

***D****ata* ***S****tructures* ***P****roject* ***R****eport*

**Group Members**

**Aheed Tahir** **21K-4517**

**Abdul Haseeb 21K-3217**

**Ashad Abdullah**  **21K-3296**

**Teacher**

**Dr. Fahad Sherwani**

Delish Advocacy: A Restaurant Recommender

using **Decision Tree**

**5th November - 4th December 2022**

# Abstract (Introduction and Background)

A crowdsourced restaurant recommendation system that filters out a restaurant from a myriad of preset ones purely based on the user preferences after a series of dichotomous questions explicitly designed to target precisely what the user desires. The decision tree uses cuisine preferred, pricing, time of the day, and reservation availability, to filter the restaurant out.

# 

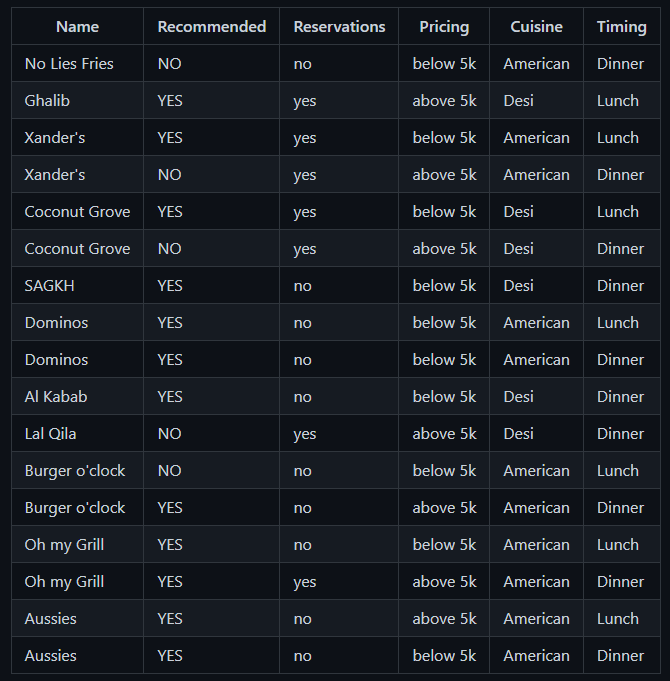
# Implementation Of Decision Tree Classification

Decision trees are a simple machine learning algorithm that uses a series of features of observation to create a prediction of a target outcome class. In our case, it is helping us classify various circumstances of user adoption and predict whether that restaurant is advocated or not. The decision tree is trained using a series of observations (training data) to create a tree that best fits the training data. This tree can then be used to make predictions on a series of observations without knowing the target outcome. In our case, the target outcome is the whether the restaurant is recommended or not.

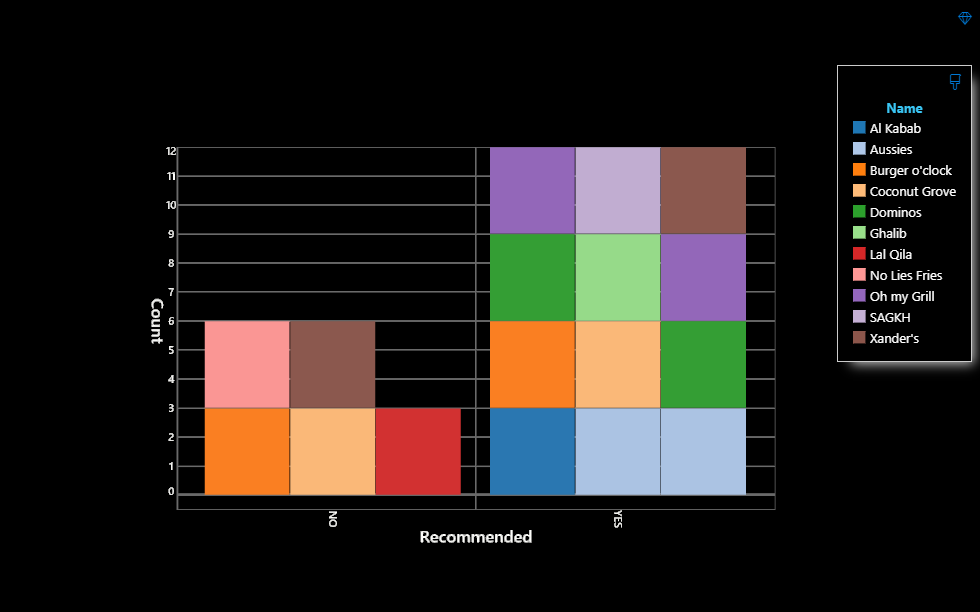
The Data Set For out model consist of these features/questions.

