1. INTRODUCTION

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.

1.1 HISTORY

Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL).

About the origin of Python, Van Rossum wrote in 1996:

Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home Computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

Downloading python :

If you don’t already have a copy of Python installed on your computer, you will need to open up your Internet browser and go to the Python download page .

(http://www.python.org/download/).

FIG1.1:INSTALLATION OF PYTHON

1.2 Python Features:

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

2) Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type print("Hello World"). It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

5) Free and Open Source:

Python is freely available for everyone. It is freely available on its official website www.python.org

It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

8) Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

10) Integrated

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

11. Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.

1.3 MACHIENE LEARNING:

M.Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term “Machine Learning ” in 1959 while at IBM. He defined machine learning as “the field of study that gives computers the ability to learn without being explicitly programmed “. However, there is no universally accepted definition for machine learning. Different authors define the term differently. We give below two more definitions.Machine learning is programming computers to optimize a performance criterion using example data or past experience . We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data.

The field of study known as machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

Definition of learning: A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks T, as measured by P , improves with experience E.

2. FUNDAMENTALS OF PYTHON

2.1 KEYWORDS

Keywords are the reserved words in Python. We cannot use a keyword as a variable name, function name or any other identifier.

Keywords in Python programming language

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

FIG2.1 KEYWORDS IN PYTHON

2.2 IDENTIFIERS

An identifier is a name given to entities like variables, class, functions, etc. It helps to differentiate one entity from another and understand the flow of entities in code.

2.21 Rules for Identifiers in Python

Combination of alphabets in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _

- Digits cannot be used as the starting of an identifier.
- Keywords cannot be used as identifiers
- Special symbols !, @, #, $, % etc. cannot be used in anidentifier
- There is no limit on the length of anidentifier.
- Ex: abz123, abc_123

2.3 VARIABLES

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.

```
Ex: counter = 100                    # An integer
    assignment miles   = 1000.0      # A floating
     point name    = "John"          # A string
```

2.4 DATA TYPES

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary

2.4.1 NUMERIC

In Python, numeric data type represent the data which has numeric value. integer, floating number , even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

2.4.2 SEQUENCE TYPE

In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

1. String

In programming terms, we usually call text a string. When you think of a string as a collection of letters, the term makes sense.

All the letters, numbers, and symbols in this book could be a string. For that matter, your name could be a string, and so could your address.

Creating Strings

In Python, we create a string by putting quotes around text. For example, we could take our otherwise useless

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing

- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search

2. Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses.

Python Expression	**Results**	**Description**
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Accessing Values in Tuples:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example − tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5, 6, 7); print "tup1[0]: ", tup1[0] print "tup2[1:5]: ", tup2[1:5]

When the above code is executed, it produces the following result − tup1[0]: physics
tup2[1:5]: [2, 3, 4, 5]

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

Built-in Tuple Functions

Python includes the following tuple functions −

SN	Function with Description
1	**cmp(tuple1, tuple2)** Compares elements of both tuples.
2	**len(tuple)** Gives the total length of the tuple.
3	**max(tuple)** Returns item from the tuple with max value.
4	**min(tuple)** Returns item from the tuple with min value.
5	**tuple(seq)** Converts a list into tuple.

3. List:

The list is a most versatile datatype available in Python which can be written as a list of comma separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type. Creating a list is as simple as putting different comma-separated values between square brackets. For example − list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5]; list3 = ["a", "b", "c", "d"]; Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Basic List Operation

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods:

SN	Function with Description
1	**cmp(list1, list2)** Compares elements of both lists.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

Python includes following list methods

SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list

4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

2.4.3 BOOLEAN

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool

Note – True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

Set

In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Creating Sets

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by 'comma'. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

Accessing elements of Sets

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Dictionary

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

Creating Dictionary

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces{}.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

3. OPERATORS

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

ARITHEMATIC OPERATORS

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2

% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

COMPARISION OPERATORS

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.

<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

ASSIGNMENT OPERATORS

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c

+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = **a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c /a

BITWISE OPERATORS

Bitwise operator works on bits and performs bitLO by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

- - - - - - - - - - - -

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means

Complement		1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

LOGICAL OPERATORS

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

MEMBERSHIP OPERATORS

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

IDENTITY OPERATORS

Identity operators compare the memory locations of two objects. There are two Identity operators explained below –

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Bitwise operators

In Python, bitwise operators are used to performing bitwise calculations on integers. The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

Note: Python bitwise operators work only on integers.

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	x & y

OPERATOR	DESCRIPTION	SYNTAX
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

4. LOOPS

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −

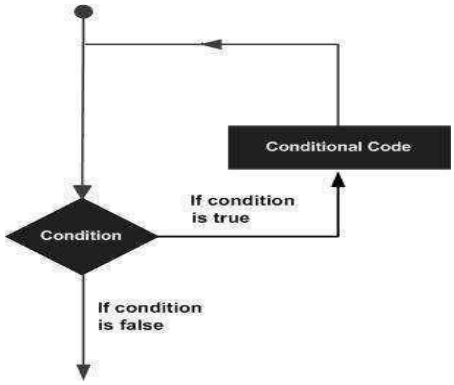


FIG4.1 IF LOOP

Python programming language provides following types of loops to handle looping requirements.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

nested loops	You can use one or more loop inside any another while, for or do..while loop.

Loop Example:

4.1 FOR LOOP:

>>> for mynum in [1, 2, 3, 4, 5]:

print ("Hello", mynum)

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5

While Loop:

>>> count = 0 >>while(count< 4):

print 'The count is:', count count = count + 1

The count is: 0

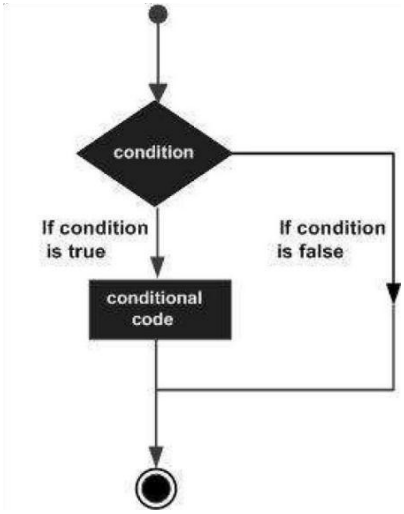
The count is: 1

The count is: 2

The count is: 3

4.2 CONDITIONAL STATEMENTS:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.



4.2.1 CONDITIONAL STATEMENT

Python programming language provides following types of decision making statements. Click the following links to check their detail.

Statement	Description
if statements	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statements	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

Example:

If Statement: a=33 b=200

If b>a: print(“b”)

If...Else Statement: a=200 b=33 if b>a:

print(“b is greater than a”) else:

print(“a is greater than b”)

5. FUNCTIONS

A function is a collection of related assertions that performs a mathematical, analytical, or evaluative operation. Python functions are simple to define and essential to intermediate-level programming. The exact criteria hold to function names as they do to variable names. The goal is to group up certain often performed actions and define a function. Rather than rewriting the same code block over and over for varied input variables, we may call the function and repurpose the code included within it with different variables.

The functions are broad of two types, user-defined and built-in functions. It aids in keeping the software succinct, non-repetitive, and well-organized.

Advantages of Functions in Python:

Python functions have the following benefits.

- By including functions, we can prevent repeating the same code block repeatedly in a program.
- Python functions, once defined, can be called many times and from anywhere in a program.
- If our Python program is large, it can be separated into numerous functions which is simple to track.
- The key accomplishment of Python functions is we can return as many outputs as we want with different arguments.
- However, calling functions has always been overhead in a Python program.

5.1 Syntax of Python Function:

```
def name_of_function( parameters ):
    """This is a docstring"""
```

- The following elements make up define a function, as seen above.
- The beginning of a function header is indicated by a keyword called def.
- name_of_function is the function's name that we can use to separate it from others. We will use this name to call the function later in the program. The same criteria apply to naming functions as to naming variables in Python.
- We pass arguments to the defined function using parameters. They are optional, though.
- The function header is terminated by a colon (:).

