# MACHINE LEARNING

(Mobile Price Classification)

*Summer Internship Report Submitted in partial*

*fulfillment of the requirement for undergraduate degree*

*of*

**Bachelor of Technology**

In

*COMPUTER SCIENCE AND ENGINEERING*

By

**OBULASETTI VENKANNA BABU**
**221710304041**

*Under the Guidance of*

**Mr.** Assistant Professor

Department Of Electronics and Communication Engineering
GITAM School of Technology

GITAM (Deemed to be University)
Hyderabad-502329

June 2019

# DECLARATION

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

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

Place: Hyderabad | Name: OBULASETTI VENKANNA BABU

Date: 12-07-2020 | Student Roll No: 221710304041 ii

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

**C ERTIFICATE**

This is to certify that the Industrial Training Report entitled **"MOBILE PRICE CLASSIFICATION"** is being submitted by O. Venkanna Babu (221710304041) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer science and Engineering** at GITAM (Deemed to Be University), Hyderabad during the academic year 2018-19

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

**Dr.S. Phani Kumar**

| | |
|---|---|
| Assistant Professor | Professor and HOD |
| Department of CSE | Department of CSE |

## ACKNOWLEDGEMENT

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

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

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

I would like to thank the respected faculties 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.

OBULASETTI VENKANNA BABU
221710304041

# ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on data set.

To classify “If the mobile with given features will be Economical or Expensive” is the main motive of this project. Real Dataset is collected from website. I have adapted the view point of looking at features of the dataset, for deep understanding of the problem. I have taken the stance of a seller and reasoned out the various factors of choice of the mobile’s customer buys differs from other companies. Different Scaling algorithms are used to identify and normalize the range of independent variables or features of dataset. It is used in data preprocessing and used to normalize minimum computational complexity. Different classifiers are used to achieve as higher accuracy as possible. Results are compared in terms of highest accuracy achieved and minimum features selected. Conclusion is made on the base of best selection algorithm and best classifier for the given dataset. This work can be used in any type of marketing and business to find optimal product (with minimum cost and maximum features). To classify the mobile price range.

# Table of Contents

## LIST OF FIGURES

# 1.MACHINELEARNING

## 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).

Human designers often produce machines that do not work as well as desired in the environments in which they are used. In fact, certain characteristics of the working environment might not be completely known at design time. Machine learning methods can be used for on-the-job improvement of existing machine designs. The amount of knowledge available about certain tasks might be too large for explicit encoding by humans. Machines that learn this knowledge gradually might be able to capture more of it than humans would want to write down. Environments change over time. Machines that can adapt to a changing environment would reduce the need for constant redesign. New knowledge about tasks is constantly being discovered by humans. Vocabulary changes. There is a constant stream of new events in the world. Continuing redesign of AI systems to conform to new knowledge is impractical, but machine learning methods might be able to track much of it.

## 1.2 IMPORTANCE OF MACHINE LEARNING:

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

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

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

The process flow depicted here represents how machine learning works

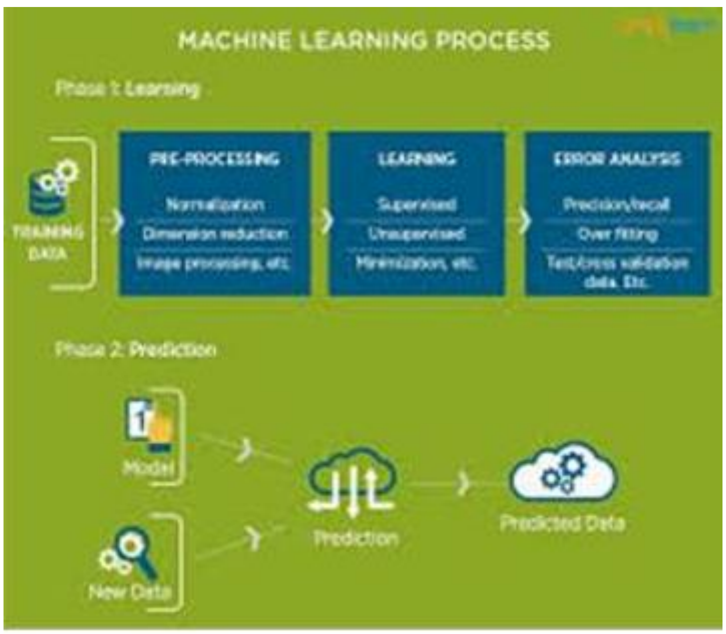



*Figure 1: The Process Flow*

## 1.3 USES OF MACHINE LEARNING:

Artificial Intelligence (AI) is everywhere. Possibility is that you are using it in one way or the other and you don't even know about it. One of the popular applications of AI is Machine Learning (ML), in which computers, software, and devices perform via cognition (very similar to human brain). Herein, we share few examples of machine learning that we use every day and perhaps have no idea that they are driven by ML. These are some the uses and applications of ML

**I. Virtual Personal Assistants:**

Siri, Alexa, Google Now are some of the popular examples of virtual personal assistants. As the name suggests, they assist in finding information, when asked over voice. All you need to do is activate them and ask "What is my schedule for today?", "What are the flights from Germany to London", or similar questions. For answering, your personal assistant looks out for the information, recalls your related queries, or send a command to other resources (like phone apps) to collect info. You can even instruct assistants for certain tasks like "Set an alarm for 6 AM next morning", "Remind me to visit Visa Office day after tomorrow".

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. Later, this set of data utilized to render results that are tailored to your preferences.

Virtual Assistants are integrated to a variety of platforms. For example:

- Smart Speakers: Amazon Echo and Google Home
- Smartphones: Samsung Bixby on Samsung S8
- Mobile Apps: Google Allo

**II. 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. While this helps in preventing the traffic and does congestion analysis, the underlying problem is that there are a smaller number of cars that are equipped with GPS. Machine learning in such scenarios helps to estimate the regions where congestion can be found on the basis of daily experiences.

Online Transportation Networks: When booking a cab, the app estimates the price of the ride. When sharing these services, how do they minimize the detours? The answer is machine learning. Jeff Schneider, the engineering lead at Uber ATC reveals in an interview that they use ML to define price surge hours by predicting the rider demand. In the entire cycle of the services, ML is playing a major role.

**III. Social Media Services:**

From personalizing your news feed to better ads targeting, social media platforms are utilizing machine learning for their own and user benefits. Here are a few examples that you must be noticing, using, and loving in your social media accounts, without realizing that these wonderful features are nothing but the applications of ML.

•People You May Know: Machine learning works on a simple concept: understanding with experiences. Facebook continuously notices the friends that you connect with, the profiles that you visit very often, your interests, workplace, or a group that you share with someone etc. On the basis of continuous learning, a list of Facebook users is suggested that you can become friends with.

•Face Recognition: You upload a picture of you with a friend and Facebook instantly recognizes that friend. Facebook checks the poses and projections in the picture, notice the unique features, and then match them with the people in your friend list. The entire process at the backend is complicated and takes care of the precision factor but seems to be a simple application of ML at the front end.

•Similar Pins: Machine learning is the core element of Computer Vision, which is a technique to extract useful information from images and videos. Pinterest uses computer vision to identify the objects (or pins) in the images and recommend similar pins accordingly.

**IV. Search Engine Result Refining:**

Google and other search engines use machine learning to improve the search results for you. Every time you execute a search, the algorithms at the backend keep a watch at how you respond to the results. If you open the top results and stay on the web page for long, the search engine assumes that the the results it displayed were in accordance to the query. Similarly, if you reach the second or third page of the search results but do not open any of the results, the search engine estimates that the results served did not match requirement. This way, the algorithms working at the backend improve the search results.

**V. Product Recommendations:**

You shopped for a product online few days back and then you keep receiving emails for shopping suggestions. If not this, then you might have noticed that the shopping website or the app recommends you some items that somehow matches with your taste. On the basis of your behavior with the website/app, past purchases, items liked or added to cart, brand preferences etc., the product recommendations are made.

**VI. Online Fraud Detection:**

Machine learning is proving its potential to make cyberspace a secure place and tracking monetary frauds online is one of its examples. For example: Paypal is using ML for protection against money laundering. The company uses a set of tools that helps them to compare millions of transactions taking place and distinguish between legitimate or illegitimate transactions taking place between the buyers and sellers.


*Fig 1.3 Uses of Machine learning*

## 1.4 TYPES OF LEARNING ALGORITHMS:

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

### 1.4.1 Supervised Learning:

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning. Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to “learn” how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data. Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign. Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

