

**AI PRACTICAL**

**Student Details :-**

* Name :- Ankit Senjaliya
* Enrollment No. :- 19BT04046
* Branch :- Computer Science And Engineering
* Semester :- 4

**Faculty Mentor Details :-**

* Dr. Saurabh Shah

**PRACTICAL = 1**

* Aim :-
* Write a program to implement the Water Jug Problem
* Theory :-
* Water Jug Problem is also known as Water Pouring Puzzles, measuring puzzles and decanting problems. These belong to a class of puzzles, in which there are a finite and specific number of water jugs having predefined integral capacities, in terms of gallons or litres.
* Code :-

print ("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

x = 0

y = 0

m = 4

n = 3

print ("\nInitial State = (0, 0)")

print ("Capacities = (4, 3)")

print ("Goal State = (2, y)")

while x! = 2:

a = int (input ("\nEnter Rule = "))

if (a == 1):

x = m

elif (a == 2):

y = n

elif (a == 3):

x = 0

elif (a == 4):

y = 0

elif (a == 5):

t = n - y

y = n

x -= t

elif (a == 6):

t = m - x

x = m

y -= t

elif (a == 7):

y += x

x = 0

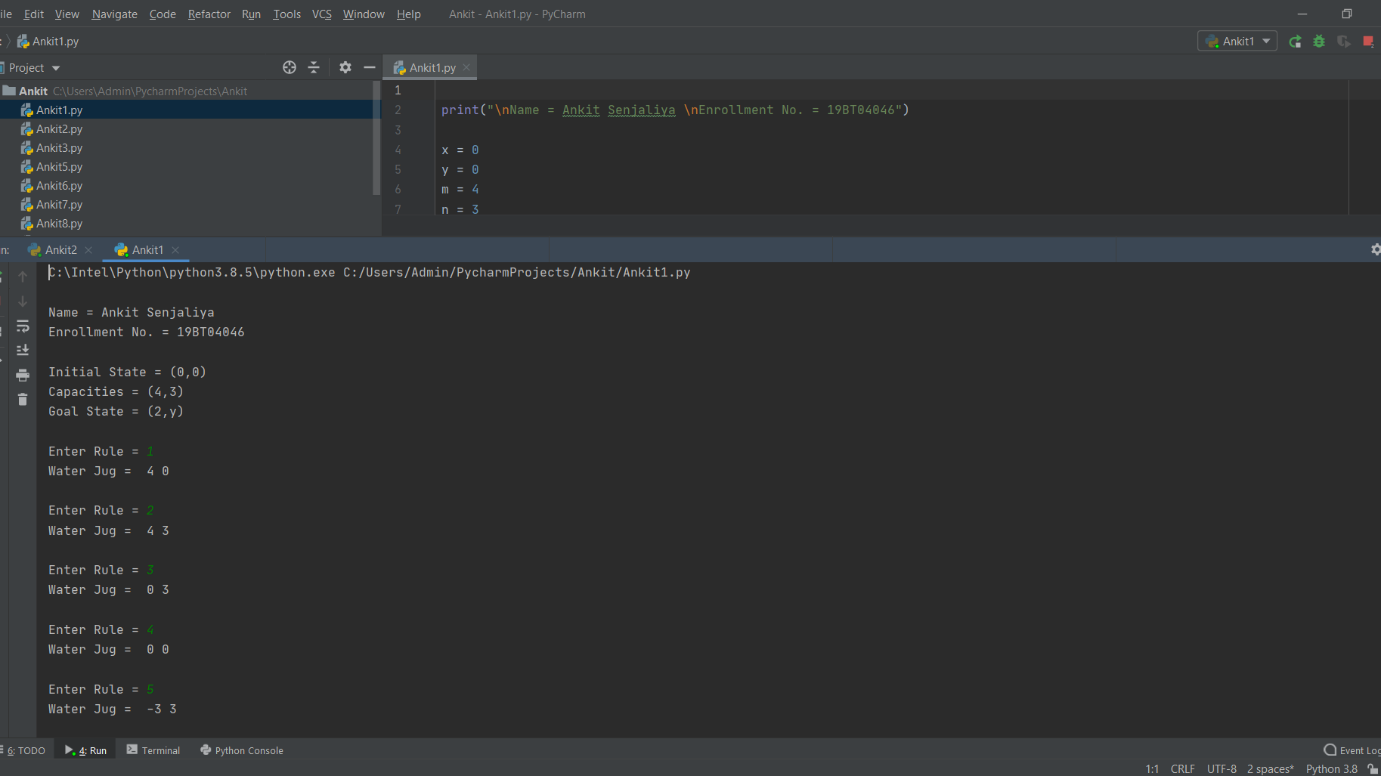
elif (a == 8):

