Report on RFDT

LIBRARIES USED: To implement this question, we have used numpy, pandas, and matplotlib.

1.) NUMPY: Numpy is a python library used when we want to work with arrays.

2.) PANDAS: Pandas are used for data analysis and associated manipulation of tabular data in data frames. A data frame is a data structure that organizes data into a 2-D table of rows and columns.

3.) MATPLOTLIB: Matplotlib is used to create 2-D graphs and plots using python scripts. Pyplot, a module of matplotlib, makes things easy for plotting by providing features to control line styles, font properties, etc.

4.) **RANDOM:** It is an inbuilt module of python that is used to generate random numbers.

```
# IMporting libraries that are used in our code
import numpy as np
import pandas as pd
from numpy import log2
import random
from collections import Counter
from matplotlib import pyplot as plt
```

PART 1. a and 1. b:

1.a) Building a decision tree by taking maximum depth as input and by randomly splitting the train set as an 80/20 split

1.b) Implementing the standard ID3 Decision tree algorithm using information gain to choose which attribute to split at each point without using scikit-learn.

STEP BY STEP IMPLEMENTATION:

STEP 1: Dataset is given as CSV (comma-separated values) file, which is being read using pandas. As we don't want to train our data using data in the attribute name," we dropped that column(axis=1) in our original data frame object(inplace=true). Then to check the data after dropping 'riskmm', the data frame is printed using head(), which by default gives five rows.

```python
# function that open the data set
def OpenDataSet(file=None):
    if file is None:
        print("Please give file name")
        return None
    df = pd.read_csv(file)
    return df


data_set = OpenDataSet("rain_predict_train.csv")
data_set.head()
```

	mint	maxt	rainfall	windd3	winds9	winds3	hum9	hum3	pres9	pres3	temp9	temp3	rain	riskmm	raint
0	7.4	25.1	0.0	WSW	4	22	44	25	1010.6	1007.8	17.2	24.3	No	0.0	No
1	12.9	25.7	0.0	WSW	19	26	38	30	1007.6	1008.7	21.0	23.2	No	0.0	No
2	9.2	28.0	0.0	E	11	9	45	16	1017.6	1012.8	18.1	26.5	No	1.0	No
3	17.5	32.3	1.0	NW	7	20	82	33	1010.8	1006.0	17.8	29.7	No	0.2	No
4	14.6	29.7	0.2	W	19	24	55	23	1009.2	1005.4	20.6	28.9	No	0.0	No

STEP 2: There are some attributes whose domain of values is continuous, so to ease our work, we will convert that into categorical data.

```
# Function that modify the categorical data
def Modify_categorical_dataSet(data_set):

    # Convert "Yes" -> 1 and "No" -> 0 in rain and raint feature
    data_set['rain'] = data_set['rain'].replace({'Yes': 1, 'No': 0})
    data_set['raint'] = data_set['raint'].replace({'Yes': 1, 'No': 0})

    # Convert "wind3" feature with there index value
    data_set['windd3'] = data_set['windd3'].replace({
        # ['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE', 'SSE',
        #    'SSW', 'SW', 'W', 'WNW', 'WSW']
        'E': 0,
        'ENE': 1,
        'ESE': 2,
        'N': 3,
        'NE': 4,
        'NNE': 5,
        'NNW': 6,
        'NW': 7,
        'S': 8,
        'SE': 9,
        'SSE': 10,
        'SSW': 11,
        'SW': 12,
        'W': 13,
        'WNW': 14,
        'WSW': 15
    })
```

So here, we have converted the data into categorical by taking each feature as we did in the code.

STEP 3: The function to split the data in 80% training and 20% testing is being made.

