

TABLE OF CONTENTS

	Description	Page No.
	Acknowledgement	i
	Declaration	ii
	Internship Certificate	iii
	Abstract	iv
	Table of Contents	v-vi
	List of Tables	vi-vii
	List of Figures	viii

Chapter No.	Chapter Name	Page No.
1	About the company	1-2
	1.1. About Company	1
	1.2. Vision and Mission	1
	1.3. Major Milestones	1
	1.4. Organization Structure	2
	1.5. Service Offered	2
2	Introduction to Python	3-10
	2.1. Hello World	3
	2.2. Indentation	3
	2.3. Variables and Types	3
	2.4. Basic Operators	6
	2.5. Conditional Expressions	7
	2.6. Loops	7
	2.7. Functions	8
	2.8. Recursion	9
	2.9. Classes and Objects	9
3	Advanced Python	11-29
	3.1. NumPy Array	11
	3.2. Pandas in Python	12
	3.3. Matplotlib in python	13
	3.4. Seaborn Library	15
	3.5. Sklearn Library	18
	3.6. Generators	26
	3.7 Lambda Function	26
	3.8. Exception Handling	26
	3.9. Decorators	28

	3.10. Map	28
	3.11. Filter	28
	3.12. Reduce	29
4	Introduction to Project	30-40
	4.1. Introduction to Project	30
	4.2. System Analysis	31
	4.3. Data Collection	32
	4.4. Data Cleaning and Preprocessing	33
	4.5. Exploratory Data Analysis	33
	4.6. Data Visualization Techniques	37
	4.7. Methodology	40
	4.8. Result	
5	Reflection Notes	41
6	Conclusion	42
7	References	43

LIST OF TABLES

Table No.	Table Name	Page No.
2.1	Program to print Hello world!!	3
2.2	Indentation Program	3
2.3	Numbers Example program	3
2.4	String Example program	4
2.5	String Function	4
2.6	List Methods	5
2.7	Tuple Methods	5
2.8	Dictionary methods	6
2.9	Operation on Sets	6
2.10	If else and elif Example Program	7
2.11	Break and continue Example program	8
3.1	Numpy basic problem	11
3.2	0-D Array	11
3.3	1-D Array	11
3.4	2-D array	12

3.5	3-D array	12
3.6	Pandas basic program	12
3.7	Read CSV file	13
3.8	Line plot Example program	13
3.9	Bar plot Example program	14
3.10	Histograms plot Example program	14
3.11	Scatter plot Example program	14
3.12	Pie Chart Example Program	15
3.13	Area Plot Example Program	15
3.14	Histplot Example Program	16
3.15	Distplot Example Program	17
3.16	Lineplot Example Program	17
3.17	Lmplot Example Program	18
3.18	Heatmap Example Program	18
3.19	Linear Regression Algorithm	21
3.20	Logistic Regression Algorithm	21
3.21	Random Forest Algorithm	22
3.22	Decision Tree Algorithm	23
3.23	Support Vector Classification	24
3.24	K Nearest Neighbors	25
3.25	Generators	26
3.26	Lambda Function	26
3.27	Exception Handling	27
3.28	Try with else Clause	27
3.29	Finally Keyword in python	27
3.30	Decorators	28
3.31	Mapping Function	28
3.32	Filter Function	30
3.33	Reduce Function	30

LIST OF FIGURES

Figure No	Figure Name	Page No
3.1	Line plot	13
3.2	Bar Plot	14
3.3	Histogram	14
3.4	Scatter Plot	14
3.5	Pie Chart	15
3.6	Area Plot	15
3.7	Histplot	16
3.8	Distplot	17
3.9	Seaborn Lineplot	17
3.10	Lmplot	18
3.11	Heatmap	18
4.1	Checking for null value	33
4.2	Dataset information	33
4.3	Dataset Description	34
4.4	Analysis of Dependent	34
4.5	Analysis of Job on Deposit	35
4.6	Analysis of Marital on Deposit	35
4.7	Analysis of Education on Deposit	36
4.8	Analysis of Housing on Deposi	36
4.9	Correlation Plot	37
4.10	Model Building	38
4.11	Decision Tree	39

CHAPTER 1

ABOUT THE COMPANY

1.1. About the Company

1.1.1. Formation of company

Aqmenz Automation Private Limited is a private incorporated on 15th October 2018. It is classified as Non-Govt company and is registered at Registrar of companies, Bangalore.

1.1.2. Brief history of company

Aqmenz Automation Pvt Ltd (AAPL) is situated in northern part of Bangalore, RT Nagar, Karnataka. AAPL provides Mechanical Design & Automation solutions to their client companies. AAPL also involved in Open source Robotics and developed different varieties of Robots.

AAPL also started INDOSKILL, a separate platform for the students to get training and work on various Real Time Industrial Projects. Indoskill offers skill-oriented hands-on training.

Field of Expertise: Open-source Robotics, Industrial Automation, Product Design, Python & Deep Learning and Embedded Systems

1.1.3. Objectives

- AAPL had a trust in Skill India mission & vision, hence our utmost priority is to add skill to the young Generation and make them Profitable and productive for the nation.
- We aim in Providing Industrial Automation Training Skill module kits to Institution, University's & Collage Lab Facilities with Lowest Possible Price for the Benefits of Technical Students.
- Identifying young entrepreneurs and Motivate, training them to establish Startup to create Employment as well as prosperity for the nation.
- Consultation, Sourcing and supplying highly skilled Manpower to Industry for better efficiency and productivity.
- Providing low cost & precise industrial automation solutions. Very eager to fetch solution for most complex industrial problems in a modest way.

1.2. Vision and Mission

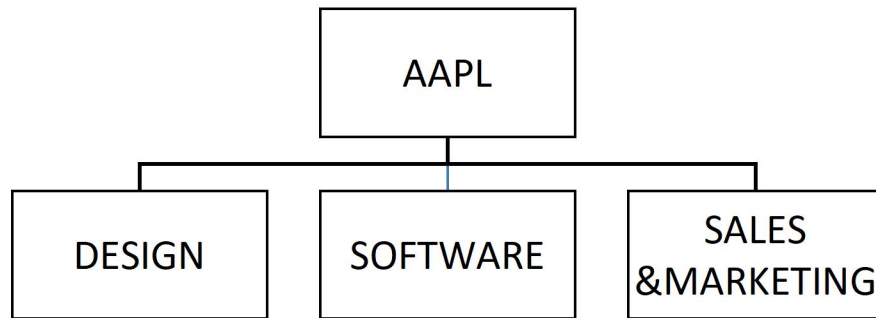
Our Motto and Vision are to create awareness & training young generation to current and future jobs demands and also help to current and future jobs demands; meanwhile help the students and employees to meet the mandatory necessities of future human resources and skill demands. We are in the 4th industrial revolution. The technological revolution is catastrophic like never before, hence continues awareness for the up-gradation environment is much essential. Aqmenz Automation Pvt. Ltd. is working to help and enhance the potential of students and employees.

1.3. Major Milestones

We have under gone many industrial projects. Our major clients are BIAL (Bangalore International Airport Limited), GE (General Electric) and Amics technologies.

1.4. Organization Structure

The organization structure is having three different departments such as design department, software department and sales and marketing.



1.5. Service Offered

- Provides Design & Automation solutions.
- All type of automation projects to companies using PLC's, SCADA embedded systems.
- We provide robots and robotic solutions to small and medium scale companies
- Embedded solutions to companies like GE
- We conduct technical skill oriented training programs to engineering colleges.
- We also provide robotics and automation lab equipments for colleges.

Number of people working in company and their responsibilities:

There are 20 persons in this company, out of which:

- Shamanna Mohan, Chief Executive Officer (CEO)
- Mohammed Azhar Hussain, Chief Technology Officer (CTO)

CHAPTER 2

INTRODUCTION TO PYTHON

2.1. Hello World!!

Python is a programming language, with a simple syntax. It encourages programmers to write code without boilerplate. One of the aspects of Python is the "print" directive, which allows you to display information on the console.

There are two versions of Python Python 2 and Python 3. These versions have differences between them. In this tutorial we will be using Python 3 as it is semantically correct and supports features.

One key distinction between Python 2 and 3 lies, in the usage of the "print" statement. In Python 2 this statement does not require parentheses when invoked. On the hand in Python 3 it is considered a function. Therefore must be called with parentheses. For example:

Program.py	Output
<code>print("Hello World!!")</code>	Hello World!!

Table 2.1: Program to print Hello world!!

2.2. Indentation

In Python indentation is used to denote blocks of code of using braces. Both tabs and spaces are accepted,. It is customary, in Python programming to use four spaces, for indentation. For example:

Program.py	Output
<code>X = 1 if X == 1: print("X is 1.")</code>	X is 1

Table 2.2: Indentation Program

2.3. Variables and Types

Python follows an object oriented approach. Does not have "static typing". You don't have to declare variables or their types before using them in Python. In Python every variable is treated as an object.

Rules for defining variable name:-

A variable name can contain alphabets (A-Z, a-z), digits (0-9), underscores.

A variable name can only start with an alphabets and underscore.

A variable can't start with a digit.

No white space is allowed to be used inside a variable name.

Example: nishant, one8, _seven etc

2.3.1. Numbers

Python supports two types of numbers - integers(whole numbers) and floating point numbers(decimals).

Program.py	Output
<code>#integers(whole numer) Myint = 8 print("Integer is:", Myint)</code>	Integer is: 8 Decimal number is: 8.0 Decimal number is: 8.0

Table 2.3: Numbers Example program

2.3.2. Strings

String is a data type in python. String is a sequence of characters enclosed in quotes. We can write string in these two ways :

Single quoted string - a = 'Nishant'

Double quoted string - b = "Nishant"

The difference between the two is that using double quotes makes it easy to include apostrophes.

Program.py	Output
mystring = "Don't worry about apostrophes" print(mystring)	Don't worry about apostrophes

Table 2.4: String Example program

2.3.2.1.String Slicing

A string in python can be sliced for getting a part of the string.

N	I	S	H	A	N	T	
0	1	2	3	4	5	6	Length = 7

The index in a string starts from 0 to (length-1) in python. In order to slice a string, we use the following syntax:

Sl = name[int_start : int_end]

Where, int_start - First index included ; int_end - Last index is not included

Slicing with skip value : we can provide a skip value as a part of our slice like this:

Word = "amazing"

Word[1:6:2] = 'mzn'

Other advanced slicing techniques

word = "amazing"

Word[:7] = word [0:7] = word[0:] = 'amazing'

2.3.2.2. String functions

Function	Description
len()	Returns the length of the string
endswith()	Tells whether the variable string ends with the string or not.
count()	Counts the total number of occurrence of any character
capitalize()	This function capitalizes the first character of any string.
find()	Finds a word and returns the index of first occurrence of that word in the string
replace()	Replaces the old word with new word in the entire string

Table 2.5: String Function

2.3.2.3. Escape sequence characters

Comprises of more than one character but represents one character when used within the string.

Examples : \n (new line), \t (tab), \' (single quote), \\ (backslash), etc

type() function and typecasting

type function is used to find the data type of a given variable in python.

```
Example : a = 31
          type(a)    => class<int>
          b = "31"
          type(b)    => class<str>
```

A number can be converted into a string and vice versa. There are many functions to convert data type into another.

```
str(31) = "31"    => Integer to string conversion
```

```
int("32") = 32    => String to integer conversion
```

```
float(32) = 32.0  => Integer to float conversion
```

input() function

This function allows the user to take input from the keyboard as string.