### 1.4.2 Unsupervised Learning:

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

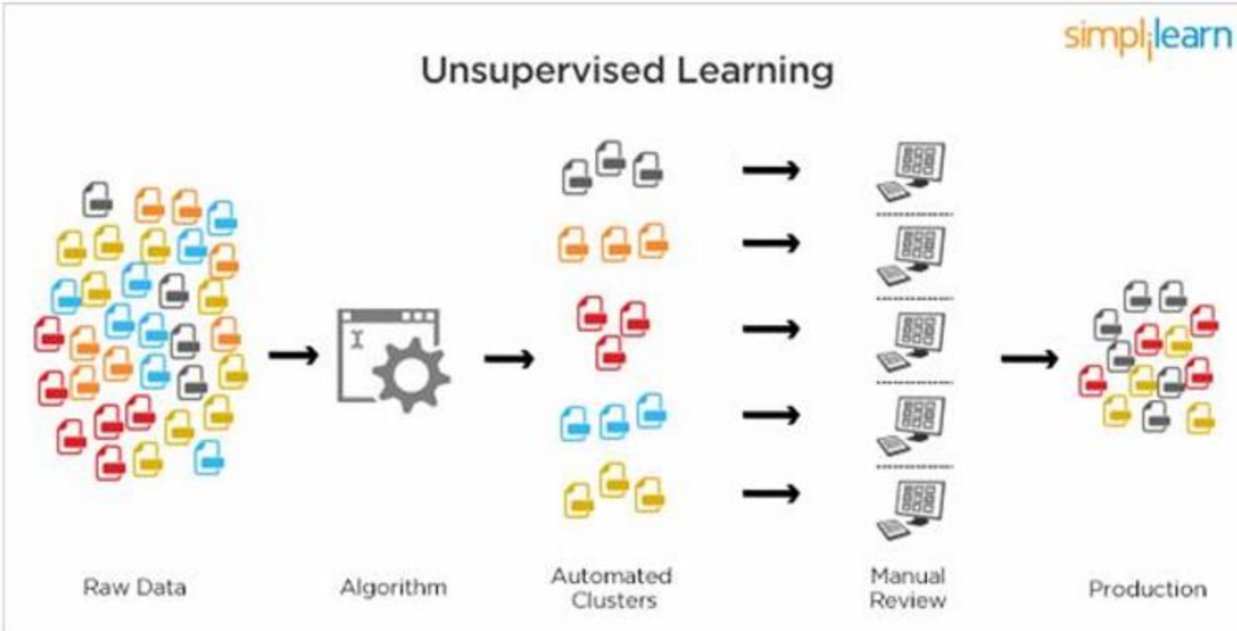



*Figure 2: Unsupervised Learning.*

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

### 1.4.3 Semi Supervised Learning:

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

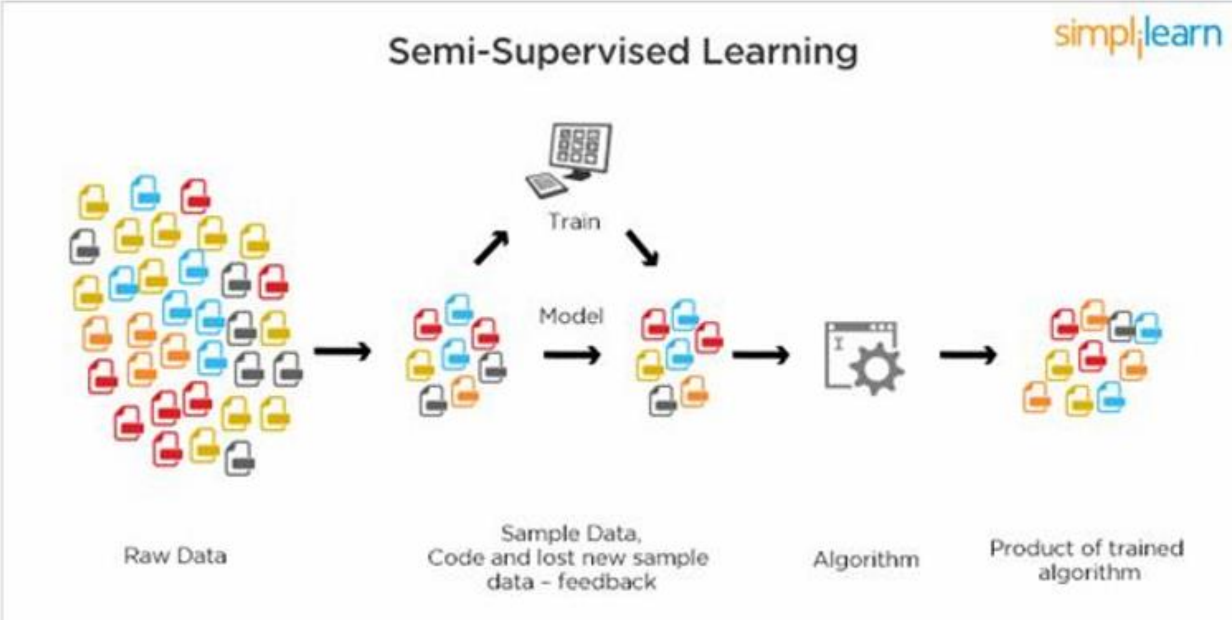



*Figure 3: Semi Supervised Learning*

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

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

# 2. PYTHON

## 2.1 Introduction:

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

## 2.2 Setup of Python:

- Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

### 2.2.1 Installation (using python IDLE):

- To start, go to python.org/downloads and then click on the button to download the latest version of Python
- We can download python IDLE in windows, mac and Linux operating systems also.


*Figure 3.2.1 : Python download*

- Run the .exe file that you just downloaded and start the installation of Python by clicking on **Install Now**
- **We can give environmental variable i.e path after completion of downloading**

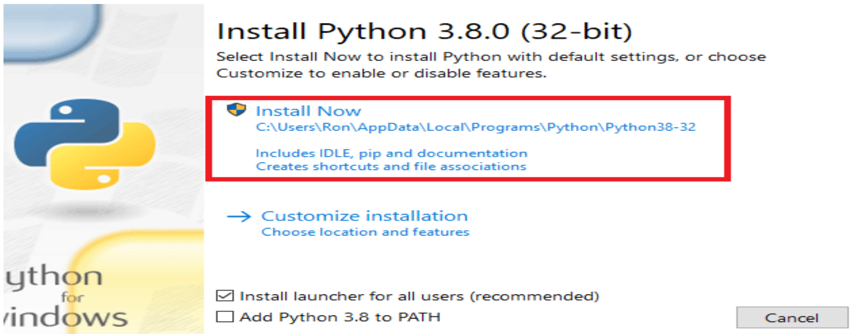



*Fig 3.2.1.1 python installation*

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

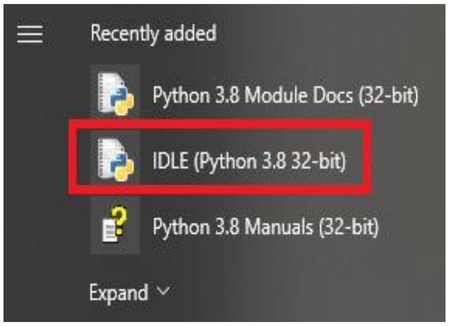


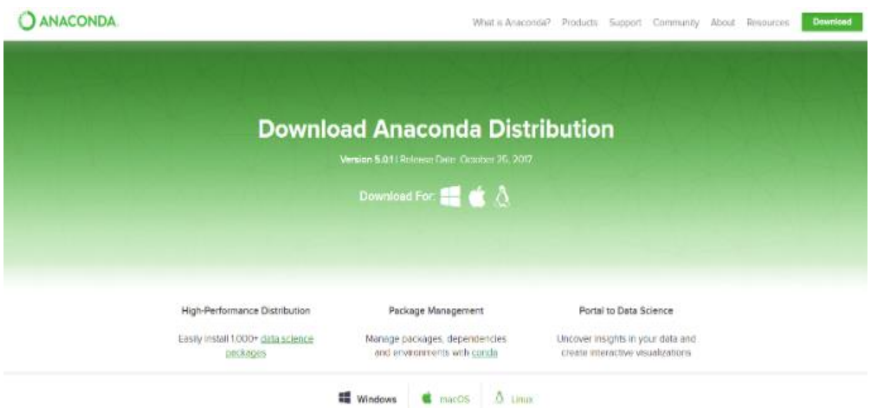

*Fig 3.2.1.2 IDLE*

## 2.2.2 Python Installation 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.

  Anaconda for Windows installation:

i. Go to the following link: Anaconda.com/downloads




ii. Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)

iii. Select path (i.e. add anaconda to path & register anaconda as default python 3.4)

iv. Click finish

v. Open Jupyter notebook

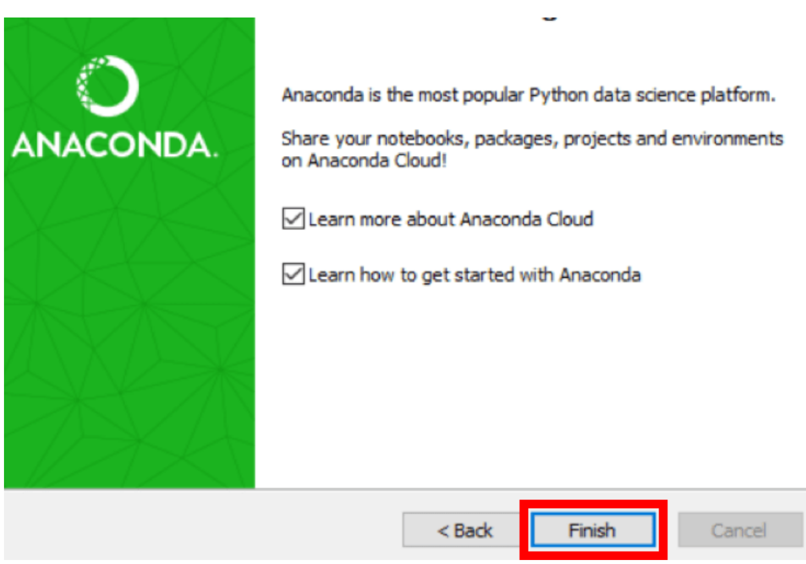



*Fig 3.2.2.1 After installation*

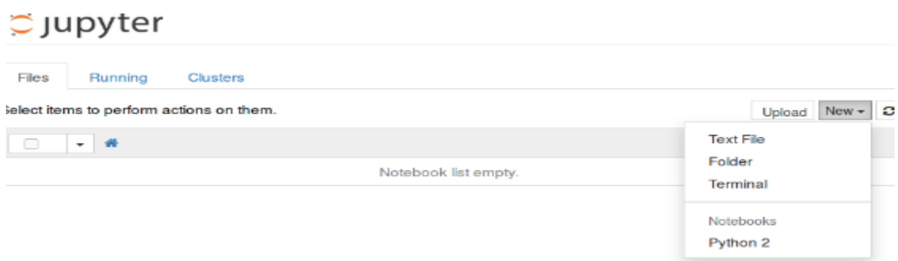



*Fig 3.2.2.2 jupyter notebook*

## 2.3 Features:

i. **Readable:** Python is a very readable language.

ii. **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

iii. **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.

iv. **Open Source:** Python is a open source programming language.

v. **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.

vi. **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.

vii. **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.

viii. **Advanced features:** Supports generators and list comprehensions. We will cover these features later.

ix. **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.

## 2.4 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. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Python has five standard data types –

- Numbers
- Strings
- Lists
- Tuples
- Dictionary

## 2.4.1 Python Numbers:

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Python supports four different numerical types −

- **int (signed integers)** − They are often called just integers or ints, are positive or negative whole numbers with no decimal point.
- **long (long integers )** − Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
- **float (floating point real values)** − Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x $10^2$ = 250).

## 2.4.2 Python Strings:

In Python, Strings can be created by simply enclosing characters in quotes. Python does not support character types. These are treated as length-one strings, and are also considered as substrings. Substrings are immutable and can’t be changed once created.Strings are the ordered blocks of text that are enclosed in single or double quotations. Thus, whatever is written in quotes, is considered as string. Though it can be written in single or double quotations, double quotation marks allow the user to extend strings over multiple lines without backslashes, which is usually the signal of continuation of an expression, e.g., ‘abc’, “ABC”.

### 2.4.3 Python lists:

- List is a collection data type in python. It is ordered and allows duplicate entries as well. Lists in python need not be homogeneous, which means it can contain different data types like integers, strings and other collection data types. It is mutable in nature and allows indexing to access the members in a list.
- To declare a list, we use the square brackets.
- List is like any other array that we declare in other programming languages. Lists in python are often used to implement stacks and queues. The lists are mutable in nature. Therefore, the values can be changed even after a list is declared.

### 2.4.4 python tuples:

- A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also

### 2.4.5 python Dictionary:

- It is a collection data type just like a list or a set, but there are certain features that make python dictionary unique. A dictionary in python is not ordered and is changeable as well. We can make changes in a dictionary unlike sets or strings which are immutable in nature. Dictionary contains key-value pairs like a map that we have in other programming languages. A dictionary has indexes. Since the value of the keys we declare in a dictionary are always unique, we can use them as indexes to access the elements in a dictionary.

## 2.5 Functions:

### 2.5.1 Defining a Function:

- Function blocks begin with the keyword def followed by the function name and parentheses (()).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

### 2.5.2 Calling a Function:

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

## 2.6 OOPs Concepts:

### 2.6.1 Class:

- Python is an object-oriented programming language. Unlike procedure-oriented programming, where the main emphasis is on functions, object-oriented programming stresses on objects.
- An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.
- We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.
- As many houses can be made from a house's blueprint, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called **instantiation**.
- Like function definitions begin with the def keyword in Python, class definitions begin with a class keyword.

