

**Artificial Intelligence**

**DA – 1**

Member 1 : Ramireddy Jeevan Reddy (21BCE5163)

Member 2 : Venu Vihari Yadav (21BCE5232)

1. **Decision Tree Algorithm**

**Basic Definition :**

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision Trees (DTs) are a non-parametric supervised learning method used for both classification and regression. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules, and the fitter the model.

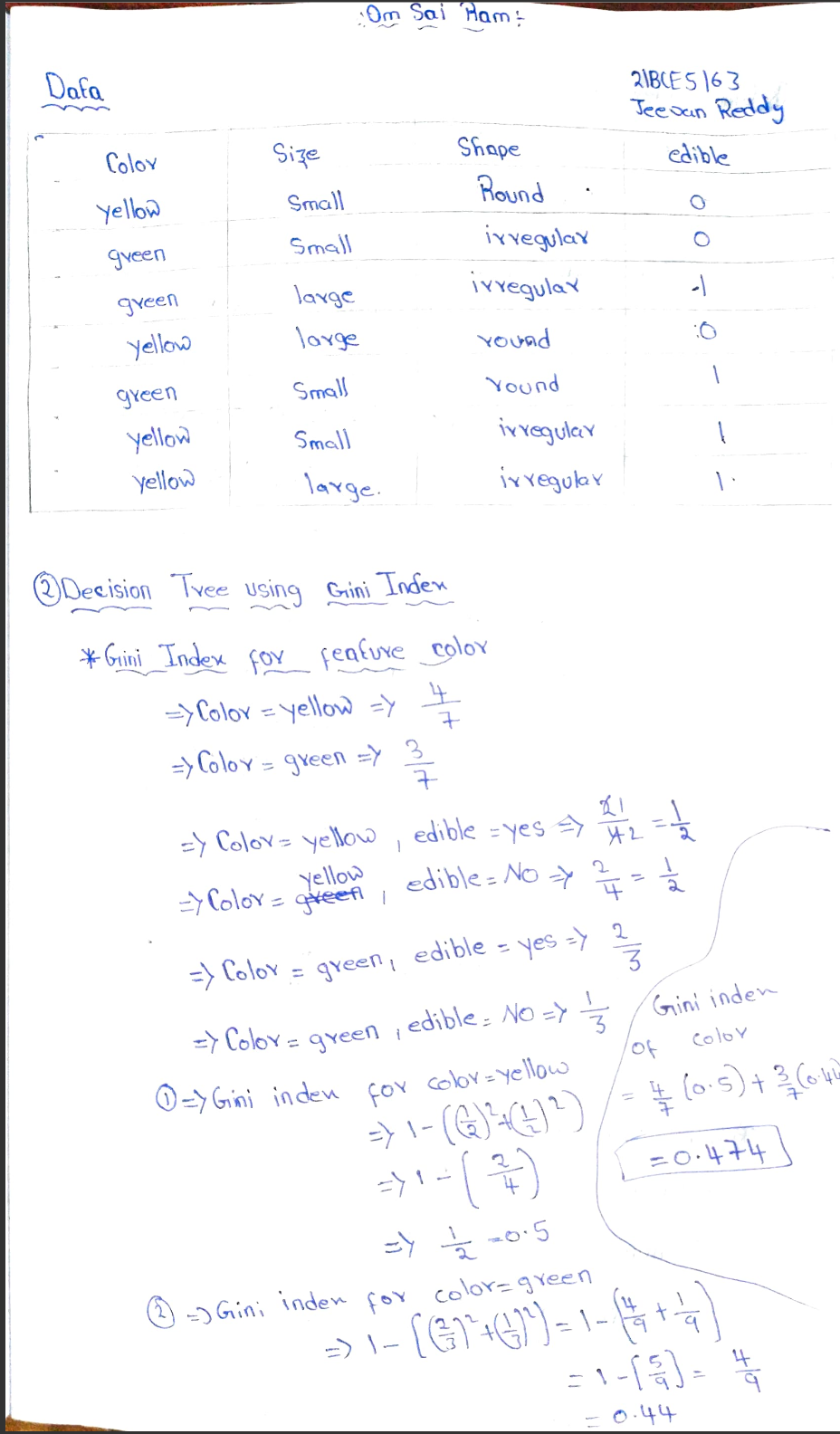
**Uses :**

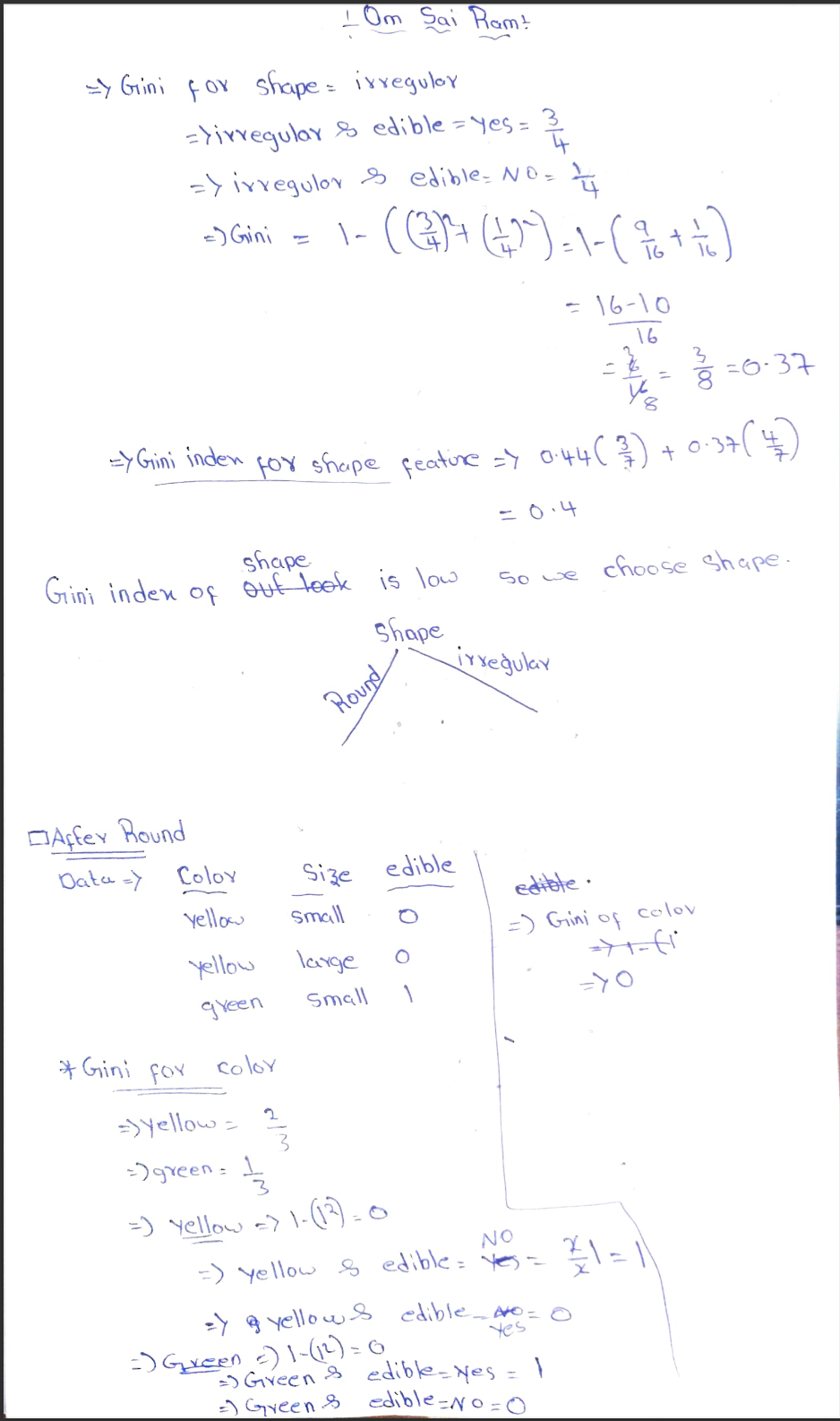
1. When the user has an objective he is trying to achieve: max profit, optimize cost
2. When there are several courses of action
3. There is a calculated measure of the benefit of the various alternatives
4. When there are events beyond the control of the decision-maker i.e environment factor.
5. Uncertainty concerning which outcome will actually happen