- We can use a documentation string called docstring in the short form to explain the purpose of the function.
- The body of the function is made up of several valid Python statements. The indentation depth of the whole code block must be the same (usually 4 spaces).
- We can use a return expression to return a value from a defined function.

5.2 Calling a Function:

A function is defined by using the def keyword and giving it a name, specifying the arguments that must be passed to the function, and structuring the code block.

After a function's fundamental framework is complete, we can call it from anywhere in the program. The following is an example of how to use the a_function function.

5.3 Function Arguments

The following are the types of arguments that we can use to call a function:

- Default arguments
- Keyword arguments
- Required arguments
- Variable-length arguments

Python Lambda Functions:

Lambda Functions in Python are anonymous functions, implying they don't have a name. The def keyword is needed to create a typical function in Python, as we already know. We can also use the lambda keyword in Python to define an unnamed function.

Syntax of Python Lambda Function:

lambda arguments: expression

This function accepts any count of inputs but only evaluates and returns one expression.

Lambda functions can be used whenever function arguments are necessary. In addition to other forms of formulations in functions, it has a variety of applications in certain coding domains. It's important to remember that according to syntax, lambda functions are limited to a single statement.

Using Lambda Function with filter()

The filter() method accepts two arguments in Python: a function and an iterable such as a list.The function is called for every item of the list, and a new iterable or list is returned that holds just those elements that returned True when supplied to the function.Here's a simple illustration of using the filter() method to return only odd numbers from a list.

Code

```
# Code to filter odd numbers from a given list

list_ = [34, 12, 64, 55, 75, 13, 63]

odd_list = list(filter( lambda num: (num % 2 != 0) , list_ ))

print(odd_list)
```

55, 75, 13, 63]

Using Lambda Function with map()

A method and a list are passed to Python's map() function.

The function is executed for all of the elements within the list, and a new list is produced with elements generated by the given function for every item.

The map() method is used to square all the entries in a list in this example.

```
#Code to calculate the square of each number of a list using the map() function

 numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]

 squared_list = list(map( lambda num: num ** 2 , numbers_list ))

 print( squared_list )
```

10.0

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- **Exception Handling** − This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- **Assertions** − This would be covered in Assertions in Python tutorial.

List of Standard Exceptions −

Sr.No.	Exception Name & Description
1	**Exception** Base class for all exceptions
2	**StopIteration** Raised when the next() method of an iterator does not point to any object.
3	**SystemExit** Raised by the sys.exit() function.
4	**StandardError** Base class for all built-in exceptions except StopIteration and SystemExit.
5	**ArithmeticError** Base class for all errors that occur for numeric calculation.
6	**OverflowError** Raised when a calculation exceeds maximum limit for a numeric type.
7	**FloatingPointError** Raised when a floating point calculation fails.
8	**ZeroDivisionError** Raised when division or modulo by zero takes place for all numeric types.
9	**AssertionError**

	Raised in case of failure of the Assert statement.
10	**AttributeError** Raised in case of failure of attribute reference or assignment.
11	**EOFError** Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	**ImportError** Raised when an import statement fails.
13	**KeyboardInterrupt** Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	**LookupError** Base class for all lookup errors.
15	**IndexError** Raised when an index is not found in a sequence.
16	**KeyError** Raised when the specified key is not found in the dictionary.

6. LIBRARIES

6.1 NumPy

NumPy stands for numeric python which is a python package for the computation and processing of the multidimensional and single dimensional array elements.Travis Oliphant created NumPy package in 2005 by injecting the features of the ancestor module Numeric into another module Numarray.It is an extension module of Python which is mostly written in C. It provides various functions which are capable of performing the numeric computations with a high speed.NumPy provides various powerful data structures, implementing multi-dimensional arrays and matrices. These data structures are used for the optimal computations regarding arrays and matrices.In this tutorial, we will go through the numeric python library NumPy.

The need of NumPy

With the revolution of data science, data analysis libraries like NumPy, SciPy, Pandas, etc. have seen a lot of growth. With a much easier syntax than other programming languages, python is the first choice language for the data scientist.NumPy provides a convenient and efficient way to handle the vast amount of data. NumPy is also very convenient with Matrix multiplication and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.There are the following advantages of using NumPy for data analysis.

1. NumPy performs array-oriented computing.
2. It efficiently implements the multidimensional arrays.
3. It performs scientific computations.
4. It is capable of performing Fourier Transform and reshaping the data stored in multidimensional arrays.
5. NumPy provides the in-built functions for linear algebra and random number generation.

Nowadays, NumPy in combination with SciPy and Mat-plotlib is used as the replacement to MATLAB as Python is more complete and easier programming language than MATLAB.

6.2 PANDAS

Pandas is defined as an open-source library that provides high-performance data manipulation in Python. The name of Pandas is derived from the word **Panel Data**, which means **an Econometrics from Multidimensional data**. It is used for data analysis in Python and developed by **Wes McKinney** in **2008**.Data analysis requires lots of processing, such as **restructuring,**

cleaning or **merging**, etc. There are different tools are available for fast data processing, such as **Numpy, Scipy, Cython**, and **Panda**. But we prefer Pandas because working with Pandas is fast, simple and more expressive than other tools.Pandas is built on top of the **Numpy** package, means **Numpy** is required for operating the Pandas.Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., **load, manipulate, prepare, model, and analyze**.

Key Features of Pandas

- It has a fast and efficient DataFrame object with the default and customized indexing.
- Used for reshaping and pivoting of the data sets.
- Group by data for aggregations and transformations.
- It is used for data alignment and integration of the missing data.
- Provide the functionality of Time Series.
- Process a variety of data sets in different formats like matrix data, tabular heterogeneous, time series.
- Handle multiple operations of the data sets such as subsetting, slicing, filtering, groupBy, re-ordering, and re-shaping.
- It integrates with the other libraries such as SciPy, and scikit-learn.
- Provides fast performance, and If you want to speed it, even more, you can use the **Cython**.

Python Pandas Data Structure

The Pandas provides two data structures for processing the data, i.e., **Series** and **DataFrame**, which are discussed below:

1) Series

It is defined as a one-dimensional array that is capable of storing various data types. The row labels of series are called the **index**. We can easily convert the list, tuple, and dictionary into series using "series' method. A Series cannot contain multiple columns. It has one parameter:

Creating Series from Array: Before creating a Series, Firstly, we have to import the numpy module and then use array() function in the program.

```
1. import pandas as pd
2. import numpy as np
3. info = np.array(['P','a','n','d','a','s'])
4. a = pd.Series(info)
5. print(a)
```

Output

```
0   P
1   a
2   n
3   d
4   a
5   s
dtype: object
```

Python Pandas DataFrame

It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

Create a DataFrame using List:

We can easily create a DataFrame in Pandas using list.

```
1. import pandas as pd
2. # a list of strings
3. x = ['Python', 'Pandas']
4. # Calling DataFrame constructor on list
5. df = pd.DataFrame(x)
6. print(df)
```

Output

```
      0
0   Python
   1     Pandas
```

6.3 MATPLOTLIB

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Installation:

Windows, Linux and macOS distributions have matplotlib and most of its dependencies as wheel packages. Run the following command to install matplotlib package :

python -mpip install -U matplotlib

Importing matplotlib :

from matplotlib import pyplot as plt

or

import matplotlib.pyplot as plt

Basic plots in Matplotlib :