2.3.3. Lists

Python lists are containers to store a set of values of any data type.

```
Li = ['apple', 'arjun', 'medini', 7, False]
```

2.3.3.1. List Methods

Methods	Description
Sort()	Sort the list
reverse()	Reverses the order of the list
append()	Adds an element at the end of the list
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value

Table 2.6: List Methods

2.3.4. Tuple

A Tuple is an immutable (cannot change) data type in python.

```
a = ()    => Empty tuple
```

```
a = (1,)   => Tuple with only one element needs a comma
```

```
a = (1, 7, 2) => Tuple with more than one element
```

Once defined tuple element can't be altered or modified

2.3.4.1. Tuple Methods

Methods	Description
count()	Will return number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Table 2.7: Tuple Methods

2.3.5. Dictionary

Is a collection of key-value pairs.

```
Dict = {"key": "value", "branch": "ise", "list": [1, 2, 9]}
```

2.3.5.1. Properties

It is unordered.

It is mutable.

It is indexed.

Cannot contain duplicate keys.

2.3.5.2. Dictionary methods

Methods	Description
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
update()	Updates the dictionary with the specified key-value pairs
get()	Returns the value of the specified key

Table 2.8: Dictionary methods

2.3.6. Sets

Sets is collection of non repetitive elements.

2.3.6.1. Properties

Sets are unordered.

Sets are unindexed.

There is no way to change items in sets.

Sets cannot duplicate values.

2.3.6.2. Operation on Sets

Operation	Description
len()	Returns the length of the set
remove()	Update the set and removes a specified element from set
Pop()	Removes an arbitrary element from the set and returns the element removed
clear()	Empties the set
intersection()	Returns a set which contains only items in both sets
union()	Returns new set with all items from both sets

Table 2.9: Operation on Sets

2.4. Basic operators

- **Arithmetic Operator** is a symbol that instructs a computer to carry out a calculation. Example: +, -, *, / etc
- **Assignment Operator** is a symbol used to assign a value to a variable. Example: =, +=, -=, *=, /= etc
- **Comparison Operator** utilizes symbols to assess the relationship, between two values.
Example =, >, <, >=, != etc
- **Logical Operators** are symbols that we use to combine two or more expressions and determine if they are true or false. Example:
 - and => true if both are true else false.
 - or => true if at least one operand is true else false.
 - not => inverts true to false and false to true
- **The “in” operator** could be used to check if specified object exists within an iterable object container.

Example:

```
name = "nishant"
if name in ["nishant", "arjun"]:
    print("Your name is either nishant or arjun.")
```

- **The “is” operator** verifies if two objects point to the object, in memory than merely checking if they hold the same value.

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x == y) #prints out True
print(x is y) #prints out False
```

2.5. Conditional Expressions

2.5.1. If else and elif

if else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax	Example
<pre>if (condition1): print("yes") elif (conditon2): print("no") else: print("maybe")</pre>	<pre>a = 22 if (a>9): print("Greater") else: print("lesser")</pre>

Table 2.10: If else and elif Example Program

Note: There can be any number of elif statements. Last else is executed only if all the conditions inside elifs fail.

2.6. Loops

Loops are a fundamental concept in programming and are essential for automating repetitive tasks. Python offers two main types of loops:

2.6.1. For loop

Used for iterating over a sequence of elements, such as a list, tuple, string, or range of numbers.

Syntax : for item in sequence: #bock of code to be executed for each item.

Example:

```
Fruits = ['orange', 'banana', 'apple']
for fruit in fruits:
    Print(fruit)
```

2.6.2. While loop

Used for iterating as long as a condition remains true.

Syntax: while condition: #block of code to be executed the condition is true.

Example:

```
i = 0
While i < 5:
    print(i)
    i +=1
```

2.6.3. break and continue statements

break is used to exit a for loop or while loop, whereas continue is used to skip the current block, and return to the for or while statement.

Break_ex.py	Output
<pre>count = 0 while True: print(count) count += 1 if count >= 5: break</pre>	0, 1, 2, 3, 4
Continue_ex.py	Output
<pre>for x in range(10): # Check if x is even if x % 2 == 0: continue print(x)</pre>	1, 3, 5, 7, 9

Table 2.11: Break and continue Example program

2.7. Functions

A function is a group of statements performing a specific task. When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of times

Syntax:

```
def functionName():
    print("Hello")
```

This function can be called any number of times, anywhere in the program.

2.7.1. Function call

Whenever we want to call a function, we put the name of the function followed by parenthesis as follows:

function1() -> this is called function call.

2.7.2. Function definition

The part containing the exact set of instruction which are executed during the function call.

2.7.3. Types of Function

- **Built in functions** - Already present in python. Example of built in function includes len(), print(), range() etc.
- **User defined function** - Defined by user. The function1() function we defined is an example of user defined function

2.7.4. Function with arguments

A function can accept some values it can work with. We can put these values in the parenthesis. A function can also return values as shown below:

```
def greet(name):
```

```
    gr = "hello"+name
```

```
    return gr
```

```
a = greet("Nishant")          #Nishant is passed to greet in name. a will contain "hello Nishant"
```

2.7.5. Default parameter value

We can have a value as default argument in a function. If we specify name= 'stranger' in the line containing def , this value is used when no argument is passed.

Example:

```
def greet(name = "stranger"):
    #function body

greet() - name will be "stranger" in function body (default).
greet("Nishant") - name will be "Nishant" in function body (passed).
```

2.8. Recursion

Recursion is a function which calls itself. It is used to directly use a mathematical formula as a function.

Example:

```
def factorial(n):
    if i == 0 or i == 1:          #base condition which doesn't call the function any further.
        Return 1
    else:
        n*factorial(n-1)        #function calling itself.
```

2.9. Classes and Object

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Example:

```
class MyClass:
    variable = "blah"
    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
```

The variable "myobjectx" holds an object of the class "MyClass" that contains the variable and the function defined within the class called "MyClass".

2.9.1. Accessing object variables

To access the variable inside of the newly created object "myobjectx" you would do the following:

Example:

```
class MyClass:
    variable = "blah"
    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
myobjectx.variable
```

2.9.2. Accessing object functions

To access a function inside of an object you use notation similar to accessing a variable:

Example:

```
class MyClass:
    variable = "blah"
    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
myobjectx.function()
```

2.9.3. init()

The `__init__()` function, is a special function that is called when the class is being initiated. It is used for assigning values in a class.

CHAPTER 3

ADVANCED PYTHON

3.1. NumPy Array

A Python package called NumPy is used to work with arrays. It also includes functions for working with matrices, the Fourier transform, and linear algebra. In the year 2005, Travis Oliphant founded NumPy. You are free to use it as it is an open source project. Numerical Python is referred to as NumPy.

Lists can be used in place of arrays in Python, although they process data slowly. With NumPy, an array object can be created up to 50 times faster than with conventional Python lists. The ndarray array object in NumPy comes with a number of helpful functions that make working with it a breeze. In data research, where speed and resources are crucial, arrays are employed a lot.

3.1.1. Installation of NumPy

C:\Users\Your Name>pip install numpy

Use a Python distribution like Anaconda, Spyder, or another one that already has NumPy installed if this command doesn't work.

3.1.2. Importing NumPy

Use the import keyword to import NumPy into your apps after it has been installed.