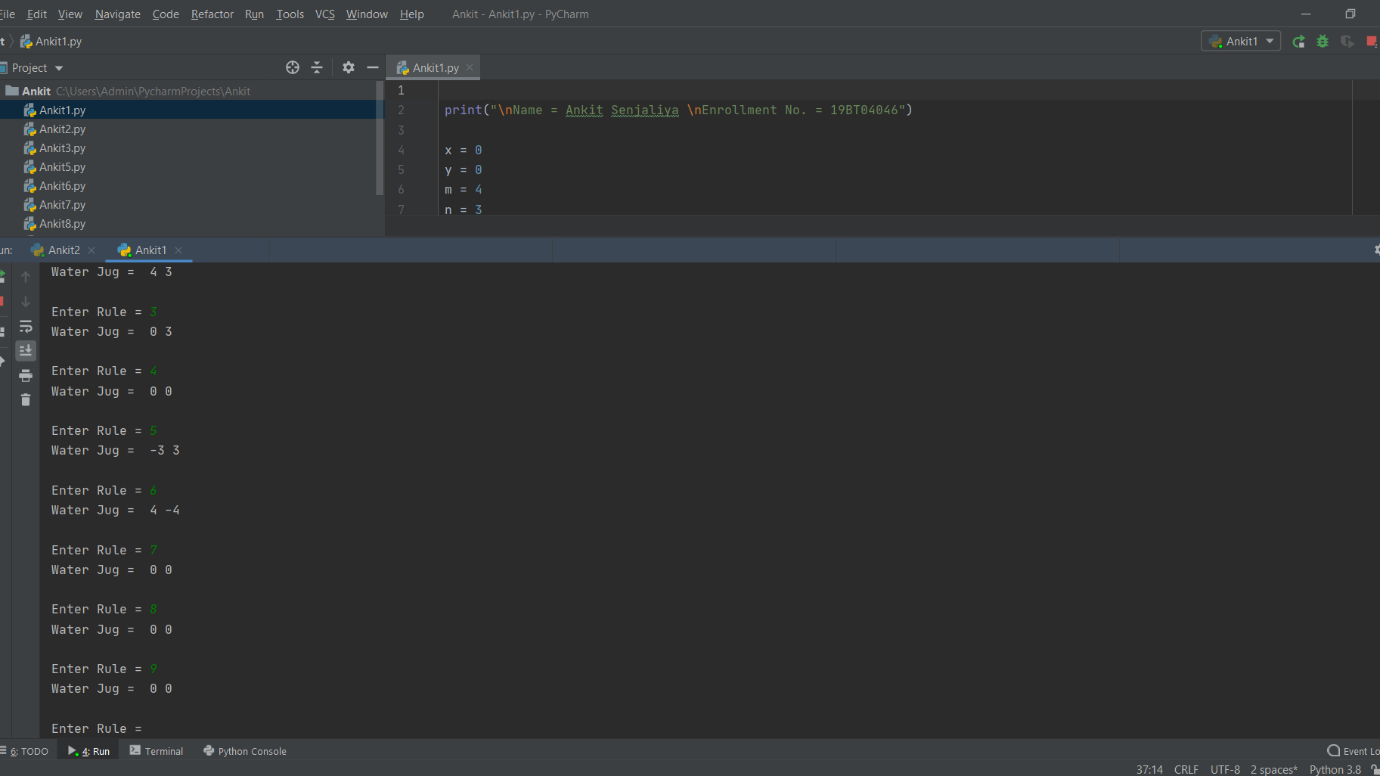
x += y

y = 0

print ("Water Jug = ", x, y)

* Output :-





**PRACTICAL = 2**

* Aim :-
* Write a program to implement Missionaries and Cannibals puzzle for state space search
* Theory :-
* The missionaries and cannibals problem, and the closely related jealous husbands problem, are classic River-Crossing logic puzzles. The missionaries and cannibals problem is a well-known Toy problem in Artificial Intelligence, where it was used by Saul Amarel as an example of problem representation.
* Code :-

print("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

right\_bank = ["M","M","M","C","C","C"]

left\_bank = []

boat = "right"

def input\_details():

m = int (input ("\nEnter Number Of Missionary = "))

c = int (input ("Enter Number Of Cannibal = "))

side = input ("Enter Side You Wanna Go = ")

if side.lower() == "right" or side.lower() == "left":

move(m, c, side.lower())

else:

print("Enter A Valid Side = ")

def move(m, c, side):

global boat

global right\_bank

global left\_bank

if m + c > 2:

print("Enter Appropriate Values = ")

elif m + c <= 0:

print("Enter Appropriate Values")

elif boat == side:

print("We Are Already On The %s Side = " % boat)

else:

if m > 0 and side != boat:

if side == "right":

if left\_bank.count("M") == 0:

print("No Missionary On The Left Bank")

elif m > 0 and left\_bank.count("C") == 0:

pass

else:

if m == 2 or m > c:

for i in range(0, m):

right\_bank.insert(0, "M")

left\_bank.remove("M")

boat = "right"

elif m == 1 and c == 1:

for i in range(0, m):

right\_bank.insert(0, "M")

left\_bank.remove("M")

if side == "left":

if right\_bank.count("M") == 0:

print("No Missionary On The Right Bank")

elif m > 0 and right\_bank.count("C") == 0:

pass

else:

if m == 2 or m > c:

for i in range(0, m):

left\_bank.insert(0, "M")

right\_bank.remove("M")

boat = "left"

elif m == 1 and c == 1:

for i in range(0, m):

left\_bank.insert(0, "M")

right\_bank.remove("M")

if c > 0 and side != boat:

if side == "right":

if left\_bank.count("C") == 0:

print("No Cannibal On The Left Bank")

elif m > 0 and left\_bank.count("M") == 0:

pass

else:

for i in range(0, c):

right\_bank.append("C")

left\_bank.remove("C")

boat = "right"

if side == "left":

if right\_bank.count("C") == 0:

print("No Cannibal On The Right Bank")

elif m > 0 and right\_bank.count("M") == 0:

pass

else:

for i in range(0, c):

left\_bank.append("C")

right\_bank.remove("C")

boat = "left"

if (left\_bank.count("M") < left\_bank.count("C") and left\_bank.count("M") > 0) or (

right\_bank.count("M") < right\_bank.count("C") and right\_bank.count("M") > 0):

print("GAME OVER !!!")