- The first string inside the class is called docstring and has a brief description about the class

```
class MyNewClass:
    '''This is a docstring. I have created a new class'''
    pass
```

*Fig 3.6.1 Class defining*

- As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

```
class Person:
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')


# Output: 10
print(Person.age)

# Output: <function Person.greet>
print(Person.greet)

# Output: 'This is my second class'
print(Person.__doc__)
```

*Fig 3.6.1.1 Example of class*

# 3. CASE STUDY

## 3.1 PROBLEM STATEMENT:

Bob has started his own mobile company. He wants to give tough fight to big companies like Apple, Samsung etc.

He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market, you cannot simply assume things. To solve this problem, he collects sales data of mobile phones of various companies.

Bob wants to find out some relation between features of a mobile phone (e.g.: - RAM, Internal Memory etc.) and its selling price. But he is not so good at Machine Learning. So, he needs your help to solve this problem.

In this problem you do not have to predict actual price but a price range indicating how high the price is

Code Text

## 3.2 DATA SET:

The dataset is taken from the Kaggle. The dataset contains several attributes of mobile phone features https://www.kaggle.com/iabhishekofficial/mobile-price-classification

In this data the attributes given are and there explanations:

- id: ID
- battery power: Total energy a battery can store in one time measured in mAh
- blue: Has Bluetooth or not
- clock speed: speed at which microprocessor executes instructions
- dual sim: 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:longest time that a single battery charge will - last when you are
- 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:

Objective of problem is to predict the price range indicating how high the price is.

## 3.4. Project Requirements:

### 3.4.1 Packages used:

- **NumPy:**

  NumPy is a python library used for working with arrays. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster that traditional Python lists. The array object in NumPy is called `ndata`, it provides a lot of supporting functions that make working with `ndarray` very easy. Arrays are very frequently used in data science, where speed and resources are very important.

- **Pandas:**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

- **Seaborn:**

  Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

- **Matplotlib:**

  Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPythonotTkinter. It can be used in Python and IPython shells, Jupyter notebook and web application servers also. Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks. Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.

# 4. MODEL BUILDING

## 4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

### 4.1.1 GETTING THE DATASET:

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

### 4.1.2 IMPORTING THE LIBRARIES:

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

**1. LOAD required packages**

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

*Fig 4.1 loading packages*

### 4.1.3 Versions of the packages:

The versions of the packages are found by following command

Know the versions of packages

```
[ ] import numpy
    import matplotlib
    print('numpy:',numpy.__version__)
    print('pandas:',pd.__version__)
    print('seaborn:',sns.__version__)
    print('matplotlib:',matplotlib.__version__)
```

```
numpy: 1.18.5
pandas: 1.0.5
seaborn: 0.10.1
matplotlib: 3.2.2
```

*Fig 4.1.3 versions of packages*

## 4.1.4 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method read_csv (). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a Data Frame 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.

```
[ ] df_train=pd.read_csv('https://raw.githubusercontent.com/venkanna831/AIML/master/Project/train.csv')
```

```
[ ] df_train.head()
```

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_cores | pc | px_height | px_width | ram | sc_h | sc_w | talk_time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 842 | 0 | 2.2 | 0 | 1 | 0 | 7 | 0.6 | 188 | 2 | 2 | 20 | 756 | 2549 | 9 | 7 | 19 |
| 1 | 1021 | 1 | 0.5 | 1 | 0 | 1 | 53 | 0.7 | 136 | 3 | 6 | 905 | 1988 | 2631 | 17 | 3 | 7 |
| 2 | 563 | 1 | 0.5 | 1 | 2 | 1 | 41 | 0.9 | 145 | 5 | 6 | 1263 | 1716 | 2603 | 11 | 2 | 9 |
| 3 | 615 | 1 | 2.5 | 0 | 0 | 0 | 10 | 0.8 | 131 | 6 | 9 | 1216 | 1786 | 2769 | 16 | 8 | 11 |
| 4 | 1821 | 1 | 1.2 | 0 | 13 | 1 | 44 | 0.6 | 141 | 2 | 14 | 1208 | 1212 | 1411 | 8 | 2 | 15 |

*Fig 4.1.4 loading dataset*

# 5.DATA PREPROCESSING/FEATURE ENGINEERING AND EDA

## 5.1 Statistical Analysis:

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 Data Frame 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.

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding nan values. Analyzes both numeric and object series, as well as Data Frame column sets of mixed data types. The output will vary depending on what is provided.

For numeric data, the result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles. By default, the lower percentile is 25 and the upper percentile is 75. The 50 percentile is the same as the median.

For object data (e.g. strings or timestamps), the result's index will include count, unique, top, and freq. The top is the most common value. The freq is the most common value's frequency. Timestamps also include the first and last items.

If multiple object values have the highest count, then the count and top results will be arbitrarily chosen from among those with the highest count.

For mixed data types provided via a Data Frame, the default is to return only an analysis of numeric columns. If the data frame consists only of object and categorical data without any numeric columns, the default is to return an analysis of both the object and categorical columns. If include='all' is provided as an option, the result will include a union of attributes of each type.

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| battery_power | 2000.0 | 1238.51850 | 439.418206 | 501.0 | 851.75 | 1226.0 | 1615.25 | 1998.0 |
| blue | 2000.0 | 0.49500 | 0.500100 | 0.0 | 0.00 | 0.0 | 1.00 | 1.0 |
| clock_speed | 2000.0 | 1.52225 | 0.816004 | 0.5 | 0.70 | 1.5 | 2.20 | 3.0 |
| dual_sim | 2000.0 | 0.50950 | 0.500035 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| fc | 2000.0 | 4.30950 | 4.341444 | 0.0 | 1.00 | 3.0 | 7.00 | 19.0 |
| four_g | 2000.0 | 0.52150 | 0.499662 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| int_memory | 2000.0 | 32.04650 | 18.145715 | 2.0 | 16.00 | 32.0 | 48.00 | 64.0 |
| m_dep | 2000.0 | 0.50175 | 0.288416 | 0.1 | 0.20 | 0.5 | 0.80 | 1.0 |
| mobile_wt | 2000.0 | 140.24900 | 35.399655 | 80.0 | 109.00 | 141.0 | 170.00 | 200.0 |
| n_cores | 2000.0 | 4.52050 | 2.287837 | 1.0 | 3.00 | 4.0 | 7.00 | 8.0 |
| pc | 2000.0 | 9.91650 | 6.064315 | 0.0 | 5.00 | 10.0 | 15.00 | 20.0 |
| px_height | 2000.0 | 645.10800 | 443.780811 | 0.0 | 282.75 | 564.0 | 947.25 | 1960.0 |
| px_width | 2000.0 | 1251.51550 | 432.199447 | 500.0 | 874.75 | 1247.0 | 1633.00 | 1998.0 |
| ram | 2000.0 | 2124.21300 | 1084.732044 | 256.0 | 1207.50 | 2146.5 | 3064.50 | 3998.0 |
| sc_h | 2000.0 | 12.30650 | 4.213245 | 5.0 | 9.00 | 12.0 | 16.00 | 19.0 |
| sc_w | 2000.0 | 5.76700 | 4.356398 | 0.0 | 2.00 | 5.0 | 9.00 | 18.0 |
| talk_time | 2000.0 | 11.01100 | 5.463955 | 2.0 | 6.00 | 11.0 | 16.00 | 20.0 |
| three_g | 2000.0 | 0.76150 | 0.426273 | 0.0 | 1.00 | 1.0 | 1.00 | 1.0 |
| touch_screen | 2000.0 | 0.50300 | 0.500116 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| wifi | 2000.0 | 0.50700 | 0.500076 | 0.0 | 0.00 | 1.0 | 1.00 | 1.0 |
| price_range | 2000.0 | 1.50000 | 1.118314 | 0.0 | 0.75 | 1.5 | 2.25 | 3.0 |

*Fig 5.1.1 Statistical data*

**Observations: -**

1.2000mah is the max Battery Power

2.Dual sim is available for 75% of the phones

3.50% of the mobile phones has 32gb of memory

4.most of the mobile phones are screen touch enabled and supports 3g, 4g and are wifi enabled

## 5.2 Data Type Conversions:

When doing data analysis, it is important to make sure you are using the correct data types; otherwise you may get unexpected results or errors. In the case of pandas, it will correctly infer data types in many cases and you can move on with your analysis without any further thought on the topic. Despite how well pandas works, at some point in your data analysis processes, you will likely need to explicitly convert data from one type to another. This article will discuss the basic panda's data types (aka dtypes), how they map to python and numpy data types and the options for converting from one pandas type to another.

```
df_train.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
```

*Fig 5.2 datatypes*

## 5.3 Handling Missing Values:

There are a number of schemes that have been developed to indicate the presence of missing data in a table or Data Frame. Generally, they revolve around one of two strategies: using a mask that globally indicates missing values, or choosing a sentinel value that indicates a missing entry. In the masking approach, the mask might be an entirely separate Boolean array, or it may involve appropriation of one bit in the data representation to locally indicate the null status of a value. In the sentinel approach, the sentinel value could be some data-specific convention, such as indicating a missing integer value with -9999 or some rare bit pattern, or it could be a more global convention, such as indicating a missing floating-point value with NaN (Not a Number), a special value which is part of the IEEE floating-point specification.

```
df_train.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
```

## 5.4 Encoding Categorical Data:

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.

In the given dataset I have not used any encoding because dataset is numerical

Categorical column in the data set

```
1 df_train.select_dtypes(exclude=['int64', 'float64']).columns
```

```
2]: Index([], dtype='object')
```

Numerical columns in the datasets

```
1 df_train.select_dtypes(include=['int64', 'float64']).columns
```

```
3]: Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
           'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
           'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
           'touch_screen', 'wifi', 'price_range'],
          dtype='object')
```

*Fig 5.5 Numerical Dataset*

## 5.6 Generating Plots:

### 5.6.1 Visualize the data between Target and the Features:

**i. Correlation:**

Correlation

```
df_train.corr() #correlation
```

5]:

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_cores | pc | px_height |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| battery_power | 1.000000 | 0.011252 | 0.011482 | -0.041847 | 0.033334 | 0.015665 | -0.004004 | 0.034085 | 0.001844 | -0.029727 | 0.031441 | 0.014901 |
| blue | 0.011252 | 1.000000 | 0.021419 | 0.035198 | 0.003593 | 0.013443 | 0.041177 | 0.004049 | -0.008605 | 0.036161 | -0.009952 | -0.006872 |
| clock_speed | 0.011482 | 0.021419 | 1.000000 | -0.001315 | -0.000434 | -0.043073 | 0.006545 | -0.014364 | 0.012350 | -0.005724 | -0.005245 | -0.014523 |
| dual_sim | -0.041847 | 0.035198 | -0.001315 | 1.000000 | -0.029123 | 0.003187 | -0.015679 | -0.022142 | -0.008979 | -0.024658 | -0.017143 | -0.020875 |
| fc | 0.033334 | 0.003593 | -0.000434 | -0.029123 | 1.000000 | -0.016560 | -0.029133 | -0.001791 | 0.023618 | -0.013356 | 0.644595 | -0.009990 |
| four_g | 0.015665 | 0.013443 | -0.043073 | 0.003187 | -0.016560 | 1.000000 | 0.008690 | -0.001823 | -0.016537 | -0.029706 | -0.005598 | -0.019236 |
| int_memory | -0.004004 | 0.041177 | 0.006545 | -0.015679 | -0.029133 | 0.008690 | 1.000000 | 0.006886 | -0.034214 | -0.028310 | -0.033273 | 0.010441 |
| m_dep | 0.034085 | 0.004049 | -0.014364 | -0.022142 | -0.001791 | -0.001823 | 0.006886 | 1.000000 | 0.021756 | -0.003504 | 0.026282 | 0.025263 |
| mobile_wt | 0.001844 | -0.008605 | 0.012350 | -0.008979 | 0.023618 | -0.016537 | -0.034214 | 0.021756 | 1.000000 | -0.018989 | 0.018844 | 0.000939 |
| n_cores | -0.029727 | 0.036161 | -0.005724 | -0.024658 | -0.013356 | -0.029706 | -0.028310 | -0.003504 | -0.018989 | 1.000000 | -0.001193 | -0.006872 |
| pc | 0.031441 | -0.009952 | -0.005245 | -0.017143 | 0.644595 | -0.005598 | -0.033273 | 0.026282 | 0.018844 | -0.001193 | 1.000000 | -0.018465 |
| px_height | 0.014901 | -0.006872 | -0.014523 | -0.020875 | -0.009990 | -0.019236 | 0.010441 | 0.025263 | 0.000939 | -0.006872 | -0.018465 | 1.000000 |

*Fig 5.6.1 correlation*

```
0] fig = plt.subplots (figsize = (6, 6))
   sns.heatmap(df_train.corr (), square = True, cbar = True, annot = True, annot_kws = {'size': 8})
   plt.title('Correlations between Attributes')
   plt.show ()
```

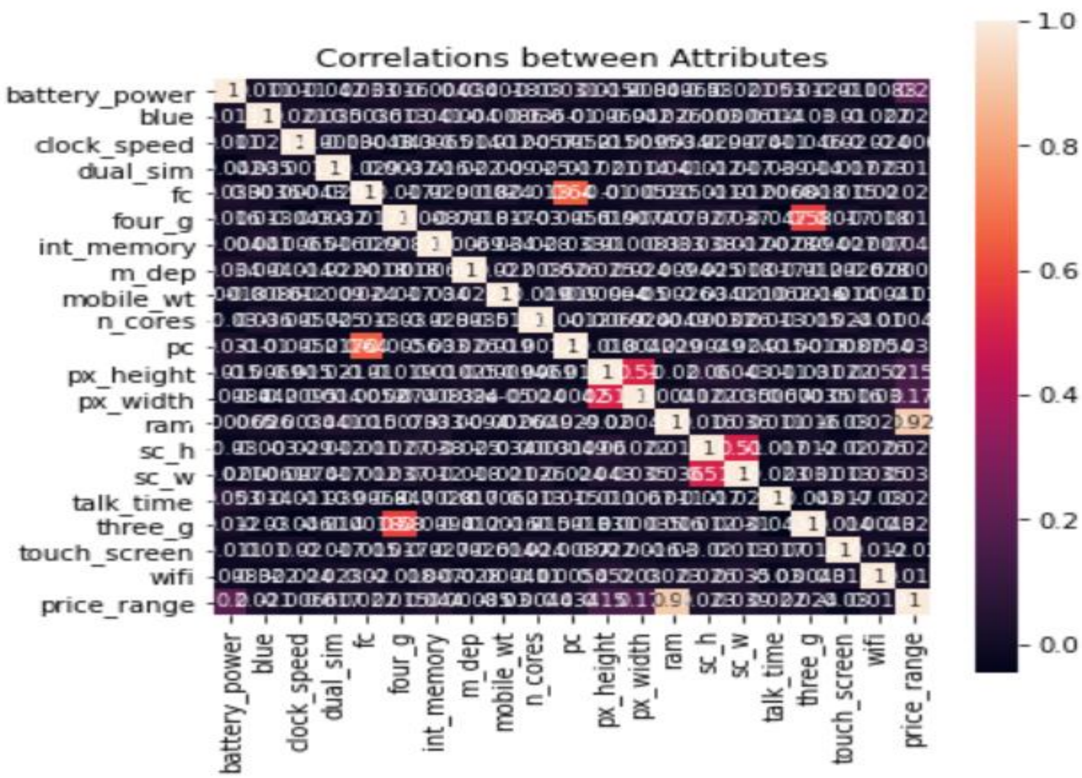


*Fig 5.6.1.1 correlation graph*

**Observations of heatmap:**

- the most influential variable is ram
- having 3G and 4G is somewhat correlated
- most of the variables have very little correlation to price range
- primary camera mega pixels and front Camera mega pixels have correlation (it makes sense because both of them reflect technology level of resolution of the related phone model) but they do not affect price range.
- there are no highly correlated inputs in our dataset, so there is no multicollinearity problem.

**ii.How does ram is affected by price:**

Effect of ram on price

```
[ ] df_train.groupby('price_range').mean()['ram'].plot(kind = 'bar', legend = True).tick_params(axis = 'x', labelrotation = 360)
```

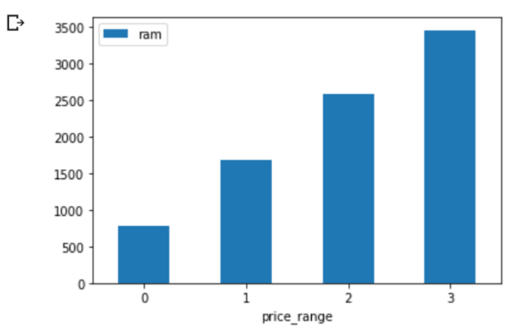



*Fig 5.6.1.2 ram vs price*

Here, we can see:

- The price range 3 having more RAM
- The price range 0 having low RAM

**iii.Bluetooth, Wifi vs price:**

**Bluetooth,Wifi vs price**

```
[ ] sns.catplot('price_range', col='blue',hue = 'wifi',data = df_train,  kind = 'count', col_wrap=2)
```

```
<seaborn.axisgrid.FacetGrid at 0x7faad8ec43c8>
```

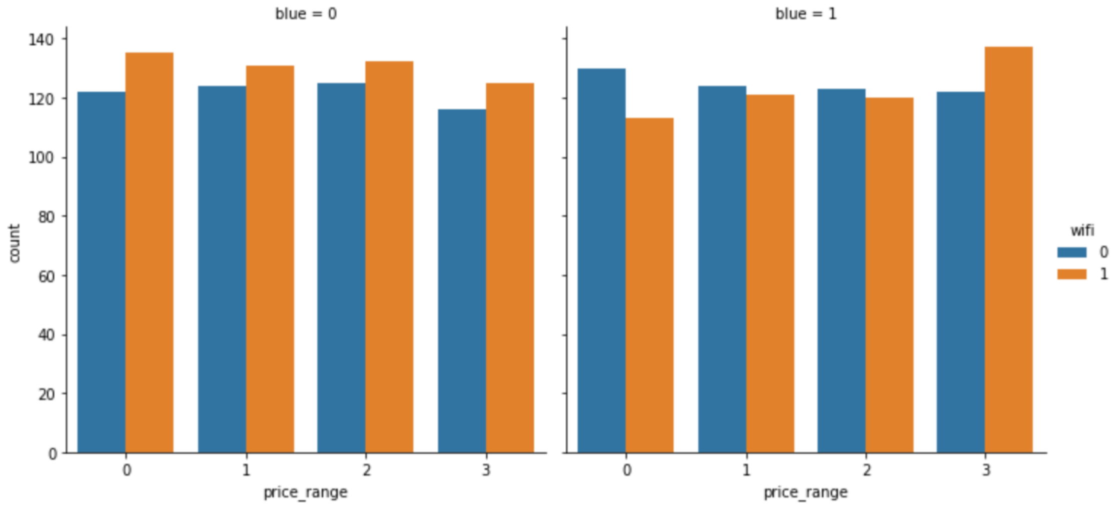



*Fig 5.6.1.3 Bluetooth, WIFI*

**Observation-**

Bluetooth and Wifi seem to not have a significant affect to phone price since they have similar distribution in every price range.

### Price range vs Internal Memory

```
[ ] sns.lineplot(y="int_memory", x="price_range", data=df_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faada7d49e8>
```

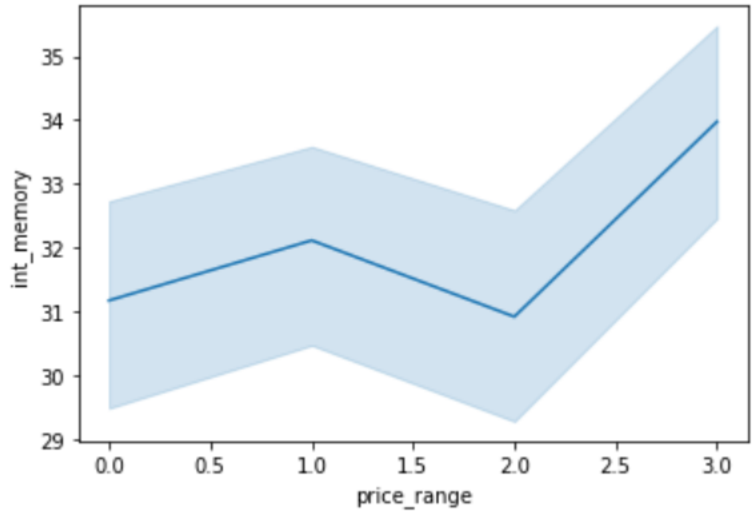



*Fig 5.6.1.4 range vs memory*

## 5.6.2. Visualize the data between all the Features:

**i.% of Phones which support 3G:**

```
[ ]  labels = ["3G-supported",'Not supported']
     values=df_train['three_g'].value_counts().values
     fig1, ax1 = plt.subplots()
     ax1.pie(values, labels=labels, autopct='%1.1f%%',shadow=True,startangle=90)
     plt.show()
```

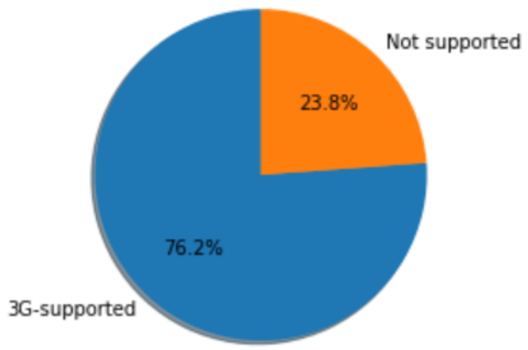



*Fig 5.6.2.1 3g support phones*

**Observations:**

- 76.2% of phones support 3g
- 23.8% of phones do not support 3g

**ii.Supports Dualism or not:**

```
x=df_train['dual_sim'].value_counts()
labels='Supports Dualsim: '+str(x[1]),'Does not support Dualsim:- '+str(x[0])
sizes=[x[1],x[0]]
fig1, ax1 = plt.subplots()
ax1.pie(sizes,labels=labels)
ax1.axis('equal')
plt.show()
```

