COMSATS University Islamabad, Lahore Campus
Department of Electrical and Computer Engineering

CSC462 –Artificial Intelligence

Lab Resource Person

Dr. Muhammad Usman Iqbal

Theory Resource Person

Dr. Farooq-i-Azam

Supervised By

Dr. Ikram Ullah Khosa

Name: _____ Program _____

Registration Number:***CIIT/*** ___ - ___ - ___ ***/LHR***Batch: _____

Fall 2023

Revision History

Sr.No.	Update	Date	Performed by
1	Lab Manual Preparation	Fall 2023	Dr. Muhammad Usman Iqbal

Preface

This comprehensive lab manual is designed to provide students with hands-on experience in essential artificial intelligence and machine learning techniques through structured, practical experiments. The manual covers a wide range of fundamental topics, from basic Python programming for AI to advanced machine learning algorithms and neural networks.

Each experiment has been carefully crafted to bridge theoretical concepts with real-world applications, guiding learners through:

- Foundational Python programming for data manipulation and analysis
- Core machine learning techniques including linear regression, decision trees, and ensemble methods
- Advanced topics such as neural networks and convolutional neural networks
- Critical data preprocessing skills including handling imbalanced data and feature engineering

The experiments follow a progressive learning path, beginning with basic syntax and building towards complex AI model implementation. Every lab includes clear objectives, step-by-step coding exercises, visualization techniques, and thought-provoking post-lab questions to reinforce learning.

Special emphasis has been placed on:

- Developing practical implementation skills
- Understanding model evaluation and optimization
- Troubleshooting common errors in machine learning pipelines
- Interpreting results through statistical and visual analysis

This manual provides the necessary tools to transition from theory to practice. The experiments are designed to work with real-world datasets while being adaptable to different learning environments.

We hope this collection of labs serves as both an educational resource and a reference guide for your journey in artificial intelligence. The skills developed through these experiments form the foundation for more advanced studies and practical applications in data science and AI.

Books

Textbook

1. Stuart Russell and Peter Norvig, Artificial Intelligence. A Modern Approach, 4 th edition,
2. Prentice Hall, Inc.

Reference Books

1. Hart, P.E., Stork, D.G. and Duda, R.O. Pattern classification. 2 nd edition, John Willey & Sons.
2. Patrick Henry Winston, Artificial Intelligence, 3 rd edition, Addison-Wesley Publishing Company

Course Learning Outcomes

Theory CLOs

1. Understand and implement key components in the field of artificial intelligence. (C3-PLO1)
2. Analyze artificial intelligence techniques for practical problem solving. (C4-PLO2)

Lab CLOs

1. Manipulate AI algorithms and techniques in practical scenarios, demonstrating the ability to develop basic expert systems. (P3-PLO5)

CLOs – PLOs Mapping

PLO / CLO	PLO1	PLO2	PLO5	PLO10	Cognitive Domain	Affective Domain	Psychomotor Domain
CLO1	x				C3		
CLO2		x			C4		
Lab CLO1			x				P3

Lab CLOs – Lab Experiment Mapping

CLO / Lab	Lab 1	Lab 2	Lab 3	Lab 4	Lab 5	Lab 6	Lab 7	Lab 8	Lab 9	Lab 10	Lab 11	Lab 12

Lab CLO1	P3	P3	P3	P3	P3	P3	P2	P3	P2	P3	P3	P3

Grading Policy

Lab Assignments: i. Lab Assignment 1 Marks (Lab marks from Labs 1-3) ii. Lab Assignment 2 Marks (Lab marks from Labs 4-6) iii. Lab Assignment 3 Marks (Lab marks from Labs 7-9) iv. Lab Assignment 4 Marks (Lab marks from Labs 10-12)	25%
Lab Mid Term = 0.5*(Lab Mid Term exam) + 0.5*(average of lab evaluation of Lab 1-6)	25%
Lab Terminal = 0.5*(Lab Terminal exam) +0.375*(average of lab evaluation of Lab 7-12) + 0.125*(average of lab evaluation of Lab 1-6)	50%

The minimum pass marks for both lab and theory shall be 50%. Students obtaining less than 50% marks (in either theory or lab, or both) shall be deemed to have failed in the course. The final marks would be computed with 75% weight to theory and 25% to lab final marks.

Software Recourses

3. Python

Lab Instructions

- All labs comprise two parts: Pre-Lab and In-Lab Exercises
- The students should complete and demonstrate each lab task separately for stepwise evaluation (please ensure that course instructor/lab engineer has signed each step after ascertaining its functional verification)

- Only those tasks that completed during the allocated lab time will be credited to the students. Students are however encouraged to practice on their own in spare time for enhancing their skills.

Safety Instructions

All students must read and understand the information in this document with regard to laboratory safety and emergency procedures prior to the first laboratory session. **Your personal laboratory safety depends mostly on YOU**. Effort has been made to address situations that may pose a hazard in the lab but the information and instructions provided cannot be considered all-inclusive.

The danger of injury or death from electrical shock, fire, or explosion is present while conducting experiments in this laboratory. To work safely, it is important that you understand the prudent practices necessary to minimize the risks and what to do if there is an accident.

Avoid contact with conductors in energized electrical circuits. Do not touch someone who is being shocked while still in contact with the electrical conductor or you may also be electrocuted. Instead, press the Emergency Disconnect (red button located near the door to the laboratory). This shuts off all power, except the lights.

Make sure your hands are dry. The resistance of dry, unbroken skin is relatively high and thus reduces the risk of shock. Skin that is broken, wet, or damp with sweat has a low resistance. When working with an energized circuit, work with only your right hand, keeping your left hand away from all conductive material. This reduces the likelihood of an accident that results in current passing through your heart.

Be cautious of rings, watches, and necklaces. Skin beneath a ring or watch is damp, lowering the skin resistance. Shoes covering the feet are much safer than sandals.

If the victim isn't breathing, find someone certified in CPR. Be quick! Some of the staff in the Department Office are certified in CPR. If the victim is unconscious or needs an ambulance, contact the Department Office for help or call 1122. If able, the victim should go to the Student Health Services for examination and treatment.

Transistors and other components can become extremely hot and cause severe burns if touched. If resistors or other components on your proto-board catch fire, turn off the power supply and notify the instructor. If electronic instruments catch fire, press the Emergency Disconnect (red button). These small electrical fires extinguish quickly after the power is shut off. Avoid using fire extinguishers on electronic instruments.

Table of Contents

Lab #1: Understand Python Basics

1.1 Objectives

Follow fundamental Python syntax to write and execute basic programs under supervision, forming the base for AI algorithm development.

1.2 Pre-Lab

- Introduction to python and its applications.
- Installing Anaconda for Python, set basic environment and install libraries and packages using Anaconda command prompt.
- Understanding the Spyder interface for executing the python codes.

1.2.1 Python

Python is a very popular programming language with many great featuresfor developing Artificial Intelligence. Many Artificial Intelligence developers allaround the world use Python. This lab experiment will provide an introduction to the PythonProgramming Language.

Why the python is so popular Artificial Intelligence developers.

4. Python is a general-purpose programming language conceived in 1989 by Dutchprogrammer Guido van Rossum.

- Python is free and open source, with development coordinated through thePython Software Foundation.
- Python has experienced rapid adoption in the last decade and is now one of themost commonly used programming languages.

1.2.2 Common Applications

Python is a general-purpose language used in almost all application domains such as

- Communications
- Web development (Flask and Django covered in the future chapters)
- CGI and graphical user interfaces
- Game development
- AI and data science (very popular)

1.2.3 Setting up Your Python Environment

The core Python package is easy to install but **not** what you should choose for these labs.

We require certain libraries and the scientific programming eco system, which;

- The core installation (e.g., Python 3.7) does not provide.
- Is painful to install one piece at a time (concept of a package manager)

And, we are using Anaconda.

1.2.3.1 Anaconda

To install Anaconda, download the binary and follow the instructions.

Important points:

5. Install the latest version.
6. If you are asked during the installation process whether you would like to make Anaconda your default Python installation, say yes.

Anaconda downloading, installation and setting up environment video tutorial is available at the following link: https://youtu.be/4kpJYuup3lg

1.2.3.2 Test Your Python Installation and Environment

Pre-Lab Task 1

Execute a simple python program to check the python installation and environment setup.

```
# Import two packages first
import numpy as np
import matplotlib.pyplot as plt
# The program starts here.
values = np.random.randn(100)
plt.plot(values)

plt.title('Random Noise using Test Program')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

After executing the program, you should see an output similar to the following image.

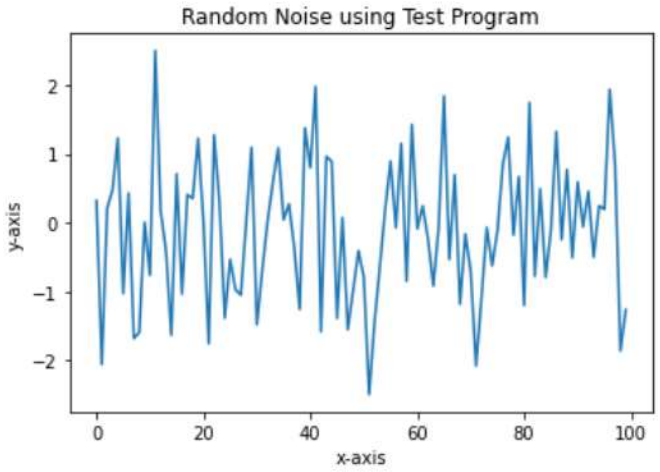


Figure 1: Image displayed by the sample program.

1.2.3.3 Using Help and Documentation in Python

To use help on any function or command, simply type in the prompt and a small window will appear for help.

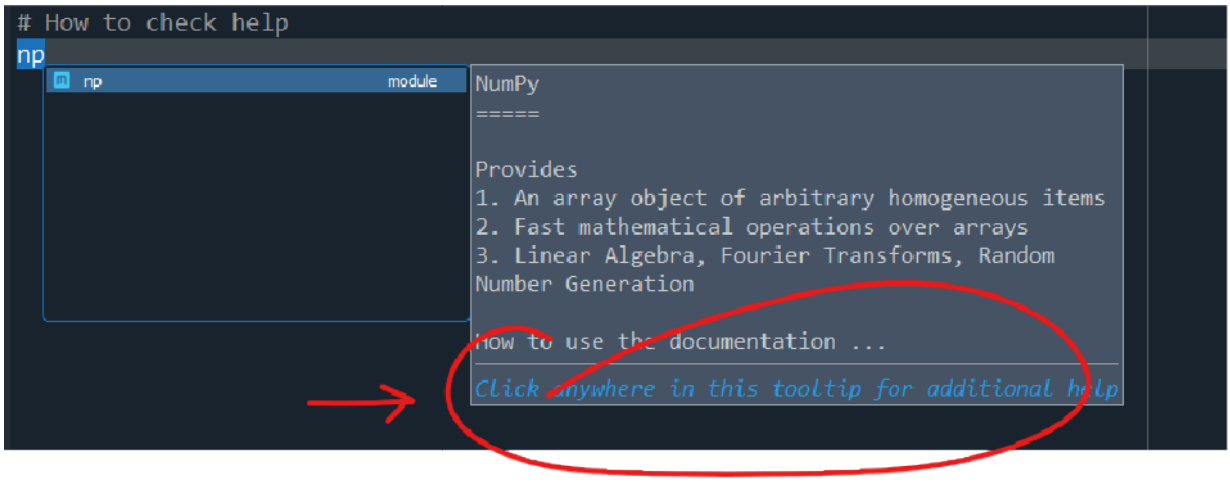


Figure 2: Using Help in Python

1.3 1.2 In-Lab

Write the following code in Spyder IDE and learn about printing a string in python.

```
# Print "Hello World"
# ============================================================================
print('Hello world')
```

In-Lab Task 1

Print the following lines using one print command.

```
Hello World!
My name is Dr. Muhammad Usman Iqbal
My Registration ......
My First AI Lab
```

Use the line breaks and a new line in printing the string.

7. \ (Learn to use)
8. \n (Learn to use)
9. You can also add helpful comments to your code with the # symbol. Any linestarting with a # is not executed by the interpreter.

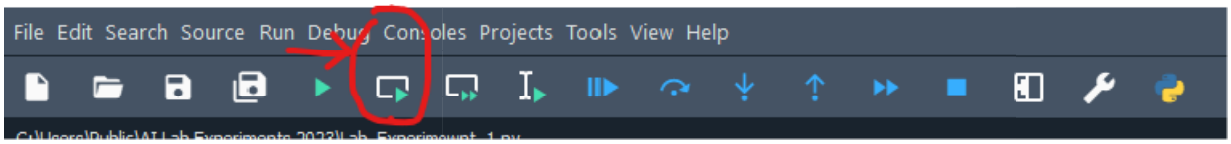
1.3.1.1 Learn to use a Running a Single Cell of Code

The python code can be sectioned into cells which we can execute separately. The cells can be created by using the following syntax " #%% "

In-Lab Task 2

```
#%%
# ========================================================================
# Print 'Hello World!'
# ========================================================================
# print ('Hello World!')