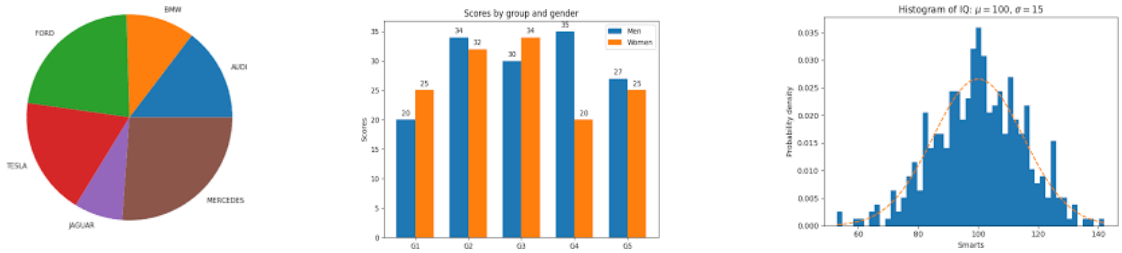


FIG6.3.1:BASIC PLOTS

7. MACHIENE LEARNING ALGORITHMS

There are many types of Machine Learning Algorithms specific to different use cases. As we work with datasets, a machine learning algorithm works in two stages. We usually split the data around 20%-80% between testing and training stages. Under supervised learning, we split a dataset into a training data and test data in Python ML. Followings are the Algorithms of Python Machine Learning –

7.1 SUPERVISED LEARNING:

1. LINEAR REGRESSION

Linear regression is one of the supervised Machine learning algorithms in Python that observes continuous features and predicts an outcome. Depending on whether it runs on a single variable or on many features, we can call it simple linear regression or multiple linear regression. This is one of the most popular Python ML algorithms and often under-appreciated. It assigns optimal weights to variables to create a line ax+b to predict the output. We often use linear regression to estimate real values like a number of calls and costs of houses based on continuous variables. The regression line is the best line that fits Y=a*X+b to denote a relationship between independent and dependent variables.

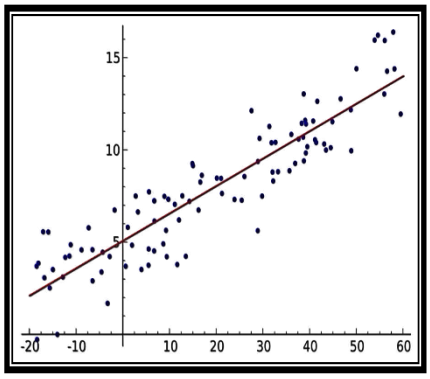


FIG 7.1: LINEAR REGRESSION

2. LOGISTIC REGRESSION

Logistic regression is a supervised classification is unique Machine Learning algorithms in Python that finds its use in estimating discrete values like 0/1, yes/no, and true/false. This is based on a given set of independent variables. We use a logistic function to predict the probability of an event and this

gives us an output between 0 and 1. Although it says 'regression', this is actually a classification algorithm. Logistic regression fits data into a logit function and is also called logit regression.

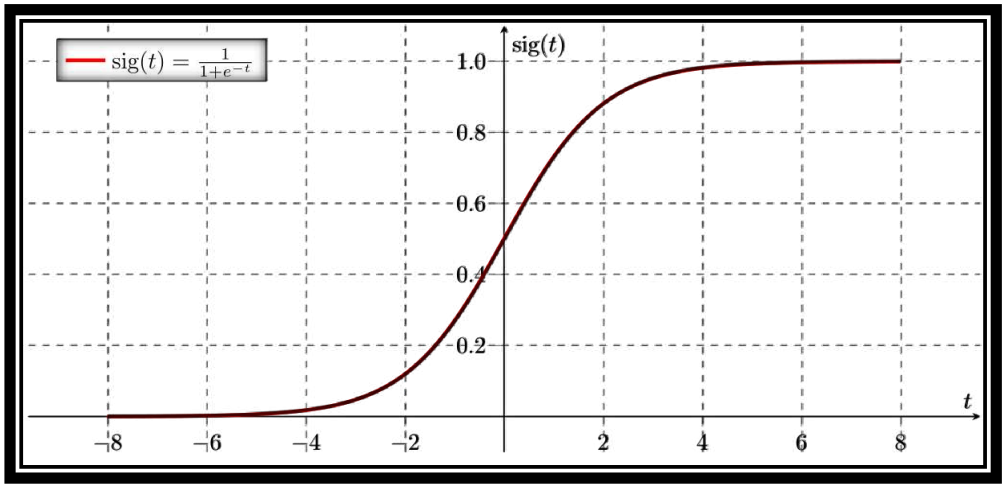


FIG7.2:LOGISTIC REGRESSION

3. DECISION TREES-

A decision tree falls under supervised Machine Learning Algorithms in Python and comes of use for both classification and regression- although mostly for classification. This model takes an instance, traverses the tree, and compares important features with a determined conditional statement. Whether it descends to the left child branch or the right depends on the result. Usually, more important features are closer to the root. Decision Tree, a Machine Learning algorithm in Python can work on both categorical and continuous dependent variables. Here, we split a population into two or more homogeneous sets. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

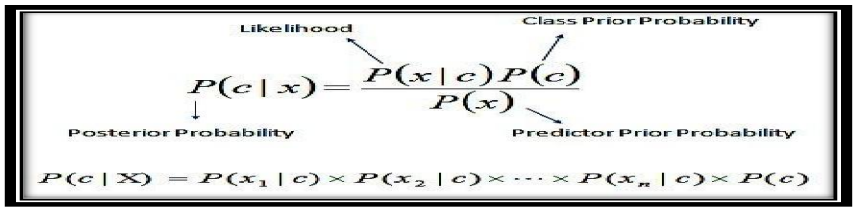
4 .SVM-

SVM is a supervised classification is one of the most important Machines Learning algorithms in Python, that plots a line that divides different categories of your data. In this ML algorithm, we calculate the vector to optimize the line. This is to ensure that the closest point in each group lies farthest from each other. While you will almost always find this to be a linear vector, it can be other than that. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. In addition

to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. When data are unlabeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups.

5. NAIVE BAYES

Naive Bayes is a classification method which is based on Bayes' theorem. This assumes independence between predictors. A Naive Bayes classifier will assume that a feature in a class is unrelated to any other. Consider a fruit. This is an apple if it is round, red, and 2.5 inches in diameter. A Naive Bayes classifier will say these characteristics independently contribute to the probability of the fruit being an apple. This is even if features depend on each other. For very large data sets, it is easy to build a Naive Bayesian model. Not only is this model very simple, it performs better than many highly sophisticated classification methods. Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.



6. KNN -

This is a Python Machine Learning algorithm for classification and regression- mostly for classification. This is a supervised learning algorithm that considers different centroids and uses a usually Euclidean function to compare distance. Then, it analyzes the results and classifies each point to the group to optimize it to place with all closest points to it. It classifies new cases using a majority vote of k of its neighbors. The case it assigns to a class is the one most common among its K nearest neighbors. For this, it uses a distance function. *k*-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. *k*-NN is a special case of a variablebandwidth, kernel density "balloon" estimator with a uniform kernel.

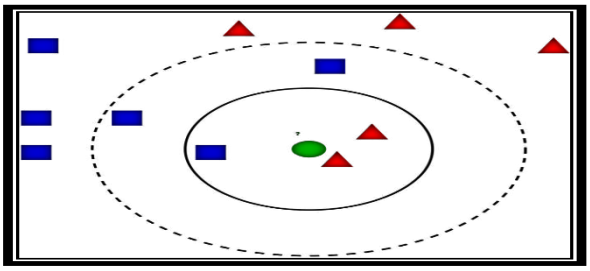


FIG 7.3:KNN CLASSIFIER

7. RANDOM FOREST :

A random forest is an ensemble of decision trees. In order to classify every new object based on its attributes, trees vote for class- each tree provides a classification. The classification with the most votes wins in the forest. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision tree at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

7.2 UNSUPERVISED LEARNING:

1. K-MEANS :

k-Means is an unsupervised algorithm that solves the problem of clustering. It classifies data using a number of clusters. The data points inside a class are homogeneous and heterogeneous to peer groups. *k*-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. *k*-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. *k*-means clustering is rather easy to apply to even large data sets, particularly when using heuristics such as Lloyd's algorithm. It often is used as a preprocessing step for other algorithms, for example to find a starting configuration. The problem is computationally difficult (NP-hard). *k*-means originates from signal processing, and still finds use in this domain. In cluster analysis, the *k*-means algorithm can be used to partition the input data set into *k* partitions (clusters). *k*-means clustering has been used as a feature learning (or dictionary learning) step, in either (semi-)supervised learning or unsupervised learning.

