MACHINE LEARNING

(MOBILE PRICE CLASSIFICATION)

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

Bachelor of Technology

In

Computer Science Engineering

By

Sai Rohith Pasham

221710315046

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 "**Mobile Price Classification**" to GITAM (Deemed To Be University), Hyderabad in partial fulfilment 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 StudentName

Sai Rohith pasham

Date: StudentRollNo

221710315046

GITAM(DEEMED TO BE UNIVERSITY)
Hyderabad-502329, India

CERTIFICATE

This is to certify that the Industrial Training Report entitled “MOBILE PRICE CLASSIFICATION” is being submitted by Sai Rohith Pasham (221710315046) 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

The machine learning is the field in which the computers can make successful predictions based on the past experiences. It is a field which has shown an impressive development in the recent years with the help of the rapid increase in the storage capacity and processing power of the computers. By using the machine learning algorithms we are going to predict the classification of the mobile price range in our dataset based on its features. Mobile phones are one of the most popular devices of the 21st century and almost everyone has one. So, in this project we design a model that predicts the price range of the mobile that can not only help buyers but also manufacturers about the price they can expect for the particular features.

INDEX

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 organisation.

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

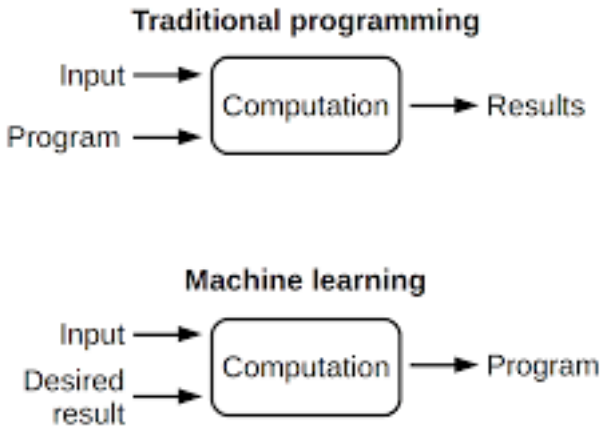

FIGURE 1.3.1 TRADITIONAL PROGRAMMING VS MACHINE LEARNING

- 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 realise 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 labelled.
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 an 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".

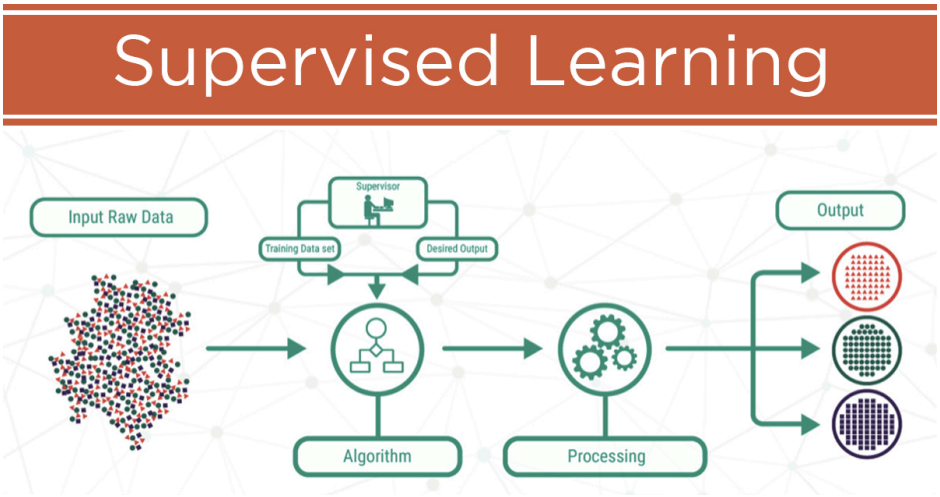

FIGURE 1.5.1.1 SUPERVISED LEARNING

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 categorise it in dogs and cats. But it can categorise them according to their similarities, patterns, and differences i.e., we can easily categorise 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 behaviour.
- **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.

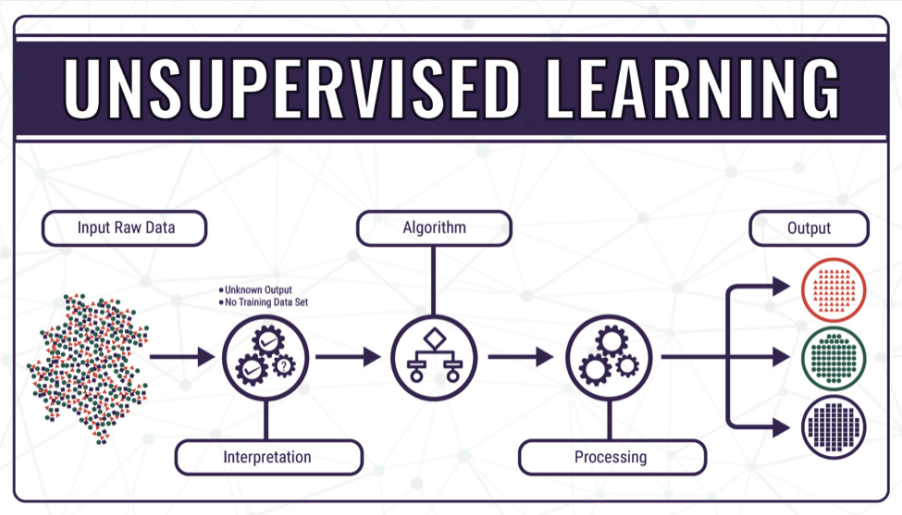

FIGURE 1.5.2.1 UNSUPERVISED LEARNING

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 labelling 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.

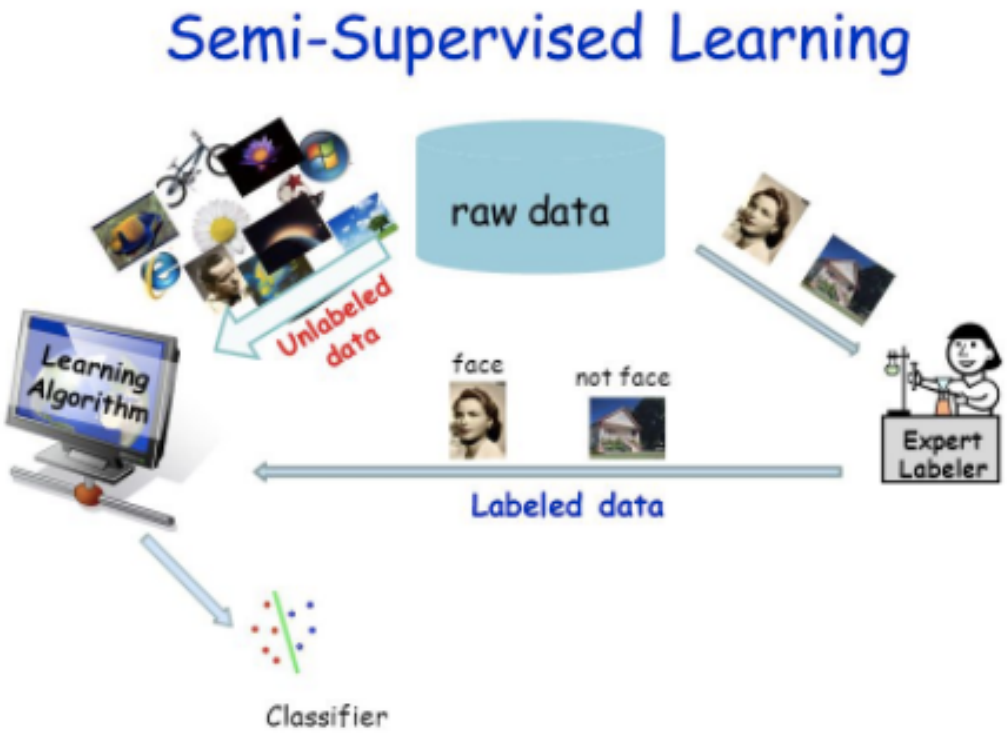

FIGURE 1.5.3.1 SEMI SUPERVISED LEARNING

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.

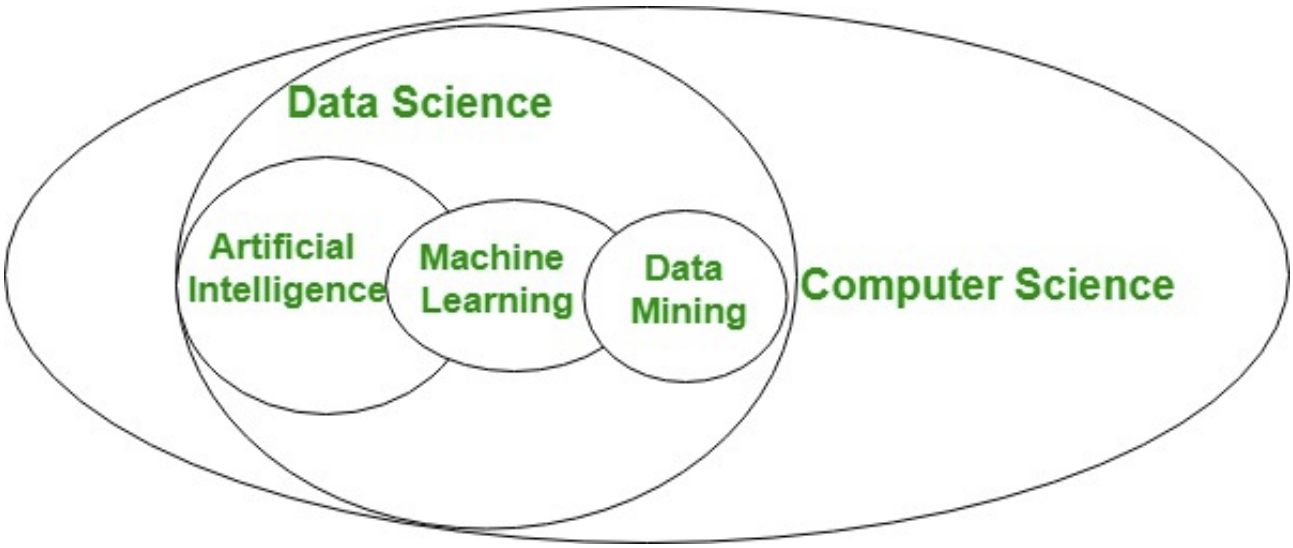

FIGURE 1.6.1 RELATION BETWEEN DATAMINING, DATA SCIENCE AND MACHINE LEARNING

CHAPTER 2 PYTHON

2.1 INTRODUCTION

Python is developed by Guido van Rossum. He started implementing Python in 1989. Python is a very simple programming language so even if you are new to programming, you can learn python without facing any issues.

