USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 01
Python Programming a Review

Objective:

This lab provides a comprehensive revision of basic programming concepts and their implementation in Python. The lab also revises some of the implementations specific to Python programming

Name of Student: Sawera Fazal

Roll No:21A-026-SE **Sec.21A**

Date of Experiment: 8 oct 2023

Marks Obtained/Remarks: _____

Signature: _____

Lab Task: Go through and attempt all the notebooks provided to you along with this lab.

Homework: Attempt all the exercises provided to you inside the homework folder. Your understanding of Python will be evaluated through a Quiz before the next lab.

```
# Make a dictionary with {} and : to signify a key and a value
sawera={'career1':'web_developer', 'career2':'cybersecurity', 'career3':'software_Architecture',"salary":['11akh','5lakh','10
```

[3] ✓ 0.0s Python

```
# Call values by their key
sawera['career1']
```

[4] ✓ 0.0s Python

```
'web_developer'
```

Its important to note that dictionaries are very flexible in the data types they can hold. For example:

```
saw = {'key1':123,'key2':[12,23,33],'key3':['sa','we','ra']}
```

[5] ✓ 0.0s Python

14

```
#Lets call items from the dictionary
saw['key3']
```

[6] ✓ 0.0s Python

```
['sa', 'we', 'ra']
```

```
# Can call an index on that value
saw['key3'][0]
```

[7] ✓ 0.0s Python

```
'sa'
```

```
#Can then even call methods on that value
saw['key3'][0].upper()
```

[8] ✓ 0.0s Python

```
'SA'
```

We can effect the values of a key as well. For instance:

```
saw['key3']
```

[9] ✓ 0.0s Python

```
['sa', 'we', 'ra']
```

```
# Subtract 123 from the value
saw['key1'] = saw['key1'] - 123
```

[10] ✓ 0.0s

```
#Check
saw['key1']
```

[11] ✓ 0.0s

```
0
```

A quick note, Python has a built-in method of doing a self subtraction or addition (or multiplication or division). We could have also used += or -= for the above statement. For example:

```
# Set the object equal to itself minus 123
saw['key1'] -= 123
saw['key1']
```

[12] ✓ 0.0s

```
-123
```

+ Code + Markdown

We can also create keys by assignment. For instance if we started off with an empty dictionary, we could continually add to it:

```
# Create a new dictionary
A = {}
```

[13] ✓ 0.0s

```
# Create a new key through assignment
A['Star'] = 'Pisces'
```

[14] ✓ 0.0s

```
# Can do this with any object
A['bdate'] = 18
```

[15] ✓ 0.0s

```
#Show
A
```

[16] ✓ 0.0s

```
{'Star': 'Pisces', 'bdate': 18}
```

```
# Dictionary nested inside a dictionary nested in side a dictionary
era = {'key1':{'key2':{'key3':'chill'}}}
```
[17] ✓ 0.0s

Wow! Thats a quite the inception of dictionaries! Let's see how we can grab that value:

```
# Keep calling the keys
era['key1']['key2']
```
[18] ✓ 0.0s

```
{'key3': 'chill'}
```

A few Dictionary Methods

There are a few methods we can call on a dictionary. Let's get a quick introduction to a few of them:

```
# Create a typical dictionary
ra = {'key1':'s','key2':'a','key3':'w'}
```
[19] ✓ 0.0s

```
# Method to return a list of all keys
ra.keys()
```
[20] ✓ 0.0s

```
dict_keys(['key1', 'key2', 'key3'])
```

```
# Method to grab all values
ra.values()
```
[21] ✓ 0.0s

```
dict_values(['s', 'a', 'w'])
```

```
# Method to return tuples of all items  (we'll learn about tuples soon)
ra.items()
```
[22] ✓ 0.0s

```
dict_items([('key1', 's'), ('key2', 'a'), ('key3', 'w')])
```

```
# We'll learn how to automate this sort of list in the next lecture
list = [1,2,3,4,5,6,7,8,9,10]
```

[1]

```
for num in list:
    print (num)
```

[2]

```
1
2
3
4
5
6
7
8
9
10
```

Great! Hopefully this makes sense. Now lets add a if statement to check for even numbers.

Example 2

Let's print only the even numbers from that list!

```
for num in list:
    if num % 2 == 0:
        print (num)
```

[3]

```
2
4
6
8
10
```

We could have also put in else statement in there:

```
for num in list:
    if num % 2 == 0:
        print ('even number')
    else:
        print ('Odd number')
```

[6]

```
Odd number
even number
Odd number
even number
Odd number
even number
Odd number
even number
Odd number
```

```
# Start sum at zero
Lsum = 0

for num in list:
    Lsum = Lsum + num

print (Lsum)
```

```
55
```

Great! Read over the above cell and make sure you understand fully what is going on. Also we could have implemented a += to to the addition towards the sum. For example:

```
# Start sum at zero
list_sum = 0

for num in list:
    list_sum += num

print (list_sum)
```

```
55
```

Example 4

We've used for loops with lists, how about with strings? Remember strings are a sequence so when we iterate through them we will be accessing each item in that string.

```
for letter in 'My name is Sawera.':
    print (letter)
```

```
M
y

n
a
m
e

i
s

S
a
w
e
r
a
.
```

```
tup = (1,2,3,4,5)

for t in tup:
    print (t)
```

[10]

```
1
2
3
4
5
```

Example 6

Tuples have a special quality when it comes to **for** loops. If you are iterating through a sequence that contains tuples, the item can actually be the tuple itself, t
tuple!

```
l = [(2,4),(6,8),(10,12)]
```

[12]

```
for tup in l:
    print (tup)
```

[13]

```
(2, 4)
(6, 8)
(10, 12)
```

```
# Now with unpacking!
for (t1,t2) in l:
    print (t2)
```

[14]

```
4
8
12
```

Cool! With tuples in a sequence we can access the items inside of them through unpacking! The reason this is important is because many object will deliver th

Example 7

```
d = {'k1':1,'k2':2,'k3':3}
```

[16]

```
for keys in d:
    print (keys)
```

[20]

```
k1
k2
```

Lab 01: Python Programming a Review

```
def say_hi():
    print ('hi')
```

Call the function

```
say_hi()
```

```
hi
```

###Example 2: A simple greeting function Let's write a function that greets people with their name.

```
def greeting(name):
    print ('Hello %s '  %name)
```

```
greeting('sawera')
```

```
Hello sawera
```

Using return

Let's see some example that use a return statement. return allows a function to *return* a result that can then be stored as a variable, or used in whatever manner a user wants.

Example 3: Addition function

```
def add_num(num1,num2):
    return num1+num2
```

```
add_num(4,5)
```

```
9
```

```
# Can also save as variable due to return
result = add_num(4,5)
```

```
print (result)
```

```
9
```

What happens if we input two strings?

```
print (add_num('one','two'))
```

[8]

```
onetwo
```

Note that because we don't declare variable types in Python, this function could be used to add numbers or sequences together! We'll later learn about adding in checks to make sure a user puts in the correct arguments into a function.

Lets also start using *break*,*continue*, and *pass* statements in our code. We introduced these during the while lecture.

Finally lets go over a full example of creating a function to check if a number is prime (a common interview exercise).

We know a number is prime if that number is only evenly divisible by 1 and itself. Let's write our first version of the function to check all the numbers from 1 to N and perform modulo checks.

```
def is_prime(num):
    '''
    Naive method of checking for primes.
    '''
    for n in range(2,num):
        if num % n == 0:
            print ('not prime')
            break
    else: # If never mod zero, then prime
        print ('prime')
```

[12]

```
is_prime(16)
```

[10]

```
not prime
```

Note how we break the code after the print statement! We can actually improve this by only checking to the square root of the target number, also we can disregard all even numbers after checking for 2. We'll also switch to returning a bool

```
import math

def is_prime(num):
    '''
    Better method of checking for primes.
    '''
    if num % 2 == 0 and num > 2:
        return False
    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if num % i == 0:
            return False
    return True
```

[11]

```
is_prime(14)
```

[12]

```
False
```

```
x = True

if x:
    print ('x was True!')
else:
    print ('I will be printed in any case where x is not true')
```

```
x was True!
```

Multiple Branches

Let's get a fuller picture of how far if, elif, and else can take us!

We write this out in a nested structure. Take note of how the if,elif,and else line up in the code. This can help you see what if is related to what elif or else statements.

We'll reintroduce a comparison syntax for Python.

```
loc = 'Bank'

if loc == 'Auto Shop':
    print ('Welcome to the Auto Shop!')
elif loc == 'Bank':
    print ('Welcome to the bank!')
else:
    print ("Where are you?")
```

```
Welcome to the bank!
```

Note how the nested if statements are each checked until a True boolean causes the nested code below it to run. You should also note that you can put in as many elif statements as you want before you close off with an else.

Let's create two more simple examples for the if,elif, and else statements:

```
person = 'Sawera'

if person == 'Sawera':
    print ('Welcome !')
else:
    print ("Welcome, what's your name?" )
```

```
Welcome !
```

```
person='sawera'

if person == 'Sammy':
    print ('Welcome Sammy!')
elif person =='George':
    print ("Welcome George!")
else:
    print ("Welcome, what's your name?" )
```

```
Welcome, what's your name?
```

```
my_list = ['A string',23,100.232,'o']
```

Just like strings, the len() function will tell you how many items are in the sequence of the list.

```
len(my_list)
```

```
4
```

Indexing and Slicing

Indexing and slicing works just like in strings. Let's make a new list to remind ourselves of how this works:

```
my_list = ['one','two','three',4,5]
```

```
# Grab element at index 0
my_list[4]
```

```
5
```

```
# Grab index 1 and everything past it
my_list[1:]
```

```
['two', 'three', 4, 5]
```

```
# Grab everything UP TO index 3
my_list[:3]
```

```
['one', 'two', 'three']
```

We can also use + to concatenate lists, just like we did for strings.

```
my_list + ['new item']
```

```
['one', 'two', 'three', 4, 5, 'new item']
```

Note: This doesn't actually change the original list!

```
# Create a new list
l = [1,2,3]
```

Use the **append** method to permanently add an item to the end of a list:

```
# Append
l.append('append me!')
```

```
# Show
l
```

```
[1, 2, 3, 'append me!']
```

Use **pop** to "pop off" an item from the list. By default pop takes off the last index, but you can also specify which index to pop off. Let's see an example:

```
# Pop off the 0 indexed item
l.pop(0)
```

```
1
```

```
# Show
l
```

```
[2, 3, 'append me!']
```

```
# Assign the popped element, remember default popped index is -1
popped_item = l.pop()
```

```
popped_item
```

```
'append me!'
```

```
# Show remaining list
l
```

```
[2, 3]
```

It should also be noted that lists indexing will return an error if there is no element at that index. For example:

Nesting Lists

A great feature of of Python data structures is that they support *nesting*. This means we can have data structures within data structures. For example: A list inside a list.

Let's see how this works!

```
# Let's make three lists
lst_1=[1,2,3]
lst_2=[4,5,6]
lst_3=[7,8,9]

# Make a list of lists to form a matrix
matrix = [lst_1,lst_2,lst_3]
```

[29] ✓ 0.0s

```
# Show
matrix
```

[30] ✓ 0.0s

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Now we can again use indexing to grab elements, but now there are two levels for the index. The items in the matrix object, and then the items inside that list!

```
# Grab first item in matrix object
matrix[0]
```

[31] ✓ 0.0s

```
[1, 2, 3]
```

+ Code + Markdown

```
# Grab first item of the first item in the matrix object
matrix[0][0]
```

[32] ✓ 0.0s

```
1
```

List Comprehensions

Python has an advanced feature called list comprehensions. They allow for quick construction of lists. To fully understand list comprehensions we need to understand for loops. So don't worry if you don't completely understand this section, and feel free to just skip it since we will return to this topic later.

But in case you want to know now, here are a few examples!

```
# Build a list comprehension by deconstructing a for loop within a []
first_col = [row[0] for row in matrix]
```

[33] ✓ 0.0s

```
first_col
```

[34] ✓ 0.0s

```
[1, 4, 7]
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 02
Python Programming a Review
Cont.

Objective:

This lab provides a comprehensive revision of Advanced programming concepts and their implementation in Python. The lab also revises some of the implementations specific to Python programming

Name of Student: **Sawera Fazal**

Roll No: **21A-026-SE** **Sec.21A**

Date of Experiment: **16OCT 2023**

Student Exercise

Lab Task: Go through and attempt all the notebooks provided to you along with this lab.

Homework: Attempt all the exercises provided to you inside the homework folder. Your understanding of Python will be evaluated through a Quiz before the next lab.

```
def cgpaCalculator():
    #This is the total Score Obtainable based on the Number Of Courses Offerred
    TotalScoreOfferable = 0
    #This is the score obtained by the person which is intially zero
    obtainedGrade = 0

    # The user Enters the Number of Courses he Offered
    numberOfCourses = int(input("Please Enter the number of Courses you Offered: "))
    print ("**********************************************************************************************************************************************")
    # this creates a for loop to collect all Courses offered
    for x in range(numberOfCourses):
        # This takes in the name of the course
        Course1 = input("Enter The Course code of the  course you took:")
        # THis takes in the number of unit of the course
        unit = int(input ("How many Unit is the Course you took: "))
        #This takes in the person score in the course
        score = int(input("Please Enter your Score:"))
        print ("**********************************************************************************************************************************************")

        #This increament the totalScore obtainable based on the unit of the course
        TotalScoreOfferable += unit* 5


        # these are cases for grade (70-100 = 5 points , 60-70 = 4 points , 50-60 = 3 points ,50-45 = 2 points , 45-40 = 1 point , lesser than 40 = 0 points )
        if (score >= 70):
            grade = 5
        elif(score < 70 and  score >= 60):
            grade = 4
        elif(score < 60 and  score >= 50 ):
            grade = 3
        elif(score < 50 and  score >=45):
            grade = 2
        elif (score < 45 and  score>=40):
            grade = 1
        else :
            grade = 0


        obtainedGrade += unit*grade

    Cgpa =float((obtainedGrade / TotalScoreOfferable) * 5)
    print("THANKS FOR ALL YOUR INPUT YOUR CGPA IS : " + str(Cgpa))
cgpaCalculator()
```

✓ 1m 29.5s

```
**********************************************************************************************************************************************
**********************************************************************************************************************************************
**********************************************************************************************************************************************
**********************************************************************************************************************************************
**********************************************************************************************************************************************
**********************************************************************************************************************************************
THANKS FOR ALL YOUR INPUT YOUR CGPA IS : 5.0
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 03
Intelligent Agent in Python

Objective:

This experiment demonstrates implementation of an intelligent agent using Python. It guides students through the working of a two player tic tac toe agent. Such techniques can be used to implement similar agent programs.

Name of Student: Sawera Fazal

Roll No: 21A-026-SE Sec.21A

Date of Experiment: 24oct-23

Marks Obtained/Remarks:

Student Exercise

Task 1

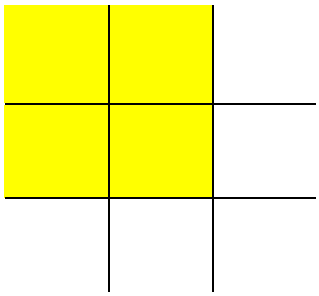
Consider the given Tic Tac Toe program designed for a match between Human and Agent. Convert it in to a program that demonstrates a play between Agent vs Agent, using two approaches:

i. Simple Reflex Agent:
   a. Am I winning the game?
   b. Am I losing the game?
   c. Go for a random move.

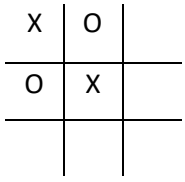
ii. Lookup Table:

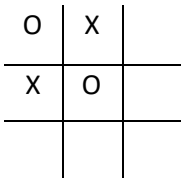
To reduce the number of possibilities your lookup table should hold, start by identifying 4 boxes that will always be filled at the start of you game. Then plan for the remaining game accordingly. For example:

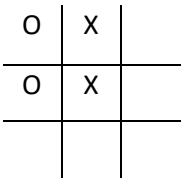
Your game always starts with any four boxes (of your choice) already filled in any specific order, again of your choice.

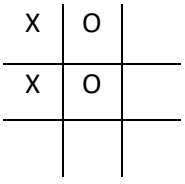


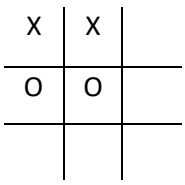
Now develop the reflex and lookup agents that can take the game forward from all possible configurations of these 4 boxes. In the sequence you are always playing 'X' and its always your turn next.

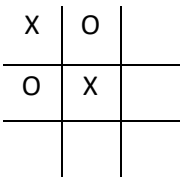


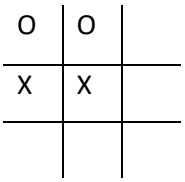












Task 2

Alter the agents that you have written so that they can handle the scenario when the computer goes first or the player/agent goes first.

Task 3

Alter the agents you have written so that they can handle all the combinations that may exist for the 4 cells you have selected.

```python
# Importing the 'copy' for deepcopy method
import copy

# Importing the heap methods from the python
# library for the Priority Queue
from heapq import heappush, heappop

# This particular var can be changed to transform
# the program from 8 puzzle(n=3) into 15
# puzzle(n=4) and so on ...
n = 3

# bottom, left, top, right
rows = [ 1, 0, -1, 0 ]
cols = [ 0, -1, 0, 1 ]

# creating a class for the Priority Queue
class priorityQueue:

    # Constructor for initializing a
    # Priority Queue
    def __init__(self):
        self.heap = []

    # Inserting a new key 'key'
    def push(self, key):
        heappush(self.heap, key)

    # funct to remove the element that is minimum,
    # from the Priority Queue
    def pop(self):
        return heappop(self.heap)

    # funct to check if the Queue is empty or not
    def empty(self):
        if not self.heap:
            return True
        else:
```

```
        else:
            return False

# structure of the node
class nodes:

    def __init__(self, parent, mats, empty_tile_posi,
                 costs, levels):

        # This will store the parent node to the
        # current node And helps in tracing the
        # path when the solution is visible
        self.parent = parent

        # Useful for Storing the matrix
        self.mats = mats

        # useful for Storing the position where the
        # empty space tile is already existing in the matrix
        self.empty_tile_posi = empty_tile_posi

        # Store no. of misplaced tiles
        self.costs = costs

        # Store no. of moves so far
        self.levels = levels

    # This func is used in order to form the
    # priority queue based on
    # the costs var of objects
    def __lt__(self, nxt):
        return self.costs < nxt.costs

# method to calc. the no. of
# misplaced tiles, that is the no. of non-blank
# tiles not in their final posi
def calculateCosts(mats, final) -> int:
```

```
    count = 0
    for i in range(n):
        for j in range(n):
            if ((mats[i][j]) and
                (mats[i][j] != final[i][j])):
                count += 1

    return count

def newNodes(mats, empty_tile_posi, new_empty_tile_posi,
             levels, parent, final) -> nodes:

    # Copying data from the parent matrixes to the present matrixes
    new_mats = copy.deepcopy(mats)

    # Moving the tile by 1 position
    x1 = empty_tile_posi[0]
    y1 = empty_tile_posi[1]
    x2 = new_empty_tile_posi[0]
    y2 = new_empty_tile_posi[1]
    new_mats[x1][y1], new_mats[x2][y2] = new_mats[x2][y2], new_mats[x1][y1]

    # Setting the no. of misplaced tiles
    costs = calculateCosts(new_mats, final)

    new_nodes = nodes(parent, new_mats, new_empty_tile_posi,
                      costs, levels)
    return new_nodes

# func to print the N by N matrix
def printMatsrix(mats):

    for i in range(n):
        for j in range(n):
            print("%d " % (mats[i][j]), end = " ")

        print()

# func to know if (x, y) is a valid or invalid
```

```python
# func to know if (x, y) is a valid or invalid
# matrix coordinates
def isSafe(x, y):

    return x >= 0 and x < n and y >= 0 and y < n

# Printing the path from the root node to the final node
def printPath(root):

    if root == None:
        return

    printPath(root.parent)
    printMatsrix(root.mats)
    print()

# method for solving N*N - 1 puzzle algo
# by utilizing the Branch and Bound technique. empty_tile_posi is
# the blank tile position initially.
def solve(initial, empty_tile_posi, final):

    # Creating a priority queue for storing the live
    # nodes of the search tree
    pq = priorityQueue()

    # Creating the root node
    costs = calculateCosts(initial, final)
    root = nodes(None, initial,
                 empty_tile_posi, costs, 0)

    # Adding root to the list of live nodes
    pq.push(root)

    # Discovering a live node with min. costs,
    # and adding its children to the list of live
    # nodes and finally deleting it from
    # the list.
    while not pq.empty():
```

```
        # live nodes
        minimum = pq.pop()

        # If the min. is ans node
        if minimum.costs == 0:

            # Printing the path from the root to
            # destination;
            printPath(minimum)
            return

        # Generating all feasible children
        for i in range(n):
            new_tile_posi = [
                minimum.empty_tile_posi[0] + rows[i],
                minimum.empty_tile_posi[1] + cols[i], ]

            if isSafe(new_tile_posi[0], new_tile_posi[1]):

                # Creating a child node
                child = newNodes(minimum.mats,
                                 minimum.empty_tile_posi,
                                 new_tile_posi,
                                 minimum.levels + 1,
                                 minimum, final,)

                # Adding the child to the list of live nodes
                pq.push(child)

# Main Code

# Initial configuration
# Value 0 is taken here as an empty space
initial = [ [ 1, 2, 3 ],
            [ 5, 6, 0 ],
            [ 7, 8, 4 ] ]

# Final configuration that can be solved
```

```python
# Initial configuration
# Value 0 is taken here as an empty space
initial = [ [ 1, 2, 3 ],
            [ 5, 6, 0 ],
            [ 7, 8, 4 ] ]

# Final configuration that can be solved
# Value 0 is taken as an empty space
final = [ [ 1, 2, 3 ],
          [ 5, 8, 6 ],
          [ 0, 7, 4 ] ]

# Blank tile coordinates in the
# initial configuration
empty_tile_posi = [ 1, 2 ]

# Method call for solving the puzzle
solve(initial, empty_tile_posi, final)
```

[6] ✓ 0.0s

```
1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4
```

Task2

Solve depth first search and breadth first search of following graph starting from node A and reaching goal node G:

In [8]:
```python
#TASK2 DFS
graph_dfs = {
    'A': ['B', 'D'],
    'B': ['A', 'C', 'E'],
    'C': ['B', 'E'],
    'D': ['E','H','G','A','B'],
    'E': ['D','F','C','B'],
    'F': ['E'],
     'G': ['D','H'],
     'H': ['D','G']
}

# Define a DFS (Depth-First Search) function that takes a graph, a start node, and a visited set (optional) as input.
def DFS(graph, start, visited=None):
    if visited is None:
        visited = set()  # Initialize the visited set if not provided.

    visited.add(start)  # Mark the current node as visited.
    print(start)  # Print the current node.

    for neighbor in graph[start]:
        if neighbor not in visited:
            # Recursively call DFS for unvisited neighbors.
            DFS(graph, neighbor, visited)

# Calling DFS for Graph 2
print("\nDFS for Graph 2:")
DFS(graph_dfs, 'A')
```

```
DFS for Graph 2:
A
B
C
E
D
H
G
F
```

In [6]:
```python
##TASK2 BFS
# Import the 'deque' class from the 'collections' module, which will be used to create a queue for BFS.
from collections import deque
# Example Graph 1 for BFS
graph_bfs = {
   'A': ['B', 'D'],
    'B': ['A', 'C', 'E'],
    'C': ['B', 'E'],
    'D': ['E','H','G','A','B'],
    'E': ['D','F','C','B'],
    'F': ['E'],
     'G': ['D','H'],
     'H': ['D','G']
}

# Define a BFS (Breadth-First Search) function that takes a graph and a starting node as input.
def BFS(graph, start):
    # Create a set to keep track of visited nodes.
    visited = set()

    # Create a deque (queue) and initialize it with the starting node.
    queue = deque([start])

    # Mark the starting node as visited.
    visited.add(start)

    # Continue while the queue is not empty.
    while queue:
        # Dequeue the first vertex from the queue.
        vertex = queue.popleft()

        # Print the current vertex to demonstrate the order of traversal.
        print(vertex)

        # Explore the neighbors of the current vertex.
        for neighbor in graph[vertex]:
            # If the neighbor has not been visited yet, enqueue it, mark it as visited, and continue the search.
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)

# Calling BFS for Graph 1
print("BFS for Graph 1:")
BFS(graph_bfs, 'A')
```

```
BFS for Graph 1:
A
B
D
C
E
H
G
F
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 04
Uninformed Search Algorithms

Objective:

This experiments models problems as search problems and then explores their possible solutions using different uniformed search algorithms.

Name of Student: Sawera Fazal

Roll No: 21A-026-SE **Sec.21A**

Date of Experiment: 29 oct 23

Marks Obtained/Remarks: _____

Sign:

Student Exercise

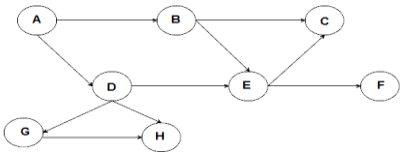
Task 1

Solve 8- Puzzle problem:

- Consist of 3 × 3 board with eight numbered tiles and a blank space.
- A tile adjacent to the blank space can slide into the space.
- The objective is to reach a specified goal state, such as the one shown in the discussion above.

Task2

Solve depth first search and breadth first search of following graph starting from node A and reaching goal node G:




```
import heapq

def astar(graph, start, goal):
    open_list = []  # Priority queue for nodes to be evaluated
    closed_set = set()  # Set of nodes already evaluated
    came_from = {}  # Mapping of nodes to their predecessors
    g_score = {node: float('inf') for node in graph}  # Cost from start to node
    g_score[start] = 0
    f_score = {node: float('inf') for node in graph}  # Estimated total cost from start to goal
    f_score[start] = heuristic(start, goal)

    # Priority queue initialization with start node
    heapq.heappush(open_list, (f_score[start], start))

    while open_list:
#This line pops the node with the lowest estimated total cost (based on the priority) from the
open_list. #The node is stored in the current variable.
        _, current = heapq.heappop(open_list)

        if current == goal:
# This line checks if the current node is the goal node
            return reconstruct_path(came_from, current)

        closed_set.add(current)

        for neighbor in graph[current]:
#: This line checks if the neighbor has already been evaluated. If it has, the code skips the
neighbor and #continues to the next iteration of the loop.
            if neighbor in closed_set:
                continue  # Skip nodes already evaluated
```

```
#This line calculates the tentative cost from the start node to the neighbor by adding the cost to
reach #the current node and the cost to move from the current node to the neighbor.
            tentative_g_score = g_score[current] + distance(current, neighbor)

            if tentative_g_score < g_score[neighbor]:
                # This path to the neighbor is better
                came_from[neighbor] = current
#This line updates the g_score for the neighbor with the new, lower cost.
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic(neighbor, goal)
```

```
                    if neighbor not in open_list:
                        heapq.heappush(open_list, (f_score[neighbor], neighbor))

        return None  # No path found

    def heuristic(node, goal):

        # Return 0 for Dijkstra's-like behavior (heuristic is always 0).
        return 0

    def reconstruct_path(came_from, current):
        path = [current]
        while current in came_from:
            current = came_from[current]
            path.append(current)
        path.reverse()
        return path

    def distance(node1, node2):
        return 1  # Default to a uniform cost of 1 for simplicity

    # Example usage:
    graph = {
        'A': {'B': 1, 'C': 3,'D': 1},
        'B': {'D': 2},
        'C': {'D': 1},
        'D': {'E': 3},
        'E': {}
    }
    start_node = 'A'
    goal_node = 'E'

    path = astar(graph, start_node, goal_node)
    print(path)
```

✓ 0.0s

```
['A', 'D', 'E']
```

```
graph = {
    'A': {'B': 9, 'C': 4, 'D': 7},
    'B': {'A': 10, 'E': 11},
    'C': {'A': 4, 'E': 17, 'F': 12},
    'D': {'A': 7, 'F': 14},
    'E': {'B': 11, 'C': 17, 'Z': 5},
    'F': {'C': 12, 'D': 14, 'Z': 9},
```

```
    'Z': {}
}

heuristics = {
    'A': 21,
    'B': 14,
    'C': 18,
    'D': 18,
    'E': 5,
    'F': 8,
    'Z': 0
}
import heapq

def astar(graph, start, goal, heuristics):
    open_list = []  # Priority queue for nodes to be evaluated
    closed_set = set()  # Set of nodes already evaluated
    came_from = {}  # Mapping of nodes to their predecessors
    g_score = {node: float('inf') for node in graph}  # Cost from start to node
    g_score[start] = 0
    f_score = {node: float('inf') for node in graph}  # Estimated total cost from start to goal
    f_score[start] = heuristics[start]

    # Priority queue initialization with start node
    heapq.heappush(open_list, (f_score[start], start))

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == goal:
            return reconstruct_path(came_from, current)

        closed_set.add(current)

        for neighbor, cost in graph[current].items():
            if neighbor in closed_set:
                continue  # Skip nodes already evaluated

            tentative_g_score = g_score[current] + cost

            if tentative_g_score < g_score[neighbor]:
                # This path to the neighbor is better
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristics[neighbor]

                if neighbor not in open_list:
                    heapq.heappush(open_list, (f_score[neighbor], neighbor))

    return None  # No path found
```

```python
def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path

# Example usage:
start_node = 'A'
goal_node = 'Z'

path = astar(graph, start_node, goal_node, heuristics)
print(path)
```

```
[2]  ✓ 0.0s
...  ['A', 'C', 'F', 'Z']
```

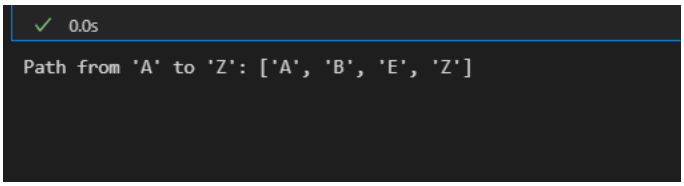
```python
import heapq

def uniform_cost_search(graph, start, goal, heuristics):
    frontier = [(0, start)]
    explored = set()
    while frontier:
        (cost, current) = heapq.heappop(frontier)
        if current == goal:
            return cost
        if current in explored:
            continue
        explored.add(current)
        for neighbor, weight in graph[current].items():
            if neighbor not in explored:
                heapq.heappush(frontier, (cost + weight, neighbor))
    return -1

def uniform_cost_search_path(graph, start, goal, heuristics):
    frontier = [(0, [start])]
    explored = set()
    while frontier:
        (cost, path) = heapq.heappop(frontier)
        current = path[-1]
        if current == goal:
            return path
        if current in explored:
            continue
        explored.add(current)
        for neighbor, weight in graph[current].items():
            if neighbor not in explored:
```

```
                new_cost = cost + weight
                new_path = path + [neighbor]
                heapq.heappush(frontier, (new_cost, new_path))
    return []

graph = {
    'A': {'B': 9, 'C': 4, 'D': 7},
    'B': {'A': 10, 'E': 11},
    'C': {'A': 4, 'E': 17, 'F': 12},
    'D': {'A': 7, 'F': 14},
    'E': {'B': 11, 'C': 17, 'Z': 5},
    'F': {'C': 12, 'D': 14, 'Z': 9},
    'Z': {}
}

heuristics = {
    'A': 21,
    'B': 14,
    'C': 18,
    'D': 18,
    'E': 5,
    'F': 8,
    'Z': 0
}

print(uniform_cost_search_path(graph, "A", "Z", heuristics))
```

```
✓ 0.0s
['A', 'B', 'E', 'Z']
```

```
import heapq

def best_first_search_path(graph, start, goal, heuristics):
    frontier = [(heuristics[start], start, [start])]
    explored = set()
    while frontier:
        _, current, path = heapq.heappop(frontier)
        if current == goal:
            return path  # Return the path
        if current not in explored:
            explored.add(current)
            for neighbor, weight in graph[current].items():
                if neighbor not in explored:
                    new_path = path + [neighbor]
                    heapq.heappush(frontier, (heuristics[neighbor], neighbor, new_path))
    return []  # Return an empty list if the goal is not reachable
```

```python
# Example usage
graph = {
    'A': {'B': 9, 'C': 4, 'D': 7},
    'B': {'A': 10, 'E': 11},
    'C': {'A': 4, 'E': 17, 'F': 12},
    'D': {'A': 7, 'F': 14},
    'E': {'B': 11, 'C': 17, 'Z': 5},
    'F': {'C': 12, 'D': 14, 'Z': 9},
    'Z': {}
}

heuristics = {
    'A': 21,
    'B': 14,
    'C': 18,
    'D': 18,
    'E': 5,
    'F': 8,
    'Z': 0
}

start_node = 'A'
goal_node = 'Z'

path = best_first_search_path(graph, start_node, goal_node, heuristics)
if path:
    print(f"Path from '{start_node}' to '{goal_node}': {path}")
else:
    print(f"No path from '{start_node}' to '{goal_node}' exists.")
```



USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 05
Optimal Search

Objective:

This experiment demonstrates the use of A* as an efficient Search Strategy. It also develops the concept of admissible and un-admissible heuristics.

Name of Student: **Sawera Fazal**

Roll No: **21A-026-SE** **Sec.21A**

Date of Experiment 29 oct-23

Marks Obtained/Remarks: _____

Signature: _____

Student Exercise

Task 1

For the following graph and heuristic values program the variants of search algorithms mentioned below.

Graph	Heuristic
'A': {'B':9, 'C':4, 'D':7}	'A':21
'B': {'A':9, 'E':11}	'B':14
'C': {'A':4, 'E':17, 'F':12}	'C':18
'D': {'A':7, 'F':14}	'D':18
'E': {'B':11, 'C':17, 'Z': 5}	'E':5
'F': {'C':12, 'D':14, 'Z': 9}	'F':8
	'Z':0

- **Uniform Cost Search**
- **Best First Search**
- **A* Search**

Task 2

Develop code to implement the A* algorithm in order to find the optimal path in the Travel in Romania problem. Use the heuristic given in the text above. Furthermore, propose an admissible heuristic of your own and compare the two heuristics utilized. Suggest which of the two heuristics is a better choice for the travel in Romania problem and why?

```
import heapq

def astar(graph, start, goal):
    open_list = []
    closed_set = set()  # evaluated
    PreNodes = {}
    goal_score = {node: float('inf') for node in graph}
    goal_score[start] = 0
    f_score = {node: float('inf') for node in graph}  #start to goal
    f_score[start] = heuristic(start, goal)

    # Priority queue initialization with start node
    heapq.heappush(open_list, (f_score[start], start))

    while open_list:
```

Lab 05: Optimal Search

```
#This line pops the node with the lowest estimated total cost (based on the priority) PreNodes the
open_list. #The node is stored in the current variable.
        _, current = heapq.heappop(open_list)

        if current == goal:
# This line checks if the current node is the goal node
            return reconstruct_path(PreNodes, current)

        closed_set.add(current)

        for neighbor in graph[current]:
#: This line checks if the neighbor has already been evaluated. If it has, the code skips the
neighbor and #continues to the next iteration of the loop.
            if neighbor in closed_set:
                continue  # Skip nodes already evaluated
#This line calculates the tentative cost PreNodes the start node to the neighbor by adding the cost
to reach #the current node and the cost to move PreNodes the current node to the neighbor.
            tentative_goal_score = goal_score[current] + distance(current, neighbor)

            if tentative_goal_score < goal_score[neighbor]:
                # This path to the neighbor is better
                PreNodes[neighbor] = current
#This line updates the goal_score for the neighbor with the new, lower cost.
                goal_score[neighbor] = tentative_goal_score
                f_score[neighbor] = goal_score[neighbor] + heuristic(neighbor, goal)

                if neighbor not in open_list:
                    heapq.heappush(open_list, (f_score[neighbor], neighbor))

    return None  # No path found

def heuristic(node, goal):

    # Return 0 for Dijkstra's-like behavior (heuristic is always 0).
    return 0

def reconstruct_path(PreNodes, current):
    path = [current]
    while current in PreNodes:
        current = PreNodes[current]
        path.append(current)
    path.reverse()
    return path

def distance(node1, node2):
    return 1  # Default to a uniform cost of 1 for simplicity

# Example usage:
graph = {
    'A': {'B': 1, 'C': 3,'D': 1},
```

Lab 05: Optimal Search

```
    'B': {'D': 2},
    'C': {'D': 1},
    'D': {'E': 3},
    'E': {}
}
start_node = 'A'
goal_node = 'E'

path = astar(graph, start_node, goal_node)
print(path)
```

```
['A', 'D', 'E']
```

```
graph = {
    'A': {'B': 9, 'C': 4, 'D': 7},
    'B': {'A': 10, 'E': 11},
    'C': {'A': 4, 'E': 17, 'F': 12},
    'D': {'A': 7, 'F': 14},
    'E': {'B': 11, 'C': 17, 'Z': 5},
    'F': {'C': 12, 'D': 14, 'Z': 9},
    'Z': {}
}

heuristics = {
    'A': 21,
    'B': 14,
    'C': 18,
    'D': 18,
    'E': 5,
    'F': 8,
    'Z': 0
}
import heapq

def astar(graph, start, goal, heuristics):
    open_list = []  # Priority queue for nodes to be evaluated
    closed_set = set()  # Set of nodes already evaluated
    PreNodes = {}  # Mapping of nodes to their predecessors
    goal_score = {node: float('inf') for node in graph}  # Cost PreNodes start to node
    goal_score[start] = 0
    f_score = {node: float('inf') for node in graph}  # Estimated total cost PreNodes start to goal
    f_score[start] = heuristics[start]

    # Priority queue initialization with start node
    heapq.heappush(open_list, (f_score[start], start))

    while open_list:
        _, current = heapq.heappop(open_list)
```

```
        if current == goal:
            return reconstruct_path(PreNodes, current)

        closed_set.add(current)

        for neighbor, cost in graph[current].items():
            if neighbor in closed_set:
                continue  # Skip nodes already evaluated

            tentative_goal_score = goal_score[current] + cost

            if tentative_goal_score < goal_score[neighbor]:
                # This path to the neighbor is better
                PreNodes[neighbor] = current
                goal_score[neighbor] = tentative_goal_score
                f_score[neighbor] = goal_score[neighbor] + heuristics[neighbor]

                if neighbor not in open_list:
                    heapq.heappush(open_list, (f_score[neighbor], neighbor))

    return None  # No path found

def reconstruct_path(PreNodes, current):
    path = [current]
    while current in PreNodes:
        current = PreNodes[current]
        path.append(current)
    path.reverse()
    return path

# Example usage:
start_node = 'A'
goal_node = 'Z'

path = astar(graph, start_node, goal_node, heuristics)
print(path)
```

```
[2]
['A', 'C', 'F', 'Z']
```

```
import heapq

def uniform_cost_search(graph, start, goal, heuristics):
    frontier = [(0, start)]
    explored = set()
    while frontier:
        (cost, current) = heapq.heappop(frontier)
        if current == goal:
            return cost
        if current in explored:
```

```
            continue
        explored.add(current)
        for neighbor, weight in graph[current].items():
            if neighbor not in explored:
                heapq.heappush(frontier, (cost + weight, neighbor))
    return -1

def uniform_cost_search_path(graph, start, goal, heuristics):
    frontier = [(0, [start])]
    explored = set()
    while frontier:
        (cost, path) = heapq.heappop(frontier)
        current = path[-1]
        if current == goal:
            return path
        if current in explored:
            continue
        explored.add(current)
        for neighbor, weight in graph[current].items():
            if neighbor not in explored:
                new_cost = cost + weight
                new_path = path + [neighbor]
                heapq.heappush(frontier, (new_cost, new_path))
    return []

graph = {
    'A': {'B': 9, 'C': 4, 'D': 7},
    'B': {'A': 10, 'E': 11},
    'C': {'A': 4, 'E': 17, 'F': 12},
    'D': {'A': 7, 'F': 14},
    'E': {'B': 11, 'C': 17, 'Z': 5},
    'F': {'C': 12, 'D': 14, 'Z': 9},
    'Z': {}
}

heuristics = {
    'A': 21,
    'B': 14,
    'C': 18,
    'D': 18,
    'E': 5,
    'F': 8,
    'Z': 0
}

print(uniform_cost_search_path(graph, "A", "Z", heuristics))
```

Lab 05: Optimal Search

```
['A', 'B', 'E', 'Z']
```

```python
import heapq

def best_first_search_path(graph, start, goal, heuristics):
    frontier = [(heuristics[start], start, [start])]
    explored = set()
    while frontier:
        _, current, path = heapq.heappop(frontier)
        if current == goal:
            return path  # Return the path
        if current not in explored:
            explored.add(current)
            for neighbor, weight in graph[current].items():
                if neighbor not in explored:
                    new_path = path + [neighbor]
                    heapq.heappush(frontier, (heuristics[neighbor], neighbor, new_path))
    return []  # Return an empty list if the goal is not reachable

# Example usage
graph = {
    'A': {'B': 9, 'C': 4, 'D': 7},
    'B': {'A': 10, 'E': 11},
    'C': {'A': 4, 'E': 17, 'F': 12},
    'D': {'A': 7, 'F': 14},
    'E': {'B': 11, 'C': 17, 'Z': 5},
    'F': {'C': 12, 'D': 14, 'Z': 9},
    'Z': {}
}

heuristics = {
    'A': 21,
    'B': 14,
    'C': 18,
    'D': 18,
    'E': 5,
    'F': 8,
    'Z': 0
}

start_node = 'A'
goal_node = 'Z'

path = best_first_search_path(graph, start_node, goal_node, heuristics)
if path:
    print(f"Path PreNodes '{start_node}' to '{goal_node}': {path}")
else:
    print(f"No path PreNodes '{start_node}' to '{goal_node}' exists.")
```

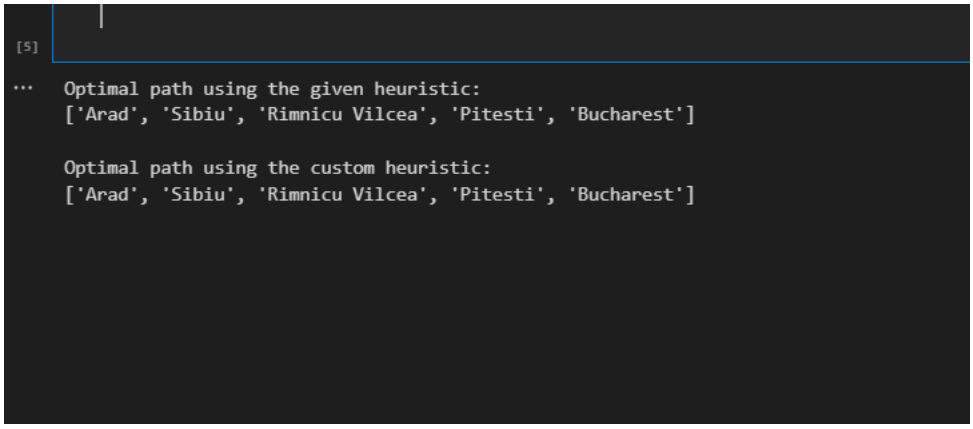
Lab 05: Optimal Search

```
Path PreNodes 'A' to 'Z': ['A', 'B', 'E', 'Z']
```

```python
#Task2
import heapq

# Define the graph representing the cities in Romania and their distances.
romania_graph = {
    'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
    'Zerind': {'Arad': 75, 'Oradea': 71},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},
    'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},
    'Giurgiu': {'Bucharest': 90},
    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},
    'Hirsova': {'Urziceni': 98, 'Eforie': 86},
    'Eforie': {'Hirsova': 86},
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Iasi': {'Vaslui': 92, 'Neamt': 87},
    'Neamt': {'Iasi': 87}
}

# Heuristic given in the text
heuristic_given = {
    'Arad': 366,
    'Bucharest': 0,
    'Craiova': 160,
    'Drobeta': 242,
    'Eforie': 161,
    'Fagaras': 178,
    'Giurgiu': 77,
    'Hirsova': 151,
    'Iasi': 226,
    'Lugoj': 244,
    'Mehadia': 241,
    'Neamt': 234,
    'Oradea': 380,
    'Pitesti': 100,
    'Rimnicu Vilcea': 193,
    'Sibiu': 253,
    'Timisoara': 329,
    'Urziceni': 80,
```

```
    'Vaslui': 199,
    'Zerind': 374
}

# Define an admissible heuristic of your own (e.g., half of the given heuristic values)
heuristic_custom = {
    city: value // 2 for city, value in heuristic_given.items()
}

def astar(graph, start, goal, heuristics):
    open_list = []  # Priority queue for nodes to be evaluated
    closed_set = set()  # Set of nodes already evaluated
    PreNodes = {}  # Mapping of nodes to their predecessors
    goal_score = {city: float('inf') for city in graph}  # Cost PreNodes start to city
    goal_score[start] = 0
    f_score = {city: float('inf') for city in graph}  # Estimated total cost PreNodes start to goal
    f_score[start] = heuristics[start]

    # Priority queue initialization with start city
    heapq.heappush(open_list, (f_score[start], start))

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == goal:
            return reconstruct_path(PreNodes, current)

        closed_set.add(current)

        for neighbor, cost in graph[current].items():
            if neighbor in closed_set:
                continue  # Skip cities already evaluated

            tentative_goal_score = goal_score[current] + cost

            if tentative_goal_score < goal_score[neighbor]:
                # This path to the neighbor is better
                PreNodes[neighbor] = current
                goal_score[neighbor] = tentative_goal_score
                f_score[neighbor] = goal_score[neighbor] + heuristics[neighbor]

                if neighbor not in open_list:
                    heapq.heappush(open_list, (f_score[neighbor], neighbor))

    return None  # No path found

def reconstruct_path(PreNodes, current):
    path = [current]
    while current in PreNodes:
        current = PreNodes[current]
```

```
        path.append(current)
    path.reverse()
    return path

# Example usage with the given heuristic
start_city = 'Arad'
goal_city = 'Bucharest'

optimal_path_given = astar(romania_graph, start_city, goal_city, heuristic_given)
print("Optimal path using the given heuristic:")
print(optimal_path_given)

# Example usage with the custom heuristic
optimal_path_custom = astar(romania_graph, start_city, goal_city, heuristic_custom)
print("\nOptimal path using the custom heuristic:")
print(optimal_path_custom)
```



USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 06
Introduction To IPCV

Objective:

This lab provides a comprehensive Introduction Image processing and Computer vision, this lab is designed to help students for their semester project

Name of Student: Sawera Fazal

Roll No: 21A-026-SE Sec.21A

Date of Experiment: 6 nov 23

Marks Obtained/Remarks: _____

Lab 06: Intro to IPCV

Code + Markdown | Run All | Clear All Outputs | Outline ...