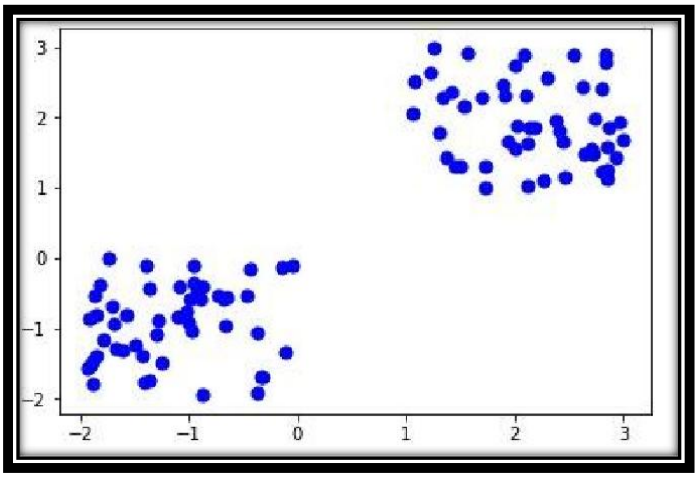


FIG 7.4 K-MEANS ALGORITHM

8. EVALUATION TECHNIQUES

8.1 MEAN ABSOLUTE ERROR (MAE):

Mean absolute error, also known as L1 loss is one of the simplest loss functions and an easy-to-understand evaluation metric. It is calculated by taking the absolute difference between the predicted values and the actual values and averaging it across the dataset. Mathematically speaking, it is the arithmetic average of absolute errors. MAE measures only the magnitude of the errors and doesn't concern itself with their direction. The lower the MAE, the higher the accuracy of a model.

Mathematically, MAE can be expressed as follows,

$$MAE = \frac{1}{n} \sum_{i=1}^{n} | y_i - \hat{y}_i |$$

where y i = actual value, y_hat_i = predicted value, n = sample size

8.2 MEAN SQUARED ERROR (MSE)

MSE is one of the most common regression loss functions. In Mean Squared Error also known as L2 loss, we calculate the error by squaring the difference between the predicted value and actual value and averaging it across the dataset. MSE is also known as Quadratic loss as the penalty is not proportional to the error but to the square of the error. Squaring the error gives higher weight to the outliers, which results in a smooth gradient for small errors. Optimization algorithms benefit from this penalization for large errors as it is helpful in finding the optimum values for parameters. MSE will never be negative since the errors are squared. The value of the error ranges from zero to infinity. MSE increases exponentially with an increase in error. A good model will have an MSE value closer to zero.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

8.3 ROOT MEAN SQUARE ERROR(RMSE)

RMSE is computed by taking the square root of MSE. RMSE is also called the Root Mean Square Deviation. It measures the average magnitude of the errors and is concerned with the

deviations from the actual value. RMSE value with zero indicates that the model has a perfect fit. The lower the RMSE, the better the model and its predictions. A higher RMSE indicates that there is a large deviation from the residual to the ground truth. RMSE can be used with different features as it helps in figuring out if the feature is improving the model's prediction or not.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

8.4 ROOT MEAN SQUARE LOG ERROR(RMSLE)

Taking the log of the RMSE metric slows down the scale of error. The metric is very helpful when you are developing a model without calling the inputs. In that case, the output will vary on a large scale.To control this situation of RMSE we take the log of calculated RMSE error and resultant we get as RMSLE.

8.5 R SQUARED(R2)

R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform.In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides. The same we have in classification problems which we call a threshold which is fixed at 0.5. So basically R2 squared calculates how must regression line is better than a mean line.Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.

$$\textbf{R2 Squared} = 1 - \frac{SSr}{SSm}$$

SSr = Squared sum error of regression line

SSm = Squared sum error of mean line

9. TASK: MOBILE PRICE PREDICTION

In this Modern Era, Smartphones are an integral part of the lives of human beings. When a smartphone is purchased ,many factors like the Display, Processor, Memory, Camera, Thickness, Battery, Connectivity and others are taken into account . One factor that people do not consider is whether the product is worth the cost . As there are no resources to cross validate the price , people fail in taking the correct decision. This paper looks to solve the problem by taking the historical data pertaining to the key features of smartphones along with its cost and develop a model that will predict the approximate price of the new smartphone with a reasonable accuracy. The dataset[12] used for this purpose has taken into consideration 21 different parameters for predicting the price of the phone . Random Forest Classifier, Support Vector Machine and Logistic Regression have been used primarily. Based on the accuracy , the appropriate algorithm has been used to predict the prices of the smartphone. This not only helps the customers decide the right phone to purchase , it also helps the owners decide what should be the appropriate pricing of the phone for the features that they offer. This idea of predicting the price will help the people make informed choice when they are purchasing a phone in the future. Among the three classifiers chosen , Logistic Regression and Support Vector Machine had highest accuracy of 81%. Further Logistic Regression was used to predict the prices of the phone. Keywords: Support Vector Machine, Logistics Regression , Smartphone Prices, Random Forest.

9.1 MODULES

9.1.1. DATA COLLECTION

Data collection is the process of collecting data from different sources to predict the price of mobile. For this Task I've collected data from kaggle with the link as https://www.kaggle.com/code/vikramb/mobile-price-prediction/data. The features are as follows

1. battery_power : Total energy a battery can store in one time measured in mAh
2. blue : Has bluetooth or not
3. clock_speed : speed at which microprocessor executes instructions
4. dual _sim : Has dual sim support or not

5. fc : Front Camera megapixels
6. four_g : Has 4G or not
7. int_memory : Internal Memory in Gigabytes
8. m_dep : Mobile Depth in cm
9. mobile_wt : Weight of mobile phone
10. n_cores : Number of cores of processor
11. pc : Primary Camera megapixels
12. px_height : Pixel Resolution Height
13. px_width : Pixel Resolution Width
14. ram : Random Access Memory in Megabytes
15. sc_h : Screen Height of mobile in cm
16. sc_w : Screen Width of mobile in cm
17. talk_time : longest time that a single battery charge will last when you are
18. three_g : Has 3G or not
19. touch_screen : Has touch screen or not
20. wifi : Has wifi or not
21. price_range : This is the target variable with values of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).

9.1.2. DATA PREPROCESSING

Data preprocessing is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data-gathering methods are often loosely controlled, resulting in out-of-range values, impossible data combinations, missing values, etc.

In this project you might be performing lot of preprocessing steps. Because in this dataset is not downloaded from Kaggle or any other data source website. This data retrieve from E-Commerce website. But after I was get the dataset I was make a dataset for model prediction. So you need not to and data preprocessing steps except handling the missing values.

9.1.2. EXPLORATORY DATA ANALYSIS

In statistics, exploratory data analysis (***EDA***) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but

primarily ***EDA*** is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

9.1.3. FEATURE SELECTION

Feature Selection is the process where you automatically or manually **s**elect 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.

9.1.4. RESULT

Cost prediction is the very important factor of marketing and business. To predict the cost same procedure can be performed for all types of products for example Cars, Foods, Medicine, Laptops etc. Best marketing strategy is to find optimal product (with minimum cost and maximum specifications). So products can be compared in terms of their specifications, cost, manufacturing company etc. By specifying economic range a good product can be suggested to a costumer.