2.2 FEATURES OF PYTHON

1. **Readable**: Python is a very readable language
2. **Easy to Learn:** Learning python is easy as this is a expressive and high level programming language, which means it is easy to understand the language and thus easy to learn.
3. **Cross platform:** Python is available and can run on various operating systems such as Mac, Windows, Linux, Unix etc. This makes it a cross platform and portable language.
4. **Open Source:** Python is a open source programming language.
5. **Large standard library:** Python comes with a large standard library that has some handy codes and functions which we can use while writing code in Python.
6. **Free:** Python is free to download and use. This means you can download it for free and use it in your application. Python is an example of a FLOSS (Free/Libre Open Source Software), which means you can freely distribute copies of this software, read its source code and modify it.
7. **Supports exception handling:** If you are new, you may wonder what is an exception? An exception is an event that can occur during program exception and can disrupt the normal flow of program. Python supports exception handling which means we can write less error prone code and can test various scenarios that can cause an exception later on.
8. **Advanced features:** Supports generators and list comprehensions. We will cover these features later.
9. **Automatic memory management:** Python supports automatic memory management which means the memory is cleared and freed automatically. You do not have to bother clearing the memory.

2.3 What Can You Do with Python?

There are so many applications of Python and here are some of the them.
1. **Web development** – Web framework like Django and Flask are based on Python. They help you write server side code which helps you manage database, write backend programming logic, mapping urls etc.

2. **Machine learning** – There are many machine learning applications written in Python. Machine learning is a way to write a logic so that a machine can learn and solve a particular problem on its own. For example, products recommendation in websites like Amazon, Flipkart, eBay etc. is a machine learning algorithm that recognises user's interest. Face recognition and Voice recognition in your phone is another example of machine learning.
3. **Data Analysis** – Data analysis and data visualisation in form of charts can also be developed using Python.
4. **Scripting** – Scripting is writing small programs to automate simple tasks such as sending automated response emails etc. Such type of applications can also be written in Python programming language.
5. **Game development** – You can develop games using Python.
6. You can develop Embedded applications in Python.
7. **Desktop applications** – You can develop desktop application in Python using library like TKinter or QT.

2.4 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.

2.4.1 FROM THE PYTHON WEBSITE

- To install the Python on your operating system, go to this link: **https://www.python.org/downloads/**.

FIGURE 2.4.1.1 DOWNLOAD FROM PYTHON WEBSITE

- Now click on the download Python 3.6.4(or whatever the latest version is).
- After downloading, accept all the terms and conditions and you are good to go.

2.4.2 USING ANACONDA

- Python programs can also be executed using anaconda software which contains of Jupyter notebook where we can write the code
- 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.
- Step 1: Open Anaconda.com/downloads in web browser.
- Step 2: Download python 3.4 version for (32-bit graphic 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)

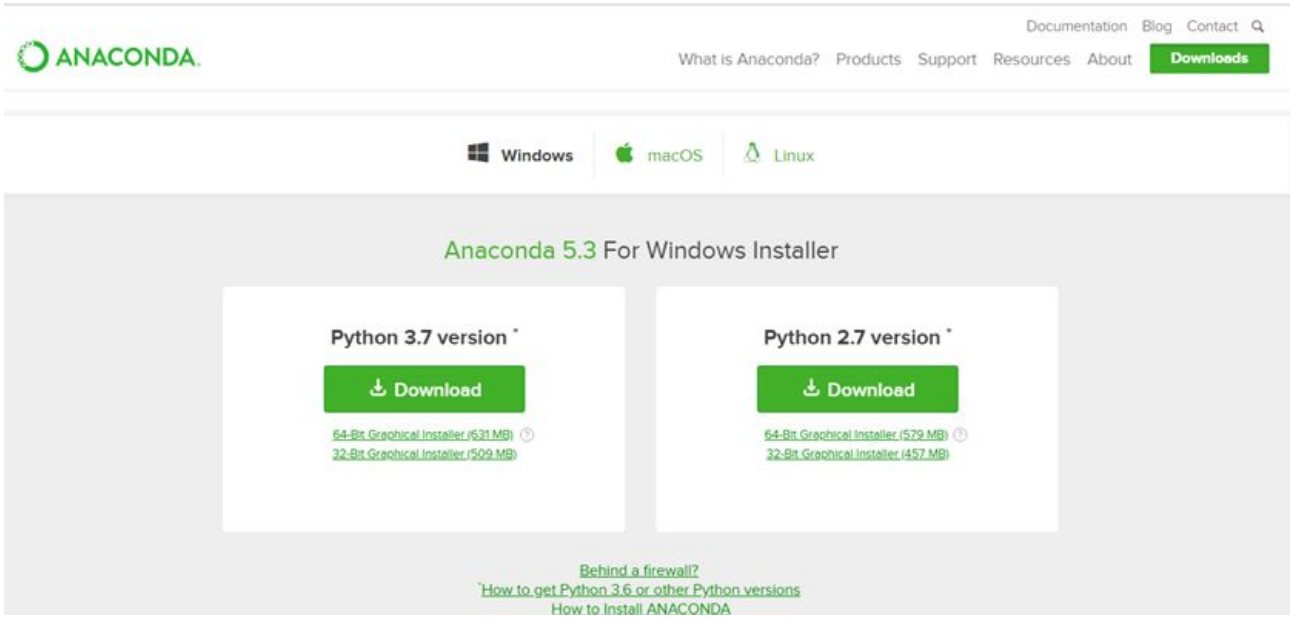

FIGURE 2.4.2.1 DOWNLOAD FROM ANACONDA WEBSITE

2.5 Python variables and datatypes

- Variables are used to store data, they take memory space based on the type of value we assigning to them.
- Creating variables in Python is simple, you just have write the variable name on the left side of = and the value on the right side

Variable name is known as identifier. There are few rules that you have to follow while naming the variables in Python.

1. The name of the variable must always start with either a letter or an underscore (_). For example: _str, str, num, _num are all valid name for the variables.
2. The name of the variable cannot start with a number. For example: 9num is not a valid variable name.
3. The name of the variable cannot have special characters such as %, $, # etc, they can only have alphanumeric characters and underscore (A to Z, a to z, 0-9 or _).
4. Variable name is case sensitive in Python which means num and NUM are two different variables in python.

Data Types

A data type defines the type of data, for example 123 is an integer data while “hello” is a String type of data. The data types in Python are divided in two categories:
1. Immutable data types – Values cannot be changed.
2. Mutable data types – Values can be changed

Immutable data types in Python are:
1. Numbers
2. Strings
3. Tuples

Mutable data types in Python are:
1. List
2. Dictionary
3. Sets

2.5.1 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).

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

- A **list** is a data structure in Python that is a mutable, or changeable, ordered sequence of elements.
- Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ].
- Each item in a list corresponds to an index number, which is an integer value, starting with the index number
- Lists can be sliced i.e only the some elements of the list can be extracted by using the : operator between indexes
- The list data type is a flexible data type that can be modified throughout the course of your program.

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 PythonDictionary

- Python's dictionaries are kind of hash table type.
- 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

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- In Python a function is defined using the def keyword
- To call a function, use the function name followed by parenthesis
- Information can be passed into functions as arguments.

- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma
- To let a function return a value, use the return statement

2.7 Applications 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-maintain.
- A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **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 customise 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.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

CHAPTER-3 CASE STUDY

3.1 PROBLEM STATEMENT

A mobile phone is a device that can make and receive telephone calls over a radio link while moving around a wide geographic area. They have become so popular these days that one can't imagine life without them. There have been many kind of mobile phones that have been released in to the market with different configurations.

So, Here we try to find out the relation between the features and the price of the mobile phone and try to make it easy for anyone who is going to buy a mobile phone by predicting what the price range is for their desired features.

3.2 DATA SET

The given data set contains the following columns:

- id: ID
- battery_power: Capacity of the battery measured in mAh
- blue: Has bluetooth or not
- clock_speed: How many clock cycles the processor can perform and it is measured in GHz
- dual_sim: Has dual sim support or not
- fc: Front Camera mega pixels
- four_g: Has 4G or not
- int_memory: Internal Memory in Gigabytes
- m_dep: Mobile Depth in cm
- mobile_wt: Weight of mobile phone
- n_cores: Number of cores of processor
- pc: Primary Camera mega pixels
- px_height: Pixel Resolution Height
- px_width: Pixel Resolution Width
- ram: Random Access Memory in Megabytes
- sc_h: Screen Height of mobile in cm
- sc_w: Screen Width of mobile in cm
- talk_time: the duration of constant use supported by a device's single fully charged battery while the device is performing a cellular call.
- three_g: Has 3G or not
- touch_screen: Has touch screen or not
- wifi:Has wifi or not

3.3 Objective of the Case Study

The Dataset comprises of different features of mobile phones based on which its Price Range is to be found out. So, Here our objective is to use the classification algorithms on the dataset and classify them according to their price ranges and select the best one that gives the most accuracy.
We have 4 ranges of price and they are
0 - low price range
1 - medium price range
2 - high price range
3 - premium price range

CHAPTER-4: MODEL BUILDING

4.1 PREPROCESSING OF THE DATA

Preprocessing of the data actually involves the following steps:

4.1.1 COLLECTING THE DATASET

The dataset can be collected from multiple sources like from a database or from online data scientists community like kaggle.
The dataset that is used in this project has been taken from kaggle from the following link:
https://www.kaggle.com/iabhishekofficial/mobile-price-classification

4.1.2 IMPORTING THE LIBRARIES

We have to import the libraries as per the requirement 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.
- **seaborn** 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 and the following are the sub packages

- **train_test_split** module is used to split the data into training and testing sets.
- LogisticRegression module is used to fit a LogisticRegression model.
- StandardScaler module is used to scale the data
- accuracy_score is used for the calculation of the accuracy
- classification_report is used to give the classification report for the model.
- RandomForestClassifier is used to fit a Random forest model.
- GradientBoostingClassifier is used to fit a Gradient boosting model
- XGBoost is used to fit a XGBoost classifier.