#%%
```

Now using the run only cell in Spyder IDE, we can execute a single cell instead of complete code.



1.3.1.2 Indentation

Python uses indentation to indicate parts of the code that need to be executedtogether. Both tabs and spaces (usually four per level) are supported, and mypreference is for tabs.

In-Lab Task 3

```
# %%
# ========================================================================
# Indentation
# ========================================================================
x = 1
if x == 1:
    print("x is 1")


#%%
```

To assign a variable with a specific value, use =.

To test whether a variable has a specific value, use the Boolean operators:

- equal: ==
- not equal: !=
- greater than: >
- less than: <

1.3.1.3 Variables and Types

Python is not “statically-typed.” This means you do not have to declare all yourvariables before you can use them. You can create new variables whenever you want. Python is also “object-oriented,” which means that every variable is an object.

In-Lab Task 4

Numbers

```
# ================================================================================
# Numbers
# ================================================================================
# 1. Integers
print('Print Value of Integer ...')
integer_us = 7
print(integer_us)
# notice that the class printed is currently int
print ('Class of the interger is' )
print(type(integer_us))

# 2. Float
print('Print Value of Float ...')
float_us = 7.0
print(float_us)
# Or you could convert the integer you already have
myfloat = float(integer_us)
# Note how the term `float` is green. It's a protected term.
print(myfloat)


# 3. Convert float to integer
# Now see what happens when you convert a float to an int
print('Convert Value of Float to Integer ...')
myint = int(7.3)
print(myint)
```

Strings

```
# ================================================================================
# Srings
# ================================================================================

mystring = "Hello, World!"
print(mystring)
# and demonstrating how to use an apostrophe in a string
mystring = "Let's talk about apostrophes..."
print(mystring)


# Assign strings as variables
one = 1
two = 2
three = one + two
print(three)
hello = "Hello,"
world = "World!"
helloworld = hello + " " + world
print(helloworld)
# assign multiple variables simultaneously
a, b = 3, 4
print(a, b)

# Mixing different data variables
print(one + two + hello)
```

Lists

`Lists` are an ordered list of any type of variable. You can combine as manyvariables as you like, and they could even be of multiple data types. Ordinarily,unless you have a specific reason to do so, lists will contain variables of one type.You can also iterate over a list (use each item in a list in sequence).A list is placed between square brackets: `[]`.

```
mmylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
# Each item in a list can be addressed directly.
# The first address in a Python list starts at 0
print(mylist[0])
# The last item in a Python list can be addressed as -1.
# This is helpful when you don't know how long a list is
→likely to be.
print(mylist[-1])
# You can also select subsets of the data in a list like this
print(mylist[1:3])
# You can also loop through a list using a `for` statement.
# Note that `x` is a new variable which takes on the value of
# each item in the list in order.
```

```
# Let's imagine we have a list of unordered names that somehow got some random
# numbers included.
#  we want to print the alphabetised list of names without the numbers.

names = ["John", 3234, 2342, 3323, "Eric", 234, "Jessica",
         734978234, "Lois", 2384]

print("Number of names in list: {}".format(len(names)))


# First, let's get rid of all the weird integers.
new_names = []

for n in names:
    if isinstance(n, str):
    # Checking if n is a string
    # And note how we're now indented twice into this new component
            new_names.append(n)
# We should now have only names in the new list. Let's sort them.


new_names.sort()

print("Cleaned-up number of names in list: {}".format(len(new_names)))

# Lastly, let's print them.
for i, n in enumerate(new_names):
    # Using both i and n in a formated string
    # Adding 1 to i because lists start at 0
    print("{}. {}".format(i+1, n))
```

1.4 1.3 Post-Lab

Post-Lab Task

Understand the following syntax for the for loop, while loop and if condition in python and sample program for each.

for Loop

```
# ================================================================================
# For Loop Syntax
# ================================================================================

for variable in iterable:
    # Code to be executed in each iteration
```

for: *This is the keyword that indicates the start of a for loop.*

variable: *This is a variable that takes on the value of each item in the iterable during each iteration of the loop. You can choose any valid variable name.*

in: *This is the keyword used to specify that you are iterating over the elements of the iterable.*

iterable: *This is an object that you want to iterate over. It can be a list, tuple, string, dictionary, range, or any other iterable object.*

: (colon): *A colon is used to denote the beginning of the indented block of code that will be executed in each iteration.*

while Loop

```
# %%

 ================================================================================
# while Loop Syntax
# ================================================================================

while condition:
    # Code to be executed repeatedly as long as the condition is True
```

while:This is the keyword that indicates the start of a while loop.

condition:This is a Boolean expression that is evaluated before each iteration of the loop. If the condition is True, the loop continues to execute; if it's False, the loop terminates.

: (colon): *A colon is used to denote the beginning of the indented block of code that will be executed in each iteration.*

if Syntax

```
# ================================================================================
# # if syntax
# ================================================================================
if condition:
    # Code to be executed if the condition is True
```

if: This is the keyword that indicates the start of an if statement.

condition: This is a Boolean expression that is evaluated. If the condition is True, the code block indented below the if statement is executed; if it's False, the code block is skipped.

: (colon): A colon is used to denote the beginning of the indented block of code that will be executed if the condition specified in the if statement is True.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 2: Python Data Types, Basic Operators, Logical Conditions and Loops

2.1 Objectives

Manipulate Python data types, operators, logical conditions, and control loops with guided practice to handle and prepare data for AI tasks.

2.2 Pre-Lab

- Review the python data types studied in Lab Experiment 1
- Review basic loops syntax studied in Lab Experiment 1

1.3 In-Lab

2.2.1 Dictionaries

Dictionaries are one of the most useful and versatile data types in Python. They aresimilar to arrays but consist of ***key:value*** pairs. Each value stored in a dictionaryis accessed by its key, and the value can be any sort of object (string, number, list,etc.).
This allows you to create structured records. Dictionaries are placed within {}.

In-Lab Task 1

Create the dictionaries according to the following code and display results. You should use names and phone number of your friends.

```
# ================================================================================
# Data Tyeps - Dictionaries
# ================================================================================
phonebook = {}
phonebook["John"] = {"Phone": "012 794 794",
"Email": "john@email.com"}
phonebook["Jill"] = {"Phone": "012 345 345",
"Email": "jill@email.com"}
phonebook["Joss"] = {"Phone": "012 321 321",
"Email": "joss@email.com"}
print(phonebook)
%--------------------------------------------------------------------------------
```

You can iterate over a dictionary just like a list, using the dot term***.items().***

In-Lab Task 2

Use for loop to print data from the dictionary.

```
# ================================================================================
# Usig for loop to extract data from Dictionaries
# ================================================================================
for name, record in phonebook.items():
    print("{}'s phone number is {}, and  email is {}" .format(name,
          record["Phone"], record["Email"]))
#--------------------------------------------------------------------------------
```

Similarly, you add new records as shown above, and you remove records with del or pop. They each have a different effect.

In-Lab Task 3

Learn and practice the commands 'del' and 'pop' for dictionaries.

```
# ================================================================================
# First `del`
del phonebook["John"]
for name, record in phonebook.items():
    print("{}'s phone number is {}, \
        and their email is {}".format(name, record["Phone"], record["Email"]))

# Pop returns the record, and deletes it
jill_record = phonebook.pop("Jill")
print(jill_record)
for name, record in phonebook.items():
    # You can see that only Joss is still left in the system
    print("{}'s phone number is {}, \
          and their email is {}".format(name, record["Phone"], record["Email"]))

# If you try and delete a record that isn't in the dictionary,you get an error
del phonebook["John"]
#--------------------------------------------------------------------------------
```

2.2.2 Basic Operators

Operators are the various algebraic symbols (such as +, -, *, /, %, etc.). Once 'youhave learned the syntax, programming is mostly mathematics.

Arithmetic Operators

As you would expect, you can use the various mathematical operators with numbers(both integers and floats).

In-Lab Task 4

Learn and practice the arithmetic operators for python programming.

```
#%%
# ================================================================================
# Arithmatic Operators
# ================================================================================
number = 1 +2 * 3 / 4.0
# Try to predict what the answer will be .. does Python follow order
# operations hierarchy?
print(number)
# The modulo (%) returns the integer remainder of a division
remainder = 11 % 3
print(remainder)
# Two multiplications is equivalent to a power operation
squared = 7 ** 2
print(squared)
cubed = 2 ** 3
print(cubed)
```

List Operators

In-Lab Task 4

Learn and practice the list operators for python programming.

```
# ================================================================================
# List Operators
# ================================================================================
even_numbers = [2, 4, 6, 8]
# One of my first teachers in school said, "People are odd. Numbers are uneven."
# He also said, "Cecil John Rhodes always ate piles of
# unshelled peanuts in parliament in Cape Town."
# "You'd know he'd been in parliament by the huge pile of
# shells on the floor. He also never wore socks."
# "You'll never forget this." And I didn't. I have no idea if
# it's true.
uneven_numbers = [1, 3, 5, 7]
all_numbers = uneven_numbers + even_numbers
# What do you think will happen?
print(all_numbers)
# You can also repeat sequences of lists
print([1, 2 , 3] * 3)
```

```
#%%
# ================================================================================
#  Creat a project of this
# ================================================================================
x = object() # A generic Python object
y = object()
# Change this code to ensure that x_list and y_list each have
# 10 repeating objects
# and concat_list is the concatenation of x_list and y_list
x_list = [x]
y_list = [y]
concat_list = []
print("x_list contains {} objects".format(len(x_list)))
print("y_list contains {} objects".format(len(y_list)))
print("big_list contains {} objects".format(len(concat_list)))
# Test your lists
if x_list.count(x) == 10 and y_list.count(y) == 10:
    print("Almost there...")
if concat_list.count(x) == 10 and concat_list.count(y) == 10:
    print("Great!")
```

String Operators

You can do a surprising amount with operators on strings.

.

In-Lab Task 5

Print the following using string operators. This is a self-learning task.

```
Hello, World!
Hello HelloHelloHelloHelloHelloHello
```

Logical Conditions

In the section on Indentation, you were introduced to the if statement and the setofboolean operators that allow you to test different variables against each other.To that list of Boolean operators are added a new set of comparisons: and, or,and in.

In-Lab Task 6

Learn and practice the Boolean operators for python programming.

```
# ================================================================================
# # Simple boolean tests
# ================================================================================

x = 2
print(x == 2)
print(x == 3)
print(x < 3)
# Using `and`
name = "John"
print(name == "John" and x == 2)
# Using `or`
print(name == "John" or name == "Jill")
# Using `in` on lists
print(name in ["John", "Jill", "Jess"])
```

These can be used to create nuanced comparisons using if. You can use a seriesof comparisons with if, elseif, and else.Remember that code must be indented correctly, or you will get unexpectedbehavior.

```
# Unexpected results
x = 2
if x > 2:
    print("Testing x")
    print("x > 2")
# Formated correctly
if x == 2:
    print("x == 2")
```

In-Lab Task 7

Demonstrate complex if-else conditions for python programming and also explore the ‘not’ and

'is'.

```
#%%
# ============================================================================
# Demonstrating more complex if tests
# ============================================================================

x = 2
y = 10
if x > 2:
    print("x > 2")
elif x == 2 and y > 50:
    print("x == 2 and y > 50")
elif x < 10 or y > 50:
# But, remember, you don't know WHICH condition was True
    print("x < 10 or y > 50")
else:
    print("Nothing worked.")
```

Two special cases are not and is.

10. not is used to get the opposite of a particular Boolean test, e.g., not(False) returns True.
11. is would seem, superficially, to be like = =, but it tests whether the actual objects are the same, not whether the values which the objects reflect are equal.

A quick demonstration:

```
#%%
# Using `not`
name_list1 = ["John", "Jill"]
name_list2 = ["John", "Jill"]
print(not(name_list1 == name_list2))
# Using `is`
name2 = "John"
print(name_list1 == name_list2)
print(name_list1 is name_list2)
```

2.2.3 1.3.3 Loops

Loops iterate over a given sequence, and—here—it is critical to ensure yourindentation is correct or 'you will get unexpected results for what is consideredinside or outside the loop.

12. For loops, for, which loop through a list. There is also some new syntax to use in for loops:
    – In Lists you saw enumerate, which allows you to count the loop number.

    – Range creates a list of integers to loop, range(start, stop) creates alist of integers between start and stop, or range(num) creates a zero-basedlist up to num, or range(start, stop, step) steps through a list inincrements of step.

13. While loops, while, which execute while a particular condition is True. And some new syntax for while is:
– while is a conditional statement (it requires a test to return True), whichmeans we can use else in a while loop (but not for)

In-Lab Task 8

Demonstrate the reading and writing different types of data in python using For Loop.

