# 

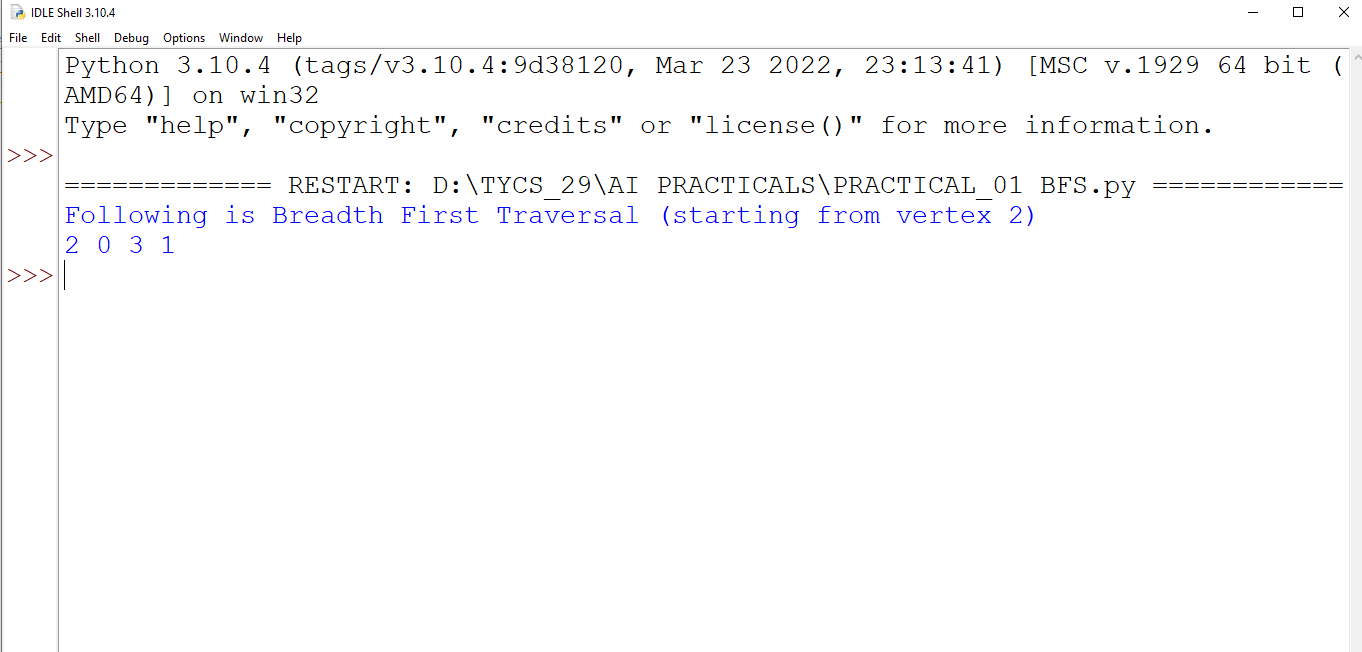
# PRACTICAL 1

**AIM: BREADTH FIRST SEARCH**

**SOURCE CODE:**

|  |
| --- |
| from collections import defaultdict  class Graph:  def \_\_init\_\_(self):  self.graph=defaultdict(list)  def addEdge(self,u,v):  self.graph[u].append(v)  def BFS(self, s):  visited = [False]\*(len(self.graph))  queue = []  queue.append(s)  visited[s] = True  while queue:  s = queue.pop(0)  print(s, end = " ")  for i in self.graph[s]:  if visited[i] == False:  queue.append(i)  visited[i] = True  g = Graph()  g.addEdge(0,1)  g.addEdge(0,2)  g.addEdge(1,2)  g.addEdge(2,0)  g.addEdge(2,3)  g.addEdge(3,3)  print("Following is Breadth First Traversal"" (starting from vertex 2)")  g.BFS(2) |

OUTPUT:



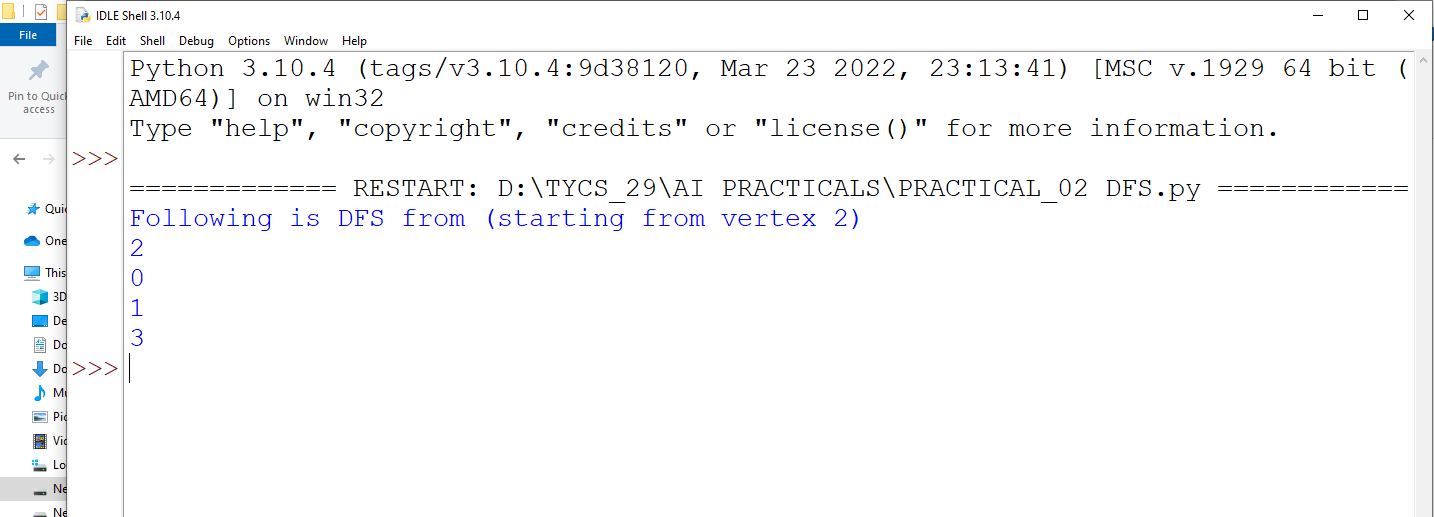
# PRACTICAL 2

AIM: Depth First Search .

Source Code:

|  |
| --- |
| from collections import defaultdict  class Graph:  def \_\_init\_\_(self):  self.graph=defaultdict(list)  def addEdge(self,u,v):  self.graph[u].append(v)  def DFSUtil(self,v,visited):  visited[v]=True  print(v)  for i in self.graph[v]:  if visited[i]==False:  self.DFSUtil(i, visited)  def DFS(self,v):  visited=[False]\*(len(self.graph))  self.DFSUtil(v,visited)  g=Graph()  g.addEdge(0,1)  g.addEdge(0,2)  g.addEdge(1,2)  g.addEdge(2,0)  g.addEdge(2,3)  g.addEdge(3,3)  print("Following is DFS from (starting from vertex 2)")  g.DFS(2) |

Output:

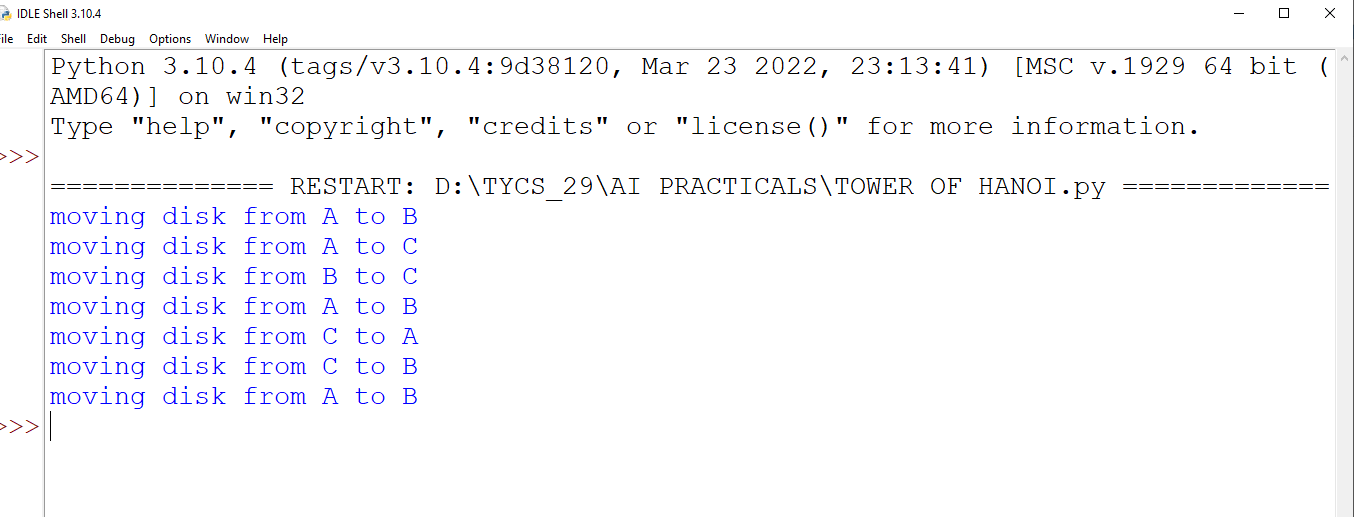


TOWER OF HANOI

Source Code:

|  |
| --- |
| def moveTower(height,fromPole,toPole,withPole):  if height>=1:  moveTower(height-1,fromPole,withPole,toPole)  moveDisk(fromPole,toPole)  moveTower(height-1,withPole,toPole,fromPole)  def moveDisk(fp,tp):  print("moving disk from",fp,"to",tp)  moveTower(3,"A","B","C") |

Output:



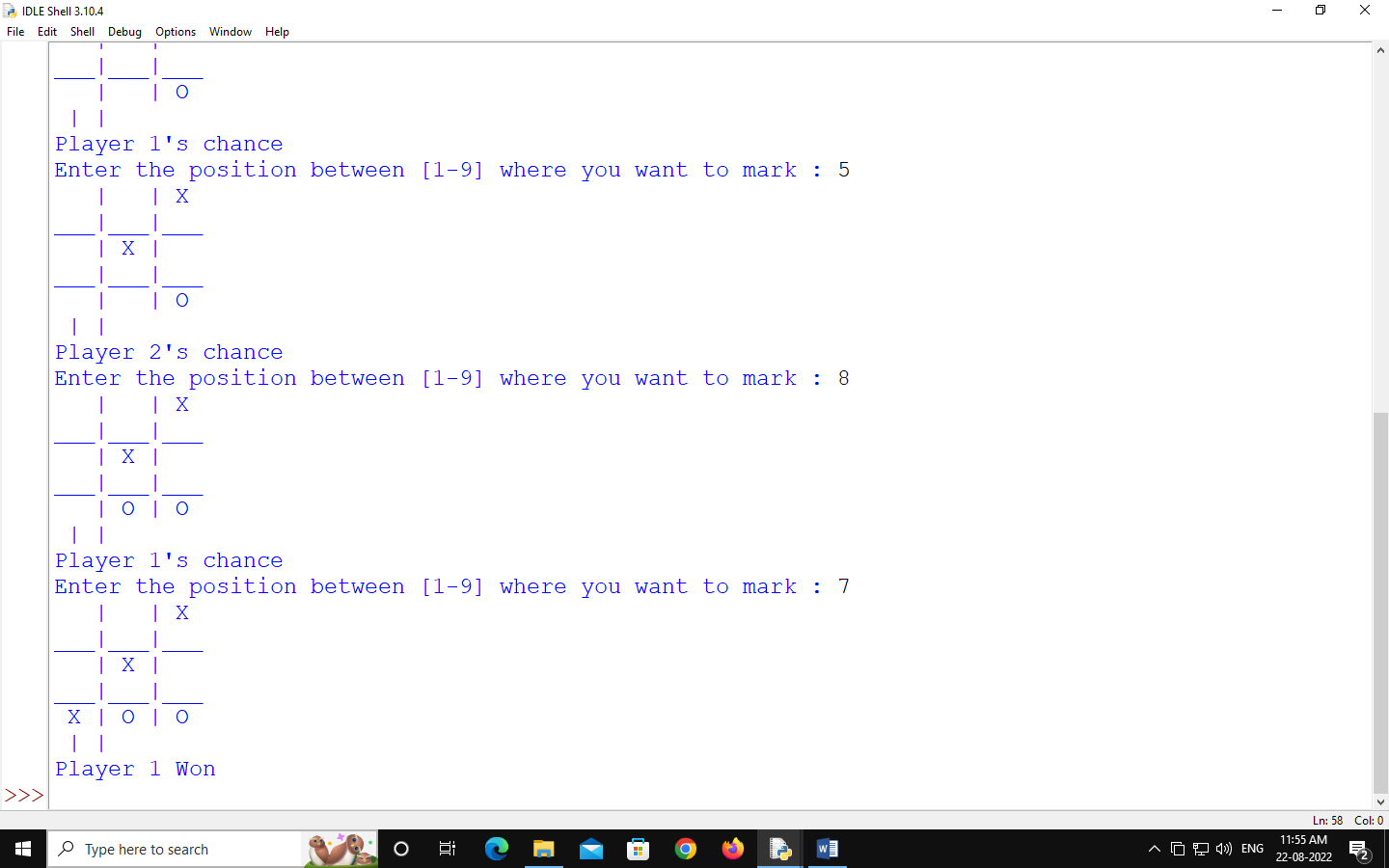
# Practical 4

Aim: Design the simulation of TIC-TAC-TOE game using Min-Max algorithm.

Source Code:

|  |
| --- |
| import os  import time  board=[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']  player =1  #####win Flags###  Win=1  Draw=-1  Running = 0  Stop=1  ##################  Game=Running  Mark='X'  #This Function Draws Game Board  def DrawBoard():  print(" %c | %c | %c " % (board[1],board[2],board[3]))  print("\_\_\_|\_\_\_|\_\_\_")  print(" %c | %c | %c " % (board[4],board[5],board[6]))  print("\_\_\_|\_\_\_|\_\_\_")  print(" %c | %c | %c " % (board[7],board[8],board[9]))  print(" | | ")  #This Function Checks position is empty or not  def CheckPosition(x):  if(board[x] == ' '):  return True  else:  return False  #This Function Checks player has won or not  def CheckWin():  global Game  #Horizontal winning condition  if (board[1]==board[2] and board[2]==board[3]and board[1] != ' '):  Game=Win  elif(board[4]==board[5] and board[5]==board[6]and board[4] != ' '):  Game=Win  elif(board[7]==board[8] and board[8]==board[9]and board[7] != ' '):  Game=Win  #Vertical Winning Condition  elif(board[1]==board[4] and board[4]==board[7]and board[1] != ' '):  Game=Win  elif(board[2]==board[5] and board[5]==board[8]and board[2] != ' '):  Game=Win  elif(board[3]==board[6] and board[6]==board[9]and board[3] != ' '):  Game=Win    #Diagonal Winnig Condition  elif(board[1]==board[5] and board[5]==board[9]and board[5] != ' '):  Game=Win  elif(board[3]==board[5] and board[5]==board[7]and board[5] != ' '):  Game=Win  #Match Tie or Draw Condition  elif(board[1]!=' ' and board[2]!=' ' and board[3] != ' ' and board[4]!=' ' and board[5]!=' ' and board[6]!=' ' and board[7]!=' 'and board[8]!=' ' and board[9]!=' '):  Game=Draw  else:  Game = Running  print("Tic-Tac-Toe Game")  print("Player 1[x] --- Player 2 [O]\n")  print()  print()  print("Please Wait...")  time.sleep(1)  while(Game==Running):  os.system('cls')  DrawBoard()  if(player % 2!= 0):  print("Player 1's chance")  Mark='X'  else:  print("Player 2's chance")  Mark='O'  choice=int(input("Enter the position between [1-9] where you want to mark : "))  if(CheckPosition(choice)):  board[choice]=Mark  player+=1  CheckWin()  os.system('cls')  DrawBoard()  if(Game==Draw):  print("Game Draw")  elif(Game==Win):  player-=1  if(player%2!=0):  print("Player 1 Won")  else:  print("Player 2 Won") |