```
import numpy as np
```

Program.py	Output
<pre>import numpy as np arr = np.array([1, 2, 3, 4, 5]) print(arr) print(type(arr))</pre>	<pre>[1 2 3 4 5] <class 'numpy.ndarray'></pre>

Table 3.1: Numpy basic problem

3.1.3. Dimensions in arrays

1) 0-D Array

0-D arrays or Scalars, are the elements in an array, each value in array is a 0-D array

Program.py	Output
<pre>import numpy as np arr = np.array(42) print(arr) print("Dimension:", arr.ndim)</pre>	<pre>42 Dimension: 0</pre>

Table 3.2: 0-D Array

2) 1-D Array

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

Program.py	Output
<pre>import numpy as np arr = np.array([1, 2, 3, 4, 5]) print(arr) print("Dimension:", arr.ndim)</pre>	<pre>[1 2 3 4 5] Dimension: 1</pre>

Table 3.3: 1-D Array

3) 2-D Array

An array that has 1-D arrays as its elements is called a 2-D array.

Program.py	Output
<pre>import numpy as np arr = np.array([1, 2, 3], [4, 5, 6]) print(arr) print("Dimension:", arr.ndim)</pre>	<pre>[[1 2 3] [4 5 6]] Dimension: 2</pre>

Table 3.4: 2-D array

4) 3-D Array

An array that has 2-D arrays as its elements is called a 3-D array.

Program.py	Output
<pre>import numpy as np arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) print(arr) print("Dimension:", arr.ndim)</pre>	<pre>[[[1 2 3] [4 5 6]] [[1 2 3] [4 5 6]]] Dimension: 3</pre>

Table 3.5: 3-D array

3.2. Pandas in Python

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008. Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

3.2.1. Installation of Pandas

C:\Users\Your Name>pip install pandas

Use a Python distribution like Anaconda, Spyder, or another one that already has NumPy installed if this command doesn't work

3.2.2. Importing Pandas

Import pandas as pd

Program.py	Output
<pre>import pandas as pd data = { "calories": [420, 380, 390], "duration": [50, 40, 45] } #load data into a DataFrame object: df = pd.DataFrame(data) print(df)</pre>	<pre> calories duration 0 420 50 1 380 40 2 390 45</pre>

Table 3.6: Pandas basic program

3.2.3. Read CSV file

A simple way to store big data sets is to use CSV files (comma separated files). CSV files contains plain text and is a well know format that can be read by everyone including Pandas. In our examples we will be using a CSV file called 'data.csv'.

Program.py	Output
import pandas as pd df = pd.read_csv('data.csv') print(df.head(2)) print(df.shape)	Duration Pulse Maxpulse Calories 0 60 110 130 409.1 1 60 117 145 479.0 (169, 4)

Table 3.7: Read CSV file

3.3. Matplotlib in python

Matplotlib is a fantastic Python visualization package for two-dimensional array charts. Based on NumPy arrays, Matplotlib is a multi-platform data visualization package intended to be used with the larger SciPy stack. In the year 2002, John Hunter made its debut. The ability to visually access vast volumes of data in a format that is simple to understand is one of visualization's biggest advantages. Many plot types, including line, bar, scatter, histogram, and more, are available in Matplotlib.

3.3.1. Installation of Matplotlib

C:\Users\Your Name>pip install matplotlib

3.3.2. Importing Matplotlib

import matplotlib as plt

3.3.3. Types of Matplotlib

Many different plots are included with Matplotlib. Plots are useful for understanding patterns, trends, and correlations. Usually, they are tools for making sense of quantitative data. This covers a few of the sample graphs.

3.3.3.1. Matplotlib Line Plot

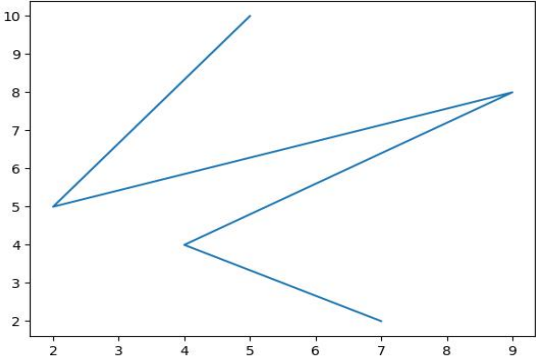
Program.py	Output
from matplotlib import pyplot as plt x = [5, 2, 9, 4, 7] y = [10, 5, 8, 4, 2] plt.plot(x, y) plt.show()	 <p>Figure 3.1: Line plot</p>

Table 3.8: Line plot Example program

3.3.3.2. Matplotlib Bar Plot

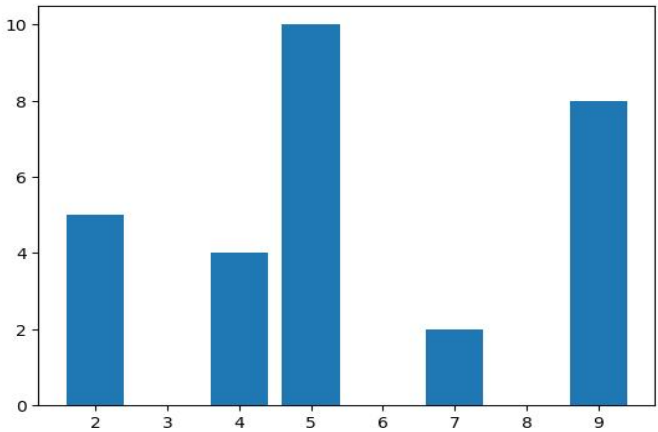
Program.py	Output
<pre>from matplotlib import pyplot as plt x = [5, 2, 9, 4, 7] y = [10, 5, 8, 4, 2] plt.bar(x, y) plt.show()</pre>	 <p>Figure 3.2: Bar Plot</p>

Table 3.9: Bar plot Example program

3.3.3.3. Matplotlib Histograms Plot

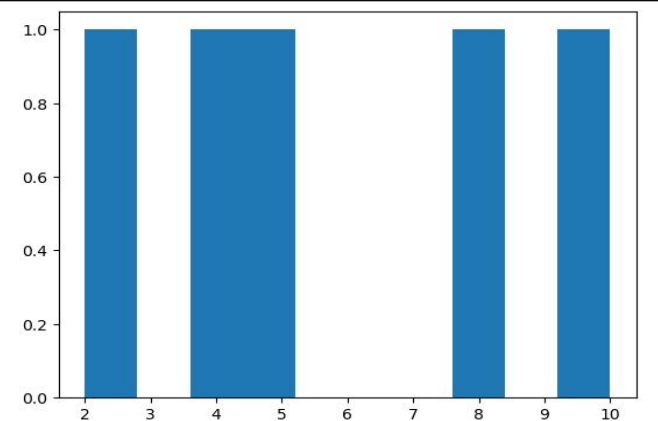
Program.py	Output
<pre>from matplotlib import pyplot as plt y = [10, 5, 8, 4, 2] plt.hist(y) plt.show()</pre>	 <p>Figure 3.3: Histogram</p>

Table 3.10: Histograms plot Example program

3.3.3.4. Matplotlib Scatter Plot

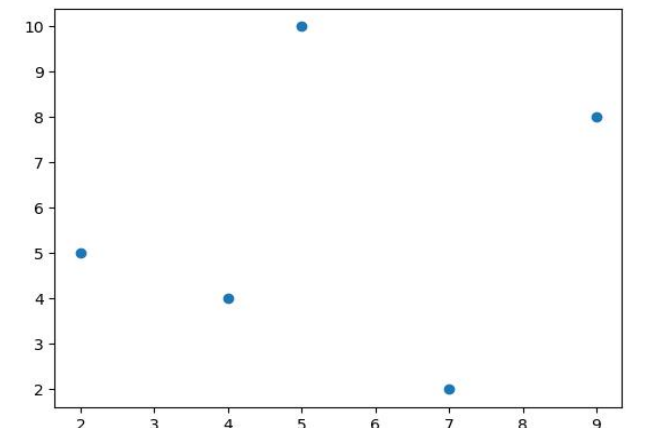
Program.py	Output
<pre>from matplotlib import pyplot as plt x = [5, 2, 9, 4, 7] y = [10, 5, 8, 4, 2] plt.scatter(x, y) plt.show()</pre>	 <p>Figure 3.4: Scatter Plot</p>

Table 3.11: Scatter plot Example program

3.3.3.5. Matplotlib Pie Charts

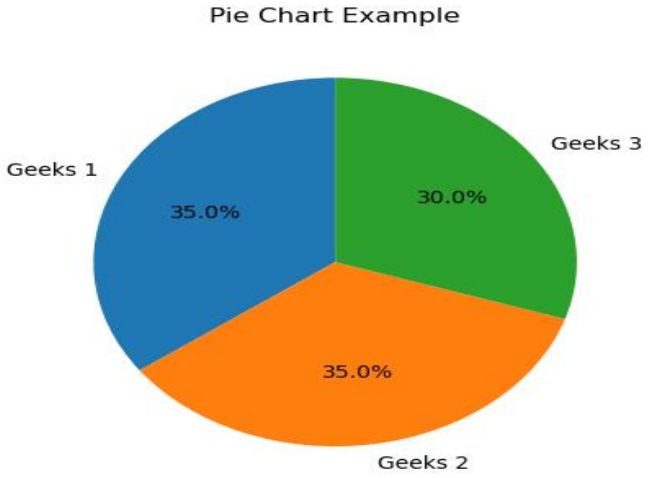
Program.py	Output
<pre>import matplotlib.pyplot as plt labels = ['Geeks 1', 'Geeks 2', 'Geeks 3'] sizes = [35, 35, 30] plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90) plt.title('Pie Chart Example') plt.show()</pre>	 <p>The pie chart is titled "Pie Chart Example". It is divided into three segments: a blue segment labeled "Geeks 1" with 35.0%, an orange segment labeled "Geeks 2" with 35.0%, and a green segment labeled "Geeks 3" with 30.0%. The chart is centered on the page.</p> <p>Figure 3.5: Pie Chart</p>

Table 3.12: Pie Chart Example Program

3.3.3.6. Matplotlib Area plot

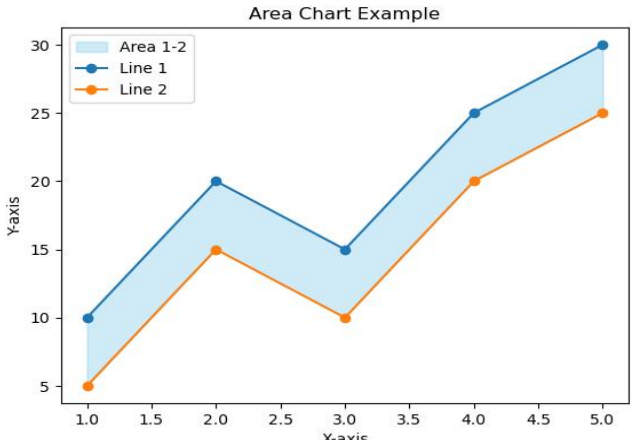
Program.py	Output
<pre>import matplotlib.pyplot as plt x = [1, 2, 3, 4, 5] y1, y2 = [10, 20, 15, 25, 30], [5, 15, 10, 20, 25] plt.fill_between(x,y1, y2, color='skyblue', alpha=0.4, label='Area 1-2') plt.plot(x, y1, label='Line 1', marker='o') plt.plot(x, y2, label='Line 2', marker='o') plt.xlabel('X-axis') plt.ylabel('Y-axis') plt.title('Area Chart Example') plt.legend(), plt.show()</pre>	 <p>The area chart is titled "Area Chart Example". It features two lines: "Line 1" (blue) and "Line 2" (orange). The area between the two lines is filled with a light blue color and labeled "Area 1-2". The x-axis is labeled "X-axis" and ranges from 1.0 to 5.0. The y-axis is labeled "Y-axis" and ranges from 5 to 30. The legend is located in the top-left corner of the plot area.</p> <p>Figure 3.6: Area Plot</p>

Table 3.13: Area Plot Example Program

3.4. Seaborn Library

A well-liked Python package for data visualization is called Seaborn. It offers a sophisticated interface for making eye-catching and educational statistical visuals. It can be used to make a variety of graphs, including heatmaps, bar plots, and scatter plots. It is based on Matplotlib, another well-liked Python plotting package. To begin using Seaborn, simply run the command "pip install seaborn" to install it using pip. After installation, you can import it into your Python script and begin utilizing its features to produce visually amazing outputs.

3.4.1. Installing Seaborn library

C:\Users\Your Name>pip install seaborn

3.4.2. Importing Seaborn Library

