**Source Code:**

#Tower of Hanoi Algorithm

def tower\_of\_hanoi(n, source, auxiliary, target):

if n == 1:

print(f"Move disk 1 from {source} to {target}")

return

tower\_of\_hanoi(n - 1, source, target, auxiliary)

print(f"Move disk {n} from {source} to {target}")

tower\_of\_hanoi(n - 1, auxiliary, source, target)

# Get the number of disks from the user

n = int(input("Enter the number of disks: "))

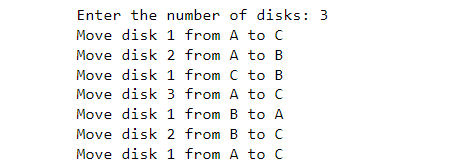
# Ensure the user enters a positive integer

if n > 0:

tower\_of\_hanoi(n, 'A', 'B', 'C')

else:

print("Please enter a positive integer for the number of disks.")

**Output:**

**Source Code:**

#BFS

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def add\_edge(self, u, v):

self.graph[u].append(v)

def bfs(self, start):

visited = set()

queue = [start]

while queue:

vertex = queue.pop(0)

if vertex not in visited:

print(vertex, end=' ')

visited.add(vertex)

queue.extend([neighbor for neighbor in self.graph[vertex] if neighbor not in visited])

# Create a graph

g = Graph()

g.add\_edge(0, 1)

g.add\_edge(0, 2)

g.add\_edge(1, 2)

g.add\_edge(2, 0)

g.add\_edge(2, 3)

g.add\_edge(3, 3)

# Get the start vertex from the user

start\_vertex = int(input("Enter the start vertex: "))

# Perform BFS from the user-specified start vertex

print("Breadth-First Traversal (starting from vertex", start\_vertex, "):")

g.bfs(start\_vertex)

**Output:**

Enter the start vertex: 2

Breadth-First Traversal (starting from vertex 2 ):

2 0 3 1

**Source Code:**

#DFS

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def add\_edge(self, u, v):

self.graph[u].append(v)

def dfs(self, vertex, visited):

visited.add(vertex)

print(vertex, end=' ')

for neighbor in self.graph[vertex]:

if neighbor not in visited:

self.dfs(neighbor, visited)

# Example usage:

g = Graph()

g.add\_edge(0, 1)

g.add\_edge(0, 2)

g.add\_edge(1, 2)

g.add\_edge(2, 0)

g.add\_edge(2, 3)

g.add\_edge(3, 3)

# Get the start vertex from the user

start\_vertex = int(input("Enter the start vertex: "))

print("Depth-First Traversal (starting from vertex", start\_vertex, "):")

visited = set()

g.dfs(start\_vertex, visited)

**Output:**

Enter the start vertex: 1

Depth-First Traversal (starting from vertex 1 ):

1 2 0 3

**Source Code:**

#Salesman Problem

def take\_input():

global n, cost

n = int(input("Enter the number of nodes: "))

cost = 0

ary = []

completed = [0] \* n

print("\nEnter the Cost Matrix:")

for i in range(n):

row = list(map(int, input().split()))

ary.append(row)

completed[i] = 0

return ary, completed

def min\_cost(city):

global cost

completed[city] = 1

print(city + 1, end=" ---> ")

ncity = least(city)

if ncity == 999:

ncity = 0

print(ncity + 1, end=" ---> ")

cost += ary[city][ncity]

return

min\_cost(ncity)

def least(c):

global cost

nc = 999

kmin = 999

i = 0

min = 999

for i in range(n):

if ary[c][i] != 0 and completed[i] == 0:

if ary[c][i] + ary[i][c] < min:

min = ary[c][i] + ary[i][c]

kmin = ary[c][i]

nc = i

if min != 999:

cost += kmin

return nc

4

ary, completed = take\_input()

print("\n\nThe Path is:")

min\_cost(0) # passing 0 because it's the starting vertex

print("\n\nMinimum cost is", cost)

**Output:**

Enter the number of nodes: 4

Enter the Cost Matrix:

0 3 4 6

1 0 3 3

4 1 0 4

2 4 3 0

The Path is:

1 ---> 2 ---> 3 ---> 4 ---> 1 --->

Minimum cost is 12

**Source Code:**

#create and load csv file

import pandas as pd

data= {'Name': ['Jai','Princi','Gaurav','Anju','Ravi','Natasha','Riya'],

'Age': [17,17,18,17,18,17,17],

'Gender': ['M','F','M','M','M','F','F'],

'Marks': [90,76,'NaN',74,65,'NaN',71]}

df =pd.DataFrame(data)

df

df = pd.read\_csv('1.car driving risk analysis.csv')

df

df.head()

df.tail()

#calculate mean , median,mode,var,sd from dataset

import csv

import random

import statistics

# Create a CSV file with random data

with open('dataset.csv', 'w', newline='') as csvfile:

csv\_writer = csv.writer(csvfile)

csv\_writer.writerow(['Value']) # Write the header

for \_ in range(100):

value = random.randint(1, 100)

csv\_writer.writerow([value])

data = []

with open('dataset.csv', 'r') as csvfile:

csv\_reader = csv.DictReader(csvfile)

for row in csv\_reader:

data.append(int(row['Value']))

df = pd.read\_csv('dataset.csv')

# Calculate statistics

mean = statistics.mean(data)

median = statistics.median(data)

mode = statistics.mode(data)

variance = statistics.variance(data)

stdev = statistics.stdev(data)

# Print the results

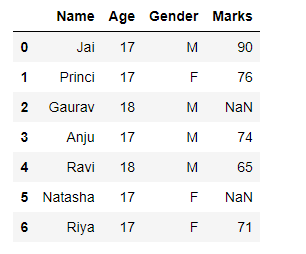
print(f"Mean: {mean}")

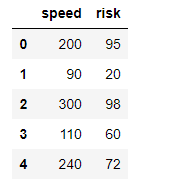
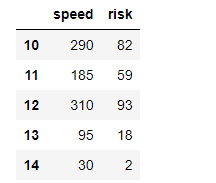
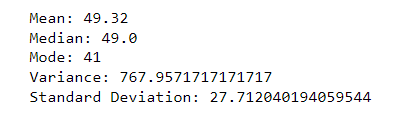
print(f"Median: {median}")

print(f"Mode: {mode}")

print(f"Variance: {variance}")

print(f"Standard Deviation: {stdev}")

**Output:**

****

**Source Code:**

**#linear Regression and plot the graph**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df=pd.read\_csv('1.car driving risk analysis.csv')

df

x=df[['speed']] #only 3rd bracket,,two dimension for input means independent variable

y=df['risk'] #only 3rd bracket,,one dimension for outpu means dependent variable

x

plt.scatter(df['speed'],df['risk'])

plt.xlabel('Speed of Car')

plt.ylabel('Risk on driving')

plt.title('Car driving speed risk')

xtest

ytrain

from sklearn.linear\_model import LinearRegression

reg=LinearRegression()

reg.fit(xtrain,ytrain)

LinearRegression()

reg.predict(xtest) #compare with ytest

plt.scatter(df['speed'],df['risk'],marker='\*',color='red')

plt.xlabel('Speed of Car')

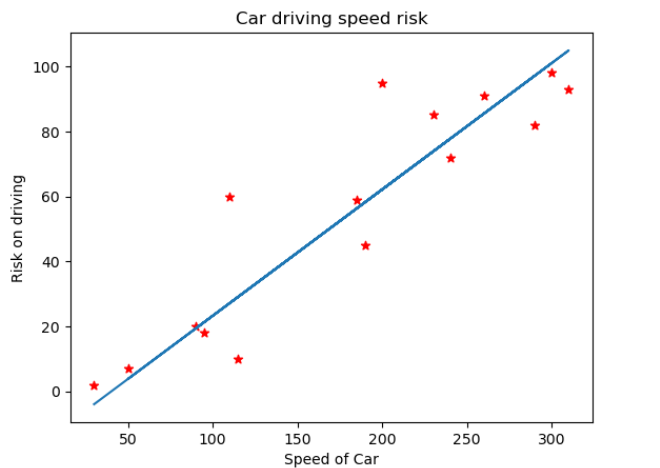
plt.ylabel('Risk on driving')