9.2. IMPLEMENTATION

9.2.1. SAMPLE CODE

```
# Importing libraries
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
font = {'size' : 14}
plt.rc('font', **font)
warnings.filterwarnings("ignore")
# Import Dataset
data = pd.read_csv("C:/Users/nikhil/Downloads/mobile_price_range_data.csv")
test_data = pd.read_csv("C:/Users/nikhil/Downloads/test.csv")
x = data.iloc[:,0:20]
y = data.iloc[:,20]
pd.set_option('display.max_columns', None)
#Inspecting the training Dataset
data.head(20)
#Checking the missing values in our dataset if any
data.isnull().sum()
#Display information about our dataset
print("------------>Information about the Dataset<------------")
```

```
print()
data.info()
#Summary statistics of the training dataset
descrip=data.describe()
#Summary statistics of the test dataset
descrip_test=test_data.describe()
print("------------>Decription of the Training Data<------------")
print()
pd.DataFrame(data = descrip)
print("------------>Decription of the Testing Data< ------------ ")
print()
pd.DataFrame(data = descrip_test)
sns.set(style="white")
# Compute the correlation matrix
corr = data.corr()
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(27,18))
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,square=True, linewidths=.5,
cbar_kws={"shrink": .5})
```

```
plt.title("Correlation of Attirubtues")
#we are not considering the correlations between X features among themselves
corr = data.corr()
corr = corr.price_range
cr = corr.sort_values(ascending = False)[1:]
sns.barplot(x=cr, y=cr.index,palette = "bright")
plt.title("Correlation between Attributes and Price Range")
#Splitting the training and testing dataset
from sklearn.model_selection import train_test_split
# Splitting of data
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size = 0.2, random_state = 0,stratify = y)
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
col = data.iloc[:,:20].columns
x_train = sc.fit_transform(x_train)
x_valid = sc.fit_transform(x_valid)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(multi_class = 'multinomial', solver = 'sag', max_iter = 10000)
lr.fit(x_train, y_train)
lr.score(x_valid,y_valid)
y_pred_lr = lr.predict(x_valid)
#Visualizing the Confusion Matrix
cm = confusion_matrix(y_valid, y_pred_lr)
```

```
# label the confusion matrix

conf_matrix=pd.DataFrame(data=cm,columns=["Low Cost", "Budgeted", "Medium Cost",
"Flagship"],index=["Low Cost", "Budgeted", "Medium Cost", "Flagship"])

# plot a heatmap

fig, ax = plt.subplots(figsize=(10,8))

sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

plt.title("Confusion Matrix for Logistic Regression")

plt.show()

#Calculate Accuracy

acc_lr = accuracy_score(y_valid, y_pred_lr)

#Features Contribution

feature_imp = pd.Series(abs(lr.coef_[0]), index=col).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index,palette="bright")

plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

plt.title("Visualizing Important Features for Logistic Regression")

from sklearn.tree import DecisionTreeClassifier

dtg = DecisionTreeClassifier(random_state=101)

dtg.fit(x_train, y_train)

y_pred_dtg = dtg.predict(x_valid)

#Visualizing the Confusion Matrix

cm = confusion_matrix(y_valid, y_pred_dtg)

# label the confusion matrix

conf_matrix=pd.DataFrame(data=cm,columns=["Low Cost", "Budgeted", "Medium Cost",
"Flagship"],index=["Low Cost", "Budgeted", "Medium Cost", "Flagship"])
```

```
# plot a heatmap
fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
plt.title("Confusion Matrix for Decison Tree (Gini)")
plt.show()
#Calculate Accuracy
acc_dtg = accuracy_score(y_valid, y_pred_dtg)
#Features Contribution
feature_imp = pd.Series(dtg.feature_importances_, index = col).sort_values(ascending=False)
sns.barplot(x=feature_imp, y=feature_imp.index,palette = "bright")
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features for Decision Tree(Gini)")
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 100, random_state=0,criterion = 'entropy',oob_score =
True)
rf.fit(x_train, y_train)
y_pred_rf = rf.predict(x_valid)
#Visualizing the Confusion Matrix
cm = confusion_matrix(y_valid, y_pred_rf)
# label the confusion matrix
conf_matrix=pd.DataFrame(data=cm,columns=["Low Cost", "Budgeted", "Medium Cost",
"Flagship"],index=["Low Cost", "Budgeted", "Medium Cost", "Flagship"])
# plot a heatmap
fig, ax = plt.subplots(figsize=(10,8))
```

```
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
plt.title("Confusion Matrix for Random Forest")
plt.show()
#Calculate Accuracy
acc_rf = accuracy_score(y_valid, y_pred_rf)
#Features Contribution
feature_imp = pd.Series(rf.feature_importances_,index=col).sort_values(ascending=False)
sns.barplot(x=feature_imp, y=feature_imp.index,palette = "bright")
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features for Random Forest")
from sklearn.neighbors import KNeighborsClassifier
# Finding optimal KNN Grid Search Method
from sklearn.model_selection import GridSearchCV
# parameters = {'n_neighbors':np.arange(1,50)}
# knn = KNeighborsClassifier()
# model = GridSearchCV(knn, parameters, cv=5)
# model.fit(x_train, y_train)
# print(model.best_params_)
#Printing optimal n-neighbors
knn = KNeighborsClassifier(n_neighbors=45)
knn.fit(x_train, y_train)
y_pred_knn = knn.predict(x_valid)
#Visualizing the Confusion Matrix
```

```
cm = confusion_matrix(y_valid, y_pred_knn)

# label the confusion matrix

conf_matrix=pd.DataFrame(data=cm,columns=["Low Cost", "Budgeted", "Medium Cost",
"Flagship"],index=["Low Cost", "Budgeted", "Medium Cost", "Flagship"])

# plot a heatmap

fig, ax = plt.subplots(figsize=(10,8))

sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

plt.title("Confusion Matrix for KNN")

plt.show()

#Calculate Accuracy

acc_knn = accuracy_score(y_valid, y_pred_knn)

from sklearn.svm import SVC

svm = SVC(kernel = 'linear',random_state = 0)

svm.fit(x_train,y_train)

y_pred_svm = svm.predict(x_valid)

#Visualizing the Confusion Matrix

cm = confusion_matrix(y_valid, y_pred_svm)

# label the confusion matrix

conf_matrix=pd.DataFrame(data=cm,columns=["Low Cost", "Budgeted", "Medium Cost",
"Flagship"],index=["Low Cost", "Budgeted", "Medium Cost", "Flagship"])

# plot a heatmap

fig, ax = plt.subplots(figsize=(10,8))

sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

plt.title("Confusion Matrix for SVM")

plt.show()
```

```
#Calculate Accuracy

acc_svm = accuracy_score(y_valid, y_pred_svm)

#Features Contribution

feature_imp = pd.Series(abs(svm.coef_[0]), index=col).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index,palette = "bright")

plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

plt.title("Visualizing Important Features for SVM")

#------------>Accuracy Comparison<------------

models = ['LR','DTG','RF','KNN','SVM']

acc_scores = [acc_lr,acc_dtg,acc_rf,acc_knn,acc_svm]

print("Models\tAccuracy\n")

for i in range(5):

    print(str(models[i]) + "\t" + str(acc_scores[i]),end='\n')

#------------>Finding the Best Algorithm<----------


print("\n------------>We decide from the Bar Graph that the best Model is SVM<------------")

plt.rcParams['figure.figsize'] = (12,9)

plt.bar(models, acc_scores, color=['lightblue', 'pink', 'lightgreen',
'cyan','red','yellow','purple','orange','brown'])

plt.ylabel("accuracy scores")

plt.title("Model Comparison based on Accuracy")

plt.show()

#------------>Using the Best Algorithm<-------------

# Remove ID column(extra)
```

```
test_data=test_data.drop('id',axis=1)


#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
test = sc.fit_transform(test_data)
# Prediction
predicted_price_range = svm.predict(test)


test_data['price_range'] = predicted_price_range


labels = ["Low Cost", "Budgeted", "Medium Cost", "Flagship"]
labels = list(labels)
values= test_data['price_range'].value_counts().values
colors = ['yellow','turquoise','lightblue', 'pink']
plt.rcParams['figure.figsize'] = (12,9)
fig1, ax1 = plt.subplots()
ax1.pie(values, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.set_title('Distribution of Predicted Classes in Test Data')
plt.show()# Prediction
predicted_price_range = svm.predict(test)


test_data['price_range'] = predicted_price_range
```

```
labels = ["Low Cost", "Budgeted", "Medium Cost", "Flagship"]

labels = list(labels)

values = test_data['price_range'].value_counts().values

colors = ['yellow','turquoise','lightblue', 'pink']

plt.rcParams['figure.figsize'] = (12,9)

fig1, ax1 = plt.subplots()

ax1.pie(values, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)

ax1.set_title('Distribution of Predicted Classes in Test Data')

plt.show()

#------------>CLassification Report<------------


# Confusion Matrix

print("\n------------>Classification Report for SVM< ------------ ")

confusion = confusion_matrix(y_valid, y_pred_svm)

# For class 0

TP_0 = confusion[0,0]

TN_0 = confusion[1,1]+confusion[2,2]+confusion[3,3]

FP_0 = confusion[1,0]+confusion[2,0]+confusion[3,0]

FN_0 = confusion[0,1]+confusion[0,2]+confusion[0,3]


precision_0 = TP_0 / ( TP_0 + FP_0)

recall_0 = TP_0 / ( TP_0 + FN_0)

specificity_0 = TN_0 / (TN_0 + FP_0)

accuracy_0 = (TP_0 + TN_0) / (TP_0 + TN_0 + FP_0 + FN_0)
```

```
fscore_0 = (2 * precision_0 * recall_0) / (precision_0 + recall_0)


# For class 1

TP_1 = confusion[1,1]

TN_1 = confusion[0,0]+confusion[2,2]+confusion[3,3]

FP_1 = confusion[0,1]+confusion[2,1]+confusion[3,1]

FN_1 = confusion[1,0]+confusion[1,2]+confusion[1,3]


precision_1 = TP_1 / ( TP_1 + FP_1)

recall_1 = TP_1 / ( TP_1 + FN_1)

specificity_1 = TN_1 / (TN_1 + FP_1)

accuracy_1 = (TP_1 + TN_1) / (TP_1 + TN_1 + FP_1 + FN_1)

fscore_1 = (2 * precision_1 * recall_1) / (precision_1 + recall_1)

# For class 2

TP_2 = confusion[2,2]

TN_2 = confusion[1,1]+confusion[0,0]+confusion[3,3]

FP_2 = confusion[1,2]+confusion[0,2]+confusion[3,2]

FN_2 = confusion[2,0]+confusion[2,1]+confusion[2,3]


precision_2 = TP_2 / ( TP_2 + FP_2)

recall_2 = TP_2 / ( TP_2 + FN_2)

specificity_2 = TN_2 / (TN_2 + FP_2)

accuracy_2 = (TP_2 + TN_2) / (TP_2 + TN_2 + FP_2 + FN_2)

fscore_2 = (2 * precision_2 * recall_2) / (precision_2 + recall_2)
```

```
# For class 3

TP_3 = confusion[3,3]

TN_3 = confusion[1,1]+confusion[2,2]+confusion[0,0]

FP_3 = confusion[0,3]+confusion[2,3]+confusion[1,3]

FN_3 =confusion[3,1]+confusion[3,2]+confusion[3,0]


precision_3 = TP_3 / ( TP_3 + FP_3)

recall_3 = TP_3 / ( TP_3 + FN_3)

specificity_3 = TN_3 / (TN_3 + FP_3)

accuracy_3 = (TP_3 + TN_3) / (TP_3 + TN_3 + FP_3 + FN_3)

fscore_3 = (2 * precision_3 * recall_3) / (precision_3 + recall_3)


print("-----------------------------------------------------------------")

print("\t " " Class 0 \t Class 1 \t Class 2 \t Class 3")

print("-----------------------------------------------------------------")

print("Accuracy:" + " " + str("%.4f" % accuracy_0) + " \t " + str("%.4f" % accuracy_1) +"\t"+" \t " +
str("%.4f" % accuracy_2) +"\t"+ " \t " + str("%.4f" % accuracy_3) + "\n")

print("Precision:" + " " + str("%.4f" % precision_0) + " \t " + str("%.4f" % precision_1) +"\t"+" \t " +
str("%.4f" % precision_2) +"\t"+ " \t " + str("%.4f" % precision_3) + "\n")

print("Recall:" + "    " + str("%.4f" % recall_0) + " \t " + str("%.4f" % recall_1) +"\t"+" \t " +
str("%.4f" % recall_2) +"\t"+ " \t " + str("%.4f" % recall_3) + "\n")

print("Specificity:" +"" + str("%.4f" % specificity_0) + " \t " + str("%.4f" % specificity_1) +"\t"+" \t "
+ str("%.4f" % specificity_2) +"\t"+ " \t " + str("%.4f" % specificity_3) + "\n")

print("F-score:" + " " + str("%.4f" % fscore_0) + " \t " + str("%.4f" % fscore_1) +"\t"+" \t " +
str("%.4f" % fscore_2) +"\t"+ " \t " + str("%.4f" % fscore_3)
```