Output:



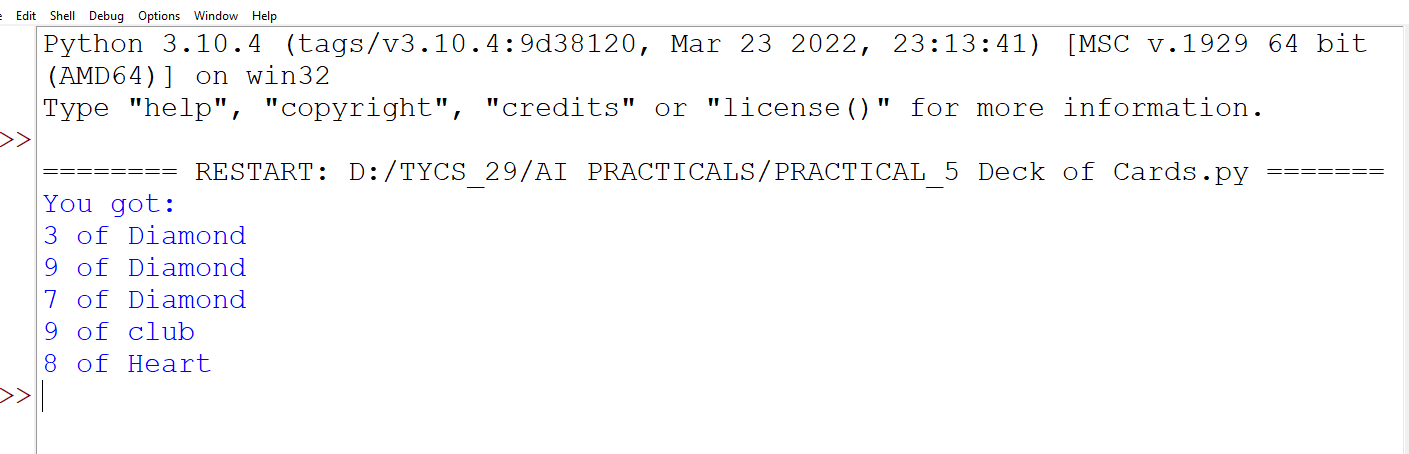
# PRACTICAL 5

AIM: Write a program to shuffle deck of cards.