```
import seaborn as sns
```

3.4.3. Basic plots using seaborn

3.4.3.1. Histplot:

- **data:** This is the array, series, or data frame that you want to visualize. It is a required parameter.
- **x:** This specifies the column in the data to use for the histogram. If your data is a data frame, you can specify the column by name.
- **y:** This specifies the column in the data to use for the histogram when you want to create a bi-variate histogram. By default, it is set to **None**, meaning that a uni-variate histogram will be plotted.
- **bins:** This specifies the number of bins to use when dividing the data into intervals for plotting. By default, it is set to “auto”, which uses an algorithm to determine the optimal number of bins.
- **kde:** This parameter controls whether to display a kernel density estimate (KDE) of the data in addition to the histogram. By default, it is set to **False**, meaning that a KDE will not be plotted.

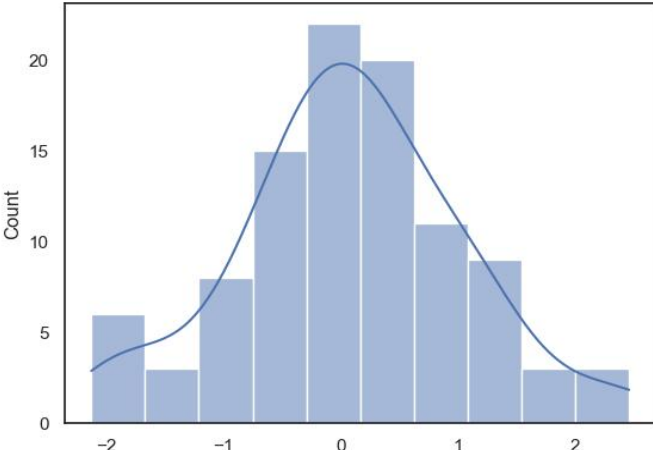
Program.py	Output
<pre>import numpy as np import seaborn as sns sns.set(style="white") rs = np.random.RandomState(10) d = rs.normal(size=100) sns.histplot(d, kde=True,)</pre>	 <p>Figure 3.7: Histplot</p>

Table 3.14: Histplot Example Program

3.4.3.2. Distplot

- a:** This is the array, series, or list of data that you want to visualize. It is a required parameter.
- bins:** This specifies the number of bins to use when dividing the data into intervals for plotting. By default, it is set to “auto”, which uses an algorithm to determine the optimal number of bins.
- kde:** This parameter controls whether to display a kernel density estimate (KDE) of the data in addition to the histogram. By default, it is set to True, meaning that a KDE will be plotted.
- hist:** This parameter controls whether to display the histogram of the data. By default, it is set to True, meaning that a histogram will be plotted.

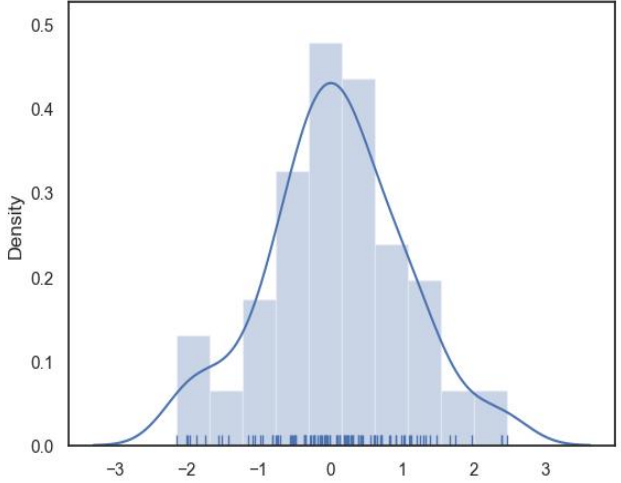
Program.py	Output
<pre>import numpy as np import seaborn as sns sns.set(style="white") rs = np.random.RandomState(10) d = rs.normal(size=100) colors = ["r", "g", "b"] sns.distplot(d, kde=True, hist=True, bins=10, rug=True, hist_kws={"alpha": 0.3})</pre>	 <p>Figure 3.8: Distplot</p>

Table 3.15: Distplot Example Program

3.4.3.3. Lineplot

One of the simplest plot types in the Seaborn Library is the line plot. The primary purpose of this plot is to display the data continuously, or as a time series.

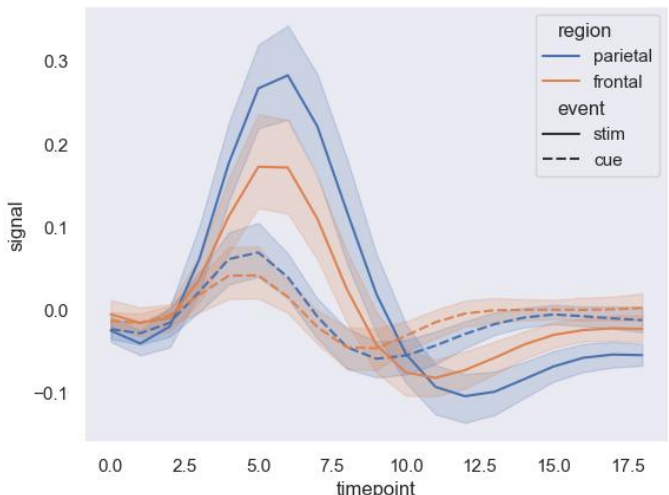
Program.py	Output
<pre>import seaborn as sns sns.set(style="dark") fmri = sns.load_dataset("fmri") sns.lineplot(x="timepoint", y="signal", hue="region", style="event", data=fmri)</pre>	 <p>Figure 3.9: Seaborn Lineplot</p>

Table 3.16: Lineplot Example Program

3.4.3.4. Lmplot

Another most basic plot is the lmplot. It displays data points on a 2D space along with a line that represents a linear regression model. The horizontal and vertical labels can be set to x and y, respectively.

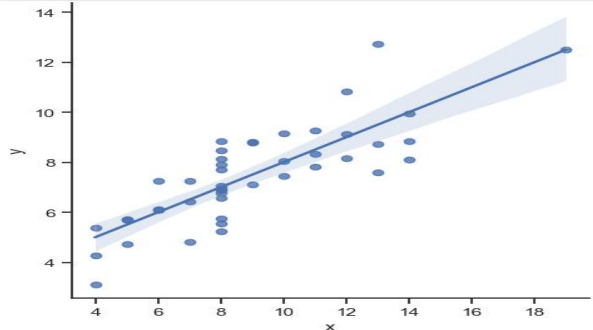
Program.py	Output
<pre>import seaborn as sns sns.set(style="ticks") df = sns.load_dataset("anscombe") sns.lmplot(x="x", y="y", data=df)</pre>	 <p>Figure 3.10: Lmplot</p>

Table 3.17: Lmplot Example Program

3.4.3.5. Heatmap

A heatmap is a two-dimensional graphical data representation that shows a variable's value using colors. A common tool for visualizing the correlation between various variables in a dataset is a heatmap.

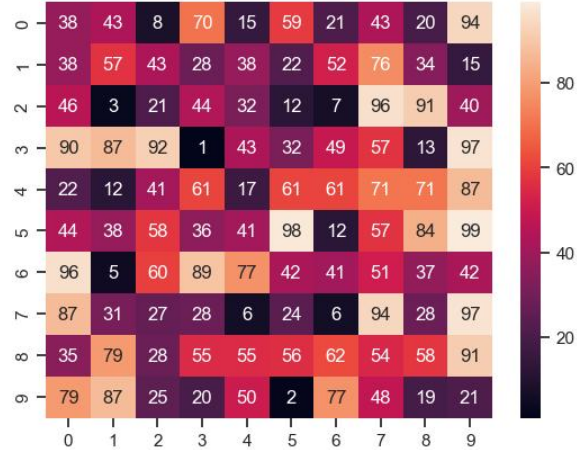
Program.py	Output
<pre>import numpy as np import seaborn as sn import matplotlib.pyplot as plt data = np.random.randint(low = 1, high = 100, size = (10, 10)) hm = sns.heatmap(data = data, annot=True) plt.show()</pre>	 <p>Figure 3.11: Heatmap</p>

Table 3.18: Heatmap Example Program

3.5. Sklearn Library

The most reliable and practical Python machine learning library is called Scikit-learn, or Sklearn. Through a Python consistency interface, it offers a range of effective tools for statistical modeling and machine learning, including as regression, clustering, classification, and dimensionality reduction. This library is based on NumPy, SciPy, and Matplotlib and is mostly developed in Python.

3.5.1. Installing sklearn library

```
C:\Users\Your Name>pip install scikit-learn
```

3.5.2. Importing sklearn

```
import sklearn
```

3.5.3. Features

Rather than focusing on loading, manipulating and summarizing data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows –

Supervised Learning algorithms – Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

Unsupervised Learning algorithms – On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

Clustering – This model is used for grouping unlabeled data.

Cross Validation – It is used to check the accuracy of supervised models on unseen data.

Dimensionality Reduction – It is used for reducing the number of attributes in data which can be further used for summarisation, visualization and feature selection.

Ensemble methods – As name suggest, it is used for combining the predictions of multiple supervised models.

Feature extraction – It is used to extract the features from data to define the attributes in image and text data.

Feature selection – It is used to identify useful attributes to create supervised models.

Open Source – It is open source library and also commercially usable under BSD license.

3.5.4. Steps to Build a Model in Sklearn

Here are the steps to build a model in Sklearn:

Step 1: Loading a Dataset

Simply put, a dataset is a collection of sample data points. A dataset typically consists of two primary parts:

Features: In our dataset, features are essentially the variables, sometimes referred to as predictors, data inputs, or attributes. Since there may be many of them, they can be represented by a feature matrix, which is often represented by the letter "X." An exhaustive list of feature names is referred to as "feature names".

Response:(also known as the output, label, or target feature) This variable is the result based on the feature of the variable. Most of the time, we just have one response column, which is represented by a response vector or column (a response vector is often indicated by the letter 'y'). All possible values for a response vector are denoted by target names.

Step 2: Splitting the Dataset

The accuracy of every machine learning model is an important factor to take into account. Now, a model may be trained using the dataset that is provided, and its accuracy can be determined by using the model to predict the target values for a different dataset.

Step 3: Training the Model

Now that the model has been trained, it is time to use the training dataset to generate predictions. Scikit-learn provides a range of machine learning techniques with an intuitive interface for fitting, prediction accuracy, and other features.

The `predict()` model class method, which returns the predicted values, can be used for this.

It is now the predicted values, can be used for this.

With the use of sklearn algorithms, we can evaluate the model's performance by comparing the predicted values with the actual values of the testing dataset. This is done using the metrics package's `accuracy_score` function.

3.5.5. Linear Regression Algorithm

The predicted result of the supervised machine learning procedure known as linear regression is the slope of a straight line. It can only be applied to values that fall inside a given range of data points.

Program.py