The original data frame after dropping the 'riskmm' and ratio of test_set(0.8) is being passed. test_size is calculated using ratio and data frame size. A list of indexes of the data frame is being made and stored in a list named indices, and we separated the indices of test_size using list indices and test_size. Then we split training data indices by dropping test_data indices from total data frame indices.

```python
# Now make a function that split our data set
def SplitDataSet(data_set, ratio):

    # data_set.shape -> (98421, 15) so take the first index
    total_rows = data_set.shape[0]
    train_size = int(ratio * total_rows)  # calculating the training size
    test_size = total_rows - train_size  # calculating the testing size

    # make a list of index for ease of random splitting
    indices = data_set.index.tolist()
    # take a random value from the list and we take upto training size
    training_indices = random.sample(indices, train_size)

    # Now we get our training data
    training_data = data_set.loc[training_indices]
    # Now we remove the training data to get test data
    testing_data = data_set.drop(training_indices)

    return training_data, testing_data  # return the data set


training_data_set, testing_data_set = SplitDataSet(data_set, 0.8)
```

STEP 4:

In the decision tree (Regression tree), we have two nodes: leaf nodes(that will store the output we need to predict) and non-leaf nodes or decision nodes, which frame the division categories. We created a class and then made all the functions that are required to build the decision tree from Information Gain, Entropy and choosing a feature, and so on,

```python
    def is_leaf_node(self):

        if self.value is not None:
            return True
        else:
            return False


# Function that calculate entropy of the feature
def entropy(y):
    unique, counts = np.unique(y, return_counts=True)
    # find the probability
    probs = counts / counts.sum()
    return -np.sum(probs * log2(probs))
```

```python
# DecisionTree class that contain root node and list of function which will use during spiltting
class DecisionTree:
    # Constructor
    '''
    Attributes: max_depth -> this prameter desctibe maximum how much level tree grows
    '''

    def __init__(self, max_depth=100):
        self.max_depth = max_depth          # max depth of our tree
        # descibe atleast 2 rows there to split the data set
        self.min_samples_split = 2
        self.n_feats = None                 # descibe how many features are there
        self.root = None                    # root node of our tree

    def fit_data(self, X, y):
        self.n_feats = X.shape[1]           # number of feature in our data set
        self.root = self.build_tree(X, y)   # Now we make our tree

    def build_tree(self, X, y, depth=0):

        n_samples = X.shape[0]              # tells the number of rows
        n_features = X.shape[1]             # tells the number of coloumn
        # return how many unique label ie. 'yes' or 'no'
        n_label = len(np.unique(y))

        # Check if stoping criteria occur or not
```

```python
# Contructor
'''
Attributes: feature_index -> index of the feature of a non-leaf Node
            threshold -> threshold value of non-leaf(internal) Node that split it into two half
            left -> left children of the root
            right -> right children of the root

            value -> value of the Node(leaf Node)
'''

def __init__(self, feature_index=None, threshold=None, left=None, right=None, value=None):
    self.feature_index = feature_index
    self.threshold = threshold
    self.left = left
    self.right = right
    self.value = value

# Function that return true if a node is a leaf node otherwise return false
def is_leaf_node(self):

    if self.value is not None:
        return True
    else:
        return False
```

```python
def choseFeature(self, X, y, feat_idxs):
    max_gain = -1
    split_indx = None
    split_threshold = None

    for idx in feat_idxs:
        X_col = X[:, feat_idxs]
        thresholds = np.unique(X_col)

        for threshold in thresholds:

            gain = self.Information_Gain(X_col, y, threshold)

            if gain > max_gain:
                max_gain = gain
                split_indx = idx
                split_threshold = threshold

    return split_indx, split_threshold

def Information_Gain(self, X, y, threshold):

    left_indxs, right_indxs = self.split(X, threshold)

    if (len(left_indxs) == 0) or (len(right_indxs) == 0):
        return 0
```