```
from zipfile import ZipFile
file_name="archive.zip"
with ZipFile(file_name,"r") as zip:
    zip.extractall()
    print("done")
```

[1]

```
done
```

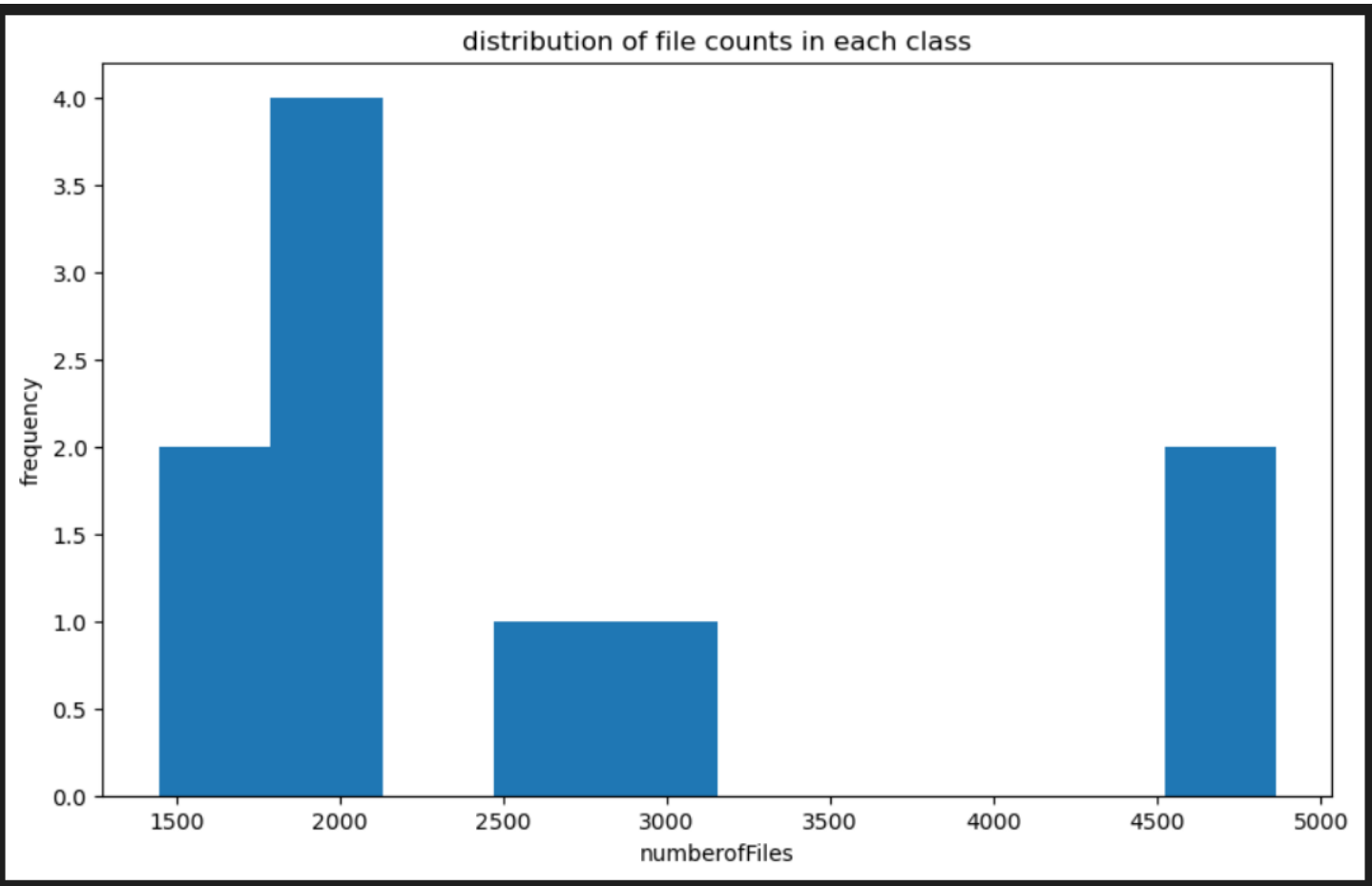
```
import os
import warnings
warnings.filterwarnings('ignore')
data_dir_list=os.listdir("raw-img")
print(data_dir_list)
```

[2]

```
['cane', 'cavallo', 'elefante', 'farfalla', 'gallina', 'gatto', 'mucca', 'pecora', 'ragno', 'scoiattolo']
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.hist([len(os.listdir(os.path.join("raw-img",d))) for d in data_dir_list],)
plt.xlabel("numberofFiles")
plt.ylabel("frequency")
plt.title("distribution of file counts in each class")
plt.show()
```

[7]

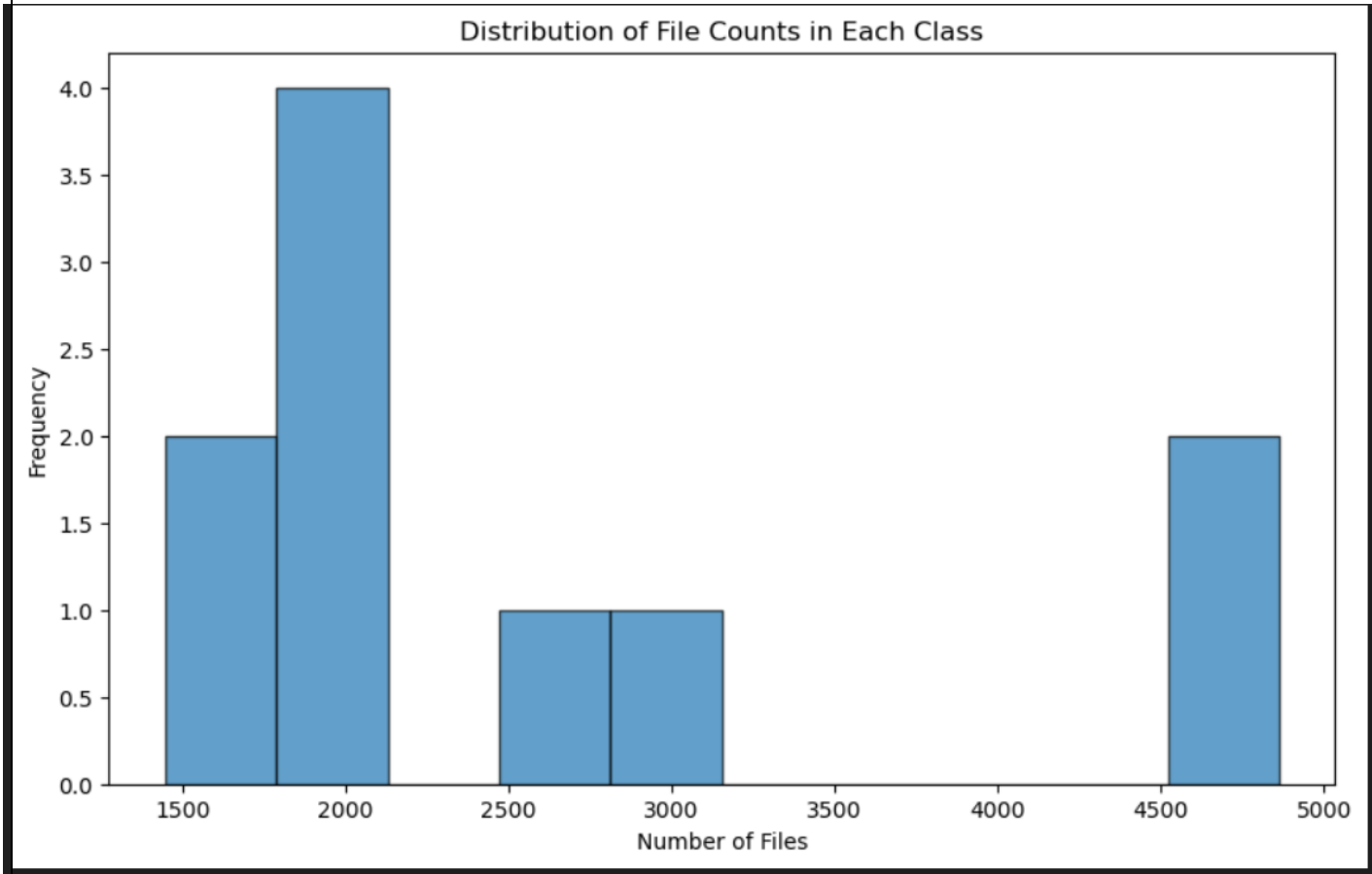


OBLEMS 37 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

ne loading configuration

```
import os
import warnings
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')


# Visualize the distribution of file counts using a histogram
plt.figure(figsize=(10, 6))
plt.hist([len(os.listdir(os.path.join("raw-img", d))) for d in data_dir_list], bins=10, alpha=0.7,
edgecolor='black')
plt.xlabel('Number of Files')
plt.ylabel('Frequency')
plt.title('Distribution of File Counts in Each Class')
plt.show()

# Calculate the minimum and maximum number of files in a class
min_files = min([len(os.listdir(os.path.join("raw-img", d))) for d in data_dir_list])
max_files = max([len(os.listdir(os.path.join("raw-img", d))) for d in data_dir_list])
print(f"Minimum number of files in a class: {min_files}")
print(f"Maximum number of files in a class: {max_files}")

# Determine the class with the highest and lowest number of files
class_with_most_files = max(data_dir_list, key=lambda x: len(os.listdir(os.path.join("raw-img",
x))))
class_with_least_files = min(data_dir_list, key=lambda x: len(os.listdir(os.path.join("raw-img",
x))))
print(f"Class with the highest number of files: {class_with_most_files}")
print(f"Class with the lowest number of files: {class_with_least_files}")
```

Lab 07: Local and Meta-Heuristic Search in AI




```python
import os
import warnings

warnings.filterwarnings('ignore')

# Get all the paths
data_dir_list = os.listdir('raw-img')

# Count the number of files in each class
for d in data_dir_list:
    class_path = os.path.join("raw-img", d)
    file_count = len([f for f in os.listdir(class_path) if os.path.isfile(os.path.join(class_path, f))])
    print(f"Class: {d} - Count: {file_count}")
```

[13]

```
Class: cane - Count: 4863
Class: cavallo - Count: 2623
Class: elefante - Count: 1446
Class: farfalla - Count: 2112
Class: gallina - Count: 3098
Class: gatto - Count: 1668
Class: mucca - Count: 1866
Class: pecora - Count: 1820
Class: ragno - Count: 4821
Class: scoiattolo - Count: 1862
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 07
Local and Meta-Heuristic Search in Artificial Intelligence

Objective:

This experiments demonstrates the use of Local Search schemes such as Hill Climbing and use of meta-heuristic based schemes such as Evolutionary and Genetic Algorithms for solution search in Artificial Intelligence.

Name of Student: Sawera Fazal

Roll No: 21A-026-SE Sec.21A

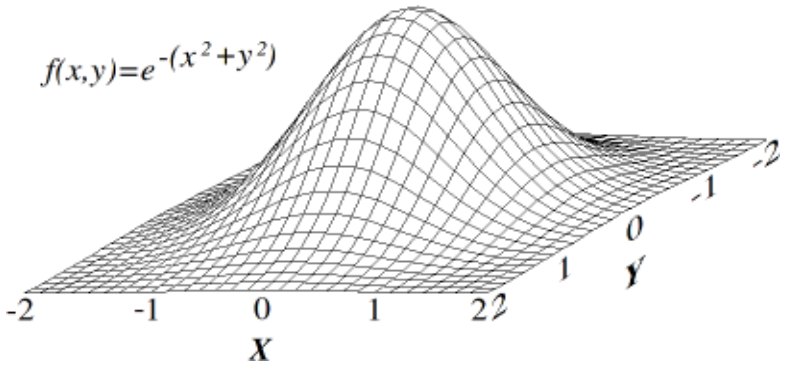
Date of Experiment: 12 nov 23

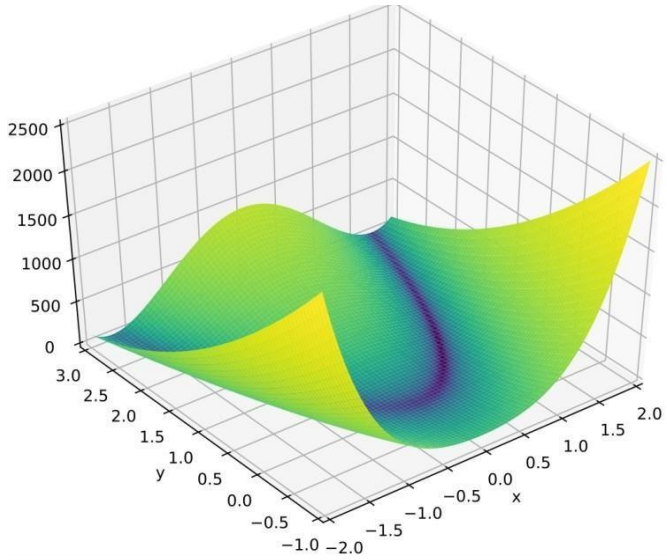
Marks Obtained/Remarks: _____

Sign _____

Student Exercise

Task 1





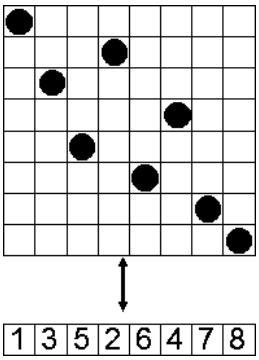
$(x, y) = (1 - x)^2 + 100(y - x^2)^2$

Implement Hill climbing solutions for the given two function. The Hill Climbing solution should provide values for the x and y parameters where the value of the function maximizes. The figures also provide the range of values the solution exist within. Restrict your search within the domain of the variables for a quicker response.

Task2

Implement a Hill climbing program for searching a configuration for the 8 queen problem. Use the representation as suggested below.

- **Penalty of one queen: the number of queens she can check.**
- **Penalty of a configuration: the sum of the penalties of all queens.**
- **Note: penalty is to be minimized**
- **Fitness of a configuration: inverse penalty to be maximized**

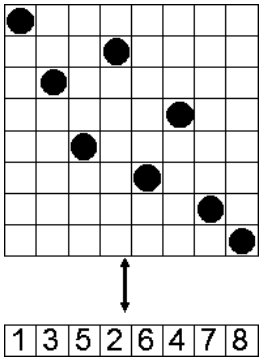


Task 3:

Implement a Genetic Algorithm using Genotypic representation for the two functions used in Task 1. Provide the optimal solution for both the functions using the GA.

Task 4:

Implement an Evolutionary Algorithm for searching a configuration for the 8 queen problem. Use the Phenotypic representation discussed below.



- **Penalty of one queen: the number of queens she can check.**
- **Penalty of a configuration: the sum of the penalties of all queens.**
- **Note: penalty is to be minimized**
- **Fitness of a configuration: inverse penalty to be maximized**

```
#TASK 1
import numpy as np
from scipy.optimize import minimize

# Function a
def func_a(params):
    x, y = params
    return -np.exp(-(x**2 + y**2))
```

```python
# Function b
def func_b(params):
    x, y = params
    return -(1 - x)**2 - 100 * (y - x**2)**2

# Hill climbing optimization
def hill_climbing(func, initial_guess, bounds):
    result = minimize(func, initial_guess, bounds=bounds, method='L-BFGS-B')
    return result.x, -result.fun

# Set the initial guess and bounds for each function
initial_guess_a = [0, 0]
bounds_a = [(-2, 2), (-2, 2)]

initial_guess_b = [0, 0]
bounds_b = [(-2, 2), (-2, 2)]

# Perform hill climbing optimization for each function
optimal_params_a, max_value_a = hill_climbing(func_a, initial_guess_a, bounds_a)
optimal_params_b, max_value_b = hill_climbing(func_b, initial_guess_b, bounds_b)

# Display results
print("Function a:")
print("Optimal Parameters:", optimal_params_a)
print("Maximum Value:", max_value_a)

print("\nFunction b:")
print("Optimal Parameters:", optimal_params_b)
print("Maximum Value:", max_value_b)
```

