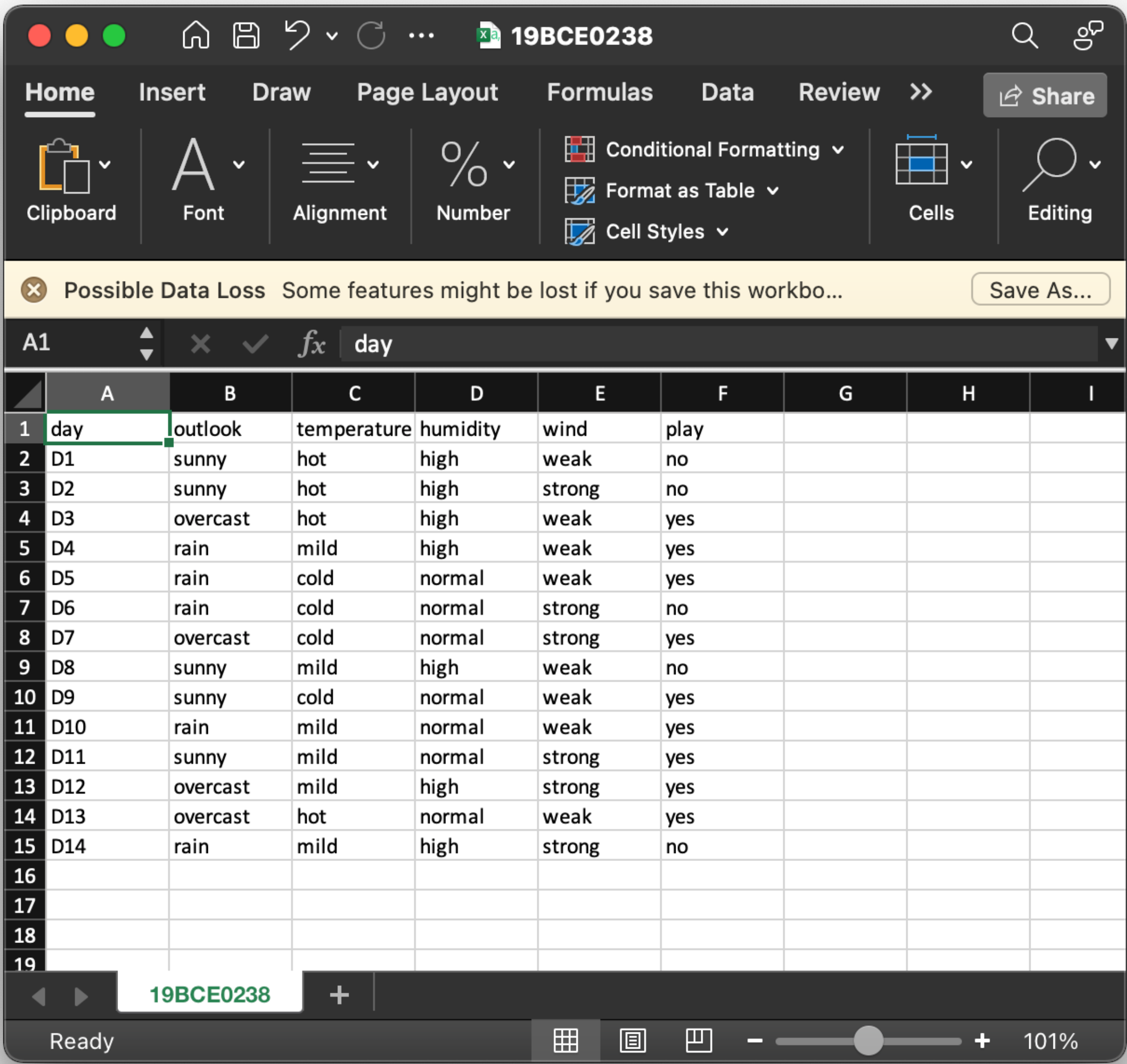
CSE4020

LAB-2

Sampath Routu
19BCE0238

Data



day	outlook	temperature	humidity	wind	play
D1	sunny	hot	high	weak	no
D2	sunny	hot	high	strong	no
D3	overcast	hot	high	weak	yes
D4	rain	mild	high	weak	yes
D5	rain	cold	normal	weak	yes
D6	rain	cold	normal	strong	no
D7	overcast	cold	normal	strong	yes
D8	sunny	mild	high	weak	no
D9	sunny	cold	normal	weak	yes
D10	rain	mild	normal	weak	yes
D11	sunny	mild	normal	strong	yes
D12	overcast	mild	high	strong	yes
D13	overcast	hot	normal	weak	yes
D14	rain	mild	high	strong	no

