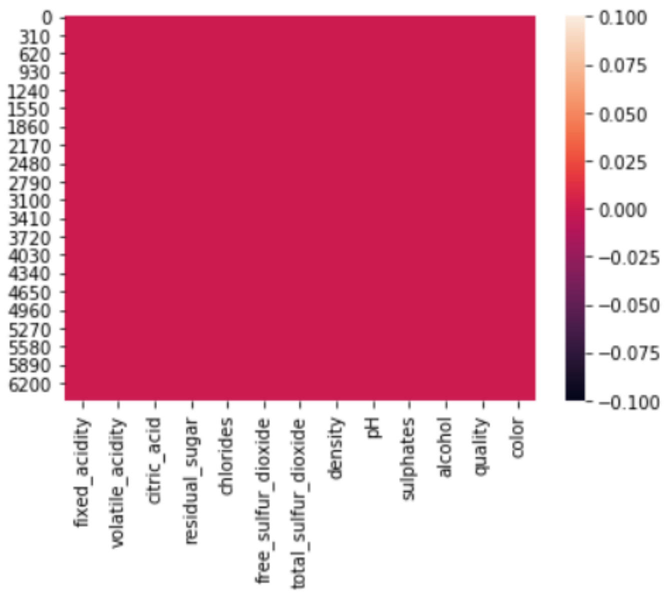


Figure 5.4.2: Checking for Missing values using Heat map

There are no missing values in my Data Set.

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 22

(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 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.

5.5 ENCODING CATEGOTICAL DATA:

● Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.

● Categorical Variables are of two types: Nominal and Ordinal

● Nominal: The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour

● Ordinal: The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

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

● Handling categorical data using dummies:

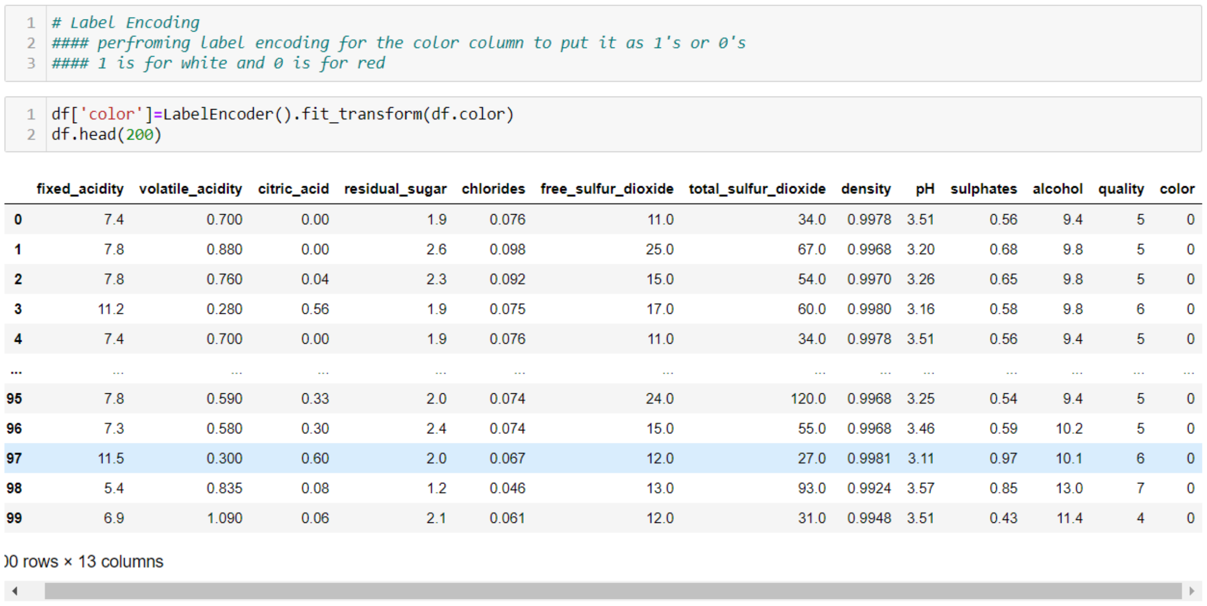
In pandas library we have a method called get_dummies() which creates dummy variables for those categorical data in the form of 0's and 1's.

Once these dummies got created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

- Getting dummies using label encoder from scikit learn package

We have a method called label encoder in scikit learn package .we need to import the label encoder method from scikitlearn package and after that we have to fit and transform the data frame to make the categorical data into dummies.

If we use this method to get dummies then in place of categorical data we get the numerical values (0,1,2….)



```
# Label Encoding
#### perfroming label encoding for the color column to put it as 1's or 0's
#### 1 is for white and 0 is for red
```

```
df['color']=LabelEncoder().fit_transform(df.color)
df.head(200)
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	0
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	0
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	0
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	7.8	0.590	0.33	2.0	0.074	24.0	120.0	0.9968	3.25	0.54	9.4	5	0
96	7.3	0.580	0.30	2.4	0.074	15.0	55.0	0.9968	3.46	0.59	10.2	5	0
97	11.5	0.300	0.60	2.0	0.067	12.0	27.0	0.9981	3.11	0.97	10.1	6	0
98	5.4	0.835	0.08	1.2	0.046	13.0	93.0	0.9924	3.57	0.85	13.0	7	0
99	6.9	1.090	0.06	2.1	0.061	12.0	31.0	0.9948	3.51	0.43	11.4	4	0

)0 rows × 13 columns

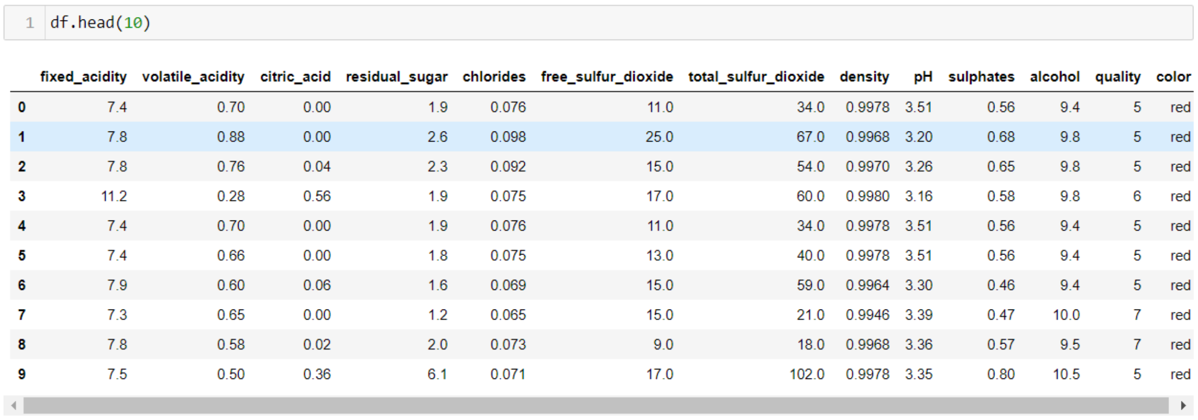
Figure 5.5.1: Label Encoding 'color' column

CHAPTER 6

FEATURE SELECTION

6.1 SELECT RELEVANT FEATURS FOR THE ANALYSIS:

The columns in the dataset has all the features we need to analyze the data.



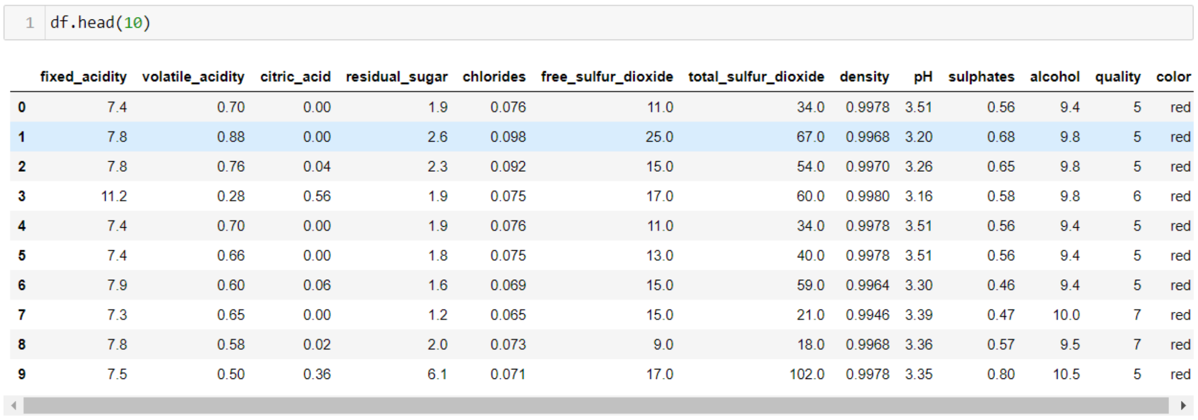
```
df.head(10)
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5	red
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5	red
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7	red
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7	red
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5	red

Figure 6.1.1: Features in the data set

6.2 DROP IRRELEVANT FEATURES:

As all the features are necessary, the dataset doesn't contain any irrelevant features. Therefore, there's no need of dropping any of the features.