right\_bank = []

return start()

else:

print(right\_bank, left\_bank, "We Are Currently On %s Side" % boat)

def start():

global boat

global right\_bank

global left\_bank

choice = input("Enter Y/y To Play Again Or Press Any Other Key To Exit = ")

if choice.lower() == 'y':

right\_bank = ["M", "M", "M", "C", "C", "C"]

left\_bank = []

boat = "right"

while right\_bank != []:

input\_details()

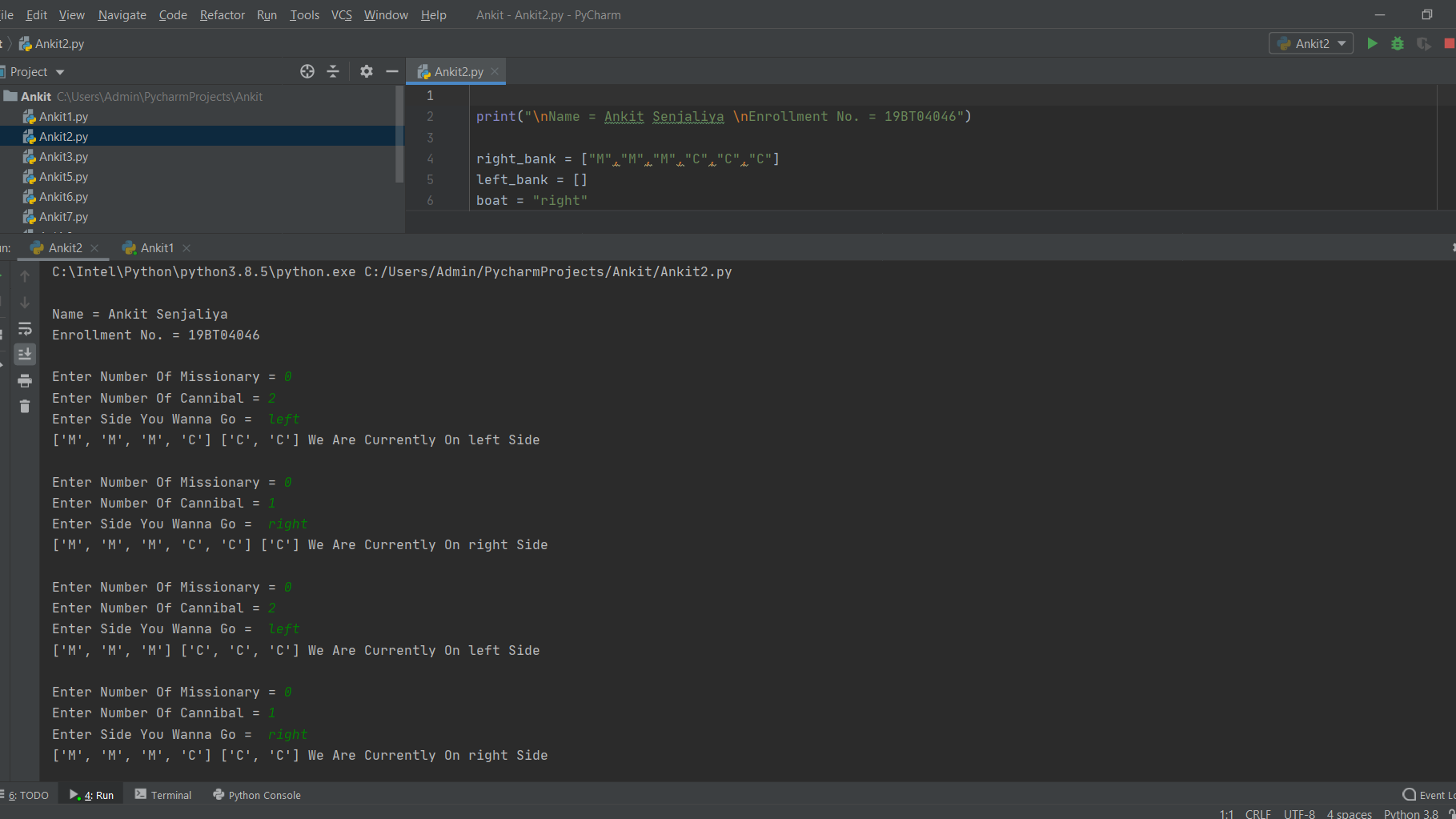
else:

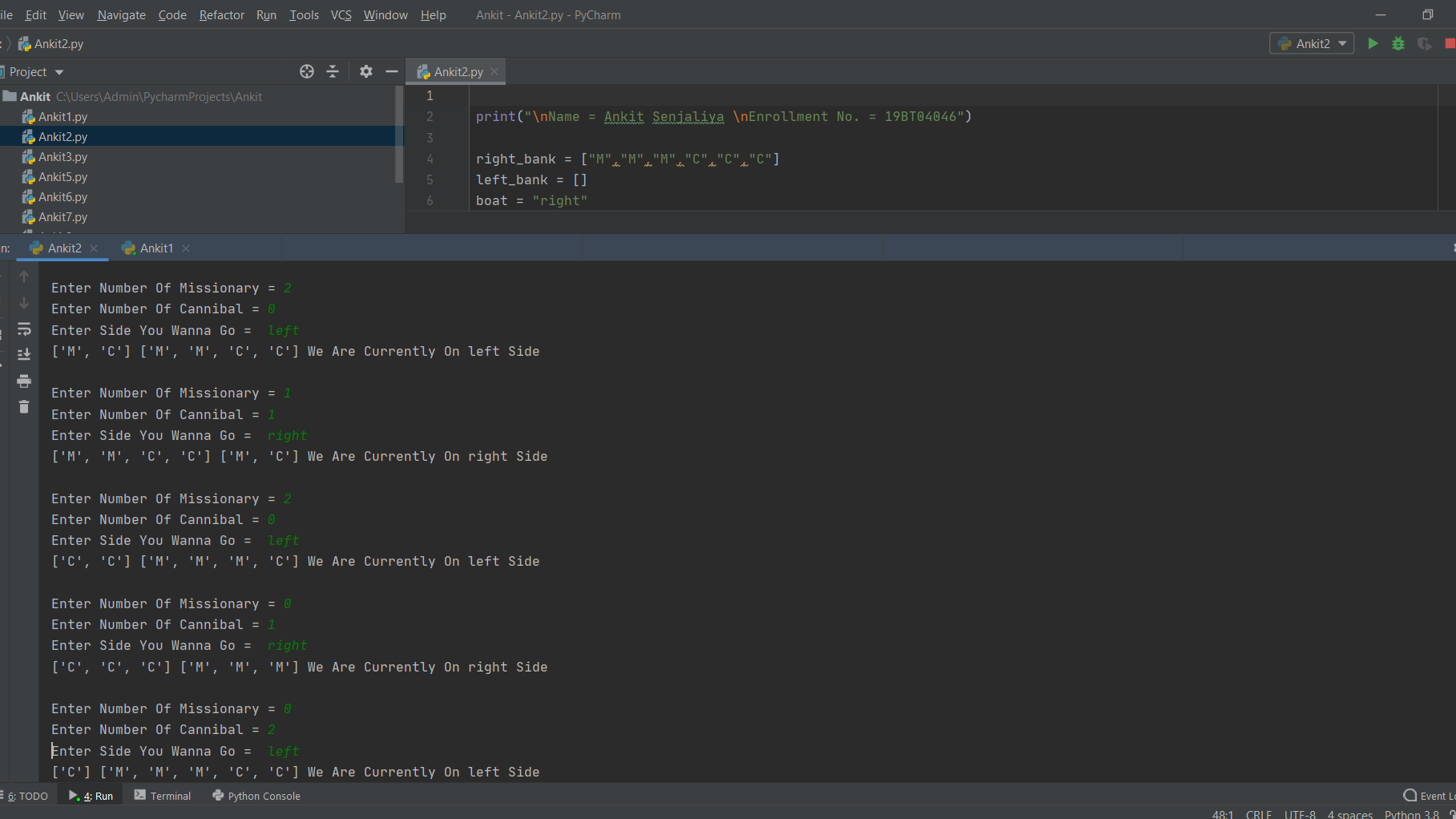
print("Thank You For Playing")

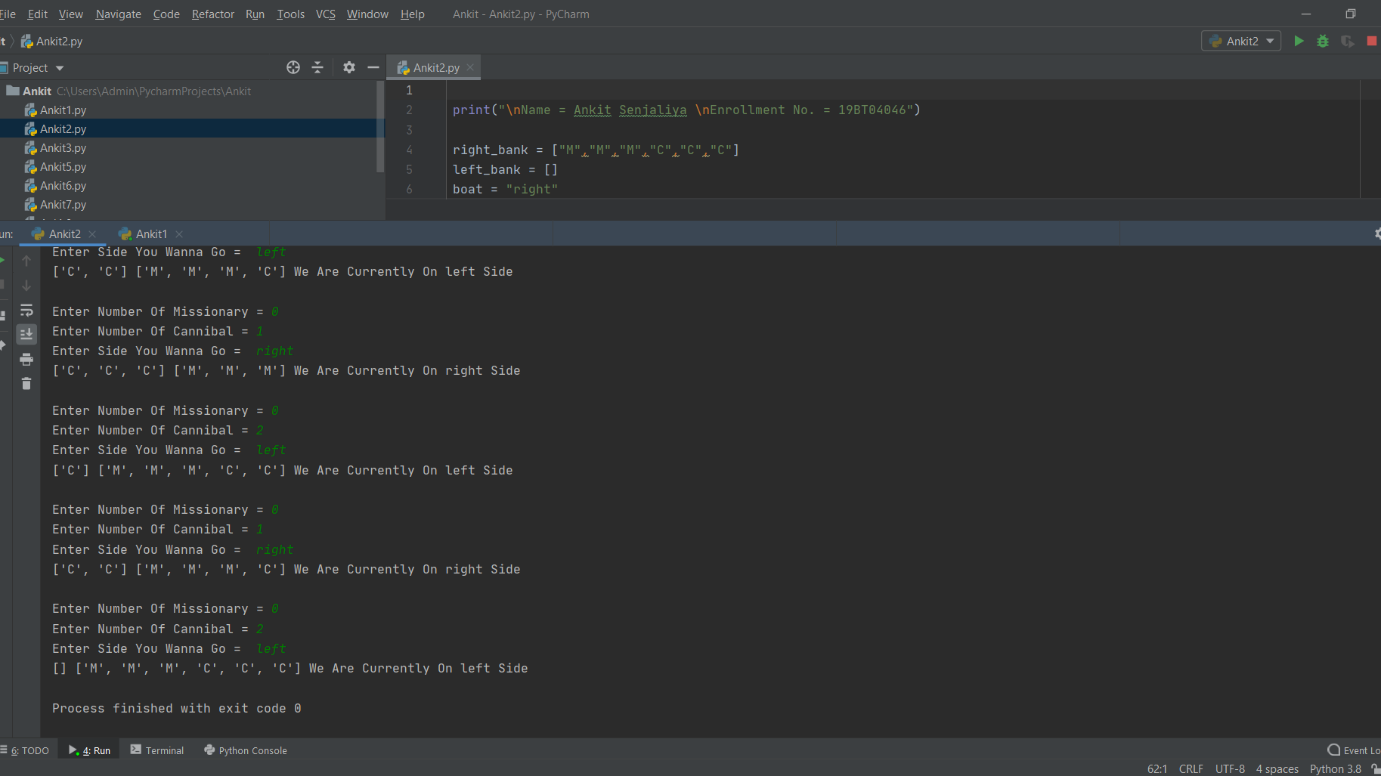
while right\_bank != []:

input\_details()

* Output :-







**PRACTICAL = 3**

* Aim :-
* Solve the 8-Puzzle problem using DFS & BFS
* Theory :-
* 8-Puzzle is an interesting game which requires a player to move blocks one at a time to solve a picture or a particular pattern. In this article I will be showing you how to write an intelligent program that could solve 8-Puzzle automatically using the A\* algorithm using Python and PyGame. Instead of a picture, we will use a pattern of numbers as shown in the figure, that is the final state.
* Code :-

print("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

class Solution:

def solve(self, board):

dict = {}

flatten = []

for i in range(len(board)):

flatten += board[i]

flatten = tuple(flatten)

dict[flatten] = 0

if flatten == (0, 1, 2, 3, 4, 5, 6, 7, 8):

return 0

return self.get\_paths(dict)

def get\_paths(self, dict):

count = 0

while True:

current\_nodes = [x for x in dict if dict[x] == count]

if len(current\_nodes) == 0:

return -1

for node in current\_nodes:

next\_moves = self.find\_next(node)

for move in next\_moves:

if move not in dict:

dict[move] = count + 1

if move == (0, 1, 2, 3, 4, 5, 6, 7, 8):

return count + 1

count += 1

def find\_next(self, node):

moves = {

0: [1, 3],

1: [0, 2, 4],

2: [1, 5],

3: [0, 4, 6],

4: [1, 3, 5, 7],

5: [2, 4, 8],

6: [3, 7],

7: [4, 6, 8],

8: [5, 7],

}

results = []

pos\_0 = node.index(0)

for move in moves[pos\_0]:

new\_node = list(node)

new\_node[move], new\_node[pos\_0] = new\_node[pos\_0], new\_node[move]

results.append(tuple(new\_node))

return results

ob = Solution()

