**Decision Trees :**

**Output**:

*=====================================================================*

*occupied:High;location:Talpiot=No*

*occupied:High;location:German-Colony=No*

*occupied:High;location:City-Center=Yes*

*occupied:High;location:Mahane-Yehuda=Yes*

*occupied:High;location:Ein-Karem=Unsure*

*occupied:Moderate;location:Talpiot;size:Large=No*

*occupied:Moderate;location:Talpiot;size:Small=Unsure*

*occupied:Moderate;location:Talpiot;size:Medium=Yes*

*occupied:Moderate;location:German-Colony;size:Large=Yes*

*occupied:Moderate;location:German-Colony;size:Small=Unsure*

*occupied:Moderate;location:German-Colony;size:Medium=No*

*occupied:Moderate;location:City-Center=Yes*

*occupied:Moderate;location:Mahane-Yehuda=Yes*

*occupied:Moderate;location:Ein-Karem=Yes*

*occupied:Low;size:Large=No*

*occupied:Low;size:Small=No*

*occupied:Low;size:Medium;price:Cheap=No*

*occupied:Low;size:Medium;price:Expensive=No*

*occupied:Low;size:Medium;price:Normal=Yes*

*=====================================================================*

*{'price': 'Cheap', 'VIP': 'No', 'music': 'Loud', 'location': 'City-Center', 'favorite\_beer': 'No', 'occupied': 'Moderate', 'size': 'Large'}*

*the attribute being checked is : occupied*

*value of the attribute which acts as the key to the next node or the class: Moderate*

*we haven't found our answer yet*

*the attribute being checked is : location*

*value of the attribute which acts as the key to the next node or the class: City-Center*

*we have our answer, and it is : Yes*

**Conclusion**: From the output shown, we conclude that the person will enjoy if he goes to the location City-Center and that this particular place is Moderately occupied (The Rules shown above are subject to change due to randomization with respect to choosing from more than one attribute having the same lowest entropy value

**About the Data Structures:**

* This program was coded using Python.
* The Data Structures that have been used for this program are Lists, Dictionaries and Classes.
* The Class, named as **Node** has properties:
  + **attribute\_name** - This would store the attribute which has the lowest entropy.
  + **childNodes** - This Dictionary would store the values related to attribute\_name as keys and the values related to the keys are Nodes containing subtrees which are recursively formed based on the records which contain that particular value for that particular key and they would contain the attribute with the lowest entropy based on those particular records as the root.
* Each line containing a training sample is read, processed into as list containing values associated with each attribute as well as the class and this list was referred to as a record.
  + Each record was added as an element in another list and this particular list of records is the one used to create the decision tree.
* We used a List to hold all the applicable attributes to use for the decision tree creation. This was known as **attributeset**.
* We used a Dictionary to store the entropy for each value. This was known as **entropy**.
  + The key would be the attribute name and the value would be the entropy value of that particular attribute.
  + Each value would be initially 0.0
* We used a Dictionary to create a filler data structure called **reset**, which would contain the classes related to the data as keys and 0 as its values. This is used to count records which have one of those classes (In this case, either **Yes** or **No**)
* We had Dictionaries called by the names of the attributes in question.(In this case, it is **size**, **occupied**, **VIP**, **price** etc)
  + each of these Dictionaries had nested had the values related to their namesake representing a particular attribute as keys.
  + The values associated with each of those keys were deep-copies of **reset**
* We used another Dictionary named **valueset**,which had the attribute names as it’s keys and each of those keys had deep-copies of Dictionaries called by the names of the attributes in question (In this case, deep-copies of **size**, **occupied**, **VIP**, **price** etc)
* Deep-copies of the **valueset**, **attributeset** and **entropy** were also made and added as elements into another Dictionary called **originals**
* **ruleset** is a list which is used to maintain a set of rules created during the process of making a decision tree

**Mechanism of the Program**:

Functions that made up the program:

**makeDT(root’, entropy’, valueset’, attributeset’, records’, setState’, parent’, key’, level’,rule’):** This is a recursive function that is used to make the decision tree. Entropies are calculated for each value and the variables referenced by the parameters are changed by the **getEntropy** and **calcEntropy** functions. The attribute with the lowest entropy is chosen and will be the root of each subtree. Each value of that particular attribute is checked and if it is an impure set, the function is recursively called with appropriate parameters, else the class of the value will be either **Yes** or **No**

* **root’**: the starting node of the decision tree and is of type **Node**
* **entropy’**: reference to **entropy**
* **attributeset’**: reference to **attributeset**
* **valueset’**: reference to **valueset**
* **records’**: reference to the set of training data records
* **setState’**: reference to the dictionary containing the deep-copies of **entropy, attributeset, valueset**
* **rule’:** reference to the string that is used to create rules
* **parent, key, level**: These are variables that were used as check parameters to make sure the program is working

**getEntropy(valuesetref, entropyref, recordsref, attributeref):** This is a function that is used to prepare the data structures referenced by the appropriate parameters for the particular attribute

* **valuesetref**: reference to **valueset’**
* **entropyref**: reference to **entropy’**
* **recordsref**: reference to **records’**
* **attributeref**: attribute for which the above data structures are updated in preparation for entropy calculation

**calcEntropy(attributeref, valuesetref, entropyref, recordsref):** This function is used to calculate the entropy of the particular attribute referenced in the parameters

* **valuesetref**: reference to **valueset’**
* **entropyref**: reference to **entropy’**
* **recordsref**: reference to **records’**
* **attributeref**: attribute for which the entropy is calculated

**getLowest(ent):** Get the attribute with the lowest entropy. If there is more than one attribute with the lowest entropy value, then we implement a randomization system to choose a random attribute from either of those particular attributes

* **ent:** reference to **entropy’**

**travel(traveller, testcase):** Travel through the now created decision tree based on the attribute:value pairs of the **testcase**, until next value represented by **traveller** is a class.

**Challenges faced:**

The challenges were in making sure that the current node in the decision tree would not be wrongly calculated by using already stale variables which were used to make calculations for its parent node. This was solved by using the **deepcopy( )** function of python to create completely fresh copies of those particular variables before their use in creating the parent node and then make appropriate changes to those copies (i.e deleting elements associated with the attribute that had the lowest entropy) and then send those copies as parameters to the recursive function that would create the current node.