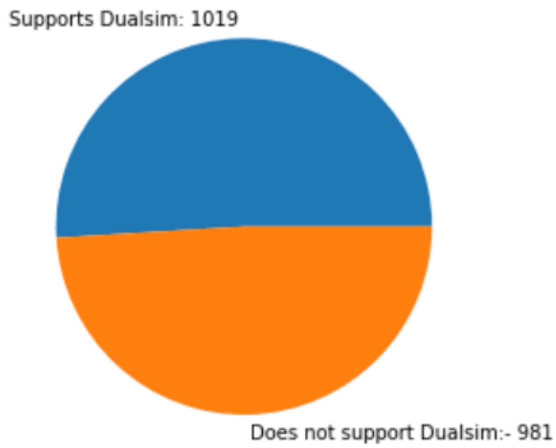



*Fig 5.6.2.2 dual sim*

Observation

- 1019 phones support dualsim
- 981 phones donot support dualsim

**% of Phones which support 4G**

```
[ ] labels4g = ["4G-supported",'Not supported']
    values4g = df_train['four_g'].value_counts().values
    fig1, ax1 = plt.subplots()
    ax1.pie(values4g, labels=labels4g, autopct='%1.1f%%',shadow=True,startangle=90)
    plt.show()
```

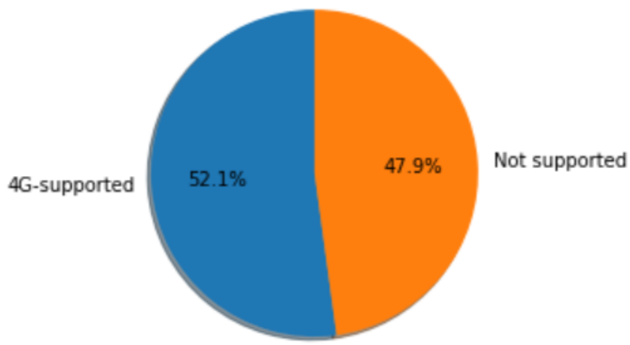



*Fig 5.6.2.3 4g support phones*

Observation

- 52.1% of phones support 4g
- 47.9% of phones donot support 4g

### Battery power vs Price Range

```
[ ] sns.pointplot(x="price_range", y="battery_power", data=df_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faadacc6b70>
```

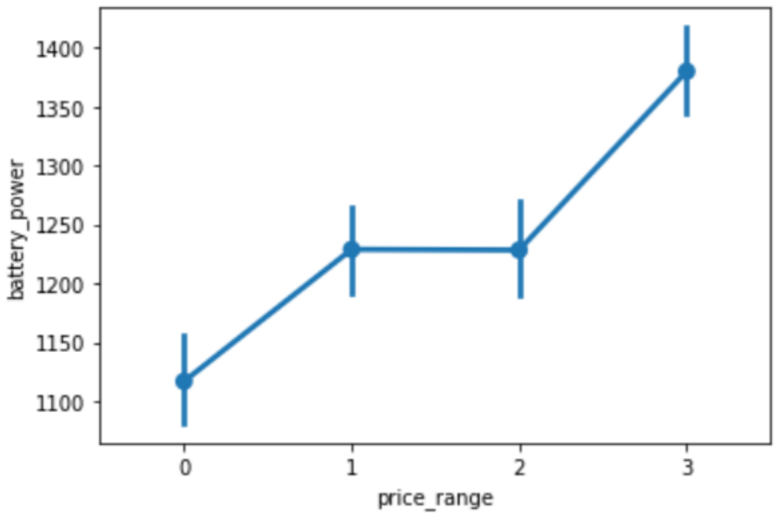



*Fig 5.6.2.4 battery power*

**Observations:-**

- when price_range is 3 then battery power greater than 1600 MAH
- when price_range is 2 then battery power greater is 1600 MAH
- when price_range is 1 then battery power is 1500 MAH
- when price_range is 0 then battery power is 1400 MAH

### No of Phones vs Camera megapixels of front and primary camera :

```
[ ] plt.figure(figsize=(10,6))
    df_train['fc'].hist(alpha=0.5,color='black',label='Front camera')
    df_train['pc'].hist(alpha=0.5,color='orange',label='Primary camera')
    plt.legend()
    plt.xlabel('MegaPixels')
```

```
Text(0.5, 0, 'MegaPixels')
```

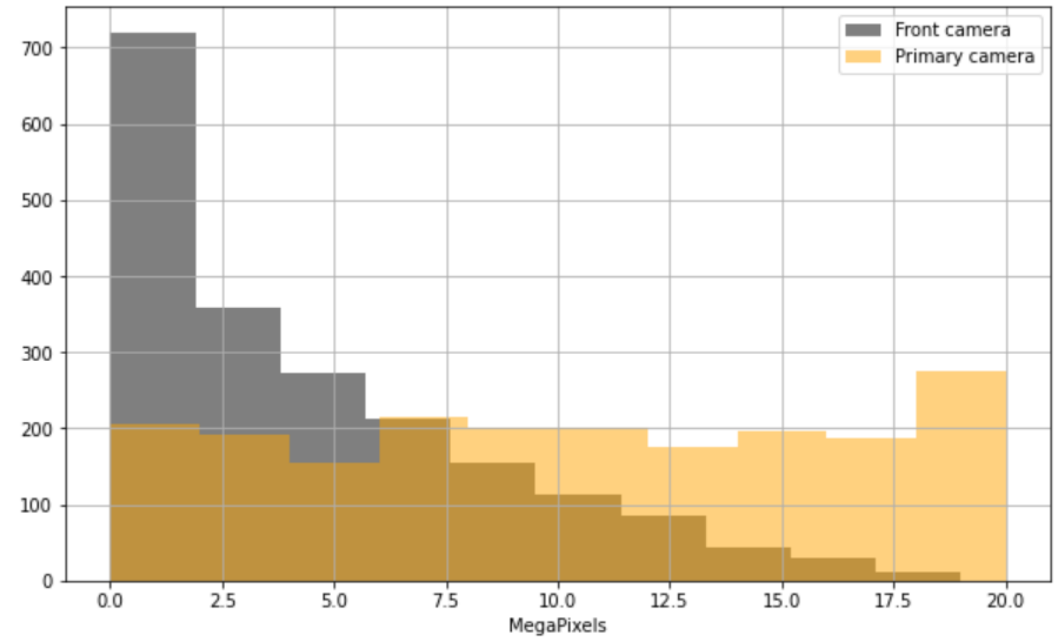



*Fig 5.6.2.4 camera*

**EDA observation:**

ram has direct impact on the price range of the phones

Features like

1.3g

2.4g

3.Dual sim

4.wifi

5.Touch screen

these have more impact on the phone’s prices

# 6. FEATURE SELECTION

## 6.1 Select relevant features for the analysis:

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

· **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.

· **Improves Accuracy**: Less misleading data means modeling accuracy improves.

· **Reduces Training Time**: fewer data points reduce algorithm complexity and algorithms train faster.

**Feature Selection Methods:**

I will share 3 Feature selection techniques that are easy to use and also gives good results.

1. Univariate Selection

2. Feature Importance

3.Correlation Matrix with Heatmap

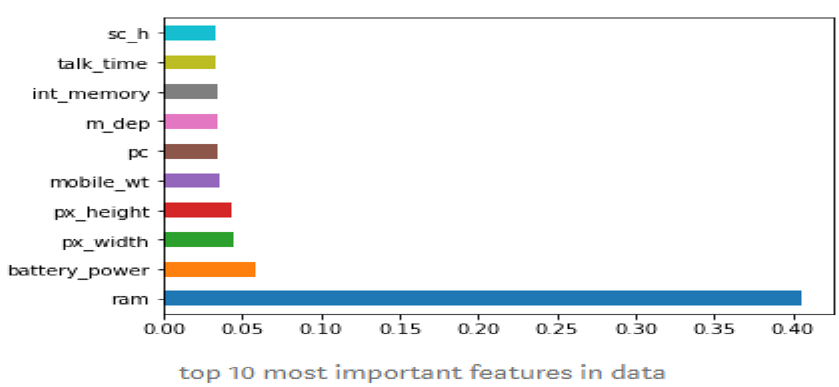



*Fig 6.1 top features in dataset*

## 6.2. Train and Test

One of the first decisions to make when starting a modeling project is how to utilize the existing data. One common technique is to split the data into two groups typically referred to as the training and testing sets. The training set is used to develop models and feature sets; they are the substrate for estimating parameters, comparing models, and all of the other activities required to reach a final model. The test set is used only at the conclusion of these activities for estimating a final, unbiased assessment of the model's performance. It is critical that the test set not be used prior to this point. Looking at the test sets results would bias the outcomes since the testing data will have become part of the model development process.

```
[ ]  y = df_train['price_range']
     X = df_train.drop('price_range', axis = 1)
```

Price range target values

```
[ ]  y.unique()
```

```
array([1, 2, 3, 0])
```

*Fig 6.2.1 dividing input and output and target values*

**Importing packages:**

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2, random_state = 101)
```

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

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

*Fig 6.2.2 Importing packages*

## 6.3 Feature Scaling:

It is a step of Data Pre-Processing which is applied to independent variables or features of data. It basically helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm. Real world dataset contains features that highly vary in magnitudes, units, and range. Normalization should be performed when the scale of a feature is irrelevant or misleading and not should Normalize when the scale is meaningful.

The algorithms which use Euclidean Distance measure are sensitive to Magnitudes. Here feature scaling helps to weigh all the features equally.

Formally, If a feature in the dataset is big in scale compared to others then in algorithms where Euclidean distance is measured this big scaled feature becomes dominating and needs to be normalized.

$$z = \frac{x - \mu}{\sigma}$$

.

*Fig 6.3.1 formula for scaling*

**Importing package:**

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
scaler = StandardScaler().fit(X_train)
```

```
scaled_X_train = pd.DataFrame(scaler.fit_transform(X_train))
scaled_X_train
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.984823 | -0.986343 | 0.345546 | 0.990050 | -0.066906 | -1.053953 | -1.156036 | 0.653456 | -0.451426 | 0.646912 | -0.312840 | -0.841894 | 0.512558 | 1.50796 |
| 1 | -0.673224 | 1.013846 | -1.250227 | -1.010051 | -0.766578 | 0.948808 | 1.044235 | -1.416900 | 0.114337 | -1.102237 | 0.849672 | -1.018037 | -0.852864 | 0.72493 |
| 2 | -0.400668 | 1.013846 | 0.222795 | 0.990050 | -0.999802 | -1.053953 | 1.429282 | -0.036663 | -0.140256 | 1.084199 | -1.641426 | -1.275477 | -1.329604 | -1.29578 |
| 3 | 1.545834 | -0.986343 | -1.250227 | -1.010051 | -0.300130 | 0.948808 | 1.704316 | -0.726781 | -0.819173 | -1.102237 | 0.019306 | -1.178372 | -1.318033 | 1.18963 |
| 4 | -1.359156 | -0.986343 | -1.250227 | -1.010051 | -0.766578 | -1.053953 | 0.494167 | 1.688634 | 1.613611 | 1.084199 | 0.019306 | -0.715433 | -1.563346 | 0.76792 |