[1]

```
Function a:
Optimal Parameters: [0. 0.]
Maximum Value: 1.0

Function b:
Optimal Parameters: [-2. -2.]
Maximum Value: 3609.0
```

```python
#TASK 2
import random

def initial_configuration(size):
    return [random.randint(0, size - 1) for _ in range(size)]

def calculate_penalty(configuration):
    penalty = 0
    size = len(configuration)

    for i in range(size):
        for j in range(i + 1, size):
```

```
            if configuration[i] == configuration[j] or abs(i - j) == abs(configuration[i] -
configuration[j]):
                penalty += 1

    return penalty

def calculate_fitness(configuration):
    penalty = calculate_penalty(configuration)
    return 1 / (1 + penalty)

def hill_climbing(size, max_iterations=1000):
    current_configuration = initial_configuration(size)
    current_fitness = calculate_fitness(current_configuration)

    for _ in range(max_iterations):
        neighbors = []

        for i in range(size):
            for j in range(size):
                if current_configuration[i] != j:
                    neighbor = list(current_configuration)
                    neighbor[i] = j
                    neighbors.append(neighbor)

        best_neighbor = min(neighbors, key=calculate_penalty)
        best_neighbor_fitness = calculate_fitness(best_neighbor)

        if best_neighbor_fitness > current_fitness:
            current_configuration = best_neighbor
            current_fitness = best_neighbor_fitness

    return current_configuration, current_fitness

# Set the size of the chessboard
board_size = 8

# Run hill climbing algorithm
final_configuration, final_fitness = hill_climbing(board_size)

# Display results
print("Final Configuration:", final_configuration)
print("Final Fitness:", final_fitness)
```

```
Final Configuration: [6, 1, 5, 7, 3, 0, 2, 4]
Final Fitness: 0.5
```

```
#TASK 3
```

```python
import numpy as np

# Function a
def func_a(params):
    x, y = params
    return -np.exp(-(x**2 + y**2))

# Function b
def func_b(params):
    x, y = params
    return -(1 - x)**2 - 100 * (y - x**2)**2

def initialize_population(population_size, genotype_size):
    return np.random.randint(0, 2, size=(population_size, genotype_size))

def decode_genotype(genotype, bounds, precision):
    decoded_values = []
    bit_str = ''.join(map(str, genotype))

    for i, bound in enumerate(bounds):
        start, end = bound
        start_bit = i * precision
        end_bit = (i + 1) * precision
        binary_str = bit_str[start_bit:end_bit]
        decimal_value = int(binary_str, 2)
        decoded_value = start + decimal_value * (end - start) / (2 ** precision - 1)
        decoded_values.append(decoded_value)
    return decoded_values

def evaluate_population(population, decode_function):
    fitness_values = []
    for genotype in population:
        decoded_values = decode_function(genotype)
        fitness_values.append(func_a(decoded_values))  # Change to func_b for the second function
    return np.array(fitness_values)

def select_parents(population, fitness_values):
    idx = np.argsort(fitness_values)
    return population[idx[:2]]

def crossover(parents):
    crossover_point = np.random.randint(1, len(parents[0]))
    child1 = np.concatenate((parents[0][:crossover_point], parents[1][crossover_point:]))
    child2 = np.concatenate((parents[1][:crossover_point], parents[0][crossover_point:]))
    return child1, child2

def mutate(child, mutation_rate):
    mutation_mask = np.random.rand(len(child)) < mutation_rate
    child[mutation_mask] = 1 - child[mutation_mask]
    return child
```

```
def genetic_algorithm(num_generations, population_size, genotype_size, bounds, precision,
mutation_rate):
    population = initialize_population(population_size, genotype_size)

    for generation in range(num_generations):
        fitness_values = evaluate_population(population, lambda genotype: decode_genotype(genotype,
bounds, precision))
        parents = select_parents(population, fitness_values)

        new_population = []
        for _ in range(population_size // 2):
            child1, child2 = crossover(parents)
            child1 = mutate(child1, mutation_rate)
            child2 = mutate(child2, mutation_rate)
            new_population.extend([child1, child2])

        population = np.array(new_population)

    # Select the best individual from the final population
    final_fitness_values = evaluate_population(population, lambda genotype:
decode_genotype(genotype, bounds, precision))
    best_individual = population[np.argmax(final_fitness_values)]
    decoded_best_solution = decode_genotype(best_individual, bounds, precision)
    return decoded_best_solution, func_a(decoded_best_solution)  # Change to func_b for the second
function

# Set parameters for the genetic algorithm
num_generations = 100
population_size = 50
genotype_size = 20  # Adjust based on the precision needed
bounds_a = [(-2, 2), (-2, 2)]
bounds_b = [(-2, 2), (-2, 2)]
precision = 8  # Number of bits for each variable
mutation_rate = 0.01

# Run the genetic algorithm for function a
result_a = genetic_algorithm(num_generations, population_size, genotype_size, bounds_a, precision,
mutation_rate)

# Run the genetic algorithm for function b
result_b = genetic_algorithm(num_generations, population_size, genotype_size, bounds_b, precision,
mutation_rate)

# Display results
print("Optimal solution for function a:", result_a)
print("Optimal solution for function b:", result_b)
```

```
Optimal solution for function a: ([2.0, 0.007843137254901933], -0.018314512240487583)
Optimal solution for function b: ([-2.0, 0.007843137254901933], -0.018314512240487583)
```

```python
#TASK 4
import random

def initialize_population(population_size, board_size):
    return [[random.randint(1, board_size) for _ in range(board_size)] for _ in
range(population_size)]

def calculate_penalty(configuration):
    return sum(configuration[i] == configuration[j] or abs(i - j) == abs(configuration[i] -
configuration[j]) for i in range(len(configuration)) for j in range(i + 1, len(configuration)))

def calculate_fitness(configuration):
    return 1 / (1 + calculate_penalty(configuration))

def select_parents(population, fitness_values):
    return [population[fitness_values.index(max(fitness_values))],
population[fitness_values.index(sorted(fitness_values)[-2])]]

def crossover(parents):
    crossover_point = random.randint(1, len(parents[0]) - 1)
    return parents[0][:crossover_point] + parents[1][crossover_point:],
parents[1][:crossover_point] + parents[0][crossover_point:]

def mutate(child, mutation_rate):
    return [random.randint(1, len(child)) if random.random() < mutation_rate else c for c in child]

def evolutionary_algorithm(num_generations, population_size, board_size, mutation_rate):
    population = initialize_population(population_size, board_size)

    for _ in range(num_generations):
        fitness_values = [calculate_fitness(ind) for ind in population]
        parents = [select_parents(population, fitness_values) for _ in range(population_size // 2)]

        population = [mutate(crossover(pair)[0], mutation_rate) for pair in parents] +
[mutate(crossover(pair)[1], mutation_rate) for pair in parents]

    return max(population, key=calculate_fitness), max(fitness_values)

# Set parameters for the evolutionary algorithm
num_generations = 1000
population_size = 50
board_size = 8
mutation_rate = 0.1

# Run the evolutionary algorithm
```

```
result, fitness = evolutionary_algorithm(num_generations, population_size, board_size,
mutation_rate)

# Display results
print("Optimal solution:", result)
print("Fitness of the optimal solution:", fitness)
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 08
Introduction to Supervised Learning

Objective:

This lab introduces the students to Supervised Learning. The lab talks about the basic libraries used for supervised learning within Python. The concepts are reinforced with the help of a simple KNN supervised learning model.

Name of Student: Sawera Fazal

Roll No: 21A-026-SE Sec.21A

Date of Experiment: 2 dec 23

Marks Obtained/Remarks: _____

Sign:

Student Exercise

Note: Each one of these tasks should be attempted in a separate notebook and all should be submitted in a NON-ZIP (NON-COMPRESSED) format on MsTeams

Task 1

During the data visualization phase, we constructed paired scatter plots among different attributes of the iris dataset. Construct a single three dimensional scatter plot among any three attributes of the iris dataset.

Task 2

Explore at least 2 more classification datasets present within scikit-learn. Implement the explored KNN classification mechanism, along with the data visualization techniques discussed on the 2 datasets.

Task3

Among the popular data sharing formats Excel file (xlsx), Comma Separated Values (CSV), and data file (.data) are three popular formats. You are provided with one dataset for each of these formats. Find a way to upload these datasets and get them into scikit-learn for exploration. Next perform some basic data visualization and try to fit in a KNN model for each of these classification problems. Evaluate your model's performance through the 80/20 training testing split.

(Note: you might have to use other libraries introduced at the start of the lab apart from scikit-learn).

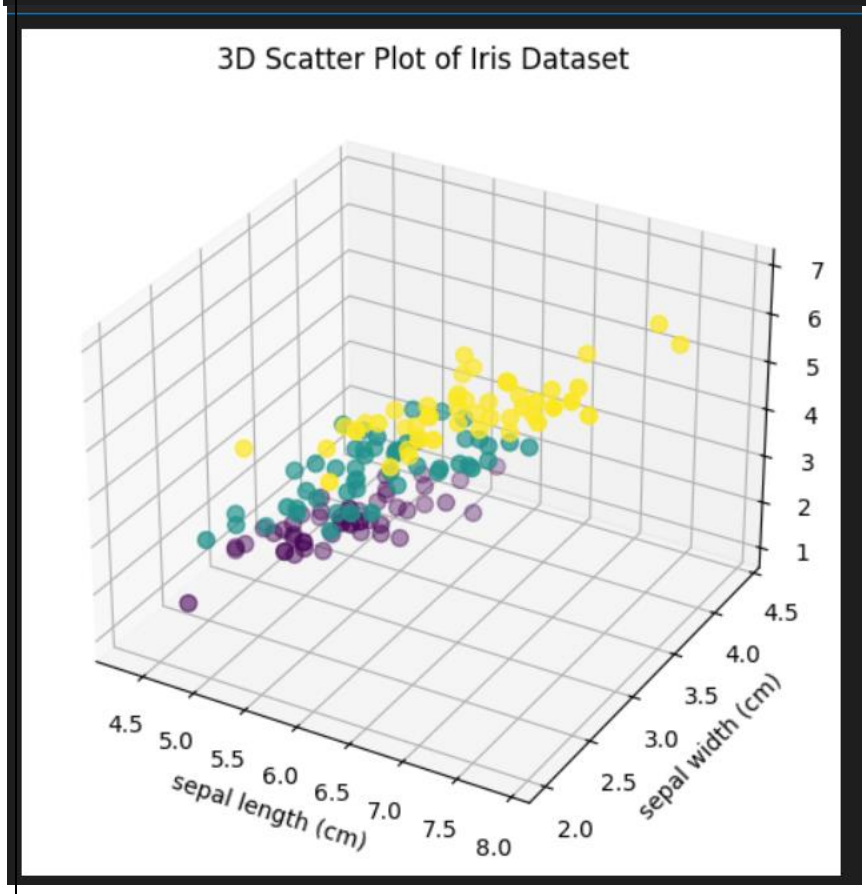
Task 4

Identify at least 3 famous data repositories where datasets for different machine learning exercises are available. Pick any one classification dataset from among these repositories and fit in a KNN classifier on your dataset. Report the accuracy of the model for different values of N (Neighbors).

Task1:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
```

```
data = iris.data
target = iris.target

# Select three attributes for the 3D scatter plot
attribute1 = 0  # Sepal Length
attribute2 = 1  # Sepal Width
attribute3 = 2  # Petal Length

# Create a 3D scatter plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(data[:, attribute1], data[:, attribute2], data[:, attribute3], c=target, cmap='viridis',
s=50)

# Set labels for each axis
ax.set_xlabel(iris.feature_names[attribute1])
ax.set_ylabel(iris.feature_names[attribute2])
ax.set_zlabel(iris.feature_names[attribute3])

plt.title('3D Scatter Plot of Iris Dataset')
plt.show()
```



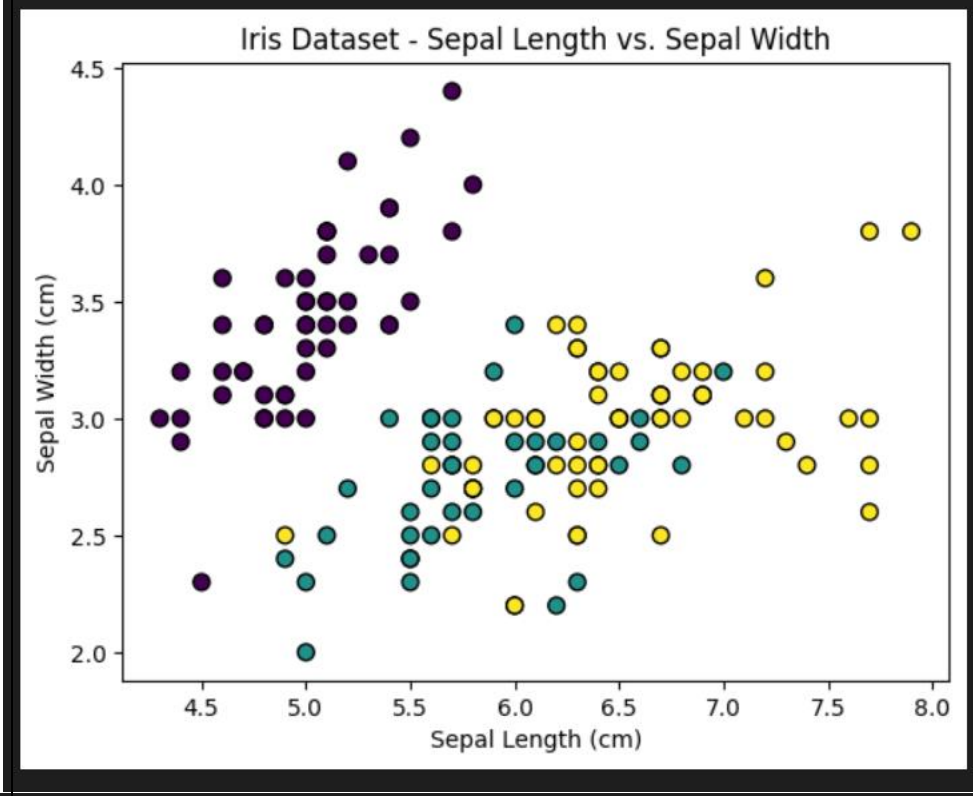
Task2:

```
#on iris
import numpy as np
```

```python
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = datasets.load_iris()
X_iris = iris.data
y_iris = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

# KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predictions on the test set
y_pred = knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Iris Dataset Accuracy: {accuracy:.2f}')

# Visualize the Iris dataset (you can choose any two features for simplicity)
plt.scatter(X_iris[:, 0], X_iris[:, 1], c=y_iris, cmap='viridis', edgecolor='k', s=50)
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Iris Dataset - Sepal Length vs. Sepal Width')
plt.show()
```

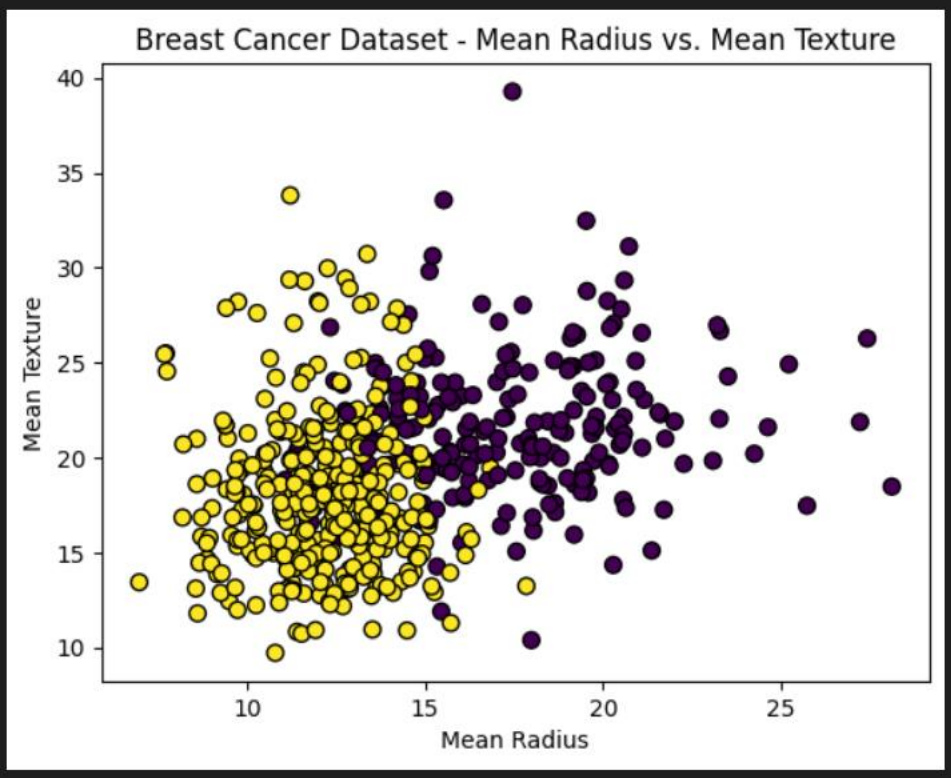
Lab 08: Introduction to Supervised Learning

```
Iris Dataset Accuracy: 1.00
```




```python
#on diabetes dataset
# Load Breast Cancer dataset
cancer = datasets.load_breast_cancer()
X_cancer = cancer.data
y_cancer = cancer.target

# Split the dataset into training and testing sets
X_train_cancer, X_test_cancer, y_train_cancer, y_test_cancer = train_test_split(X_cancer, y_cancer,
test_size=0.2, random_state=42)

# KNN Classifier
knn_cancer = KNeighborsClassifier(n_neighbors=3)
knn_cancer.fit(X_train_cancer, y_train_cancer)

# Predictions on the test set
y_pred_cancer = knn_cancer.predict(X_test_cancer)

# Calculate accuracy
accuracy_cancer = accuracy_score(y_test_cancer, y_pred_cancer)
print(f'Breast Cancer Dataset Accuracy: {accuracy_cancer:.2f}')

# Visualize the Breast Cancer dataset (you can choose any two features for simplicity)
plt.scatter(X_cancer[:, 0], X_cancer[:, 1], c=y_cancer, cmap='viridis', edgecolor='k', s=50)
plt.xlabel('Mean Radius')
plt.ylabel('Mean Texture')
plt.title('Breast Cancer Dataset - Mean Radius vs. Mean Texture')
plt.show()
```

```
Breast Cancer Dataset Accuracy: 0.93
```



Task3:

```python
#task3
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
iris_data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_data['target'] = iris.target

# Assuming the target column is 'target'
X_iris = iris_data.drop('target', axis=1)
y_iris = iris_data['target']

# Split the dataset into training and testing sets
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris,
test_size=0.2, random_state=42)

# KNN Classifier
knn_iris = KNeighborsClassifier(n_neighbors=3)
knn_iris.fit(X_train_iris, y_train_iris)

# Predictions on the test set
y_pred_iris = knn_iris.predict(X_test_iris)
```

```
# Calculate accuracy
accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
print(f'Iris Dataset Accuracy: {accuracy_iris:.2f}')

#task3
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
iris_data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_data['target'] = iris.target

# Assuming the target column is 'target'
X_iris = iris_data.drop('target', axis=1)
y_iris = iris_data['target']

# Split the dataset into training and testing sets
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris,
test_size=0.2, random_state=42)

# KNN Classifier
knn_iris = KNeighborsClassifier(n_neighbors=3)
knn_iris.fit(X_train_iris, y_train_iris)

# Predictions on the test set
y_pred_iris = knn_iris.predict(X_test_iris)

# Calculate accuracy
accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
print(f'Iris Dataset Accuracy: {accuracy_iris:.2f}')
```

Lab 08: Introduction to Supervised Learning

```
Iris Dataset Accuracy: 1.00
```

```
# Load CSV dataset
csv_data = pd.read_csv('diabetes.csv')

# Assuming the target column is 'target'
X_csv = csv_data.drop('Outcome', axis=1)
y_csv = csv_data['Outcome']

# Split the dataset into training and testing sets
X_train_csv, X_test_csv, y_train_csv, y_test_csv =

# KNN Classifier
knn_csv = KNeighborsClassifier(n_neighbors=3)
knn_csv.fit(X_train_csv, y_train_csv)

# Predictions on the test set
y_pred_csv = knn_csv.predict(X_test_csv)

# Calculate accuracy
accuracy_csv = accuracy_score(y_test_csv, y_pred_cs
print(f'CSV Dataset Accuracy: {accuracy_csv:.2f}')
```

0.0s

```
CSV Dataset Accuracy: 0.65
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import wget
from sklearn.datasets import load_svmlight_file

# Download the Breast Cancer dataset in LIBSVM format
url_breast_cancer = "https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/breast-cancer"
local_path_breast_cancer = "breast_cancer.data"
wget.download(url_breast_cancer, local_path_breast_cancer)

# Load Breast Cancer dataset in LIBSVM format (.data)
breast_cancer_data = load_svmlight_file(local_path_breast_cancer)

# Convert to Pandas DataFrame
breast_cancer_df = pd.DataFrame(data=breast_cancer_data[0].toarray(), columns=[f'feature_{i+1}' for
i in range(breast_cancer_data[0].shape[1])])
breast_cancer_df['target'] = breast_cancer_data[1]

# Assuming the target column is 'target'
X_bc = breast_cancer_df.drop('target', axis=1)
y_bc = breast_cancer_df['target']

# Split the dataset into training and testing sets
X_train_bc, X_test_bc, y_train_bc, y_test_bc = train_test_split(X_bc, y_bc, test_size=0.2,
```

```
random_state=42)

# KNN Classifier
knn_bc = KNeighborsClassifier(n_neighbors=3)
knn_bc.fit(X_train_bc, y_train_bc)

# Predictions on the test set
y_pred_bc = knn_bc.predict(X_test_bc)

# Calculate accuracy
accuracy_bc = accuracy_score(y_test_bc, y_pred_bc)
print(f'Breast Cancer Dataset Accuracy: {accuracy_bc:.2f}')
```

```
Breast Cancer Dataset Accuracy: 0.62
```

Task4:

```
#task4
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load Breast Cancer Wisconsin (Diagnostic) dataset from scikit-learn
breast_cancer = load_breast_cancer()

# Assuming the target column is 'target'
X_bc = breast_cancer.data
y_bc = breast_cancer.target

# Split the dataset into training and testing sets
X_train_bc, X_test_bc, y_train_bc, y_test_bc = train_test_split(X_bc, y_bc, test_size=0.2,
random_state=42)

# Test different values of N (Neighbors)
for n_neighbors in [1, 3, 5, 7]:
    # KNN Classifier
    knn_bc = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_bc.fit(X_train_bc, y_train_bc)

    # Predictions on the test set
    y_pred_bc = knn_bc.predict(X_test_bc)

    # Calculate accuracy
    accuracy_bc = accuracy_score(y_test_bc, y_pred_bc)
    print(f'Accuracy for N={n_neighbors}: {accuracy_bc:.2f}')
```

```
Accuracy for N=1: 0.93
Accuracy for N=3: 0.93
Accuracy for N=5: 0.96
Accuracy for N=7: 0.96
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 09
Data Pre-Processing and Performance Evaluation

Objective:

This lab introduces the students to the basic concepts of data preprocessing while implementing supervised machine learning based models. The lab also introduces some basic performance evaluation standards apart from model accuracy. The whole exploration is done using Naïve Bayes Classifiers.

Name of Student: Sawera Fazal

Roll No: 21A-026-SE **Sec.21A**

Date of Experiment: 3 dec 23

Marks Obtained/Remarks: _____

Student Exercise

Task 1

Implement Naïve Bayes Classifier on the scheduling problem discussed earlier in the manual.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
gnb = GaussianNB()

gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(y_pred)
```

```
[115. 134. 138. 150. 196.  59. 252. 196. 116. 138.  59. 152.  59. 217.
  49.  71. 220. 217. 138. 196. 142. 115. 134. 197.  90. 129. 138. 129.
  59.  71. 128. 115. 116. 129. 115. 217. 252.  71. 129. 134. 134.  63.
 124. 124. 152. 134. 134.  93.  59. 115.  59. 134. 138.  90. 217. 134.
 128. 281. 134.  64. 115. 142.  49.  63. 128. 115. 138. 138. 134. 124.
 138. 217. 152. 138. 134. 138. 217. 173.  63. 152. 134. 134.  59. 134.
 134. 182. 143. 134. 182.]
```

Task2

Download the Dermatology data set from the UCI depositary, apply Naïve Bayes classification and analyze the performance. Display the confusion matrix and discuss it. Vary the train/test split [(60%, 40%), (70%, 30%), (80%, 20%)] and discuss its effect if any.(source: https://archive.ics.uci.edu/ml/datasets/Dermatology)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix

dataset=pd.read_csv("dermdatology.csv")
dataset.head()

y=dataset['class']
x=dataset.drop('class',axis=1)

# 80% train and 20% test
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

gnb = GaussianNB()
```

```
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

print("80% train and 20% test")
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)

conf_matrix = confusion_matrix(y_test,y_pred)
print("\nConfusion Matrix")
print(conf_matrix)
```