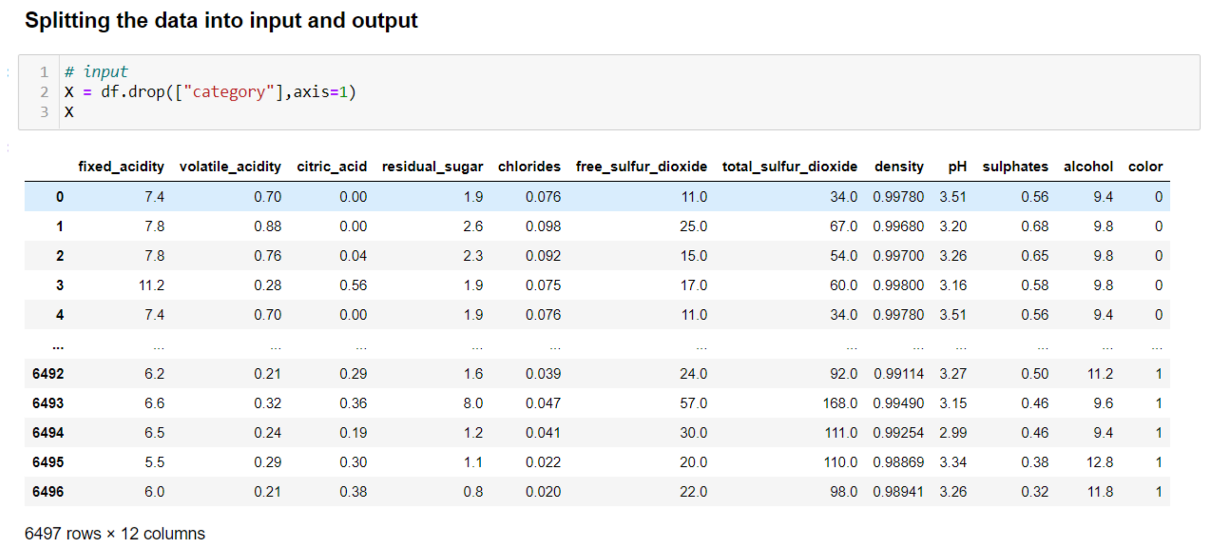


```
df.head(10)
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5	red
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5	red
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7	red
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7	red
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5	red

Figure 6.2.1: All relevant Features in the data set

6.3 TRAIN-TEST-SPLIT:



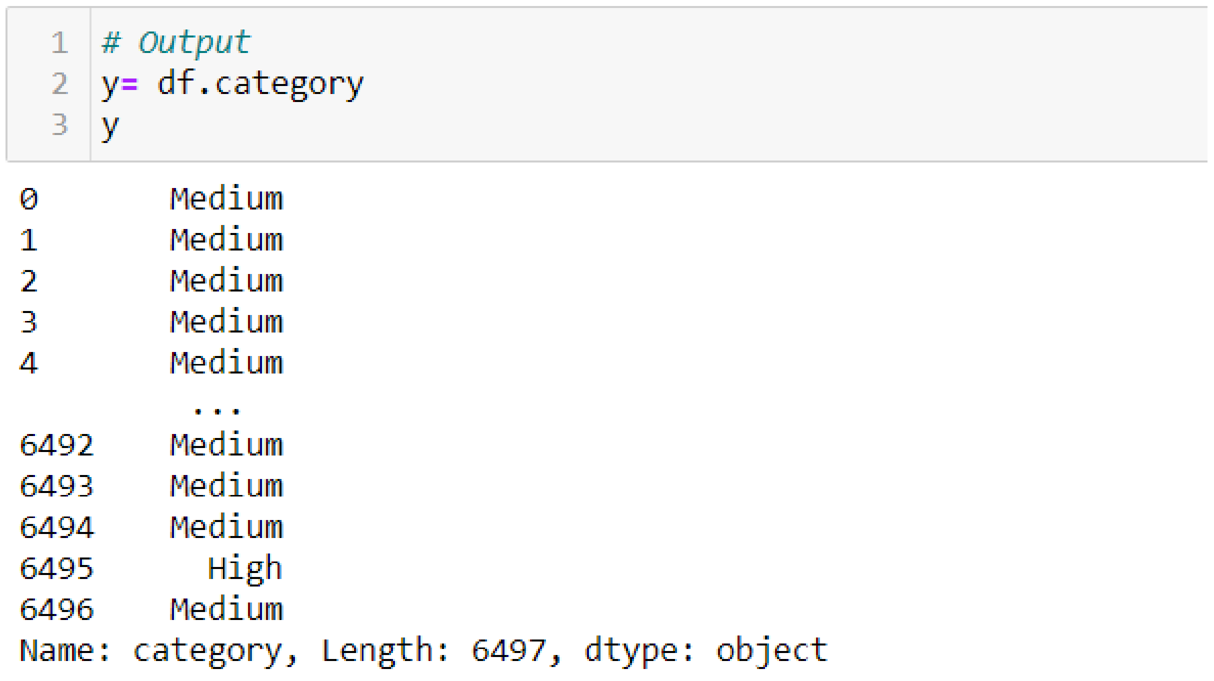
Splitting the data into input and output

```python
# input
X = df.drop(["category"],axis=1)
X
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	color
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	0
...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	1
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	1
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	1
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	1
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	1

6497 rows × 12 columns

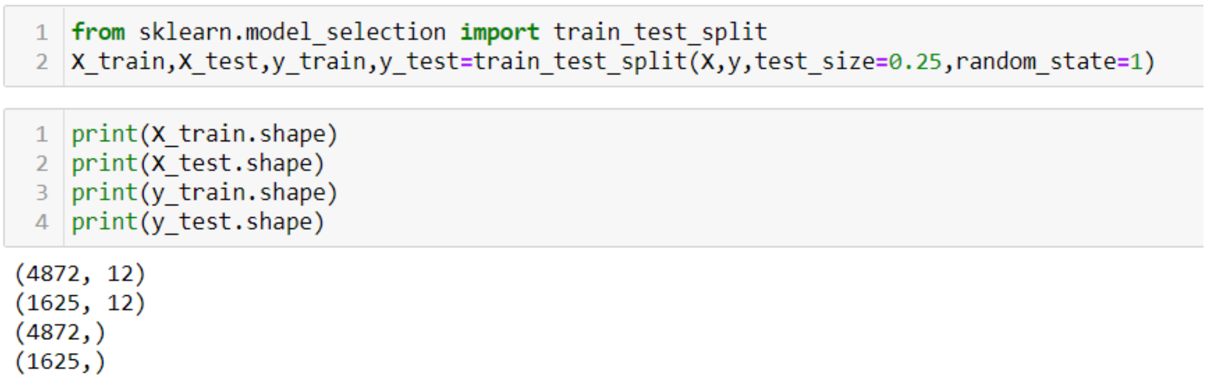
Figure 6.3.1: Splitting the Data set to X as input



```python
# Output
y= df.category
y
```

```
0        Medium
1        Medium
2        Medium
3        Medium
4        Medium
          ...
6492     Medium
6493     Medium
6494     Medium
6495       High
6496     Medium
Name: category, Length: 6497, dtype: object
```

Figure 6.3.2: Splitting the Data set to y as output



```python
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=1)
```

```python
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(4872, 12)
(1625, 12)
(4872,)
(1625,)
```

Figure 6.3.3: Splitting the input and output data

Here by using train_test_split we are splitting the input and output data to input train, input test, output train, output test. We are using test_size=0.25 it splits 75% of data as train data And the rest 25% as test data to evaluate the model performance. Here random_state is used , if random_size is not used ,then if kernel is restarted it will generate the train and test data randomly so in order to have the same train and test data we are using random_size=1.