```
# Importing the required modules and classes
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_iris
# Loading our load_iris dataset
X, Y = load_iris(return_X_y=True)
# Printing the shape of the complete dataset
print(X.shape)
# Splitting the dataset into the training and validating datasets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.4, random_state = 10)
# Printing the shape of training and validation data
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
# Training the model using the training dataset
lreg = LinearRegression()
lreg.fit(X_train, Y_train)
# Printing the Coefficients of the linear Regression model
print("Coefficients of each feature: ", lreg.coef_)
# Printing the accuracy score of the trained model
score = lreg.score(X_test, Y_test)
print("Accuracy Score: ", score)
```


Output
(150, 4) (90, 4) (90,) (60, 4) (60,) Coefficients of each feature: [-0.17894659 -0.01504423 0.30759575 0.52023143] Accuracy Score: 0.8964830427596993

Table 3.19: Linear Regression Algorithm

3.5.6. Logistic Regression Algorithm

When answering binary classification questions (i.e., when the target values are either 0 or 1), logistic regression is the recommended method. An equation that resembles linear regression can then be used to assess the findings (e.g., how probable is it that a given target value is 0 or 1?).

Program.py
<pre># Importing the required modules and classes from sklearn.datasets import load_iris from sklearn.metrics import accuracy_score from sklearn.linear_model import LogisticRegression from sklearn.model_selection import train_test_split # Loading our dataset data = load_iris() # Splitting the independent and dependent variables X = data.data Y = data.target print("The size of the complete dataset is: ", len(X)) # Creating an instance of the LogisticRegression class for implementing logistic regression log_reg = LogisticRegression() # Segregating the training and testing dataset X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = 10) # Performing the logistic regression on train dataset log_reg.fit(X_train, Y_train) # Printing the accuracy score print("Accuracy score of the predictions made by the model: ", accuracy_score(log_reg.predict(X_test), Y_test))</pre>
Output
The size of the complete dataset is: 150 Accuracy score of the predictions made by the model: 1.0

Table 3.20: Logistic Regression Algorithm

3.5.7. Random Forest Algorithm

Machine learning employs the Random Forest algorithm to facilitate ensemble learning. Compared to a single learning algorithm, the ensemble learning system generates more exceptional prediction analyses by utilizing many Decision Trees and additional machine learning algorithms.

Program.py
<pre># importing the required libraries from sklearn import datasets from sklearn.model_selection import train_test_split from sklearn.metrics import accuracy_score from sklearn.ensemble import RandomForestClassifier # Loading the dataset X, Y = load_iris(return_X_y = True) # Segregating the dataset into training and testing dataset X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.4) # creating an object of the RF classifier class rf = RandomForestClassifier(n_estimators = 100) # Training the classifier model on the training dataset rf.fit(X_train, Y_train) # Predicting the values for the test dataset Y_pred = rf.predict(X_test) # using metrics module to calculate accuracy score print("Accuracy score for the model is: ", accuracy_score(Y_test, Y_pred)) # predicting the type of flower rf.predict([[5, 7, 3, 4]])</pre>
Output
<pre>Accuracy score for the model is: 0.9333333333333333 array([1])</pre>

Table 3.21: Random Forest Algorithm

3.5.8. Decision Tree Algorithm

A well-liked machine learning approach for classification and regression applications is the decision tree algorithm. This tree-based approach divides the data according to various features in order to arrive at decisions. Every leaf node in the tree represents a class or a predicted value, whereas every internal node in the tree represents a feature.

Recursively dividing the data into subsets according to feature values is how the algorithm operates. At each stage, it chooses the optimal feature based on predetermined parameters, like information gain or Gini impurity, to partition the data. Creating branches that decrease impurity or enhance information gain will result in forecasts that are more accurate.

Progrm.py
<pre># Importing the required libraries import numpy as np from sklearn.datasets import load_iris from sklearn.tree import DecisionTreeClassifier from sklearn.model_selection import cross_val_score, train_test_split # Loading the dataset X, Y = load_iris(return_X_y = True) # Splitting the dataset in training and test data X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=0) # Creating an instance of the Decision Tree Classifier class dtc = DecisionTreeClassifier(random_state = 0) dtc.fit(X_train, Y_train) # Calculating the accuracy score of the model using cross_val_score score = cross_val_score(dtc, X, Y, cv = 10) # Printing the scores print("Accuracy scores: ", score) print("Mean accuracy score: ", np.mean(score))</pre>
Output
<pre>Accuracy scores: [1. 0.93333333 1. 0.93333333 0.93333333 0.86666667 0.93333333 1. 1. 1.] Mean accuracy score: 0.96</pre>

Table 3.22: Decision Tree Algorithm

3.5.9. Support Vector Classification

A well-liked machine learning approach for classification and regression applications is called Support Vector Machines (SVM). It works well for managing intricate datasets and determining the ideal boundary for decisions between several classes.

To implement SVM in Python, using the Scikit-learn library. The `svm` module, which comes with a number of SVM implementations like SVC (Support Vector Classification) and SVR (Support Vector Regression), is provided by the library.

Program.py
<pre> # importing libraries from sklearn.datasets import load_iris # from sklearn.preprocessing import StandardScaler from sklearn.model_selection import train_test_split from sklearn.svm import SVC #loading datasets X, Y = load_iris(return_X_y = True) # Splitting the dataset into training and test set. x_train, x_test, y_train, y_test= train_test_split(X, Y, test_size= 0.25, random_state=0) # Creating an instance of the SVC svm =SVC(random_state = 0) svm.fit(X_train, Y_train) # Calculating the accuracy score of the model using cross_val_score score = cross_val_score(svm, X, Y, cv = 10) # Printing the scores print("Accuracy scores: ", score) print("Mean accuracy score: ", np.mean(score)) </pre>
Output
<pre> Accuracy scores: [1. 0.93333333 1. 1. 1. 0.93333333 0.93333333 0.93333333 1. 1.] Mean accuracy score: 0.9733333333333334 </pre>

Table 3.23: Support Vector Classification

3.5.10. K-Nearest Neighbour

Another well-liked machine learning approach for classification and regression applications is K-Nearest Neighbors (KNN). This method, which is non-parametric, forecasts things by comparing data points.

The "K" in KNN stands for the number of closest neighbors taken into account while generating predictions. The procedure determines the majority class label (for classification) or computes the average (for regression) for each new data point by examining its K nearest neighbors in the training set.

Program.py
<pre># Import necessary modules import numpy as np from sklearn.neighbors import KNeighborsClassifier from sklearn.model_selection import train_test_split, cross_val_score from sklearn.datasets import load_iris # Loading data irisData = load_iris() # Create feature and target arrays X = irisData.data y = irisData.target # Split into training and test set X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42) knn = KNeighborsClassifier(n_neighbors=7) knn.fit(X_train, y_train) # Predict on dataset which model has not seen before print(knn.predict(X_test)) score = cross_val_score(knn, X, Y, cv = 10) print("Accuracy scores: ", score) print("Mean accuracy score: ", np.mean(score))</pre>
Output
<pre>[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0] Accuracy scores: [1. 0.93333333 1. 1. 0.86666667 0.93333333 0.93333333 1. 1. 1.] Mean accuracy score: 0.9666666666666668</pre>

Table 3.24: K Nearest Neighbors

3.6. Generators

Generators are relatively simple to use but can be a little confusing to grasp.

Iterators are created using generators, albeit in a different way. Generators are straightforward functions that provide an iterable set of objects in a unique way, one at a time.

The generator is executed when a for statement iteration over a set of items begins. When the function code of the generator reaches a "yield" statement, it returns a new value from the set and returns control of the execution back to the for loop. The generator function yields each value in turn and can generate an endless number of values.

Program.py	Output
<pre>import random def lottery(): # returns 6 numbers between 1 and 40 for i in range(6): yield random.randint(1, 40) # returns a 7th number between 1 and 15 yield random.randint(1, 15) for random_number in lottery(): print("And the next number is... %d!" %(random_number))</pre>	<p>And the next number is... 5!</p> <p>And the next number is... 6!</p> <p>And the next number is... 9!</p> <p>And the next number is... 33!</p> <p>And the next number is... 10!</p> <p>And the next number is... 34!</p> <p>And the next number is... 2!</p>

Table 3.25: Generators

3.7. Lambda Function

They are one-line definitions, akin to mini-functions. When you need a fast function without the hassle of defining a whole function, they come in incredibly helpful. They can be passed as parameters to other functions or used as anonymous functions. They're an excellent tool for writing understandable, succinct code.

Program.py	Output
<pre>a = 1 b = 2 sum = lambda x,y : x + y c = sum(a,b) print(c)</pre>	<p>3</p>

Table 3.26: Lambda Function

3.8. Exception Handling

Exception handling in Python is a powerful way to handle errors and prevent your program from crashing. It allows you to catch and handle specific types of errors that may occur during the execution of your code. You can use the 'try' and 'except' statements to handle exceptions. The code inside the 'try' block is executed, and if an exception occurs, it is caught by the corresponding 'except' block.

Program.py	Output
<pre>def fun(a): if a < 4: b = a/(a-3) print("Value of b = ", b) try: fun(3) fun(5) except ZeroDivisionError: print("ZeroDivisionError Occurred and Handled") except NameError: print("NameError Occurred and Handled")</pre>	ZeroDivisionError Occurred and Handled

Table 3.27: Exception Handling

3.8.1. Try with else clause

The else clause, which must appear after each except clause in the try-except block, is another useful tool in Python. If there is no exception raised by the try clause, the code moves into the else block.

Program.py	Output
<pre>def AbyB(a , b): try: c = ((a+b) / (a-b)) except ZeroDivisionError: print ("a/b result in 0") else: print (c) AbyB(2.0, 3.0) AbyB(3.0, 3.0)</pre>	<pre>-5.0 a/b result in 0</pre>

Table 3.28: Try with else Clause

3.8.2. Finally keyword in python

Python provides a keyword finally, which is always executed after the try and except blocks. The final block always executes after the normal termination of the try block or after the try block terminates due to some exception.

Program.py	Output
<pre>try: k = 5//0 print(k) except ZeroDivisionError: print("Can't divide by zero") finally: print('This is always executed')</pre>	<pre>Can't divide by zero This is always executed</pre>

Table 3.29: Finally Keyword in python

3.9. Decorators

Python's decorators are an incredibly useful feature that let you change a function or class's behavior without altering the original code. They let you add more functionality by acting as wrappers around the original functions or classes.

The name of the decorator function and the `{@}` sign are used to define decorators. They come right before the definition of the function or class.

Program.py	Output
<pre>def uppercase_decorator(func): def wrapper(): original_result = func() modified_result = original_result.upper() return modified_result return wrapper @uppercase_decorator def greet(): return "hello" print(greet())</pre>	HELLO

Table 3.30: Decorators

3.10. Map

The `map()` function in Python is a built-in function that applies a given function to each item in an iterable (such as a list, tuple, or string) and returns an iterator with the results.

Syntax:

`map(function, iterable)`

function: This is the function you want to apply to each element in the iterable.

iterable: This is the sequence of elements you want to apply the function to.

Program.py	Output
<pre>my_pets = ['alfred', 'tabitha', 'william', 'arla'] uppered_pets = list(map(str.upper, my_pets)) print(uppered_pets)</pre>	['ALFRED', 'TABITHA', 'WILLIAM', 'ARLA']

Table 3.31: Mapping Function

3.11. Filter

Along with `map`, Python's `filter` function is an effective tool for manipulating data with iterables. It enables you to pick particular items according to a specified condition.

Syntax:

`filter(function, iterable)`

function: This is the function that defines the test condition for each element. It should return True for elements you want to keep and False for those you want to discard.

iterable: This is the sequence of elements you want to filter.

Program.py	Output
<pre>dromes = ("demigod", "rewire", "madam", "freer", "anutforajaroftuna", "kiosk") palindromes = list(filter(lambda word: word == word[::-1], dromes)) print(palindromes)</pre>	['madam', 'anutforajaroftuna']

Table 3.32: Filter Function

3.12. Reduce

Another useful tool for working with iterables is the reduce function in Python, which allows you to combine items into a single value. It applies a specified function to pairs of items as iteratively moves through the an iterable until only one result is left.

Syntax:

`reduce(function, iterable, initial_value=None)`

Program.py	Output
<pre>from functools import reduce numbers = [3, 4, 6, 9, 34, 12] def custom_sum(first, second): return first + second result = reduce(custom_sum, numbers) print(result)</pre>	68

Table 3.33: Reduce Function

CHAPTER 4

BANK MARKETING PREDICTION

4.1. Introduction to the Project

The bank marketing prediction project uses machine learning techniques to forecast whether or not a client would sign up for a term deposit based on a variety of factors, including demographics, consumer behaviour, and the results of marketing campaigns. The goal of this research is to create a precise and effective model that can calculate a customer's chance of subscribing to a term deposit, allowing banks to better campaign targeting and optimise their marketing tactics.

Understanding client behaviour and preferences is crucial for marketing banking products and services. Banks may learn a lot about the factors that affect consumers' subscription selections by using previous customer data and machine learning techniques. This gives them the ability to target their marketing initiatives and manage resources wisely, which boosts conversion rates and raises consumer satisfaction.

The difficulty of locating prospective subscribers and improving marketing campaigns is what the bank marketing prediction initiative seeks to address. A machine learning model may discover patterns and associations that support good subscription results by analysing numerous client factors, including age, employment, education, and past campaign outcomes. The bank may then use this model to forecast the likelihood that new clients would subscribe, allowing it to focus and tailor its marketing initiatives.

There are various benefits to using machine learning to forecast bank marketing. It lessens dependence on hunches or intuition and enables more precise, data-driven decision-making. Large amounts of client data may be processed using machine learning models, which can also reveal complicated patterns that would not be seen using more conventional techniques of analysis. This enables banks to more precisely target their marketing initiatives, increasing conversion rates and improving return on investment.

In conclusion, the bank marketing prediction project makes use of machine learning techniques to anticipate the possibility that clients would subscribe to term deposits. Banks may increase consumer targeting, campaign efficacy, and marketing strategy optimization by creating a reliable prediction model. This project serves as an example of how machine learning may transform marketing strategies in the banking sector.

4.1.1. Statement of the Problem

The challenge is to build a machine learning model using the decision tree algorithm to forecast whether a consumer would sign up for a term deposit in a bank marketing campaign. Building an accurate and understandable model is the aim in order to help the bank discover potential subscribers and maximise their marketing efforts.

To encourage term deposits among its clients, the bank runs marketing initiatives. To make marketing efforts more successful and economical, it is necessary to pinpoint the clients who are most likely to subscribe to the product. The bank may target the correct clients with its resources and adjust its marketing strategy by estimating the chance of subscription.

The decision tree method is a good option for this issue since it can handle both category and numerical information and offers a model that is clear and easy to understand. It recursively divides the data into sections depending on several attributes to produce a structure resembling a tree, allowing the model to generate predictions by following a route in the tree.

The model will be trained using past customer data, including demographic details, prior campaign results, and other pertinent characteristics. To create predictions about new, unforeseen clients, the decision tree algorithm will discover patterns and correlations in the data. In order to determine how well the model performs in forecasting term deposit subscriptions, performance measures including accuracy, precision, recall, and F1-score will be used.

The project's output will be a decision tree model that the bank can use to find new customers and improve its advertising efforts. The bank hopes to boost client happiness, raise conversion rates for their marketing campaigns, and increase return on investment by utilising the capabilities of the decision tree algorithm.

The goal of the decision tree algorithm-based bank marketing prediction project is to create a predictive model that will help the bank identify clients who are likely to sign up for term deposits. The bank may deploy resources efficiently for maximum impact by using the decision tree's capacity to collect patterns and give interpretability.

4.2 System Analysis

4.2.1. Existing System

Machine learning-based bank marketing prediction techniques and methods abound. Several of the often-employed techniques include:

Logistic Regression: A common approach for binary classification problems, such as forecasting whether a consumer would sign up for a term deposit, is logistic regression. It stimulates the connection between the characteristics of the input and the likelihood of a favorable result.

Support Vector Machines (SVM): SVM is yet another popular approach for classification jobs. By identifying an ideal hyperplane that maximally divides the classes, it categorizes data points into discrete groups.

Decision Tree: This ensemble learning technique makes predictions by combining various decision trees. It can handle both numerical and categorical characteristics, manage unbalanced data, and offer feature priority rankings, making it suitable for bank marketing prediction.

Gradient Boosting: Gradient Boosting is a boosting method that sequentially assembles a group of ineffective learners, generally decision trees. It focuses on fixing the flaws in earlier models and increasing forecast accuracy.

Neural Networks: Deep learning models of neural networks, in particular, have grown in prominence in recent years due to their capacity to recognise intricate patterns in massive amounts of data. They are highly predictive and may capture complex correlations between characteristics. These currently available technologies have been used to forecast bank marketing jobs with encouraging outcomes.

The system to choose, however, will rely on a number of variables, including the dataset's size and composition, processing capabilities, the need for interpretability, and the specific objectives of the bank.

It is essential to remember that the specific dataset and issue area might have an impact on these systems' performance. Before choosing the optimal system for a particular bank marketing prediction assignment, it is advised to experiment with different models, fine-tune their parameters, and assess their performance using relevant metrics.

4.2.2 Limitations of the Existing System

- * Accuracy
- * No faster mode
- * Computational Complexity

4.3 Data Collection

4.3.1. Data Sources

For this case study, we have chosen the Real-Time data set from Kaggle. The dataset has outliers and has to do feature selection in such a way that the selected features help for getting better accuracy.

4.3.2. Data Profile

Data Types:

- 1) age: (numeric)
- 2) job: type of job (categorical: 'admin', 'blue-collar', 'entrepreneur', 'housemaid', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
- 3) marital: marital status (categorical: 'divorced', 'married', 'single', 'unknown')
- 4) education: (categorical: primary, secondary, tertiary, unknown)
- 5) default: has credit in default? (Categorical: 'no', 'yes', 'unknown')
- 6) housing: has housing loan? (Categorical: 'no', 'yes', 'unknown')
- 7) loan: has personal loan? (Categorical: 'no', 'yes', 'unknown')
- 8) balance: Balance of the individual.
- 9) contact: contact communication type (categorical: 'cellular', 'telephone')
- 10) month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- 11) day: last contact day of the week
- 12) duration: last contact duration, in seconds (numeric). ### Other attributes:
- 13) campaign: number of contacts performed during this campaign and for this client.
- 14) pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- 15) previous: number of contacts performed before this campaign and for this client (numeric)
- 16) poutcome: outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success')
- 17) deposit: has the client subscribed a term deposit? (Binary: 'yes', 'no')

Data Quality:

- Missing Values: There are some null values in the dataset, and they are filled with mean values.
- Outliers: There are a few outliers in the data, and they are shown in the visualization.

4.4 Data Cleaning & Preprocessing

4.4.1 Data Preparation

Step 1: Uni-variate Analysis

Step 2: Bi-variate Analysis

Step 3: Type Casting each column with respect to the required format.

Step 4: Encoding

4.5. Exploratory Data Analysis

4.5.1. Checking for null values.