9.2 OUTPUTSCREENS

	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
5	1859	0	0.5	1	3	0	22	0.7	164	1	7	1004	1654	1067	17	1	10
6	1821	0	1.7	0	4	1	10	0.8	139	8	10	381	1018	3220	13	8	18
7	1954	0	0.5	1	0	0	24	0.8	187	4	0	512	1149	700	16	3	5
8	1445	1	0.5	0	0	0	53	0.7	174	7	14	386	836	1099	17	1	20
9	509	1	0.6	1	2	1	9	0.1	93	5	15	1137	1224	513	19	10	12
10	769	1	2.9	1	0	0	9	0.1	182	5	1	248	874	3946	5	2	7
11	1520	1	2.2	0	5	1	33	0.5	177	8	18	151	1005	3826	14	9	13
12	1815	0	2.8	0	2	0	33	0.6	159	4	17	607	748	1482	18	0	2
13	803	1	2.1	0	7	0	17	1.0	198	4	11	344	1440	2680	7	1	4
14	1866	0	0.5	0	13	1	52	0.7	185	1	17	356	563	373	14	9	3
15	775	0	1.0	0	3	0	46	0.7	159	2	16	862	1864	568	17	15	11
16	838	0	0.5	0	1	1	13	0.1	196	8	4	984	1850	3554	10	9	19
17	595	0	0.9	1	7	1	23	0.1	121	3	17	441	810	3752	10	2	18
18	1131	1	0.5	1	11	0	49	0.6	101	5	18	658	878	1835	19	13	16
19	682	1	0.5	0	4	0	19	1.0	121	4	11	902	1064	2337	11	1	18

FIG9.2.1: DATASET

```
Out[4]: 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
```