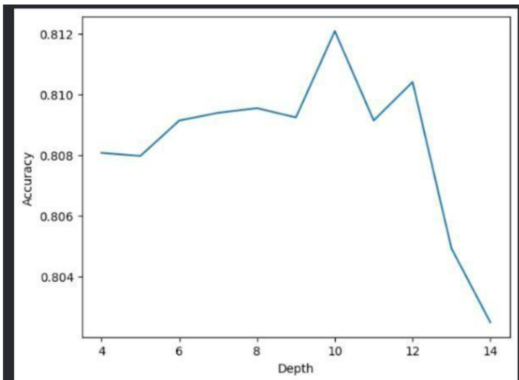
```python
# Check if stoping criteria occur or not

if (depth >= self.max_depth                 # if depth of our tree reaches to max or not
    or n_label == 1                         # check only one label is thare
        or n_samples < self.min_samples_split):  # number of rows is less than 2

    # find the most common label
    leaf_value = self.returnLabel(y)
    # return the leaf node of the tree
    return Node(value=leaf_value)

# it gives the list of random number ranging from n_features to self.n_feats
feature_idexes = np.random.choice(
    n_features, self.n_feats, replace=False)

chosen_feature, chosen_thresh = self.choseFeature(X, y, feature_idexes)

left_indxs, right_indxs = self.split(
    X[:, chosen_feature], chosen_thresh)
left = self.build_tree(X[left_indxs, :], y[left_indxs], depth + 1)
right = self.build_tree(X[right_indxs, :], y[right_indxs], depth + 1)

return Node(chosen_feature, chosen_thresh, left, right)
```

```
def traverse_tree(self, X, node):
    if node.is_leaf_node():
        return node.value

    if X[node.feature_index] <= node.threshold:
        return self.traverse_tree(X, node.left)
    else:
        return self.traverse_tree(X, node.right)

def predict(self, X):
    return np.array([self.traverse_tree(x, self.root) for x in X])
```

```
                best_testing_set = testing_data_set

        avg_accuracy = sum(accuracess)/len(accuracess)

        print(avg_accuracy)

        returning_item = [best_training_set, best_testing_set, bestTree, avg_accuracy]
        return returning_item
[27]  ✓ 0.0s

    returning_item = findBestTree()
[28]  ✓ 8.9s

... Finding the average accuracy over 10 random split....
0.8018288036576073
0.8047752095504191
0.8026924053848108
0.8060452120904242
0.8018288036576073
0.805740411480823
0.8021336042672086
0.8017780035560071
0.8083312166624333
0.8021336042672086
0.803728727457455
```

1.c) Depth V/S test Accuracy plot

```python
def findBestDepth(training_data_set, testing_data_set):

    best_depth = 0
    accuracies = []
    depths = [i for i in range(4, 15)]
    curr_depth = 4
    best_Acc = 0

    # Training part
    X_train = training_data_set.drop(['raint'], axis=1).values
    y_train = training_data_set['raint'].values

    # Testing part
    X_test = testing_data_set.drop(['raint'], axis=1).values
    y_test = testing_data_set['raint'].values

    for i in range(11):

        clf = DecisionTreeClassifier(max_depth=curr_depth)

        clf.fit(X_train, y_train)

        y_pred = clf.predict(X_test)

        acc = accuracy(y_pred, y_test)
        accuracies.append(acc)

        if acc > best_Acc:
            best_Acc = acc
            best_depth = curr_depth
```




```
The best depth is :  10
```

1.d) Implementation of Decision Tree Classifier from scikit-learn package for the best dataset split.

```python
# Now doing sklearn part

def fromSklearn(training_data_set, testing_data_set, best_depth):

    X_train = training_data_set.drop(['raint'], axis=1).values
    X_test = testing_data_set.drop(['raint'], axis=1).values
    y_train = training_data_set['raint'].values
    y_test = testing_data_set['raint'].values

    clf = DecisionTreeClassifier(criterion='entropy', max_depth= best_depth, min_samples_split=4)
    clf.fit(X_train, y_train)

    y_pred1 = clf.predict(X_train)
    print("Training Accuracy (sklearn): ", accuracy(y_train, y_pred1))

    y_pred2 = clf.predict(X_test)
    print("Testing Accuracy (sklearn): ", accuracy(y_test, y_pred2))

    # classificationReport(y_test, y_pred2)
    return clf

skl_clf = fromSklearn(training_data_set, testing_data_set, best_depth)
```

0.2s

```
Training Accuracy (sklearn):  0.8159926844137371
Testing Accuracy (sklearn):  0.8115316230632461
```