Matrix = [

[3, 1, 2],

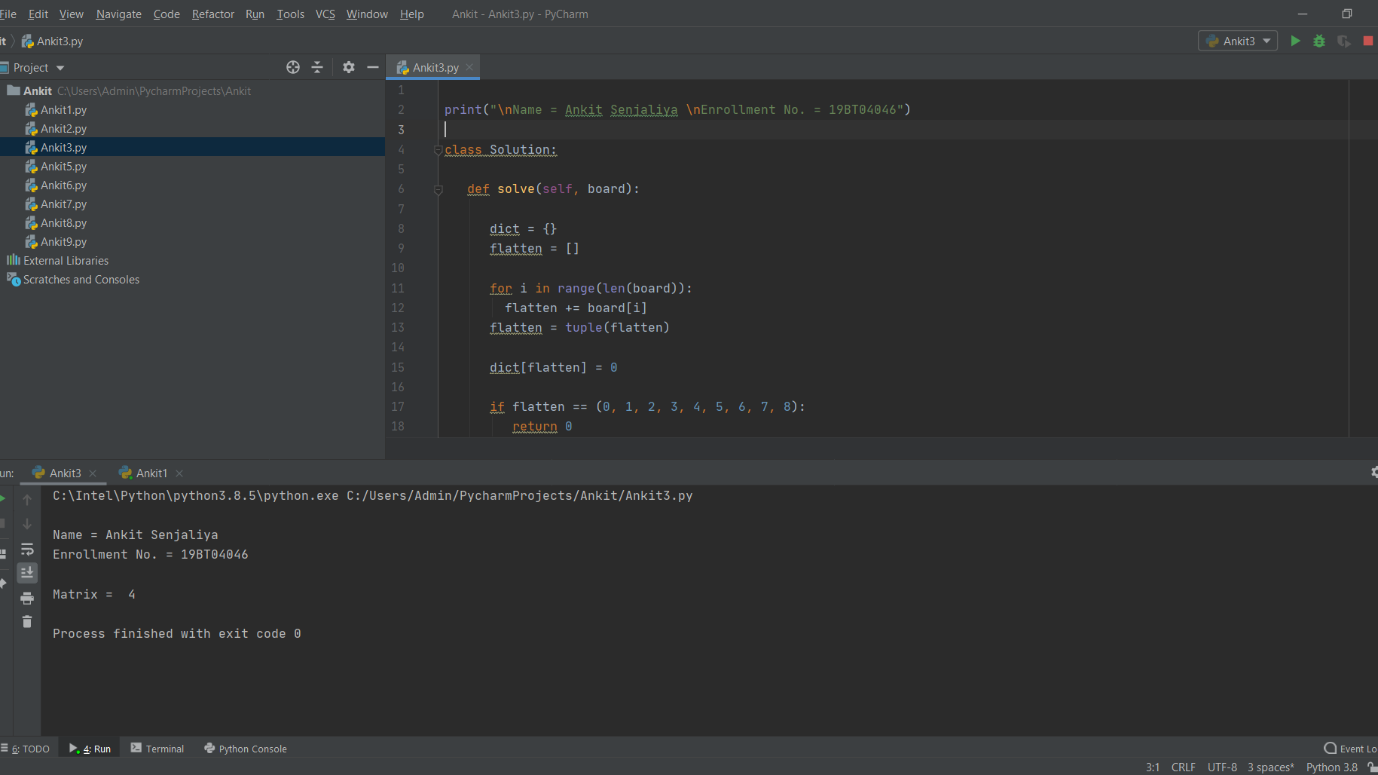
[4, 7, 5],

[6, 8, 0]

]

print("\nMatrix = ", ob.solve(Matrix))

* Output :-



**PRACTICAL = 5**

* Aim :-
* Study and Implement the Naive Bayes learner
* Theory :-
* This article discusses the theory behind the Naive Bayes classifiers and their implementation.
* Naive Bayes classifiers are a collection of classification algorithms based on **Bayes’ Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.
* Code :-

print("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

wine = datasets.load\_wine()

wine.keys()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(wine.data, wine.target, test\_size = 0.20,random\_state = 109)

model = GaussianNB()

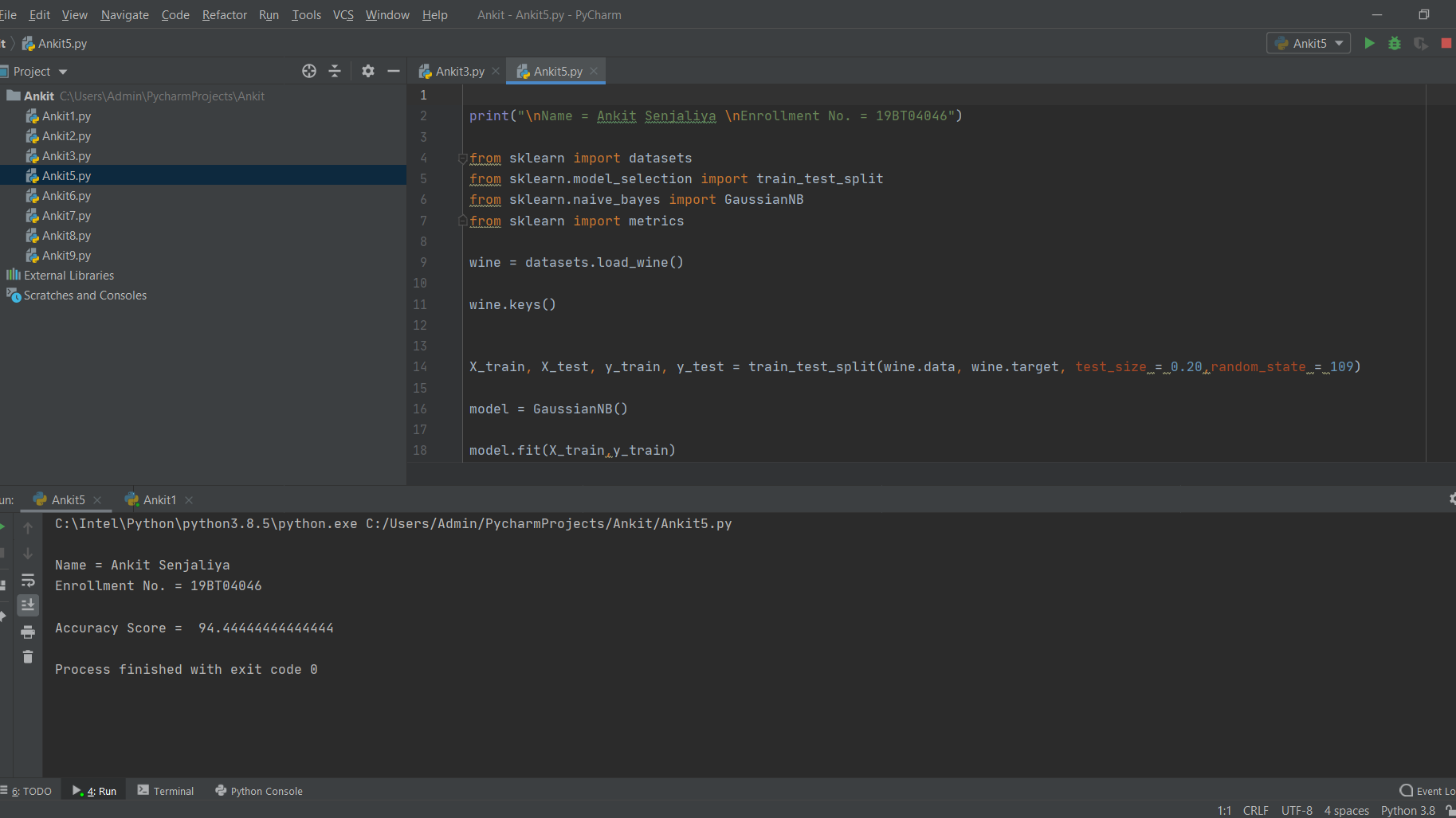
model.fit(X\_train,y\_train)

y\_pred=model.predict(X\_test)

accuracy\_score = metrics.accuracy\_score(y\_test,y\_pred)\*100

print("\nAccuracy Score = ", accuracy\_score)