```
# ================================================================================
# Loops
# ================================================================================
# For Loop for reading Writing Data
# Sample list of numeric data
numeric_data = [10, 20, 30, 40, 50]

# Using a for loop to read and process each element
for number in numeric_data:
    result = number * 2  # Perform some operation (e.g., multiplication)
    print(result)  # Print the result


# Sample string
text = "Hello, World!"

# Using a for loop to read and print each character in the string
for char in text:
    print(char)

# Using a for loop to write characters to a new string with some modifications
new_text = ""
for char in text:
    if char.isalpha():
        new_text += char.upper()  # Convert letters to uppercase
    else:
        new_text += char  # Keep non-letter characters as they are

print(new_text)

# Writing Numeric Data
# Initialize an empty list to store numeric data
numeric_data = []

# Use a for loop to add numbers from 1 to 10 into the list
for i in range(1, 11):  # This loop iterates from 1 to 10 (inclusive)
    numeric_data.append(i)  # Add each number to the list

# Print the resulting list
print(numeric_data)
```

In-Lab Task 9

Demonstrate the application of While Loop.

```python
#%%

# While Loop

# 1. for Numeric Daat
# Using a while loop to print numbers from 1 to 5
count = 1
while count <= 5:
    print(count)
    count += 1

# 2. For Strings

# Using a while loop to print each character of a string
text = "Hello"
index = 0
while index < len(text):
    print(text[index])
    index += 1


# 3. For Dictionaries

# Using a while loop to iterate through a dictionary and print key-value pairs
student_grades = {"Alice": 92, "Bob": 85, "Charlie": 78}
keys = list(student_grades.keys())  # Get the keys as a list
index = 0
while index < len(keys):
    key = keys[index]
    value = student_grades[key]
    print(f"{key}: {value}")
    index += 1
```

2.3 1.3 Post-Lab

Post-Lab Task

Your task is to create a Python program that manages student grades of at least seven (07) students and performs several operations. Your program should do the following:

1. Calculate and display the average grade for a list of students along with the list students and corresponding grades.
2. Categorize each student's grade into one of the following categories: "Excellent," "Very Good," "Good," or "Needs Improvement" based on the grade ranges (Your own assumptions).

3. Allow the user to search for a specific student's grade by entering the student's name. The program should repeatedly prompt the user for a name until a valid student name is entered, and then display the corresponding grade.

Provide the Python program that accomplishes these tasks, ensuring that it includes relevant data**, if-else conditions**, and both **for** and **while** loops.

Implement the program and demonstrate its functionality.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 3: Learning Functions and Classes in Python

3.1 Objectives

Practice developing reusable code by encapsulating tasks in functions and organizing them into basic classes for structuring simple AI algorithms.

3.2 Pre-Lab

The code from the previous lab experiments was limited to shortsnippets. Solving more complex problems means more complex code - stretchingover hundreds, to thousands, of lines, and—if you want to reuse that code—it is notconvenient to copy and paste it multiple times. Worse, any error is magnified, andany changes become tedious to manage.

In Python, “functions” are complete suite of functions grouped around a set of related tasks is called a library or module. Libraries permit you to inherit a wide variety of powerful software solutions developed and maintained by other people.

Functions in Python are defined using the def keyword, and they play a crucial role in organizing code, promoting reusability, and improving readability. Here are key aspects and concepts related to functions in Python:

14. Function Definition:
    Functions are defined using the “def” keyword, followed by the function name and a pair of parentheses. Any parameters the function takes are listed inside the parentheses.

```
def greetings(name):
    print(f"Hello, {name}!")

greetings("Alice")
```

15. Function Call:
To execute a function, you "call" it by using its name followed by parentheses. If the function takes parameters, you provide the values inside the parentheses.

```
def add(a, b):
    return a + b

result1 = add(3, 4)
```

16. Return Statement:
Functions can return a value using the **return** keyword. The returned value can be assigned to a variable or used directly.

```
def multiply(x, y):
    return x * y

result2 = multiply(5, 6)
```

17. Parameters and Arguments:
Parameters are variables listed in a function's definition. Arguments are the values passed to the function when it is called.

18. Default Parameters:
You can provide default values for parameters. If a value is not passed for a parameter, the default value is used.

```
def power(base, exponent=2):
    return base ** exponent

result4 = power(3)  # Uses default exponent of 2
```

19. Variable-Length Arguments:
Functions can accept a variable number of arguments using ***args** and ****kwargs**.

```
def sum_all(*numbers):
    return sum(numbers)

result5 = sum_all(1, 2, 3, 4)
```

20. Lambda Functions:
Also known as anonymous functions, these are small, one-line functions defined using the **lambda** keyword.

```
square = lambda x: x ** 2
result6 = square(5)
```

- **Scope:**

  Variables defined inside a function have local scope, meaning they are only accessible within that function. Variables defined outside functions have global scope.

```python
global_var = 10

def my_function():
    local_var = 5
    print(global_var)  # Accessible
    print(local_var)   # Accessible
```

3.3 In-Lab

In-Lab Task 1

Write a Python function called “count_even_numbers” that takes a list of integers as input and returns the count of even numbers in the list. Additionally, ensure that the function handles the case where the input list is empty. Follow the following instructions:

21. Define a function named “count_even_numbers” that takes one parameter: number_list (a list of integers).
22. Inside the function, use a loop to iterate through each number in the list.
23. Use a conditional statement to check if each number is even.
24. If a number is even, increment a counter variable.
25. After the loop, return the counter variable.

```python
# ================================================================================
# In - Lab Task 1
# ================================================================================
# Function definition
def count_even_numbers(number_list):
    # Initialize counter variable
    even_count = 0

    # Loop through each number in the list
    for num in number_list:
        # Check if the number is even
        if num % 2 == 0:
            even_count += 1

    # Return the count of even numbers
    return even_count

# Test cases
numbers1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numbers2 = [11, 13, 15, 17]
numbers3 = []  # Empty list

# Display results
print(count_even_numbers(numbers1))  # Expected output: 5
print(count_even_numbers(numbers2))  # Expected output: 0
print(count_even_numbers(numbers3))  # Expected output: 0
```

In-Lab Task 2

Write a Python function calculate_gpa that takes a list of dictionaries, where each dictionaryrepresents the marks of a student in 5 different courses (out of 100). Each dictionary should have keys 'name' and 'marks' (a list of 5 marks)

* Percentage Obtained in a Semester System	Grade	Grade Points
85 and above	A	4.00
80 84	A-	3.66
75 79	B+	3.33
71 - 74	B	3.00
68 70	B-	2.66
64 67	C+	2.33
61 63	C	2.00
58 60	C-	1.66
54 - 57	D+	1.30
50 53	D	1.00
Below 50	F	0.00

Calculate the Grade Point Average (GPA) for each student by averaging their Grade Points.

The function should return a list of dictionaries where each dictionary has keys 'name', 'grades' (a list of course grades), 'grade_points', and 'gpa'.

 Create a dictionary which have at least record to five students (Name and Scores in 5 Courses)

3.3.1 Class

In Python, a class is a blueprint for creating objects. Objects are instances of a class, and each object can have attributes (characteristics) and methods (functions) associated with it. Classes provide a way to bundle data and functionality together. Here's a basic explanation of how classes work in Python:

26. Class Declaration: To create a class, you use the class keyword. Here's a simple example of a class:

```
class Car:
    pass  # The 'pass' keyword is a placeholder for the class body
```

27. Attributes: You can define attributes (characteristics) inside the class. These attributes represent the data associated with objects of the class.

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model
```

In this example, the **__init__** method is a special method called a constructor. It is called when an object is created. **self** refers to the instance of the class (the object), and you can set attributes like **make** and **model** for each object.

28. Methods: You can also define methods inside the class. Methods are functions associated with the class.

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"{self.make} {self.model}")
```

The **display_info** method can be called on an instance of the **Car** class to display information about the car.

29. Creating Objects (Instances): Once a class is defined, you can create objects (instances) of that class.

```
my_car = Car("Toyota", "Camry")
```

This creates an instance of the **Car** class called **my_car** with the specified make and model.

30. Accessing Attributes and Calling Methods: You can access attributes and call methods using the dot notation.

```
my_car = Car("Toyota", "Camry")


print(my_car.make)  # Output: Toyota
my_car.display_info()  # Output: Toyota Camry
```

Here, **my_car.make** accesses the **make** attribute, and **my_car.display_info()** calls the **display_info** method.

In-Lab Task 3

Create a Python class named **student** to manage student information. The class should have the following attributes:

1. **name**: Name of the student.
2. **roll_number**: Roll number of the student.
3. **marks**: A list to store the marks of the student in different subjects.

The class should have the following methods:

1. **__init__**: A constructor to initialize the attributes.
2. **add_marks**: A method to add marks to the **marks** list.
3. **calculate_average**: A method to calculate and return the average marks of the student.

Create an instance of the **student** class, add some marks, and calculate the average.

3.4 Post-Lab

Post-Lab Task

Create a simple library management system using Python. Define a class **Book** to represent a book with attributes such as title, author, and availability status. Implement functions to borrow and return books.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 4: Data Cleaning

4.1 Objectives

Perform basic data cleaning techniques such as removing outliers, handling missing values, and applying feature scaling with instructor guidance.

4.2 Pre-Lab

Required Tools & Libraries

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Dataset

Use the **"Housing Prices"** dataset (or any dataset with missing values and outliers):

- Download: https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

4.3 In-Lab

In-Lab Task 1: Load and Inspect Data

```
# Load dataset
df = pd.read_csv("housing.csv")

# Check first 5 rows
print(df.head())

# Check for missing values
print("\nMissing Values:\n", df.isnull().sum())

# Check data statistics
print("\nData Statistics:\n", df.describe())
```

In-Lab Task 2: Handle Missing Values

Option 1: Drop Missing Values

```
df_clean = df.dropna()  # Drops all rows with missing values
```

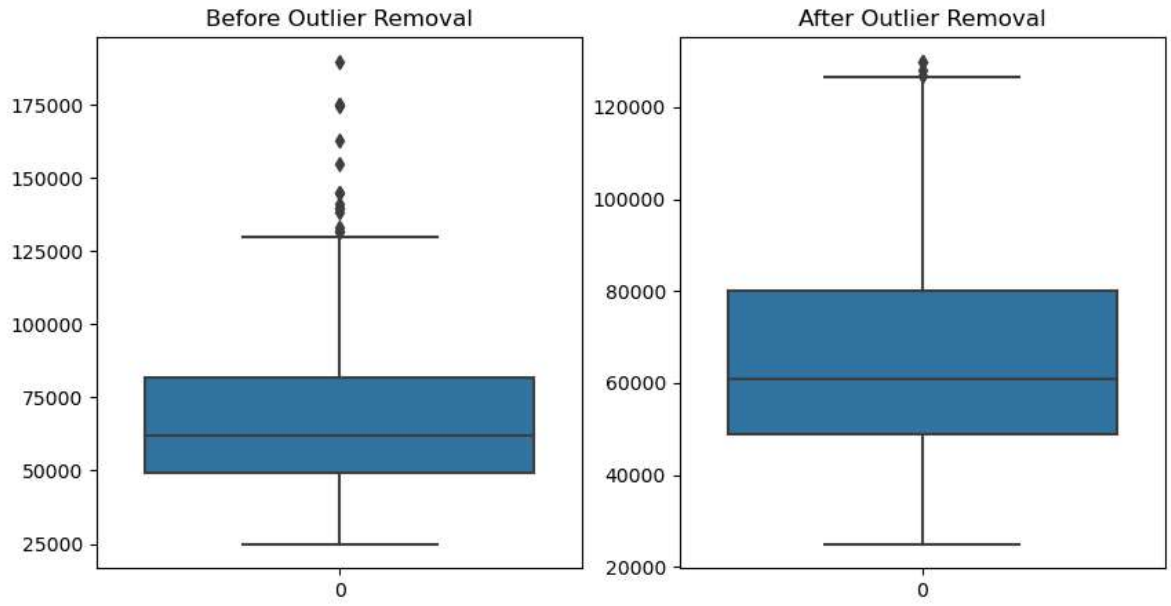
Option 2: Impute Missing Values (Mean/Median/Mode)

```
# Impute numerical columns with mean
imputer = SimpleImputer(strategy='mean')
df[['LotArea', 'SalePrice']] = imputer.fit_transform(df[['LotArea', 'SalePrice']])

# Impute categorical columns with mode
df['GarageType'] = df['GarageType'].fillna(df['GarageType'].mode()[0])
```

In-Lab Task 3: Detect and Remove Outliers

```
Q1 = df['SalePrice'].quantile(0.25)
Q3 = df['SalePrice'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
df_no_outliers = df[(df['SalePrice'] >= lower_bound) & (df['SalePrice'] <= upper_bound)]

# Visualize before & after
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.boxplot(df['SalePrice'])
plt.title("Before Outlier Removal")

plt.subplot(1, 2, 2)
sns.boxplot(df_no_outliers['SalePrice'])
plt.title("After Outlier Removal")
plt.show()
```

NOTE: Adjust the variable name errors (if occur)



In-Lab Task 3:Apply Feature Scaling

```
scaler = StandardScaler()
df[['LotArea', 'SalePrice']] = scaler.fit_transform(df[['LotArea', 'SalePrice']])
```

4.4 Post-Lab

31. What happens if we don't handle missing values before training a model?
32. When should we use standardization vs. min-max scaling?
33. Does removing outliers always improve model performance? Why or why not?

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 5: Linear Regression

5.1 Objectives

Execute Linear Regression models in Python, replicating steps to understand simple predictive modelling in AI.

5.2 Pre-Lab

5.2.1.1 Linear Regression

Linear regression is a basic predictive analytics technique that uses historical data topredict an output variable. It is popular for predictive modeling because it is easilyunderstood and can be explained using plain English.

The basic idea is that if we can fit a linear regression model to observed data, wecan then use the model to predict any future values. For example, let us assume thatwe have found from historical data that the price (P) of a house is linearly dependentupon its size (S)—in fact, we found that a house's price is exactly 90 times its size.

The equation will look like this: P = 90*S.

With this model, we can then predict the cost of any house. If we have ahouse that is 1,500 square feet, we can calculate its price to be: P = 90*1500= $135,000

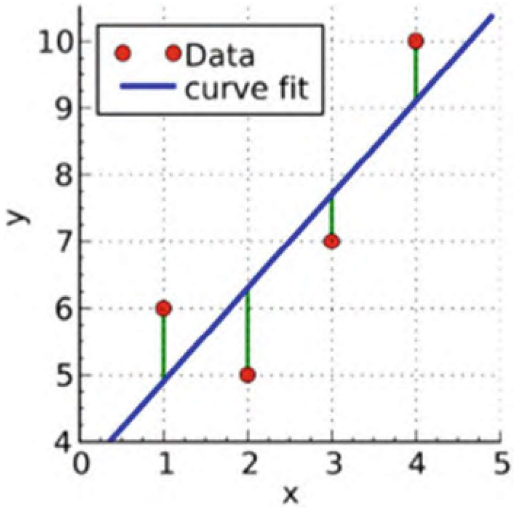
There are two kinds of variables in a linear regression model:

34. The input or predictor variable is the variable(s) that help predict the value of theoutput variable. It is commonly referred to as X.
35. The output variable is the variable that we want to predict. It is commonlyreferred to as Y.

To estimate Y using linear regression, we assume the equation: $Ye = \alpha + \beta X$where Ye is the estimated or predicted value of Y based on our linear equation. Ourgoal is to find statistically significant values of the parameters α and β that minimizethe difference between Y and Ye. If we are able to determine the optimum values ofthese two parameters, then we will have the line of best fit that we can use to predictthe values of Y, given the value of X. So, how do we estimate α and β? We can usea method called ordinary least squares.

The objective of the least squares method is to find values of α and β thatminimize the sum of the squared difference between Y and Ye.

Here, we notice that when E increases by 1, our Y increases by 0.175399. Also,when C increases by 1, our Y falls by 0.044160.



In this experiment and onwards following data set will be utilized:

Dataset: "Alumni Giving Regression (Edited).csv"

Following is the link to download dataset:

https://www.dropbox.com/s/veak3ugc4wj9luz/Alumni%20Giving%20Regression%20%28Edited%29.csv

5.3 In-Lab

In-Lab Task 1

Import the following libraries for Linear Regression. All of these should be properly working.