CHAPTER 7

MODEL BUILDING AND EVALUATION

7.1 BRIEF ABOUT THE MODEL USED:

7.1.1 Decision Tree Classification:

Decision tree learning is one of the predictive modeling approaches used in statics, data mining and machine learning. It uses a decision tree from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making.

7.1.2 Random Forest Classification:

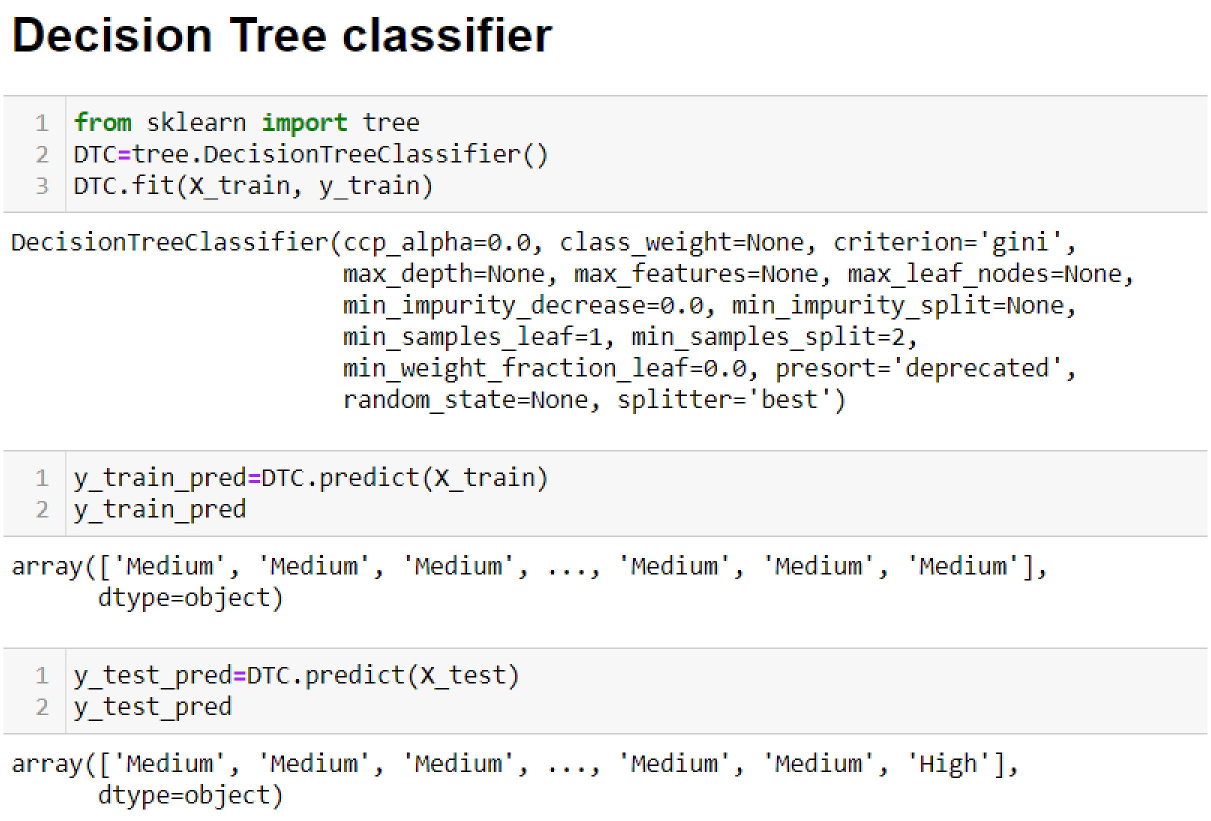
A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging.

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

7.2 TRAIN THE MODEL:

7.2.1 Training the model with Decision Tree Classifier:



Decision Tree classifier

```python
from sklearn import tree
DTC=tree.DecisionTreeClassifier()
DTC.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       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')
```

```python
y_train_pred=DTC.predict(X_train)
y_train_pred
```

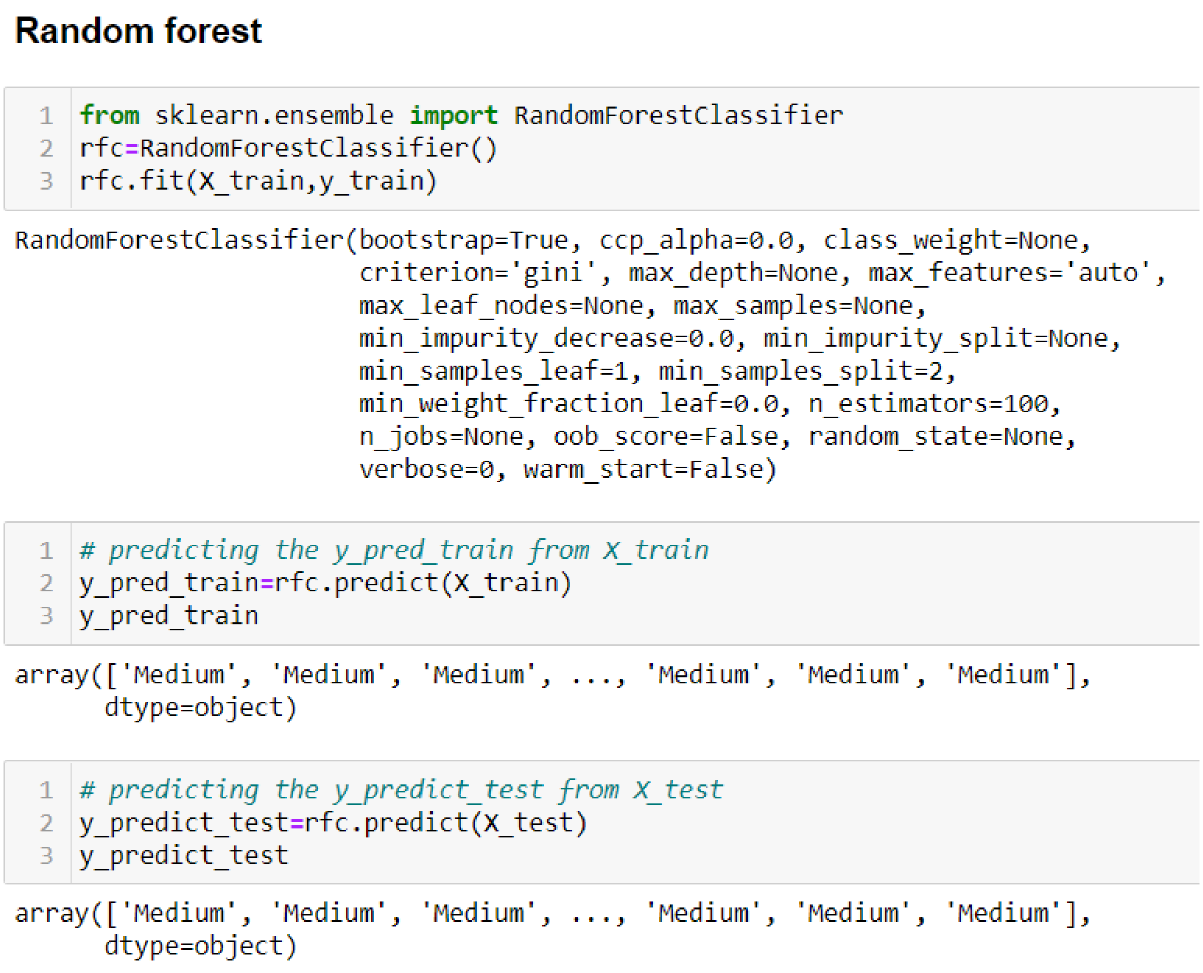
```
array(['Medium', 'Medium', 'Medium', ..., 'Medium', 'Medium', 'Medium'],
      dtype=object)
```

```python
y_test_pred=DTC.predict(X_test)
y_test_pred
```

```
array(['Medium', 'Medium', 'Medium', ..., 'Medium', 'Medium', 'High'],
      dtype=object)
```

