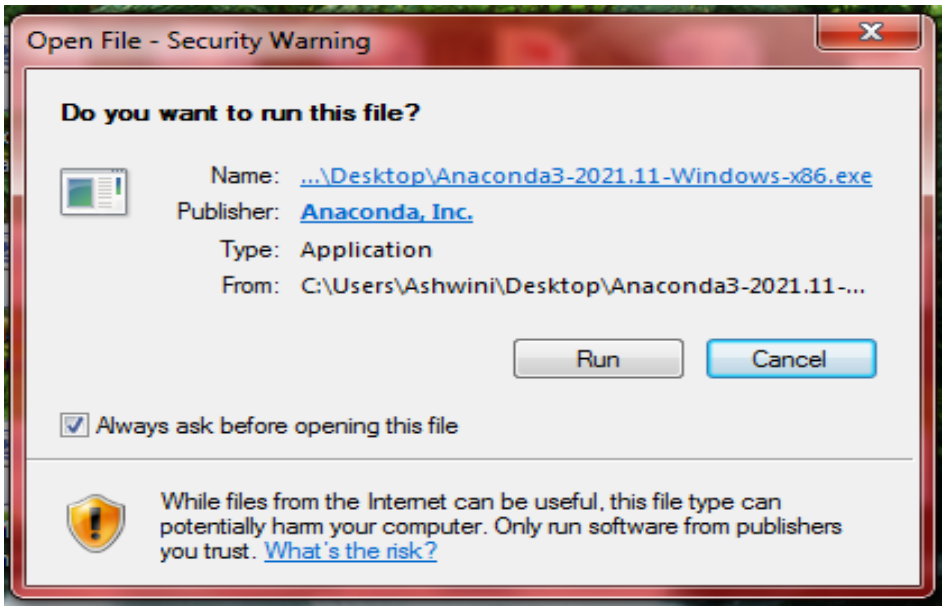
Installing Anaconda software on Windows

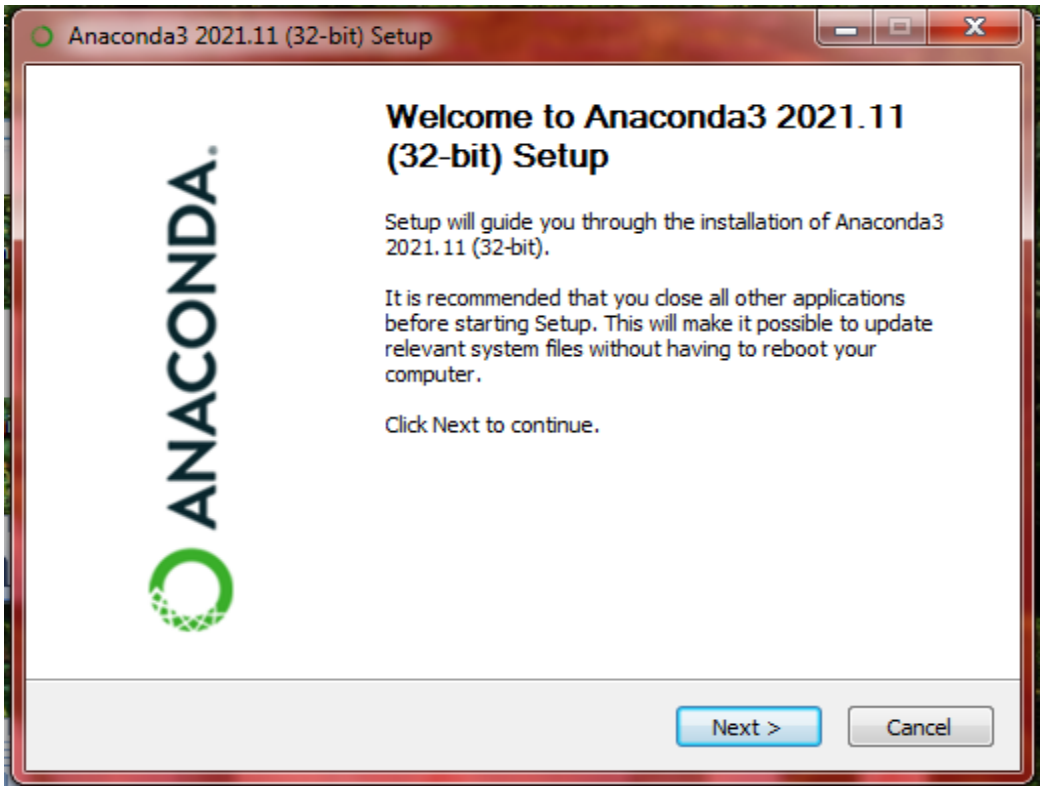
This tutorial will demonstrate how to install Anaconda, a powerful package manager, on Microsoft Windows.

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. This is advantageous as when you are working on a data science project, you will find that you need many different packages (numpy, scikit-learn, scipy, pandas to name a few), which an installation of Anaconda comes preinstalled with. If you need additional packages after installing Anaconda, you can use Anaconda's package manager, conda, or pip to install those packages. This is highly advantageous as you don't have to manage dependencies between multiple packages yourself. Conda even makes it easy to switch between Python 2 and 3 (you can learn more here). In fact, an installation of Anaconda is also the recommended way to install Jupyter Notebooks.

