1)a)demonstrate the FIND-S algorithm for finding the most specifichypothesis based on a given set of training data samples. Read the training data from a.CSV file and show the output for test cases.

```
import numpy as np
import pandas as pd
data = pd.read_csv('finds.csv')
def train(concepts, target):
    for i,val in enumerate(target):
        if val == "Yes":
            specific_h = concepts[i]
        break

    for i,h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] == specific_h[x]:
                    pass
            else:
                specific_h[x] = "?"
                return specific_h
#slicing rows and column, : means begining to end of row
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
print(train(concepts,target))
```

```
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' '?']
```

1)b)Develop an interactive program byCompareing the result by implementing LIST THEN ELIMINATE algorithm

```
import numpy as np
import pandas as pd
data = pd.read_csv('finds.csv')

def list_then_eliminate(concepts, target):
    positive_examples = concepts[target == 'Yes']
    specific_hypothesis = positive_examples[0].copy()

    for example in positive_examples[1:]:
        for i, attribute in enumerate(example):
            if attribute != specific_hypothesis[i]:
                specific_hypothesis[i] = '?'

                return specific_hypothesis
```

```
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])

print('Specific hypothesis obtained by LIST THEN ELIMINATE
algorithm:')
print(list_then_eliminate(concepts, target))
print('\nSpecific hypothesis obtained by the original code:')
print(train(concepts, target))
```

```
Specific hypothesis obtained by LIST THEN ELIMINATE algorithm:
['?' 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Specific hypothesis obtained by the original code:
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' '?']
```

2)For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm. Output a description of the set of all hypotheses consistent with the training examples

```
import numpy as np
import pandas as pd
#Loading Data from CSV file
data=pd.DataFrame(data=pd.read_csv("finds.csv"))
#Seperating concept features from Target
concepts=np.array(data.iloc[:,0:-1])
#Isolating target into seperate Dataframe
target=np.array(data.iloc[:,-1])
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))]
    #the learning iterations
    for i, h in enumerate(concepts):
        #checking if the hypothesys has a positive target
        if target[i]=="Yes":
            for x in range(len(specific_h)):
            #Change values in S&G only if values change
                if h[x]!=specific_h[x]:
                    specific_h[x]='?'
                    general_h[x][x]='?'
                #Checking if the hypothesys has a positive target
                if target[i]=="No":
                    for x in range (len(specific_h)):
                        #for negative hypothesys change the value only
in G

                        if h[x]!=specific_h[x]:
                            general_h[x][x]='?'
                            #find indices where we have empty row,
meaning those that are unchanged
                            indices=[i for i ,val in
enumerate(general_h)if val==['?','?','?','?','?','?']]
                            for i in indices:
                                #remove those rows form general_h

general_h.remove(['?','?','?','?','?','?'])
                                #return final values
        return specific_h,general_h


s_final, g_final=learn(concepts,target)
print("Final S:",s_final,sep="\n")
print("Final G:",g_final,sep="\n")
data.head()
```

```
Final S:
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Final G:
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]

   Eg    Sky Temperature Humidity    Wind Water Forecast Enjoyable
0   1  Sunny        Warm   Normal  Strong  Warm     Same       Yes
1   2  Sunny        Warm     High  Strong  Warm     Same       Yes
2   3  Rainy        Cold     High  Strong  Warm     Same       Yes
3   4  Sunny        Warm     High  Strong  Warm     Same       Yes
```

3)Demonstrate Pre processing (Data Cleaning, Integration and Transformation) activity on suitable data: For example: Identify and Delete Rows that Contain Duplicate Data by considering an appropriate dataset. Identify and Delete Columns That Contain a Single Value by considering an appropriate dataset.

```
import pandas as pd

# Sample dataset 1: contains duplicate rows
data1 = {'A': [1, 2, 2, 3, 4, 5, 5],
         'B': [5, 6, 6, 7, 8, 9, 9]}

# Sample dataset 2: contains a column with a single value
data2 = {'C': [10, 10, 10, 10, 10],
         'D': [11, 12, 13, 14, 15]}
```

```
# Creating dataframes from the datasets
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Identifying and deleting duplicate rows in df1
print("Before removing duplicates in dataset 1:")
print(df1)
df1.drop_duplicates(inplace=True)
print("\nAfter removing duplicates in dataset 1:")
print(df1)

# Identifying and deleting columns with a single value in df2
print("\nBefore removing single value columns in dataset 2:")
print(df2)
cols_to_remove = [col for col in df2.columns if df2[col].nunique() <=
1]
df2.drop(cols_to_remove, axis=1, inplace=True)
print("\nAfter removing single value columns in dataset 2:")
print(df2)
```

```
Before removing duplicates in dataset 1:
   A  B
0  1  5
1  2  6
2  2  6
3  3  7
4  4  8
5  5  9
6  5  9

After removing duplicates in dataset 1:
   A  B
0  1  5
1  2  6
3  3  7
4  4  8
5  5  9

Before removing single value columns in dataset 2:
    C   D
0  10  11
1  10  12
2  10  13
3  10  14
4  10  15

After removing single value columns in dataset 2:
    D
0  11
1  12
```

```
2  13
3  14
4  15
```

4)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 csv
import math
import random