```
scaled_X_test = pd.DataFrame(scaler.fit_transform(X_test))
scaled_X_test
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.342699 | -1.005013 | -1.264622 | -1.056599 | 1.232482 | 0.995012 | 0.546668 | -0.952194 | -1.418952 | -1.537778 | 1.599113 | 0.574258 | -0.719246 | -0.515779 |
| **1** | -1.402675 | -1.005013 | -1.264622 | -1.056599 | -0.308120 | -1.005013 | -0.342901 | -0.594562 | -1.362658 | -0.227172 | -0.971679 | 0.643609 | 1.650558 | -0.252633 |
| **2** | -1.455565 | -1.005013 | -0.167336 | 0.946433 | 1.232482 | 0.995012 | -1.343667 | 0.120701 | 0.016536 | 1.520303 | 0.635066 | -1.195289 | -1.476749 | 0.998267 |
| **3** | -0.089620 | 0.995012 | -0.411177 | -1.056599 | 0.132052 | 0.995012 | 0.991453 | 1.908858 | 0.072830 | -0.227172 | 1.599113 | 0.614526 | 0.038257 | -0.635912 |
| **4** | -0.742698 | -1.005013 | -0.655019 | 0.946433 | -0.308120 | 0.995012 | -1.566059 | -0.594562 | -0.856015 | -1.537778 | -0.971679 | 1.630171 | 0.540943 | -1.021097 |

1. **K-Means** uses the Euclidean distance measure here feature scaling matters.
2. **K-Nearest-Neighbors** also require feature scaling.
3. **Principal Component Analysis (PCA)**: Tries to get the feature with maximum variance, here too feature scaling is required.
4. **Gradient Descent**: Calculation speed increase as Theta calculation becomes faster after feature scaling

# 7.MODEL BUILDING AND EVALUATION

## 7.1 Brief about the algorithms used:

### i. Logistic Regression:

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

Before diving into the implementation of logistic regression, we must be aware of the following assumptions about the same −

- In case of binary logistic regression, the target variables must be binary always and the desired outcome is represented by the factor level 1.
- There should not be any multi-collinearity in the model, which means the independent variables must be independent of each other.
- We must include meaningful variables in our model.
- We should choose a large sample size for logistic regression.

```
from sklearn.linear_model import LogisticRegression
```

```
lr1 = LogisticRegression(multi_class = 'ovr', solver = 'sag',  max_iter = 10000)
```

```
lr1.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=10000,
                   multi_class='ovr', n_jobs=None, penalty='l2',
                   random_state=None, solver='sag', tol=0.0001, verbose=0,
                   warm_start=False)
```

*Fig 7.1.1 logistic regression of trained data*

**ii.Decision Tree:**

A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new nodes.

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.

The decision rules are generally in form of if-then-else statements. The deeper the tree, the more complex the rules and fitter the model.

```
] from sklearn.tree import DecisionTreeClassifier
```

```
] dt = DecisionTreeClassifier(random_state=101,criterion='entropy')
  dt_model = dt.fit(X_train, y_train)
```

```
] y_train_pred_dt=dt.predict(X_train)
```

```
] metrics.confusion_matrix(y_train,y_train_pred_dt)
```

```
array([[406,   0,   0,   0],
       [  0, 398,   0,   0],
       [  0,   0, 380,   0],
       [  0,   0,   0, 416]])
```

*Fig 7.1.2 Importing decision tree packages*

**iii. Random forest:**

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks). In this post we'll learn how the random forest algorithm works, how it differs from other algorithms and how to use it.

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

**Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.**

One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems.

```
[ ] from sklearn.ensemble import RandomForestClassifier
    # initialize the object for RFC
    rf = RandomForestClassifier(n_estimators = 100, random_state=101, criterion = 'entropy', oob_score = True)
    model_rf = rf.fit(X_train, y_train)
```

```
[ ] y_pred_train_1113 = rf.predict(X_train)
    from sklearn.metrics import confusion_matrix, classification_report
    print(classification_report(y_train, y_pred_train_1113))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       406
           1       1.00      1.00      1.00       398
           2       1.00      1.00      1.00       380
           3       1.00      1.00      1.00       416

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

*Fig 7.1.3 Random forest*

## 7.2 Train the Models:

**Splitting the data: after the preprocessing is done then the data is split into train and test sets**

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

● training set - a subset to train a model. (Model learns patterns between Input and Output)

● test set - a subset to test the trained model. (To test whether the model has correctly learnt )

● The amount or percentage of Splitting can be taken as specified

● First, we need to identify the input and output variables and we need to separate the input set and output set

● In scikit learn library we have a package called model selection in which train_test_split method is available. we need to import this method

● This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables.

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2, random_state = 101)
```

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

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

*Fig 7.2.1 Training data*

## 7.3 Make Predictions:

- Then we have to test the model for the test set ,that is done as follows
- We have a method called predict, using this method we need to predict the output for input test set and we need to compare the out but with the output test data
- If the predicted values and the original values are close then we can say that model is trained with good accuracy

**Predicting test in logistic regression:**

```
[ ] confusion_matrix = metrics.confusion_matrix(y_test, y_pred_lr_11)
    confusion_matrix
```

```
array([[82, 11,  1,  0],
       [25, 49, 23,  5],
       [ 0, 21, 54, 45],
       [ 0,  4,  8, 72]])
```

*Fig 7.3.1 Predicting test in logistic regression*

**Predicting test data in logistic regression with scaled data:**

```
y_test_pred_lr = lr1.predict(scaled_X_test)
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_test_pred_lr)
confusion_matrix
```

```
array([[93,  1,  0,  0],
       [10, 71, 21,  0],
       [ 0, 22, 82, 16],
       [ 0,  0,  0, 84]])
```

```
print(classification_report(y_test, y_test_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94        94
           1       0.76      0.70      0.72       102
           2       0.80      0.68      0.74       120
           3       0.84      1.00      0.91        84

    accuracy                           0.82       400
   macro avg       0.82      0.84      0.83       400
weighted avg       0.82      0.82      0.82       400
```

*Fig 7.3.2 Predicting test in scaled logistic regression*

**Predicting test data in Decision Tree:**

```
[ ]  print(metrics.confusion_matrix(y_test, y_pred_dt))

     [[ 86   8   0   0]
      [  7  84  11   0]
      [  0  10 100  10]
      [  0   0   6  78]]
```

```
[ ]  print(metrics.classification_report(y_test, y_pred_dt))

                   precision    recall  f1-score   support

                0       0.92      0.91      0.92        94
                1       0.82      0.82      0.82       102
                2       0.85      0.83      0.84       120
                3       0.89      0.93      0.91        84

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

*Fig 7.3.3 Predicting test in decision tree*

**Predicting test data in Random Forest:**

```
[ ] y_pred_rf = rf.predict(X_test)
    print(metrics.confusion_matrix(y_test, y_pred_rf))

[[ 87   7   0   0]
 [  7  85  10   0]
 [  0  14 100   6]
 [  0   0   3  81]]
```

```
[ ] print(classification_report(y_test, y_pred_rf))

              precision    recall  f1-score   support

           0       0.93      0.93      0.93        94
           1       0.80      0.83      0.82       102
           2       0.88      0.83      0.86       120
           3       0.93      0.96      0.95        84

    accuracy                           0.88       400
   macro avg       0.89      0.89      0.89       400
weighted avg       0.88      0.88      0.88       400
```

*Fig 7.3.4 Predicting test in random forest*

## 7.4 Validate the Models:

Model validation is the process of evaluating a trained model on test data set. This provides the generalization ability of a trained model. Here I provide a step by step approach to complete first iteration of model validation in minutes.

- The model is validated after completion of training and testing the model.
- Checking the accuracy scores as metrics to validate the models
- We have to check the accuracy among the models and validate best model among those.

**Test and Train accuracy in logistic regression with scaled data:**

```
[ ] acc_lr = metrics.accuracy_score(y_train, y_train_pred_lr) #train
    acc_lr
```

```
0.86875
```

```
[ ] acc_lr = metrics.accuracy_score(y_test, y_test_pred_lr) #test
    acc_lr
```

```
0.825
```

*Fig 7.4.1 Test and train accuracy in logistic regression*

**Train vs Test in scaled logistic regression:**

```
] models = ['training','testing']
  acc_scores = [0.69,0.64]
  plt.barh(models, acc_scores, color=['lightgreen', 'grey' ])
  plt.ylabel("accuracy scores")
  plt.title("train vs test")
  plt.show()
```

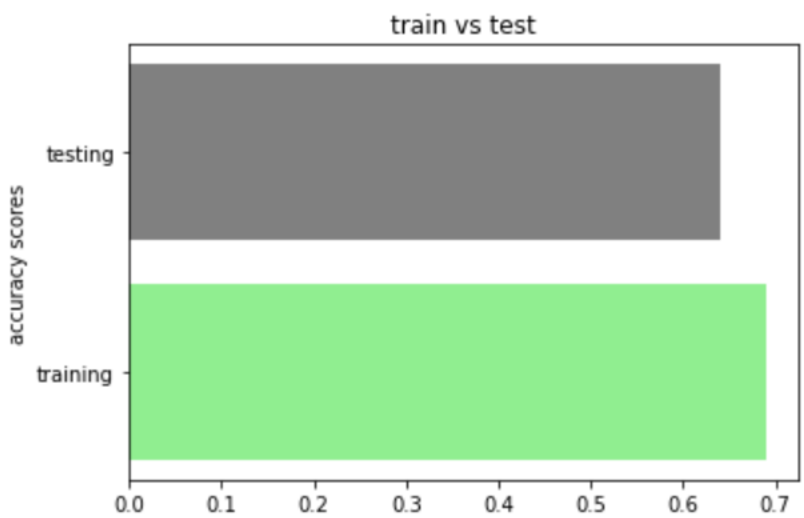



*Fig 7.4.2 test vs train in logistic regression*

**Train vs test in scaled logistic regression:**

```
[70] models = ['training','testing']
     acc_scores = [0.86,0.82]
     plt.barh(models, acc_scores, color=['lightgreen', 'grey' ])
     plt.ylabel("accuracy scores")
     plt.title("train vs test")
     plt.show()
```

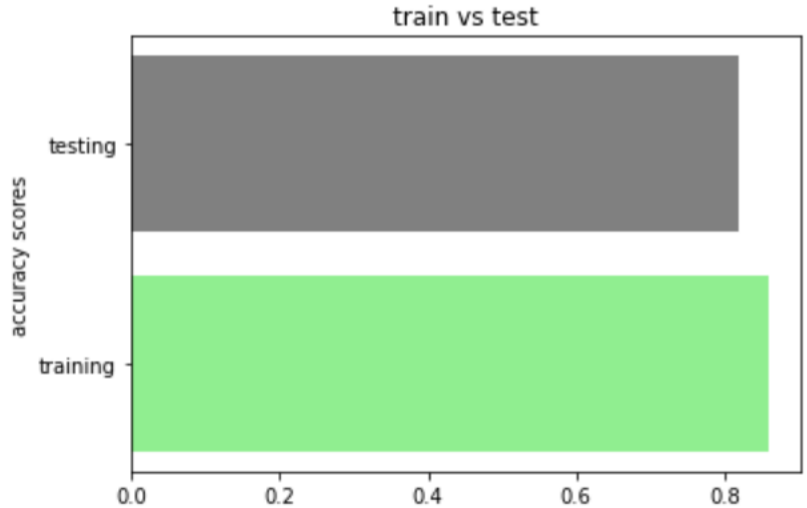



*Fig 7.4.3 test vs train in scaled logistic regression*

**Train vs Test in Decision Tree:**