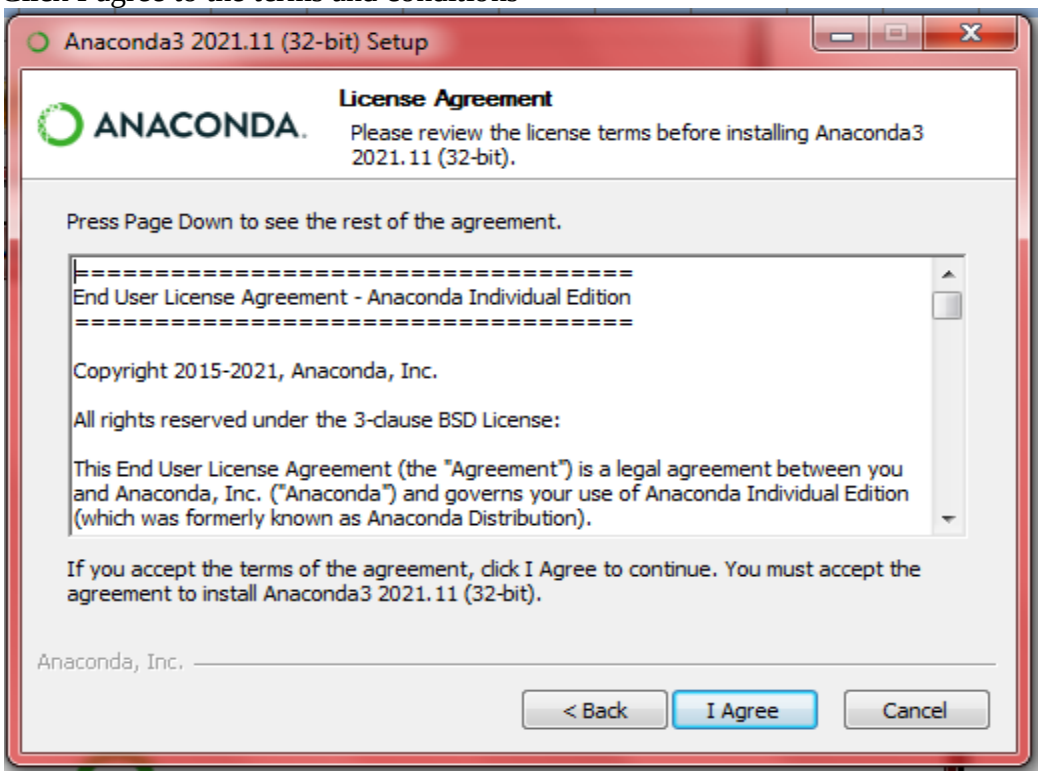
a. Go to Anaconda.com, and download the Anaconda version for Windows.
b. Download the Python 3 version for Windows.
c. Double-click on the executable file.
d. Click Run