Figure 7.2.1.1: Training the model using Decision Tree Classifier Algorithm

7.2.2 Training the model with Random Forest Classifier:



Random forest

```python
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
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)
```

```python
# predicting the y_pred_train from X_train
y_pred_train=rfc.predict(X_train)
y_pred_train
```

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

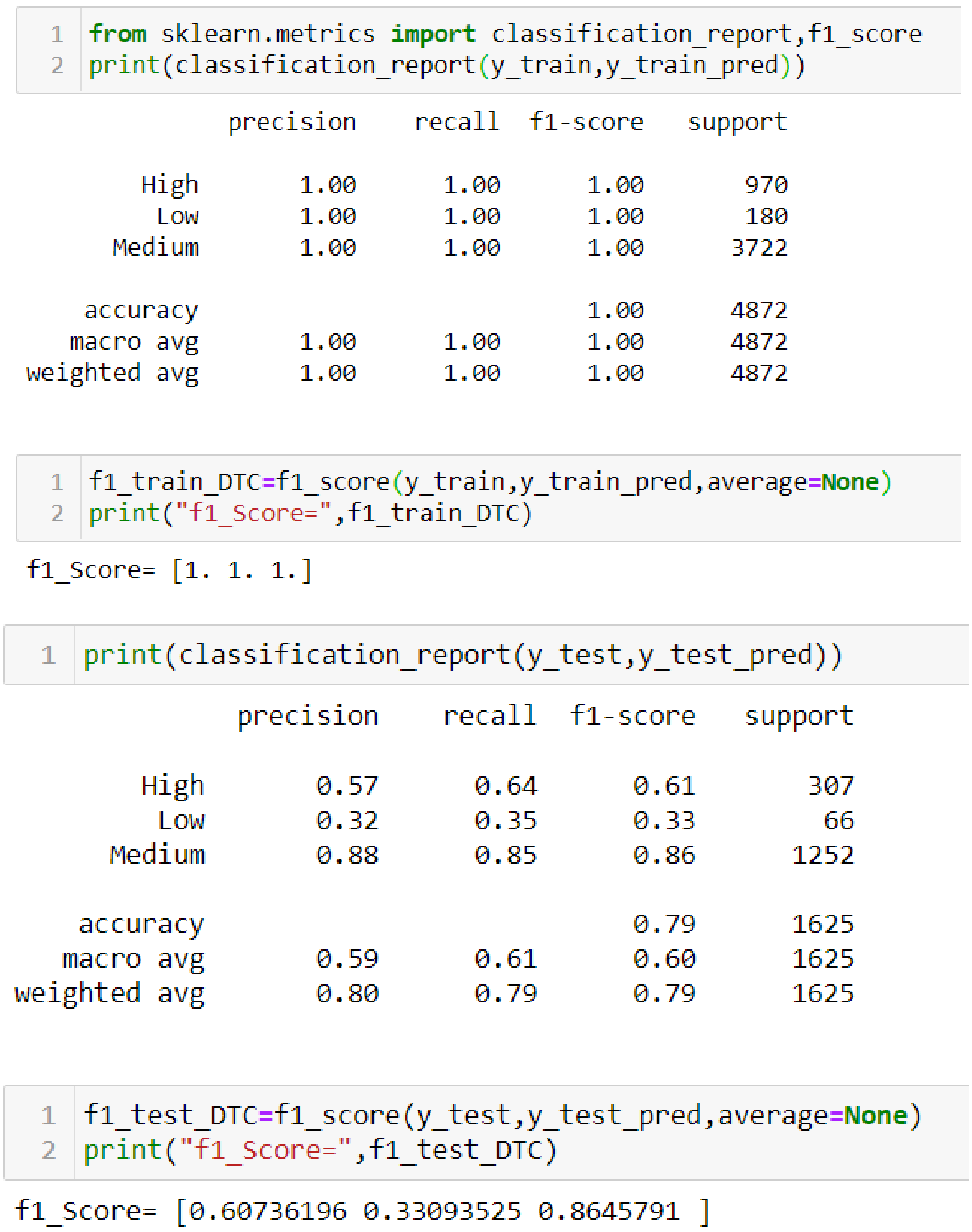
```python
# predicting the y_predict_test from X_test
y_predict_test=rfc.predict(X_test)
y_predict_test
```

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

Figure 7.2.2.1: Training the model using Random Forest Classifier Algorithm

7.3 VALIDATE THE MODELS:

7.3.1 Validating the Decision Tree classifier Model:



```
from sklearn.metrics import classification_report,f1_score
print(classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

        High       1.00      1.00      1.00       970
         Low       1.00      1.00      1.00       180
      Medium       1.00      1.00      1.00      3722

    accuracy                           1.00      4872
   macro avg       1.00      1.00      1.00      4872
weighted avg       1.00      1.00      1.00      4872
```

```
f1_train_DTC=f1_score(y_train,y_train_pred,average=None)
print("f1_Score=",f1_train_DTC)
```

```
f1_Score= [1. 1. 1.]
```

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

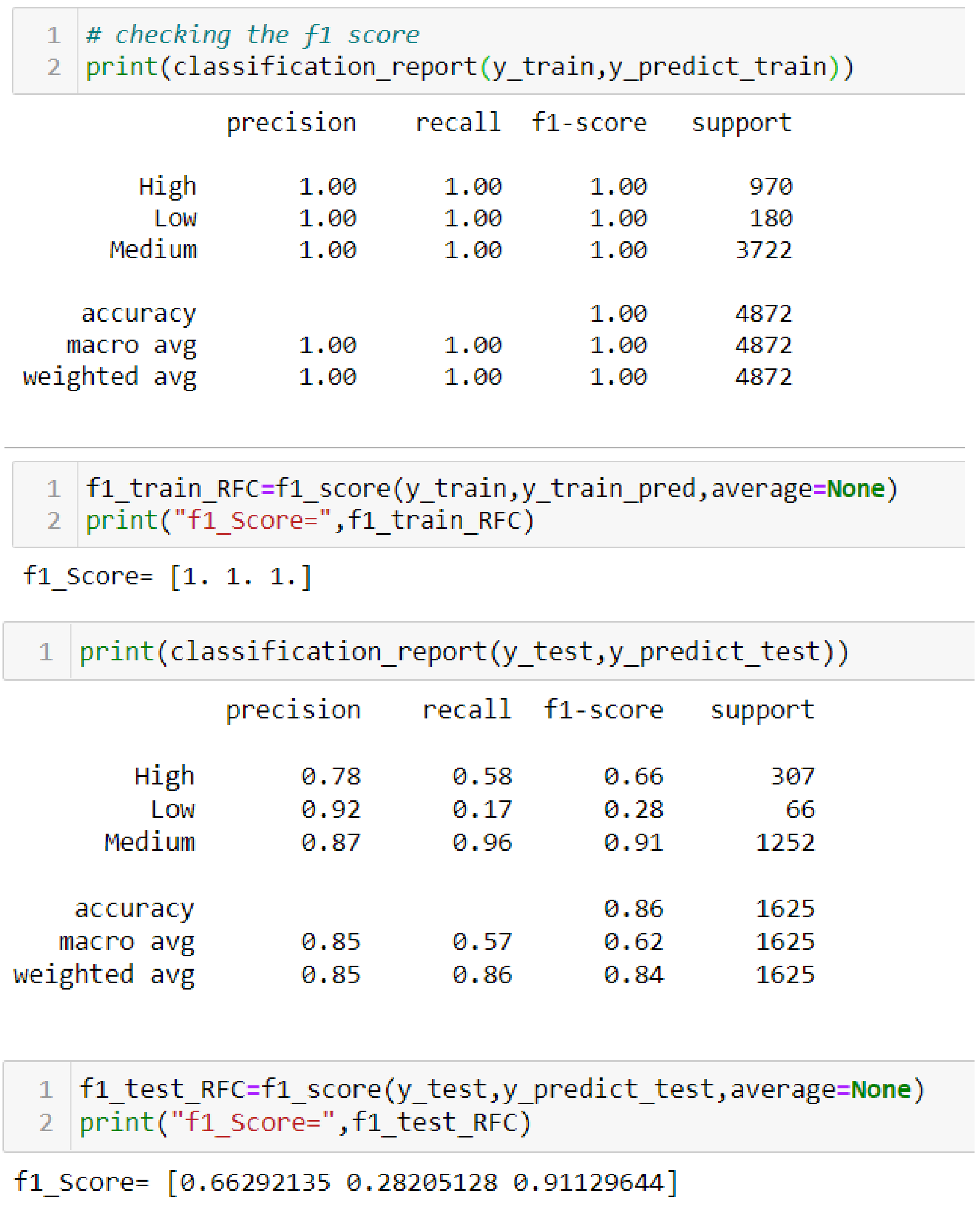
```
              precision    recall  f1-score   support

        High       0.57      0.64      0.61       307
         Low       0.32      0.35      0.33        66
      Medium       0.88      0.85      0.86      1252

    accuracy                           0.79      1625
   macro avg       0.59      0.61      0.60      1625
weighted avg       0.80      0.79      0.79      1625
```

```
f1_test_DTC=f1_score(y_test,y_test_pred,average=None)
print("f1_Score=",f1_test_DTC)
```

```
f1_Score= [0.60736196 0.33093525 0.8645791 ]
```