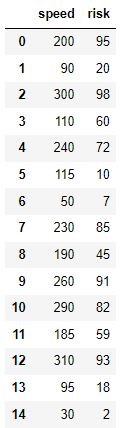
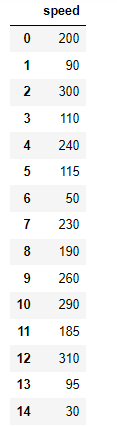
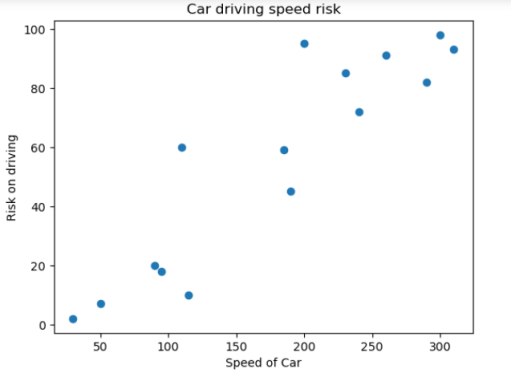
plt.title('Car driving speed risk')

plt.plot(df.speed,reg.predict(df[['speed']]))

reg.predict([[180]])

reg.coef\_

**Output:**

****

array([54.37693451])

array([0.38891318])

**Source Code:**

#Find S Algorithm

import pandas as pd

import numpy as np

df= pd.read\_csv('S algorithm.csv')

x = np.array(df)[:,:-1] #ignore enjoy spot because it's depedent variable

y = np.array(df)[:,-1] #take only last column,it's dependent variable

def train(ind,dep): #ind=independent,dep=dependent

for i,val in enumerate(dep): #i=index:0,1,2,3 /val=yes

if val =='yes':

specific= ind[i].copy()

break

for i, val in enumerate(ind): #i=0(index),val=['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

#print(i)

#print(val)

if dep[i]=='yes':

for j in range(len(specific)):

if val[j] !=specific[j]:

specific[j]='?'

else:

pass

return specific

result=train(x,y)

result

day=input("Enter 6 word to check: ")

day=day.split()

check=True

for i in range(len(result)):

if result[i]=='?' or result[i]==day[i]:

check=True;

else:

check=False;

break;

if check:

print("Enjoy spot")

else:

print("Not Enjoy")

**Output:**

array(['sunny', 'warm', '?', 'strong', '?', '?'], dtype=object)

Enter 6 word to check: ss warm

Not Enjoy

**Source Code:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

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

# Load the dataset

data = pd.read\_csv('Social\_Network\_Ads.csv')

# Select features and target variable

X = data.iloc[:, [0, 1]].values # Assuming columns 'Age' and 'EstimatedSalary' are the relevant features

y = data.iloc[:, 2].values # Assuming column 'Purchased' is the target variable

# 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)

# Feature scaling (important for SVM)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create and train the SVM model

svm\_classifier = SVC(kernel='linear', random\_state=42)

svm\_classifier.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = svm\_classifier.predict(X\_test)

# Evaluate the model

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

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

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

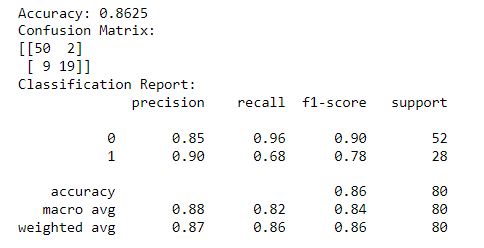
print(f'Accuracy: {accuracy}')

print('Confusion Matrix:')

print(confusion)

print('Classification Report:')

print(report)

**Output:**