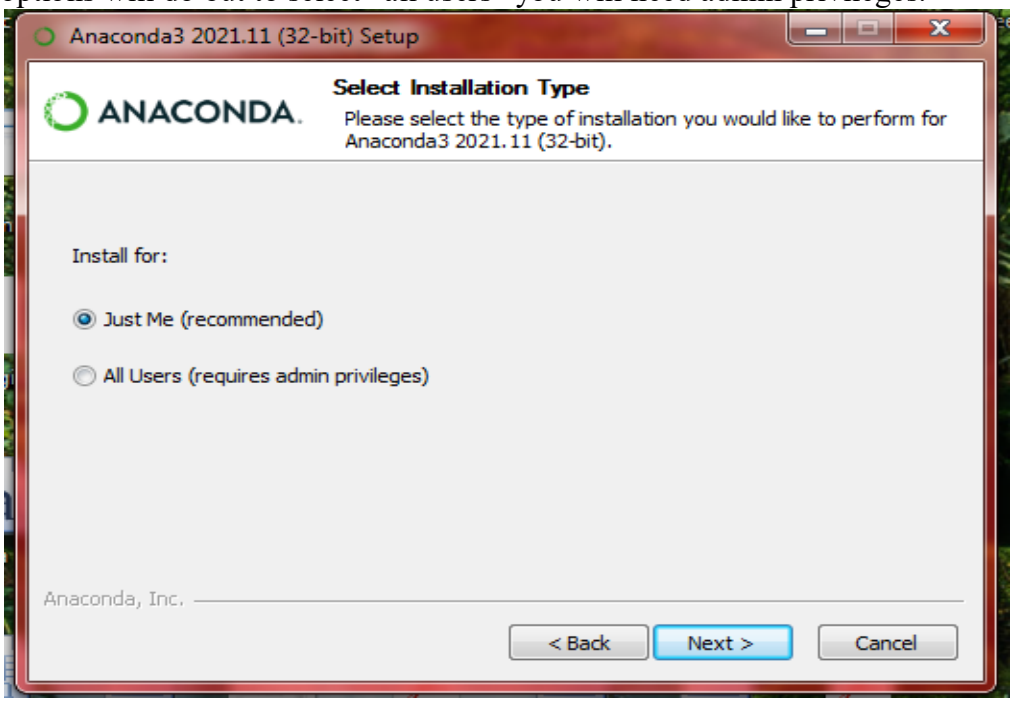
e. Click Next



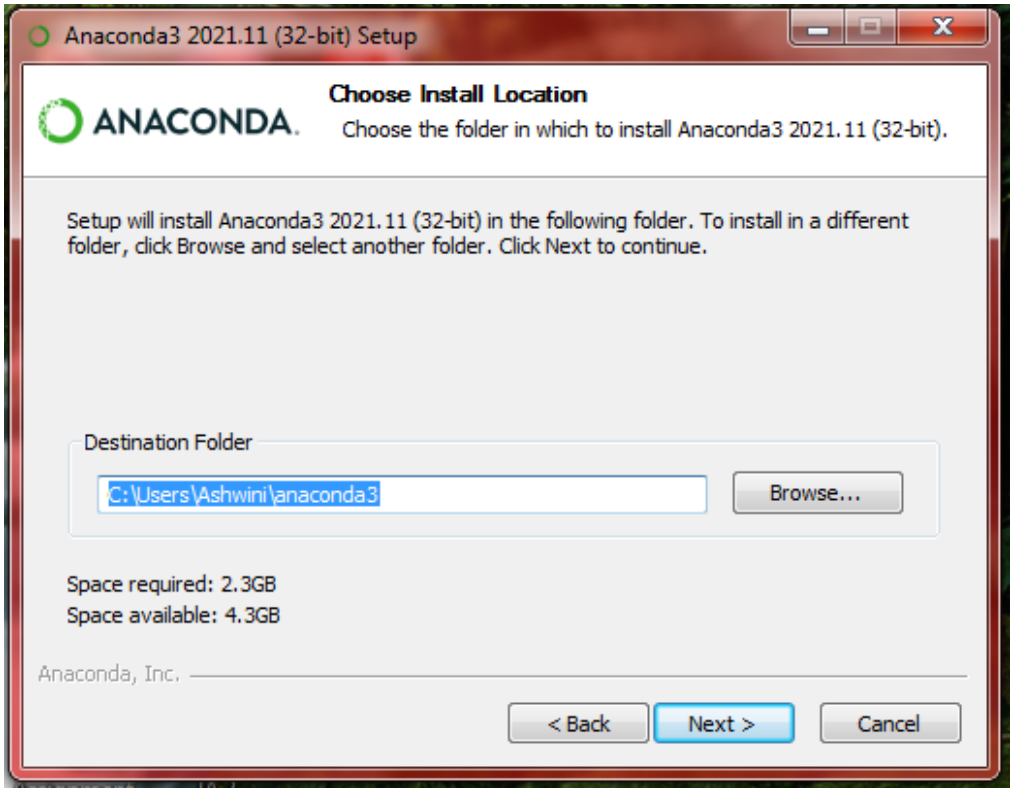
f. Click I agree to the terms and conditions



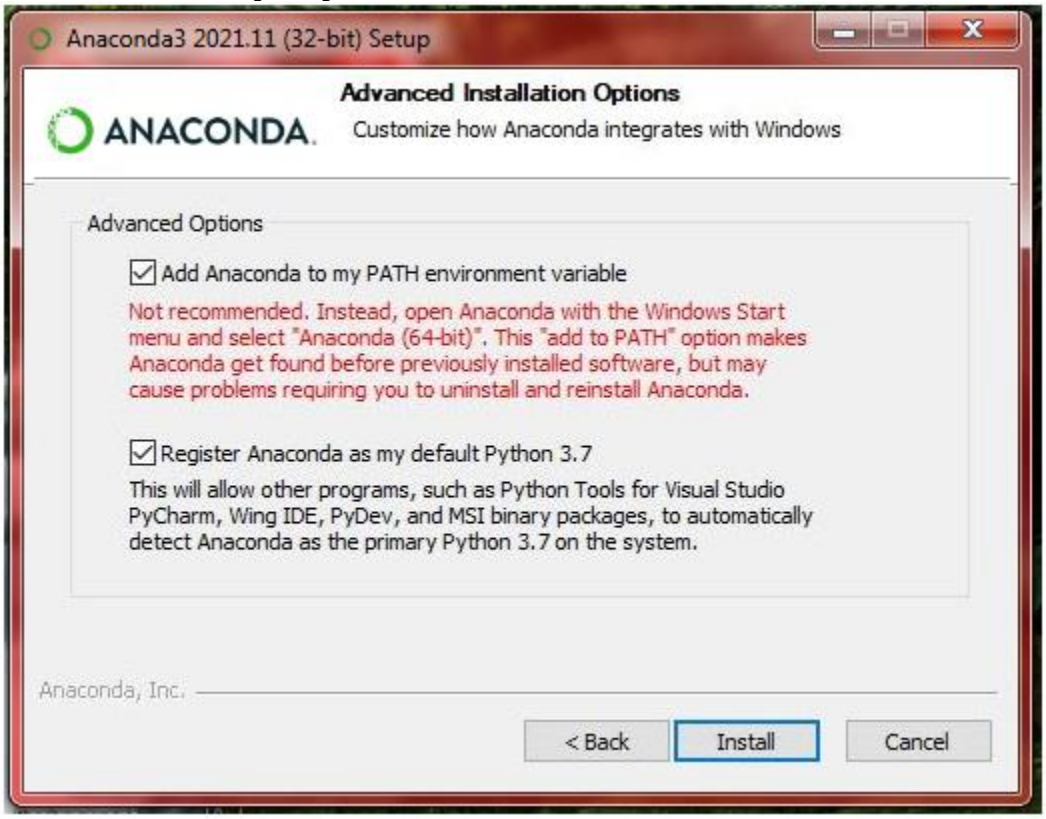
g. This step will ask you if you want to install Anaconda just for you or for all the users using this PC. Click "Just-Me", or "All users", depending on your preference. Both options will do but to select "all users" you will need admin privileges.



