Q1

Data-set Description :
Title : Balance Scale Weight & Distance Database
Number of Instances : 625 (49 balanced, 288 left, 288 right)
Number of Attributes : 4 (numeric) + class name = 5
Attribute Information:

1. Class Name (Target variable): 3
    1. L [balance scale tip to the left]
    2. B [balance scale be balanced]
    3. R [balance scale tip to the right]
2. Left-Weight: 5 (1, 2, 3, 4, 5)
3. Left-Distance: 5 (1, 2, 3, 4, 5)
4. Right-Weight: 5 (1, 2, 3, 4, 5)
5. Right-Distance: 5 (1, 2, 3, 4, 5)

Missing Attribute Values: None

6. Class Distribution:
    1. 46.08 percent are L
    2. 07.84 percent are B
    3. 46.08 percent are R

Code Snippet:
Run this program on your local python
interpreter, provided you have installed
the required libraries.

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
	balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
	sep= ',', header = None)

	# Printing the dataswet shape
	print ("Dataset Length: ", len(balance_data))
	print ("Dataset Shape: ", balance_data.shape)

	# Printing the dataset obseravtions
	print ("Dataset: ",balance_data.head())
	return balance_data

# Function to split the dataset
def splitdataset(balance_data):

	# Separating the target variable
	X = balance_data.values[:, 1:5]
	Y = balance_data.values[:, 0]

	# Splitting the dataset into train and test
	X_train, X_test, y_train, y_test = train_test_split(
	X, Y, test_size = 0.3, random_state = 100)

	return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

	# Creating the classifier object
	clf_gini = DecisionTreeClassifier(criterion = "gini",
			random_state = 100,max_depth=3, min_samples_leaf=5)
```

```
        # Performing training
        clf_gini.fit(X_train, y_train)
        return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

        # Decision tree with entropy
        clf_entropy = DecisionTreeClassifier(
                        criterion = "entropy", random_state = 100,
                        max_depth = 3, min_samples_leaf = 5)

        # Performing training
        clf_entropy.fit(X_train, y_train)
        return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

        # Predicton on test with giniIndex
        y_pred = clf_object.predict(X_test)
        print("Predicted values:")
        print(y_pred)
        return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

        print("Confusion Matrix: ",
                confusion_matrix(y_test, y_pred))

        print ("Accuracy : ",
        accuracy_score(y_test,y_pred)*100)

        print("Report : ",
        classification_report(y_test, y_pred))

# Driver code
def main():

        # Building Phase
        data = importdata()
        X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
```

```
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
    main()
```

Data Information:

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

Results Using Gini Index:

```
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
```

'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]
[0 67 18]
[0 19 71]]
Accuracy : 73.4042553191
Report :

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.73	0.79	0.76	85
R	0.74	0.79	0.76	90
avg/total	0.68	0.73	0.71	188

Results Using Entropy:

Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]
[0 63 22]
[0 20 70]]
Accuracy : 70.7446808511
Report :

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.71	0.74	0.72	85
R	0.71	0.78	0.74	90
avg / total	0.66	0.71	0.68	188

Q2.

Dataset:
Income,Credit Score,Employment Status,Loan Amount,Loan Approval
45,680,Employed,35,Yes
30,620,Unemployed,20,No
80,750,Self-Employed,50,Yes
25,590,Employed,15,No
60,710,Unemployed,45,Yes
35,640,Employed,30,No
70,720,Self-Employed,60,Yes

Code Snippet:

```
# Python code to build a decision tree for credit risk assessment

# Import necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.tree import export_text

# Load the dataset
data = pd.read_csv("credit_data.csv")  # Load your dataset here

# Prepare the data
X = data[['Income', 'Credit Score', 'Employment Status', 'Loan Amount']]
y = data['Loan Approval']

# Encode categorical features
X = pd.get_dummies(X, columns=['Employment Status'], drop_first=True)

# Split the data 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)

# Create a decision tree classifier
clf = DecisionTreeClassifier()
```

```
# Train the model
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the decision tree as text
tree_text = export_text(clf, feature_names=list(X.columns))
print(tree_text)

# Output the model's accuracy and confusion matrix
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)
```

Output:

```
            Income <= 42.5
            /         \
   Credit Score <= 645.0    Loan Amount <= 17.5
   /         \              /         \
Reject    Approve    Reject    Approve

Accuracy: 0.5
Confusion Matrix:
[[1 1]
 [1 1]]
```

Q4.Write a program for text summarization using python.

Code Snippet:

```
import gensim
from gensim.summarization import summarize
from gensim.summarization import keywords

# Random input text for summarization
text = """
Natural language processing (NLP) is a subfield of artificial intelligence (AI) that focuses
on the interaction between computers and humans through natural language. It enables
computers to understand, interpret, and generate human language in a way that is both
valuable and meaningful.

NLP has a wide range of applications, including machine translation, chatbots,
sentiment analysis, and text summarization. Summarization, in particular, is the process
of reducing a text to its core content, preserving its main ideas and key points. It's a
valuable tool for condensing lengthy documents or articles.

In this Python program, we'll use the Gensim library for extractive text summarization.
Gensim provides functions to summarize a text and extract keywords. You can
customize the length of the summary and the number of keywords extracted to suit your
needs.

Keep in mind that the quality of the summary depends on the input text and the chosen
parameters. More advanced summarization techniques can be explored using deep
learning models and other libraries like NLTK and spaCy.

Feel free to replace the random input text with your own content and use this program
to generate extractive summaries and keywords.
"""

# Extractive Summarization
summary = summarize(text, ratio=0.3)  # Adjust the ratio as needed for the desired
summary length.

print("Extractive Summary:")
print(summary)
```

```
# Extract Keywords (optional)
key_words = keywords(text, ratio=0.1)  # Adjust the ratio as needed for the number of keywords.
print("\nKeywords:")
print(key_words)
```

Output:

Extractive Summary:
Natural language processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language. It enables computers to understand, interpret, and generate human language in a way that is both valuable and meaningful.

NLP has a wide range of applications, including machine translation, chatbots, sentiment analysis, and text summarization. Summarization, in particular, is the process of reducing a text to its core content, preserving its main ideas and key points. It's a valuable tool for condensing lengthy documents or articles.

Feel free to replace the random input text with your own content and use this program to generate extractive summaries and keywords.

Keywords:
text
NLP
AI
applications
machine translation
chatbots
sentiment analysis
condensing
documents
articles