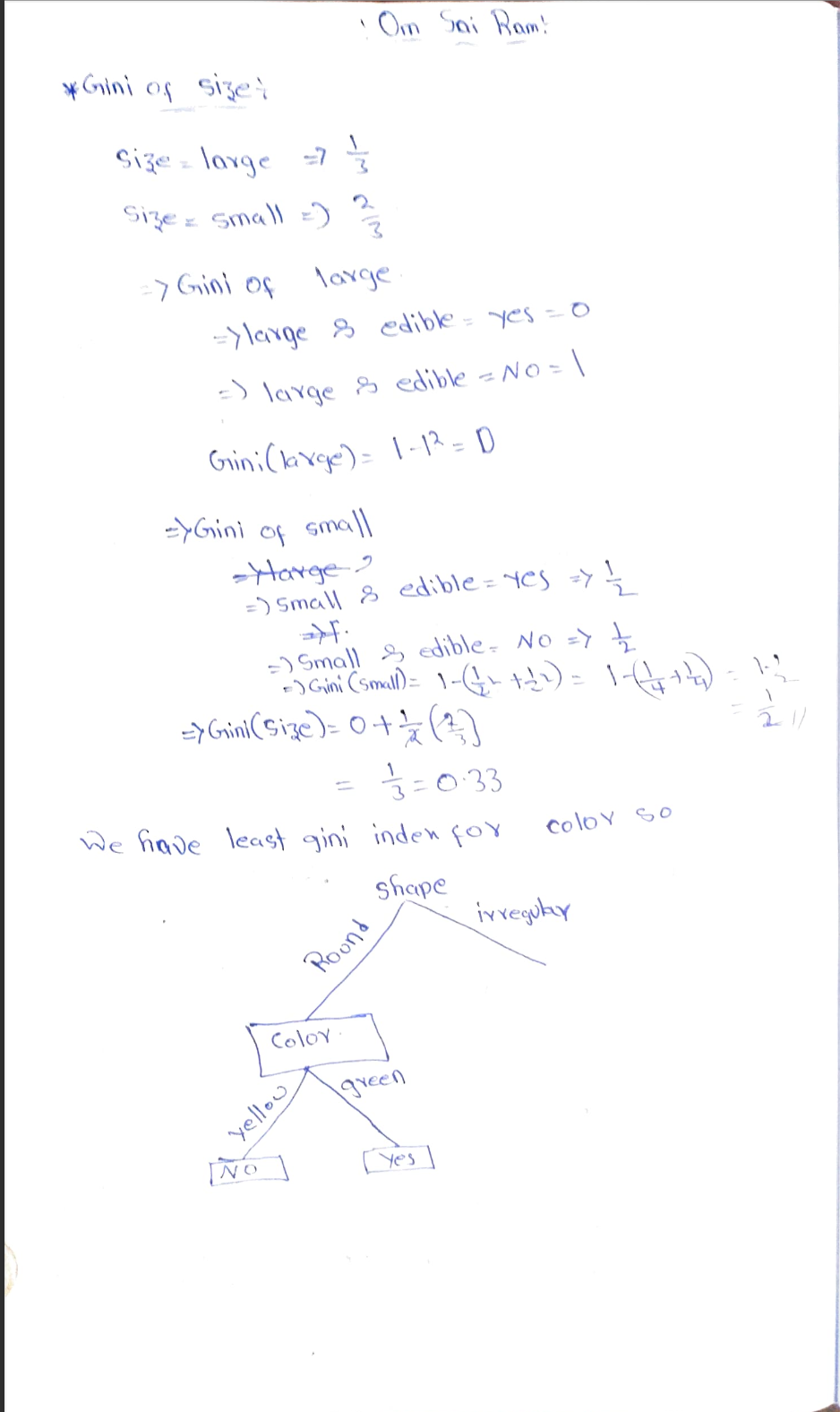
**Important Terms :**

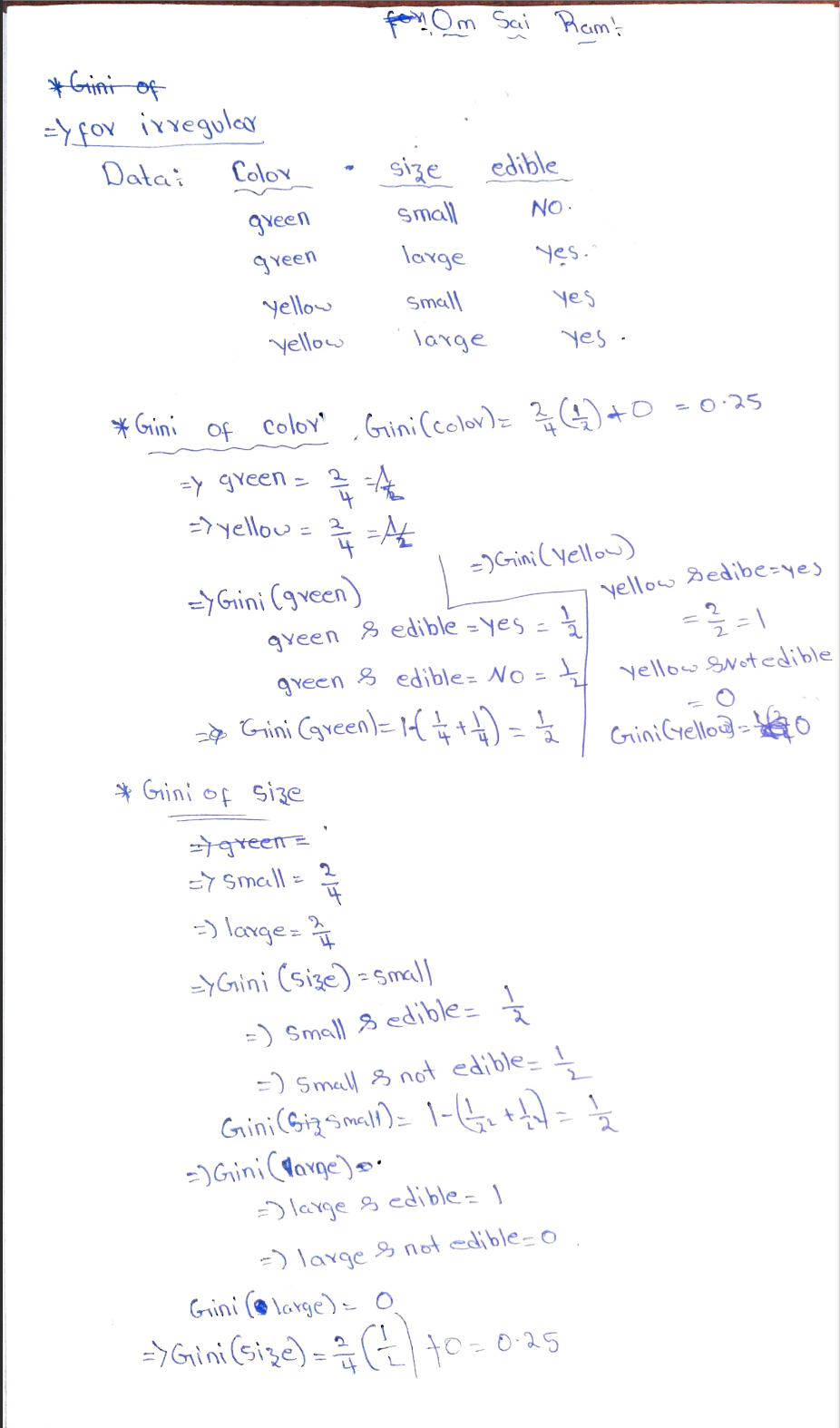
1. **Root Node**- It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting**-It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node**-When a sub-node splits into further sub-nodes, then it is called a decision node.
4. **Leaf/ Terminal Node**-Nodes do not split is called Leaf or Terminal node.
5. **Branch / Sub-Tree** - A subsection of the entire tree is called branch or sub-tree.
6. **Parent and Child Node** - A node, which is divided into sub-nodes is called the parent node of sub-nodes whereas sub-nodes are the child of the parent node.
7. **Information Gain**-The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding an attribute that returns the highest information gain (i.e., the most homogeneous branches).
8. **Internal Node**-An internal node (also known as an inner node, inode for short, or branch node) is any node of a tree that has child nodes