h. Select the installation location

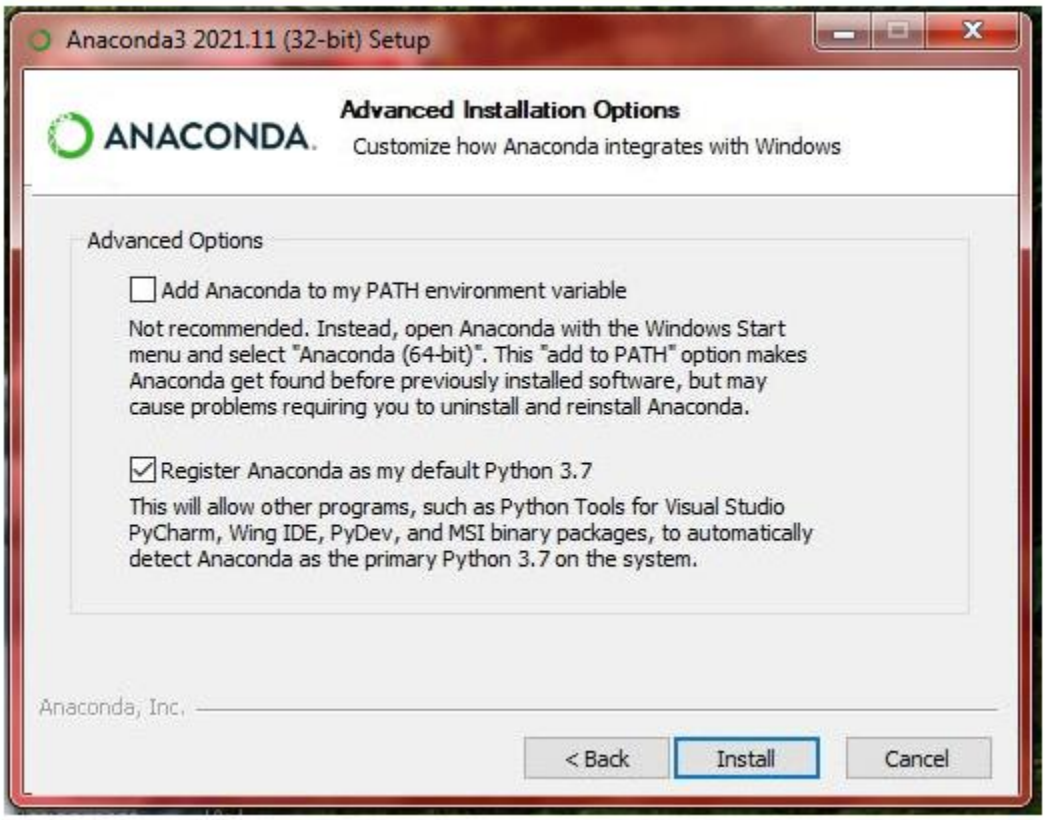


i. Selecting the *Add Anaconda to my PATH environment variable* option allow us to use Anaconda in the command prompt.

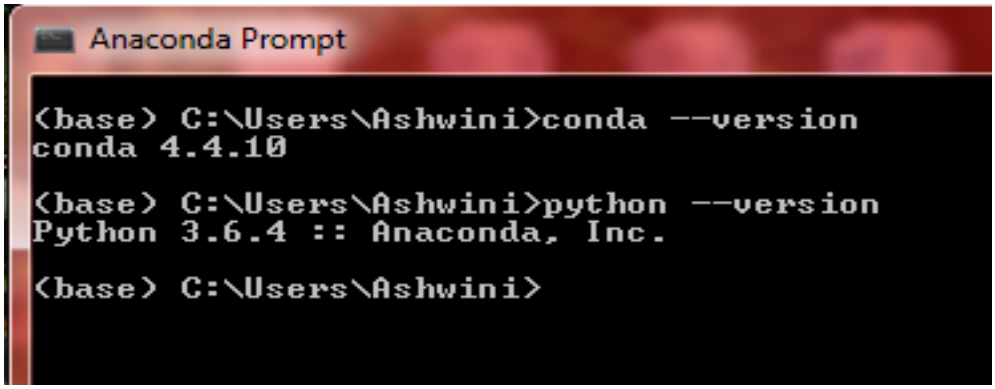


Check whether python is installed from the command prompt by executing command to check python and conda version.

j. Leaving it unchecked means that we have to use Anaconda Command Prompt in order to use Anaconda.



Check whether python is installed from the Anaconda prompt. by executing command to check python and conda version.



1. **Implement A* Search algorithm.**

```
class Graph:
    def __init__(self,adjac_list):
        self.adjac_list=adjac_list
        print("Input Graph:\n",self.adjac_list)
    def get_neighbors(self,v):
        return self.adjac_list[v]
    def h(self,n):
        H={'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0, }
        return H[n]
    def AStar(self, start, stop):
        open_list=set([start])
        closed_list=set([])
        g={}
        g[start]=0
        parents={}
        parents[start]=start
        while len(open_list)>0:
            n=None
            for v in open_list:
                if n==None or g[v]+self.h(v)<g[n]+self.h(n):
                    n=v
            if n==None:
               print('Path does not exist!')
               return None
            if n==stop:
               reconst_path=[]
               while parents[n]!=n:
```

```
                    reconst_path.append(n)
                    n=parents[n]
                reconst_path.append(start)
                reconst_path.reverse()
                print('path found: {}'.format(reconst_path))
                print('cost of the path is:',g[stop])
                return reconst_path
            for(m,weight) in self.get_neighbors(n):
                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    parents[m]=n
                    g[m]=g[n]+weight
                else:
                    if g[m]>g[n]+weight:
                        g[m]=g[n]+weight
                        parents[m]=n
                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)
            open_list.remove(n)
            closed_list.add(n)

        print('Path doesnot exists!')
        return None
adjac_list={'A': [('B', 2), ('E', 3)],
 'B': [('C', 1),('G', 9)],
 'C': None,
 'E': [('D', 6)],
 'D': [('G', 1)] }
graph1=Graph(adjac_list)
graph1.AStar('A','G')
```