```
# Importing libraries needed
# Note that keras is generally used for deep learning as well
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn import linear_model
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import pandas as pd
import csv

import matplotlib.pyplot as plt
```

In-Lab Task 2

Import Data Set

```
# ========================================================================
# Read Data and fix seed
# ========================================================================
# fix random seed for reproducibility
np.random.seed(7)
df = pd.read_csv("Alumni Giving Regression (Edited).csv", delimiter="," )
dd_df_1=df.head()
```

In-Lab Task 3

Write code to visualize the Data Pattern and Description of Data

Data Pattern

	A	B	C	D	E	F
0	24	0.42	0.16	0.59	0.81	0.08
1	19	0.49	0.04	0.37	0.69	0.11
2	18	0.24	0.17	0.66	0.87	0.31
3	8	0.74	0.00	0.81	0.88	0.11
4	8	0.95	0.00	0.86	0.92	0.28

Description of Data

	A	B	C	D	E	F
count	123.000000	123.000000	123.000000	123.000000	123.000000	123.000000
mean	17.772358	0.403659	0.136260	0.645203	0.841138	0.141789
std	4.517385	0.133897	0.060101	0.169794	0.083942	0.080674
min	6.000000	0.140000	0.000000	0.260000	0.580000	0.020000
25%	16.000000	0.320000	0.095000	0.505000	0.780000	0.080000
50%	18.000000	0.380000	0.130000	0.640000	0.840000	0.130000
75%	20.000000	0.460000	0.180000	0.785000	0.910000	0.170000
max	31.000000	0.950000	0.310000	0.960000	0.980000	0.410000

In-Lab Task 4

Compute the correlation of data and understand the relationship between variables.

The term “correlation” refers to a mutual relationship or association betweenquantities (numerical number). In almost any business, it is very helpful to expressone quantity in terms of its relationship with others. We are concerned aboutthis because business plans and departments are not isolated! For example, salesmight increase when the marketing department spends more on advertisements,or a customer’s average purchase amount on an online site may depend on hisor her characteristics. Often, correlation is the first step to understanding theserelationships and subsequently building better business and statistical models.

Compute the correlation of the data set using the following code and understand the meaning of correlation results.

```
# ========================================================================
# Commpute Correlation
# ========================================================================
corr=df.corr(method ='pearson')
corr
print (corr)
```

"D" and "E" have a strong correlation of 0.93, which means thatwhen D moves in the positive direction E is likely to move in that direction too.Here, notice that the correlation of `A` and `A` is 1. Of course, `A` would be perfectlycorrelated with `A`.

	A	B	C	D	E	F
A	1.000000	-0.691900	0.414978	-0.604574	-0.521985	-0.549244
B	-0.691900	1.000000	-0.581516	0.487248	0.376735	0.540427
C	0.414978	-0.581516	1.000000	0.017023	0.055766	-0.175102
D	-0.604574	0.487248	0.017023	1.000000	0.934396	0.681660
E	-0.521985	0.376735	0.055766	0.934396	1.000000	0.647625
F	-0.549244	0.540427	-0.175102	0.681660	0.647625	1.000000

In-Lab Task 5

Splitting of Data into Train and Test Segments

In general, we would need to test our model. `train_test_split`is a function in `Sklearn`model selection for splitting data arrays into two subsets for training data and for testing data. With this function, you do not need to divide the dataset manually. You can use from the function `train_test_split`using the following code `sklearn.model_selection import train_test_split`.

By default, Sklearntrain_test_split will make random partitions for the two subsets. However, you can also specify a random state for the operation. Here, takes note that we will need to pass in the `X` and `Y` to the function. `X` refers to the `features` while `Y` refers to the `target` of the dataset.

```
# ================================================================================
# Train/ Test Split
Y_POSITION = 5

# ================================================================================
# Here, Y_POSITION is set to 5, indicating that the target variable is located
# in the 5th column of the DataFrame. model_1_features is then created as a l
# ist containing the indices [0, 1, 2, 3, 4], representing the columns from 0
# to 4 (excluding 5).
# ================================================================================
```

```
# ================================================================================

model_1_features = [i for i in range(0,Y_POSITION)]

# ================================================================================
# This code extracts the features (X) and the target variable (Y) from the
# DataFrame (df). The features are obtained by selecting all rows (:) and the
# columns specified in model_1_features. The target variable (Y) is obtained by
# selecting all rows and the column specified by Y_POSITION.
# ================================================================================

# x refers to as features of the data
X = df.iloc[:,model_1_features]

# target data
Y = df.iloc[:,Y_POSITION]
# create model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,
                                                    random_state=2020)

# ================================================================================
# This line uses the train_test_split function from scikit-learn to split the
# data into training and testing sets. The arguments are:

# X: Features.
# Y: Target variable.
# test_size: The proportion of the dataset to include in the test split (here,
# 20%).
# random_state: Seed for the random number generator to ensure reproducibility.
# The function returns four sets:

# X_train: Training features.
# X_test: Testing features.
# y_train: Training target variable.
# y_test: Testing target variable.
#
```

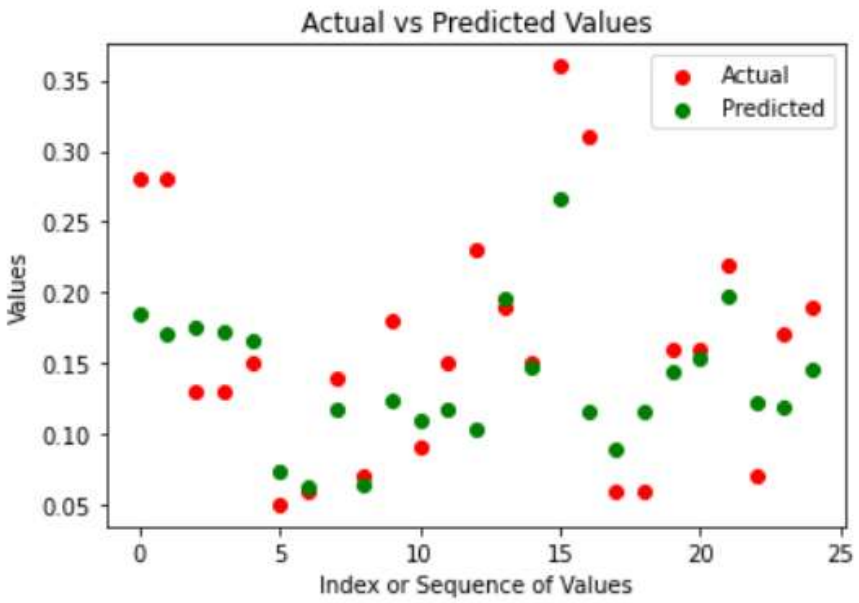
In-Lab Task 4

Perform Linear Regression and display result.

```
# ================================================================================
# #Model 1 : linear regression

#  This line creates an instance of the linear regression model using
# scikit-learn's LinearRegression class.
model1 = linear_model.LinearRegression()

# This line trains the linear regression model (model1) using the training
# features (X_train) and the corresponding target variable (y_train).
model1.fit(X_train, y_train)

# This line uses the trained model to make predictions on the training
# set (X_train) and stores the predicted values in y_pred_train1.
y_pred_train1 = model1.predict(X_train)
```

```
# The Root Mean Squared Error (RMSE) is a measure of the average magnitude of
# the errors between predicted and actual values. This line calculates and
# prints the RMSE for the training set.
print("Regression")
print("================================")
RMSE_train1 = mean_squared_error(y_train,y_pred_train1)

# This line uses the trained model to make predictions on the testing
# set (X_test) and stores the predicted values in y_pred1.
print("Regression Train set: RMSE {}".format(RMSE_train1))
print("================================")
y_pred1 = model1.predict(X_test)
RMSE_test1 = mean_squared_error(y_test,y_pred1)

print("Regression Test set: RMSE {}".format(RMSE_test1))
print("================================")
coef_dict = {}
for coef, feat in zip(model1.coef_,model_1_features):
    coef_dict[df.columns[feat]] = coef
print(coef_dict)
```