Importing the Libraries

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

FIGURE 4.1.2.1 IMPORTING THE LIBRARIES

4.1.3 IMPORTING THE 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 data frame. Any missing value or NaN value have to be cleaned.

```
## Reading the dataset
df = pd.read_csv("datasets_11167_15520_train.csv")
df
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2

2000 rows × 21 columns

FIGURE 4.1.3.1 IMPORTING THE DATASET

To know about the number of rows and columns shape we can use shape or we can see it from the bottom left hand corner when we import and view the dataset

Hence we have 2000 rows and 21 columns

4.1.4 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.

Checking for null values in the dataset

- It can be done by using the isnull() function along with the sum() function as shown below

```
# null values in each column
df.isnull().sum()
```

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

FIGURE 4.1.4.1 CHECKING FOR NULL VALUES

As we can see there are no null values in any of the columns in our dataset.

```
#checking for any null values in the dataset
import missingno as msno
msno.matrix(df)
plt.show()
```

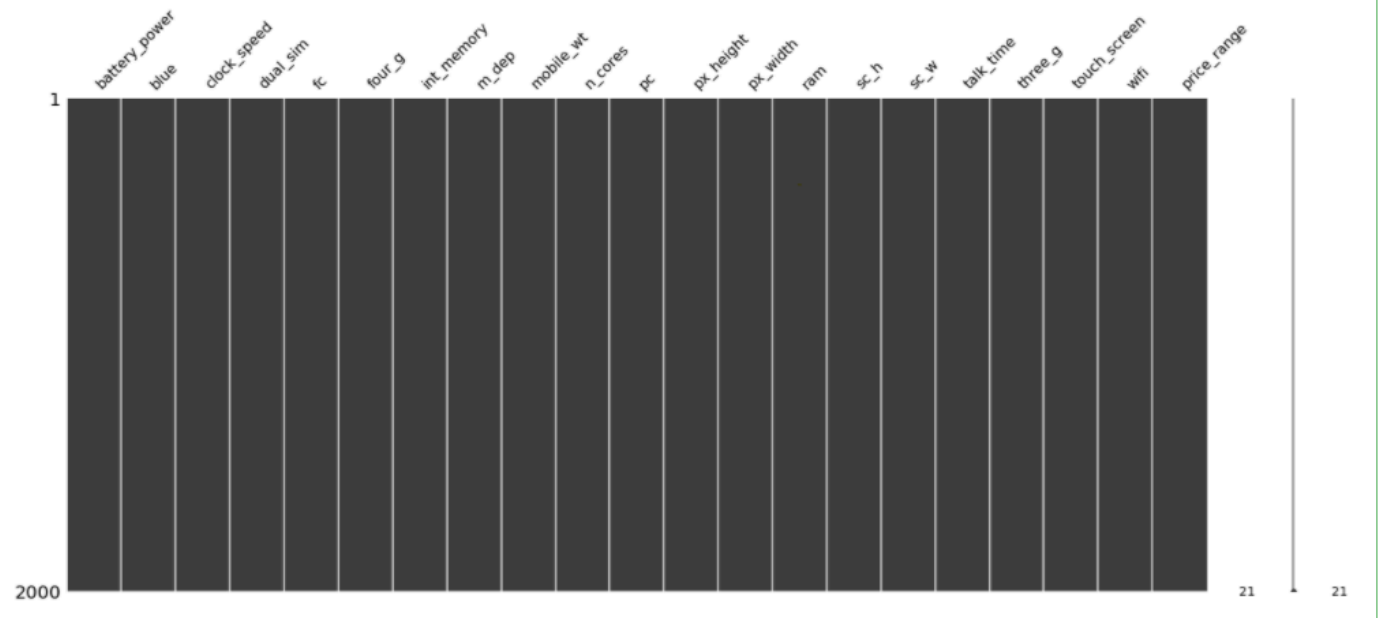

FIGURE 4.1.4.2 CHECKING FOR NULL USING MISSINGNO

We can also visualise the null values using the missingno library.
Using matrix of missingno we can easily visualise the null values of the dataset.

Since we don't have any of the null values, we can clearly observe complete bars in every column and if there was any missing values we would have got incomplete bars.

Data types of the columns

The datatypes of the columns can be known using dataframe_name.dtypes as shown in figure.

```
# datatypes of the columns
df.dtypes
```

```
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

FIGURE 4.1.4.3 CHECKING THE DATATYPES OF THE COLUMNS

There are 19 integer and 2 floating point columns in the dataset.
The statistics of the data can be viewed using the describe function.

```
# Statistical analysis of data
df.describe()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000

FIGURE 4.1.4.4 STATISTICAL ANALYSIS OF THE DATA

It can be used to find the count, mean, max, min … of each of the column

4.2 Correlation

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. Correlation is described as the analysis which lets us know the association or the absence of the relationship between two variables ‘x’ and ‘y’. It is a statistical measure that represents the strength of the connection between pairs of variables.

The correlation of the data can be visualised by using the corr() function.

```
# visualizing the correleation between the columns
corr = df.corr()
plt.figure(figsize=(15,12))
sns.heatmap(corr,annot=True)
plt.show()
```

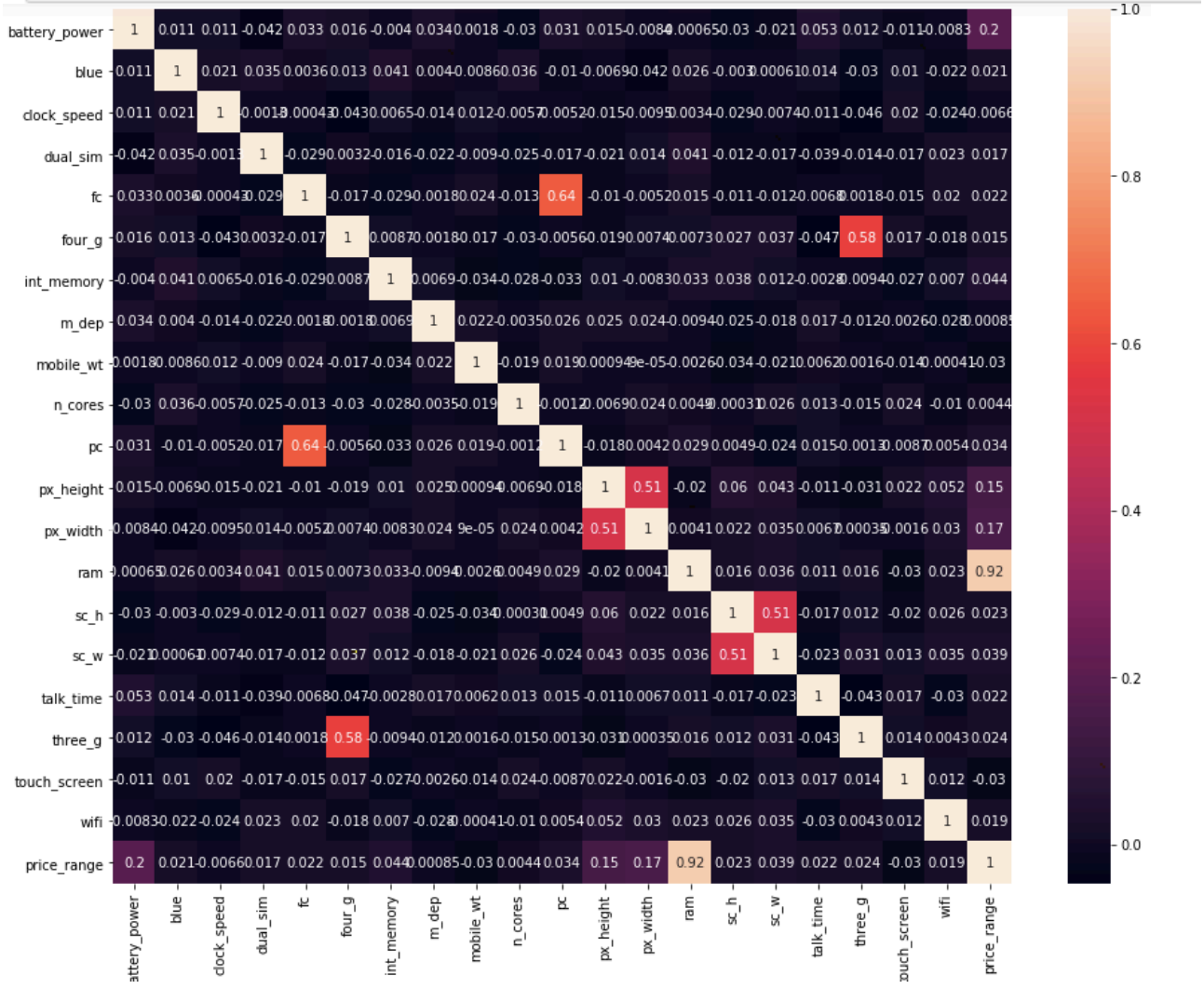

FIGURE 4.2.1 CORRELATION

We can see that the correlation between the columns ranges from -1 to 1.
So now lets try to visualise those columns which have high correlation