```
In [4]: df.isnull().sum()
Out[4]: age          0
        job          0
        marital      0
        education    0
        default      0
        balance      0
        housing      0
        loan         0
        contact      0
        day          0
        month        0
        duration     0
        campaign     0
        pdays        0
        previous     0
        poutcome     0
        deposit      0
        dtype: int64
```

Figure 4.1: Checking for null value

4.5.2. Dataset Information.

```
bank.info()
executed in 81ms, finished 20:10:25 2023-06-20

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   age         11162 non-null  int64
1   job         11162 non-null  object
2   marital     11162 non-null  object
3   education   11162 non-null  object
4   default     11162 non-null  object
5   balance     11162 non-null  int64
6   housing     11162 non-null  object
7   loan        11162 non-null  object
8   contact     11162 non-null  object
9   day         11162 non-null  int64
10  month       11162 non-null  object
11  duration    11162 non-null  int64
12  campaign    11162 non-null  int64
13  pdays       11162 non-null  int64
14  previous    11162 non-null  int64
15  poutcome    11162 non-null  object
16  deposit     11162 non-null  object
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
```

Figure 4.2: Dataset information

4.5.3. Dataset Description.

bank.describe()							
executed in 58ms, finished 20:10:26 2023-06-20							
	age	balance	day	duration	campaign	pdays	previous
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
mean	41.231948	1528.538524	15.658036	371.993818	2.508421	51.330407	0.832557
std	11.913369	3225.413326	8.420740	347.128386	2.722077	108.758282	2.292007
min	18.000000	-6847.000000	1.000000	2.000000	1.000000	-1.000000	0.000000
25%	32.000000	122.000000	8.000000	138.000000	1.000000	-1.000000	0.000000
50%	39.000000	550.000000	15.000000	255.000000	2.000000	-1.000000	0.000000
75%	49.000000	1708.000000	22.000000	496.000000	3.000000	20.750000	1.000000
max	95.000000	81204.000000	31.000000	3881.000000	63.000000	854.000000	58.000000

Figure 4.3: Dataset Description

4.6. Data Visualization Techniques

4.6.1 Analysis of Dependent Variable (Count)

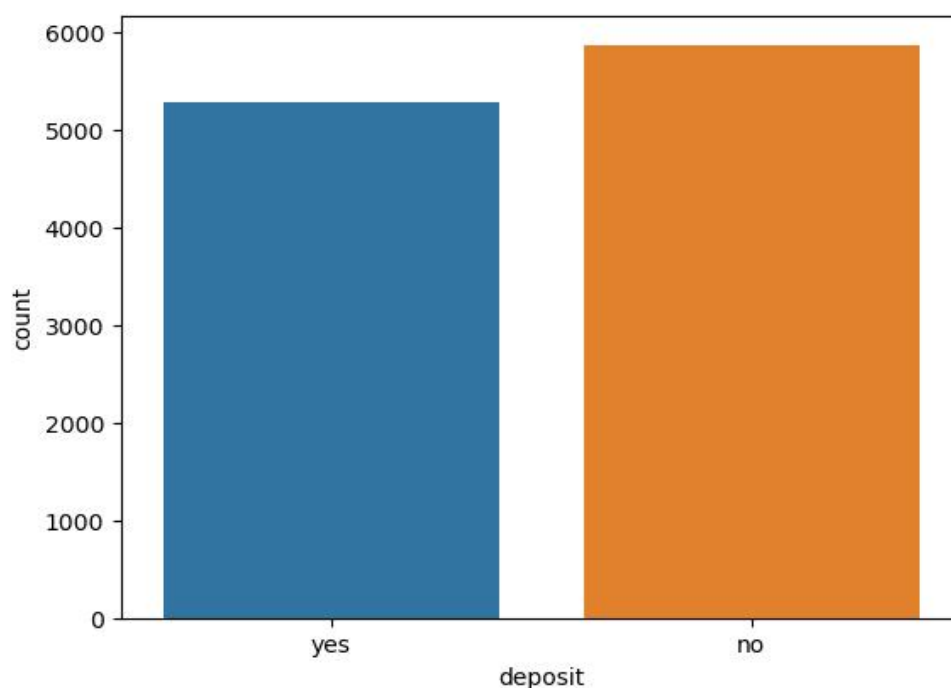


Figure 4.4: Analysis of Dependent

4.6.2. Analysis of Job on Deposit

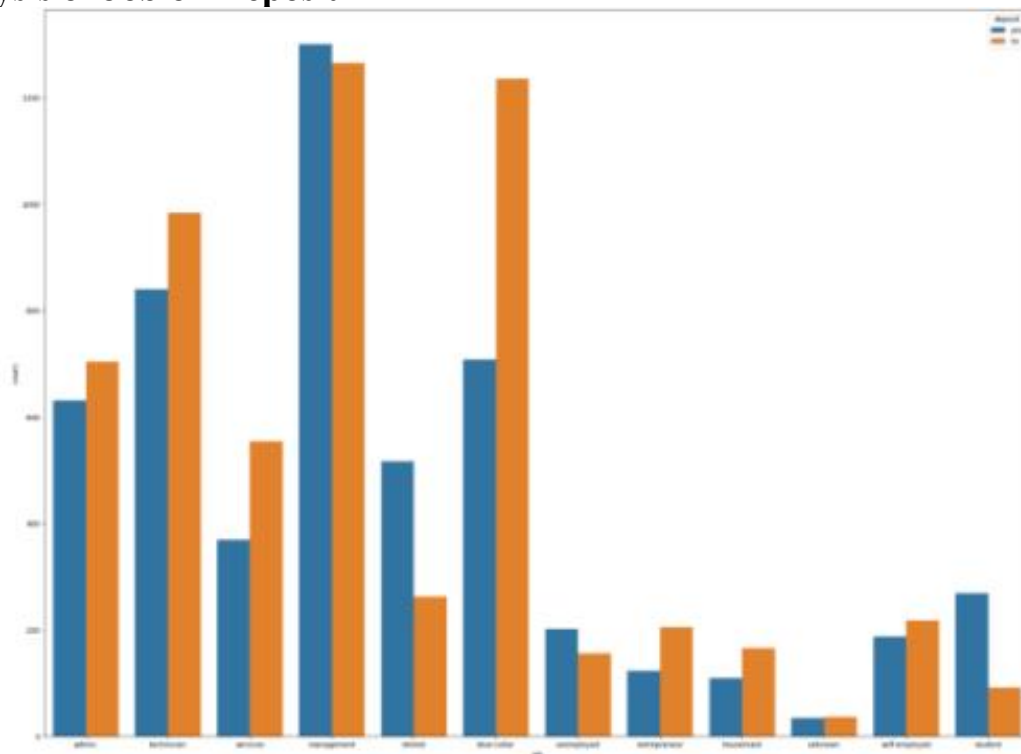


Figure 4.5: Analysis of Job on Deposit

4.6.3. Analysis of Marital on Deposit

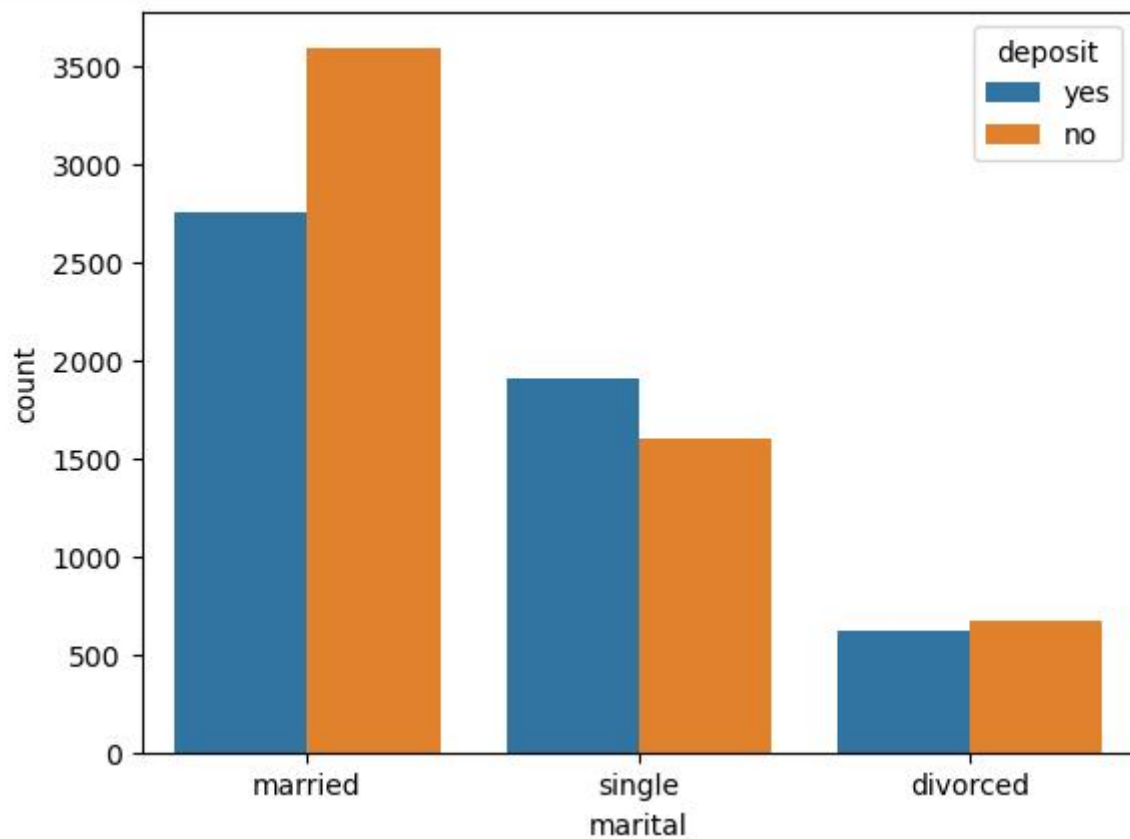


Figure 4.6: Analysis of Marital on Deposit

4.6.4. Analysis of education on deposit

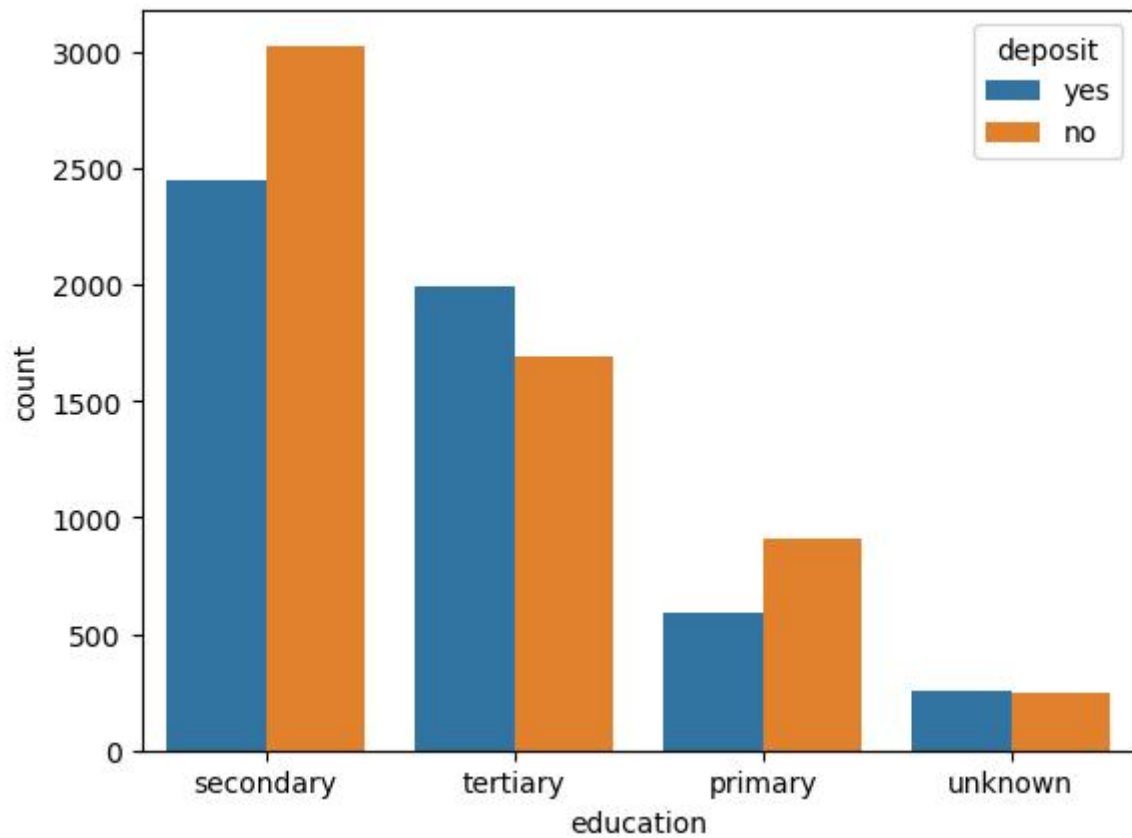


Figure 4.7: Analysis of Education on Deposit

4.6.5. Analysis of housing on Deposit

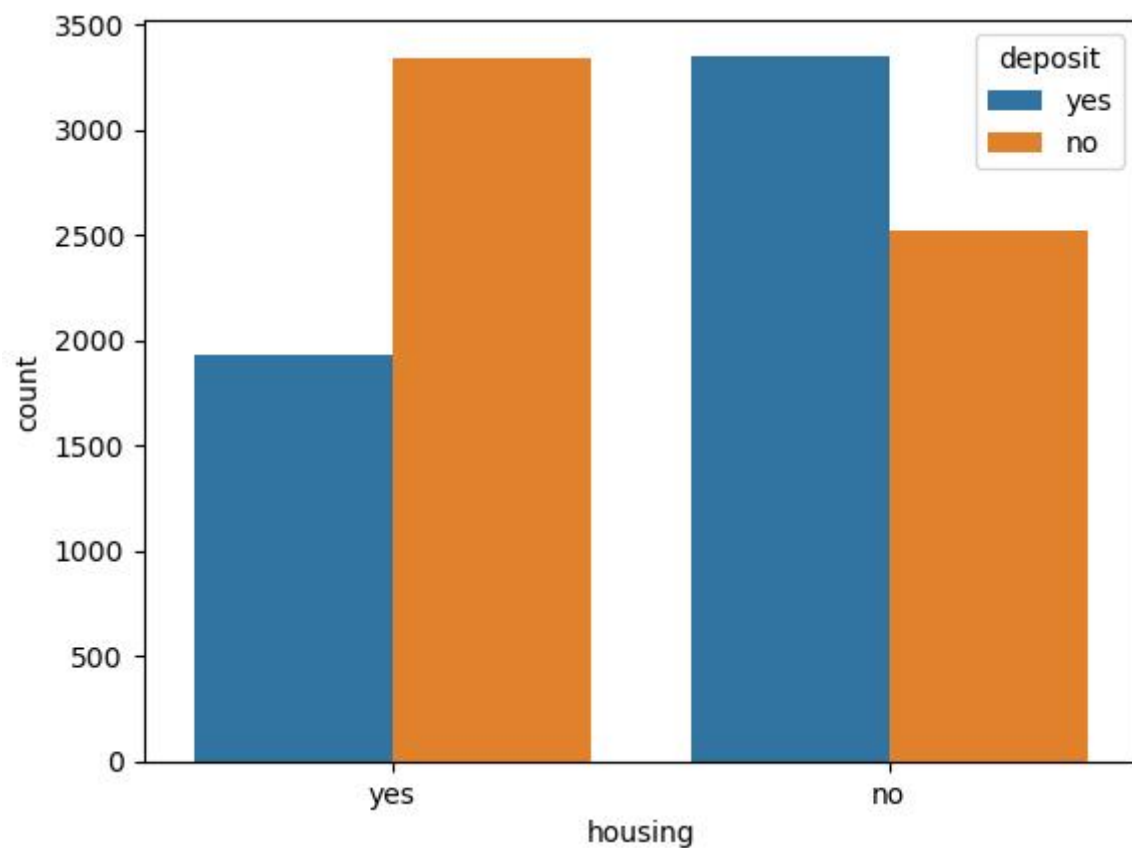


Figure 4.8: Analysis of Housing on Deposit

4.6.6. Correlation Plot

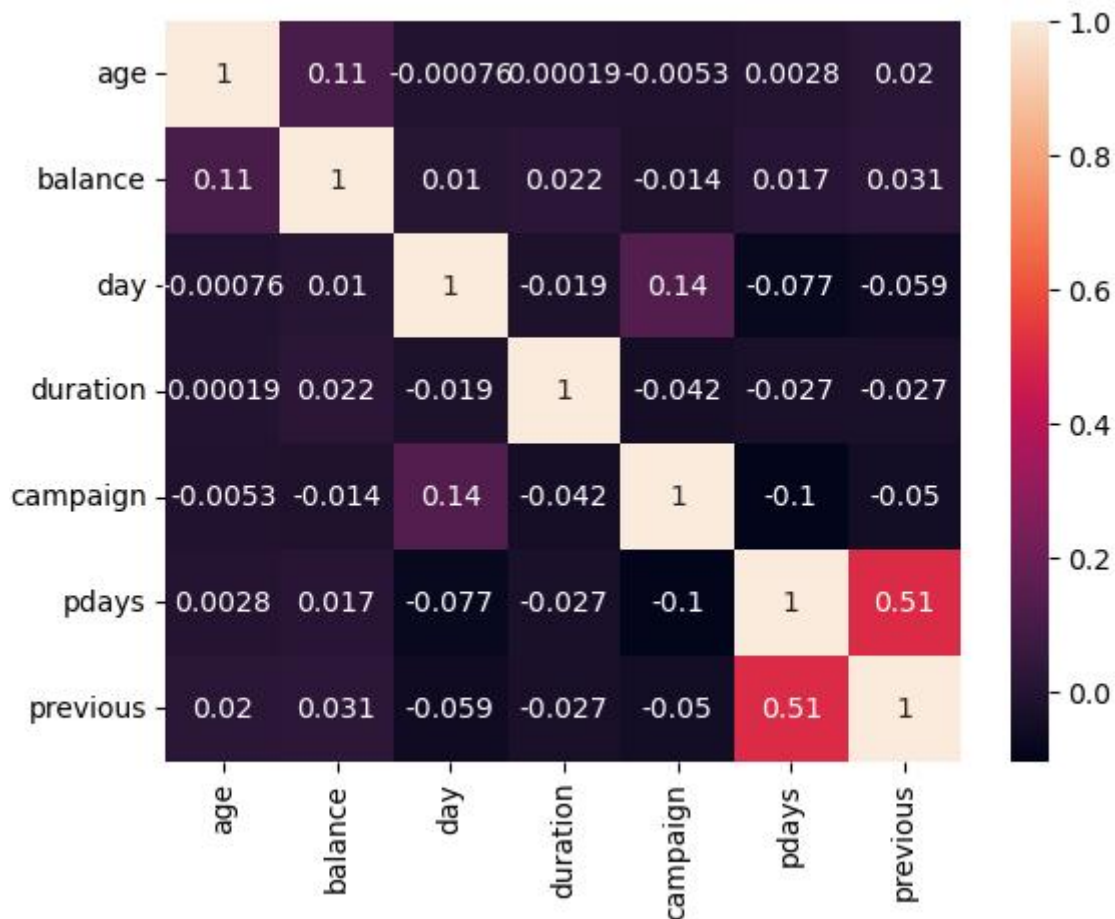


Figure 4.9: Correlation Plot

4.7. Methodology

Data collection: The first step is to collect relevant data that can be used to build the model. The data should include information about Drugs.

Data pre-processing: The collected data must be pre-processed to handle missing values, deal with outliers, and convert categorical variables into numerical ones. The pre-processing step is critical for ensuring the data quality and the model's accuracy.

Feature selection: The next step is to select the features that will be used to build the model. This step involves identifying the most important features that significantly impact the target variable.

Splitting the data: The data must be split into two parts: training data and testing data. The training data will be used to build the model, while the testing data will be used to evaluate its performance.

Model training: The next step is to train the Decision Tree model using the training data. The model will use the selected features to learn the relationship between the features and the target variable.

Model evaluation: The trained model must be evaluated using the testing data. This step will provide insights into the model's performance and allow for any necessary adjustments. The evaluation metrics used to assess the performance of the model may include accuracy, precision etc.

4.7.1. Data Models