```
x_values = np.arange(len(y_test))
# Scatter plot of actual values (y_test) in red
plt.scatter(x_values, y_test, color='red', label='Actual')
# Scatter plot of predicted values (y_pred) in green
plt.scatter(x_values, y_pred1, color='green', label='Predicted')
plt.xlabel('Index or Sequence of Values')
plt.ylabel('Values')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()
```



Use the results of the model coefficients to predict a value on paper and verify using code.

5.4 Post Lab

Post-Lab Task

Based on the linear regression analysis you performed, what steps would you take to improve the model's predictive accuracy, and how would you validate the model's performance on unseen data?

Select a data set from freely available datasets on internet and provide implementation of your suggested improvements.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 6: Decision Tree Regression and Random Forest

6.1 Objectives

Apply Decision Tree Regression and Random Forest algorithms in Python through step-by-step guidance, gaining familiarity with tree-based AI models.

6.2 Pre-Lab

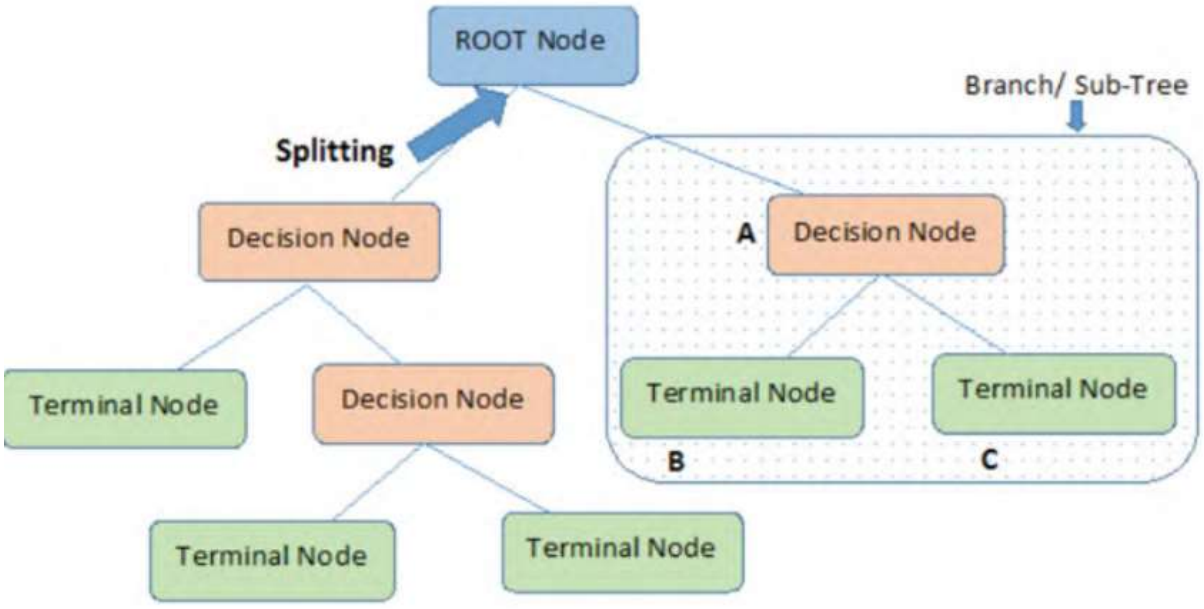
6.2.1 Decision Tree Regression

A decision tree is arriving at an estimate by asking a series of questions to the data,each question narrowing our possible values until the model gets confident enoughto make a single prediction. The order of the question and their content are beingdetermined by the model. In addition, the questions asked are all in a True/Falseform.

This is a little tough to grasp because it is not how humans naturally think, andperhaps the best way to show this difference is to create a real decision tree from. Inthe above problem x1, x2 are two features that allow us to make predictions for thetarget variable y by asking True/False questions.

The decision of making strategic splits heavily affects a tree's accuracy. Thedecision criteria are different for classification and regression trees. Decision treesregression normally use mean squared error (MSE) to decide to split a node intotwo or more sub-nodes. Suppose we are doing a binary tree; the algorithm will firstpick a value and split the data into two subsets. For each subset, it will calculate theMSE separately. The tree chooses the value with results in smallest MSE value.

Let us examine how is Splitting Decided for Decision Trees Regressor in moredetail. The first step to create a tree is to create the first binary decision.



1. We need to pick a variable and the value to split on such that the two groups are as different from each other as possible.
2. For each variable, for each possible value of the possible value of that variable see whether it is better.
3. Take weighted average of two new nodes (mse*num_samples).

To sum up, we now have:

36. A single number that represents how good a split is, which is the weighted average of the mean squared errors of the two groups that create.
37. A way to find the best split, which is to try every variable and to try every possible value of that variable and see which variable and which value gives us a split with the best score.
38.

Training of a decision tree regressor will stop when some stopping condition ismet:

1. When you hit a limit that was requested (for example: `max_depth`).
2. When your leaf nodes only have one thing in them (no further split is possible, MSE for the train will be zero but will overfit for any other set—not a useful model).

6.2.2 Random Forest

What is a Random Forest? And how does it differ from a Decision Tree?The fundamental concept behind random forest is a simple but powerful one—the wisdom of crowds. In data science speak, the reason that the random forestmodel works so well is: A large number of relatively uncorrelated models (trees)operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments withlow correlations (like stocks and bonds) come together to form a portfolio thatis greater than the sum of its parts, uncorrelated models can produce ensemblepredictions that are more accurate than any of the individual predictions. The reasonfor this wonderful effect is that the trees protect each other from their individualerrors (as long as they do not constantly all err in the same direction). While sometrees may be wrong, many other trees will be right, so as a group the trees are ableto move in the correct direction. So the prerequisites for random forest to performwell are:

1. There needs to be some actual signals in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

So how does random forest ensure that the behavior of each individual tree is nottoo correlated with the behavior of any of the other trees in the model? It uses thefollowing two methods:

Bagging (Bootstrap Aggregation)—Decision trees are very sensitive to the datathey are trained on—small changes to the training set can result in significantlydifferent tree structures. Random forest takes advantage of this by allowing eachindividual tree to randomly sample from the dataset with replacement, resultingin different trees. This process is known as bagging.

Feature Randomness—In a normal decision tree, when it is time to split a node,we consider everypossible feature and pick the one that produces the mostseparation between the observations in the left node vs. those in the right node.In contrast, each tree in a random forest can pick only from a random subsetof features. This forces even more variation among the trees in the model andultimatelyresults in lower correlation across trees and more diversification.

As Random Forest is actually a collection of Decision Trees, this makes thealgorithm slower and less effective for real-time predictions. In general, RandomForest can be fast to train, but quite slow tocreate predictions once they are trained.

This is due to the fact that it has to run predictions on each individual tree and thenaverage their predictions to create the final prediction.Each individual tree in the random forest splits out a class prediction and the classwith the most votes become our model's prediction. Decision Trees do suffer fromoverfitting while Random Forest can prevent overfitting resulting in better prediction most of the time.

6.3 In-Lab

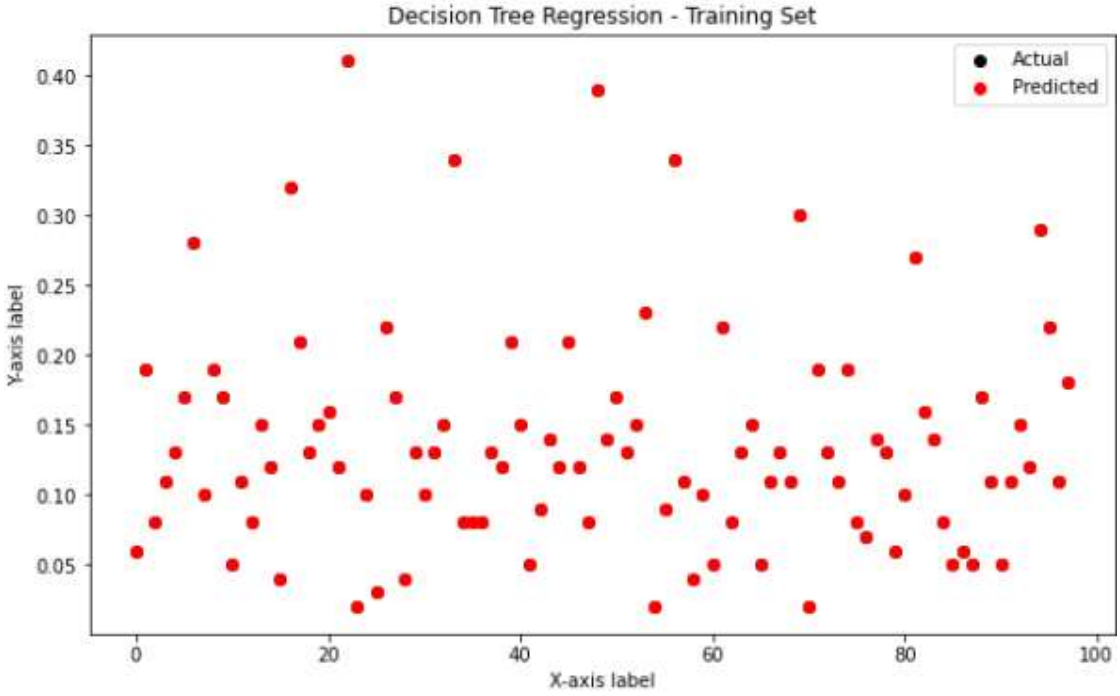
In-Lab Task 1

Write the following decision tree implementation in python.

```
# ================================================================================
# Model 2 decision tree
# ================================================================================

model2 = tree.DecisionTreeRegressor()
model2.fit(X_train, y_train)
print("Decision Tree")
print("==============================")
y_pred_train2 = model2.predict(X_train)
RMSE_train2 = mean_squared_error(y_train,y_pred_train2)
print("Decision Tree Train set: RMSE {}".format(RMSE_train2))

y_pred_test2 = model2.predict(X_test)
RMSE_test2 = mean_squared_error(y_test,y_pred_test2)
print("Decision Tree Test set: RMSE {}".format(RMSE_test2))
print("==============================")
```

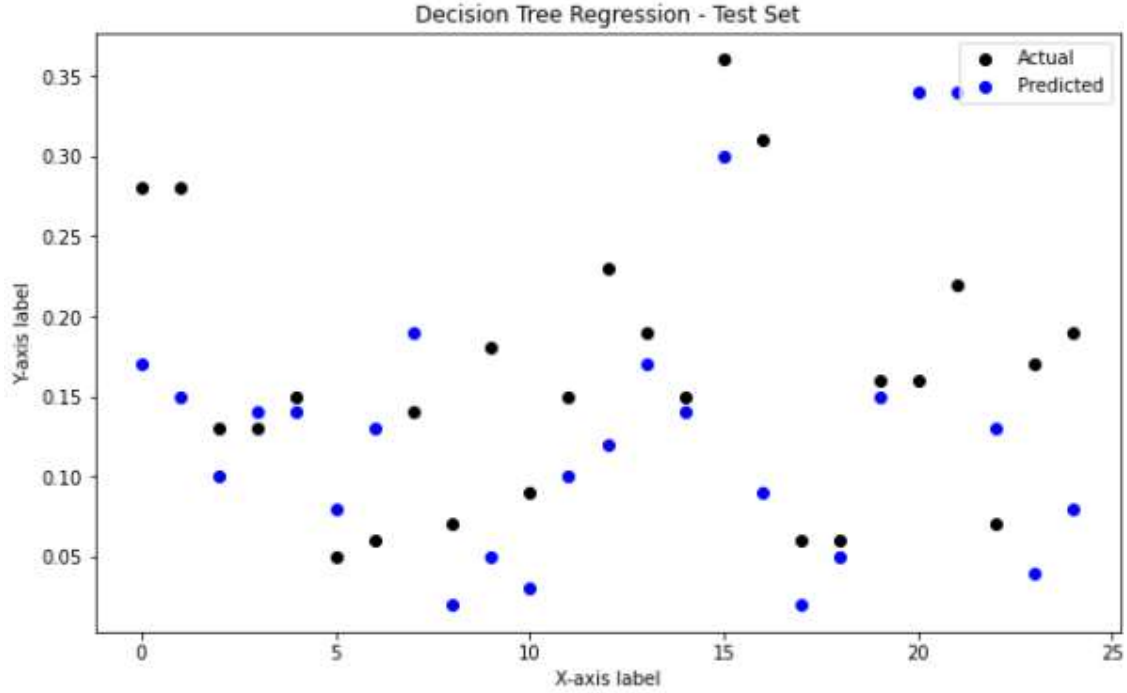
In-Lab Task 2

Display the results of Decision Tree Regression for the Training set.



In-Lab Task 3

Display the results of Decision Tree Regression for the Test set.



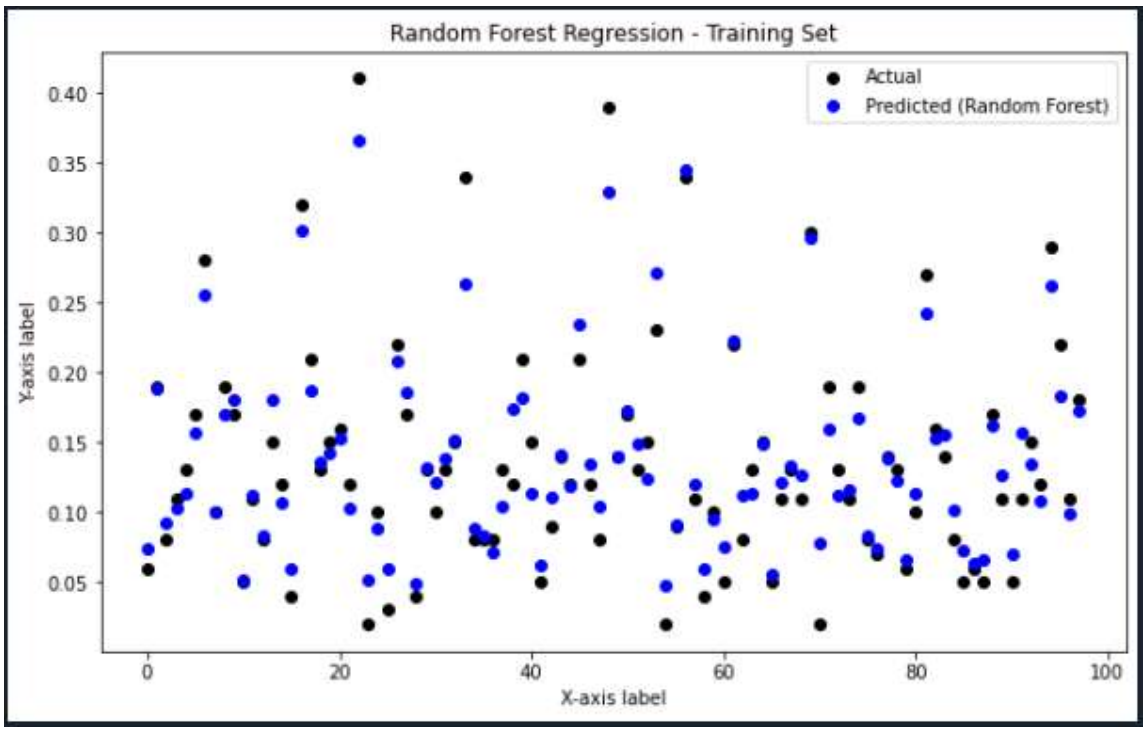
In-Lab Task 4

Implement the Random Forest using Python. Understand and discuss the results.