```
80% train and 20% test
Accuracy: 0.9054054054054054

Confusion Matrix
[[31  0  0  0  0  0]
 [ 0  3  0  4  2  0]
 [ 0  0 13  0  0  0]
 [ 0  0  0  8  0  0]
 [ 0  1  0  0  9  0]
 [ 0  0  0  0  0  3]]
```

Task 3

Download the Bank authentication data set from the UCI depositary, apply KNN algorithmand Naïve Bayes Classifier analyze the performance. Display the confusion matrix and discuss it.
(source: https://archive.ics.uci.edu/ml/datasets/banknote+authentication)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix

dataset=pd.read_csv("dermdatology.csv")
dataset.head()

y=dataset['class']
x=dataset.drop('class',axis=1)

# 70% train and 30% test
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

gnb = GaussianNB()

gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)
```

```
conf_matrix = confusion_matrix(y_test,y_pred)
print("\nConfusion Matrix")
print(conf_matrix)
```

```
Accuracy: 0.8727272727272727

Confusion Matrix
[[40  0  0  0  0  0]
 [ 0  3  0 11  2  0]
 [ 0  0 19  0  0  0]
 [ 0  0  0 15  0  0]
 [ 0  1  0  0 15  0]
 [ 0  0  0  0  0  4]]
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix

dataset=pd.read_csv("dermdatology.csv")
dataset.head()

y=dataset['class']
x=dataset.drop('class',axis=1)

# 60% train and 40% test
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)

gnb = GaussianNB()

gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy)

conf_matrix = confusion_matrix(y_test,y_pred)
print("\nConfusion Matrix")
print(conf_matrix)
```

```
[23]
...  Accuracy: 0.8775510204081632

     Confusion Matrix
     [[52  0  0  0  0  0]
      [ 0  7  0 13  4  0]
      [ 0  0 27  0  0  0]
      [ 0  0  0 18  0  0]
      [ 0  1  0  0 20  0]
      [ 0  0  0  0  0  5]]
```

```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00267/data_banknote_authentication.txt"
column_names = ["variance", "skewness", "curtosis", "entropy", "class"]
data = pd.read_csv(url, names=column_names)

# Separate features and target variable
X = data.drop("class", axis=1)
y = data["class"]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply KNN algorithm
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
accuracy =  accuracy_score(y_test, knn_predictions)
print("Accuracy for KNN:", accuracy)
confusion = confusion_matrix(y_test, knn_predictions)
print(f"Confusion Matrix for KNN:\n", confusion)
print("")


# Apply Naïve Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)
accuracy =  accuracy_score(y_test, nb_predictions)
print("Accuracy for Nave Bayes:", accuracy)
confusion = confusion_matrix(y_test, nb_predictions)
print(f"Confusion Matrix for Naive Bayes:\n", confusion)
```

```
print("")
```

```
Accuracy for KNN: 1.0
Confusion Matrix for KNN:
 [[148   0]
 [  0 127]]

Accuracy for Nave Bayes: 0.8072727272727273
Confusion Matrix for Naive Bayes:
 [[133  15]
 [ 38  89]]

Accuracy for Naïve Bayes: 0.8072727272727273
Confusion Matrix for Naïve Bayes:
 [[133  15]
 [ 38  89]]
```

Task 4

Apply both Naïve Bayes and KNN on the dataset of your choice. Compare and analyze theperformance using F-measure and Accuracy.

```
from sklearn.metrics import f1_score, accuracy_score

# Evaluating F1 score of KNN Model
print("Knn Model F1 score")
f_measure = f1_score(y_test, knn_predictions, average='weighted')
print(f"F-measure: {f_measure}")
print("")

# Evaluating F1 score of Namiva byes Model
print("Naive Bayes Model F1 score")
f_measure = f1_score(y_test, nb_predictions, average='weighted')
print(f"F-measure: {f_measure}")
```

[5]

```
Knn Model F1 score
F-measure: 1.0

Naive Bayes Model F1 score
F-measure: 0.8046259278861161
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 10
Linear Regression & Decision Tree

Objective:

This experiment introduces the students to the basics of Linear Regression and Decision Tree models. The lab explains the induction of a decision tree. It also discusses how regression can be used to make predictions on different data sets and used as a tool for the Supervised form of Artificial Intelligence.

Name of Student: **Sawera Fazal**

Roll No: **21A-026-SE** **Sec.21A**

Date of Experiment: **10 dec 23**

Marks Obtained/Remarks: _____

Sign:

Lab 08: Linear Regression & Decision Tree

Student Exercise

Task 1

Implement the examples given in the manual and elaborate the results.

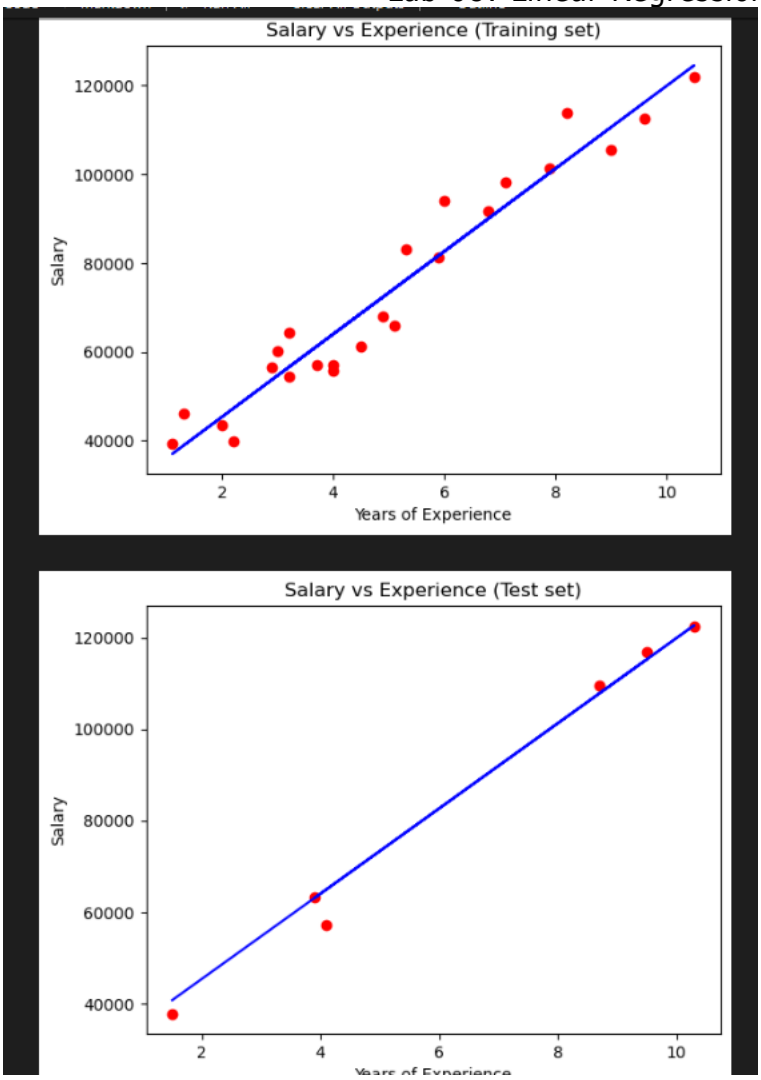
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Import the dataset
dataset = pd.read_csv('Salary_Data.csv')

print(dataset)

# Extracting features and target variable
X = dataset['YearsExperience'].values.reshape(-1, 1)
y = dataset['Salary'].values.reshape(-1, 1)

dataset.plot(x='YearsExperience', y='Salary', style='o')
plt.title('YearsExperience VS Salary')  # Use plt.title instead of plot.title
plt.xlabel('Years Experience')
plt.ylabel('Salary')
plt.show()

# Split the dataset into the training set and test set
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.2, random_state=0)

# Create a linear regression model
linearRegressor = LinearRegression()

# Fit the model to the training data
linearRegressor.fit(xTrain, yTrain)

yPrediction = linearRegressor.predict(xTest)

df = pd.DataFrame({'Actual': yTest.flatten(), 'Predicted': yPrediction.flatten()})

print(df)

plt.scatter(xTrain, yTrain, color = 'red')
plt.plot(xTrain, linearRegressor.predict(xTrain), color = 'blue')
plt.title('Salary vs Experience (Training set)')
```

```
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

plt.scatter(xTest, yTest, color = 'red')
plt.plot(xTest, linearRegressor.predict(xTest), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

print('Mean Absolute Error:', metrics.mean_absolute_error(yTest, yPrediction))
print('Mean Squared Error:', metrics.mean_squared_error(yTest, yPrediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yTest, yPrediction)))
```

```
...    YearsExperience    Salary
0                  1.1   39343.0
1                  1.3   46205.0
2                  1.5   37731.0
3                  2.0   43525.0
4                  2.2   39891.0
5                  2.9   56642.0
6                  3.0   60150.0
7                  3.2   54445.0
8                  3.2   64445.0
9                  3.7   57189.0
10                 3.9   63218.0
11                 4.0   55794.0
12                 4.0   56957.0
13                 4.1   57081.0
14                 4.5   61111.0
15                 4.9   67938.0
16                 5.1   66029.0
17                 5.3   83088.0
18                 5.9   81363.0
19                 6.0   93940.0
20                 6.8   91738.0
21                 7.1   98273.0
22                 7.9  101302.0
23                 8.2  113812.0
...
26                 9.5  116969.0
27                 9.6  112635.0
28                10.3  122391.0
29                10.5  121872.0
```

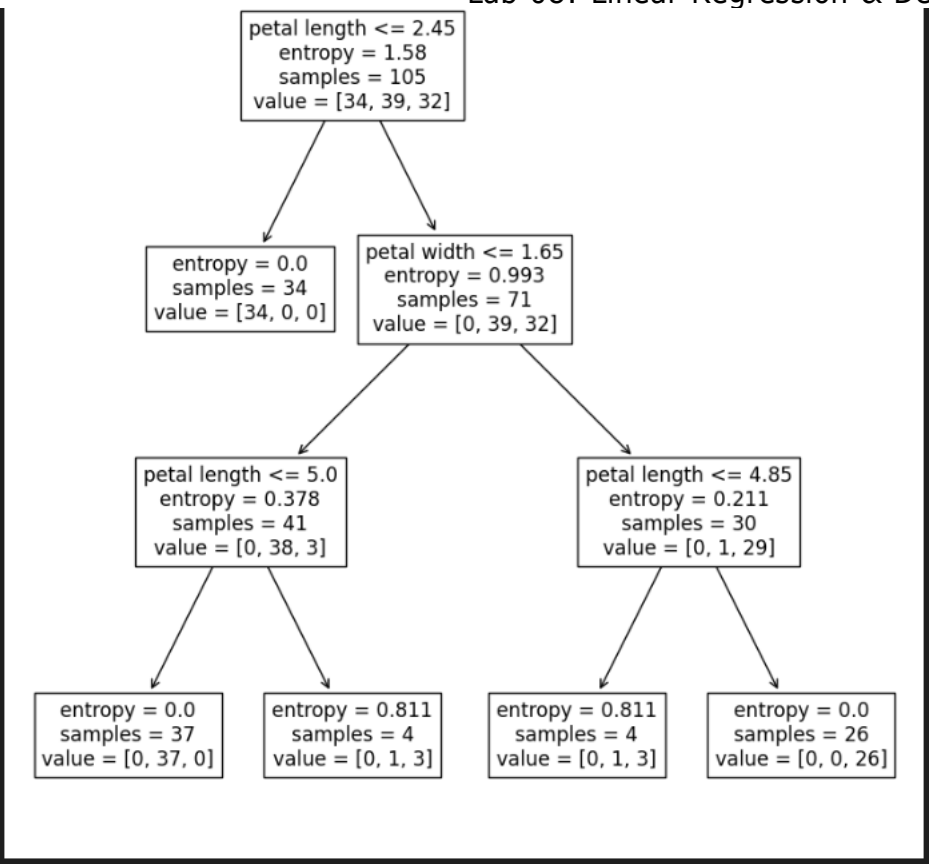

```
      Actual      Predicted
0   37731.0   40748.961841
1  122391.0  122699.622956
2   57081.0   64961.657170
```

Lab 08: Linear Regression & Decision Tree




```python
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn import tree
from matplotlib import pyplot as plt

names = ['Sepal Length', 'Sepal Width','Petal length','Petal Width','Species']
data = pd.read_csv('Iris.csv', names=names)
df = pd.DataFrame(data)
X = df.drop(['Species'], axis=1)
Y = df['Species']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 100)

#build decision tree
clf_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=4,min_samples_leaf=4)
#max_depth represents max level allowed in each tree, min_samples_leaf minumum samples storable in
leaf node
```

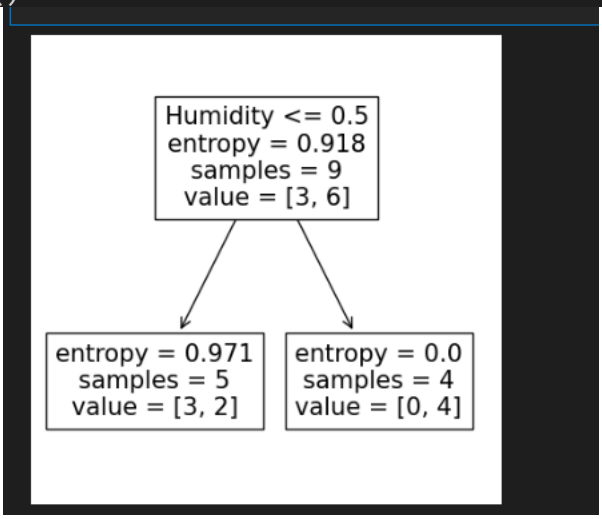
```
clf_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4,min_samples_leaf=4)
#fit the tree to iris dataset
clf_gini.fit(X_train,y_train)
clf_entropy.fit(X_train,y_train)

#plot decision tree
fig, ax = plt.subplots(figsize=(10, 10)) #figsize value changes the size of plot
tree.plot_tree(clf_entropy,ax=ax,feature_names=['sepal length','sepal width','petal length','petal
width'])
plt.show()

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values on:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))

y_pred_gini = prediction(X_test,clf_gini)
#y_pred_entropy = prediction(X_test,clf_entropy)
df_pred=pd.DataFrame()
df_pred['y_test']=y_test
df_pred['y_pred']=y_pred_gini
df_pred.head(n=10)

cal_accuracy(y_test,y_pred_gini)
```




```
Predicted values on:
['Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa'
 'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa'
```

```python
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn import tree
from matplotlib import pyplot as plt

le = LabelEncoder()
data = pd.read_csv('PlayTennis.csv')
df = pd.DataFrame(data)
df['Outlook']= le.fit_transform(df['Outlook'])
df['Temperature']= le.fit_transform(df['Temperature'])
df['Humidity']= le.fit_transform(df['Humidity'])
df['Wind']= le.fit_transform(df['Wind'])
df['Play Tennis']= le.fit_transform(df['Play Tennis'])
X = df.drop(['Play Tennis'],axis=1)
y = df['Play Tennis']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
```

```
df.head()
df.columns

#build decision tree
clf_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=4,min_samples_leaf=4)
#max_depth represents max level allowed in each tree, min_samples_leaf minumum samples storable in
leaf node
clf_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4,min_samples_leaf=4)
#fit the tree to iris dataset
clf_gini.fit(X_train,y_train)
clf_entropy.fit(X_train,y_train)

#plot decision tree
fig, ax = plt.subplots(figsize=(5, 5)) #figsize value changes the size of plot
tree.plot_tree(clf_entropy,ax=ax,feature_names=['Outlook', 'Temperature', 'Humidity', 'Wind', 'Play
Tennis'])
plt.show()
```



Task 2

Apply linear regression on the dataset provided with the manual (weather.xlsx) and predict the minimum temperature based on the maximum temperature (Two Columns Only). Elaborate the results.

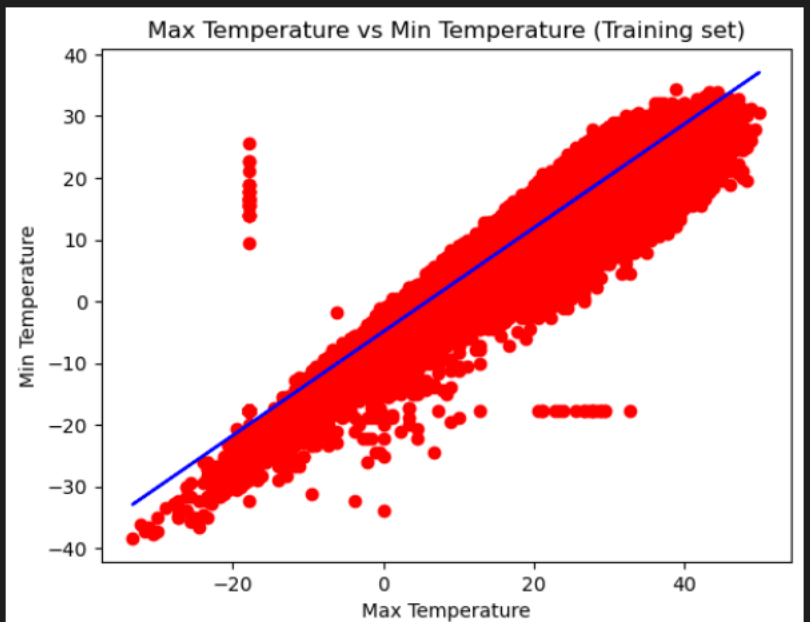
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

# Import the dataset
dataset = pd.read_csv('Weather.csv')
```

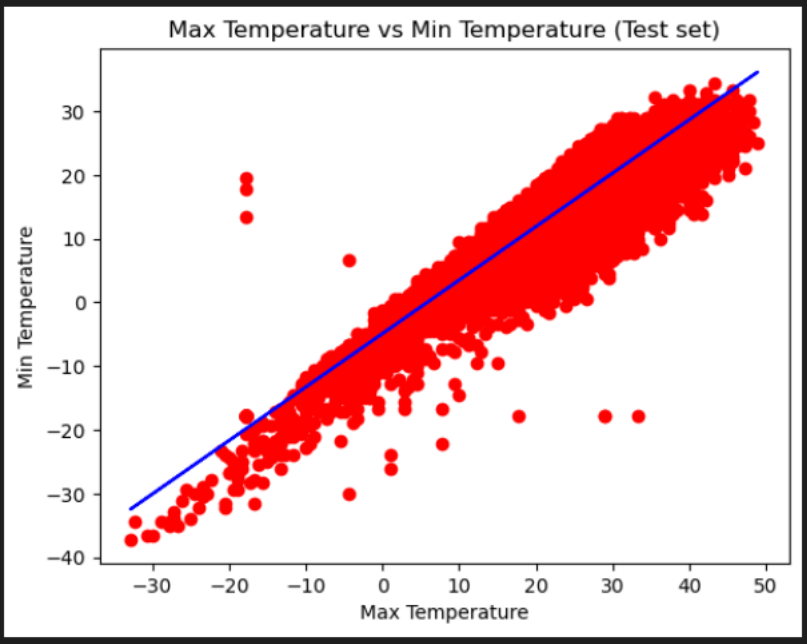
```
# Extracting features and target variable
X = dataset['MaxTemp'].values.reshape(-1, 1)
y = dataset['MinTemp'].values.reshape(-1, 1)

# Split the data into training and testing sets
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
linearRegressor = LinearRegression()

# Fit the model to the training data
linearRegressor.fit(xTrain, yTrain)

yPrediction = linearRegressor.predict(xTest)

df_results = pd.DataFrame({'Actual': yTest.flatten(), 'Predicted': yPrediction.flatten()})

print(df_results)

plt.scatter(xTrain, yTrain, color='red')
plt.plot(xTrain, linearRegressor.predict(xTrain), color='blue')
plt.title('Max Temperature vs Min Temperature (Training set)')
plt.xlabel('Max Temperature')
plt.ylabel('Min Temperature')
plt.show()

plt.scatter(xTest, yTest.flatten(), color='red')  # Updated here to use yTest.flatten()
plt.plot(xTest, yPrediction.flatten(), color='blue')  # Updated here to use yPrediction.flatten()
plt.title('Max Temperature vs Min Temperature (Test set)')
plt.xlabel('Max Temperature')
plt.ylabel('Min Temperature')
plt.show()

print('Mean Absolute Error:', metrics.mean_absolute_error(yTest, yPrediction))
print('Mean Squared Error:', metrics.mean_squared_error(yTest, yPrediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yTest, yPrediction)))
```

Code + Markdown | Run All | Clear All Outputs | Outline ...

	Actual	Predicted
0	18.333333	20.275142
1	21.111111	19.342373
2	21.111111	18.875989
3	26.666667	23.073449
4	27.222222	27.737294
...	...	...
23803	-4.444444	-3.044082
23804	-1.666667	-0.245775
23805	19.444444	19.342373
23806	21.111111	19.342373
23807	24.444444	18.875989

[23808 rows x 2 columns]





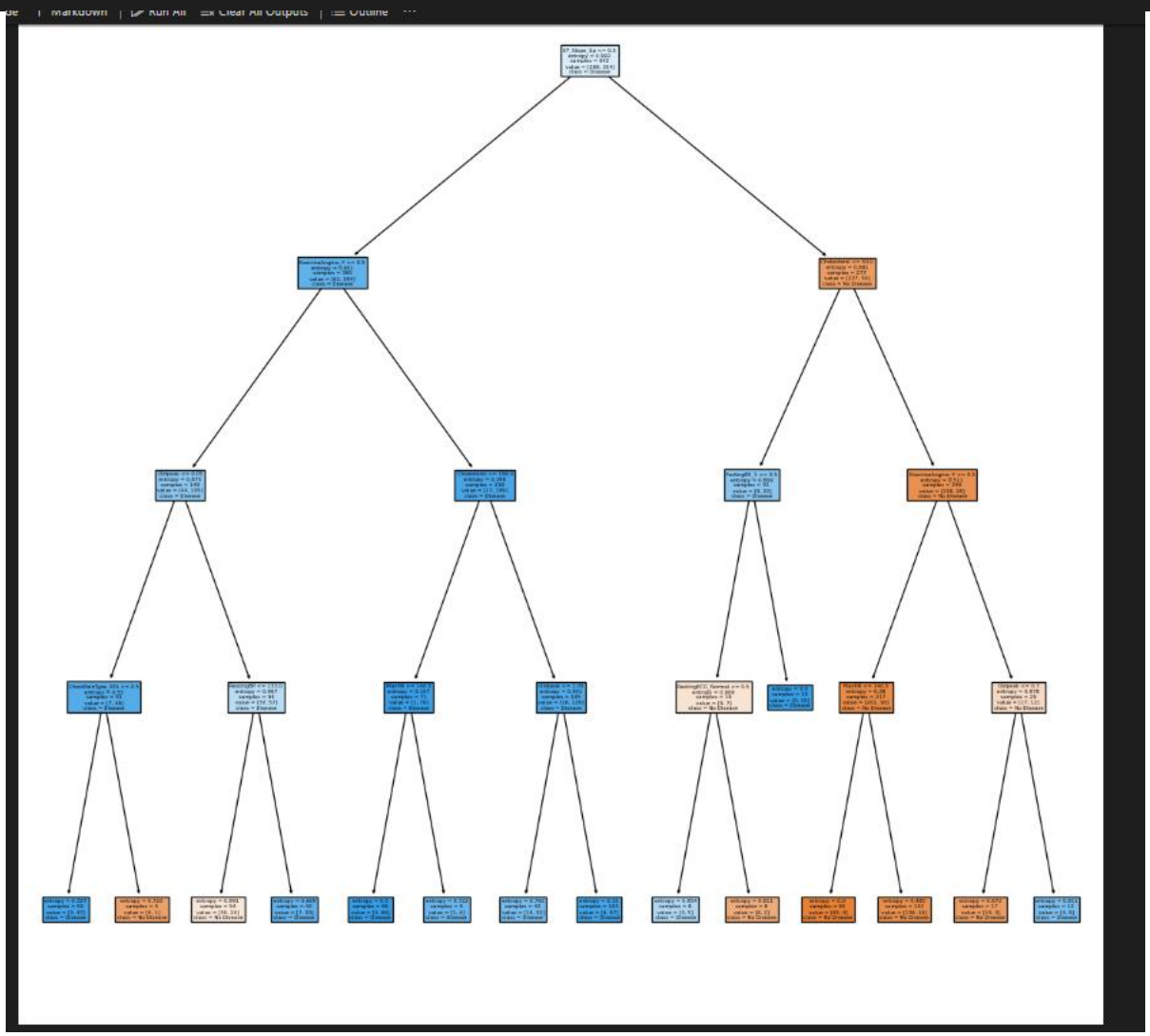