```
# highly positive correlation
sns.heatmap(df.corr()[df.corr()>=0.7],annot=True)
plt.show()
```

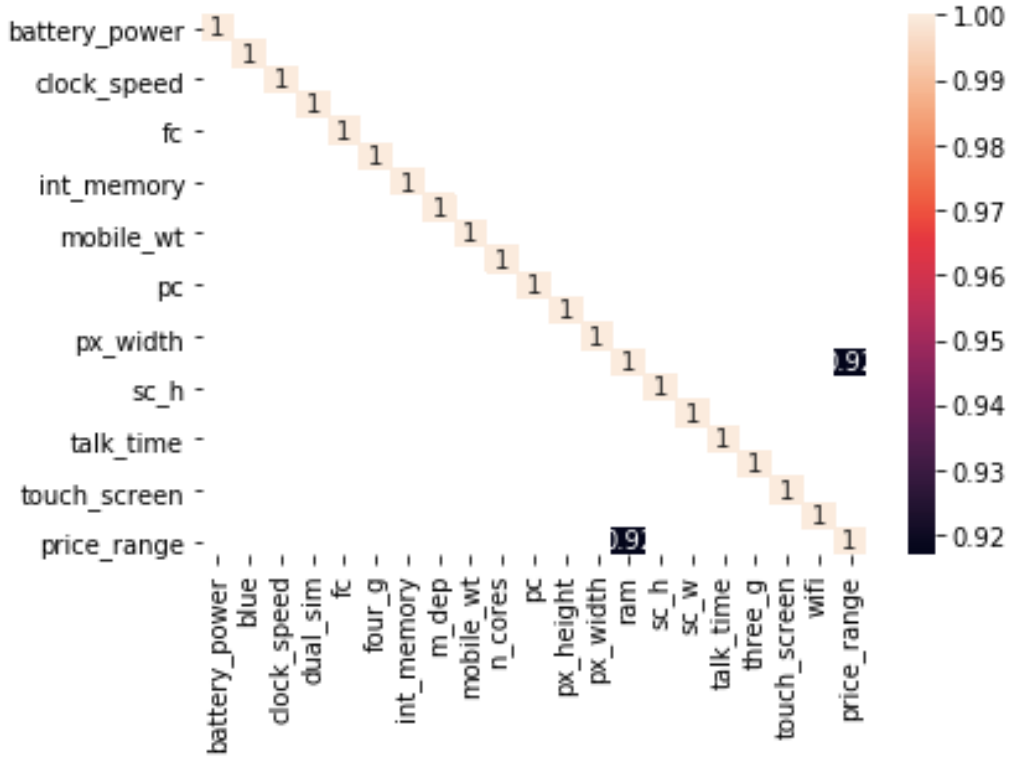

FIGURE 4.2.2 STRONG POSITIVE CORRELATION

We can observe that the ram and the price_range are highly positively correlated.

4.3 DATA VISUALISATION

Since the ram size values range from 0 to 4000 and to make it easy for visualisation, I divided the ram column in to the ram_range column which contains a range of sizes of ram by using the cut attribute of the pandas library.

```
# dividing the ram column into ram range and visualising
bin_labels_9 = ['0-500', '500-1000', '1000-1500', '1500-2000', '2000-2500','2500-3000','3000-3500','3500-4000']
cut_bins = [0,500,1000,1500,2000,2500,3000,3500,4000]
df1['ram_range']=pd.cut(df['ram'],cut_bins,labels=bin_labels_9)
```

FIGURE 4.3.1 CUT FUNCTION

Ram range frequency according to the price range

```
# visualizing ram according to price_range
plt.figure(figsize=(12,8))
sns.countplot("ram_range",data=df1,hue="price_range")
plt.show()
```

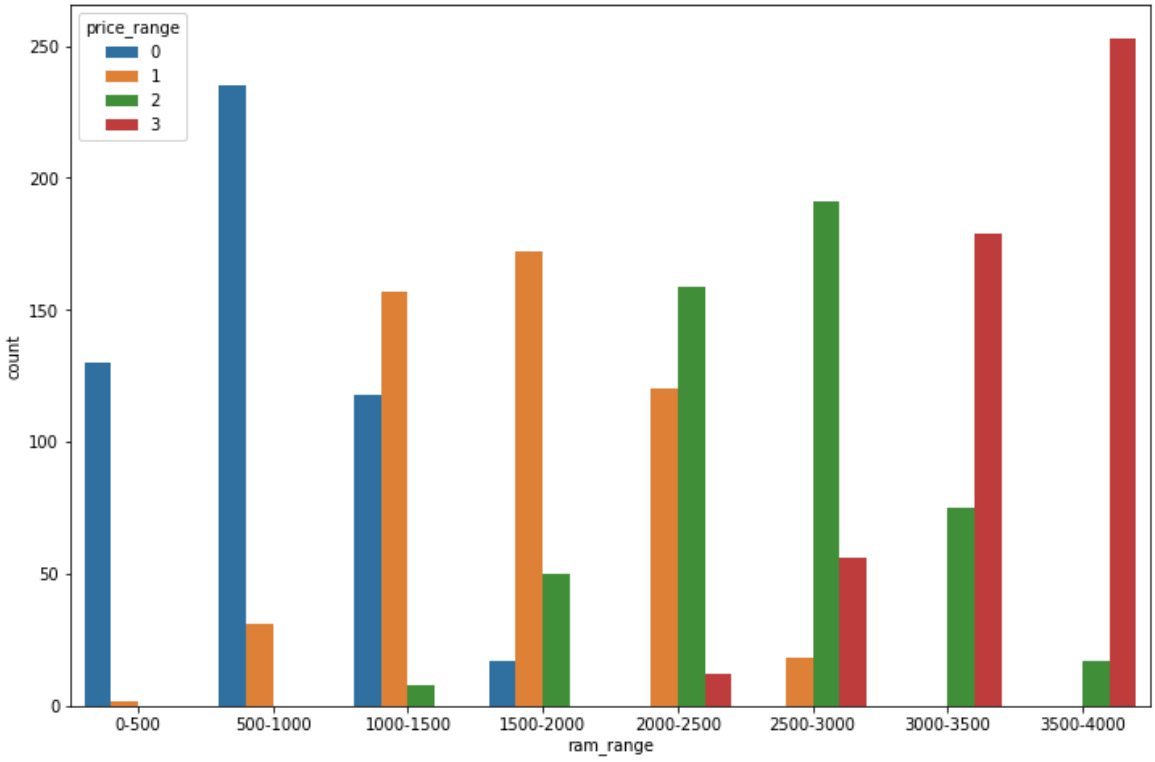

FIGURE 4.3.2 RAM RANGE ACCORDING TO THE PRICE RANGE

From the above count plot we can observe the frequency of the ram size according to the price range.

We can see that in the ram range of 0-500 there are only 2 price ranges: 0 and 1 of which 0 has more frequency. Just like that we can observe for each ram range what price ranges have the highest frequencies.

Frequency of different price ranges

```
# frequency of the price_range
sns.countplot("price_range",data=df)
plt.show()
```

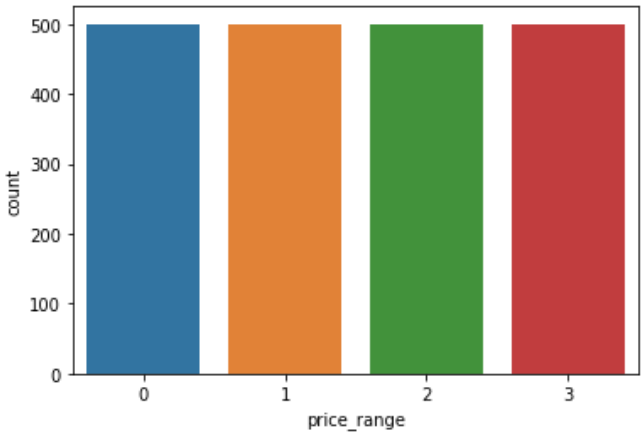

FIGURE 4.3.3 FREQUENCY OF THE PRICE RANGES

From the above count plot we can observe the count of the price ranges.
We can see that all the price ranges have equal number of count of 500.

Ram range vs Battery power

```
# visualising ram vs battery power according to the price_range
plt.figure(figsize=(12,8))
sns.scatterplot("ram","battery_power",hue="price_range",data=df,palette=['red','green','yellow','blue'])
plt.show()
```

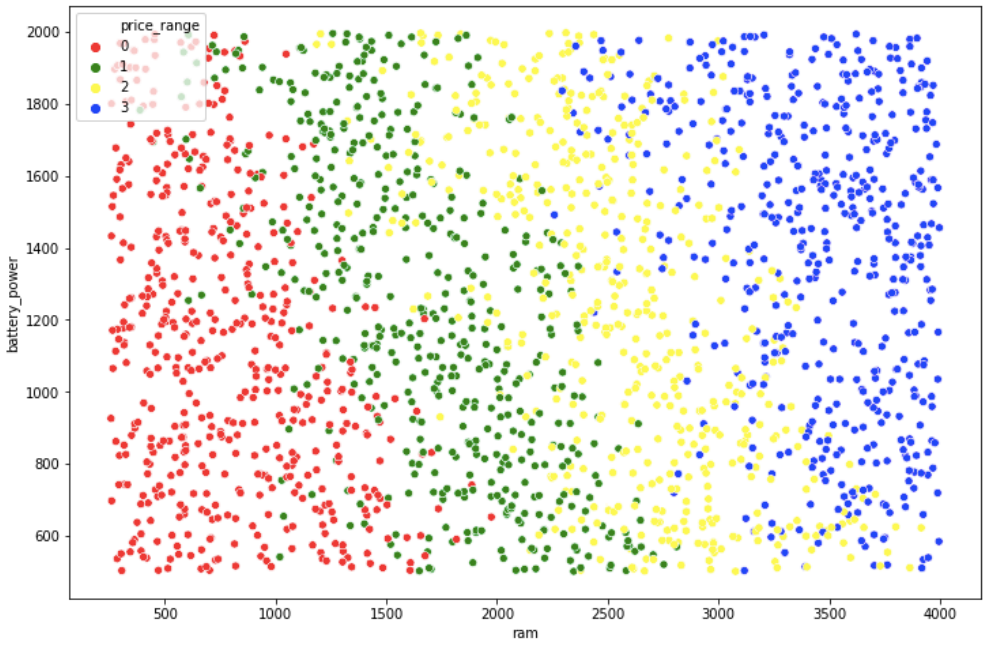

FIGURE 4.3.4 RAM RANGE VS BATTERY

From the above scatter plot we can infer how the price_range is changing according to the ram and battery power.
It can be observed that at high values of ram and at high values of battery power the price range is maximum and as we move towards lower ranges the price range reduces too.

Pie plot to visualise the 4-G and 3-G feature

```
# visualizing the percentage of phones which support 3-g
plt.figure(figsize=(8,8))
label = ["supports 3-G",'Doesnt support']
values=df['three_g'].value_counts().values
plt.pie(values,labels=label,colors=['orange','cyan'],autopct='%1.2f%%')
plt.show()
```

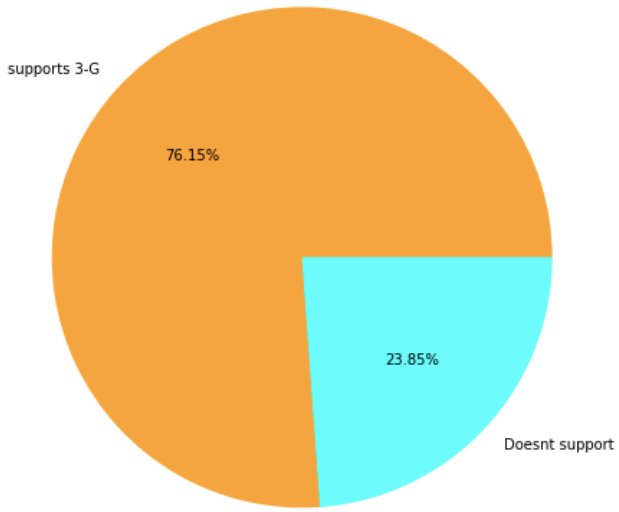

FIGURE 4.3.5 3-G PIE PLOT

From the above pie plot we can infer that 76.15% of the mobiles in our data set supports the 3-G feature and 23.85% of the mobiles don’t support the 3-G feature.