* Output :-



**PRACTICAL = 6**

* Aim :-
* Study and Implement k-nearest neighbor classification
* Theory :-
* K Nearest Neighbor (KNN) is a very simple, easy to understand, versatile and one of the topmost machine learning algorithms. KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. In Credit ratings, financial institutes will predict the credit rating of customers. In loan disbursement, banking institutes will predict whether the loan is safe or risky. In political science, classifying potential voters in two classes will vote or won’t vote. KNN algorithm used for both classification and regression problems. KNN algorithm based on feature similarity approach.
* Code :-

print("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

wine = datasets.load\_wine()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(wine.data, wine.target, test\_size=0.20,random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=5)

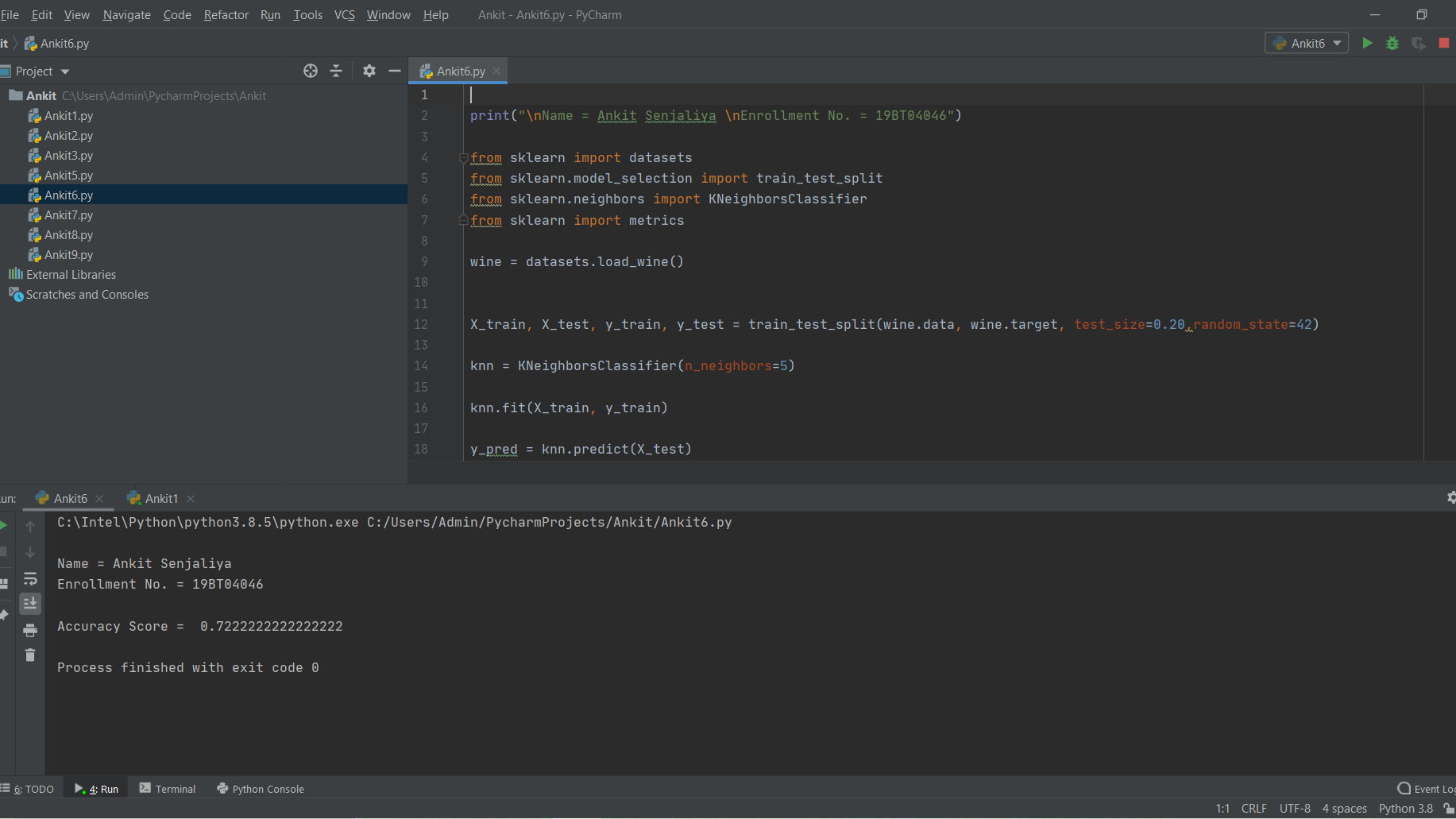
knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy\_score = metrics.accuracy\_score(y\_test, y\_pred)

print("\nAccuracy Score = ", accuracy\_score)

