```
import pandas as pd
import numpy as np
from sklearn import preprocessing as prepro

dataframe_project=pd.read_csv(r'/content/MAlayalam_char_glcm_features.csv')
dataframe_project
```

	dissimilarity_0	dissimilarity_45	dissimilarity_90	dissimilarity_135	correlation_0	correlation_45	correlation_90	correlation_
0	17.216435	21.119725	20.889887	21.543330	0.491436	0.390284	0.389736	0.378
1	12.143708	15.748918	14.495192	15.859307	0.504656	0.375442	0.414030	0.371
2	17.216435	21.119725	20.889887	21.543330	0.491436	0.390284	0.389736	0.378
3	12.019049	15.824242	14.639564	16.002597	0.536540	0.396562	0.428353	0.388
4	13.591560	16.543238	12.819549	14.969519	0.398153	0.288259	0.437464	0.355
...	...	...	...	...	...	...	...	
3282	11.751418	12.237659	7.450290	10.809173	0.267380	0.257278	0.559467	0.352
3283	13.098692	14.121247	11.730000	13.799184	0.464952	0.434585	0.521399	0.447
3284	11.884975	11.776303	6.736288	11.370043	0.313443	0.339262	0.657564	0.361
3285	13.969495	15.416160	11.979269	15.652676	0.557603	0.517844	0.628131	0.510
3286	13.866753	16.438666	13.206963	16.520985	0.596966	0.506300	0.604121	0.502

3287 rows × 25 columns

```
# Creating a table of the the new dataset
table = {
    'age': ['<=30', '<=30', '31...40', '>40', '>40', '>40', '31...40', '<=30', '<=30', '>40', '<=30', '31...40', '31...40', '>40'],
    'income': ['high', 'high', 'high', 'medium', 'low', 'low', 'low', 'medium', 'low', 'medium', 'medium', 'medium', 'high', 'medium'],
    'student': ['no', 'no', 'no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no'],
    'credit_rating': ['fair', 'excellent', 'fair', 'fair', 'fair', 'excellent', 'excellent', 'fair', 'fair', 'fair', 'excellent', 'excellent'
    'buys_computer': ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no', 'yes', 'yes', 'yes', 'yes', 'yes', 'no']
}
dataframe_new = pd.DataFrame(table) # Uploading the new dataframe
dataframe_new
```

	age	income	student	credit_rating	buys_computer
0	<=30	high	no	fair	no
1	<=30	high	no	excellent	no
2	31...40	high	no	fair	yes
3	>40	medium	no	fair	yes
4	>40	low	yes	fair	yes
5	>40	low	yes	excellent	no
6	31...40	low	yes	excellent	yes
7	<=30	medium	no	fair	no
8	<=30	low	yes	fair	yes
9	>40	medium	yes	fair	yes
10	<=30	medium	yes	excellent	yes
11	31...40	medium	no	excellent	yes
12	31...40	high	yes	fair	yes
13	>40	medium	no	excellent	no

```
# Converting all categorical values into numerical values
from sklearn import preprocessing as prep

label_encoder = prep.LabelEncoder()
```

```
for column in dataframe_new.columns:
    dataframe_new[column] = label_encoder.fit_transform(dataframe_new[column])
dataframe_new
```

	age	income	student	credit_rating	buys_computer
0	1	0	0	1	0
1	1	0	0	0	0
2	0	0	0	1	1
3	2	2	0	1	1
4	2	1	1	1	1
5	2	1	1	0	0
6	0	1	1	0	1
7	1	2	0	1	0
8	1	1	1	1	1
9	2	2	1	1	1
10	1	2	1	0	1
11	0	2	0	0	1
12	0	0	1	1	1
13	2	2	0	0	0

```
def entropyBeforeSplit(data_field):
    class_label = data_field.unique()
    entropy = 0
    total_instances = len(data_field)
    for label in class_label:
        probability = len(data_field[data_field == label]) / total_instances
        entropy -= probability * np.log2(probability)
    return entropy
```

```
def entropyAfterSplit(data_field, target):
    total_instances = len(data_field)
    weighted_entropy_after_split = 0
    for value in data_field.unique():
        positive_indices = data_field[data_field == value].index
        subset_target = target[positive_indices]
        positive_instances = len(positive_indices)
        weighted_entropy_after_split += (positive_instances/total_instances) * entropyBeforeSplit(subset_target)
    return weighted_entropy_after_split
```

```
# Finding the attribute that can be used for making a decision tree
def information_gain(data_field, target):
    entropy_before_split = entropyBeforeSplit(target)
    entropy_after_split = entropyAfterSplit(data_field, target)
    information_gain_value = entropy_before_split - entropy_after_split
    return information_gain_value
```

```
# Calculate the information gain of all the features
features = input_data.columns
information_gain_values = {}
for feature in features:
    information_gain_values[feature] = information_gain(input_data[feature], output_data)
    print("The information gain for ", feature, " is: ", information_gain_values[feature], "\n")
```

```
The information gain for  age  is:  0.24674981977443933

The information gain for  income  is:  0.02922256565895487

The information gain for  student  is:  0.15183550136234159

The information gain for  credit_rating  is:  0.04812703040826949
```

```
# Finding the feature with the highest information gain
root_node = max(information_gain_values, key = information_gain_values.get)
```