Output:

```
Input Graph:
 {'A': [('B', 2), ('E', 3)], 'B': [('C', 1), ('G', 9)], 'C': None, 'E':
[('D', 6)], 'D': [('G', 1)]}


path found: ['A', 'E', 'D', 'G']


cost of the path is: 10
```

2. Implement AO* Search algorithm.

```
class Graph:
    def __init__(self, graph, hVals, startNode):
        self.graph = graph
        self.H=hVals
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}
    def getNeighbors(self, v):
        return self.graph.get(v,'')
    def getStatus(self,v):
        return self.status.get(v,0)
    def setStatus(self,v, val):
        self.status[v]=val
    def getHval(self, n):
        return self.H.get(n,0)
    def setHval(self, n, value):
        self.H[n]=value
    def printSolution(self):
        print("Final HEURISTIC VALUES :\n", self.H)
        print()
        print("Best Path to goal state:")
        print(self.solutionGraph)
        print("\n With minimum cost", self.H[self.start])

    def computeMinCost(self, v):
        minimumCost=0
        costList={}
        costList[minimumCost]=[]
        flag=True
        for nodes in self.getNeighbors(v):
            cost=0
```

```
            nodeList=[]
            for c, weight in nodes:
                cost=cost+self.getHval(c)+weight
                nodeList.append(c)

            if flag==True:
                minimumCost=cost
                costList[minimumCost]=nodeList
                flag=False
            else:
                if minimumCost>cost:
                    minimumCost=cost
                    costList[minimumCost]=nodeList
        return minimumCost, costList[minimumCost]
    def AOStar(self, v, backTracking):
        if self.getStatus(v) >= 0:
            minimumCost, childList = self.computeMinCost(v)
            print(v,minimumCost)
            self.setHval(v, minimumCost)
            self.setStatus(v,len(childList))
            solved=True
            for childNode in childList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False
            if solved==True:
                self.setStatus(v,-1)
                self.solutionGraph[v]=childList
            if v!=self.start:
                self.AOStar(self.parent[v], True)
            if backTracking==False:
                for childNode in childList:
                    self.setStatus(childNode,0)
                    self.AOStar(childNode, False)
```

```
h1 = {'A': 0, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7,
'I': 7, 'J': 1}


graph1 = {
'A': [[('B', 1), ('C', 1)], [('D', 1)]],
'B': [[('G', 1)], [('H', 1)]],
'C': [[('J', 1)]],
'D': [[('E', 1), ('F', 1)]],
'G': [[('I', 1)]]
}
print('Input Graph:',graph1)
print()
print('Initial Heuristic values',h1)
G1= Graph(graph1, h1, 'A')
G1.AOStar('A',False)
G1.printSolution()

'''h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H':
7}
graph2 = {
'A': [[('B', 1), ('C', 1)], [('D', 1)]],
'B': [[('G', 1)], [('H', 1)]],
'D': [[('E', 1), ('F', 1)]]
}
G2 = Graph(graph2, h2, 'A')
G2.applyAOStar()
G2.printSolution()'''
```

Output:

Input Graph: {'A': [[('B', 1), ('C', 1)], [('D', 1)]], 'B': [[('G', 1)], [('H', 1)]], 'C': [[('J', 1)]], 'D': [[('E', 1), ('F', 1)]], 'G': [[('I', 1)]]}

Initial Heuristic values {'A': 0, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}

HEURISTIC VALUES :

{'A': 5, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

Best Path to goal state:

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

With minimum cost 5

3. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithmto output a description of the set of all hypotheses consistent with the training examples.

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts,'\n')
target = np.array(data.iloc[:,-1])
print(target,'\n')

def disp(g):
    for i in range(len(g)):
        print(g[i])

def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]

    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(" steps of Candidate Elimination Algorithm",i+1)
```

```
        print("specific_h",i+1,"\n",specific_h)
        print("\n general_h",i+1,)
        disp(general_h)
        print('\n')
    indices = [i for i, val in enumerate(general_h) if val == ['?',
'?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("\nFinal General_h:")
disp(g_final)
```

Output:

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

['yes' 'yes' 'no' 'yes']

 steps of Candidate Elimination Algorithm 1
specific_h 1
 ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

 general_h 1
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
```

steps of Candidate Elimination Algorithm 2

specific_h 2

['sunny' 'warm' '?' 'strong' 'warm' 'same']

general_h 2

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

steps of Candidate Elimination Algorithm 3

specific_h 3

['sunny' 'warm' '?' 'strong' 'warm' 'same']

general_h 3

['sunny', '?', '?', '?', '?', '?']

['?', 'warm', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', 'same']

steps of Candidate Elimination Algorithm 4

specific_h 4

['sunny' 'warm' '?' 'strong' '?' '?']

general_h 4

['sunny', '?', '?', '?', '?', '?']

['?', 'warm', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

Final Specific_h:

['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:

['sunny', '?', '?', '?', '?', '?']

['?', 'warm', '?', '?', '?', '?']