Figure 7.3.1.1: Validating the Decision Tree Classifier model based on Classification report

7.3.2 Validating the Random Forest Classifier Model:



```
# checking the f1 score
print(classification_report(y_train,y_predict_train))
```

```
              precision    recall  f1-score   support

        High       1.00      1.00      1.00       970
         Low       1.00      1.00      1.00       180
      Medium       1.00      1.00      1.00      3722

    accuracy                           1.00      4872
   macro avg       1.00      1.00      1.00      4872
weighted avg       1.00      1.00      1.00      4872
```

```
f1_train_RFC=f1_score(y_train,y_train_pred,average=None)
print("f1_Score=",f1_train_RFC)
```

```
f1_Score= [1. 1. 1.]
```

```
print(classification_report(y_test,y_predict_test))
```

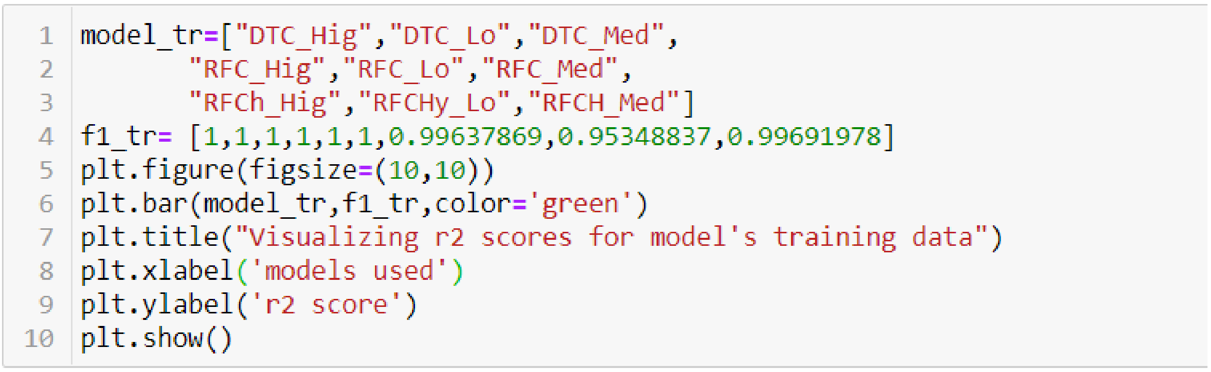
```
              precision    recall  f1-score   support

        High       0.78      0.58      0.66       307
         Low       0.92      0.17      0.28        66
      Medium       0.87      0.96      0.91      1252

    accuracy                           0.86      1625
   macro avg       0.85      0.57      0.62      1625
weighted avg       0.85      0.86      0.84      1625
```

```
f1_test_RFC=f1_score(y_test,y_predict_test,average=None)
print("f1_Score=",f1_test_RFC)
```

```
f1_Score= [0.66292135 0.28205128 0.91129644]
```

Figure 7.3.2.1: Validating the Random Forest Classifier model based on Classification report.

Here we are using F1 score to validate between the 2 models. As the classes that I have created in the data set are unbalanced, so we are preferring F1 score over accuracy.F1 score is a combination of recall and precision.



```
model_tr=["DTC_Hig","DTC_Lo","DTC_Med",
       "RFC_Hig","RFC_Lo","RFC_Med",
       "RFCh_Hig","RFCHy_Lo","RFCH_Med"]
f1_tr= [1,1,1,1,1,1,0.99637869,0.95348837,0.99691978]
plt.figure(figsize=(10,10))
plt.bar(model_tr,f1_tr,color='green')
plt.title("Visualizing r2 scores for model's training data")
plt.xlabel('models used')
plt.ylabel('r2 score')
plt.show()
```

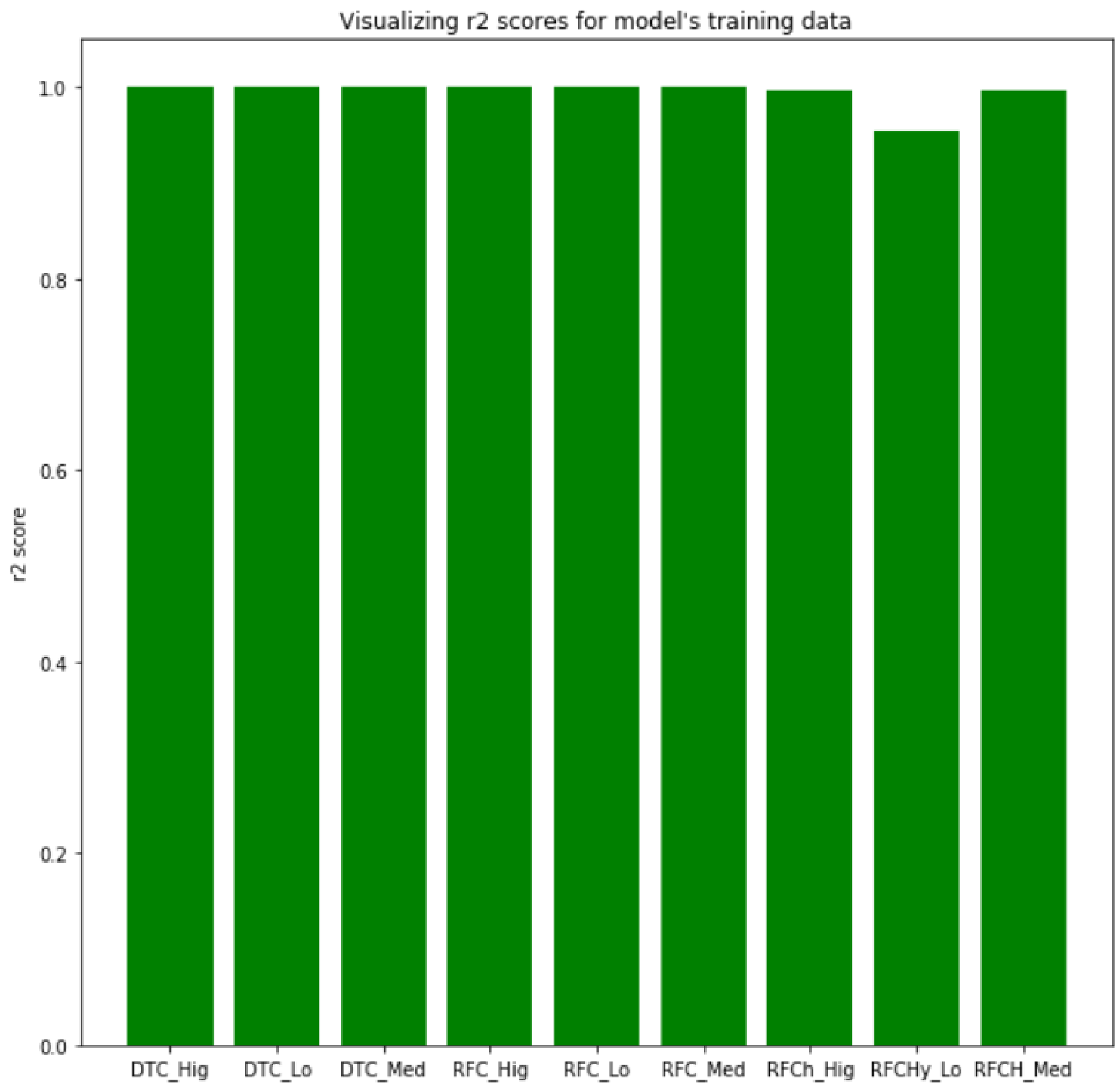
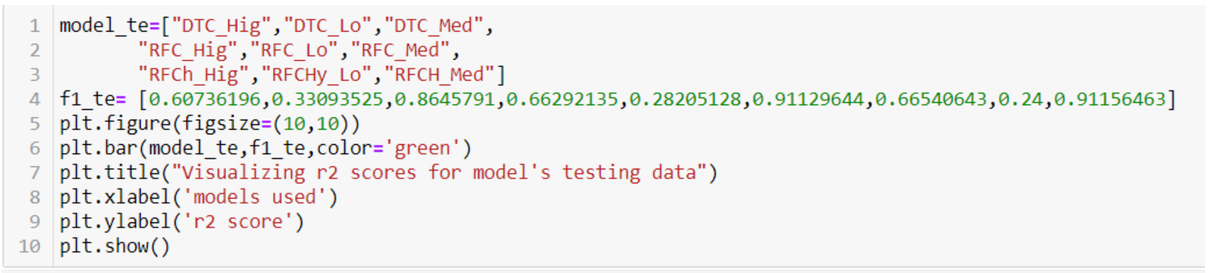