1. Is the reservation available
2. The price range for that restaurant
3. Cuisine style
4. Timing of choice

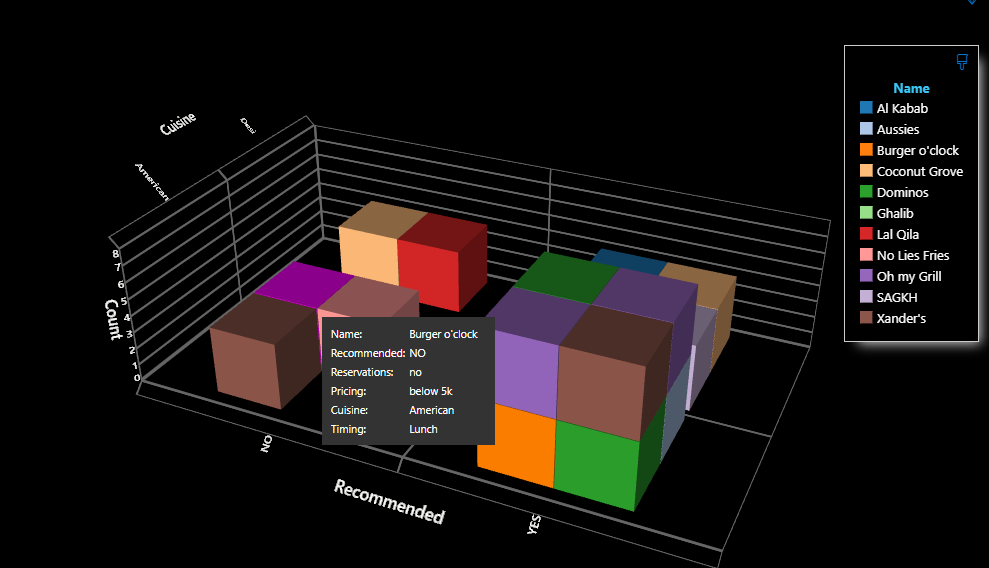