FIG9.2.2: CHECKING NULL VALUES

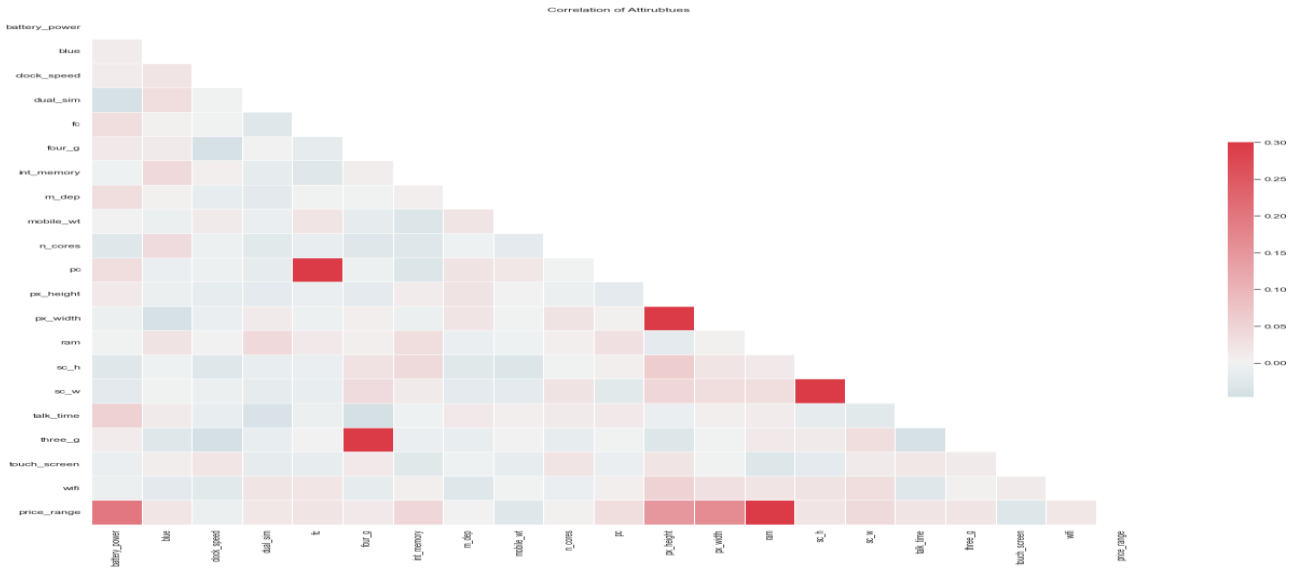


FIG9.2.3 CORRELATION OF ATTRIBUTES

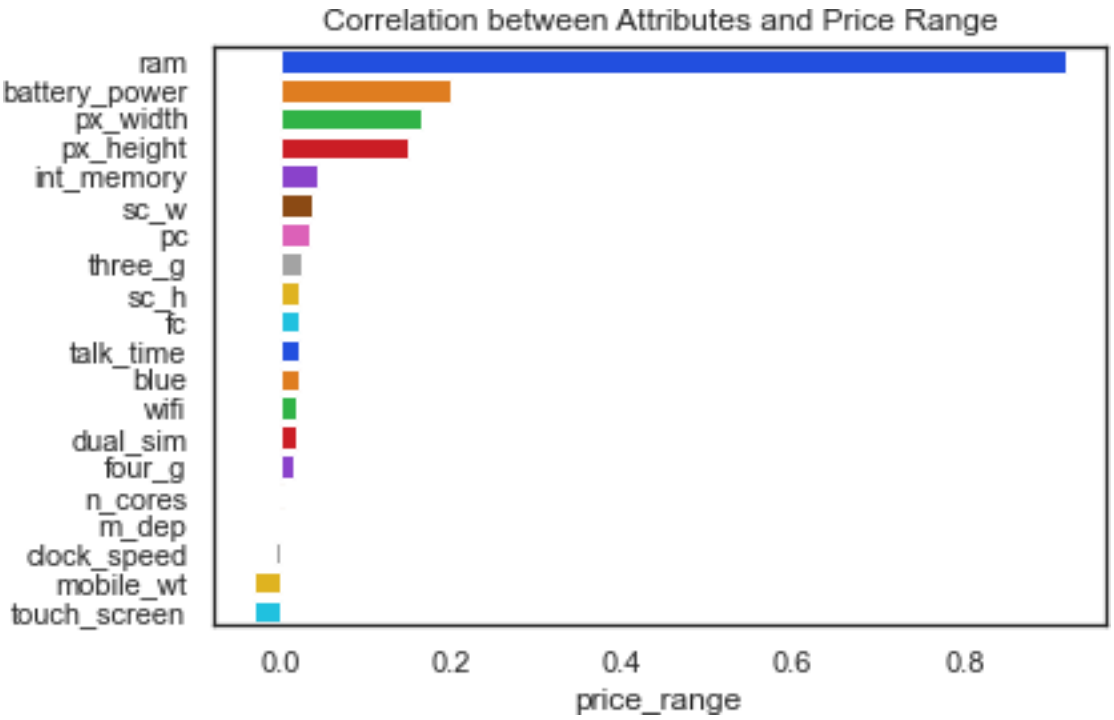


FIG9.2.4 CORRELATION ALONG WITH PRICE

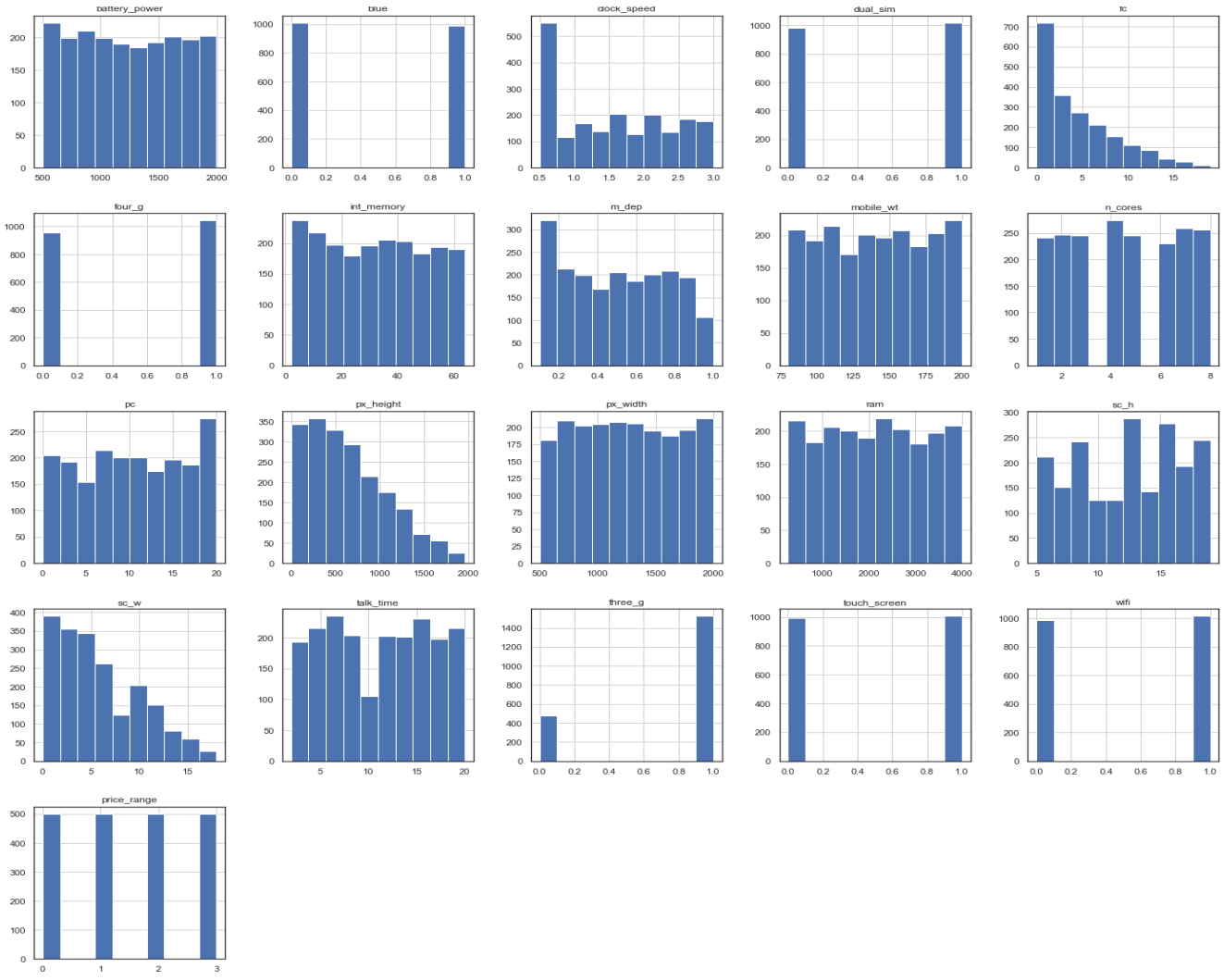


FIG9.2.5 ANALYSIS OF ATTRIBUTES

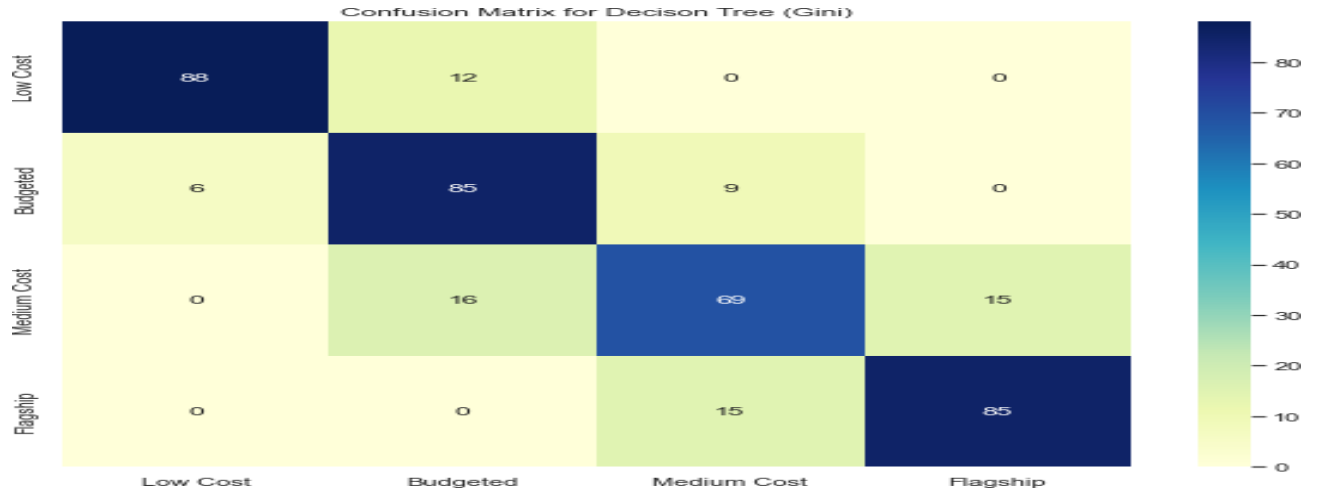


FIG9.2.6 DECISION TREE MATRIX

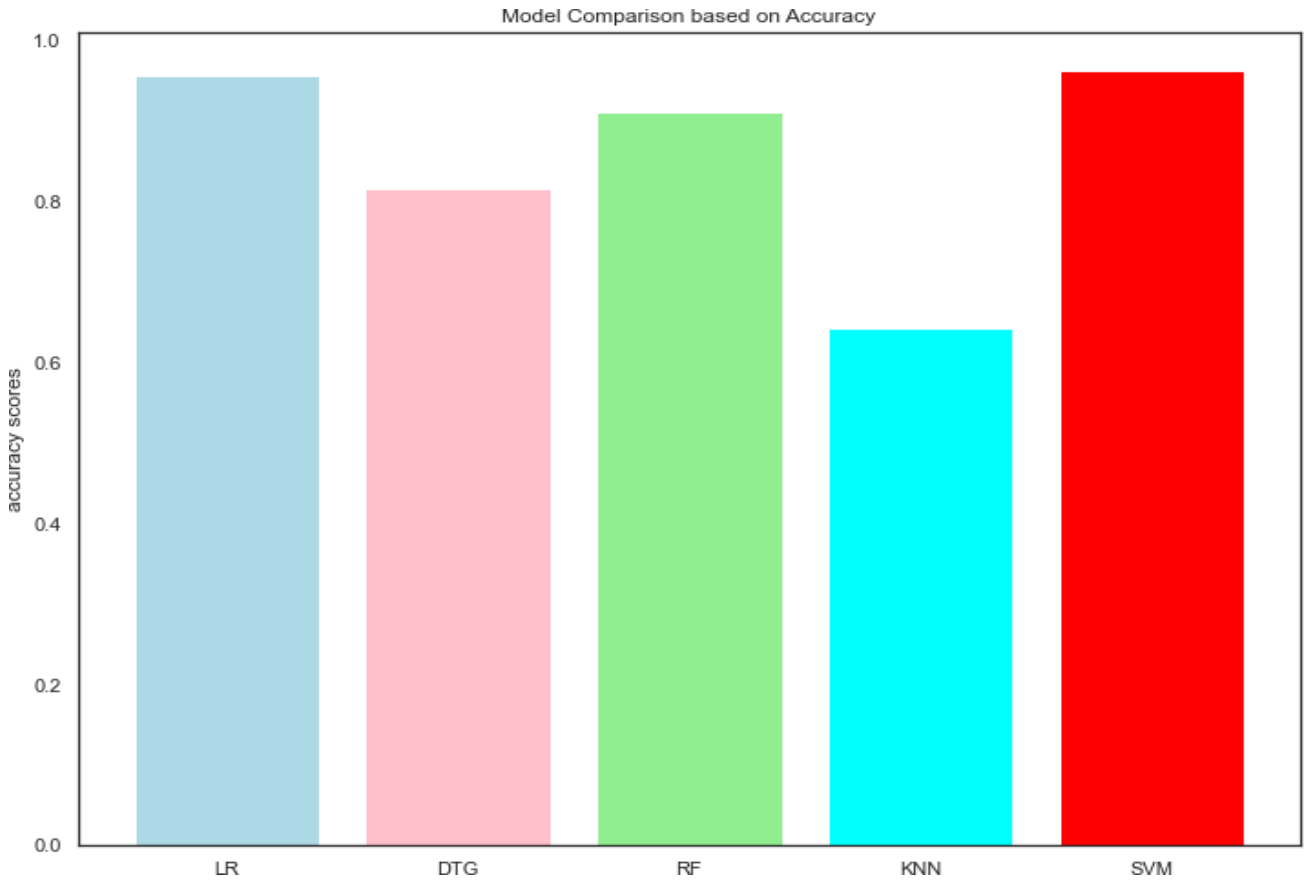


FIG9.2.7 FINDING BEST ALGORITHM

```
------------>Classification Report for SVM<------------
-----------------------------------------------------------------
           Class 0       Class 1         Class 2         Class 3
-----------------------------------------------------------------
Accuracy:  0.9821        0.9722          0.9796          0.9897

Precision: 0.9515        0.9684          0.9510          0.9800

Recall:    0.9800        0.9200          0.9700          0.9800

Specificity:0.9829       0.9899          0.9829          0.9931

F-score:   0.9655        0.9436          0.9604          0.9800
-----------------------------------------------------------------
```

FIG 9.2.8 RESULT

10. CONCLUSION

I believe the trial has shown conclusively that it is both possible and desirable to use Python as the principal teaching language, It is Free (as in both cost and source code)..It is a flexible tool that allows both the teaching of traditional procedural programming and modern OOP; It can be used to teach a large number of transferable skills. It is a real-world programming language that can be and is used in academia and the commercial world. It appears to be quicker to learn and, in combination with its many libraries, this offers the possibility of more rapid student development allowing the course to be made more challenging and varied and most importantly, its clean syntax offers increased understanding and enjoyment for students.

11.REFERENCES

- https://www.kaggle.com/code/vikramb/mobile-price-prediction/data/
- https://www.pythonanywhere.com/
- https://www.python.org/
- https://docs.anaconda.com/anaconda/install/windows/
- https://www.geeksforgeeks.org/machine-learning/