Figure 7.3.3.1: Validating the F1 Score of the Models for train data



```python
model_te=["DTC_Hig","DTC_Lo","DTC_Med",
       "RFC_Hig","RFC_Lo","RFC_Med",
       "RFCh_Hig","RFCHy_Lo","RFCH_Med"]
f1_te= [0.60736196,0.33093525,0.8645791,0.66292135,0.28205128,0.91129644,0.66540643,0.24,0.91156463]
plt.figure(figsize=(10,10))
plt.bar(model_te,f1_te,color='green')
plt.title("Visualizing r2 scores for model's testing data")
plt.xlabel('models used')
plt.ylabel('r2 score')
plt.show()
```

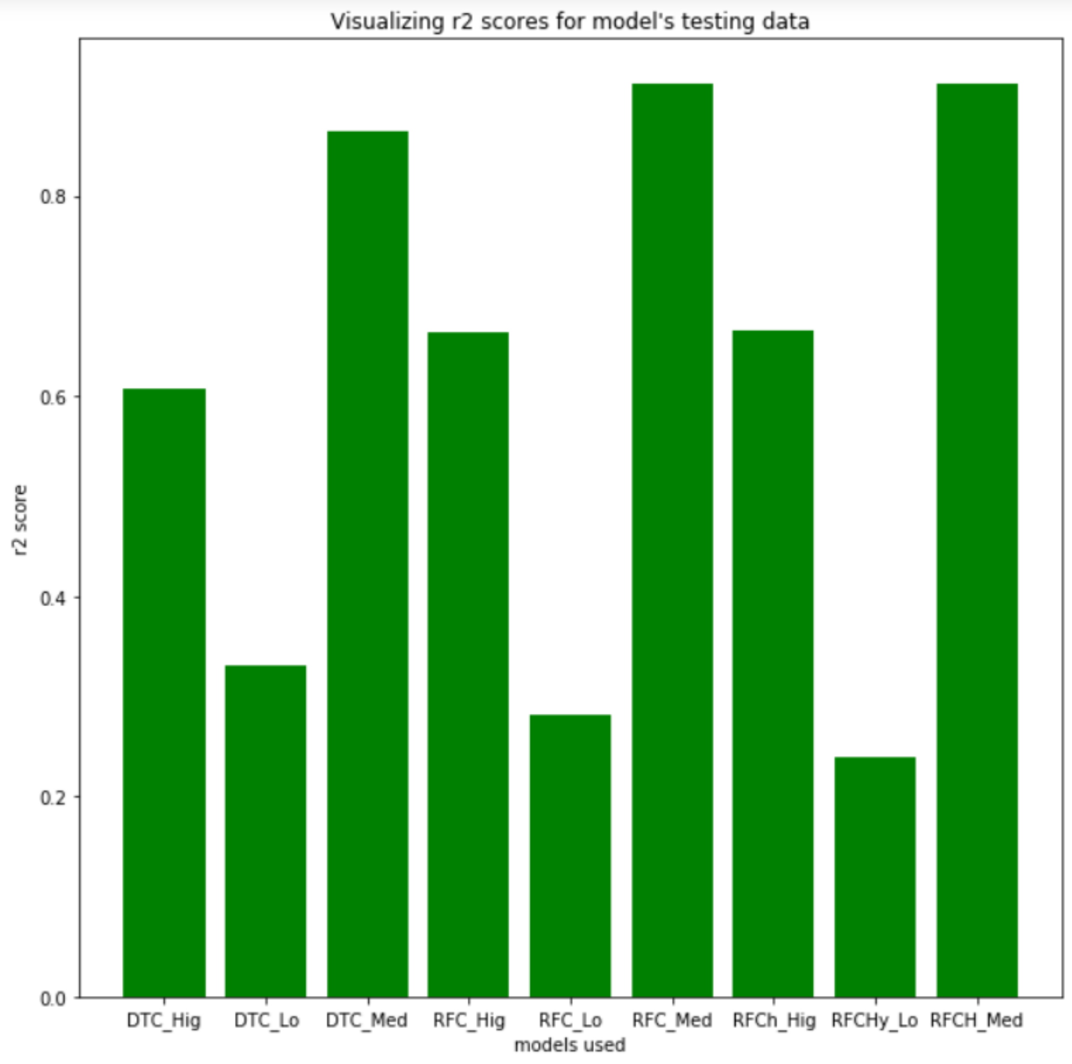
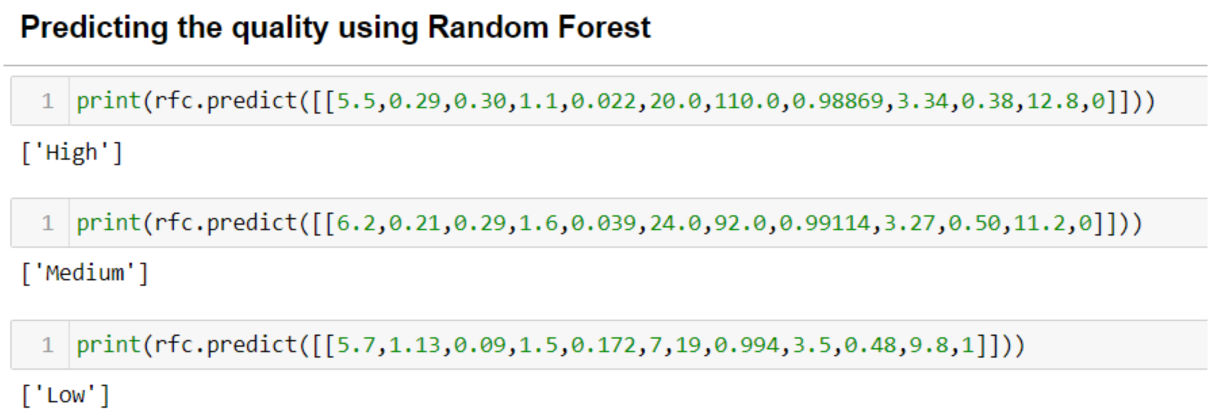