**Mathematical Insights :**

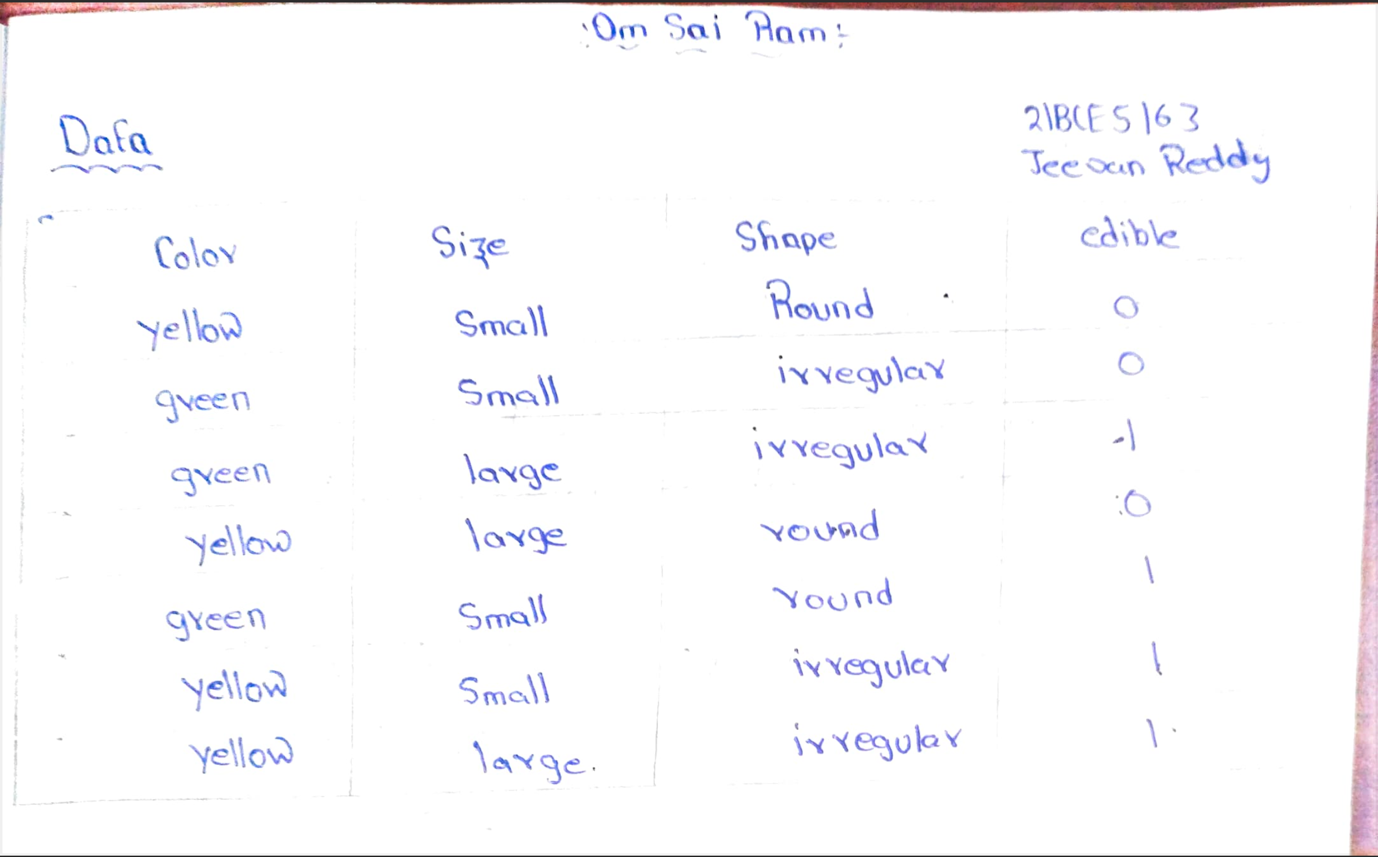
****

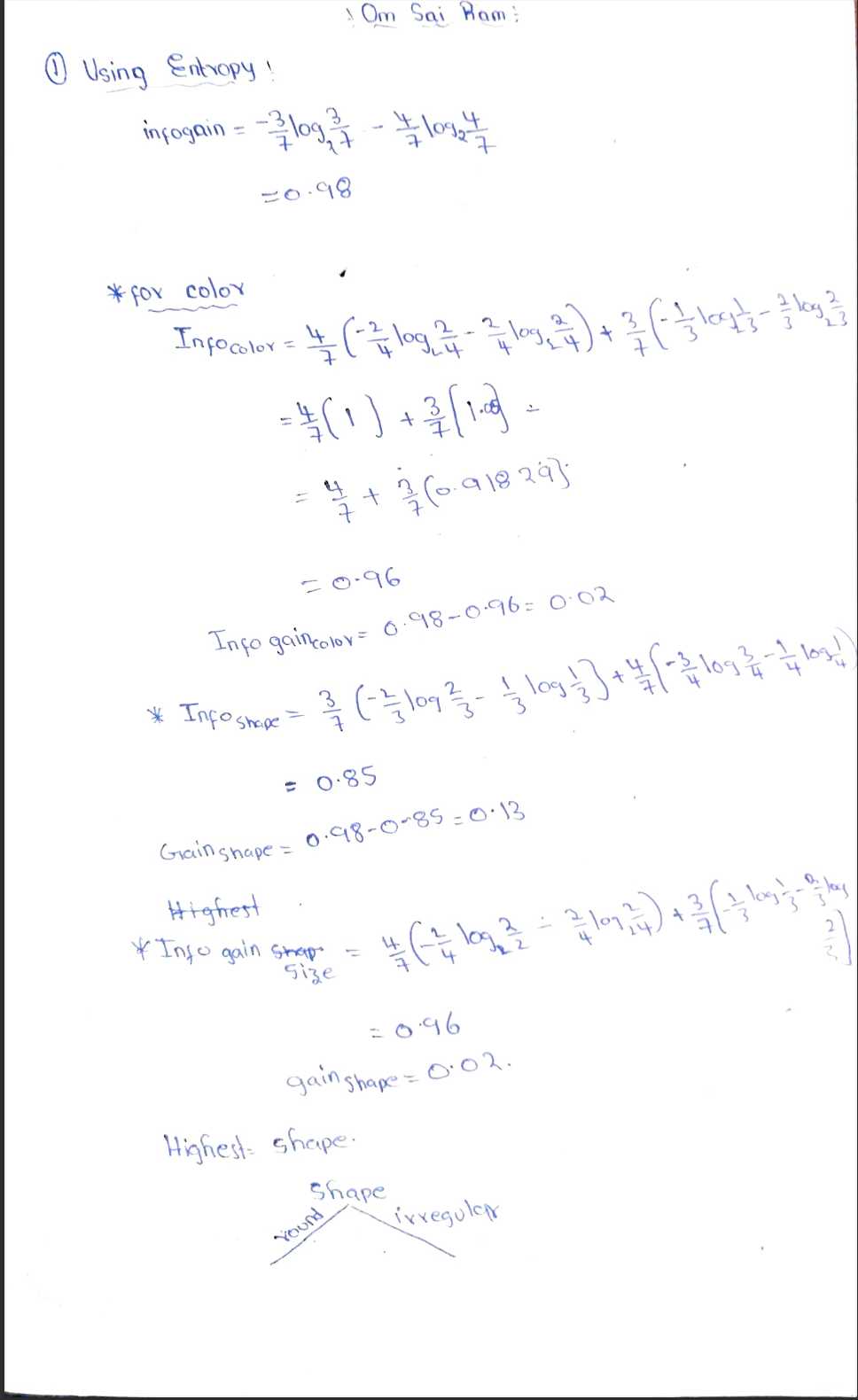
****

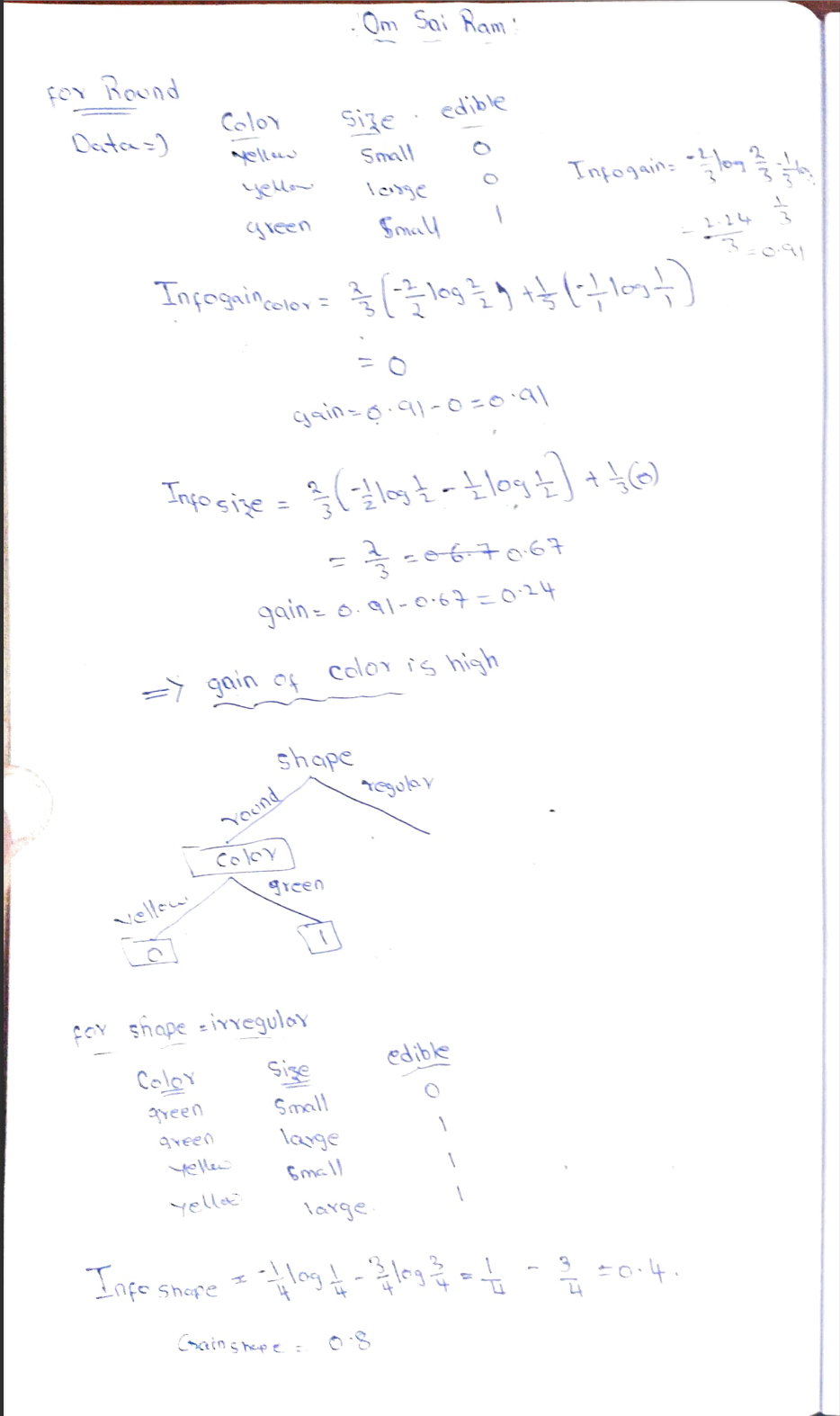
****

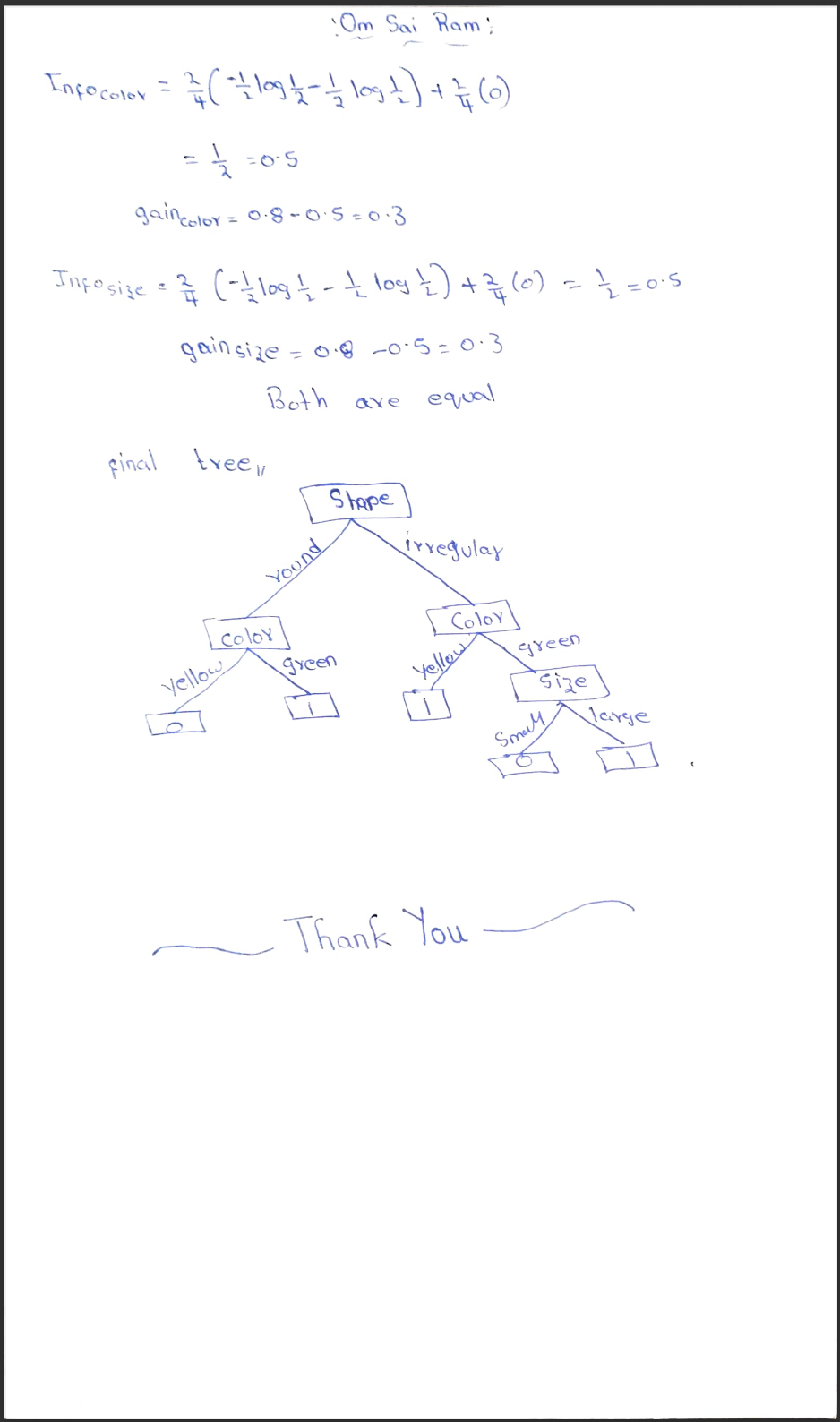
****

****

****

****

****

****

**Algorithm :**

import pandas as pd

import numpy as np

from pprint import pprint

#Import the dataset and define the feature as well as the target datasets / columns#

dataset = pd.read\_csv('zoo.csv',

names=['animal\_name','hair','feathers','eggs','milk',

'airbone','aquatic','predator','toothed','backbone',

'breathes','venomous','fins','legs','tail','domestic','catsize','class',])#Import all columns omitting the fist which consists the names of the animals

#We drop the animal names since this is not a good feature to split the data on

dataset=dataset.drop('animal\_name',axis=1)

def entropy(target\_col):

elements,counts = np.unique(target\_col,return\_counts = True)

entropy = np.sum([(-counts[i]/np.sum(counts))\*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])

return entropy

def InfoGain(data,split\_attribute\_name,target\_name="class"):

#Calculate the entropy of the total dataset

total\_entropy = entropy(data[target\_name])