4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("playtennis.csv")
features = [x for x in data]
print(features)
features.remove("answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
```

```
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
printTree(root)
```

Output:

```
['Outlook', 'Temperature', 'Humidity', 'Wind', 'answer']
Outlook
    overcast -> ['yes']

    rain
        Wind
            strong -> ['no']

            weak -> ['yes']

    sunny
        Humidity
            high -> ['no']

            normal -> ['yes']
```

5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np

x=np.array(([2,9],[1,5],[3,6]), dtype=float)
y=np.array(([92],[86],[89]), dtype=float)
print('Input:\n',x)



x=x/np.amax(x,axis=0)
y=y/100



def sigmoid(x):
    return 1/(1+np.exp(-x))

def derivative_sigmoid(x):
    return x*(1-x)
epoch=7000
lr=0.1

input_units=2
hidden__units=3
output_units=1
wh=np.random.uniform(size=(input_units,hidden__units))
bh=np.random.uniform(size=(1,hidden__units))
wout=np.random.uniform(size=(hidden__units,output_units))
bout=np.random.uniform(size=(1,output_units))
for i in range(epoch):
    hinp=np.dot(x,wh)+bh
    hout=sigmoid(hinp)

    outinp=np.dot(hout,wout)+bout
```

```
    output=sigmoid(outinp)

    EO=y-output
    ogradient=derivative_sigmoid(output)
    doutput=EO*ogradient
    EH=doutput.dot(wout.T)
    hgradient=derivative_sigmoid(hout)
    dhidden=EH*hgradient

    wout+=hout.T.dot(doutput)*lr
    wh+=x.T.dot(dhidden)*lr

print('Normalized Input\n',x)
print("actual output:\n", y)
print("predicted output:\n",output)
```

Output:

```
Input:
 [[2. 9.]
 [1. 5.]
 [3. 6.]]
Normalized Input
 [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
actual output:
 [[0.92]
 [0.86]
 [0.89]]
predicted output:
 [[0.89760287]
 [0.87539739]
 [0.89630106]]
```

6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
train=pd.read_csv("playtennis.csv")
target='answer'
features=train.columns[train.columns!=target]
classes=train[target].unique()
test=pd.read_csv("playtennis.csv", skiprows=range(1,10), nrows=4)
print('Test dataset\n',test)

priorprob={}
likeprob={}
for x in classes:
    traincl=train[train[target]==x][features]
    tot=len(traincl)
    priorprob[x]=float(tot/len(train))
    clsp={}
    for col in traincl.columns:
        colp={}
        for val, cnt in traincl[col].value_counts().iteritems():
            pr=cnt/tot
            colp[val]=pr
        clsp[col]=colp
    likeprob[x]=clsp

def postprobs(x):
    postprob={}
    for cl in classes:
        pr=priorprob[cl]
        for col, val in x.iteritems():
            try:
                pr*=likeprob[cl][col][val]
            except:
```

```
                pr=0
            postprob[cl]=pr
        #print("Posterior Probability",postprob)
        return postprob
def classify(x):
    postprob=postprobs(x)
    probvalue=0
    maxclass=''
    for cl, pr in postprob.items():
        if pr>probvalue:
            probvalue=pr
            maxclass=cl
    return maxclass
b=[]
for i in train.index:
    b.append(classify(train.loc[i,features])==train.loc[i,target])
    #print(b)
print(sum(b),"correct of", len(train))
print("Accuracy:", sum(b)/len(train))


b=[]
for i in test.index:
    b.append(classify(test.loc[i,features])==test.loc[i,target])
    #print(b)
print(sum(b),"correct of", len(test))
print("Accuracy:", sum(b)/len(test))
```

Output:

Test dataset

	Outlook	Temperature	Humidity	Wind	answer
0	rain	mild	normal	weak	yes
1	sunny	mild	normal	strong	yes
2	overcast	mild	high	strong	yes
3	overcast	hot	normal	weak	yes

13 correct of 14

Accuracy: 0.9285714285714286

4 correct of 4

Accuracy: 1.0

7. **Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

```
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.mixture import GaussianMixture

iris=datasets.load_iris()
X=iris.data
Y=iris.target

X,Y = shuffle(X,Y)
model=KMeans(n_clusters=3,random_state=3425)
model.fit(X)
Y_Pred1=model.labels_
Y_Pred1
cm=confusion_matrix(Y,Y_Pred1)
print("K-Means Model")
print("Confusion Matrix\n",cm)
print("Accuracy score=",accuracy_score(Y,Y_Pred1))



model2=GaussianMixture(n_components=3,random_state=3425)
model2.fit(X)
Y_Pred2= model2.predict(X)
cm=confusion_matrix(Y,Y_Pred2)
print("EM Model")
print("Confusion Matrix\n",cm)
print("Accuracy score=",accuracy_score(Y,Y_Pred2))
```

Output:

K-Means Model

Confusion Matrix

[[50 0 0]

[0 2 48]

[0 36 14]]

Accuracy score= 0.44

EM Model

Confusion Matrix

[[50 0 0]

[0 5 45]

[0 50 0]]

Accuracy score= 0.36666666666666664

8. **Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.**

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris=datasets.load_iris()
iris_data=iris.data
iris_types=iris.target

X_train, X_test, y_train, y_test=
train_test_split(iris_data,iris_types,test_size=0.20)
print("\n Actual Output\n",y_test)

classifier=KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train,y_train)

y_pred=classifier.predict(X_test)
print("\n Predicted Output by KNeighborsClassifier with k=3\n",y_pred)
print("\n Classification Accuracy:", classifier.score(X_test,y_test))

print("\n Confusion matrix\n",confusion_matrix(y_test,y_pred))

print("\nAccuracy matrix\n",classification_report(y_test,y_pred))
```