Figure 7.3.3.2: Validating the F1 Score of the Models for test data

7.4 MAKE PREDICTIONS:

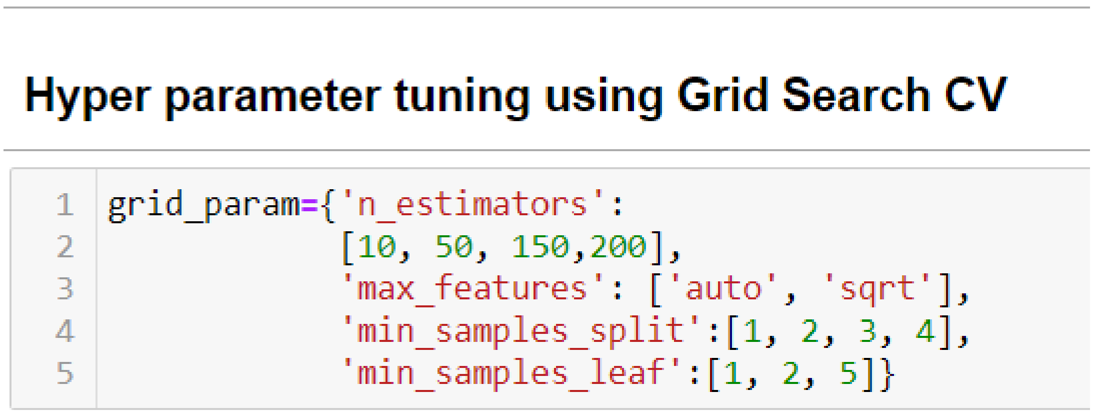

Predicting the quality using Random Forest

```
print(rfc.predict([[5.5,0.29,0.30,1.1,0.022,20.0,110.0,0.98869,3.34,0.38,12.8,0]]))
```
```
['High']
```
```
print(rfc.predict([[6.2,0.21,0.29,1.6,0.039,24.0,92.0,0.99114,3.27,0.50,11.2,0]]))
```
```
['Medium']
```
```
print(rfc.predict([[5.7,1.13,0.09,1.5,0.172,7,19,0.994,3.5,0.48,9.8,1]]))
```
```
['Low']
```

Figure 7.4.1: Predicting the wine quality based on Random Forest Classification

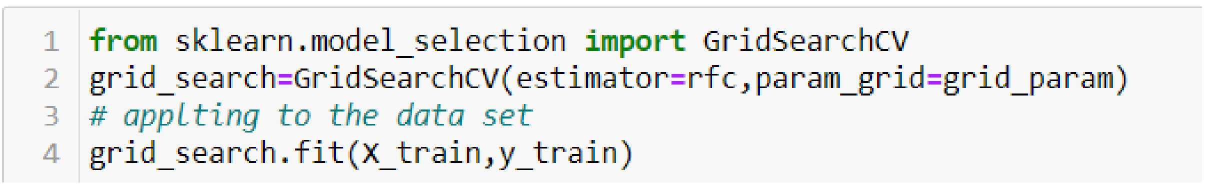
Random Forest Classification is used to make predictions on the quality of wine which it did without miss classifying.

7.5 PARAMETER TUNING:


Hyper parameter tuning using Grid Search CV

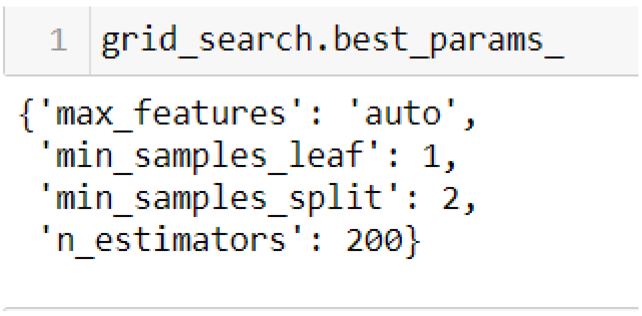
```
grid_param={'n_estimators':
            [10, 50, 150,200],
            'max_features': ['auto', 'sqrt'],
            'min_samples_split':[1, 2, 3, 4],
            'min_samples_leaf':[1, 2, 5]}
```

Figure 7.5.1: Assigning the parameters for Parameter Tuning



```python
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rfc,param_grid=grid_param)
# applting to the data set
grid_search.fit(X_train,y_train)
```

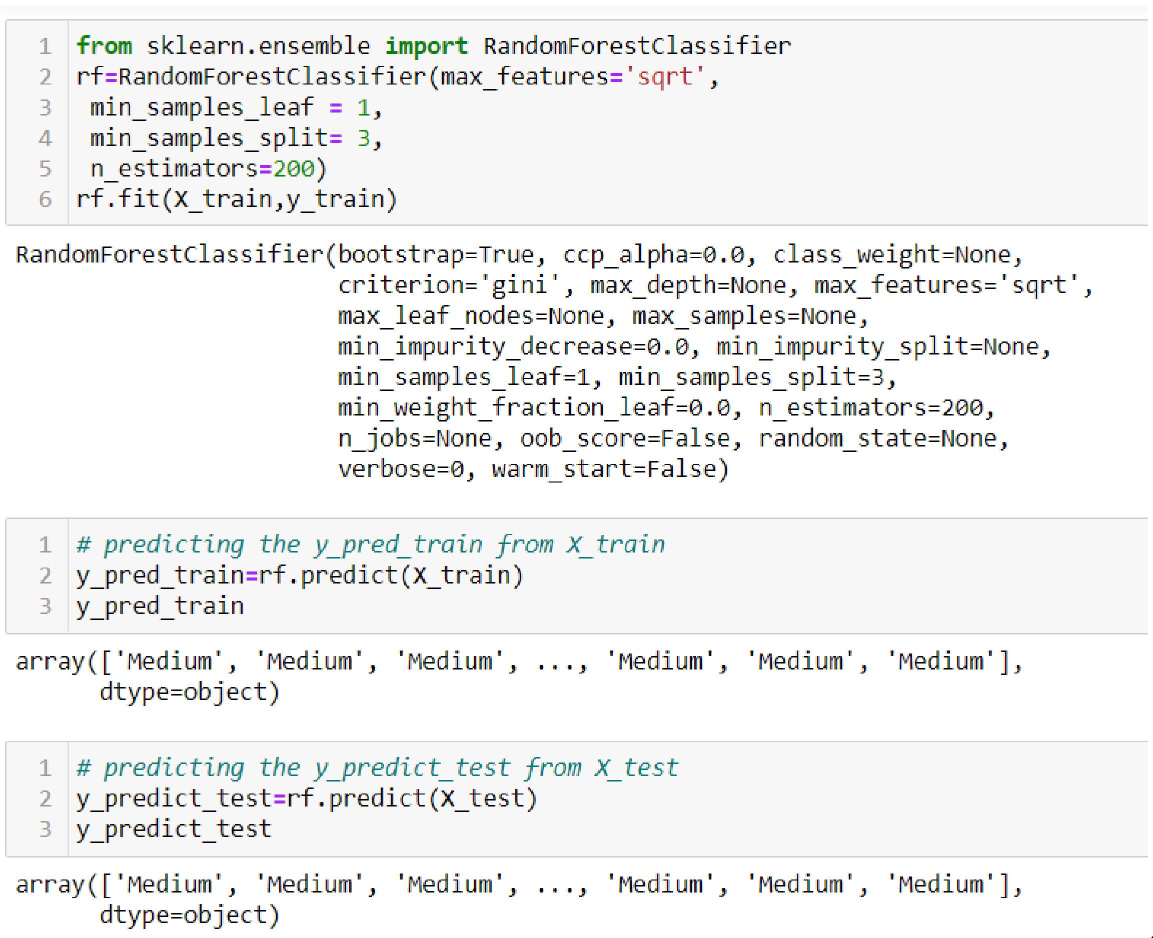
Figure 7.5.2: Fitting each and every parameter in the model



```python
grid_search.best_params_
```

```
{'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

Figure 7.5.3: Getting the best of the above parameters



```python
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(max_features='sqrt',
 min_samples_leaf = 1,
 min_samples_split= 3,
 n_estimators=200)
rf.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=3,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