* Output :-



**PRACTICAL = 7**

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

print("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

import pandas as pd

dataset = pd.read\_csv("C:/Ankit Senjaliya/Ankit1.csv")

dataset.head()

dataset.shape

X = dataset.drop("Class", axis=1)

y = dataset["Class"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

classifier = DecisionTreeClassifier()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

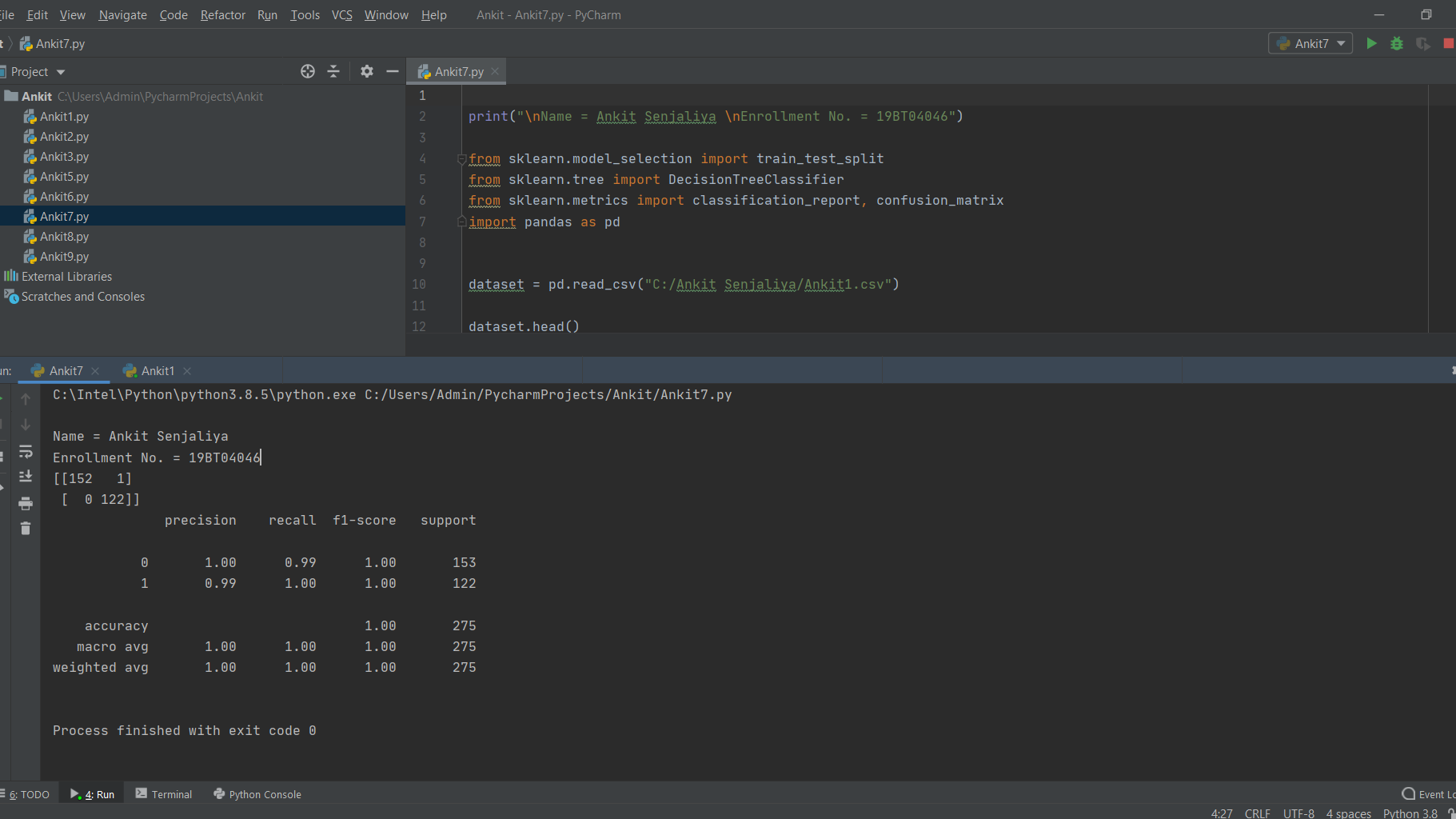
confusion\_matrix = confusion\_matrix(y\_test, y\_pred)

print(confusion\_matrix)

classification\_report = classification\_report(y\_test, y\_pred)

print(classification\_report)

* Output :-



**PRACTICAL = 9**

* Aim :-
* Implement the Linear Regression and Logistic Regression models
* Theory :-
* Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems. The description of both the algorithms is given below along with difference table.
* Code :-

print("\nName = Ankit Senjaliya \nEnrollment No. = 19BT04046")

from sklearn import datasets

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = datasets.load\_boston()

data.keys()

data.feature\_names

df = pd.DataFrame(data.data, columns=data.feature\_names)

target = pd.DataFrame(data.target, columns=["MEDV"])

model = LinearRegression()

X = df

y = target["MEDV"]

X = np.asarray(X.RM)

x = pd.DataFrame(X, columns =["RM"])

X\_train ,X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

model.fit(X\_train,y\_train)

predict = model.predict(X\_test)

df1 = pd.DataFrame(predict, columns=["predict"])

df1["test"] = y\_test

mean\_squared\_error = metrics.mean\_squared\_error(y\_test,predict)

print("\nMean Squared Error = ", mean\_squared\_error)

* Output :-