Source Code:

|  |
| --- |
| import itertools, random  deck= list(itertools.product(range(1,14),['spade','Heart','Diamond','club']))  random.shuffle(deck)  print("You got:")  for i in range(5):  print(deck[i][0], 'of', deck[i][1]) |

Output:



# PRACTICAL 6

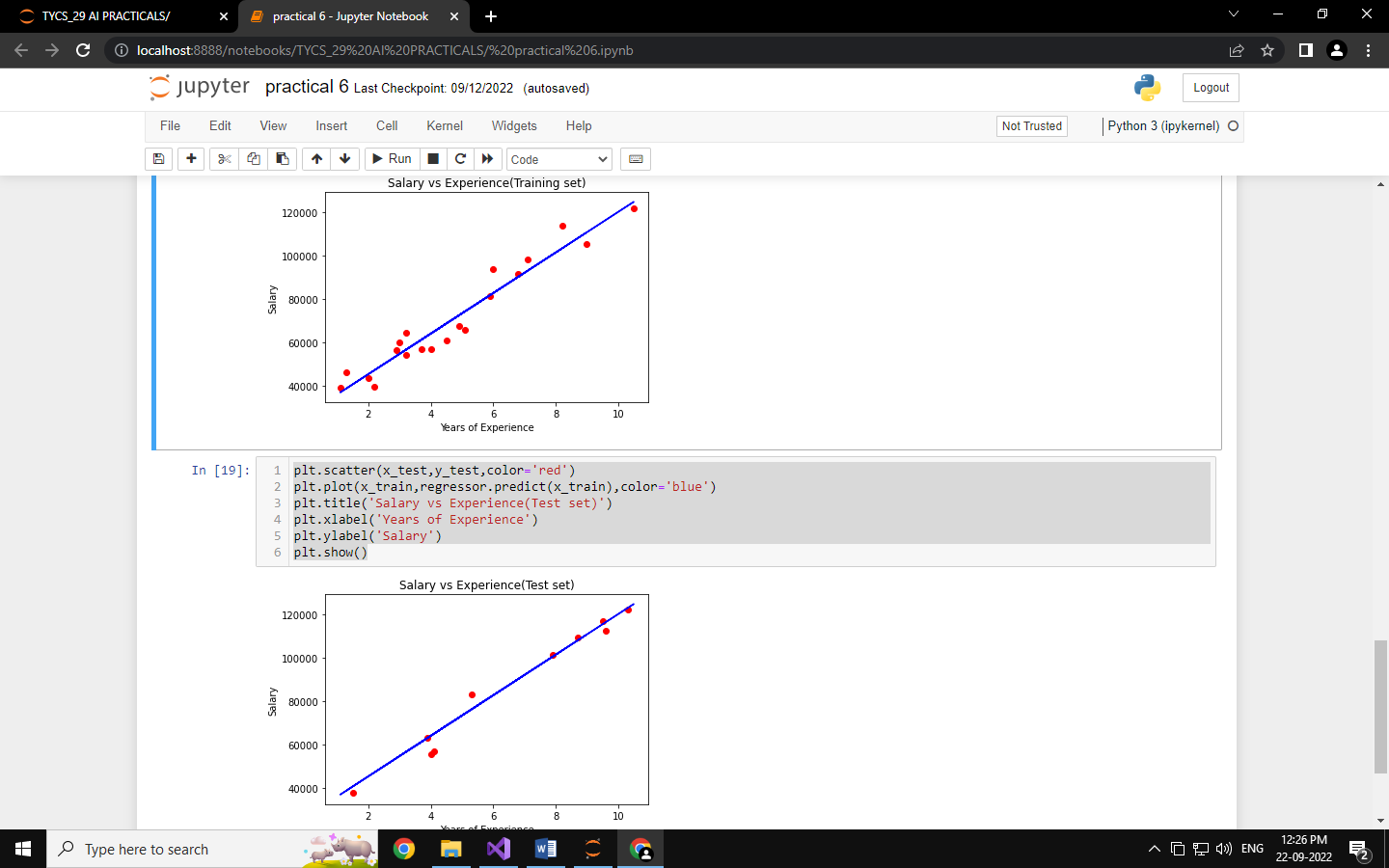
AIM: Write a program to implement simple linear regression machine learning Model(Supervised Learning)

SOURCE CODE:

OPEN JUPYTER NOTEBOOK:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  #Importing the Dataset  dataset=pd.read\_csv('Salary\_Data.csv')  x=dataset.iloc[:, :-1].values  y=dataset.iloc[:, -1].values  print(x)  then run  print(y)  then run  from sklearn.model\_selection import train\_test\_split  x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,test\_size=1/3,random\_state=0)  RUN  print(x\_train)  RUN  print(x\_test)  RUN  print(y\_train)  RUN  print(y\_test)  RUN  from sklearn.linear\_model import LinearRegression  regressor=LinearRegression()  regressor.fit(x\_train,y\_train)  RUN  #Predicting the test set results  y\_pred=regressor.predict(x\_test)  **visualising the test set results**  plt.scatter(x\_train,y\_train,color='red')  plt.plot(x\_train,regressor.predict(x\_train),color='blue')  plt.title('Salary vs Experience(Training set)')  plt.xlabel('Years of Experience')  plt.ylabel('Salary')  plt.show()  RUN  plt.scatter(x\_test,y\_test,color='red')  plt.plot(x\_train,regressor.predict(x\_train),color='blue')  plt.title('Salary vs Experience(Test set)')  plt.xlabel('Years of Experience')  plt.ylabel('Salary')  plt.show() |

RUN

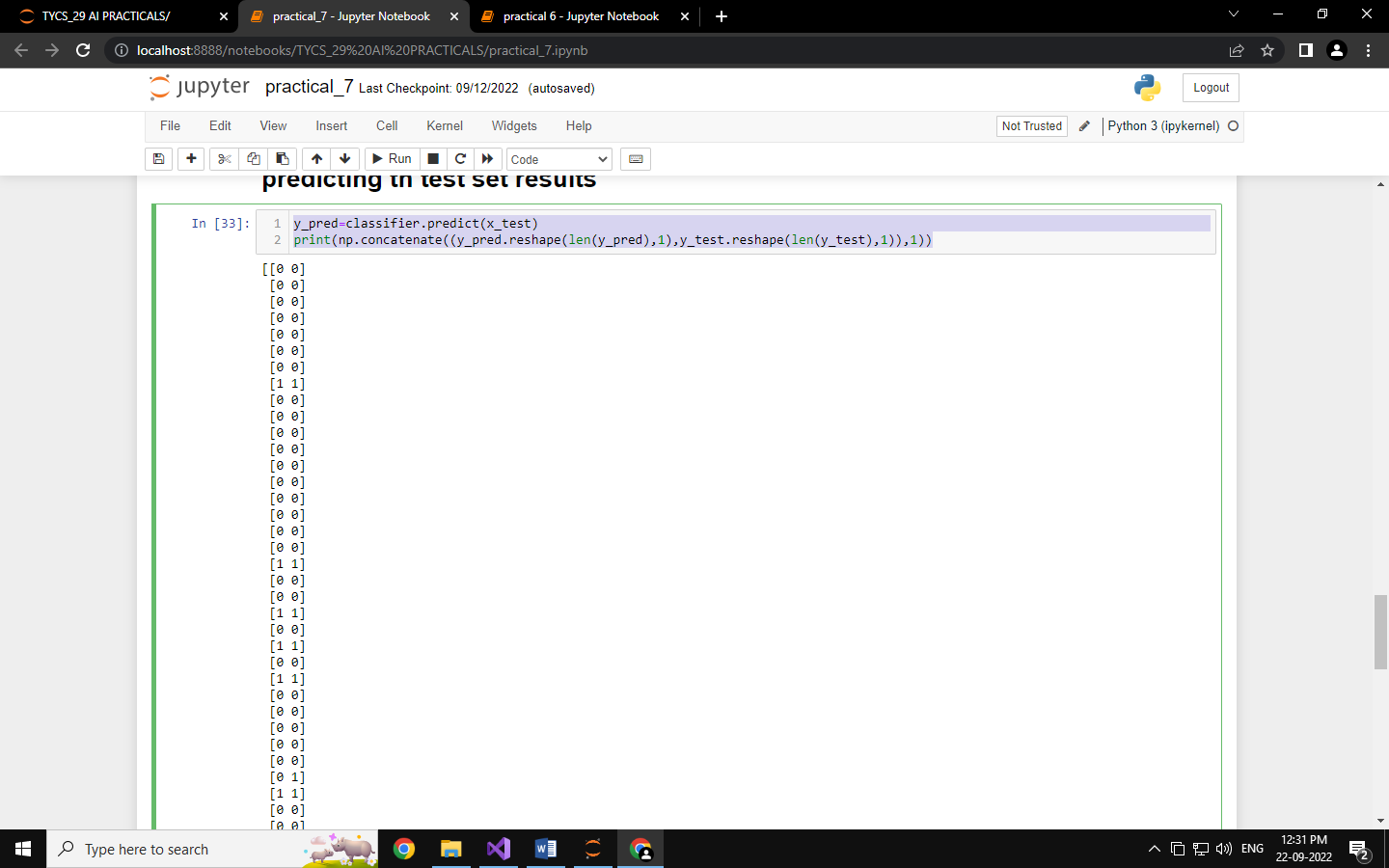


# PRACTICAL 7

SOURCE CODE:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  **importing the dataset**  dataset =pd.read\_csv('Social\_Network\_Ads.csv')  x=dataset.iloc[:, :-1].values  y=dataset.iloc[:, -1].values  **Splitting the dataset into the Training set and test set**  from sklearn.model\_selection import train\_test\_split  x\_train, x\_test, y\_train, y\_test=train\_test\_split(x, y, test\_size=0.25, random\_state=0)  RUN  print(x\_train)  RUN  print(y\_train)  