```
[81] models = ['training','testing']
     acc_scores = [1,0.87]
     plt.barh(models, acc_scores, color=['lightgreen', 'grey' ])
     plt.ylabel("accuracy scores")
     plt.title("train vs test")
     plt.show()
```

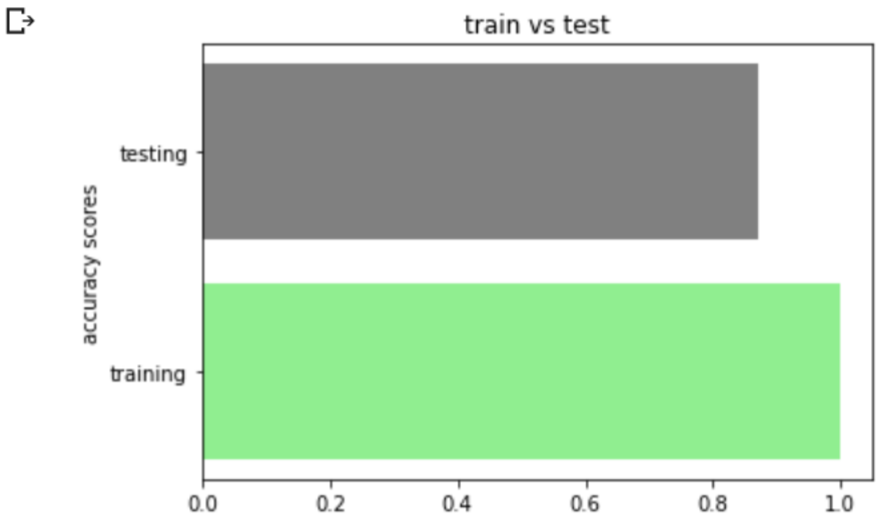



*Fig 7.4.4 test vs train in decision tree*

**Train vs Test in Random Forest:**

```
[93] models = ['training','testing']
     acc_scores = [1,0.88]
     plt.barh(models, acc_scores, color=['lightgreen', 'grey' ])
     plt.ylabel("accuracy scores")
     plt.title("train vs test")
     plt.show()
```

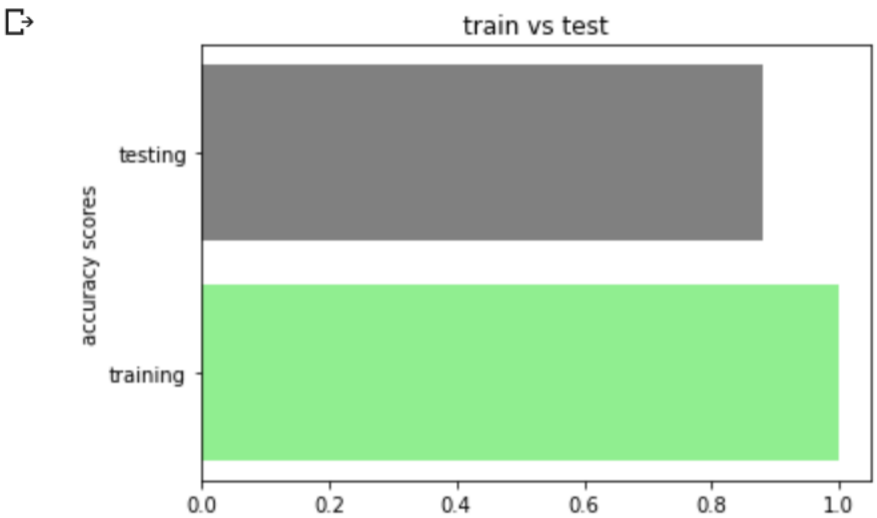



*Fig 7.4.5 test vs train in random forest*

## 7.5 Parameter Tuning:

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss. Cross-validation is often used to estimate this generalization performance.

**i. Grid search:**

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

For example, a typical soft-margin SVM classifier equipped with an RBF kernel has at least two hyperparameters that need to be tuned for good performance on unseen data: a regularization constant C and a kernel hyperparameter $\gamma$. Both parameters are continuous, so to perform grid search, one selects a finite set of "reasonable" values for each, say

**ii.Random search:**

Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm In this case, the optimization problem is said to have a low intrinsic dimensionality. Random Search is also embarrassingly parallel, and additionally allows the inclusion of prior knowledge by specifying the distribution from which to sample.

**iii.Bayesian optimization:**

Bayesian optimization is a global optimization method for noisy black-box functions. Applied to hyperparameter optimization, Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set. By iteratively evaluating a promising hyperparameter configuration based on the current model, and then updating it, Bayesian optimization, aims to gather observations revealing as much information as possible about this function and, in particular, the location of the optimum. It tries to balance exploration (hyperparameters for which the outcome is most uncertain) and exploitation (hyperparameters expected close to the optimum). In practice, Bayesian optimization has been shown to obtain better results in fewer evaluations compared to grid search and random search, due to the ability to reason about the quality of experiments before they are run.

### We use Grid Search cv to increse the testing accuracy in decision tree

```
[83] from sklearn.tree import DecisionTreeClassifier
     dt=DecisionTreeClassifier()
     grid_params={
         'criterion':['gini', 'entropy'],
         'splitter':['best','random'],
         'max_depth': range(1,11,2),
         'min_samples_leaf':range(1,6,3)
     }
     from sklearn.model_selection import GridSearchCV
     grid=GridSearchCV(estimator=dt,param_grid=grid_params)
     grid.fit(X_train,y_train)
     grid.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 9,
 'min_samples_leaf': 4,
 'splitter': 'best'}
```

```
[84] final_model=DecisionTreeClassifier(criterion='entropy',
                                        max_depth=9,
                                        min_samples_leaf=6,
                                        splitter='best')
     final_model.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=9, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=6, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

*Fig 7.5.1 grid search cv*

Predicting the test data

```
[85] pred_test = final_model.predict(X_test)
```

classification report of actual values and predicted values(gridsearch)

```
[86] print(classification_report(y_test,pred_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93        94
           1       0.78      0.88      0.83       102
           2       0.90      0.78      0.83       120
           3       0.92      0.93      0.92        84

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

**Testing Accuracy**

```
[87] acc_dt = metrics.accuracy_score(y_test, pred_test)
     acc_dt
```

```
0.8725
```

In grid based Cv Accuracy of decision tree testing accuracy is 87%

## 7.6 Confusion Matrix in Algorithm:

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

Prediction on test data

```
[ ] y_pred_rf = rf.predict(X_test)
    print(metrics.confusion_matrix(y_test, y_pred_rf))
```

```
[[ 87   7   0   0]
 [  7  85  10   0]
 [  0  14 100   6]
 [  0   0   3  81]]
```

*Fig 7.6.1 Confusion Matrix of test dataset in Random Forest*

```
sns.heatmap(y,annot=True,annot_kws={"size": 16}, fmt='g') #test heatmap
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1b83dc470>
```

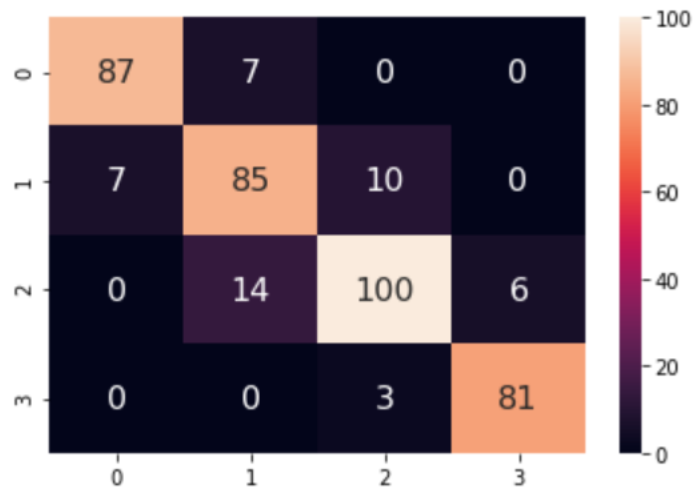



*Fig 7.6.2 Heat Map of confusion matrix*

- **Positive (P) :** Observation is positive.
- **Negative (N) :** Observation is not positive.
- **True Positive (TP) :** Observation is positive, and is predicted to be positive.
- **False Negative (FN) :** Observation is positive, but is predicted negative.
- **True Negative (TN) :** Observation is negative, and is predicted to be negative.

- **False Positive (FP) :** Observation is negative, but is predicted positive.

**Classification Rate/Accuracy:**

Classification Rate or Accuracy is given by the relation:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

*Fig 7.6.3 Accuracy*

**Recall:**

$$\textit{Recall} = \frac{TP}{TP + FN}$$

*Fig 7.6.4 Recall*

**Precision:**

$$\textit{Precision} = \frac{TP}{TP + FP}$$

*Fig 7.6.5 Precision*

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labelled as positive is indeed positive (a small number of FP).

**High recall, low precision:** This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

**Low recall, high precision:** This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP).

## 7.7 Classification Report:

The classification report shows a representation of the main classification metrics on a per-class basis. This gives a deeper intuition of the classifier behavior over global accuracy which can mask functional weaknesses in one class of a multiclass problem. Visual classification reports are used to compare classification models to select models The metrics are defined in terms of true and false positives, and true and false negatives. Positive and negative in this case are generic names for the classes of a binary classification problem. In the example above, we would consider true and false occupied and true and false unoccupied. Therefore a true positive is when the actual class is positive as is the estimated class. A false positive is when the actual class is negative but the estimated class is positive.

```
[75] print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        94
           1       0.80      0.83      0.82       102
           2       0.88      0.83      0.86       120
           3       0.93      0.96      0.95        84

    accuracy                           0.88       400
   macro avg       0.89      0.89      0.89       400
weighted avg       0.88      0.88      0.88       400
```

*Fig 7.7.1 Test Classification Report*

- **Precision:** Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.
- **Recall:** Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.
- **F1-Score :** The $F_1$ score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, $F_1$ scores are lower than accuracy

measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of $F_1$ should be used to compare classifier models, not global accuracy.

- **Support:** Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

### 7.7.1 ROC Curve:

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection[1] in machine learning. The false-positive rate is also known as probability of false alarm[1] and can be calculated as (1 − specificity). It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity or recall as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from {\displaystyle -\infty }-\infty to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis.

ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.