```
# visualizing the percentage of phones which support 4-g
plt.figure(figsize=(8,8))
label = ["supports 4-G",'Doesnt support']
values=df['four_g'].value_counts().values
plt.pie(values,labels=label,colors=['orange','cyan'],autopct='%1.2f%%',startangle=80)
plt.show()
```

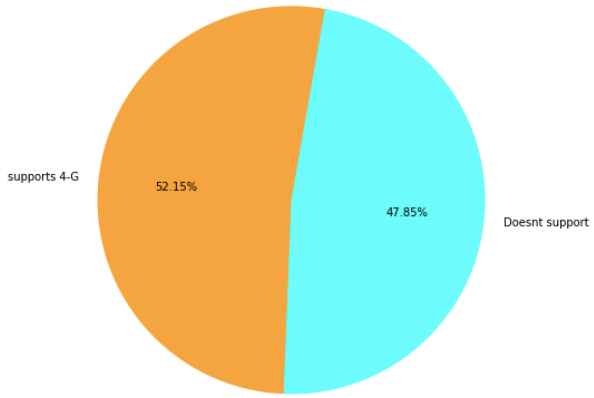

FIGURE 4.3.6 4-G PIEPLOT

From the above pie plot we can understand that the 52.15% of the mobiles in our data set support 4-G feature and 47.85% of the mobiles in our dataset do not support 4-G

Battery power

```
# visualizing the battery power
plt.xlabel("batterypower")
plt.ylabel("count")
plt.hist(df['battery_power'])
plt.show()
```

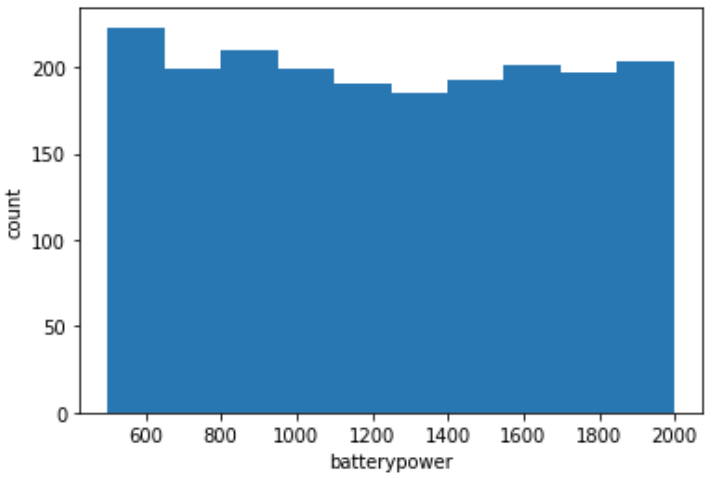

FIGURE 4.3.7 BATTERY POWER

From the above histogram we can observe that the lower battery powered phones are more in count than the higher battery powered phones.

Clock speed frequency according to the price range

From the above plot we can see how many values of each clock speed are distributed according to their price range.

We can clearly see that the most of the mobiles have 0.5 clock speed.

```
# frequency of clock speed according to their price change
plt.figure(figsize=(20,10))
sns.countplot("clock_speed",data=df,hue="price_range")
plt.show()
```

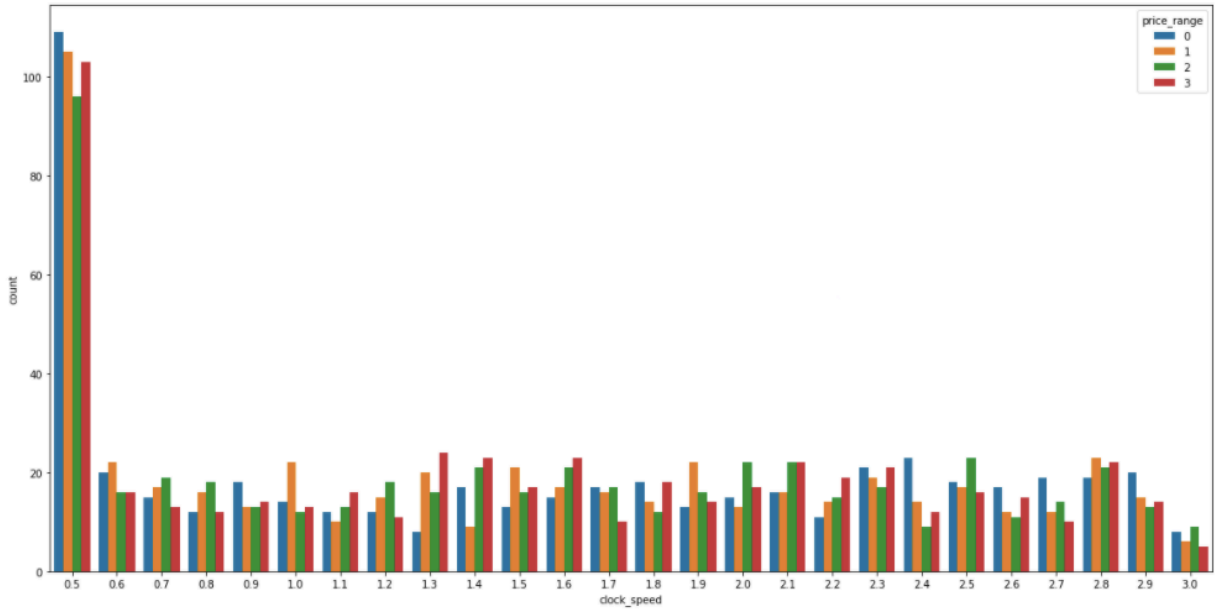

FIGURE 4.3.8 CLOCK SPEED FREQUENCY

Plotting frequency of the front camera and the primary camera Mega Pixels

```
# frequency of the primary camera megapixels
plt.figure(figsize=(10,6))
plt.xlabel("Mega pixels")
plt.ylabel("Frequency")
plt.hist("pc",data=df,bins=10,color="tomato",)
plt.show()
```

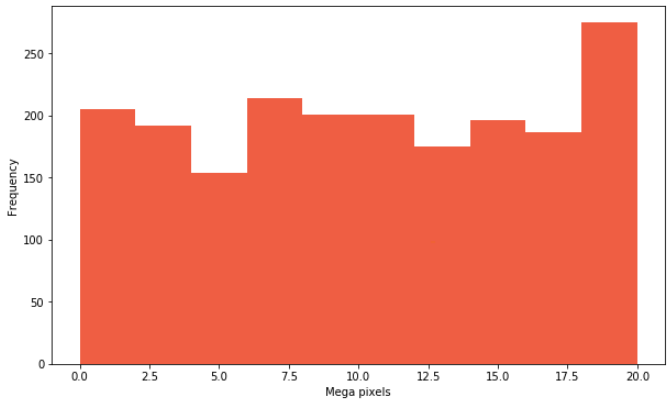

FIGURE 4.3.9 FREQUENCY OF PRIMARY CAMERA MEGA PIXELS

From the above histogram we can observe that most mobiles have 18-20 Megapixels Primary camera

```
# frequency of the front camera megapixels
plt.figure(figsize=(10,6))
plt.xlabel("Mega pixels")
plt.ylabel("Frequency")
plt.hist("fc",data=df,bins=10,color="olive")
plt.show()
```

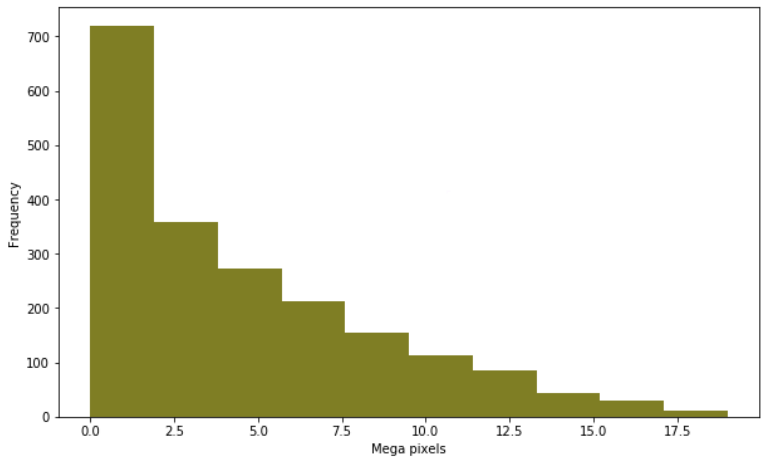

FIGURE 4.3.10 FREQUENCY OF FRONT CAMERA MEGA PIXELS

Similarly here we can observe that the 0-2 Megapixels is the most frequent occurrence in the mobile phones.

Internal memory

```
# internal memory
plt.xlabel("internal memory")
plt.ylabel("count")
plt.hist(df['int_memory'])
plt.show()
```

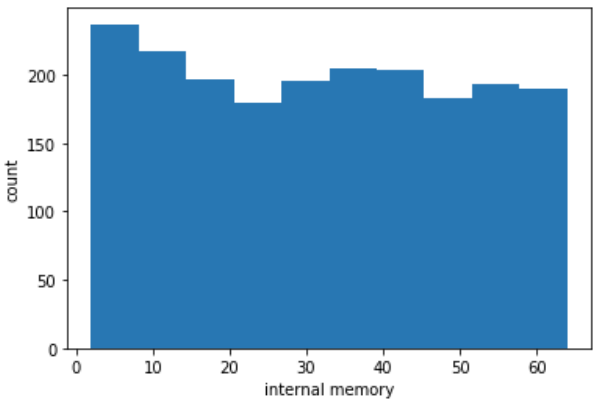

FIGURE 4.3.11 FREQUENCY OF INTERNAL MEMORY

From the above histogram we can observe that more mobiles have lower range of internal memory.

Mobile weight