Output:

Actual Output

[2 0 2 1 1 1 1 0 2 2 1 0 2 2 2 1 1 0 0 1 0 1 0 0 2 1 2 2 1 1]

Predicted Output by KNeighborsClassifier with k=3

[2 0 2 1 1 1 1 0 2 2 1 0 2 2 2 1 1 0 0 1 0 1 0 0 2 1 2 2 1 1]

Classification Accuracy: 1.0

Confusion matrix

[[8 0 0]

[0 12 0]

[0 0 10]]

Accuracy matrix

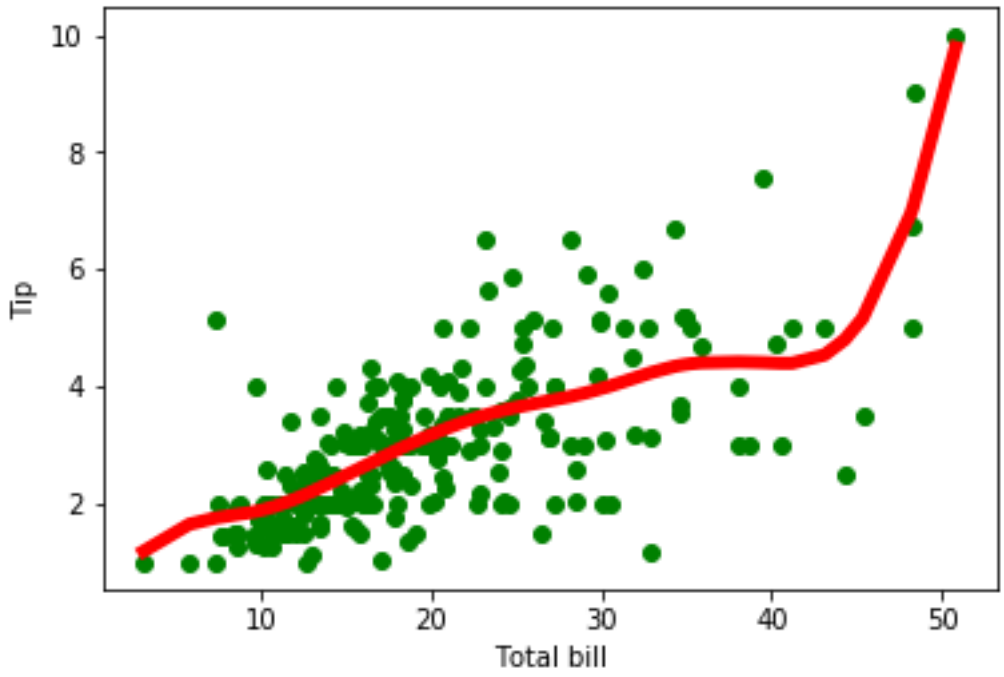
	precision	recall	f1-score	support
0	1.00	1.00	1.00	8
1	1.00	1.00	1.00	12
2	1.00	1.00	1.00	10
avg / total	1.00	1.00	1.00	30

9. **Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs**.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point,xmat, k):
      m,n = np.shape(xmat)
      weights = np.mat(np.eye((m)))
      for j in range(m):
            diff = point - X[j]
            weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
      return weights
def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
def localWeightRegression(xmat,ymat,k):
      m,n = np.shape(xmat)
      ypred = np.zeros(m)
      for i in range(m):
            ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
      return ypred
def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0)
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show()
data = pd.read_csv('data10.csv')
```

```
bill = np.array(data.total_bill)
tip = np.array(data.tip)
mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

Output:



Viva questions

1. What is machine learning?
   Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions.
2. Define supervised learning
   Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

   $Y = f(X)$

3. Define unsupervised learning
   Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

4. Define semi supervised learning
   Labeled data is used to help identify *that* there are specific groups of webpage types present in the data and what they *might be.* The algorithm is then trained on unlabeled data to define the boundaries of those webpage types and may even identify new types of web pages that were unspecified in the existing human-inputted labels.

5. Define reinforcement learning
   Reinforcement learning is an important type of Machine Learning where an agent learn how to behave in a environment by performing actions and seeing the results.

6. What do you mean by hypotheses
   A hypothesis is an educated prediction that can be tested. You will discover the purpose of a hypothesis then learn how one is developed and written.

7. What is classification
   classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

8. What is clustering
   Clustering: is the assignment of a set of observations into subsets (calledclusters) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning
9. Define precision, accuracy and recall
   precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Accuracy is

the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

10. Define entropy
    Entropy, as it relates to machine learning, is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.
11. Define regression
    Regression is basically a statistical approach to find the relationship between variables. In machine learning, this is used to predict the outcome of an event based on the relationship between variables obtained from the data-set.
12. How Knn is different from k-means clustering

    K-Means: it is an Unsupervised learning technique. It is used for Clustering. n training phase of K-Means, K observations are arbitrarily selected (known as centroids). Each point in the vector space is assigned to a cluster represented by nearest (euclidean distance) centroid. Once the clusters are formed, for each cluster the centroid is updated to the mean of all cluster members. And the cluster formation restarts with new centroids. This repeats until the centroids themselves become mean of clusters,

    KNN: It is a Supervised learning technique. It is used mostly forClassification, and sometimes even for Regression. K-NN doesn't have a training phase as such. But the prediction of a test observation is done based on the K-Nearest (often euclidean distance) Neighbours (observations) based on weighted averages/votes.

