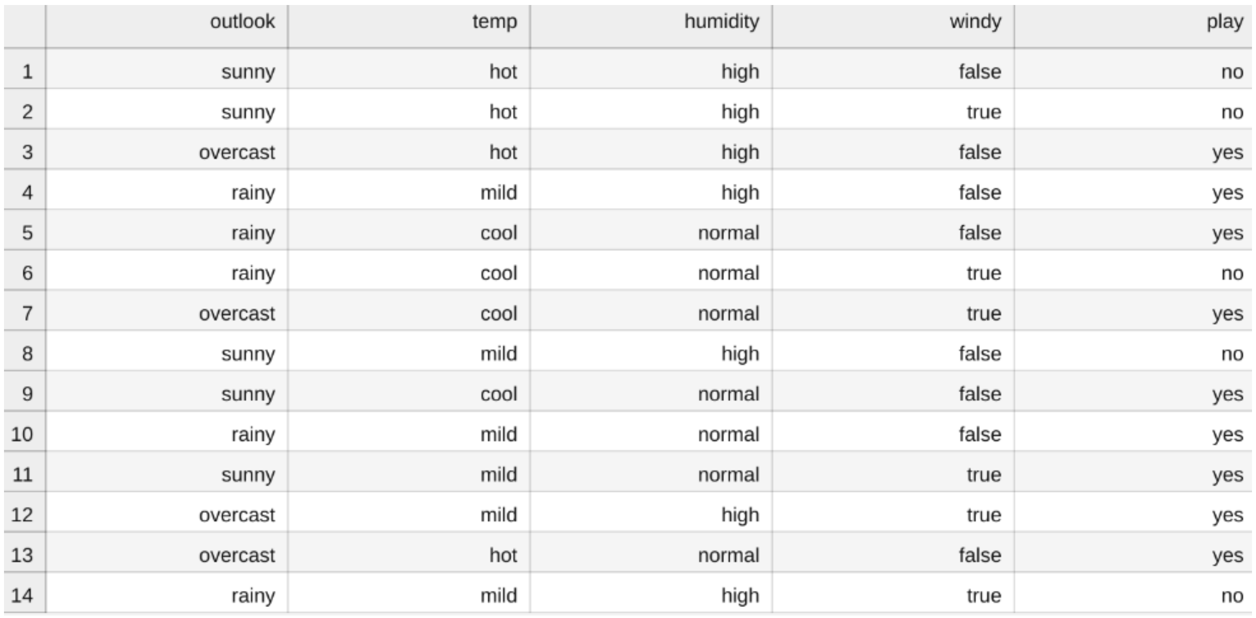
# Assignment 2 - Decision Tree

In this assignment, we will be implementing the decision tree algorithm using Python.

You are provided with the PlayTennis binary classification dataset (data.csv) to use while testing your implementation. In this dataset, the target attribute is the *"play"* column, which states whether a game of tennis was scheduled on a particular day (with some input weather attributes) or not.



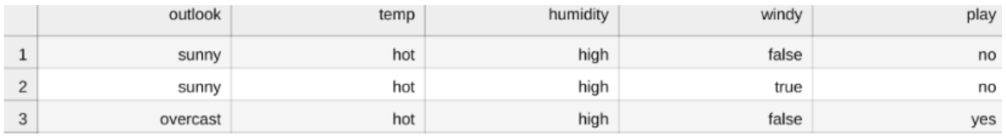
| | outlook | temp | humidity | windy | play |
|---|---|---|---|---|---|
| 1 | sunny | hot | high | false | no |
| 2 | sunny | hot | high | true | no |
| 3 | overcast | hot | high | false | yes |
| 4 | rainy | mild | high | false | yes |
| 5 | rainy | cool | normal | false | yes |
| 6 | rainy | cool | normal | true | no |
| 7 | overcast | cool | normal | true | yes |
| 8 | sunny | mild | high | false | no |
| 9 | sunny | cool | normal | false | yes |
| 10 | rainy | mild | normal | false | yes |
| 11 | sunny | mild | normal | true | yes |
| 12 | overcast | mild | high | true | yes |
| 13 | overcast | hot | normal | false | yes |
| 14 | rainy | mild | high | true | no |

## Tasks

### Entropy

We have provided the overall structure of the DecTree class. You need to complete its implementation by implementing the required methods:

1. Implement the `pmf_target` method. This method takes a dataset as input (pandas DataFrame) and returns a dictionary containing the target attribute values as keys and their normalized counts as values.
   E.g. for the below input DataFrame consisting of two "no" and one "yes" sample:



| | outlook | temp | humidity | windy | play |
|---|---|---|---|---|---|
| 1 | sunny | hot | high | false | no |
| 2 | sunny | hot | high | true | no |
| 3 | overcast | hot | high | false | yes |

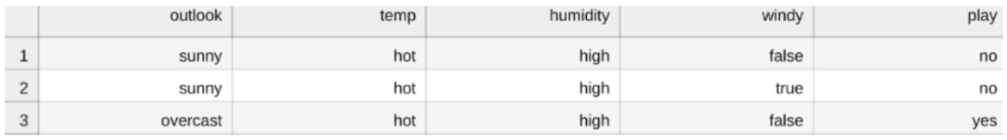
   Output will be `{'no': 0.6666666666666666, 'yes': 0.3333333333333333}`

2. Implement the `entropy` method. This will take the counts dictionary generated by the `pmf_target` method and compute the entropy:

$$\boxed{Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)}$$

Consider $\frac{0}{0} log_2(\frac{0}{0}) = 0.$

3. Implement the `cal_entropy_df` method. This method will take a DataFrame as input and use the `pmf_target` and `entropy` methods to compute the entropy.
E.g. for the below input DataFrame



| | outlook | temp | humidity | windy | play |
|---|---|---|---|---|---|
| 1 | sunny | hot | high | false | no |
| 2 | sunny | hot | high | true | no |
| 3 | overcast | hot | high | false | yes |