```
# Mobile weight
plt.xlabel("mobile weight")
plt.ylabel("count")
plt.hist(df['mobile_wt'])
plt.show()
```

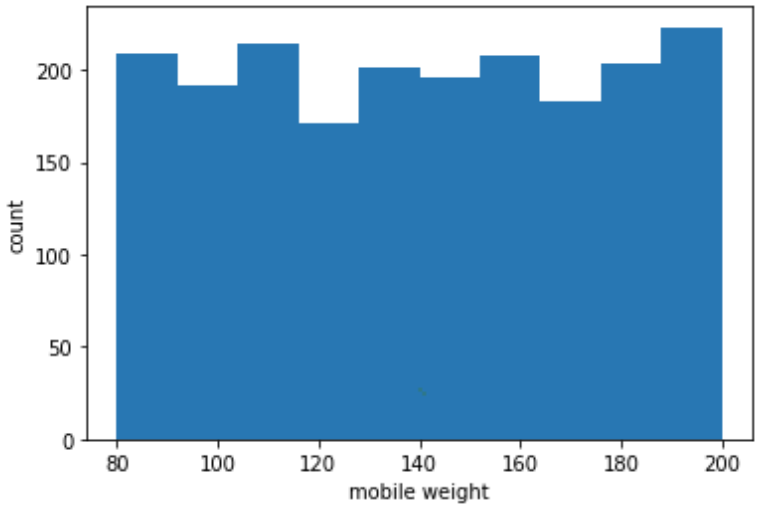

FIGURE 4.3.12 FREQUENCY OFMOBILE WEIGHT

From the above plot, we can see that the weights are evenly distributed.

Wifi

```
In [37]: # wifi vs price range
         plt.figure(figsize=(6,6))
         sns.countplot(df['wifi'],hue=df['price_range'])
         plt.legend(loc="lower right")
         plt.show()
```

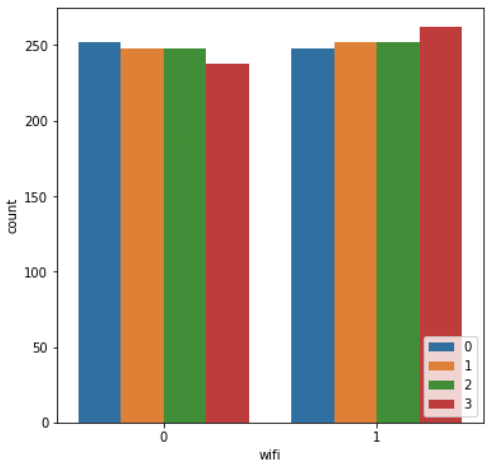

FIGURE 4.3.13 WIFI ACCORDING TO PRICE RANGE

In the above count plot, the prices of mobiles increases considerably for mobiles with wifi

So having wifi or not plays an important role in the classification.

Dual sim

```
In [38]: # price range according to dual sim
         plt.figure(figsize=(6,6))
         sns.countplot(df['dual_sim'],hue=df['price_range'])
         plt.legend(loc="lower right")
         plt.show()
```

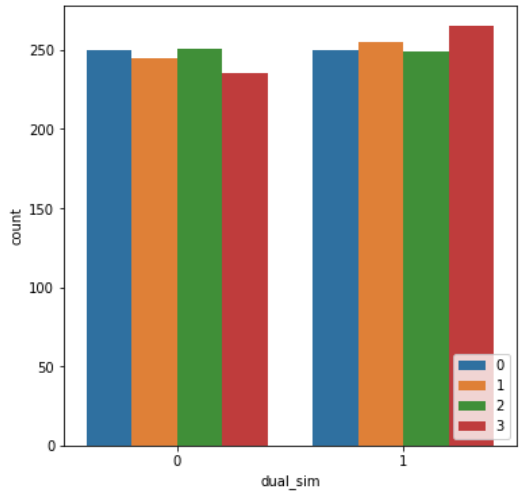

FIGURE 4.3.14 PRICE RANGE ACCORDING TO DUAL SIM

From the above count plot we can see that the price of mobiles increases considerably for the mobiles with dual sim, so this plays an important role in the classification.

Four-G and Three-G vs the price range

```
In [40]: plt.figure(figsize=(6,6))
         sns.countplot(df['four_g'],hue=df['price_range'])
         plt.legend(loc="lower right")
         plt.show()
```

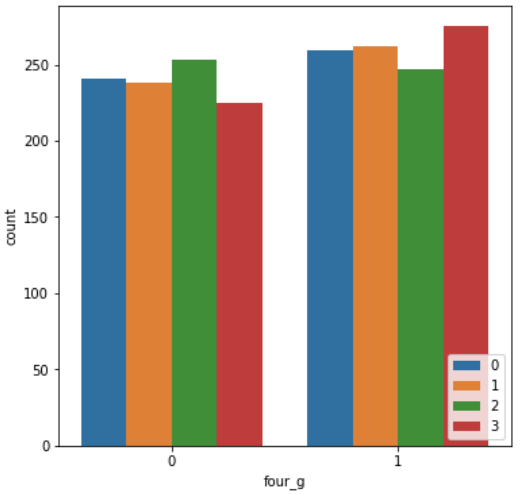

FIGURE 4.3.15 PRICE RANGE ACCORDING TO 4-G

```
In [42]: plt.figure(figsize=(6,6))
         sns.countplot(df['three_g'],hue=df['price_range'])
         plt.legend(loc="lower right")
         plt.show()
```

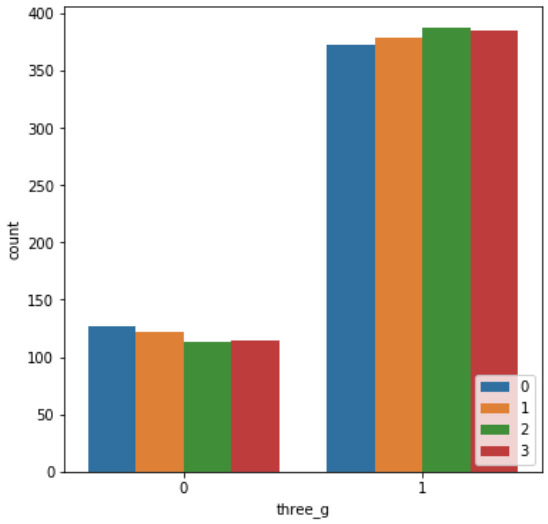

FIGURE 4.3.16 PRICE RANGE ACCORDING TO 3-G

From the above 2 count plots we can see that the price of mobiles increases considerably for the mobiles with 4-G and 3-G, so they play an important role in the classification.

4.4 Data Splitting

Now we need to split the data in to training and testing

The training data is used to build the model and the testing data is used to find the accuracy of the model.

In this dataset, we consider all the rows except the price_range as the inputs and price_range as output.

X is used to represent the input and y as output and then after splitting we denote X_train for the input training data and X_test for the input of testing data. Similarly for the outputs we use y_train for the output for the training data and y_test for the output values of the testing data.

First we need to assign the input and outputs as shown below

```
# X-input,y=output
X = df.drop(['price_range'],axis=1)
y = df['price_range']
```

FIGURE 4.4.1 INPUT AND OUTPUT

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.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=18)
```

FIGURE 4.4.2 TRAIN TEST SPLIT

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.

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

```
(1600, 20)
(400, 20)
(1600,)
(400,)
```

FIGURE 4.4.3 SHAPE OF THE TRAINING AND TESTING DATA

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

Scaling the training and test data

```
In [32]: 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
```

Out[32]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	
0	0.380924	-0.981423	-1.267619	0.977748	-0.530357	0.958366	0.102975	1.022485	-1.217939	-0.679028	-0.814322	0.856761	0.263889	1.(
1	-0.767419	-0.981423	-1.267619	0.977748	1.987658	0.958366	1.361102	1.369973	0.885540	-1.114826	1.306893	0.332430	0.182497	-1.(
2	-1.510331	-0.981423	-1.144996	-1.022759	-0.759267	0.958366	0.595286	-0.019981	0.004353	-0.679028	-0.324811	-0.958937	-1.091870	0.4
3	-1.476356	-0.981423	1.552719	-1.022759	-0.530357	-1.043443	-0.936348	-1.062446	-1.644320	-0.679028	0.164700	-0.920013	-0.926760	0.4
4	-0.626991	1.018929	-0.041385	0.977748	1.300927	-1.043443	1.032895	-0.714958	0.317032	0.628367	0.980552	-0.203350	-1.468598	0.7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
395	0.192931	1.018929	1.184849	0.977748	1.300927	0.958366	-0.225232	-1.062446	-0.279901	-0.243230	0.327870	-1.100896	-1.473249	-0.2
396	0.756909	1.018929	-0.899749	0.977748	0.614195	0.958366	-1.319256	1.022485	0.800263	0.192568	-0.324811	-0.150688	-0.036099	1.2
397	-0.029037	-0.981423	-1.022373	0.977748	0.385285	-1.043443	1.525206	-1.409935	-0.962111	-1.114826	0.817382	-1.448924	0.731312	0.2
398	0.460198	-0.981423	-1.144996	0.977748	-0.988178	0.958366	1.142298	-0.019981	-0.990536	-1.550625	0.327870	-0.890247	-0.694211	-0.(
399	-0.248740	1.018929	-0.164009	-1.022759	1.758748	-1.043443	1.142298	-0.367469	-1.615894	0.192568	0.654211	-0.077419	0.273191	0.7

400 rows × 20 columns

FIGURE 4.4.5 SCALING THE 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 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.