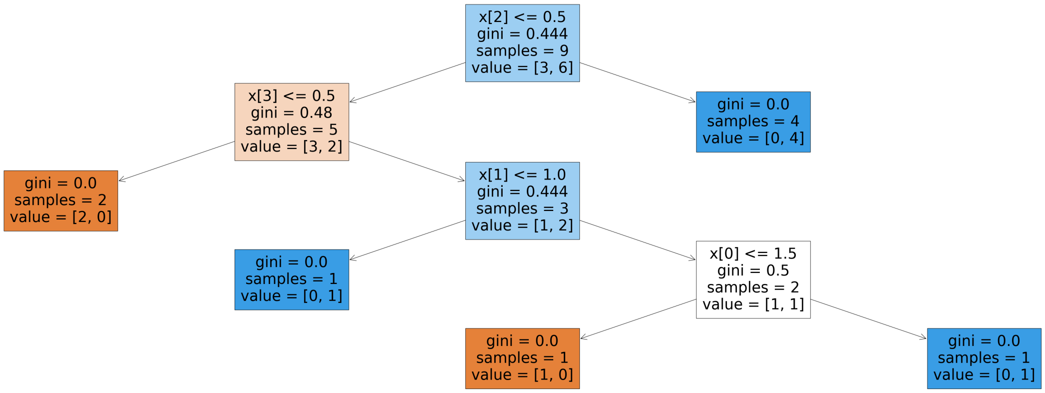
```
print("Therefore the root node that we can use for our decision tree is: ", root_node)
# This is the root node that can be used to determine the decision tree
```

```
     Therefore the root node that we can use for our decision tree is:  age
```

```
# Question A2:
# Creating a Decision Tree and finding the depth of the tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
decision_model = DecisionTreeClassifier()
decision_model.fit(train_input, train_output)
decision_tree_depth = decision_model.get_depth()
print("The depth of the decision tree is: ", decision_tree_depth)
decision_tree_prac = decision_model.score(test_input, test_output)
print("The decision tree accuracy of the dataframe is: ", decision_tree_prac, "\n")
```

```
     The depth of the decision tree is:  4
     The decision tree accuracy of the dataframe is:  0.4
```

```
#A3
from matplotlib import pyplot as plt
plt.figure(figsize=(70,20))
plot_tree(decision_model, filled=True)
plt.show()
```



```
from sklearn import preprocessing as prepro

label_encoder = prepro.LabelEncoder()
for i in label_encoding_columns:
    data[i] = label_encoder.fit_transform(data[i])

# One hot encoding schema
data = pd.get_dummies(data, columns = one_hot_encoding_columns)
data
```

```
# Question A4:
# Applying the decision tree on the project data
# Normalization of the project dataset
scaler = prep.StandardScaler()
scaled_columns = dataframe_project.columns[2:]
data_copy = dataframe_project
scaled_df = scaler.fit_transform(data_copy[scaled_columns])
scaled_df = pd.DataFrame(scaled_df, columns = scaled_columns)
for i in scaled_columns:
    dataframe_project[i] = scaled_df[i]
dataframe_project # Normalized dataset
input_data_pro = dataframe_project.drop(columns=['Label', 'Filename'])
output_data_pro = dataframe_project['Label']
```

```
# Applying the entropy on the project data
input_pro_data_train, input_pro_data_test, output_pro_data_train, output_pro_data_test = train_test_split(input_data_pro, output_data_pro, te
features = input_data_pro.columns
information_gain_values_pro = {}
for feature in features:
    information_gain_values_pro[feature] = information_gain(input_data_pro[feature], output_data_pro)
    print("The information gain for ", feature, " is: ", information_gain_values_pro[feature], "\n")


# Finally choosing the root node of the decision model
root_node_project = max(information_gain_values_pro, key = information_gain_values_pro.get)
print("The final project root node is: ", root_node_project)


# Decision tree creation for project data
from sklearn import metrics
decision_model_pro = DecisionTreeClassifier()
decision_model_pro.fit(input_pro_data_train, output_pro_data_train)
print("The decision tree accuracy of the training data of the dataframe is: ", decision_model_pro.score(input_pro_data_train, output_pro_data
decision_tree_depth_pro = decision_model_pro.get_depth()
print("The depth of the decision tree is: ", decision_tree_depth_pro)
decision_tree_acc = decision_model_pro.score(input_pro_data_test, output_pro_data_test)
print("The decision tree accuracy of the testing data of the dataframe is: ", decision_tree_acc, "\n")
plt.figure(figsize = (20, 10))
plot_tree(decision_model_pro, filled=True)
plt.show()


# Question A5:
# Construct a decision tree with a maximum depth constraint

# Decision tree model creation and fitting the training data with entropy with maximum depth 7
decision_model_pro_dep = DecisionTreeClassifier(max_depth = 7)
decision_model_pro_dep.fit(input_pro_data_train, output_pro_data_train)