```
plt.show()
```

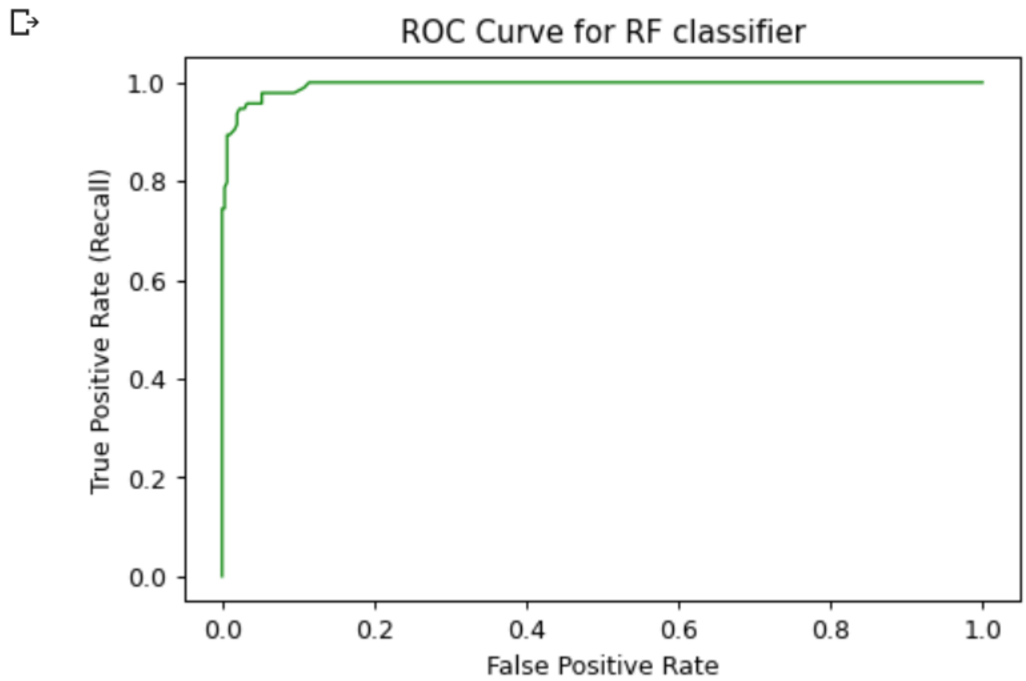



*Fig 7.7.1 Roc curve for random forest*

## 7.8 Predictions from raw data:

- After completion of accuracy we have to predict best model among them onto raw data
- Load the test dataset to predict
- Check the rows and columns and also shape
- After completion predict using best algorithm onto train dataset
- Add new column of predicted data i.e. output

Relation between logistic regression, scaled logistic regression, Decision Tree, Random Forest

**Predicting the best algorith among logistic regression, sacled logistic regression, decision tree, random forest**

```
[ ] models = ['scaled Logistic regression','logistic regression', 'decision tree','Random Forest']
    acc_scores = [0.82, 0.64, 0.87,0.882]
    plt.bar(models, acc_scores, color=['lightblue', 'pink', 'lightgrey'])
    plt.ylabel("accuracy scores")
    plt.title("Which model is the most accurate?")
    plt.show()
```

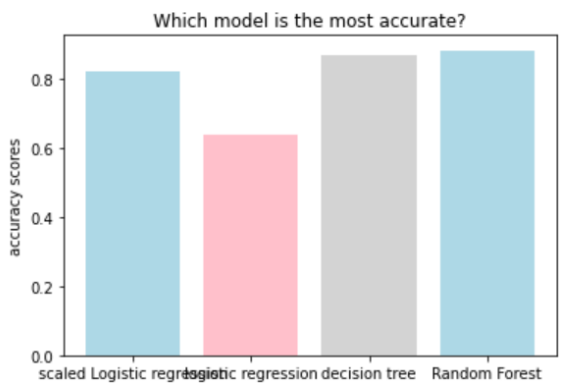



*Fig 7.8.1 best accuracy model algorithm*

**Observations:**

- Testing accuracy of logistic regression is 64%
- Testing accuracy of scaled logistic regression is 82%
- Testing accuracy of Decision tree is 87%
- Testing accuracy of Random Forest is 88%

Random Forest has best accuracy. So, we choose random Forest

**Prediction on testing dataset**

```
[94] df_test=pd.read_csv('https://raw.githubusercontent.com/venkanna831/AIML/master/Project/test.csv')
```

```
[95] df_test.head()
```

| | id | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_cores | pc | px_height | px_width | ram | sc_h | sc_w | talk_ti |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1043 | 1 | 1.8 | 1 | 14 | 0 | 5 | 0.1 | 193 | 3 | 16 | 226 | 1412 | 3476 | 12 | 7 | |
| **1** | 2 | 841 | 1 | 0.5 | 1 | 4 | 1 | 61 | 0.8 | 191 | 5 | 12 | 746 | 857 | 3895 | 6 | 0 | |
| **2** | 3 | 1807 | 1 | 2.8 | 0 | 1 | 0 | 27 | 0.9 | 186 | 3 | 4 | 1270 | 1366 | 2396 | 17 | 10 | |
| **3** | 4 | 1546 | 0 | 0.5 | 1 | 18 | 1 | 25 | 0.5 | 96 | 8 | 20 | 295 | 1752 | 3893 | 10 | 0 | |
| **4** | 5 | 1434 | 0 | 1.4 | 0 | 11 | 1 | 49 | 0.5 | 108 | 6 | 18 | 749 | 810 | 1773 | 15 | 8 | |

*Fig 7.8.2 Loading Test data set*

### Predicting test data using Random Forest

```
[98] predicted_price_range = rf.predict(df_test)
```

```
[99] predicted_price_range
```

```
array([3, 3, 2, 3, 1, 3, 3, 1, 3, 0, 3, 3, 0, 0, 2, 0, 2, 1, 3, 2, 1, 3,
       1, 1, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 0, 1, 3, 1, 2, 1, 1, 2, 0, 0,
       0, 0, 0, 3, 1, 2, 1, 0, 3, 0, 3, 1, 3, 1, 1, 3, 3, 2, 0, 1, 1, 1,
       1, 2, 1, 2, 1, 2, 2, 3, 3, 0, 2, 0, 2, 3, 0, 3, 3, 0, 3, 0, 3, 1,
       3, 0, 1, 1, 2, 0, 2, 1, 0, 2, 1, 3, 1, 0, 0, 3, 1, 2, 0, 1, 2, 3,
       3, 3, 1, 3, 3, 3, 3, 1, 3, 0, 0, 3, 2, 1, 1, 0, 3, 2, 3, 1, 0, 2,
       1, 1, 3, 1, 1, 0, 3, 2, 1, 3, 1, 3, 2, 3, 3, 3, 2, 3, 2, 3, 0, 0,
       3, 2, 3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 1, 0, 3, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 1, 2, 0, 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 1, 0, 3, 1, 0, 2, 2,
       2, 3, 1, 2, 3, 3, 3, 1, 2, 1, 0, 0, 1, 2, 0, 2, 3, 3, 0, 2, 0, 3,
       2, 3, 3, 0, 0, 1, 0, 3, 0, 1, 0, 2, 2, 1, 3, 0, 3, 0, 3, 1, 2, 0,
       0, 2, 1, 3, 3, 3, 1, 1, 3, 0, 0, 2, 3, 3, 1, 3, 1, 1, 3, 2, 1, 2,
       3, 3, 3, 1, 0, 1, 1, 3, 1, 1, 3, 2, 0, 3, 0, 1, 2, 0, 0, 3, 2, 3,
       3, 2, 1, 3, 3, 2, 3, 2, 2, 1, 1, 0, 2, 3, 1, 0, 0, 3, 0, 3, 0, 1,
       2, 0, 2, 3, 1, 3, 2, 2, 1, 2, 0, 0, 0, 1, 3, 2, 0, 0, 0, 3, 1, 0,
       3, 3, 1, 2, 3, 2, 3, 1, 3, 3, 2, 2, 3, 3, 3, 0, 3, 0, 3, 1, 3, 1,
       2, 3, 0, 1, 1, 3, 1, 3, 1, 3, 0, 0, 0, 0, 2, 0, 0, 2, 1, 1, 2, 3,
       2, 0, 1, 0, 0, 3, 3, 0, 3, 1, 2, 2, 1, 1, 3, 1, 1, 2, 2, 1, 2, 0,
       1, 1, 0, 3, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 2, 2, 3, 2, 3, 0, 3,
       0, 3, 0, 1, 0, 1, 2, 0, 3, 2, 3, 3, 1, 3, 1, 3, 1, 3, 2, 1, 2, 2,
       1, 1, 0, 0, 0, 1, 2, 1, 0, 3, 3, 0, 2, 3, 0, 0, 3, 1, 1, 1, 3, 2,
```

*Fig 7.8.3 Predicting test dataset using random forest*

### Add new column price range

Adding the new column "price_range" in the test dataset

```
[100] df_test['price_range'] = predicted_price_range
```

```
[101] df_test.head()
```

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_cores | pc | px_height | px_width | ram | sc_h | sc_w | talk_time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1043 | 1 | 1.8 | 1 | 14 | 0 | 5 | 0.1 | 193 | 3 | 16 | 226 | 1412 | 3476 | 12 | 7 | 2 |
| 1 | 841 | 1 | 0.5 | 1 | 4 | 1 | 61 | 0.8 | 191 | 5 | 12 | 746 | 857 | 3895 | 6 | 0 | 7 |
| 2 | 1807 | 1 | 2.8 | 0 | 1 | 0 | 27 | 0.9 | 186 | 3 | 4 | 1270 | 1366 | 2396 | 17 | 10 | 10 |
| 3 | 1546 | 0 | 0.5 | 1 | 18 | 1 | 25 | 0.5 | 96 | 8 | 20 | 295 | 1752 | 3893 | 10 | 0 | 7 |
| 4 | 1434 | 0 | 1.4 | 0 | 11 | 1 | 49 | 0.5 | 108 | 6 | 18 | 749 | 810 | 1773 | 15 | 8 | 7 |

*Fig 7.6.4 Added new column price_range*

### Range of target in train dataset:

```
[102] df_train['price_range']
```

```
0       1
1       2
2       2
3       2
4       1
       ..
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 2000, dtype: int64
```

*Fig 7.8.5 range of target in train dataset*

**Example of predicted dataset:**

```
[103] pd.DataFrame({'ram' : df_test['ram'],'price_range' : predicted_price_range})
```

| | ram | price_range |
|---|---|---|
| **0** | 3476 | 3 |
| **1** | 3895 | 3 |
| **2** | 2396 | 2 |
| **3** | 3893 | 3 |
| **4** | 1773 | 1 |
| **...** | ... | ... |
| **995** | 2121 | 2 |
| **996** | 1933 | 1 |
| **997** | 1223 | 0 |
| **998** | 2509 | 2 |
| **999** | 2828 | 2 |

1000 rows × 2 columns

*Fig 7.8.7 example of predicted test dataset*

Here we have compared the ram with price range.

**Observations:**

- If ram is 3476 then price range is classified has 3
- It is similar to train dataset

# 8.CONCLUSION

It is concluded after performing thorough Exploratory Data analysis which include Statistics models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set and its come to point of getting the solution for the problem statement being , that the bob should open a mobile shop by classified features of a mobile phone(e.g.:-RAM, Internal Memory etc.) and its selling price and gives tough fight to big companies like Apple, Samsung etc., By classified selling price range .

# 9.REFERENCES


- https://en.wikipedia.org/wiki/Machine_learning
- https://towardsdatascience.com/supervised-machine-learning-model-validation-a-step-by-step-approach-771109ae0253
- https://www.kaggle.com/iabhishekofficial/mobile-price-classification/kernels
- https://builtin.com/data-science/random-forest-algorithm