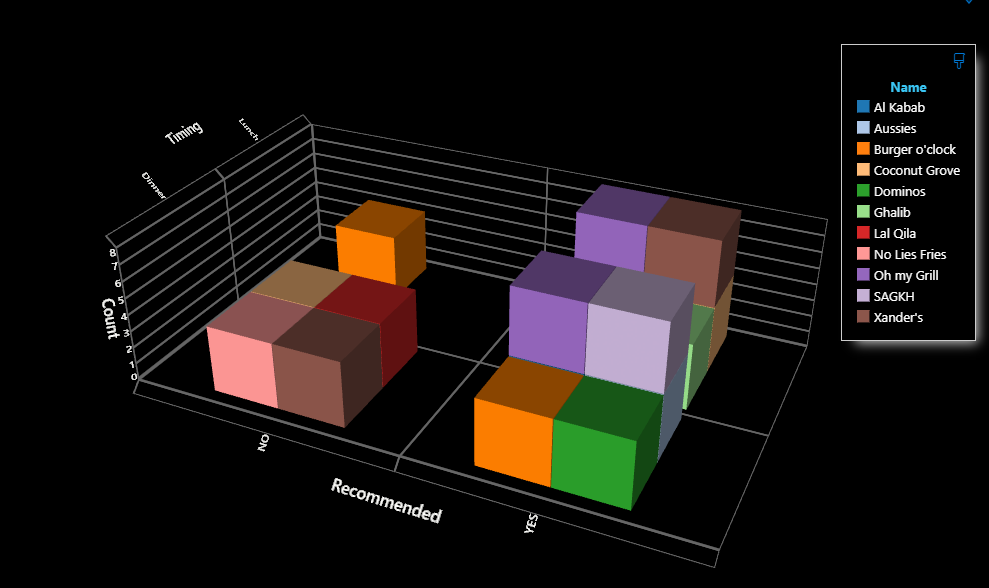
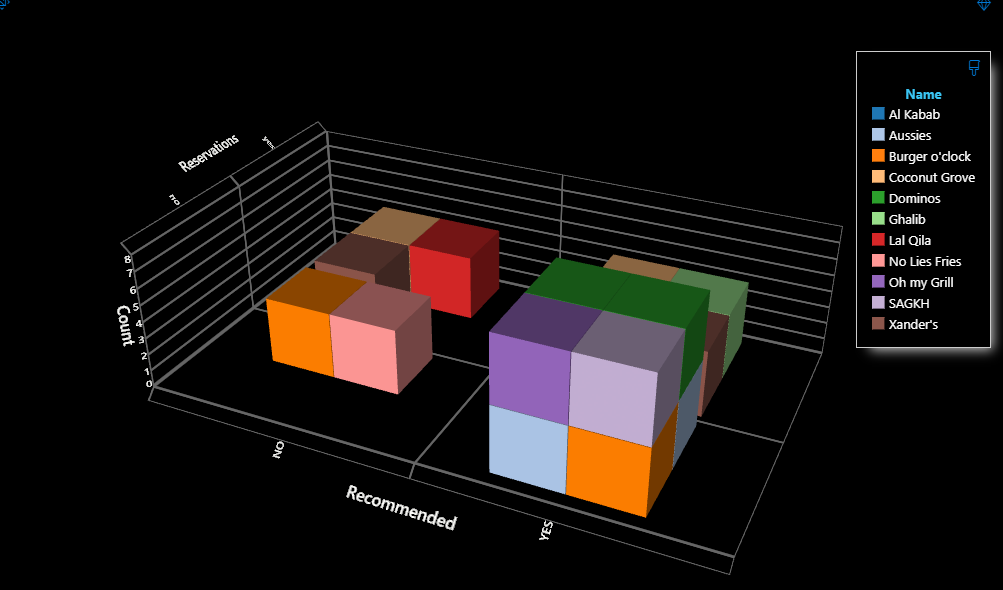
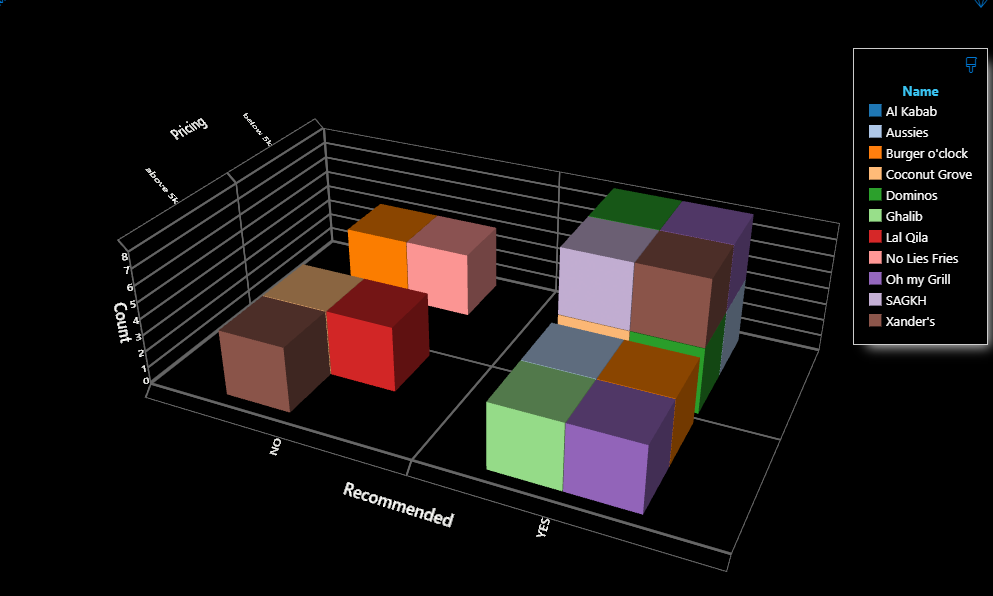
Providing that, the data set of our model looked like this:



* At first glance, out data looked like this:



* But after breaking the data into various categories recursively, the data transformed like this (first recursion)

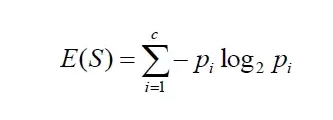


Technology Used: [sanddance](https://www.microsoft.com/en-us/garage/profiles/sanddance/)

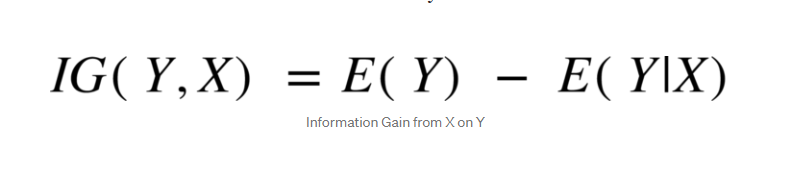
# How the decision tree was made!

To train a decision tree classifier,

* The model has a number of measurements of categorical feature vectors that keep control of the whole data for each node of a tree.
* We used [ID3](https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1) algorithm to make a decision tree.
* To make sure which feature stands out the most, we use entropy measurements



* For each feature, we calculated the info gain for every table.
* Selected the column/feature that maximizes the info gain, and assign that to the node.



* Split the data based on that feature to create two new sub-trees and two tables
* Repeat the splitting of nodes in each sub-tree until only one outcome is left (info gain = 0)

# Inputs

The data is imported using a CSV reader function which reads the data in strings and returns a 2D vector (string). Next to simplify the problem that the decision tree can be run on any data provided we converted whole data in integers/bits using bit\_masking: it encoded the strings to integers and that is input to the decision tree as training data

The first column of names is omitted from the input and each column will contain all the question user is asked, a column is representing a feature or an individual question. The second column (first column of training data) represents outcomes, whether the restaurant is recommended or not.

# Output

We asked the user several questions (features of our data) and predict the outcome for that user. NOTE: There will be one less column in the user data as the outcome would be unknown.

# Data Structure used

***Vectors***

A vector/dynamic array of pointers for all features, where every element points to an array with the value of that feature for each training observation.

We used several dimensions of vectors based on various purposes.

* For output, a 1-D vector is used
* For training data, the 2-D vector is used (named Matrix)
* For splitting the data, two 2-D vectors are used (named as TwoMatrix)

***Maps***

To keep track of counts: maps (like dictionaries) are used. Which helps us in BIt\_masking, column\_deletion, info\_gain, and entropy calculations.

NOTE: maps are self-balancing red-black trees that are sorted.