```
# ================================================================================
# Model 3 Random Forest
# ================================================================================

model3 = RandomForestRegressor()
model3.fit(X_train, y_train)
print("Random Forest Regressor")
print("================================")
y_pred_train3 = model3.predict(X_train)
RMSE_train3 = mean_squared_error(y_train,y_pred_train3)
print("Random Forest Regressor TrainSet: RMSE {}".format(RMSE_train3))
print("================================")
y_pred_test3 = model3.predict(X_test)
RMSE_test3 = mean_squared_error(y_test,y_pred_test3)
print("Random Forest Regressor TestSet: RMSE {}".format(RMSE_test3))
print("================================")
```

In-Lab Task 5

Display the result of the model and discuss the results



6.4 Post Lab

Post-Lab Task

Compare the performance of the three models. Which model performs better on the test set, and why do you think it outperforms the others?

Discuss on the strengths and limitations of each model. Discuss any insights gained from the visualizations.

Suggest potential strategies to improve the models' performance. Could feature engineering or hyperparameter tuning enhance the predictions?

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 7: Neural Networks

7.1 Objectives

Attempt to implement a simple Neural Network using guided examples to grasp foundational concepts and architecture in neural modelling.

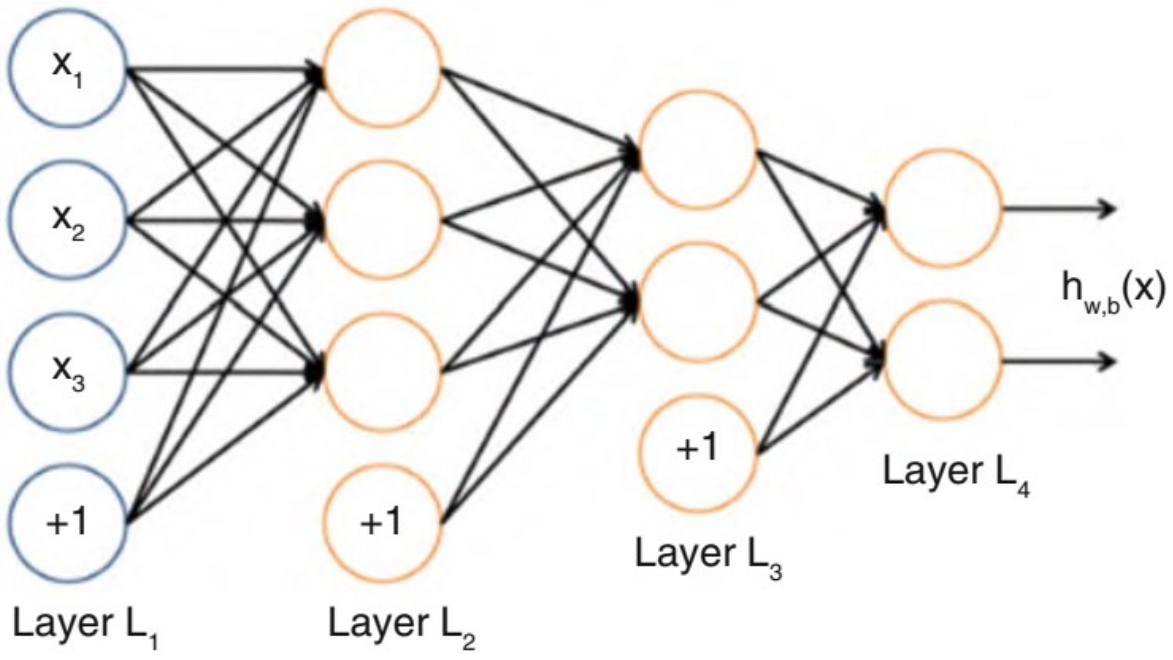
7.2 Pre-Lab

7.2.1 Neural Networks

Neural networks represent deep learning using artificial intelligence. Certain application scenarios are too heavy or out of scope for traditional machine learning algorithms to handle. As they are commonly known, Neural Network pitches in such scenarios and fills the gap.

Neural networks are the representation we make of the brain: neurons interconnected to other neurons, which forms a network. A simple information transits in a lot of them before becoming an actual thing, like “move the hand to pick up thispencil.”

The operation of a complete neural network is straightforward: one enters variables as inputs (for example, an image if the neural network is supposed to tellwhat is on an image), and after some calculations, an output is returned (probabilityof whether an image is a cat).

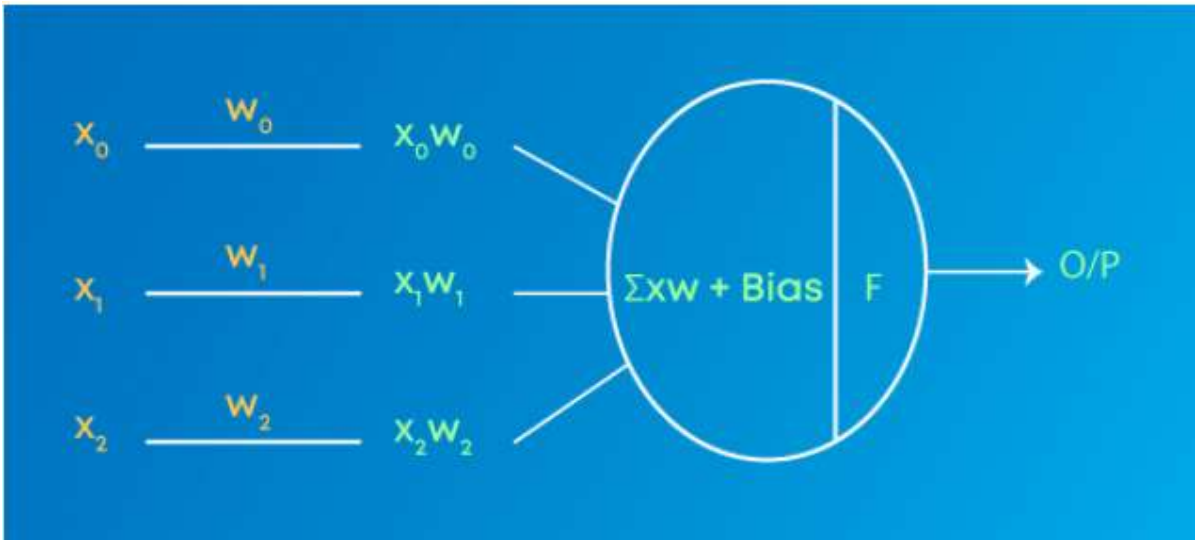


When an input is given to the neural network, it returns an output. On the first try, it cannot get the right output by its own (except with luck) and that is why, during the learning phase, every input comes with its label, explaining what output the neural network should have guessed. If the choice is the good one, actual parameters are kept, and the next input is given. However, if the obtained output

does not match the label, weights are changed. Those are the only variables that can be changed during the learning phase. This process may be imagined as multiple buttons that are turned into different possibilities every time an input is not guessed correctly. To determine which weight is better to modify, a particular process, called "backpropagation" is done.

7.2.2 Components of Neural Networks

39. **Weights** are numeric values that are multiplied by inputs. In backpropagation, they are modified to reduce the loss. In simple words, weights are machine learned values from Neural Networks. They self-adjust depending on the difference between predicted outputs vs training inputs.
40. **Activation Function** is a mathematical formula that helps the neuron to switch ON/OFF.



41. **The input layer** represents dimensions of the input vector.
42. **The hidden layer** represents the intermediary nodes that divide the input space into regions with (soft) boundaries. It takes in a set of weighted input and produces output through an activation function.
43. **Output layer** represents the output of the neural network.

7.2.3 Types of neural networks models are listed below:

The nine types of neural networks are:

44. Perceptron
45. Feed Forward Neural Network
46. Multilayer Perceptron
47. Convolutional Neural Network
48. Radial Basis Functional Neural Network
49. Recurrent Neural Network
50. LSTM – Long Short-Term Memory

51. Sequence to Sequence Models
52. Modular Neural Network

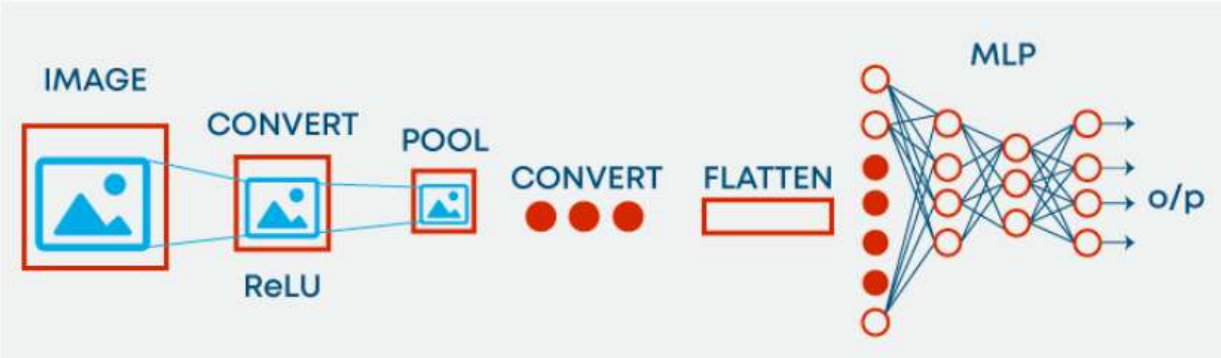
7.2.4 Convolutional Neural Network

Convolution neural network contains a three-dimensional arrangement of neurons instead of the standard two-dimensional array. The first layer is called a convolutional layer. Each neuron in the convolutional layer only processes the information from a small part of the visual field. Input features are taken in batch-wise like a filter. The network understands the images in parts and can compute these operations multiple times to complete the full image processing. Processing involves conversion of the image from RGB or HSI scale to grey scale. Furthering the changes in the pixel value will help to detect the edges and images can be classified into different categories.

These layers perform operations that alter the data with the intent of learning features specific to the data. Three of the most common layers are convolution, activation or ReLU, and pooling.

53. **Convolution** puts the input images through a set of convolutional filters, each of which activates certain features from the images.
54. **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as *activation*, because only the activated features are carried forward into the next layer.
55. **Pooling** simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn.

These operations are repeated over tens or hundreds of layers, with each layer learning to identify different features.



7.2.4.1 Advantages of Convolution Neural Network:

56. Used for deep learning with few parameters.
57. Less parameters to learn as compared to fully connected layer.

7.2.4.2 Disadvantages of Convolution Neural Network:

58. Comparatively complex to design and maintain.
59. Comparatively slow [depends on the number of hidden layers]

7.3 In-Lab

7.3.1 Implementation of CNN using Python

7.3.1.1 Import Necessary Libraries:

```
# Import necessary libraries
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.datasets import mnist
from keras.utils import to_categorical
```

7.3.1.2 Load and Preprocess Data:

The MNIST dataset is loaded using. mnist.load_data().

```
# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

Images are reshaped to have a single channel (grayscale) and normalized to values between 0 and 1.

```
# One-hot encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

7.3.1.3 Build the CNN Model:

The model is created as a sequential stack of layers.

- Convolutional layers (Conv2D) are added with ReLU activation.
- Max pooling layers (MaxPooling2D) are added to down-sample the spatial dimensions.
- A flattening layer (Flatten) is added to convert the 2D feature maps to a vector.
- Dense (fully connected) layers are added with ReLU activation.
- The output layer uses softmax activation for multi-class classification.

```
# Build the CNN model
model = Sequential()

# Step 1: Convolutional Layer with ReLU activation
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

# Step 2: Max Pooling Layer
model.add(MaxPooling2D((2, 2)))

# Step 3: Convolutional Layer with ReLU activation
model.add(Conv2D(64, (3, 3), activation='relu'))

# Step 4: Max Pooling Layer
model.add(MaxPooling2D((2, 2)))

# Step 5: Flatten Layer
model.add(Flatten())

# Step 6: Dense (Fully Connected) Layer with ReLU activation
model.add(Dense(64, activation='relu'))

# Step 7: Output Layer with Softmax activation (for multi-class classification)
model.add(Dense(10, activation='softmax'))
```

7.3.1.4 Compile the Model:

The model is compiled with the Adam optimizer, categorical cross-entropy loss (appropriate for multi-class classification), and accuracy as the metric to monitor during training.

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

7.3.1.5 Train the Model:

The model is trained using the training data. Adjust the number of epochs and batch size based on your needs.

```
# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images,
                                                                                test_labels))
```

7.3.1.6 Evaluate the Model:

The trained model is evaluated on the test set, and the accuracy is printed.

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')
```