Decision Tree: A hierarchical structure called a decision tree is used to depict decisions and their outcomes. To produce a model that resembles a tree, it recursively splits the data according to the feature values. Each leaf node represents a class label or outcome, whereas each internal node represents a judgement made in response to a feature.

In machine learning, the decision tree is a well-liked data model for predicting bank marketing. This approach for supervised learning may be applied to both classification and regression applications. The decision tree model may be used in the context of bank marketing to forecast whether a consumer would sign up for a term deposit depending on different input variables.

4.7.2. Model Building

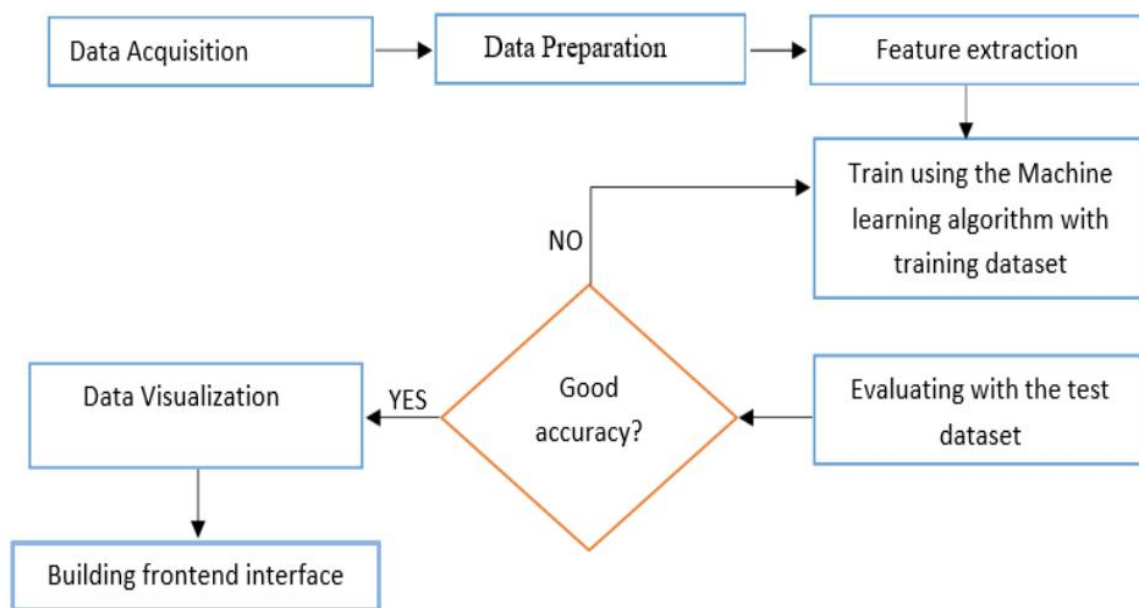


Figure 4.10: Model Building

Data acquisition: The first step in any machine learning project is to acquire the relevant data. This could involve collecting data from different data sources such as Kaggle and Kdneggets. It is important to ensure that the data is of high quality and free of errors or biases. And in this project, we have taken datasets from the 365datascience.com website.

Data preparation: Once the data has been acquired, it needs to be cleaned and preprocessed before it can be used to train a machine learning model. This might involve removing missing values, handling outliers, normalizing the data, and converting categorical variables into numerical ones.

Feature extraction: After the data has been cleaned and preprocessed, relevant features need to be extracted from the data. Feature extraction involves selecting the most relevant variables that can help the machine learning algorithm learn patterns in the data. This could involve techniques such as principal component analysis (PCA), feature scaling, or feature selection.

Split the dataset: Once the data has been preprocessed and features have been extracted, the dataset needs to be split into training and testing sets. The training set is used to train the machine learning model, while the testing set is used to evaluate the performance of the model.

Training model: With the data split into training and testing sets, the next step is to train the machine learning model. This could involve using algorithms such as logistic regression, decision trees, and support vector machines. Decision Tree is one of the ensemble methods that can be used to improve the accuracy of the model.

Evaluation of model: After training the machine learning model, it is important to evaluate its performance on the testing set. This could involve using metrics such as accuracy, precision, recall and F1 score. It is important to ensure that the model is not overfitting the training data and is able to generalize to new data.

Data visualization: Data visualization is an important step in any machine learning project. It involves using visual tools such as scatter plots, histograms, and heatmaps to explore the data and identify patterns or relationships. Visualization can help in feature selection and determining the most important features.

4.7.3. Model Selection & Evaluation

4.7.3.1. Model Selection

We selected Decision Tree for this project, Decision Tree is a powerful machine learning algorithm that is often used for classification, regression, and feature selection.

Decision Tree is an ensemble learning method that combines multiple decision trees to improve the accuracy and robustness of the prediction. Each decision tree is trained on a random subset of the data and a random subset of the features, and the final prediction is based on the majority vote of the individual trees. Decision Tree is a popular model selection algorithm because it is robust to noise and overfitting, and it can handle both categorical and continuous variables.

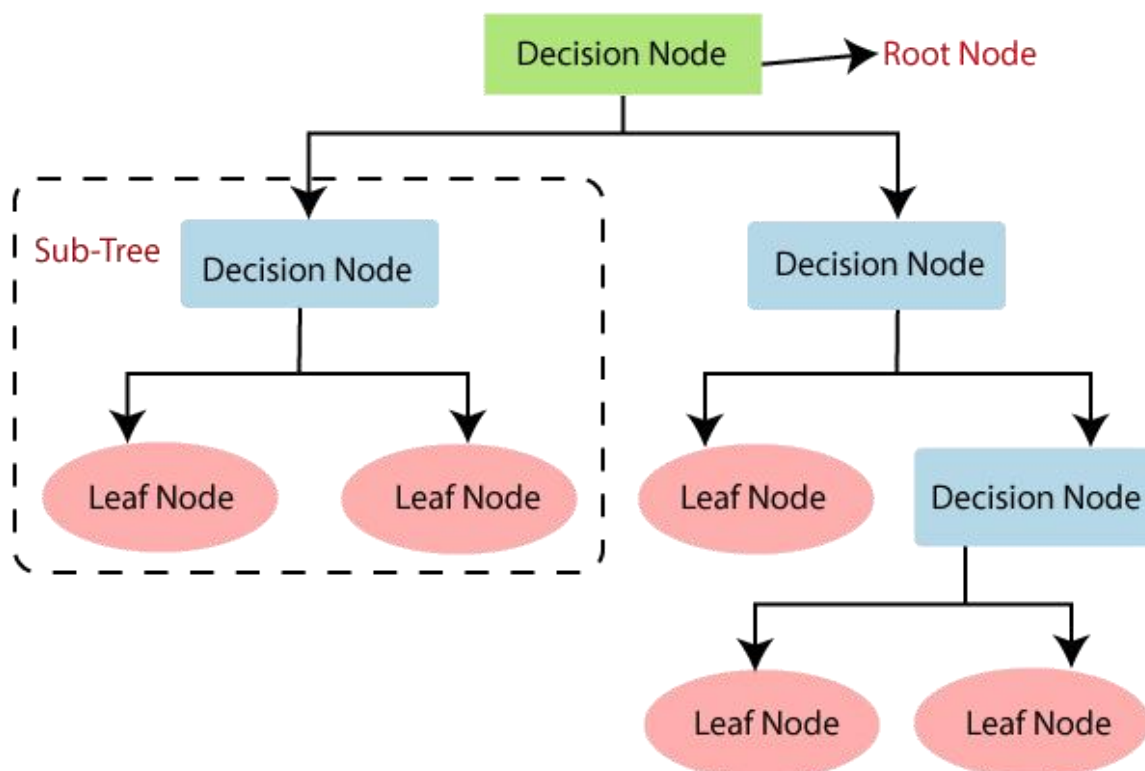


Figure 4.11: Decision Tree

4.7.3.2. Model Evaluation

A decision Tree is a popular machine-learning algorithm that can be used for predicting Bank Marketing Prediction. In order to evaluate the performance of a Decision Tree model for this problem, we can use a variety of metrics.