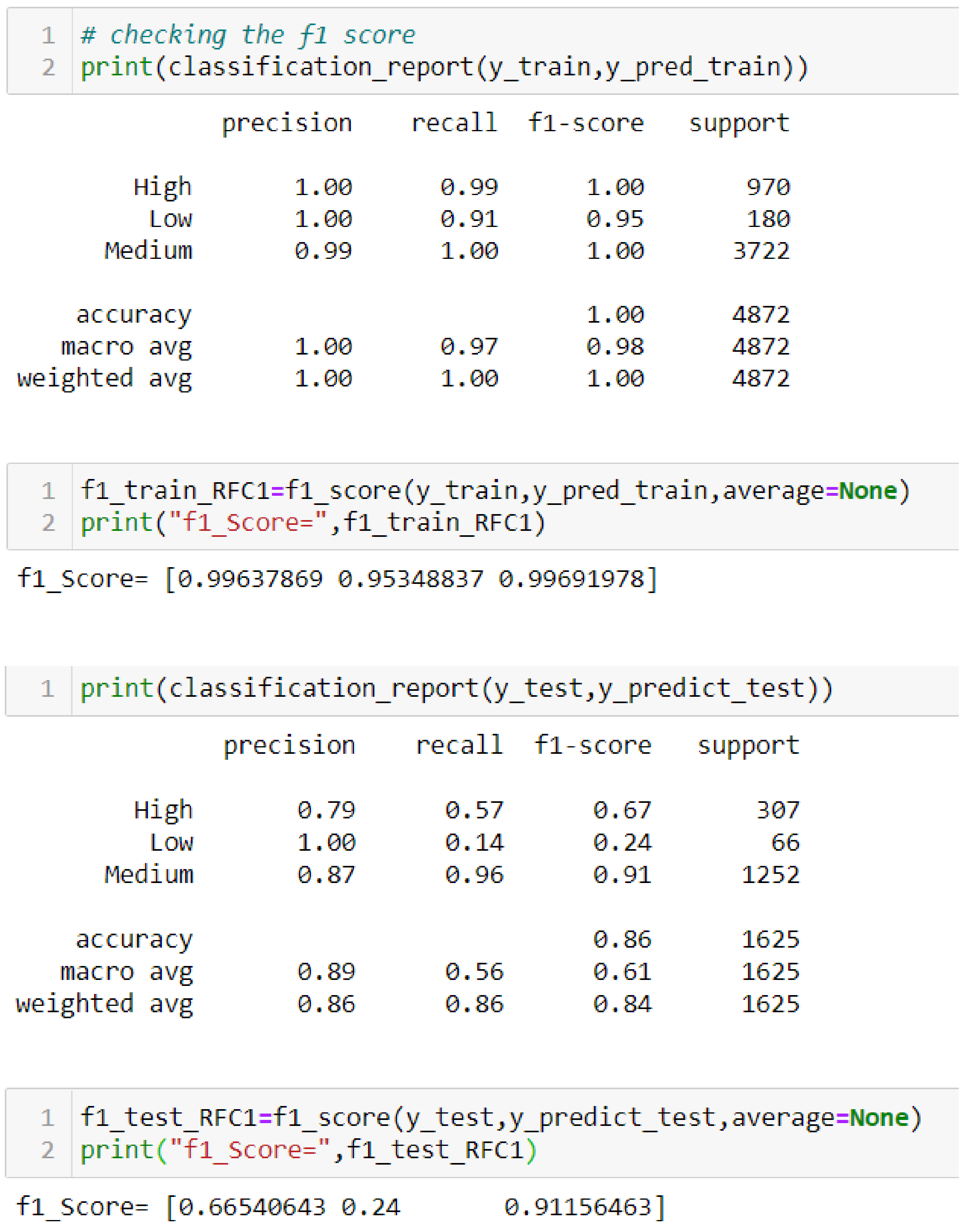
```python
# predicting the y_pred_train from X_train
y_pred_train=rf.predict(X_train)
y_pred_train
```

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

```python
# predicting the y_predict_test from X_test
y_predict_test=rf.predict(X_test)
y_predict_test
```

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

Figure 7.5.4: Fitting the best parameters into the train and test data



```
# checking the f1 score
print(classification_report(y_train,y_pred_train))
```

```
              precision    recall  f1-score   support

        High       1.00      0.99      1.00       970
         Low       1.00      0.91      0.95       180
      Medium       0.99      1.00      1.00      3722

    accuracy                           1.00      4872
   macro avg       1.00      0.97      0.98      4872
weighted avg       1.00      1.00      1.00      4872
```

```
f1_train_RFC1=f1_score(y_train,y_pred_train,average=None)
print("f1_Score=",f1_train_RFC1)
```

```
f1_Score= [0.99637869 0.95348837 0.99691978]
```

```
print(classification_report(y_test,y_predict_test))
```

```
              precision    recall  f1-score   support

        High       0.79      0.57      0.67       307
         Low       1.00      0.14      0.24        66
      Medium       0.87      0.96      0.91      1252

    accuracy                           0.86      1625
   macro avg       0.89      0.56      0.61      1625
weighted avg       0.86      0.86      0.84      1625
```

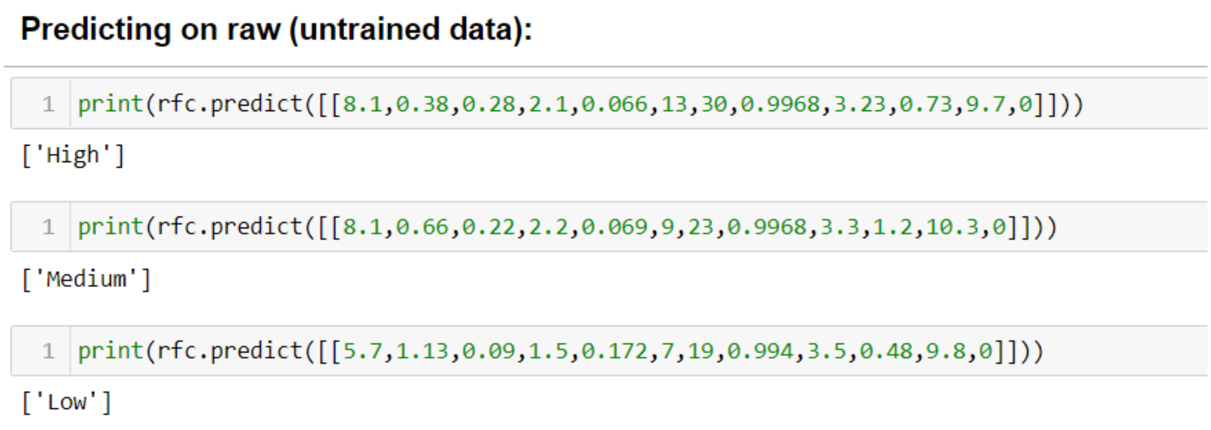
```
f1_test_RFC1=f1_score(y_test,y_predict_test,average=None)
print("f1_Score=",f1_test_RFC1)
```

```
f1_Score= [0.66540643 0.24       0.91156463]
```

Figure 7.5.5: Validating the score after fitting with the parameter tuning

As the score after parameter tuning is almost similar to the default parameters, I am going with the predicting the model with default parameters.

7.6 PREDICTIONS ON RAW DATA:


Predicting on raw (untrained data):

```
print(rfc.predict([[8.1,0.38,0.28,2.1,0.066,13,30,0.9968,3.23,0.73,9.7,0]]))
```

```
['High']
```

```
print(rfc.predict([[8.1,0.66,0.22,2.2,0.069,9,23,0.9968,3.3,1.2,10.3,0]]))
```

```
['Medium']
```

```
print(rfc.predict([[5.7,1.13,0.09,1.5,0.172,7,19,0.994,3.5,0.48,9.8,0]]))
```

```
['Low']
```

Figure 7.6.1: Predicting the wine quality on RAW(untrained data)

Random Forest Classifier was able to predict on the raw data from keggle and the predictions were correct.

CONCLUSION

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy , f1 score and also Bar plot which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case) and its come to point of getting the solution for the problem statement being , predicting the quality of wine based on various physicochemical components that are mentioned in the data set.

The quality of the wine can easily be predicted using Machine Learning without the use of any experts that involves in lot of investment. As for a sommelier (a person who tastes the wine) his salary is very much and thus with the use of Machine Learning it can be done efficiently and at a lesser costs there by helping the winery to maintain or develop the quality of the wine.

In order to choose a good or higher quality wine we mainly need to focus on higher contents of citric acid, free sulfur dioxide , sulphates , alcohol and lower content of fixed acidity, volatile acidity, chlorides, total sulfur dioxide.

REFERENCE

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

[2 https://en.wikipedia.org/wiki/Artificial_intelligence

[3]https://www.investopedia.com/terms/d/deep-learning.asp#:~:text=Deep%20learning%20is%20a%20subset,learning%20or%20deep%20neural%20network.

[4]https://en.wikipedia.org/wiki/Decision_tree

[5]https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[6]https://drive.google.com/drive/folders/1bokhl5nDNW9Cyq_LXnhWNHjfzbAWitLO?usp=sharing