```
Mean Absolute Error: 3.0902650416857216
Mean Squared Error: 15.67229406673744
Root Mean Squared Error: 3.958824834055864
```

Task 3

For the heart disease dataset (heart.csv), use the categorical variable to learn a decision tree to classify the records. You model should be learnt against a 70-30 testing training split. All the performance evaluation matrices should be reported.

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

heart_data = pd.read_csv('heart.csv')


# Identify categorical columns for decision tree
categorical_columns = ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseAngina',
'ST_Slope']

# Separate features (X) and target variable (Y)
X = heart_data.drop('HeartDisease', axis=1)
Y = heart_data['HeartDisease']

# One-hot encode categorical columns
X_encoded = pd.get_dummies(X, columns=categorical_columns, drop_first=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, Y, test_size=0.3, random_state=100)

# Build decision tree classifiers
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=4, min_samples_leaf=4)
clf_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_leaf=4)

# Fit the decision tree models
clf_gini.fit(X_train, y_train)
clf_entropy.fit(X_train, y_train)

# Plot decision tree (using entropy criterion)
fig, ax = plt.subplots(figsize=(15, 15))
tree.plot_tree(clf_entropy, ax=ax, feature_names=X_encoded.columns, class_names=['No Disease',
'Disease'], filled=True)
plt.show()

# Function to make predictions
def prediction(X_test, clf_object):
```

```
    y_pred = clf_object.predict(X_test)
    print("Predicted values on:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
    print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
    print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
    print("Report : ", classification_report(y_test, y_pred))

# Make predictions and evaluate performance
y_pred_gini = prediction(X_test, clf_gini)
df_pred = pd.DataFrame({'y_test': y_test, 'y_pred': y_pred_gini})
df_pred.head(n=10)

cal_accuracy(y_test, y_pred_gini)
```



```
Predicted values on:
[1 1 0 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 1 1 1 1
 1 0 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 1 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 1 0 0 0 1 1
 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0
 0 0 0 1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 1
 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0
 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 1 1 0
 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1]
Confusion Matrix:  [[ 97  25]
 [ 19 135]]
Accuracy :  84.05797101449275
Report :               precision    recall  f1-score   support

           0       0.84      0.80      0.82       122
           1       0.84      0.88      0.86       154

    accuracy                           0.84       276
   macro avg       0.84      0.84      0.84       276
weighted avg       0.84      0.84      0.84       276
```

Task 4

Explore the Weather_Notebook_Kaggle provided to you along with the lab material.

```python
def overview():
    data = pd.read_csv("Weather.csv")
    # Print the first 5 lines of data
    print("First 5 lines of data \n\n")
    print(data.head())

    # Print data type
    print("\n\n\nDatatype\n")
    print(data.dtypes)

    # Print number of null values
    print("\n\n\nNumber of null values\n")
    print(data.isnull().sum())

    # Print data summary
    print("\n\n\nData summary\n")
    print(data.describe())

    # Print data shape
    print("\n\n\nData shape\n")
    print("Data has {} rows and {} columns".format(data.shape[0], data.shape[1]))

    return data

data = overview()
```

```
First 5 lines of data


     STA      Date Precip  WindGustSpd    MaxTemp    MinTemp   MeanTemp  \
0  10001  1942-7-1  1.016          NaN  25.555556  22.222222  23.888889
1  10001  1942-7-2      0          NaN  28.888889  21.666667  25.555556
2  10001  1942-7-3   2.54          NaN  26.111111  22.222222  24.444444
3  10001  1942-7-4   2.54          NaN  26.666667  22.222222  24.444444
4  10001  1942-7-5      0          NaN  26.666667  21.666667  24.444444

   Snowfall PoorWeather  YR  ...  FB  FTI ITH  PGT  TSHDSBRSGF  SD3  RHX  RHN  \
0       0.0         NaN  42  ... NaN  NaN NaN  NaN         NaN  NaN  NaN  NaN
1       0.0         NaN  42  ... NaN  NaN NaN  NaN         NaN  NaN  NaN  NaN
2       0.0         NaN  42  ... NaN  NaN NaN  NaN         NaN  NaN  NaN  NaN
3       0.0         NaN  42  ... NaN  NaN NaN  NaN         NaN  NaN  NaN  NaN
4       0.0         NaN  42  ... NaN  NaN NaN  NaN         NaN  NaN  NaN  NaN

  RVG  WTE
0 NaN  NaN
1 NaN  NaN
2 NaN  NaN
3 NaN  NaN
4 NaN  NaN

[5 rows x 31 columns]
...

Data shape
```

```
# Dropping NaN rows
data = data.dropna(subset = ['Snowfall', 'PRCP', 'MAX', 'MIN', 'MEA', 'SNF'])

# Dropping redundant column
data = data.drop(columns = ['PRCP'])

# Dropping NaN columns
data = data.dropna(axis = 'columns')

# Taking a look at whats left
data.isnull().sum()
```

[21]

```
STA         0
Date        0
Precip      0
MaxTemp     0
MinTemp     0
MeanTemp    0
Snowfall    0
YR          0
MO          0
DA          0
MAX         0
MIN         0
MEA         0
SNF         0
dtype: int64
```

```
def cat_variable(df):
    return list(df.select_dtypes(include = ['category', 'object']))

def num_variable(df):
    return list(df.select_dtypes(exclude = ['category', 'object']))

categorical_variable = cat_variable(data)
numerical_variable = num_variable(data)

# Create a function to process outlier data
def outlier(data):
    z = np.abs(stats.zscore(data[numerical_variable]))
    z_data = data[(z < 3).all(axis=1)] # Remove any outliers with Z-score > 3 or < -3
    return z_data

data = outlier(data)
```

[22]

```
# Removing non-numeric data and converting date to datetime format
data['Precip'] = pd.to_numeric(data['Precip'], errors='coerce')
data['Year'] = pd.DatetimeIndex(data['Date']).year
data.applymap(np.isreal)
```

[23]

...

	STA	Date	Precip	MaxTemp	MinTemp	MeanTemp	Snowfall	YR	MO	DA	MAX	MIN	MEA	SNF	Year
0	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True
1	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True
2	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True
3	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True
4	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
119035	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True
119036	True	False	True	True	True	True	True	True	True	True	True	True	True	True	True

```
    print('Before cleaning: \n\n{}'.format(data.isnull().sum()))

    data = data.fillna(method='ffill').fillna(method='bfill')

    print('\nAfter cleaning: \n\n{}'.format(data.isnull().sum()))
```

[24]

```
Before cleaning:

STA            0
Date           0
Precip     15549
MaxTemp        0
MinTemp        0
MeanTemp       0
Snowfall       0
YR             0
MO             0
DA             0
MAX            0
MIN            0
MEA            0
SNF            0
Year           0
dtype: int64

After cleaning:

STA        0
Date       0
Precip     0
MaxTemp    0
...
MEA        0
SNF        0
Year       0
dtype: int64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
data = data.set_index('Date')
axes = data[["MaxTemp", "MinTemp", "MeanTemp", "Precip"]].plot(figsize=(11, 9), subplots=True, linewidth=1)
```

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 11
Introduction To Unsupervised Learning

Objective:

This lab provides a comprehensive Introduction about different Unsupervised learning algorithm

Name of Student: Sawera Fazal

Roll No: 21A-026-SE **Sec.21A**

Date of Experiment: 10 dec 23

Marks Obtained/Remarks: _____

Sign _____

Lab Task

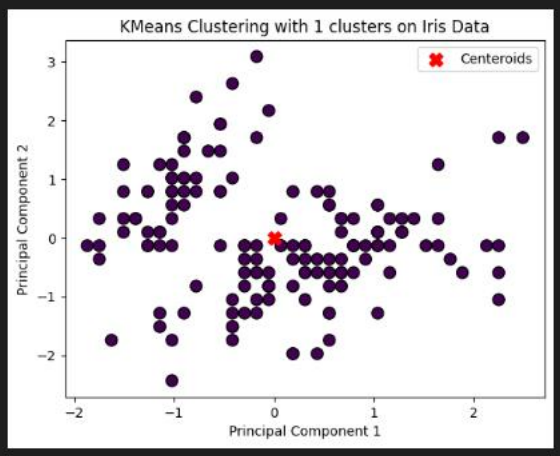
Task 1: Exploratory Data Analysis (EDA) and K-means Clustering

Load the Iris dataset from scikit-learn (load_iris). Perform exploratory data analysis to understand the structure of the dataset.

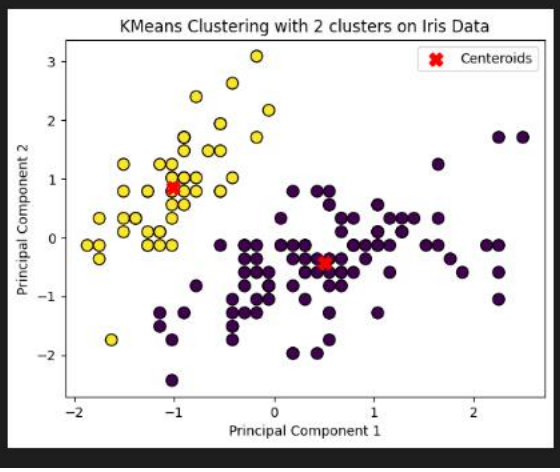
Use K-means clustering to identify clusters in the Iris dataset. Experiment with different values of K and visualize the results using scatter plots.

Apply the Elbow Method to determine the optimal number of clusters for the Iris dataset.

```
super()._check_params_vs_input(X, default_n_init=10)
```

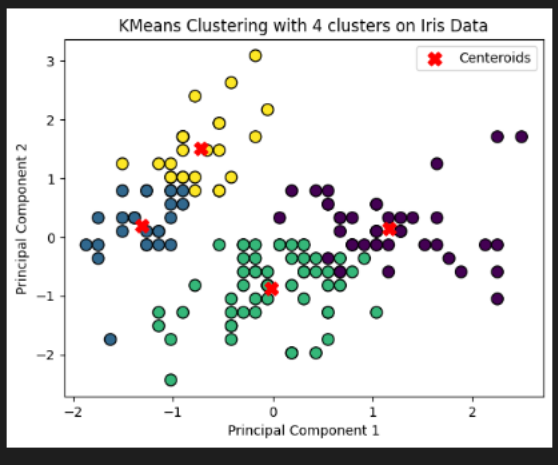


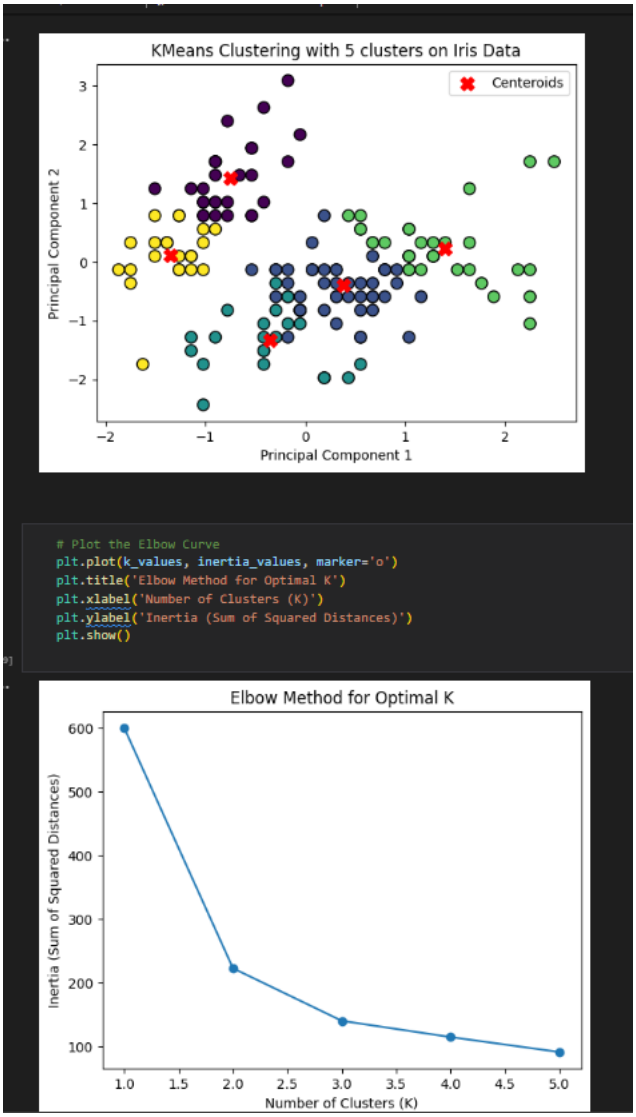

```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2
  super()._check_params_vs_input(X, default_n_init=10)
```



```
  super()._check_params_vs_input(X, default_n_init=10)
```


```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p
  super()._check_params_vs_input(X, default_n_init=10)
```





Task 2: Dimensionality Reduction and Visualization

Load the Wine dataset from scikit-learn (load_wine). Apply Principal Component Analysis (PCA) to reduce the dimensionality of the dataset to two components.

Visualize the reduced Wine dataset in a 2D scatter plot, coloring the points based on their class labels.

Experiment with different clustering algorithms (e.g., K-means, hierarchical clustering) on the reduced Wine dataset and compare the results.

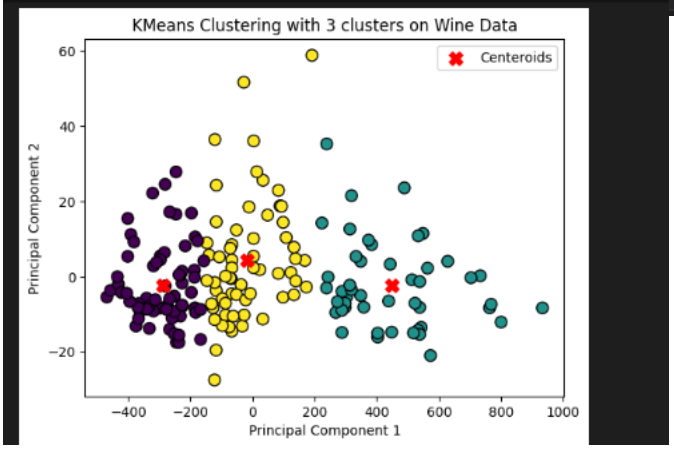
Task 2: Dimensionality Reduction and Visualization

Load the Wine dataset from scikit-learn (load_wine). Apply Principal Component Analysis (PCA) to reduce the dimensionality of the dataset to two components. V
clustering) on the reduced Wine dataset and compare the results.

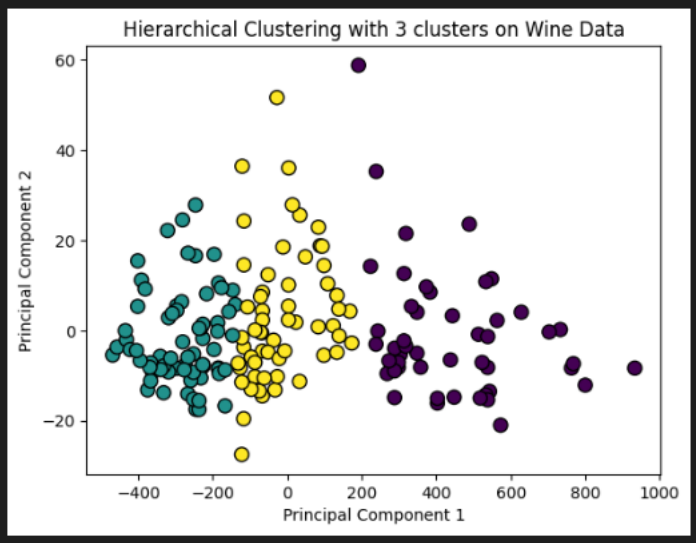
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA


# Load the Iris dataset
wine = load_wine()
X_wine = wine.data
y_wine = wine.target

num_clusters = 3

# Reduce the data to 2D for visualization (using PCA)
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X_wine)
```

```
kmeans = KMeans(n_clusters=num_clusters, random_state=1)
labelsK = kmeans.fit_predict(X_2d)

plt.scatter(X_2d[:, 0], X_2d[:, 1], c=labelsK, cmap='viridis', edgecolor='k', s=75)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red', marker = 'X', s = 100, label = 'Centeroids')
plt.title(f'KMeans Clustering with {num_clusters} clusters on Wine Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

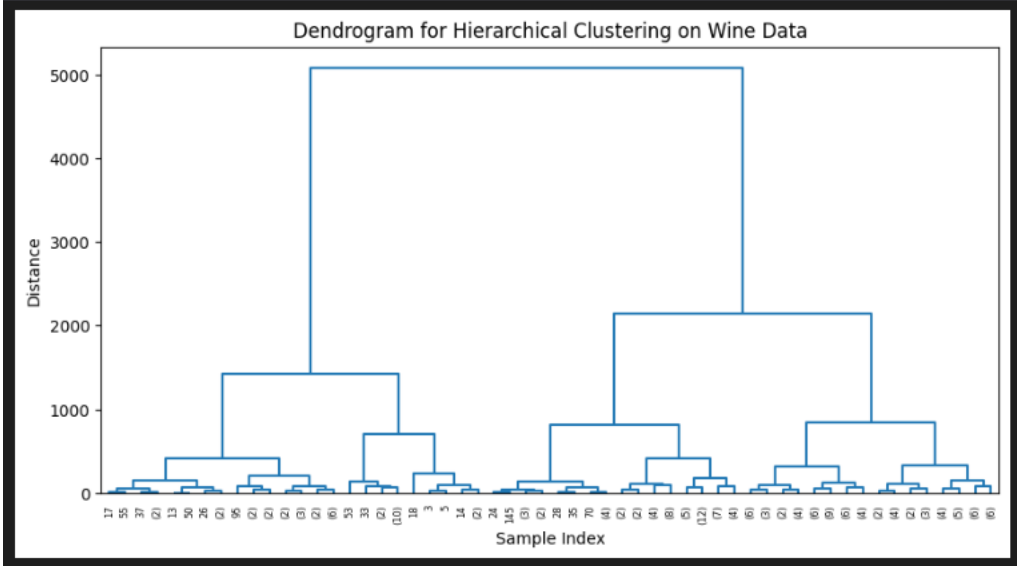
de + Markdown | Run All | Clear All Outputs | Outline ···



```python
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

# Use AgglomerativeClustering instead of KMeans
agg_clustering = AgglomerativeClustering(n_clusters=num_clusters)
labelsH = agg_clustering.fit_predict(X_2d)

# Plot the clusters
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=labelsH, cmap='viridis', edgecolor='k', s=75)
plt.title(f'Hierarchical Clustering with {num_clusters} clusters on Wine Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



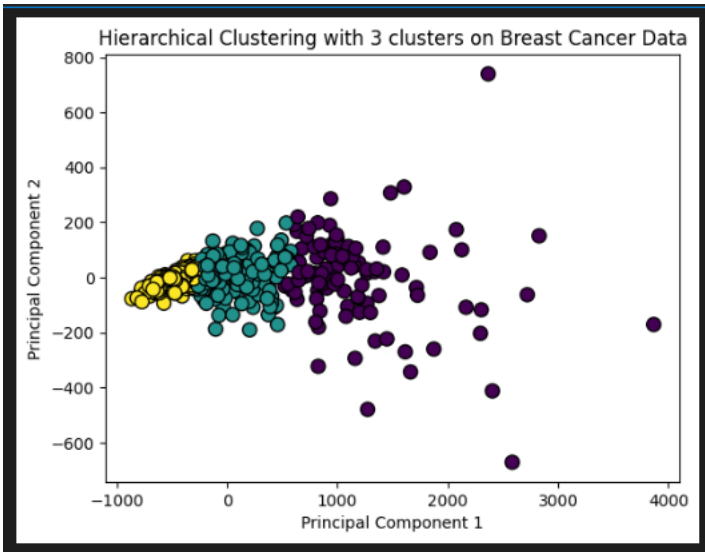

```python
from scipy.cluster.hierarchy import dendrogram, linkage

# Calculate linkage matrix
linkage_matrix = linkage(X_2d, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, p=5, truncate_mode='level', color_threshold=0)
plt.title('Dendrogram for Hierarchical Clustering on Wine Data')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



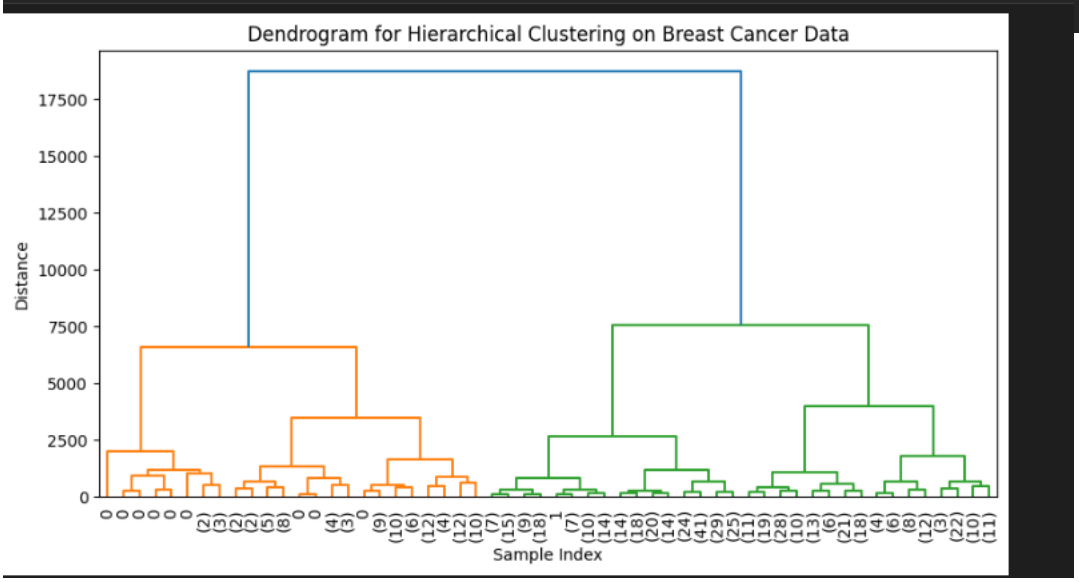
Task 3: Hierarchical Clustering

Load the Breast Cancer dataset from scikit-learn (load_breast_cancer). Apply hierarchical clustering to identify clusters. Visualize the resulting dendrogram and discuss the potential number of clusters.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA