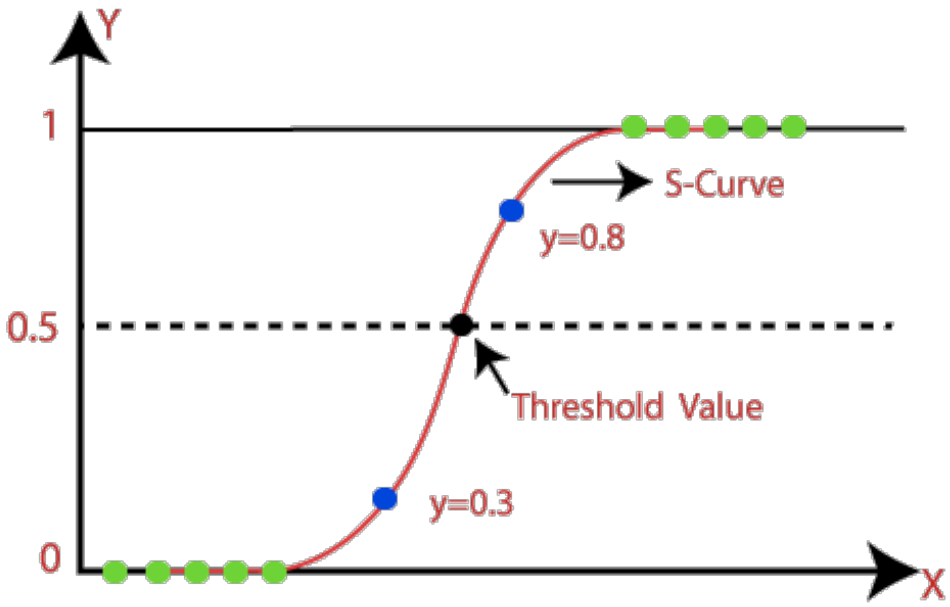

FIGURE 4.5.1.1 LOGISTIC REGRESSION

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.

Logistic regression

```
In [33]: # importing the model class
         from sklearn.linear_model import LogisticRegression
         # creating a object for that model
         lm = LogisticRegression()
         # fitting the input and output of training data to the object and building the model.
         lm.fit(scaled_X_train,y_train)

Out[33]: 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 4.5.1.2 BUILDING THE MODEL

Now that we have built the model, next step is to check the accuracy of the training data and inorder 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

```
In [34]: # predicting the output of the training data
         y_train_pred = lm.predict(scaled_X_train)
         y_train_pred

Out[34]: array([1, 2, 2, ..., 1, 2, 0])
```

FIGURE 4.5.1.3 PREDICTING THE OUTPUT

Now that we got the predicted values for the training data, next we need to test the accuracy by using the accuracy_score as shown below

As we can see that we have got an accuracy score of 0.98 which is 98% on the training data.

Now we need to do the same steps for the training data and check for its accuracy

```
In [35]: from sklearn.metrics import accuracy_score
         print("Training data accuracy:",accuracy_score(y_train,y_train_pred))

         Training data accuracy: 0.98125
```

FIGURE 4.5.1.4 TRAINING ACCURACY

```
In [36]: y_test_pred = lm.predict(scaled_X_test)
         y_test_pred

Out[36]: array([3, 0, 1, 1, 2, 1, 3, 0, 0, 3, 3, 1, 2, 2, 3, 2, 2, 0, 0, 1, 3, 1,
                0, 1, 2, 0, 2, 0, 0, 0, 3, 0, 1, 3, 0, 1, 3, 1, 2, 1, 3, 3, 3, 2,
                3, 2, 0, 2, 3, 1, 3, 2, 0, 2, 1, 1, 0, 3, 1, 0, 0, 1, 0, 1, 0, 0,
                0, 2, 3, 0, 2, 2, 3, 0, 2, 0, 0, 3, 0, 2, 1, 3, 3, 2, 0, 0, 1, 2,
                2, 3, 0, 1, 0, 2, 1, 2, 3, 1, 3, 0, 1, 2, 3, 2, 2, 2, 0, 1, 2, 0,
                3, 3, 2, 3, 0, 1, 2, 3, 1, 1, 0, 2, 1, 0, 3, 2, 1, 2, 1, 2, 3, 1,
                2, 0, 3, 1, 2, 3, 0, 2, 1, 1, 2, 1, 3, 1, 3, 1, 1, 1, 3, 1, 3, 0,
                1, 2, 3, 3, 3, 1, 3, 3, 0, 0, 2, 1, 1, 0, 0, 2, 0, 3, 0, 0, 0, 2,
                2, 0, 3, 1, 2, 1, 2, 1, 3, 0, 3, 2, 2, 3, 1, 2, 0, 2, 3, 2, 1, 3,
                1, 0, 3, 0, 1, 1, 1, 0, 3, 1, 1, 2, 2, 3, 0, 3, 1, 2, 2, 3, 3, 0,
                0, 0, 0, 3, 2, 1, 3, 2, 0, 2, 0, 1, 2, 1, 3, 1, 1, 2, 3, 2, 0, 2,
                2, 0, 3, 1, 0, 3, 2, 0, 0, 1, 0, 1, 1, 3, 0, 2, 1, 1, 1, 2, 0, 3,
                0, 3, 2, 2, 0, 1, 1, 1, 0, 1, 3, 2, 1, 3, 0, 1, 1, 0, 0, 0, 1, 1,
                2, 0, 0, 1, 1, 3, 0, 2, 0, 2, 0, 3, 1, 1, 2, 1, 2, 1, 0, 2, 3, 1,
                2, 0, 2, 3, 3, 1, 1, 3, 0, 3, 3, 3, 0, 0, 1, 3, 0, 0, 2, 3, 1, 3,
                3, 2, 1, 1, 3, 3, 2, 2, 1, 0, 2, 2, 0, 0, 2, 1, 1, 1, 1, 1, 3, 0,
                0, 0, 0, 1, 0, 1, 2, 1, 3, 2, 0, 2, 2, 2, 3, 3, 1, 3, 0, 0, 3, 2,
                3, 3, 2, 3, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 0, 2, 0, 2, 0, 2, 2, 1,
                3, 2, 1, 3])

In [37]: print("Testing data accuracy:",accuracy_score(y_test,y_test_pred))

         Testing data accuracy: 0.965
```

FIGURE 4.5.1.5 TESTING ACCURACY

The accuracy that we got for the testing data is 0.965 i.e 96.5%.

4.5.2 Random Forest Classifier

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.

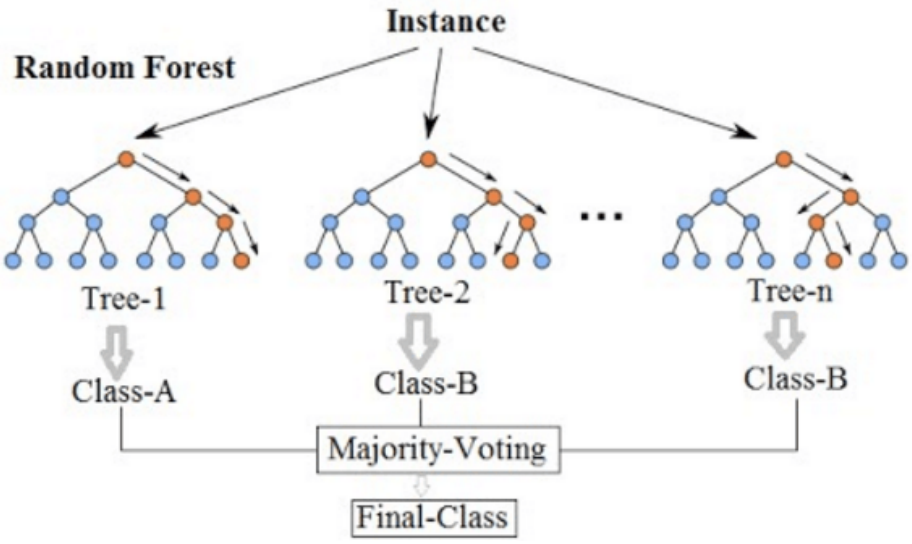

FIGURE 4.5.2.1 RANDOM FOREST

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.

```
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 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 inorder 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

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

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       403
           1       1.00      1.00      1.00       391
           2       1.00      1.00      1.00       394
           3       1.00      1.00      1.00       412

    accuracy                           1.00      1600
   macro avg       1.00      1.00      1.00      1600
weighted avg       1.00      1.00      1.00      1600
```

FIGURE 4.5.2.3 TRAIN CLASSIFICATION REPORT

```
y_pred_test = rfc.predict(X_test)
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        97
           1       0.85      0.81      0.83       109
           2       0.86      0.80      0.83       106
           3       0.88      0.94      0.91        88

    accuracy                           0.87       400
   macro avg       0.87      0.87      0.87       400
weighted avg       0.87      0.87      0.87       400
```

```
print("Accuracy of the testing data=",accuracy_score(y_test,y_pred_test))
```

```
Accuracy of the testing data= 0.8675
```

FIGURE 4.5.2.4 TEST CLASSIFICATION REPORT

Classification report:

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report

Confusion Matrix

Confusion matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

It is extremely useful for measuring Recall, Precision, Specificity, Accuracy.

Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

```
In [40]: from sklearn.metrics import confusion_matrix
         sns.heatmap(confusion_matrix(y_test,y_test_pred), annot=True,
                     fmt='d', annot_kws={'va':'top','ha':'right'})
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe7235545d0>
```

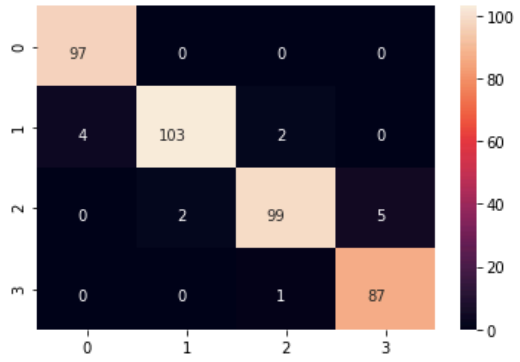

FIGURE 4.5.2.5 CONFUSION MATRIX

From the above figure we visualised the confusion matrix using the heat map and confusion_matrix of the sklearn.metrics package.

Here, we can observe that the diagonal values are correctly classified classes and all the non diagonals are the mis-classified classified.

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

```
print("Accuracy of the training data=",accuracy_score(y_train,y_pred_train))
Accuracy of the training data= 1.0
```

FIGURE 4.5.2.6 TRAIN ACCURACY

By using the accuracy_score metric we have calculated the accuracy of the training data as 1.0 i.e 100%.

So, the accuracy of the testing data is calculated to be 0.867 or 86.7%
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

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

```
0.875625
```

FIGURE 4.5.2.7 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 0.875 or 87.5% 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 initialised before training a model.
Hyperparameters are important because they directly control the behaviour 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 trail and error, it is very difficult to find out which hyperparameters will give the optimal value so, it is trail 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 optimised 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
First we need to create a dictionary with all the parameters that we want to change

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

FIGURE 4.5.2.8 HYPER PARAMETERS

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 4.5.2.9 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': 'entropy', 'max_depth': 9, 'min_samples_leaf': 4}
```

FIGURE 4.5.2.10 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 9 and min_samples_leaf as 4

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

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(criterion= 'gini', max_depth= 9, min_samples_leaf= 4)
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)
```

```
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

           0       1.00      1.00      1.00       403
           1       1.00      1.00      1.00       391
           2       1.00      1.00      1.00       394
           3       1.00      1.00      1.00       412

    accuracy                           1.00      1600
   macro avg       1.00      1.00      1.00      1600
weighted avg       1.00      1.00      1.00      1600
```

FIGURE 4.5.2.11 TRAINING ACCURACY

Here we can observe that the accuracy of the training data is 100%

So now let's check for the test data.