Output will be 0.918295

4. Implement the `info_gain_attribute` method. This method will take a DataFrame and attribute/column name as input. It will compute the information gain value achieved when splitting is done at that column:

$$\boxed{GAIN_{split} = Entropy(p) - \left( \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)}$$

Parent Node, p is split into k partitions;

$n_i$ is number of records in partition i

For the attribute “windy” and the input data used in part 3, the information gain will be 0.2516291

5. Implement the `max_info_gain_attribute` method. This will take a DataFrame and a list of attribute names as input and will return the attribute with the highest information gain
E.g. for the dataset in part 3, the “outlook” attribute is the optimal attribute with regards to information gain, with an information gain value of 0.9182958340.

## Build Tree

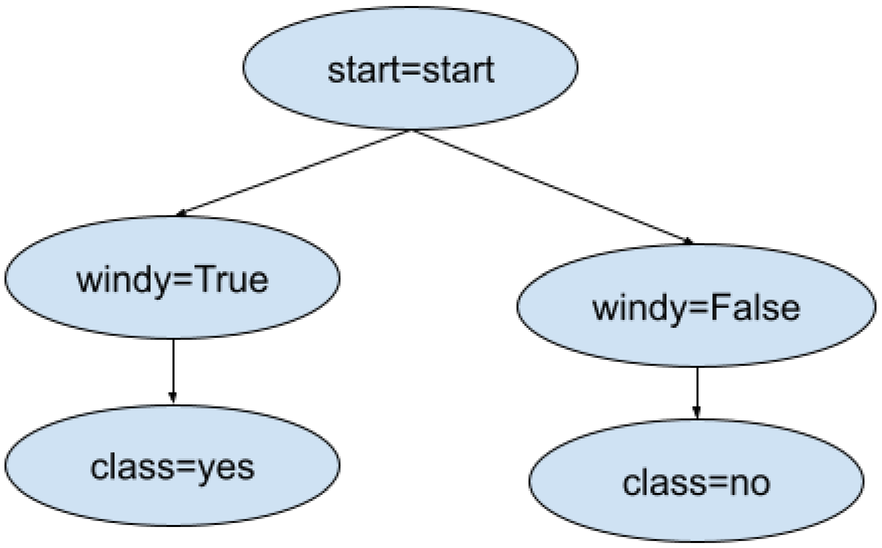
We will be using the tree data structure from the anytree library. The `DecTreeNode` class has been created for you. This will be used to represent the nodes of the decision tree.
Our decision tree representation will be formed from the following conventions:

a. Each node will represent a split.
E.g. if a node represents the split “outlook=overcast”, then during tree traversal, only those data samples which have the outlook column value as overcast will be passed to

the child subtree of this node.

b. Each node will have the following main internal variables:
    i. `attribute`: This will be the column name, at which the split is performed at a particular node.
    ii. `name`: This will be the column value at which the split is performed.
    E.g. if at a particular node, splitting is performed at "outlook=overcast", then outlook will be the `attribute` and overcast will be the `name`.
    iii. `attr_value`: A string representation of the `attribute` and `name` values.
    iv. `parent`: The parent node of a particular node.
c. The tree will have a placeholder node at its root with `attribute` and `name` set to "start"
d. The leaf nodes will represent prediction values, which will be output by the algorithm. At these nodes, the `attribute` will be set to "class" and name will be the predicted value (e.g. "yes" or "no")

A possible decision tree is shown in the below figure:



This tree will be implemented in code as follows:

```
start_node  = DecTreeNode("start", "start", parent=None)
windy_true  = DecTreeNode("True", "windy", parent=start_node)
windy_false  = DecTreeNode("False", "windy", parent=start_node)
class_yes = DecTreeNode("yes", "class", parent=windy_true)
class_no = DecTreeNode("no", "class", parent=windy_false)
```

Implement the `build_tree_infgain` method. In this method, you will implement the tree construction logic according to the recursive ID3 algorithm.
Refer to this link for the pseudo code.

Now, use the `generate_tree` (provided) method with the entire dataset to construct the

decision tree and visualize it with the `print_tree` (provided) method.
The tree should be constructed as follows:

```
 start=start
├──  outlook=sunny
│    ├──  humidity=high
│    │    └──  class=no
│    └──  humidity=normal
│         └──  class=yes
├──  outlook=overcast
│    └──  class=yes
└──  outlook=rainy
     ├──  windy=False
     │    └──  class=yes
     └──  windy=True
          └──  class=no
```

**Inference**

Now, implement the `predict` method. This method will use the constructed decision tree to perform predictions on a provided testing dataset (DataFrame). Output should be a list consisting of prediction classes.

If you feel the need to store additional information during tree construction, then you may do so by adding additional variables to the DecTreeNode class.

Also, perform predictions with your trained tree on the original dataset and print them.

## Evaluation on the cars dataset

You have been provided with another larger dataset. Train your decision tree model and perform predictions on data from the *cars_train.csv* and *cars_test.csv* files respectively.

Report the accuracy of your predictions. Also show the confusion matrix.

## Submission Instructions

Submit a zip file containing your notebook and all the three data files. The zip file should be named as *<your-roll-number_a2>.zip* e.g. *msds20001_a2.zip.*

- DO NOT PLAGIARIZE

- Late submissions will be penalized
- In case of any queries, post a comment in Google Classroom under the Assignment 2 post.