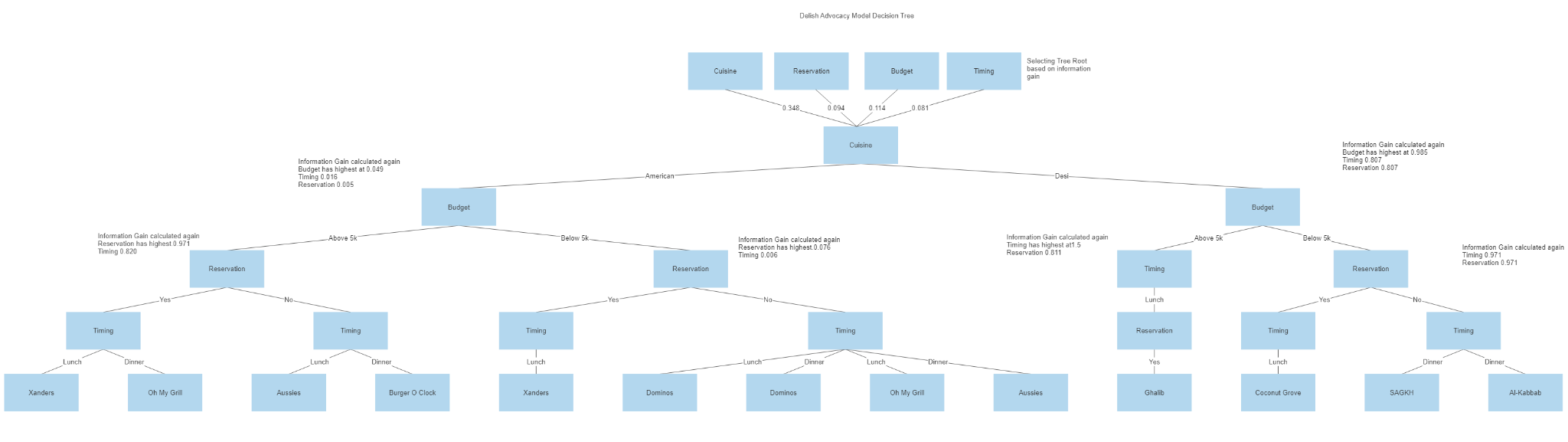
***Trees***

The binary tree is used for decision-making where every left node goes with higher info gain than that of the right node.

Decision Trees classify labels by sorting them down the tree from the root node to some leaf node. Each node specifies a test of some *attribute* of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.

Our basic algorithm learns the decision tree by constructing it top-down, beginning with the question, “Which attribute should be tested at the root of the tree?” To answer this question, each attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected as the root of the tree. Our next task is to find which node will be next after the root. In this case, we would like to again choose the attribute which is most useful to classify training examples. Then repeat the process until we find the leaf node.

An example of the tree on the above data looked like this:



# Architecture and Design

Each Tree Node has the following attributes and methods:

1. *TrainData* - Specific table data for every node.
2. *bestSplit* - contains two variables: resEntropy and Feature of that node.
3. *leftChild* - pointer to a left node of the tree.
4. *rightChild* - pointer to the right node of the tree.
5. *CountResults()* - will give counts of every feature in the table.
6. *EntropyOfCol()* - will give the entropy of a specific column required for the counts of features.
7. *ColGain()* - will use the entropy of the column function to calculate the gain of the column.
8. *GetInfoGain()* - will calculate the info gain of the whole table by using the above methods.

Data\_Retreival class contains 2 methods:

1. *RETRIEVE\_DATA* - Fetch the data from a CSV file and save it in a 2D-vector.
2. *Bit\_Mask* - Encode the data to bits and omits the miscellaneous column.

decision tree class contains the following attributes and methods:

1. A Data retrieval object - used to retrieve data from CSV which will be used by the root Node
2. *Root* - A tree node object.
3. *DecisionTree()* - constructor to initialize and set attributes required for first use
4. *ConstructTree()* - Construct the tree using the training data.
5. *Predicts()* - will take the output data and returns 0/1, whether the restaurant is recommended or not.