Aim: To implement the ID3 algorithm

Procedure:

1. Calculate the Information Gain of each feature.
2. Considering that all rows don't belong to the same class, split the dataset **S** into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
df  = pd.read_csv("19BCE0238.csv").drop('day', axis =1)
df.head()
```

Out[2]:

	outlook	temperature	humidity	wind	play
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cold	normal	weak	yes

In [3]:

```
def calc_total_entropy(train_data, label, class_list):
    total_row = train_data.shape[0]
    total_entr = 0

    for c in class_list:
        total_class_count = train_data[train_data[label] == c].shape[0]
        total_class_entr = - (total_class_count/total_row)*np.log2(total_class_count
        total_entr += total_class_entr

    return total_entr


def calc_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0

    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape
        entropy_class = 0
        if label_class_count != 0:
            probability_class = label_class_count/class_count
            entropy_class = - probability_class * np.log2(probability_class)
        entropy += entropy_class
    return entropy
```

In [4]:

```
def calc_info_gain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique()
    total_row = train_data.shape[0]
    feature_info = 0.0

    for feature_value in feature_value_list:
        feature_value_data = train_data[train_data[feature_name] == feature_value]
        feature_value_count = feature_value_data.shape[0]
        feature_value_entropy = calc_entropy(feature_value_data, label, class_list)
        feature_value_probability = feature_value_count/total_row
        feature_info += feature_value_probability * feature_value_entropy

    return calc_total_entropy(train_data, label, class_list) - feature_info
```

In [5]:

```
def find_most_informative_feature(train_data, label, class_list):
    feature_list = train_data.columns.drop(label)
    max_info_gain = -1
    max_info_feature = None

    for feature in feature_list:
        feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
        if max_info_gain < feature_info_gain:
            max_info_gain = feature_info_gain
            max_info_feature = feature

    return max_info_feature
```

In [6]:

```
def generate_sub_tree(feature_name, train_data, label, class_list):
    feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
    tree = {} #sub tree or node

    for feature_value, count in feature_value_count_dict.iteritems():
        feature_value_data = train_data[train_data[feature_name] == feature_value]

        assigned_to_node = False
        for c in class_list:
            class_count = feature_value_data[feature_value_data[label] == c].shape[(

            if class_count == count:
                tree[feature_value] = c
                train_data = train_data[train_data[feature_name] != feature_value]
                assigned_to_node = True
        if not assigned_to_node:
            tree[feature_value] = "?"

    return tree, train_data
```

In [7]:

```
def make_tree(root, prev_feature_value, train_data, label, class_list):
    if train_data.shape[0] != 0:
        max_info_feature = find_most_informative_feature(train_data, label, class_li
        tree, train_data = generate_sub_tree(max_info_feature, train_data, label, cl
        next_root = None

        if prev_feature_value != None:
            root[prev_feature_value] = dict()
            root[prev_feature_value][max_info_feature] = tree
            next_root = root[prev_feature_value][max_info_feature]
        else:
            root[max_info_feature] = tree
            next_root = root[max_info_feature]

        for node, branch in list(next_root.items()):
            if branch == "?":
                feature_value_data = train_data[train_data[max_info_feature] == node
                make_tree(next_root, node, feature_value_data, label, class_list)
```

In [8]:

```
def id3(df, label):
    id3_tree = {}
    class_list = df[label].unique()
    make_tree(id3_tree, None, df, label, class_list)
    return id3_tree
```

In [9]:

```
tree = id3(df, 'play')
```

In [10]:

```
import json
print(json.dumps(tree, indent = 4))
```

```
{
    "outlook": {
        "rain": {
            "wind": {
                "strong": "no",
                "weak": "yes"
            }
        },
        "sunny": {
            "humidity": {
                "normal": "yes",
                "high": "no"
            }
        },
        "overcast": "yes"
    }
}
```