##Calculate the entropy of the dataset

#Calculate the values and the corresponding counts for the split attribute

vals,counts= np.unique(data[split\_attribute\_name],return\_counts=True)

#Calculate the weighted entropy

Weighted\_Entropy = np.sum([(counts[i]/np.sum(counts))\*entropy(data.where(data[split\_attribute\_name]==vals[i]).dropna()[target\_name]) for i in range(len(vals))])

#Calculate the information gain

Information\_Gain = total\_entropy - Weighted\_Entropy

return Information\_Gain

def ID3(data,originaldata,features,target\_attribute\_name="class",parent\_node\_class = None):

#Define the stopping criteria --> If one of this is satisfied, we want to return a leaf node#

#If all target\_values have the same value, return this value

if len(np.unique(data[target\_attribute\_name])) <= 1:

return np.unique(data[target\_attribute\_name])[0]

#If the dataset is empty, return the mode target feature value in the original dataset

elif len(data)==0:

return np.unique(originaldata[target\_attribute\_name])[np.argmax(np.unique(originaldata[target\_attribute\_name],return\_counts=True)[1])]

#If the feature space is empty, return the mode target feature value of the direct parent node --> Note that

#the direct parent node is that node which has called the current run of the ID3 algorithm and hence

#the mode target feature value is stored in the parent\_node\_class variable.

elif len(features) ==0:

return parent\_node\_class

#If none of the above holds true, grow the tree!

else:

#Set the default value for this node --> The mode target feature value of the current node

parent\_node\_class = np.unique(data[target\_attribute\_name])[np.argmax(np.unique(data[target\_attribute\_name],return\_counts=True)[1])]

#Select the feature which best splits the dataset

item\_values = [InfoGain(data,feature,target\_attribute\_name) for feature in features] #Return the information gain values for the features in the dataset

best\_feature\_index = np.argmax(item\_values)

best\_feature = features[best\_feature\_index]

#Create the tree structure. The root gets the name of the feature (best\_feature) with the maximum information

#gain in the first run

tree = {best\_feature:{}}

#Remove the feature with the best inforamtion gain from the feature space

features = [i for i in features if i != best\_feature]

#Grow a branch under the root node for each possible value of the root node feature

for value in np.unique(data[best\_feature]):

value = value

#Split the dataset along the value of the feature with the largest information gain and therwith create sub\_datasets

sub\_data = data.where(data[best\_feature] == value).dropna()

#Call the ID3 algorithm for each of those sub\_datasets with the new parameters --> Here the recursion comes in!

subtree = ID3(sub\_data,dataset,features,target\_attribute\_name,parent\_node\_class)

#Add the sub tree, grown from the sub\_dataset to the tree under the root node

tree[best\_feature][value] = subtree

return(tree)

def predict(query,tree,default = 1):

for key in list(query.keys()):

if key in list(tree.keys()):

try:

result = tree[key][query[key]]

except:

return default

result = tree[key][query[key]]

if isinstance(result,dict):

return predict(query,result)

else:

return result

def train\_test\_split(dataset):

training\_data = dataset.iloc[:80].reset\_index(drop=True)#We drop the index respectively relabel the index

#starting form 0, because we do not want to run into errors regarding the row labels / indexes

testing\_data = dataset.iloc[80:].reset\_index(drop=True)

return training\_data,testing\_data

training\_data = train\_test\_split(dataset)[0]

testing\_data = train\_test\_split(dataset)[1]

def test(data,tree):

#Create new query instances by simply removing the target feature column from the original dataset and

#convert it to a dictionary

queries = data.iloc[:,:-1].to\_dict(orient = "records")

#Create a empty DataFrame in whose columns the prediction of the tree are stored

predicted = pd.DataFrame(columns=["predicted"])

#Calculate the prediction accuracy

for i in range(len(data)):

predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)

print('The prediction accuracy is: ',(np.sum(predicted["predicted"] == data["class"])/len(data))\*100,'%')

tree = ID3(training\_data,training\_data,training\_data.columns[:-1])

pprint(tree)

test(testing\_data,tree)

**Advantage/Features of Decision Tree**

1. Decision trees require less effort for data preparation as compared to other algorithms during pre-processing.
2. Do not require normalization of data.
3. Do not require scaling of data as well.
4. Missing values in the data also do NOT affect the process of building a decision tree to any considerable extent.
5. A Decision tree model is very intuitive and easy to explain to technical teams as well as stakeholders.

**Disadvantages/Shortcomings of Decision Tree**

1. A small change in the data can cause a large change in the structure of the decision tree causing instability.
2. For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
3. It often involves a higher time to train the model.
4. Its training is relatively expensive because the complexity and time taken are more.
5. Decision Tree algorithm is inadequate for applying regression and predicting continuous values.