Results

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 19s 2us/step
Epoch 1/5
938/938 [==============================] - 27s 28ms/step - loss: 0.1663 - accuracy: 0.9512 -
val_loss: 0.0491 - val_accuracy: 0.9833
Epoch 2/5
938/938 [==============================] - 27s 29ms/step - loss: 0.0513 - accuracy: 0.9843 -
val_loss: 0.0399 - val_accuracy: 0.9872
Epoch 3/5
938/938 [==============================] - 27s 28ms/step - loss: 0.0373 - accuracy: 0.9880 -
val_loss: 0.0377 - val_accuracy: 0.9882
Epoch 4/5
938/938 [==============================] - 27s 29ms/step - loss: 0.0286 - accuracy: 0.9910 -
val_loss: 0.0298 - val_accuracy: 0.9897
Epoch 5/5
938/938 [==============================] - 28s 30ms/step - loss: 0.0221 - accuracy: 0.9930 -
val_loss: 0.0301 - val_accuracy: 0.9896
313/313 [==============================] - 2s 8ms/step - loss: 0.0301 - accuracy: 0.9896
Test Accuracy: 0.9896000027656555
```

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 8: Improving the Regression Model

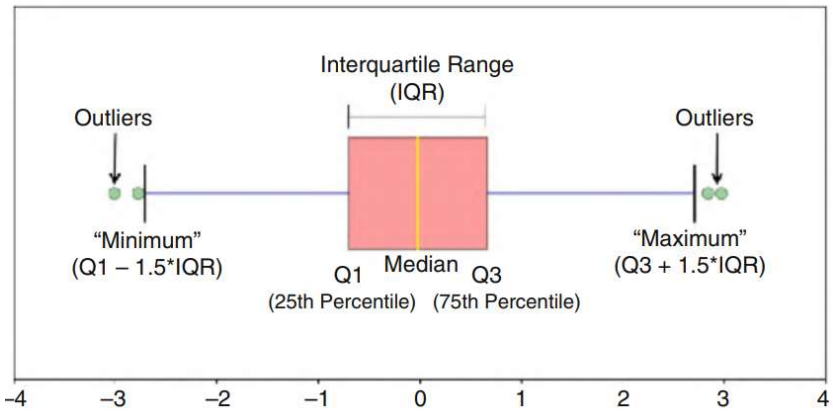
8.1 Objectives

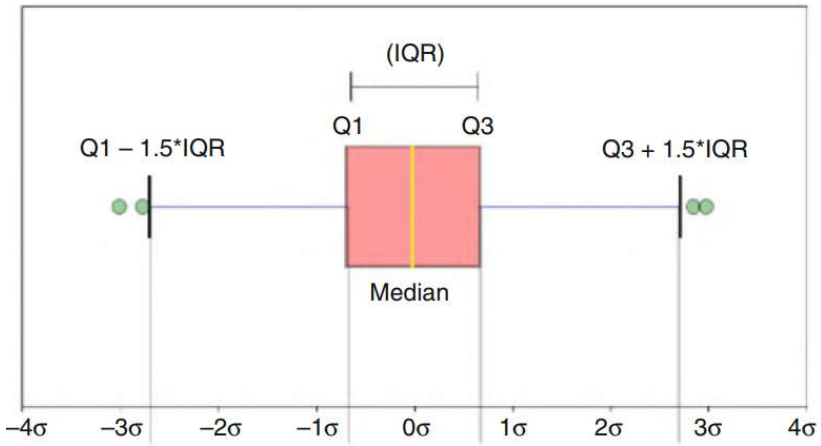
Use feature importance tools under supervision to practice selecting relevant features for improving AI regression models.

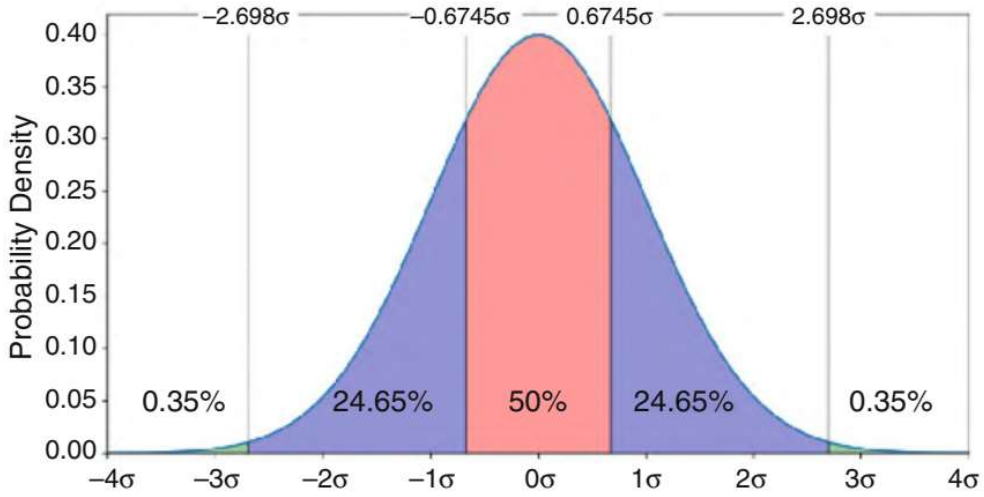
8.2 Pre-Lab

8.2.1 Box Plot

A boxplot is a standardized way of displaying the distribution of data based on a five number summaries ("minimum," first quartile (Q1), median, third quartile (Q3),and "maximum").It tells you about your outliers and what their values are. It canalso tell you if your data is symmetrical, how tightly your data is grouped, and ifand how your data is skewed. Here is an image that shows normal distribution on a boxplot:







As seen, a boxplot is a great way to visualize your dataset.

8.3 In-Lab

Now, let us try toremove the outliers using our boxplot plot. This can be easily achieved with pandasdataframe. But do note that the dataset should be numerical to do this.

In-Lab Task 1

Write the following code to visualize data using box plot.

```
# Importing libraries needed
# Note that keras is generally used for deep learning as well
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn import linear_model
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import pandas as pd
import csv

import matplotlib.pyplot as plt

# =============================================================================
# Read Data and fix seed
# =============================================================================
# fix random seed for reproducibility
np.random.seed(7)
df = pd.read_csv("Alumni Giving Regression (Edited).csv", delimiter="," )
dd_df_1=df.head()
```

```
import seaborn as sns
import pandas as pd
boxplot = pd.DataFrame(df).boxplot()
```

Discuss the result.

8.3.1 Remove Outlier

Removing the outliers observed through the results in the boxplot can improve the model. The outlier can be removed through passing a quantile value.

In-Lab Task 2

Write the following code to remove outliers in the data. Discuss the difference between two pieces of code.

```
# %%
quantile99 = df.iloc[:,0].quantile(0.99)
df1 = df[df.iloc[:,0] < quantile99]
df1.boxplot()


# %%
quantile1 = df.iloc[:,0].quantile(0.01)
quantile99 = df.iloc[:,0].quantile(0.99)
df2 = df[(df.iloc[:,0] > quantile1) & (df.iloc[:,0] <quantile99)]
df2.boxplot()
```

Discuss the results obtained.

8.3.2 Remove NA

Drop the “NA” values in the data to improve the result. Remove “NA” values from data using the following code:

```
# %%
df.dropna()
```

8.3.3 Feature Importance

Apart from data cleaning, we can apply use variables that we deem to be importantto us. One way of doing so is via feature importance of random forest trees. In manyuse cases it is equally important to not only have an accurate but also an interpretablemodel. Oftentimes, apart from wanting to know what our model’s house priceprediction is, we also wonder why it is these high/low and which features aremost important in determining the forecast. Another example might be predictingcustomer churn—it is very nice to have a model that is successfully predicting whichcustomers are prone to churn, but identifying which variables are important can helpus in early detection and maybe even improving the product/service.Knowing feature importance indicated by machine learning models can benefityou in multiple ways, for example:

60. By getting a better understanding of the model’s logic you can not only verify it being correct but also work on improving the model by focusing only on the important variables.
61. The above can be used for variable selection—you can remove x variables thatare not that significant and have similar or better performance in much shortertraining time.
62. In some business cases it makes sense to sacrifice some accuracy for the sakeof interpretability.

For example, when a bank rejects a loan application, it mustalso have a reasoning behind the decision, which can also be presented to thecustomer.

```
# %%
df.dropna()

# %%
# ================================================================================
# Feature Ranking
# ================================================================================
RF = model3
importances = RF.feature_importances_
std = np.std([tree.feature_importances_ for tree in RF.estimators_],axis=0)
indices = np.argsort(importances)[::-1]
# Print the feature ranking
print("Feature ranking:")
for f in range(X.shape[1]):
    print("%d. feature (Column index) %s (%f)" % (f + 1,indices[f],
                                                  importances[indices[f]]))
```

List here the Feature Ranks obtained through this code:

Explain the feature ranks with relation to the correlation obtained in the previous labs.

In-Lab Task 3

Write the following code to improve the regression model using top three ranked feature and compare your results obtained in the previous labs.

```
# %%
indices_top3 = indices[:3]
print(indices_top3)
dataset=df
df = pd.DataFrame(df)
Y_position = 5
TOP_N_FEATURE = 3
X = dataset.iloc[:,indices_top3]
Y = dataset.iloc[:,Y_position]
# create model
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,
                                                    random_state=2020)
#Model 1 : linear regression
model1 = linear_model.LinearRegression()
model1.fit(X_train, y_train)
y_pred_train1 = model1.predict(X_train)
print("Regression")
print("================================")
RMSE_train1 = mean_squared_error(y_train,y_pred_train1)
print("Regression TrainSet: RMSE {}".format(RMSE_train1))
print("================================")
y_pred1 = model1.predict(X_test)
RMSE_test1 = mean_squared_error(y_test,y_pred1)
print("Regression Testset: RMSE {}".format(RMSE_test1))
print("================================")
```

Discuss Results and Compare with previous labs.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 9: Decision Tree Classification

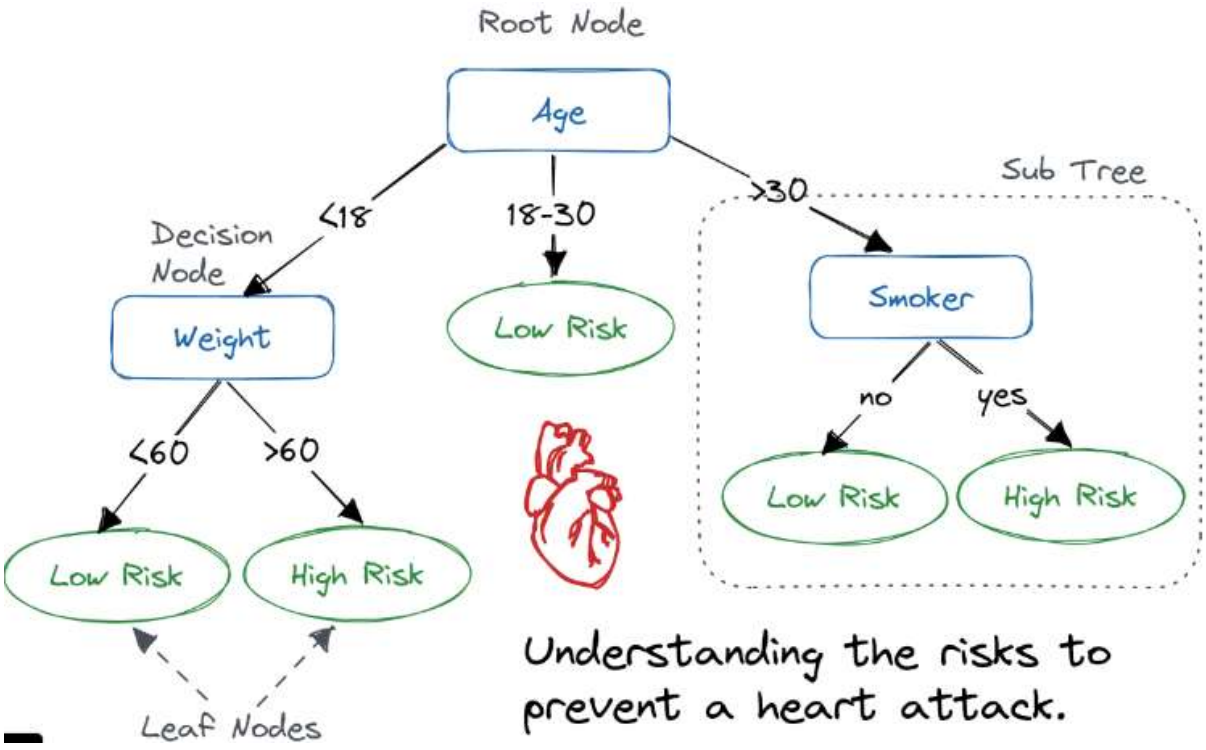
9.1 Objectives

Perform Decision Tree Classification using Python while following structured procedures, and observe how data is classified based on features.

9.2 Pre-Lab

The Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents a feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome.The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision-making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

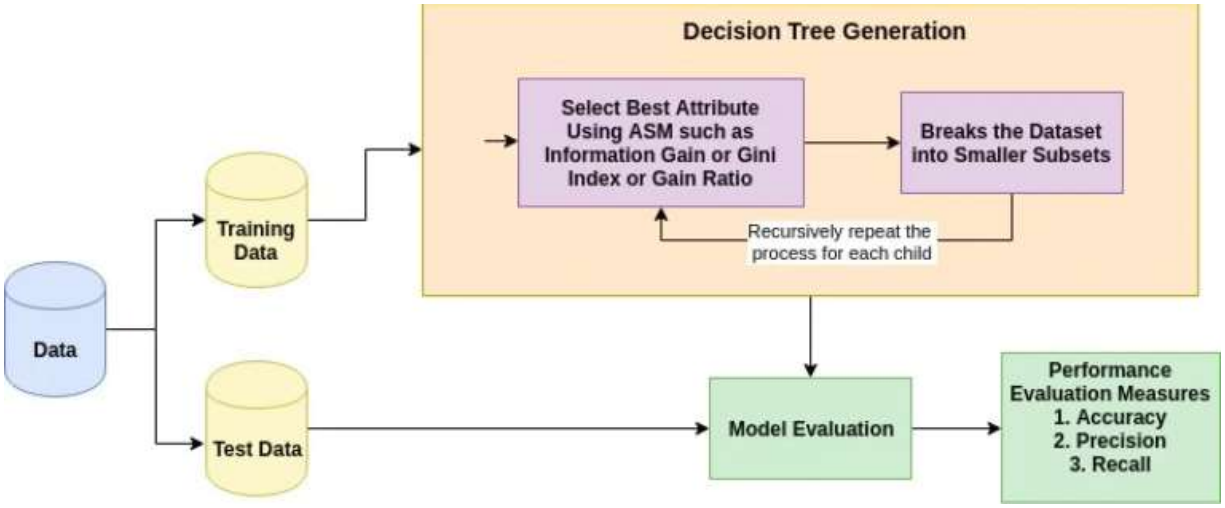


A decision tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as with a neural network. Its training time is faster compared to the neural network algorithm.

The time complexity of decision trees is a function of the number of records and attributes in the given data. The decision tree is a distribution-free or non-parametric method which does not depend upon probability distribution assumptions. Decision trees can handle high-dimensional data with good accuracy.

The basic idea behind any decision tree algorithm is as follows:

63. Select the best attribute using Attribute Selection Measures (ASM) to split the records.
64. Make that attribute a decision node and breaks the dataset into smaller subsets.
65. Start tree building by repeating this process recursively for each child until one of the conditions will match:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.



9.3 In-Lab

Decision Tree Classifier Building in Scikit-learn

9.3.1 Importing Required Libraries

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifi
from sklearn.model_selection import train_test_split # Import train_test_split
from sklearn import metrics #Import scikit-learn metrics module for accuracy ca
```

9.3.2 Loading Data

You can download the dataset using following link:

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

diabetes.csv(23.87 kB)