Part 2. a)Pruning the tree with reduced error pruning without using scikit-learn

```
# Now doing the pruning part

def reduced_error_pruning(training_data_set, testing_data_set):

    X_train = training_data_set.drop(['raint'], axis=1).values
    X_test = testing_data_set.drop(['raint'], axis=1).values
    y_train = training_data_set['raint'].values
    y_test = testing_data_set['raint'].values

    curr_acc = 0
    curr_depth = 11

    while curr_depth >= 3:

        clf = DecisionTree(max_depth=curr_depth)
        clf.fit_data(X_train, y_train)

        y_pred = clf.predict(X_test)

        acc = accuracy(y_test, y_pred)
        # print(acc)

        if acc > curr_acc:
            curr_acc = acc
        else:
            break

        curr_depth -= 1

    returning_object = [curr_depth, y_pred, curr_acc]
```

```
    curr_acc = 0
    curr_depth = 11

    while curr_depth >= 3:

        clf = DecisionTree(max_depth=curr_depth)
        clf.fit_data(X_train, y_train)

        y_pred = clf.predict(X_test)

        acc = accuracy(y_test, y_pred)
        # print(acc)

        if acc > curr_acc:
            curr_acc = acc
        else:
            break

        curr_depth -= 1

    returning_object = [curr_depth, y_pred, curr_acc]
    return returning_object

returning_object = reduced_error_pruning(training_data_set, testing_data_set)
print("With out sklearn (accuracy): ", returning_object[2])
```

[41] 9.2s

```
0.8086360172720345
0.8103632207264414
0.808737617475235
With out sklearn (accuracy):  0.8103632207264414
```

2.b) Reduced error pruning using scikit-learn

```
            curr_depth -= 1

        return clf_prune, curr_acc

    clf_prune, prun_Acc = pruning_sklearn(training_data_set, testing_data_set)

    print("With sklearn (accuracy): ", prun_Acc)
[53]  ✓ 0.3s
...  0.808737617475235
     0.8106680213360427
     0.8091440182880366
     With sklearn (accuracy):  0.8106680213360427
```

2.c) Using the entire training set to learn a decision tree with and without pruning.

```
print(f"On Entire Data Set\nWith out pruning: {avg_accuracy}\nWithour pruning: {returning_object[2]}")
[55]  ✓ 0.1s                                                                                    Python
... On Entire Data Set
    With out pruning: 0.803728727457455
    Withour pruning: 0.8103632207264414
```