print("The decision tree accuracy on the training data of the dataframe is: ", decision_model_pro_dep.score(input_pro_data_train, output_pro_
decision_tree_depth_pro_dep = decision_model_pro_dep.get_depth()
print("The depth of the decision tree is: ", decision_tree_depth_pro_dep)

# Decision tree model testing
decision_tree_acc_dep = decision_model_pro_dep.score(input_pro_data_test, output_pro_data_test)
print("The decision tree accuracy on the testing data of the dataframe is: ", decision_tree_acc_dep, "\n")

plt.figure(figsize = (20, 10))
plot_tree(decision_model_pro_dep, filled=True)
plt.show()


# Question A6:
# Constructing the decision tree using the entropy as criterion
input_dec_tree_data_train = input_pro_data_train
input_dec_tree_data_test = input_pro_data_test
output_dec_tree_data_train = output_pro_data_train
output_dec_tree_data_test = output_pro_data_test

# Decision tree model creation and fitting the training data with entropy as the criteria
decision_model_pro_dep = DecisionTreeClassifier(criterion='entropy')
decision_model_pro_dep.fit(input_dec_tree_data_train, output_dec_tree_data_train)
print("The decision tree accuracy of the training data of the dataframe is: ", decision_model_pro.score(input_dec_tree_data_train, output_dec

# Decision tree model testing
output_dec_tree_data_pred = decision_model_pro_dep.predict(input_dec_tree_data_test)
decision_tree_depth_pro_dep = decision_model_pro_dep.get_depth()
print("The depth of the decision tree is: ", decision_tree_depth_pro_dep)
decision_tree_acc_dep = metrics.accuracy_score(output_dec_tree_data_test, output_dec_tree_data_pred)
print("The decision tree accuracy of the dataframe is: ", decision_tree_acc_dep, "\n")

plt.figure(figsize = (20, 10))
plot_tree(decision_model_pro_dep, filled=True)
plt.show()


# Question A7:
# Construct a random forest on our project dataset
from sklearn.ensemble import RandomForestClassifier

# Giving the model variables to the random forest
input_rand_for_data_train = input_pro_data_train
input_rand_for_data_test = input_pro_data_test
output_rand_for_data_train = output_pro_data_train
output_rand_for_data_test = output_pro_data_test

# Random forest model creation and fitting the training data
model_forest = RandomForestClassifier(n_estimators=50)
```

```
model_forest.fit(input_rand_for_data_train, output_rand_for_data_train)
print("The random forest accuracy of the training data of the dataframe is: ", decision_model_pro.score(input_rand_for_data_train, output_ran

# Random forest model testing
output_rand_for_data_pred = model_forest.predict(input_rand_for_data_test)
accuracy_random_forest = metrics.accuracy_score(output_rand_for_data_test, output_rand_for_data_pred)
print("The accuracy of the testing data random forest is: ", accuracy_random_forest, "\n")



# Comparing the accuracy between decision tree and random forest
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Performance metrics of the decision tree
print("The classification report of the decision tree")
print(classification_report(output_dec_tree_data_test, output_dec_tree_data_pred))
print("The confusion matrix of the decision tree")
print(confusion_matrix(output_dec_tree_data_test, output_dec_tree_data_pred))

# Performance metrics of the random forest
print("The classification report of the random forest")
print(classification_report(output_rand_for_data_test, output_rand_for_data_pred))
print("The confusion matrix of the random forest")
print(confusion_matrix(output_rand_for_data_test, output_rand_for_data_pred))
```