One common evaluation metric for Decision Tree models is accuracy. Accuracy measures the proportion of correctly classified instances out of all the instances. In our case, it would measure the proportion of Bank Marketing Prediction predictions.

So, we used evaluation metrics for binary classification problems like this including precision, recall, and F1 score.

4.8. Result

- Accuracy of the Decision Tree Is 75.25%
- Accuracy score of training data: 100%
- Accuracy score of test data: 75.27989252127183%

REFLECTION NOTES

The document provides a comprehensive overview of the application of Python programming and machine learning techniques in the context of bank marketing prediction. It highlights the importance of understanding client behavior and preferences in marketing banking products and services, emphasizing the potential impact of machine learning models in optimizing marketing strategies.

The inclusion of detailed examples and program outputs, such as string manipulation, conditional expressions, and data visualization using libraries like Matplotlib and Seaborn, offers practical insights into the application of Python in data analysis and visualization.

The project's methodology section outlines a systematic approach to data collection, preprocessing, feature selection, model training, and evaluation, providing a clear road map for implementing machine learning models in real-world scenarios.

The Decision Tree algorithm is identified as the model of choice, and its strengths in handling both categorical and continuous variables, as well as its robustness to noise and over-fitting, are highlighted. The model evaluation process, including the use of accuracy, precision, recall, and F1 score metrics, demonstrates a thorough approach to assessing model performance.

Overall, the document serves as a valuable resource for understanding the practical application of Python and machine learning in the banking sector, providing a foundation for further exploration and implementation of similar projects.

CONCLUSION

The bank marketing prediction project discussed in this document aims to use machine learning techniques, specifically the Decision Tree algorithm, to forecast whether a client would sign up for a term deposit based on various factors. The project emphasizes the importance of understanding client behavior and preferences in marketing banking products and services. By utilizing machine learning models, banks can target their marketing initiatives more effectively, manage resources wisely, and ultimately increase conversion rates and consumer satisfaction.

The project also highlights the significance of data collection, data cleaning, and preprocessing in preparing the dataset for model building. Exploratory data analysis techniques, such as checking for null values, dataset information, and data visualization, are crucial in understanding the dataset and identifying patterns or relationships.

The methodology section outlines the steps involved in building the model, including data acquisition, model training, evaluation, and data visualization. The Decision Tree algorithm is selected as the model of choice due to its ability to handle both categorical and continuous variables, robustness to noise, and overfitting.

The model evaluation process involves assessing the accuracy, precision, recall, and F1 score of the Decision Tree model. The accuracy of the Decision Tree model is reported to be 75.25%, with a 100% accuracy score on the training data and 75.28% accuracy score on the test data.

In conclusion, the bank marketing prediction project demonstrates the potential of machine learning techniques, particularly the Decision Tree algorithm, in forecasting client behavior and optimizing marketing strategies in the banking sector. The project serves as a valuable example of how machine learning can transform marketing strategies and improve decision-making processes in the banking industry.

REFERENCES

- [1]. <https://learnpython.org/> for python basics and advanced
- [2]. <https://www.w3schools.com/python/> for NumPy Array, Pandas in python and matplotlib library
- [3]. <https://www.codewithharry.com/tutorial/python/> for python basics
- [4]. <https://www.geeksforgeeks.org/python-programming-language/?ref=shm> for sklearn library
- [5]. <https://github.com/Harshas5/TCR-PROJECT> for project