```
y_pred_test = rfc.predict(X_test)
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91        97
           1       0.86      0.82      0.84       109
           2       0.86      0.84      0.85       106
           3       0.90      0.93      0.92        88

    accuracy                           0.88       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.87      0.88      0.87       400
```

FIGURE 4.5.2.12 TEST ACCURACY

The accuracy of the testing data is 0.88 or 88% and it is slightly increased from the previous models.

4.5.3 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 optimised.
- A weak learner to make predictions.
- An additive model to add weak learners to minimise the loss function.

Gradient boosting is a greedy algorithm and can overfit a training dataset quickly.
It can benefit from regularisation methods that penalise 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

```
from sklearn.ensemble import GradientBoostingClassifier
gdc = GradientBoostingClassifier(learning_rate=0.5,n_estimators=100)
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))
```

```
train accuracy:  1.0
test accuracy:  0.9125
```

FIGURE 4.5.3.1 GBC ACCURACY

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.

Here we can observe the performance of the gradient boosting classifier model.
We got the accuracy of the train data as 1.0 or 100%
We got the accuracy of the test data as 0.9125 or 91.25%

4.5.4 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 regularised version of GBM.

The features of XGBoost are as follows

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

2. Parallel Processing: XGBoost utilises 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 up to 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))
```

```
train accuracy:  1.0
test accuracy:  0.9
```

FIGURE 4.5.4.1 XGB ACCURACY

Here we can observe the performance of the gradient boosting classifier model.

We got the accuracy of the train data as 1.0 or 100%

We got the accuracy of the test data as 0.9 or 90%

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.

```
In [102]: train_acc = {'Logistic_train':log_train,'RandomForest_train':ran_train,'GradientBoost_train':grad_train,'XGBoost_train'
          test_acc = {'Logistic_test':log_test,'RandomForest_test':ran_test,'GradientBoost_test':grad_test,'XGBoost_test':xg_test
          x = train_acc.keys()
          y = train_acc.values()
          p = test_acc.keys()
          q = test_acc.values()
```

FIGURE 4.6.1 MAKING DICTIONARIES FOR ACCURACIES

I have created two dictionaries of all the accuracies of training and testing data and labels and now we use bar plot to visualise it as shown above

```
plt.figure(figsize=(10,6))
plt.xlabel("Classifier")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparition")
plt.bar(list(x),list(y))
plt.show()
```

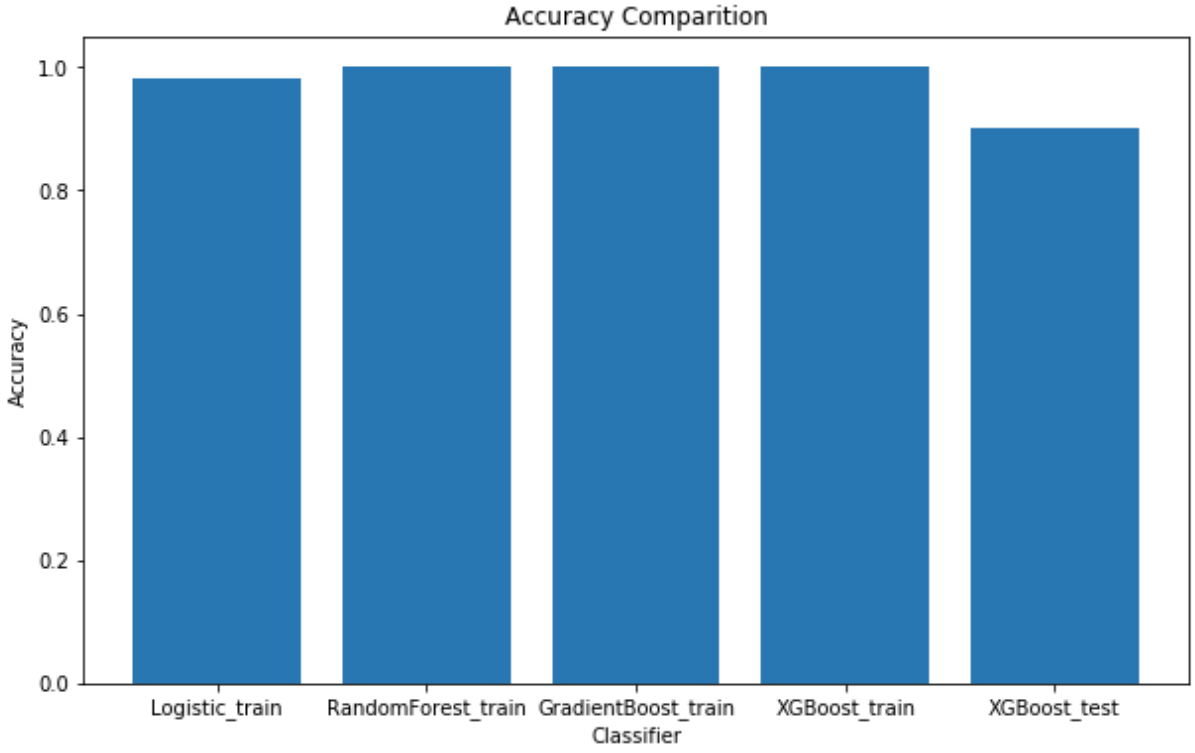

FIGURE 4.6.2 TRAIN MODELS ACCURACY

The above plot plots the accuracies of all the models on train data.

```
plt.figure(figsize=(10,6))
plt.xlabel("Classifier")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparition")
plt.bar(list(p),list(q))
plt.show()
```

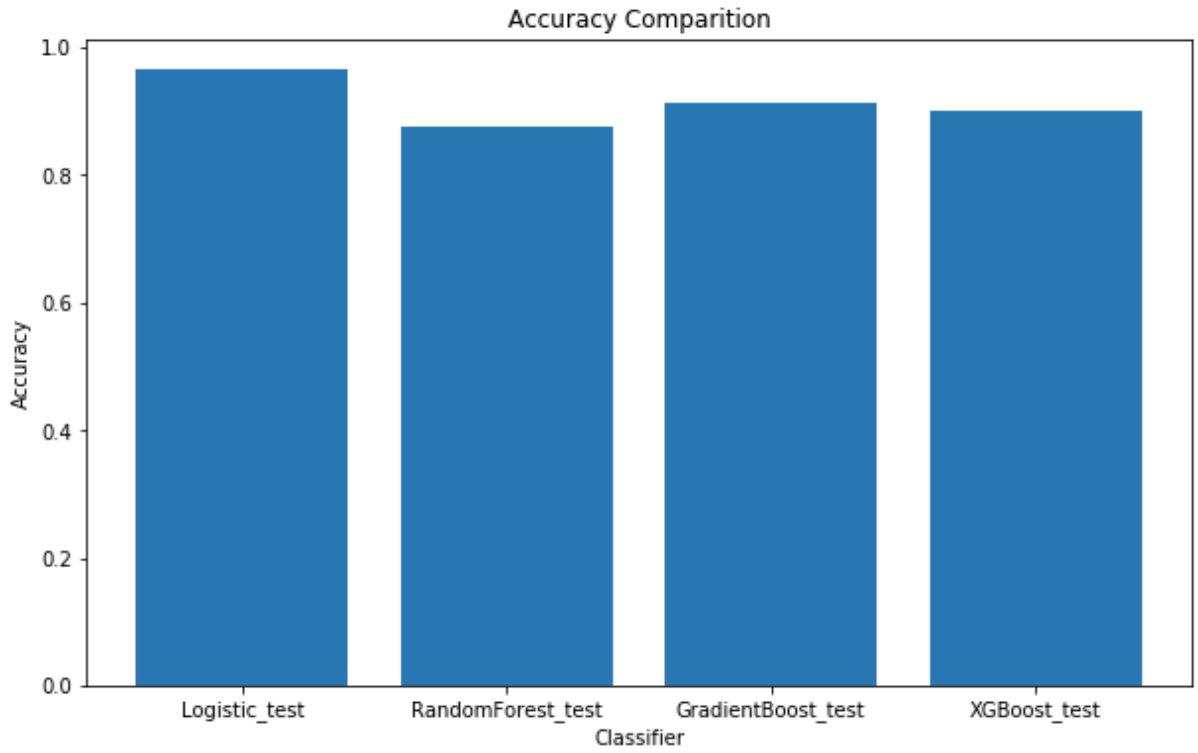

FIGURE 4.6.3 TEST MODELS ACCURACY

The above plot plots the accuracies of all the models on test data.

From the above 2 bar plots we can infer that the logistic regression model is in fact the best performer of all the models.

So logistic regression is the best predictive algorithm for our data.

Testing the model with a sample input using logistic regression

So first we build the model and then pass the sample input to the predict function.

```
# building the model.
from sklearn.linear_model import LogisticRegression
lm = LogisticRegression()
lm.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 4.6.4 BUILD THE BEST MODEL

```
input_sample = [[1043,1,1.8,1,14,0,5,0.1,193,3,16,226,1412,3476,12,7,2,0,1,0]]
a = int(lm.predict(input_sample))
list4 = ['low','medium','high','premium']
print(list4[a])
```

```
premium
```

FIGURE 4.6.5 PREDICTING THE PRICE RANGE FOR A SAMPLE INPUT

we predicted an output of premium which means we got the price range as 3

4.7 Conclusion

In this particular project, we observed the key features which affect the mobile phone prices by using various machine learning algorithms. From all the algorithms, I've found that logistic regression performed the best with an accuracy of about 98%.

So this model can be used to predict the mobile phone prices accurately.

4.8 References

- https://www.kaggle.com/iabhishekofficial/mobile-price-classification
- 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://github.com/sairohithpasham/mobile-price-classification/tree/master