Remove the data header row which already in the data and use new names of Col. This can be done manually by deleting the header row in .csv file or either through the python before applying new names to the col. in data.

```
# ================================================================================
# Load Dataset
# https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
# ================================================================================
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)


pima_df= pima.head()
print (pima_df)
```

9.3.3 Feature Selection

Divide given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

```
# ================================================================================
# Feature Selection
# ================================================================================
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
```

9.3.4 Splitting Data

Split data into train and test with the ration of 70% and 30%. Students can change the ration to evaluate impact of increasing or decreasing training data size.

```
# ================================================================================
# Splitting Data
# ================================================================================
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# 70% training and 30% test
```

9.3.5 Building Decision Tree Model

```
# ================================================================================
# Building Decision Tree Model
# ================================================================================

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

# ================================================================================
#  Test Model
# ================================================================================

y_pred = clf.predict(X_test)
```

9.3.6 Evaluating the Model

```
# ================================================================================
# Evaluating the Model
# ================================================================================

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

9.3.7 Visualizing Decision Trees

For this you are required to download the "Graphviz" from the following link:

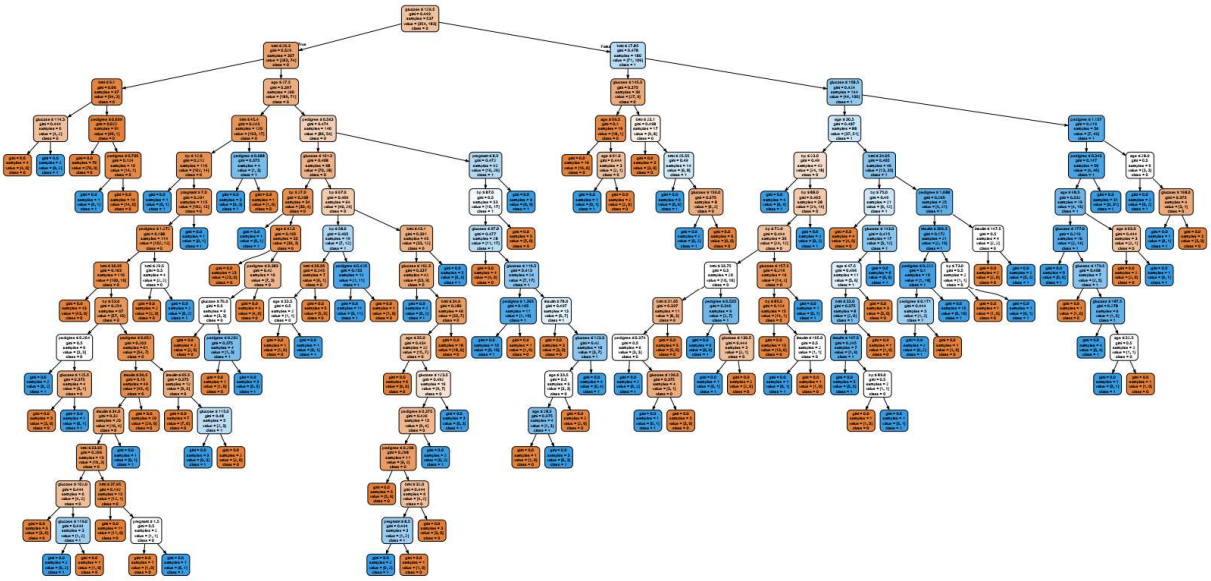
https://graphviz.gitlab.io/download/

Download and install.

>>>>>>> During Install: Select option "Add **in windows path for all users**"

```
# ================================================================================
# Visualizing Decision Trees
# ================================================================================

from sklearn.tree import export_graphviz
import graphviz

# Export the decision tree to DOT format
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=X_train.columns,  # Replace with your feature names
                           class_names=[str(x) for x in clf.classes_],  # Convert class names to stri
                           filled=True, rounded=True, special_characters=True)

# Create and display the graph
graph = graphviz.Source(dot_data)
graph.render("decision_tree")  # Saves the visualization as a file (e.g., "decision_tree.pdf")
graph.view("decision_tree")  # Opens the visualization using the default viewer
```

Result of Decision Tree



To visualize again: close this first then run code.

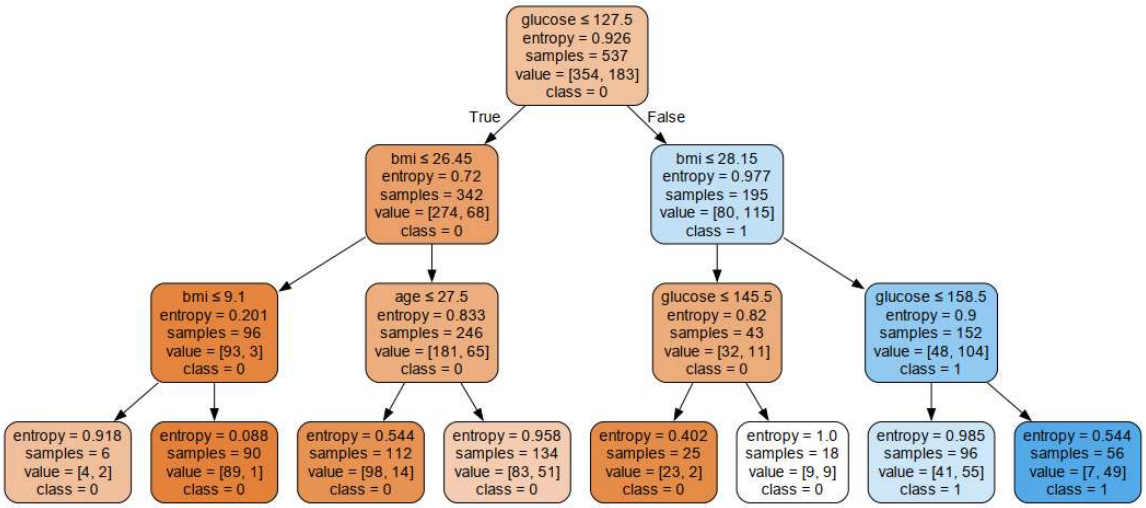
9.3.8 Optimizing Decision Tree Performance

Explore more options to improve results.

```
# =============================================================================
# Optimizing Decision Tree Performance
# =============================================================================
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

9.3.9 Visualizing Decision Trees

Using the already used code again.



Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 10: Handling Imbalanced Data and Hyperparameter Tuning

10.1 Objectives

Apply data preprocessing techniques to handle imbalanced datasets and practice hyperparameter tuning under instructor guidance to improve classification model performance.

10.2 Pre-Lab

Concepts Of Data Imbalance and Its Impact on Model Performance

In real-world datasets, it is common to encounter class imbalance — a situation where one class (e.g., "fraudulent") has significantly fewer instances than another (e.g., "non-fraudulent"). This imbalance can cause machine learning models to perform poorly on the minority class, often predicting only the majority class to achieve high overall accuracy. However, this is misleading, as the model fails to detect important but rare cases.

To address this, several techniques can be applied. One method is oversampling, where more examples of the minority class are generated to balance the data. A widely used approach for this is SMOTE (Synthetic Minority Oversampling Technique), which creates new, synthetic instances based on the existing minority class data. Alternatively, undersampling can be used, which reduces the number of majority class instances to match the minority. While this helps balance the data, it may result in the loss of valuable information from the majority class.

Another common approach is class weighting. Here, we modify the learning algorithm to assign higher importance to the minority class. This means that misclassifying a minority class example carries a greater penalty, encouraging the model to learn from these examples more effectively. Many machine learning algorithms such as logistic regression, decision trees, and neural networks support class weighting.

After handling data imbalance, another critical step is hyperparameter tuning. Machine learning models come with hyperparameters—preset values that control how the model learns. Finding the right combination of these can significantly improve performance.

There are different ways to tune hyperparameters. Grid search is the most straightforward method, where we specify several possible values for each parameter, and the algorithm tests every combination. This is thorough but can be slow with many parameters. Random search addresses this by selecting random combinations of parameters to test, making it faster and often just as effective. A more advanced method is Bayesian optimization, which builds a probabilistic model of the function being optimized and uses it to choose the next most promising set of parameters. This method learns from each attempt and intelligently decides which combination to try next.

By applying data balancing techniques such as SMOTE or class weighting, and optimizing model hyperparameters using grid search, random search, or Bayesian optimization, we can significantly improve the performance and reliability of AI models, especially in classification tasks with imbalanced data.

10.3 In-Lab

10.3.1 Load and Analyze Imbalanced Dataset

```
from sklearn.datasets import make_classification
from collections import Counter

# Create an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=2,
                           n_redundant=10, n_classes=2, weights=[0.9, 0.1],
                           random_state=42)

# Summarize class distribution
print("Original dataset shape:", Counter(y))
```

10.3.2 Apply SMOTE for Oversampling

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Summarize new class distribution
print("Resampled dataset shape:", Counter(y_res))
```

10.3.3 Hyperparameter Tuning with Grid Search

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Initialize classifier and grid search
clf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='f1')

# Fit the model
grid_search.fit(X_res, y_res)

# Print best parameters
print("Best parameters:", grid_search.best_params_)
```

10.4 Post-Lab

66. Compare model performance before and after handling data imbalance
67. Analyze the impact of different hyperparameter combinations
68. Discuss when to use each technique in real-world scenarios

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 11: Building a Rule-Based Expert System

11.1 Objectives

Practice designing a basic rule-based expert system in Python by applying logical rules with support from tutorials or templates..

11.2 Pre-Lab

Understand components of rule-based systems:

- Knowledge base
- Inference engine
- Working memory

Learn about forward and backward chaining

- Review Python dictionary and control structures for rule implementation

11.3 In-Lab

Task 1: Create Knowledge Base

```
# Define rules for a medical diagnosis system
knowledge_base = {
    "rule1": {
        "if": ["fever", "cough"],
        "then": "common_cold",
        "certainty": 0.7
    },
    "rule2": {
        "if": ["fever", "headache", "stiff_neck"],
        "then": "meningitis",
        "certainty": 0.9
    },
    "rule3": {
        "if": ["common_cold", "lasts_more_than_10_days"],
        "then": "sinus_infection",
        "certainty": 0.8
    }
}
```

Task 2: Implement Inference Engine

```
def expert_system(symptoms, knowledge_base):
    conclusions = []

    for rule_name, rule in knowledge_base.items():
        if all(symptom in symptoms for symptom in rule["if"]):
            conclusions.append((rule["then"], rule["certainty"]))

    return conclusions
```

```
# Example usage
symptoms = ["fever", "cough", "headache"]
diagnoses = expert_system(symptoms, knowledge_base)
print("Possible diagnoses:", diagnoses)
```

Task 3: Add Simple User Interface

```
def run_expert_system():
    print("Medical Diagnosis Expert System")
    print("Enter symptoms separated by commas (e.g., fever,cough,headache):")
    user_input = input().strip().lower()
    symptoms = [s.strip() for s in user_input.split(",")]

    diagnoses = expert_system(symptoms, knowledge_base)

    if diagnoses:
        print("\nPossible conditions:")
        for condition, certainty in diagnoses:
            print(f"- {condition} (certainty: {certainty*100}%)")
    else:
        print("No matching conditions found for these symptoms.")

run_expert_system()
```

11.4 Post-Lab

1. Extend the knowledge base with additional rules
2. Implement confidence-based conflict resolution
3. Discuss limitations of rule-based systems

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

Lab # 12: Ensemble Techniques for Complex Problems

12.1 Objectives

Manipulate and integrate multiple AI algorithms through guided implementation of ensemble methods to solve complex problems.

12.2 Pre-Lab

1. Understand ensemble learning concepts:
    - Bagging
    - Boosting
    - Stacking
2. Review popular ensemble algorithms:
    - Random Forest
    - AdaBoost
    - Gradient Boosting
3. Learn about voting classifiers

12.3 In-Lab

Task 1: Implement Voting Classifier

```python
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define individual models
model1 = LogisticRegression(max_iter=1000, random_state=42)
model2 = DecisionTreeClassifier(random_state=42)
model3 = SVC(probability=True, random_state=42)

# Create voting classifier
ensemble = VotingClassifier(
    estimators=[('lr', model1), ('dt', model2), ('svc', model3)],
    voting='soft'
)

# Train and evaluate
ensemble.fit(X_train, y_train)
print("Ensemble accuracy:", ensemble.score(X_test, y_test))
```

Task 2: Implement Stacking Classifier

```
from sklearn.ensemble import StackingClassifier
from sklearn.naive_bayes import GaussianNB

# Define base models
base_models = [
    ('lr', LogisticRegression(max_iter=1000, random_state=42)),
    ('dt', DecisionTreeClassifier(random_state=42)),
    ('nb', GaussianNB())
]

# Define meta-model
meta_model = LogisticRegression()

# Create stacking classifier
stacking = StackingClassifier(
    estimators=base_models,
    final_estimator=meta_model,
    cv=5
)

# Train and evaluate
stacking.fit(X_train, y_train)
print("Stacking accuracy:", stacking.score(X_test, y_test))
```

Task 3: Compare Individual Models

```
# Evaluate individual models
for name, model in [('Logistic Regression', model1),
                    ('Decision Tree', model2),
                    ('SVM', model3)]:
    model.fit(X_train, y_train)
    print(f"{name} accuracy: {model.score(X_test, y_test)}")
```

12.4 Post-Lab

1. Analyze why ensemble methods often outperform individual models
2. Experiment with different combinations of base models
3. Discuss computational trade-offs of ensemble methods

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____