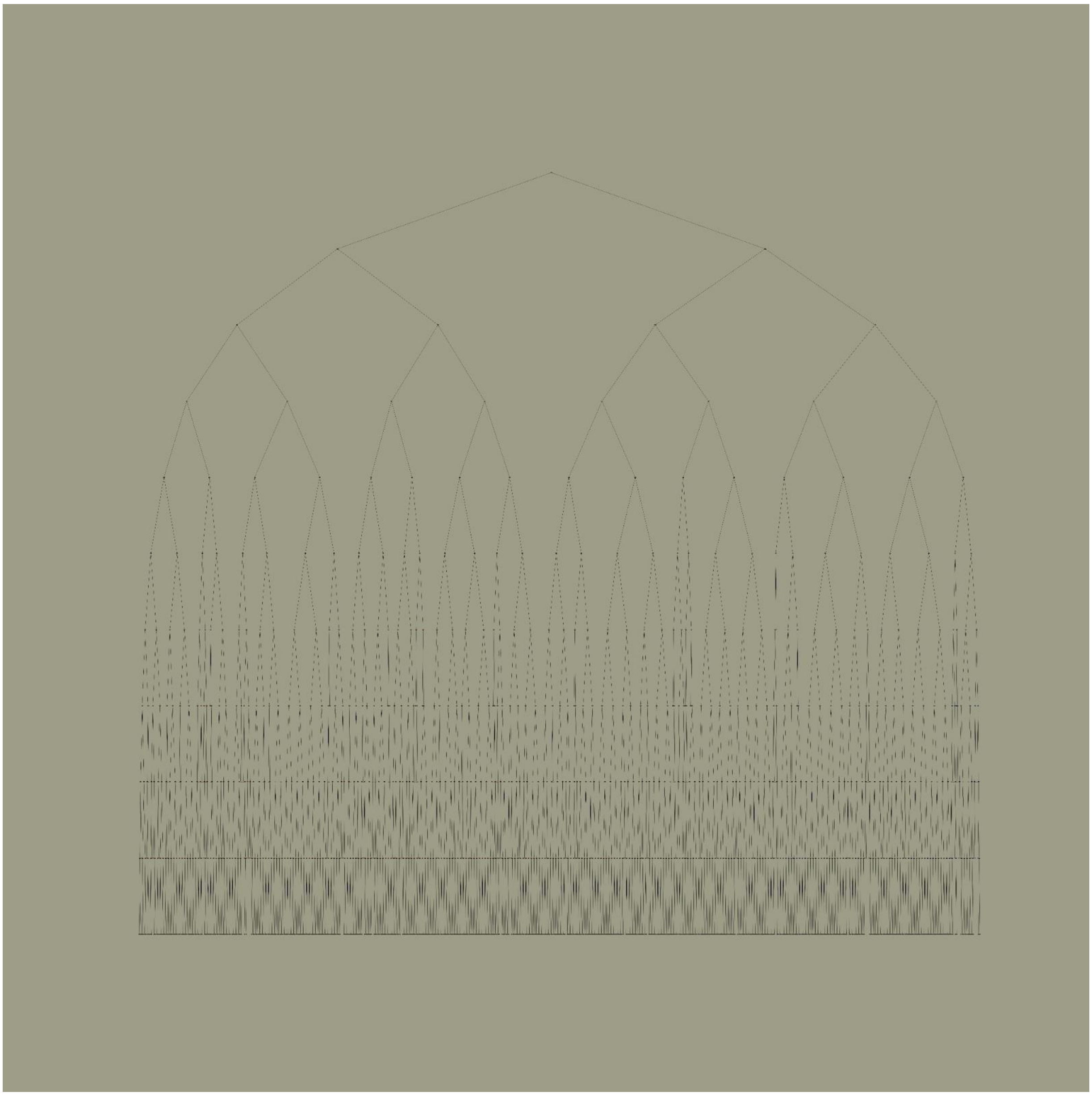
3. a) Classification report for both the trees in tabular form. (with and without pruning).
b) calculation of accuracy, precision, recall, f1-score, and support on the test set.

```
# Now doing the 3rd part

def classificationReport(testing_data_set, d_tree, prune_pred):
    from sklearn.metrics import classification_report

    y_test = testing_data_set['raint'].values
    X_test = testing_data_set.drop(['raint'], axis=1).values

    # d_tree = DecisionTree(max_depth=5)
    # d_tree.fit_data(X_train, y_train)

    y_pred = d_tree.predict(X_test)
    print("Classification report (without pruning):-\n",classification_report(y_test, y_pred))

    print("\nClassification report (with pruning):-\n", classification_report(y_test, prune_pred))

classificationReport(testing_data_set, d_tree, returning_object[1])
✓ 0.1s
```

```
Classification report (without pruning):-
              precision    recall  f1-score   support

           0       0.84      0.94      0.88     15364
           1       0.61      0.35      0.44      4321

    accuracy                           0.81     19685
   macro avg       0.72      0.64      0.66     19685
weighted avg       0.79      0.81      0.79     19685

Classification report (with pruning):-
              precision    recall  f1-score   support

           0       0.85      0.92      0.88     15364
           1       0.59      0.40      0.48      4321

    accuracy                           0.81     19685
   macro avg       0.72      0.66      0.68     19685
weighted avg       0.79      0.81      0.79     19685
```

4. final decision tree obtained from parts 1 a) and b)

```
# Now doing 4th part ie. Visulization to plot the graph
def plotTree(clf, file_name):
    fig = plt.figure(figsize=(120, 120))
    p = tree.plot_tree(
        clf,
        filled=True,
        feature_names=training_data_set.drop(['raint'], axis=1).columns.to_list(),
        class_names=('low', 'high')
    )
    fig.savefig(file_name)

plotTree(skl_clf, "tree.png")
✓ 1m 23.6s
```



```
# Plot tree with pruning
plotTree(clf_prune, "tree2.png")
```
✓ 35.4s