13. What is concept learning
    Concept learning also refers to a learning task in which a human or machine learner is trained to classify objects by being shown a set of example objects along with their class labels. The learner simplifies what has been observed by condensing it in the form of an example.
14. Define specific boundary and general boundary
    The general boundary G, with respect to hypothesis space H and training data D, is the set of maximally general hypotheses consistent with D.
    The specific boundary S, with respect to hypothesis space H and training data D, is the set of maximally specific hypotheses consistent with D
15. Define target function
    This is a function that knows and maps a full-relationship of the features/input variables to the Response/Output variable.
16. Define decision tree
    Decision Tree create a training model which can use to predict class or value of target variables by **learning decision rules** inferred from prior data(training data)
17. What is ANN
    An artificial neuron network (ANN) is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the ANN because a neural network changes - or learns, in a sense - based on that input and output.
18. Explain gradient descent approximation

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

19. State Bayes theorem

Bayes' Theorem is the fundamental result of probability theory – it puts the posterior probability P(H|D) of a hypothesis as a product of the probability of the data given the hypothesis(P(D|H)), multiplied by the probability of the hypothesis (P(H)), divided by the probability of seeing the data.

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

20. Define Bayesian belief networks

It is a probabilistic graphical model (a type of statistical model) that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG).

21. Differentiate hard and soft clustering

In hard clustering, each data point either belongs to a cluster completely or not. In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned.

22. Define variance

Variance is a measurement of the spread between numbers in a data set. The variance measures how far each number in the set is from the mean. Variance is calculated by taking the differences between each number in the set and the mean, squaring the differences (to make them positive) and dividing the sum of the squares by the number of elements in the set.

23. What is inductive machine learning

From the perspective of inductive learning, we are given input samples (x) and output samples (f(x)) and the problem is to estimate the function (f). Specifically, the problem is to generalize from the samples and the mapping to be useful to estimate the output for new samples in the future.

24. Define pruning

Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

25. Define Bias

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

26. What is Artificial Intelligence?

Artificial Intelligence is the field of computer science concerned with building intelligent machines or computer systems, capable of simulating human intelligence. The machines created using Artificial Intelligence can work and react like humans without human intervention. Speech Recognition, Customer service, Recommendation Engine, Natural Language Processing (NLP) are some of the applications of Artificial Intelligence.

27. What are some real-life applications of Artificial Intelligence?

- Social Media: The most common use of Artificial Intelligence in social media is facial detection and verification. Artificial Intelligence, along with machine learning, is also used to design your social media feed.
- Personalized online shopping: Shopping sites use AI-powered algorithms to curate the list of buying recommendations for users. They use data like users' search history and recent orders to create a list of suggestions that users might like.
- Agriculture: Technologies, especially Artificial Intelligence embedded systems, help farmers protect their crops from various adversities like weather, weeds, pests, and changing prices.
- Smart cars: Smart cars are another one of the real-life applications of AI. Artificial intelligence collects data from a car's radar, camera, and GPS to operate the vehicle when the autopilot mode is on.
- Healthcare: Artificial Intelligence has come out as a reliable friend of doctors. From intelligent testing to medical recommendations, they assist medical professionals in every possible way

28. What are different platforms for Artificial Intelligence (AI) development?

Some different software platforms for AI development are-

1. Amazon AI services
2. Tensorflow
3. Google AI services
4. Microsoft Azure AI platform
5. Infosys Nia
6. IBM Watson
7. H2O
8. Polyaxon
9. PredictionIO

29. What are the programming languages used for Artificial Intelligence?
    Python, LISP, Java, C++, R are some of the programming languages used for Artificial Intelligence.

30. What is the future of Artificial Intelligence?
    Artificial Intelligence has affected many humans and almost every industry, and it is expected to continue to do so. Artificial Intelligence has been the main driver of emerging technologies like the Internet of Things, big data, and robotics. AI can harness the power of a massive amount of data and make an optimal decision in a fraction of seconds, which is almost impossible for a normal human. AI is leading areas that are important for mankind such as cancer research, cutting-edge climate change technologies, smart cars, and space exploration. It has taken the center stage of innovation and development of computing, and it is not ceding the stage in the foreseeable future. Artificial Intelligence is going to impact the world more than anything in the history of mankind.

31. What is the difference between Artificial Intelligence and Machine learning?

Artificial Intelligence	Machine Learning
Artificial Intelligence is the ability of machines to function like the human brain.	Machine learning is processing data, learning from it, and then making informed decisions.
The goal of AI is to allow machines to think for themselves without the need for human involvement.	The purpose of machine learning is to allow a machine to learn from its previous experiences.
AI is capable of dealing with both structured and semi-structured data.	Machine learning works with both organized and semi-structured data.
AI is a subset of data science.	Machine Learning is a subset of AI.
Example- Google Search engine	Example- Image recognition

32. How are Artificial Intelligence and Machine Learning related?

    Artificial Intelligence and Machine Learning are two popular and often misunderstood words. Artificial Intelligence is a domain of computer science that enables machines to mimic human intelligence and behaviour. On the other hand, Machine Learning is a subset of Artificial Intelligence and is all about feeding computers with data so that they can learn on their own from all the patterns and models. Machine Learning models are used to implement Artificial Intelligence frequently.