# Load the Iris dataset
breast_cancer = load_breast_cancer()
X_breast_cancer = breast_cancer.data
y_breast_cancer = breast_cancer.target

num_clusters = 3

# Reduce the data to 2D for visualization (using PCA)
pca = PCA(n_components=2)
X_PCA = pca.fit_transform(X_breast_cancer)
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

num_clusters = 3

# Use AgglomerativeClustering instead of KMeans
agg_clustering = AgglomerativeClustering(n_clusters=num_clusters)
labelsBC = agg_clustering.fit_predict(X_PCA)

# Plot the clusters
plt.scatter(X_PCA[:, 0], X_PCA[:, 1], c=labelsBC, cmap='viridis', edgecolor='k', s=75)
plt.title(f'Hierarchical Clustering with {num_clusters} clusters on Breast Cancer Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```




```python
from scipy.cluster.hierarchy import dendrogram, linkage

# Calculate linkage matrix
linkage_matrix = linkage(X_PCA, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, labels=y_breast_cancer, leaf_rotation=90, leaf_font_size=10, truncate_mode='level', p=5)
plt.title('Dendrogram for Hierarchical Clustering on Breast Cancer Data')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
```

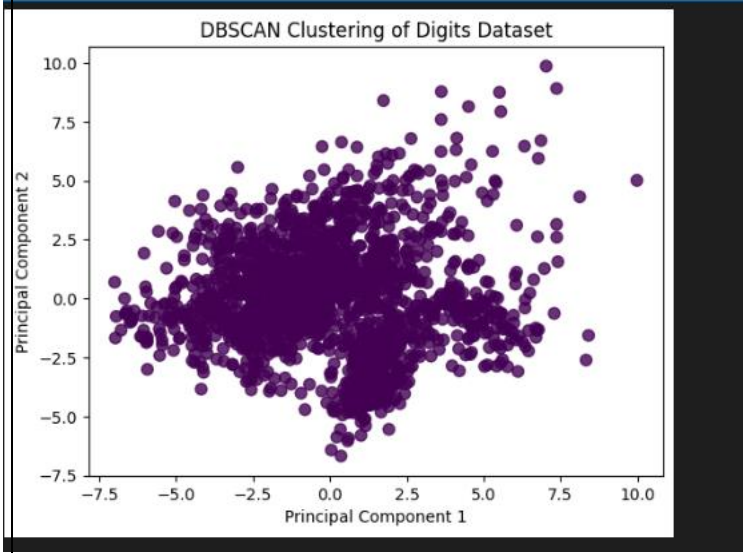


Task 4: Density-Based Clustering

Load the Digits dataset from scikit-learn (load_digits). Apply DBSCAN (Density-Based Spatial Clustering of Applications with Noise) on the dataset. Discuss the strengths and weaknesses of DBSCAN compared to K-means.

```python
import matplotlib.pyplot as plt
```

```python
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA

# Load the Digits dataset
digits = load_digits()
X = digits.data
y = digits.target

# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=1, min_samples=5)  # You may need to adjust parameters based on your data
labels = dbscan.fit_predict(X_std)

# Visualize the clusters using PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_std)

# Scatter plot the points with color-coded clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis', marker='o', s=50, alpha=0.8)
plt.title('DBSCAN Clustering of Digits Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 12
Introduction to Neural Networks

Objective:

Objective: The Neural Network Basics Lab aims to impart a foundational understanding of neural networks, covering key concepts such as architecture, activation functions, and the implementation of a simple perceptron in Python using NumPy. Through hands-on tasks, participants will enhance their skills in modifying activation functions, handling multi-feature perceptrons, introducing bias, and applying neural networks to real-world datasets, fostering a practical understanding of neural network fundamentals.

Name of Student: Sawera Fazal

Roll No: 21A-026-SE **Sec.21A**

Date of Experiment: 17 dec 23

Marks Obtained/Remarks:

Lab Tasks¶

Task 1: Expand the Perceptron Function

Modify the perceptron function to use a different activation function, such as the sigmoid function or the hyperbolic tangent (tanh) function. Update the hands-on code to test the modified perceptron function with the new activation function. Compare the results and discuss the differences in the decision boundaries.

Task 1: Expand the Perceptron Function Modify the perceptron function to use a different activation function, such as the sigmoid functio
hands-on code to test the modified perceptron function with the new activation function. Compare the results and discuss the difference

```python
# Import NumPy
import numpy as np

# Define the perceptron function
def perceptron(input_data, weights):
    # Calculate the weighted sum
    weighted_sum = np.dot(input_data, weights)
    z = 1/(1 + np.exp(-weighted_sum))
    # Apply a simple step function as activation
    output = 1 if z> 0 else 0

    return output

# Sample data
input_data = np.array([1, 2])
weights = np.array([0.5, -0.5])

# Test the perceptron
output = perceptron(input_data, weights)
print("Perceptron Output:", output)
```

[4]

```
Perceptron Output: 1
```

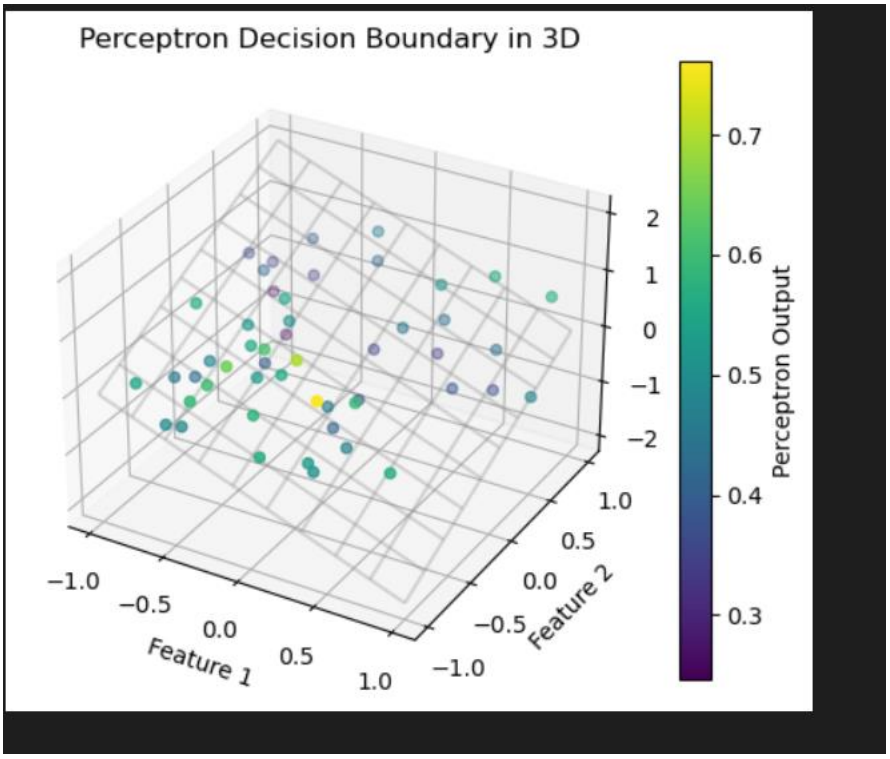
```python
# Import NumPy
import numpy as np

# Define the perceptron function
def perceptron(input_data, weights):
    # Calculate the weighted sum
    weighted_sum = np.dot(input_data, weights)
    z = 1/(1 + np.tanh(-weighted_sum))

    # Apply a simple step function as activation
    output = 1 if weighted_sum > 0 else 0

    return output

# Sample data
input_data = np.array([1, 2])
weights = np.array([0.5, -0.5])

# Test the perceptron
output = perceptron(input_data, weights)
print("Perceptron Output:", output)
```

```
Perceptron Output: 0
```

Task 2: Multi-feature Perceptron Extend the perceptron function to handle more than two features. Update the sample data accordingly and test the modified perceptron function with the new dataset. Visualize the decision boundary in a 3D plot or use color to represent the classification in a scatter plot.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the extended perceptron function with sigmoid activation
def perceptron_sigmoid(input_data, weights):
    # Calculate the weighted sum
    weighted_sum = np.dot(input_data, weights)
    # Apply sigmoid activation function
    output = 1 / (1 + np.exp(-weighted_sum))
    return output

# Generate random data for a binary classification task with three features
np.random.seed(42)
num_samples = 50
features_3d = np.random.rand(num_samples, 3) * 2 - 1  # Random values between -1 and 1
weights_3d = np.array([0.5, -0.5, 0.5])

# Classify data using the extended perceptron with sigmoid activation
predictions_3d = [perceptron_sigmoid(sample, weights_3d) for sample in features_3d]
```

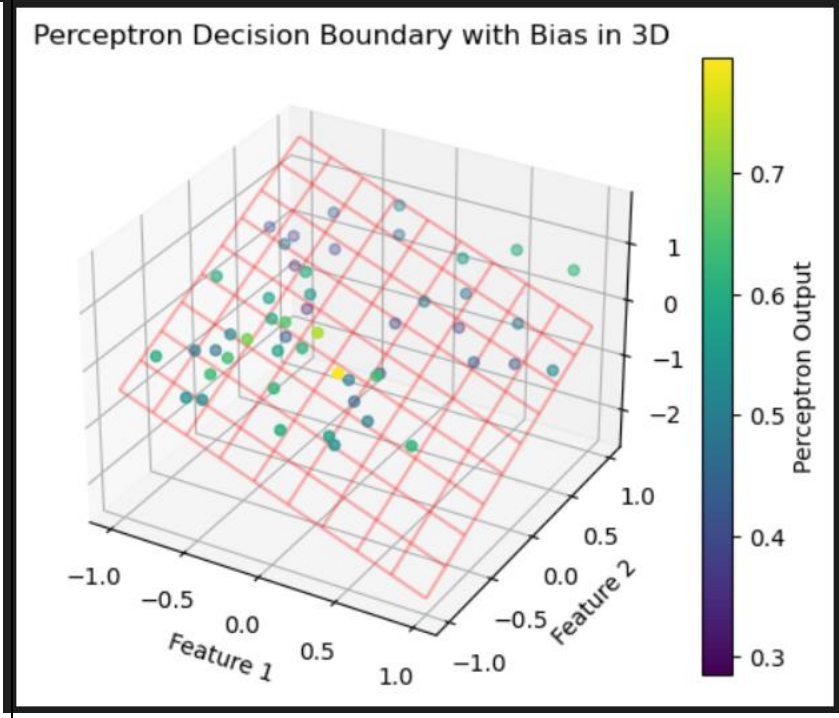
```python
# Visualize the data and decision boundary in a 3D scatter plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatter plot of the input data
scatter = ax.scatter(features_3d[:, 0], features_3d[:, 1], features_3d[:, 2], c=predictions_3d,
cmap='viridis')

# Set labels for each axis
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')

# Plot the decision boundary (a plane defined by the perceptron)
# Generate a meshgrid for the plane
xx, yy = np.meshgrid(np.linspace(-1, 1, 10), np.linspace(-1, 1, 10))
zz = (-weights_3d[0] * xx - weights_3d[1] * yy) / weights_3d[2]

# Plot the decision boundary as a wireframe
ax.plot_wireframe(xx, yy, zz, color='gray', alpha=0.3, label='Decision Boundary')

plt.title('Perceptron Decision Boundary in 3D')

# Add a colorbar
cbar = plt.colorbar(scatter)
cbar.set_label('Perceptron Output')

plt.show()
```



Task 3: Introduce Bias Modify the perceptron function to include a bias term. Update the sample data and weights accordingly. Discuss how the introduction of a bias term affects the decision boundary and the perceptron's ability to capture patterns in the data.

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the perceptron function with sigmoid activation and bias
def perceptron_sigmoid_bias(input_data, weights, bias):
    # Calculate the weighted sum
    weighted_sum = np.dot(input_data, weights) + bias
    # Apply sigmoid activation function
    output = 1 / (1 + np.exp(-weighted_sum))
    return output

# Generate random data for a binary classification task with three features
np.random.seed(42)
num_samples = 50
features_3d = np.random.rand(num_samples, 3) * 2 - 1  # Random values between -1 and 1
weights_3d = np.array([0.5, -0.5, 0.5])
bias = 0.2  # Introduce a bias term

# Classify data using the extended perceptron with sigmoid activation and bias
predictions_3d_bias = [perceptron_sigmoid_bias(sample, weights_3d, bias) for sample in features_3d]

# Visualize the data and decision boundary in a 3D scatter plot
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatter plot of the input data
scatter = ax.scatter(features_3d[:, 0], features_3d[:, 1], features_3d[:, 2],
c=predictions_3d_bias, cmap='viridis')

# Set labels for each axis
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')

# Plot the decision boundary (a plane defined by the perceptron)
# Generate a meshgrid for the plane
xx, yy = np.meshgrid(np.linspace(-1, 1, 10), np.linspace(-1, 1, 10))
zz = (-weights_3d[0] * xx - weights_3d[1] * yy - bias) / weights_3d[2]

# Plot the decision boundary as a wireframe
ax.plot_wireframe(xx, yy, zz, color='red', alpha=0.3, label='Decision Boundary')

plt.title('Perceptron Decision Boundary with Bias in 3D')

# Add a colorbar
cbar = plt.colorbar(scatter)
cbar.set_label('Perceptron Output')

plt.show()
```

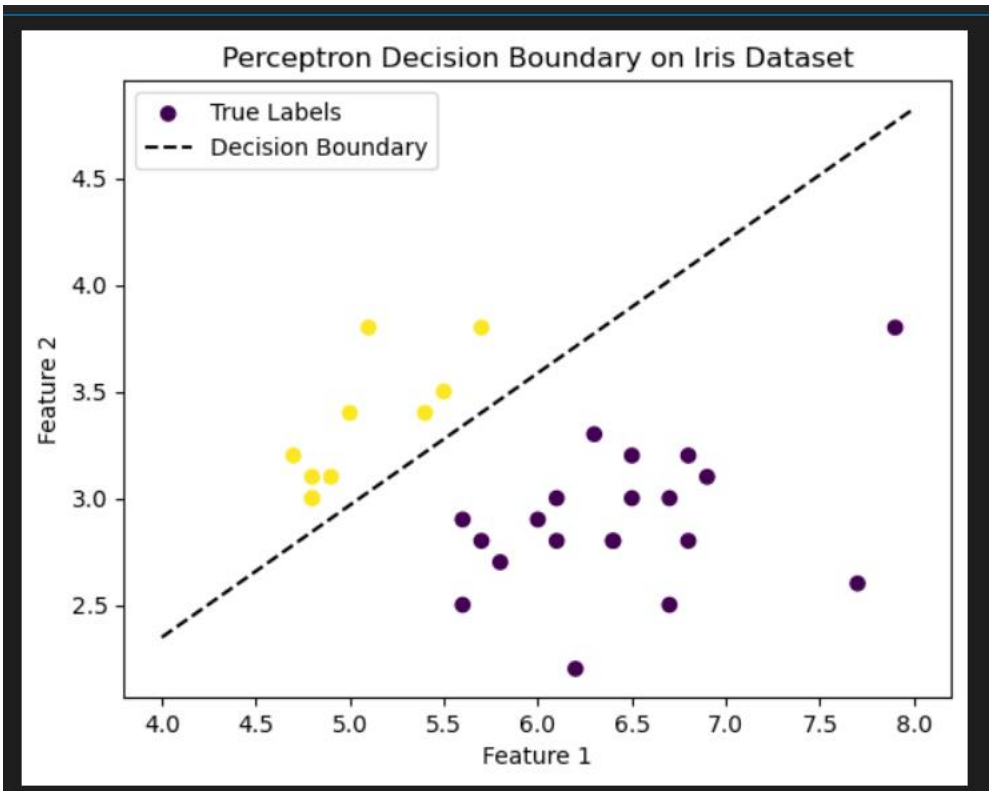


Lab Tasks

Task 4: Real-world Data Classification Use a real-world dataset (e.g., from sklearn.datasets) and adapt the perceptron code to perform binary classification on the dataset. Visualize the data points and the decision boundary. Discuss the challenges and limitations of using a simple perceptron for real-world datasets.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Take the first two features for simplicity
y = (iris.target == 0).astype(int)  # Binary classification (setosa or not)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the perceptron function with sigmoid activation and bias
def perceptron_sigmoid_bias(input_data, weights, bias):
    # Calculate the weighted sum
    weighted_sum = np.dot(input_data, weights) + bias
    # Apply sigmoid activation function
    output = 1 / (1 + np.exp(-weighted_sum))
    return output

# Train the perceptron on the training set
def train_perceptron(X_train, y_train, learning_rate=0.01, epochs=100):
    num_features = X_train.shape[1]
    weights = np.zeros(num_features)
    bias = 0

    for epoch in range(epochs):
        for i in range(len(X_train)):
            input_data = X_train[i]
            target = y_train[i]
            # Make a prediction
            prediction = perceptron_sigmoid_bias(input_data, weights, bias)
            # Update weights and bias using gradient descent
            weights += learning_rate * (target - prediction) * prediction * (1 - prediction) *
input_data
            bias += learning_rate * (target - prediction) * prediction * (1 - prediction)

    return weights, bias

# Train the perceptron
```

```
weights, bias = train_perceptron(X_train, y_train)

# Visualize the data points and decision boundary
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', label='True Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot the decision boundary
x_line = np.linspace(4, 8, 10)
y_line = (-weights[0] * x_line - bias) / weights[1]
plt.plot(x_line, y_line, color='black', linestyle='--', label='Decision Boundary')

plt.legend()
plt.title('Perceptron Decision Boundary on Iris Dataset')
plt.show()
```



USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 13
Neural Networks Cont

Objective:

The Model Architectures Lab aims to deepen participants' understanding of neural network components by exploring activation functions and experimenting with model architectures. Through TensorFlow/Keras implementation, participants will delve into the functionality of activation functions such as ReLU and Sigmoid, gaining insights into their impact on model behavior. The lab further encourages hands-on experimentation, allowing participants to modify the model architecture, explore diverse activation functions, and analyze the resulting effects on training performance and classification outcomes..

Name of Student: Sawera Fazal

Roll No: 21A-026-SE Sec.21A

Date of Experiment: 31 dec 23

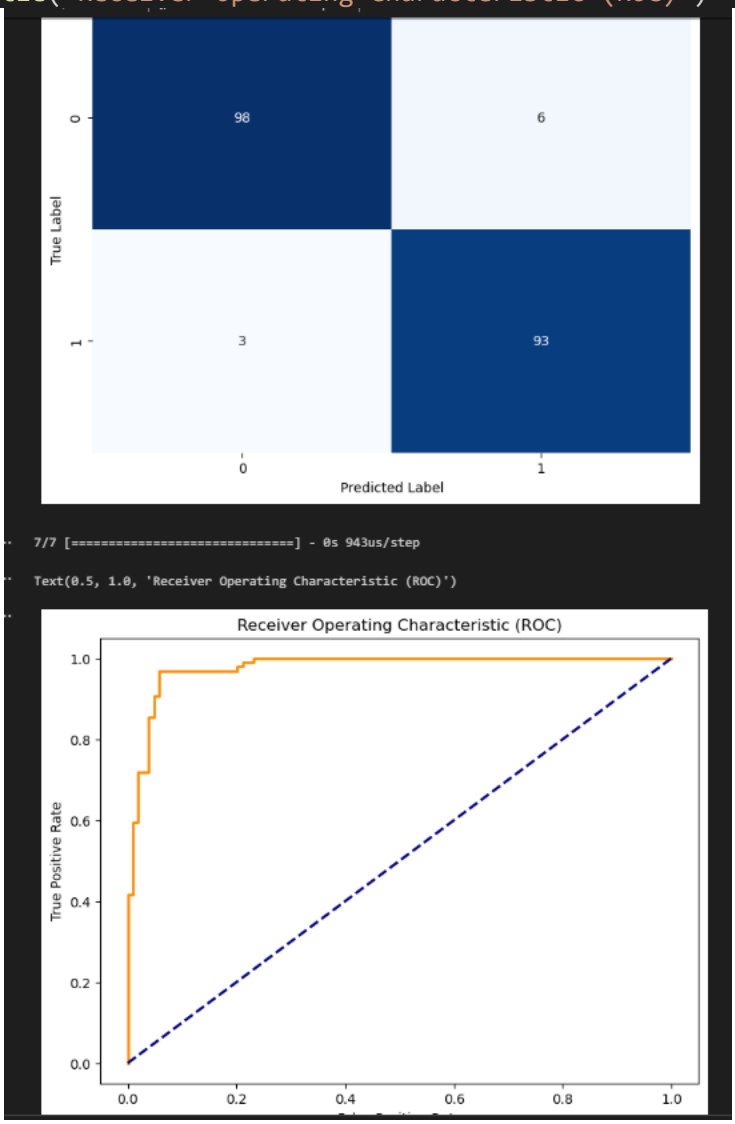
Marks Obtained/Remarks: _____

Lab Tasks

Task 1 Deeply study different activation functions also explain their functionality.Modify the previous model to use different activation functions.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc

# Create a synthetic dataset (assuming it's classification)
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_classes=2,
random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build a simple feedforward neural network using TensorFlow/Keras with Leaky ReLU and Sigmoid
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation=tf.keras.layers.LeakyReLU(alpha=0.01),
input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model with binary cross-entropy loss and Adam optimizer
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate the model on the test set
y_pred = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set: {accuracy * 100:.2f}%")

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```python
# Compute ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, model.predict(X_test))
roc_auc = auc(fpr, tpr)