RUN  **Feature Scaling**  from sklearn.preprocessing import StandardScaler  sc=StandardScaler()  x\_train=sc.fit\_transform(x\_train)  x\_test=sc.transform(x\_test)  RUN  print(x\_train)  RUN  print(x\_test)  RUN  **TRaining the SVM model on the training set**  from sklearn.svm import SVC  classifier =SVC(kernel='linear', random\_state=0)  classifier.fit(x\_train,y\_train)  RUN  **Predicting a new result**  print(classifier.predict(sc.transform([[30,87000]])))  RUN  **predicting th test set results**  y\_pred=classifier.predict(x\_test)  print(np.concatenate((y\_pred.reshape(len(y\_pred),1),y\_test.reshape(len(y\_test),1)),1)) |

RUN



# PRACTICAL 8

SOURCE CODE:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  **Importing the dataset**  dataset=pd.read\_csv('Mall\_Customers.csv')  x=dataset.iloc[:, [3,4]].values  RUN  print(x)  RUN  from sklearn.cluster import KMeans  kmeans=KMeans(n\_clusters=5,init = 'k-means++', random\_state=42)  y\_kmeans=kmeans.fit\_predict(x)  print(y\_kmeans)  RUN  plt.scatter(x[y\_kmeans==0,0],x[y\_kmeans==0,1],s=100, c='red',label='cluster 1')  plt.scatter(x[y\_kmeans==1,0],x[y\_kmeans==1,1],s=100, c='blue',label='cluster 2')  plt.scatter(x[y\_kmeans==2,0],x[y\_kmeans==2,1],s=100, c='green',label='cluster 3')  plt.scatter(x[y\_kmeans==3,0],x[y\_kmeans==3,1],s=100, c='cyan',label='cluster 4')  plt.scatter(x[y\_kmeans==4,0],x[y\_kmeans==4,1],s=100, c='magenta',label='cluster 5')  plt.scatter(kmeans.cluster\_centers\_[:, 0],kmeans.cluster\_centers\_[:, 1],s=300,c='yellow', label='Centroids')  plt.title('Clusters of customers')  plt.xlabel('Annual Income(k$)')  plt.ylabel('Spending Score (1-100)')  plt.legend()  plt.show() |

RUN