#Function tells which class has more entries in given data-set
def majorClass(attributes, data, target):
    freq = {}
    index = attributes.index(target)
    for tuple in data:
        if tuple[index] in freq:
            freq[tuple[index]] += 1
        else:
            freq[tuple[index]] = 1
    max = 0
    major = ""
    for key in freq.keys():
        if freq[key]>max:
            max = freq[key]
            major = key
    return major

# Calculates the entropy of the data given the target attribute
def entropy(attributes, data, targetAttr):
    freq = {}
    dataEntropy = 0.0
    i = 0
    for entry in attributes:
        if (targetAttr == entry):
            break
        i = i + 1

    i = i - 1
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]] = 1.0
    for freq in freq.values():
        dataEntropy += (-freq/len(data)) * math.log(freq/len(data), 2)
    return dataEntropy

# Calculates the information gain (reduction in entropy) in the data
when a particular attribute is chosen for splitting the data.
def info_gain(attributes, data, attr, targetAttr):
    freq = {}
    subsetEntropy = 0.0
    i = attributes.index(attr)
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]] = 1.0

    for val in freq.keys():
        valProb = freq[val] / sum(freq.values())
        dataSubset = [entry for entry in data if entry[i] == val]
        subsetEntropy += valProb * entropy(attributes, dataSubset,
targetAttr)

    return (entropy(attributes, data, targetAttr) - subsetEntropy)

# This function chooses the attribute among the remaining attributes
which has the maximum information gain.
def attr_choose(data, attributes, target):
    best = attributes[0]
    maxGain = 0;
    for attr in attributes:
        newGain = info_gain(attributes, data, attr, target)
        if newGain>maxGain:
            maxGain = newGain
            best = attr

    return best

# This function will get unique values for that particular attribute
from the given data
def get_values(data, attributes, attr):
    index = attributes.index(attr)
    values = []
    for entry in data:
        if entry[index] not in values:
            values.append(entry[index])

    return values

# This function will get all the rows of the data where the chosen
"best" attribute has a value "val"
def get_data(data, attributes, best, val):
    new_data = [[]]
    index = attributes.index(best)
    for entry in data:
        if (entry[index] == val):
            newEntry = []
            for i in range(0,len(entry)):
                if(i != index):
                    newEntry.append(entry[i])
            new_data.append(newEntry)

    new_data.remove([])
    return new_data

def build_tree(data, attributes, target):
    data = data[:]
    vals = [record[attributes.index(target)] for record in data]
    default = majorClass(attributes, data, target)
    if not data or (len(attributes) - 1) <= 0:
        return default
    elif vals.count(vals[0]) == len(vals):
        return vals[0]
    else:
        best = attr_choose(data, attributes, target)
        tree = {best: {}}
        for val in get_values(data, attributes, best):
            new_data = get_data(data, attributes, best, val)
            newAttr = attributes[:]
            newAttr.remove(best)
            subtree = build_tree(new_data, newAttr, target)
            tree[best][val] = subtree
        return tree


def execute_decision_tree():
    data = []
    #load file
    with open("weather.csv") as tsv:
        for line in csv.reader(tsv):
            data.append(tuple(line))
    print("Number of records:", len(data))

    #set attributes
    attributes = ['outlook', 'temperature', 'humidity', 'wind',
'play']
    target = attributes[-1]

    #set training data
    acc = []
    training_set = [x for i, x in enumerate(data)]
    tree = build_tree(training_set, attributes, target)
    print(tree)

    #execute algorithm on test data
    results = []
    test_set = [('rainy', 'mild', 'high', 'strong')]
    for entry in test_set:
        tempDict = tree.copy()
        result = ""
        while isinstance(tempDict, dict):
            child = []
            nodeVal = next(iter(tempDict))
            child = tempDict[next(iter(tempDict))].keys()
            tempDict = tempDict[next(iter(tempDict))]
            index = attributes.index(nodeVal)
            value = entry[index]

            if value in tempDict.keys():
                result = tempDict[value]
                tempDict = tempDict[value]
            else:
                result = "Null"
                break

        if result != "Null":
            results.append(result == entry[-1])

        print(result)


if __name__ == "__main__":
    execute_decision_tree()
```

```
Number of records: 15
{'outlook': {'id': 'wind', '1': 'weak', '2': 'strong', '3': 'weak',
'4': 'weak', '5': 'weak', '6': 'weak', '7': 'strong', '8': 'weak',
'9': 'weak', '10': 'weak', '11': 'strong', '12': 'strong', '13':
'weak', '14rainy': 'no'}}
Null
```

5) Demonstrate the working of the Random forest algorithm. Use an appropriate data set for building and apply this knowledge toclassify a new sample.

```
import csv
import random
import math

def loadCsv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]

def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if vector[-1] not in separated:
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):
    return sum(numbers) / float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x - avg) ** 2 for x in numbers]) /
float(len(numbers) - 1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries

def calculateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev,
2))))
```

```python
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent

def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean,
stdev)
    return probabilities

def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries,
inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(testSet))) * 100.0

def main():
    filename = 'pima-indians-diabetes.csv'
    splitRatio = 0.67
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingSet), len(testSet)))
    summaries = summarizeByClass(trainingSet)
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: {0}%'.format(accuracy))
```

```
main()
```

```
Split 768 rows into train=514 and test=254 rows
Accuracy: 75.98425196850394%
```

6)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.