# Visualize the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
```

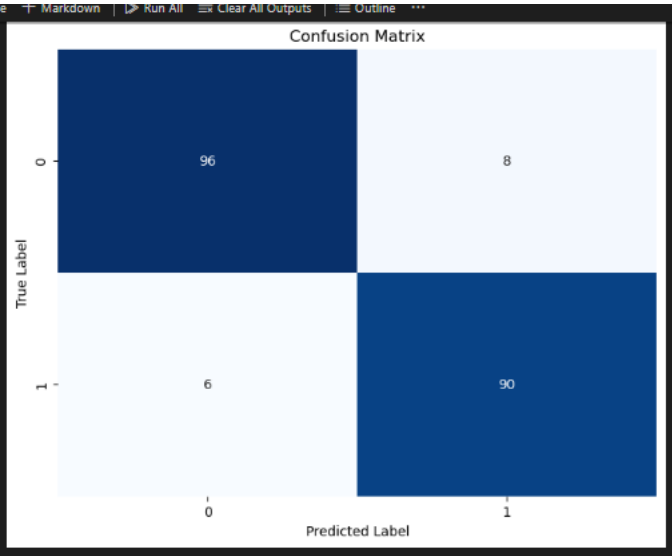


Task 2 Experiment with model architecture changes.

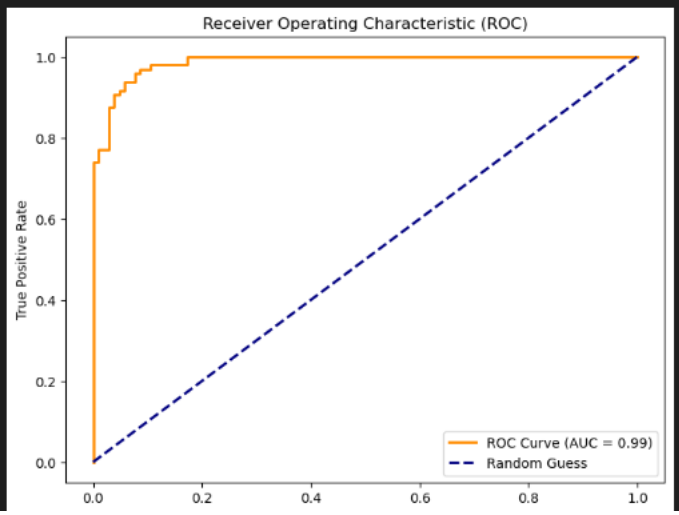
```python
# Experimenting with Model Architecture Changes
#1)  Adding more hidden layers

model = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation=tf.keras.layers.LeakyReLU(alpha=0.01),
input_shape=(X_train.shape[1],)),
```

```
    tf.keras.layers.Dropout(0.3),  # Adding dropout for regularization
    tf.keras.layers.Dense(128, activation=tf.keras.layers.LeakyReLU(alpha=0.01)),
    tf.keras.layers.Dropout(0.3),  # Adding dropout for regularization
    tf.keras.layers.Dense(64, activation=tf.keras.layers.LeakyReLU(alpha=0.01)),
    tf.keras.layers.Dropout(0.3),  # Adding dropout for regularization
    tf.keras.layers.Dense(32, activation=tf.keras.layers.LeakyReLU(alpha=0.01)),
    tf.keras.layers.Dropout(0.3),  # Adding dropout for regularization
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model with binary cross-entropy loss and Adam optimizer
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model on the training data
model.fit(X_train, y_train, epochs=15, batch_size=32, validation_split=0.1)

# Evaluate the model on the test set
y_pred = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set: {accuracy * 100:.2f}%")

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Compute ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, model.predict(X_test))
roc_auc = auc(fpr, tpr)

# Visualize the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend()
plt.show()
```

Code + Markdown | Run All Clear All Outputs | Outline ...




```
7/7 [==============================] - 0s 2ms/step
```



USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science
CS321 Artificial Intelligence

Lab# 14 and 15
Recommendation System and
Open Ended Lab

Objective:

This lab provides a comprehensive revision of basic programming concepts and their implementation in Python. The lab also revises some of the implementations specific to Python programming

Name of Student: Sawera Fazal

Roll No: 21A-026-SE Sec.21A

Date of Experiment: _____

Marks Obtained/Remarks: _____

Lab Task:

Design and implement a recommendation system using your preferred content type (e.g., songs, books) and your chosen dataset from Kaggle. Choose an appropriate recommendation algorithm, perform necessary data preprocessing, and evaluate the system's performance using relevant metrics. Present your findings, including any challenges encountered and insights gained during the process.

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import missingno as msno
%matplotlib inline

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

disney_titles = pd.read_csv(r"C:\Users\DELL\Desktop\AI\titles.csv")
titles=pd.DataFrame(disney_titles)
titles.head()
```

[18] Python

	id	title	type	description	release_year	age_certification	runtime	genres	production_countries	seasons	imdb_id	imdb_score	imdb_votes	tmdb_popularity	tmdb_score
0	tm89464	Miracle on 34th Street	MOVIE	Kris Kringle, seemingly the embodiment of Sant...	1947	G	96	['family', 'comedy', 'drama']	['US']	NaN	tt0039628	7.9	50969.0	23.515	7.388
1	tm61729	The Adventures of Ichabod and Mr. Toad	MOVIE	The Wind in the Willows: Concise version of Ke...	1950	G	68	['horror', 'fantasy', 'animation', 'family', '...	['US']	NaN	tt0041094	6.9	16502.0	16.194	6.500
2	tm61052	Cinderella	MOVIE	Cinderella has faith her dreams of a better li...	1950	G	74	['fantasy', 'animation', 'family', 'romance']	['US']	NaN	tt0042332	7.3	164292.0	93.547	7.035
3	tm87946	Dumbo	MOVIE	Dumbo is a baby elephant born with over-sized ...	1941	G	64	['animation', 'drama', 'family', 'fantasy']	['US']	NaN	tt0033563	7.2	135045.0	54.265	6.999
4	tm74391	Fantasia	MOVIE	Walt Disney's timeless masterpiece is an extra...	1941	G	119	['animation', 'family', 'fantasy', 'music']	['US']	NaN	tt0032455	7.7	98764.0	43.609	7.382

```
# Get the shape of the dataset rows,columns
titles.shape
```

[19] Python

```
(1854, 15)
```

```
# Get summary statistics of the DataFrame
titles.describe()
```

[20]

	release_year	runtime	seasons	imdb_score	imdb_votes	tmdb_popularity	tmdb_score
count	1854.000000	1854.000000	540.000000	1339.000000	1.328000e+03	1839.000000	1708.000000
mean	2006.193635	59.206580	2.618519	6.620687	6.961624e+04	25.745671	6.817494
std	20.629561	38.566305	3.268281	1.066208	1.846213e+05	79.389568	1.165990
min	1928.000000	1.000000	1.000000	1.600000	5.000000e+00	0.600000	2.000000
25%	2002.000000	24.000000	1.000000	5.950000	3.267500e+02	2.998500	6.217500
50%	2014.000000	48.000000	2.000000	6.700000	3.228000e+03	9.148000	6.873000
75%	2020.000000	91.000000	3.000000	7.400000	3.273050e+04	23.152500	7.500000
max	2023.000000	182.000000	36.000000	9.500000	1.403757e+06	2159.377000	10.000000

```
titles.dtypes
```

[21]

```
id                     object
title                  object
type                   object
description            object
release_year            int64
age_certification      object
runtime                 int64
genres                 object
production_countries   object
seasons               float64
imdb_id                object
imdb_score            float64
imdb_votes            float64
tmdb_popularity       float64
tmdb_score            float64
dtype: object
```

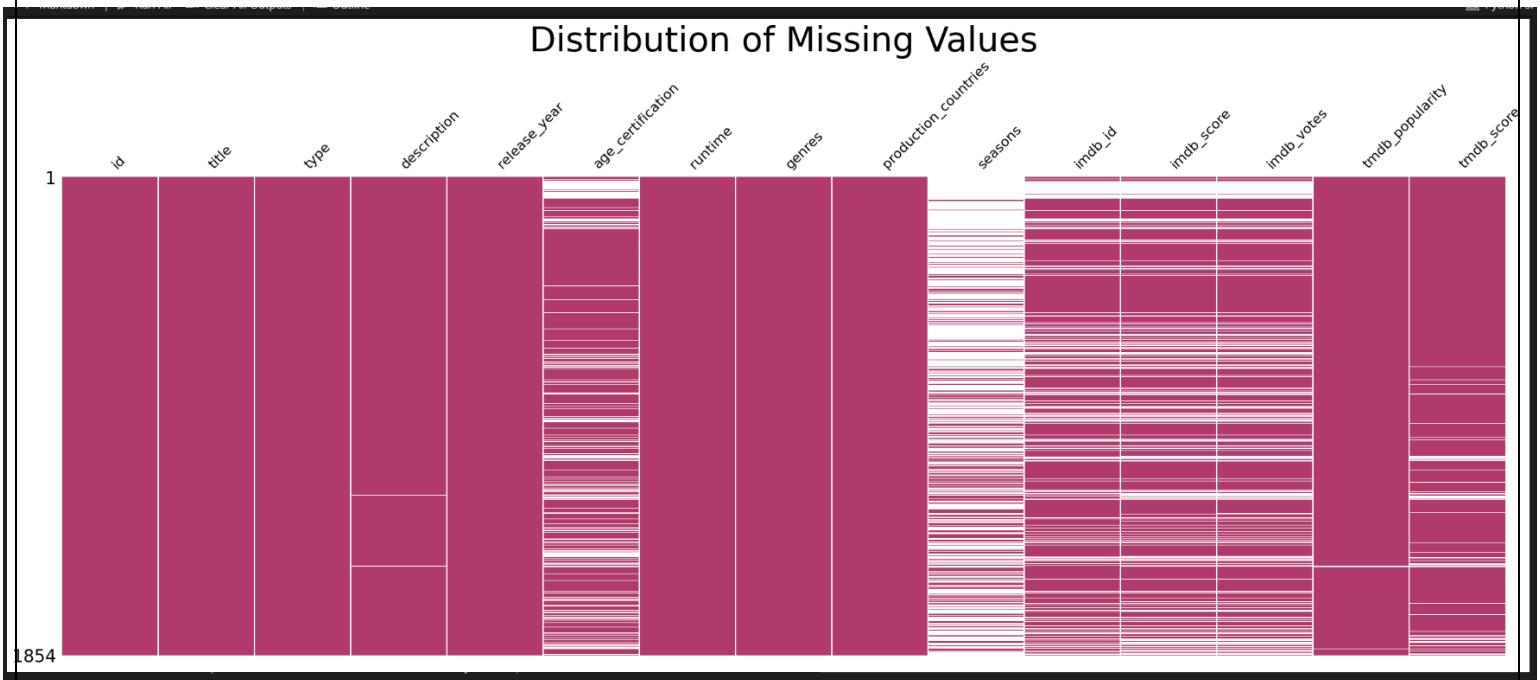
```
null = titles.isnull().sum()
print(null.count)
```

]

```
<bound method Series.count of id                      0
title                   0
type                    0
description             9
release_year            0
age_certification     451
runtime                 0
genres                  0
production_countries    0
seasons              1314
imdb_id               478
imdb_score            515
imdb_votes            526
tmdb_popularity        15
tmdb_score            146
dtype: int64>
```

```
msno.matrix(titles, sparkline=False, figsize=(30,10), color=(0.69,0.23,0.42))
plt.title('Distribution of Missing Values', fontsize=40)
```

]




```
    #get theh dtypes non-null also check for missing values
    titles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1854 entries, 0 to 1853
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    1854 non-null   object 
 1   title                 1854 non-null   object 
 2   type                  1854 non-null   object 
 3   description           1845 non-null   object 
 4   release_year          1854 non-null   int64  
 5   age_certification     1403 non-null   object 
 6   runtime               1854 non-null   int64  
 7   genres                1854 non-null   object 
 8   production_countries  1854 non-null   object 
 9   seasons               540 non-null    float64
 10  imdb_id               1376 non-null   object 
 11  imdb_score            1339 non-null   float64
 12  imdb_votes            1328 non-null   float64
 13  tmdb_popularity       1839 non-null   float64
 14  tmdb_score            1708 non-null   float64
dtypes: float64(5), int64(2), object(8)
memory usage: 217.4+ KB
```

```
# For genres
titles['genres'] = titles['genres'].str.replace(r'[','').str.replace(r"'",'').str.replace(r']','')
titles['genre'] = titles['genres'].str.split(',').str[0]

# For countries
titles['production_countries'] = titles['production_countries'].str.replace(r"[", '').str.replace(r"'", '').str.replace(r"]", '')
titles['production_country'] = titles['production_countries'].str.split(',').str[0]
```

```
titles.drop(['genres', 'production_countries'], axis=1, inplace=True)
```

```
titles['genre'].unique()
```

```
array(['family', 'horror', 'fantasy', 'animation', 'comedy', 'thriller',
       'romance', 'action', 'documentation', 'crime', 'western', '',
       'drama', 'scifi', 'history', 'music', 'reality', 'sport', 'war'],
      dtype=object)
```

```
titles['production_country'].unique()
```

```
array(['US', 'GB', 'CA', '', 'FR', 'SE', 'AU', 'SK', 'ES', 'JP', 'PL',
       'IN', 'CN', 'BR', 'DE', 'NZ', 'AR', 'CI', 'NL', 'AE', 'KR', 'IT',
       'CO', 'ZA', 'CL', 'AT', 'PH', 'BW', 'GR', 'TW', 'MX', 'DK', 'TR',
       'PR'], dtype=object)
```

```
titles['genre'] = titles['genre'].replace('', np.nan)
titles['production_country'] = titles['production_country'].replace('',np.nan)
```

```
#Handling the 'seasons' column
len(titles.loc[(titles['seasons'].isna()) & (titles['type'] == 'MOVIE')]) == titles.seasons.isna().sum()
```

```
True
```

+ Code +

```
titles['seasons'].fillna(0, inplace=True)
```

```
#Handling the rest of the null values
titles.isna().sum()
```

```
id                    0
title                 0
type                  0
description           9
release_year          0
age_certification     451
runtime               0
seasons               0
imdb_id               478
imdb_score            515
imdb_votes            526
tmdb_popularity       15
tmdb_score            146
genre                 39
production_country    52
dtype: int64
```

```
titles.drop(['imdb_id','age_certification'], axis=1,inplace=True)
```

```
#Let's also get rid of some NaN values that are still left in our dataset.

titles.dropna(inplace=True)
titles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1251 entries, 0 to 1852
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
```

+ Code + Markdown | Run All Clear All Outputs | Outline ...

```
#description based Recommender
titles['description'].head()
```

[17]

```
0    Kris Kringle, seemingly the embodiment of Sant...
1    The Wind in the Willows: Concise version of Ke...
2    Cinderella has faith her dreams of a better li...
3    Dumbo is a baby elephant born with over-sized ...
4    Walt Disney's timeless masterpiece is an extra...
Name: description, dtype: object
```

```
#Adding the Streaming Platform for the titles
lt = []
for i in titles['id']:
    movie_streaming = []

    if i in disney_titles['id'].values:
        movie_streaming.append('disney+')
    lt.append(movie_streaming)
titles['streaming_platform'] = lt
titles.head()
```

[42]

	id	title	type	description	release_year	runtime	seasons	imdb_score	imdb_votes	tmdb_popularity	tmdb_s:o
0	tm89464	Miracle on 34th Street	MOVIE	Kris Kringle, seemingly the embodiment of Sant...	1947	96	0.0	7.9	50969.0	23.515	7.3
1	tm61729	The Adventures of Ichabod and Mr. Toad	MOVIE	The Wind in the Willows: Concise version of Ke...	1950	68	0.0	6.9	16502.0	16.194	6.5
2	tm61052	Cinderella	MOVIE	Cinderella has faith her dreams of a better li...	1950	74	0.0	7.3	164292.0	93.547	7.0
3	tm87946	Dumbo	MOVIE	Dumbo is a baby elephant born with over-sized ...	1941	64	0.0	7.2	135045.0	54.265	6.9
4	tm74391	Fantasia	MOVIE	Walt Disney's timeless masterpiece is an extra...	1941	119	0.0	7.7	98764.0	43.609	7.3

```
#Separating the data in Movies and TV Shows
movies = titles[titles['type'] == 'MOVIE'].copy().reset_index()
movies.drop(['index'], axis=1, inplace=True)

shows = titles[titles['type'] == 'SHOW'].copy().reset_index()
shows.drop(['index'], axis=1, inplace=True)
movies.head()
```

	id	title	type	description	release_year	runtime	seasons	imdb_score	imdb_votes	tmdb_popularity	tmdb_score	genre	production_country
0	tm89464	Miracle on 34th Street	MOVIE	Kris Kringle, seemingly the embodiment of Sant...	1947	96	0.0	7.9	50969.0	23.515	7.388	family	US
1	tm61729	The Adventures of Ichabod and Mr. Toad	MOVIE	The Wind in the Willows: Concise version of Ke...	1950	68	0.0	6.9	16502.0	16.194	6.500	horror	US
2	tm61052	Cinderella	MOVIE	Cinderella has faith her dreams of a better li...	1950	74	0.0	7.3	164292.0	93.547	7.035	fantasy	US
3	tm87946	Dumbo	MOVIE	Dumbo is a baby elephant born with over-sized ...	1941	64	0.0	7.2	135045.0	54.265	6.999	animation	US
4	tm74391	Fantasia	MOVIE	Walt Disney's timeless masterpiece is an extra...	1941	119	0.0	7.7	98764.0	43.609	7.382	animation	US

```
shows.head()
```

	id	title	type	description	release_year	runtime	seasons	imdb_score	imdb_votes	tmdb_popularity	tmdb_score	genre	production_country	streaming_plat
0	ts30718	Schoolhouse Rock!	SHOW	Schoolhouse Rock! is an American interstitial ...	1973	3	7.0	8.2	4108.0	3.125	7.800	animation	US	[disr
1	ts22470	The Muppet Show	SHOW	Go behind the curtains as Kermit the Frog and ...	1976	25	5.0	8.4	25122.0	17.728	8.014	comedy	GB	[disr
2	ts20949	Zorro	SHOW	Diego de la Vega, the son of a wealthy landown...	1957	30	3.0	8.2	3799.0	59.219	7.720	action	US	[disr
3	ts24939	Spider-Woman	SHOW	When Jessica Drew was bitten by a poisonous sp...	1979	21	1.0	5.8	873.0	17.427	7.900	animation	US	[disr
4	ts27944	Spin and Marty	SHOW	Spin and Marty is a popular series of televisi...	1955	12	1.0	8.1	323.0	2.725	6.000	western	US	[disr

+ Code + Markdown

```
#Define a TF-IDF Vectorizer Object.
#This remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix_movies = tfidf.fit_transform(movies['description'])
tfidf_matrix_shows = tfidf.fit_transform(shows['description'])

#Output the shape of tfidf_matrix
print(f'Shape for Movies: {tfidf_matrix_movies.shape}')
print(f'Shape for Shows: {tfidf_matrix_shows.shape}')
```

```
Shape for Movies: (816, 7455)
Shape for Shows: (435, 4987)
```

```
#cosine similarity to calculate a numeric quantity that denotes the similarity between two movies/shows.

# Compute the cosine similarity matrix
cosine_sim_movies = linear_kernel(tfidf_matrix_movies, tfidf_matrix_movies)
cosine_sim_shows = linear_kernel(tfidf_matrix_shows, tfidf_matrix_shows)


#create a way to identify the index of a movie/show in our data, given its title.

indices_movies = pd.Series(movies.index, index=movies['title'])
indices_shows = pd.Series(shows.index, index=shows['title'])
```

```
def get_title(title,indices):
    """
    Function that gets the 'index searcher' and searches
    the user's title index.
    """

    try:
        index = indices[title]
    except:
```

```
        print("\n  Title not found")
        return None

    if isinstance(index, np.int64):
        return index

    else:
        rt = 0
        print("Select a title: ")
        for i in range(len(index)):
            print(f"{i} - {movies['title'].iloc[index[i]]}", end=' ')
            print(f"({movies['release_year'].iloc[index[i]]})")
        rt = int(input())
        return index[rt]
# define functions that accept a movie/show title as input and produce a list of the 10 most
similar titles.

def get_recommendations_movie(title, cosine_sim=cosine_sim_movies):
    """
    A function that takes a movie title as input and prints on the screen
    the 10 most similar movies based on the input description.
    """

    title = get_title(title, indices_movies)
    if title == None:
        return

    idx = indices_movies[title]

    print(f"Title: {movies['title'].iloc[idx]} |  Year: {movies['release_year'].iloc[idx]}")

    print('**' * 40)

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    print(movies[['title', 'release_year','streaming_platform']].iloc[movie_indices])

    print('**' * 40)
def get_recommendations_show(title, cosine_sim=cosine_sim_shows):
    # """
    # A function that takes a show title as input and prints on the screen
```

```
    # the 10 most similar shows based on the input description.
    # """

    title = get_title(title, indices_shows)
    if title == None:
        return

    idx = indices_shows[title]

    print(f"Title: {shows['title'].iloc[idx]} | Year: {shows['release_year'].iloc[idx]}")

    print('**' * 40)

    # Get the pairwsie similarity scores of all shows with that show
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the shows based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar shows
    sim_scores = sim_scores[1:11]

    # Get the show indices
    show_indices = [i[0] for i in sim_scores]

    print(shows[['title', 'release_year', 'streaming_platform']].iloc[show_indices])

    print('**' * 40)
```

```
get_recommendations_movie('Rocky')
```

```
Title not found
```

```
get_recommendations_show('Spider-Woman')
```

```
Title: Spider-Woman | Year: 1979
********************************************************************************
                                title  release_year streaming_platform
21                          Spider-Man          1994         [disney+]
231                 Marvel's Spider-Man          2017         [disney+]
12    Spider-Man and His Amazing Friends          1981         [disney+]
15                          Spider-Man          1981         [disney+]
222              Marvel's The Defenders          2017         [disney+]
115        Marvel's Ultimate Spider-Man          2012         [disney+]
171              Marvel's Jessica Jones          2015         [disney+]
253           Marvel Rising: Initiation          2018         [disney+]
62   Spider-Man: The New Animated Series          2003         [disney+]
255        Marvel Super Hero Adventures          2017         [disney+]
********************************************************************************
```