Menu Class Contains the following attributes and methods:

1. loadques(); ,void loadname(); void loader(); void showcredits(); - are all for loading screen gui for the program
2. ccolor(int);- to set text color attribute in the console
3. userinput(); function that takes the user input for their desired restaurant
4. dname(); - displays program name at the top
5. gotoXY(int, int); - enables cursor positon setting in console
6. mainmenu(); - displays a main menu / home screen for the user
7. timingmenu(), reservmenu(), budgetmenu(), cuisinemenu(); - all user menus that input the user choices

Other attributes and functions:

1. *isIdentical()* - helper function for bit\_mask
2. *PrintData1()* - for printing 1-D vectors
3. *PritnData2()* - for printing 2-D vectors
4. *PrintList1() - for printing Restaurant name lists from Vector*

The menu class contains various functions for user interaction. Every part is divided into many readable functions which make the interface very pleasing to the user.

# 

# Results and Discussion

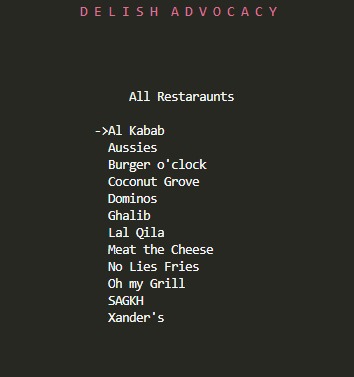
We used console graphics to implement a eye pleasing intro to our program , which includes a loader and animated texts.

Furthermore below attached is a walkthrough of our program with both possible outcomes:

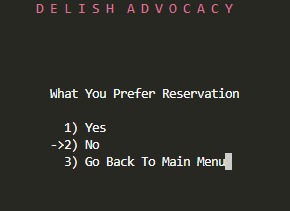
Initially after the loader execution ends , the user is welcomed by this main interface

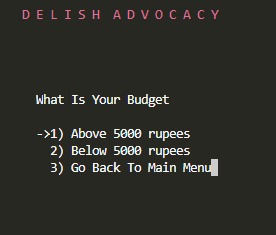
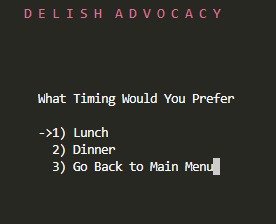
We wanted to keep it minimalistic.

The List of Restaraunts is as shown below

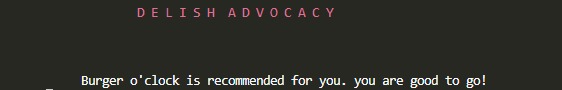


Selecting any of these would set the restaurant as the user restaurant and now our program will target question the user to mark out their preferences



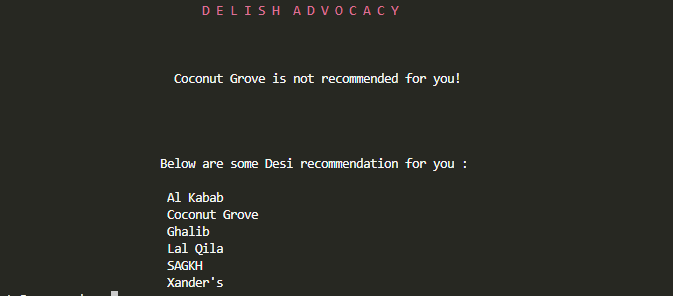


These series of Dichotomous questions help filter out unwanted results and we can compare it to the User Selected Restaraunt, if the answers to these questions match the properties of the Restaurant in question , we will recommend it to the user!



Incase we detect the user preferences to be different from the Restaraunt they chose, we will not recommend it to them

However! We will instead recommend some other restaurants that our program decides is better suitable for the user!



# 

# 

# 

# 

# Conclusion

As a jist , we used vectors to store our data , used entoropy/gini to get information gain and then implemented a form of data structures called Desicion Trees to help filter our data and leave us with what we need.

Our recommendation system is made to remove the hassle of trying out new restaurants wasting time and money. This is a really needed program which we hope to implement on a national scale one day that can help us in finding out restaurants and places to eat that sync with our taste and preferences.

# GitHub URL

* [code in main-2](https://github.com/Ashad001/Restaurant-Recommneder)