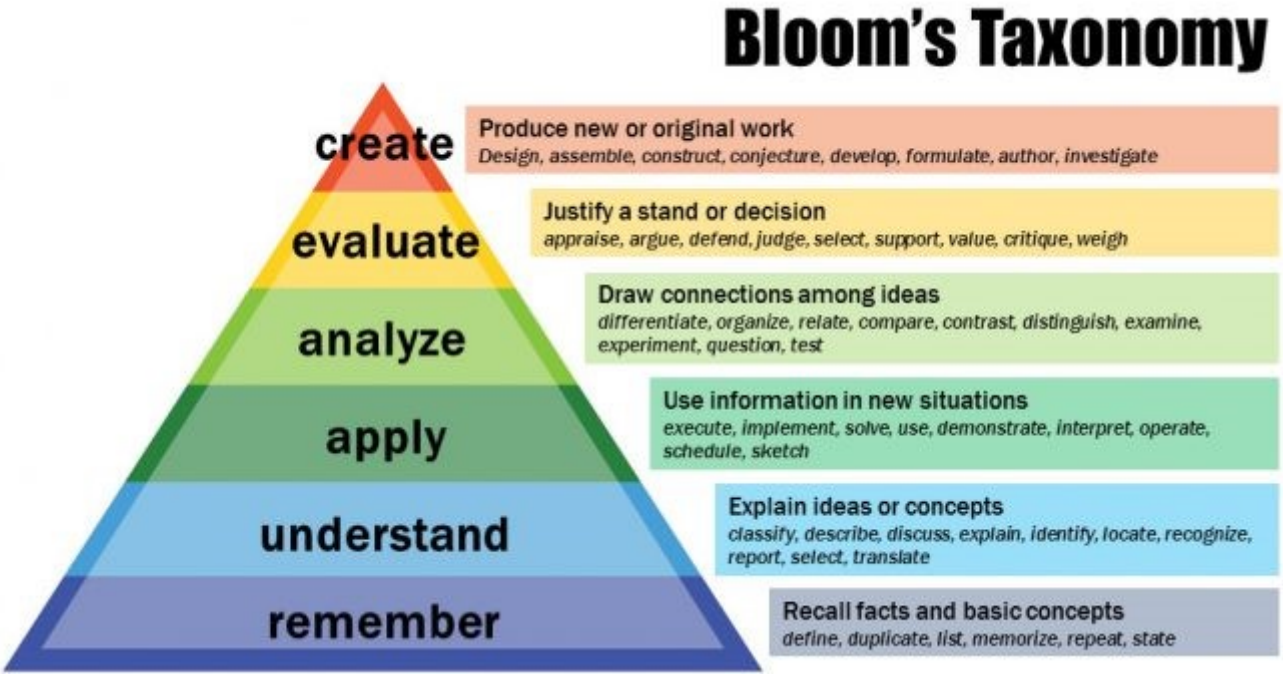
To find the decision based on a given scenario from a dataset using Decision Tree Classifier.

Problem Statement:

A dataset collected in a cosmetics shop showing details of customers and whether or not they responded to a special offer to buy a new lip-stick is shown in table below. Use this dataset to build a decision tree, with Buys as the target variable, to help in buying lip-sticks in the future. Find the root node of decision tree. According to the decision tree you have made from previous training data set, what is the decision for the test data: [Age < 21, Income = Low, Gender = Female, Marital Status = Married]?

ID	Age	Income	Gender	Marital Status	Buys
1	< 21	High	Male	Single	No
2	< 21	High	Male	Married	No
3	21-35	High	Male	Single	Yes
4	>35	Medium	Male	Single	Yes
5	>35	Low	Female	Single	Yes
6	>35	Low	Female	Married	No
7	21-35	Low	Female	Married	Yes
8	< 21	Medium	Male	Single	No
9	<21	Low	Female	Married	Yes
10	> 35	Medium	Female	Single	Yes
11	< 21	Medium	Female	Married	Yes
12	21-35	Medium	Male	Married	Yes
13	21-35	High	Female	Single	Yes
14	> 35	Medium	Male	Married	No

Blooms Taxonomy



Bloom’s Taxonomy was created by Benjamin Bloom in 1956, published as a kind of classification of learning outcomes and objectives, it is a set of three hierarchical models used to classify educational learning objectives into levels of complexity and specificity. The three lists cover the learning objectives in cognitive, affective and sensory domains. The cognitive domain list has been the primary focus of most traditional education and is frequently used to structure curriculum learning objectives, assessments and activities. This document is a reflection of learning objectives in a cognitive domain.

Objectives:

- **Evaluate** various classification algorithms
- **Analyze** the data and determine why decision tree is the adviced method for the given data
- **Apply** decision tree algorithm on the given data
- **Understand and Remember** the concept of entropy, decision tree and the applications of decision tree

Why machine learning?

- Let X be the set of input features and Y be the set of output label such that X_i^j is the ith element of input feature j in set X and similarly Y_i is the output label for the set of ith input feature.
- For each feature j in set X there exits one-to-many mapping i.e. the mapping fuction is **surjective**.
- This justifies the use of machine learning as it can used to resolve **non deterministic mapping from X to Y.**

Here the machine creates a decision system based on the available dataset. The learning process takes place when the algorithm determines the importance of existing features in the decision making capacity of the system.

Why supervised learning?

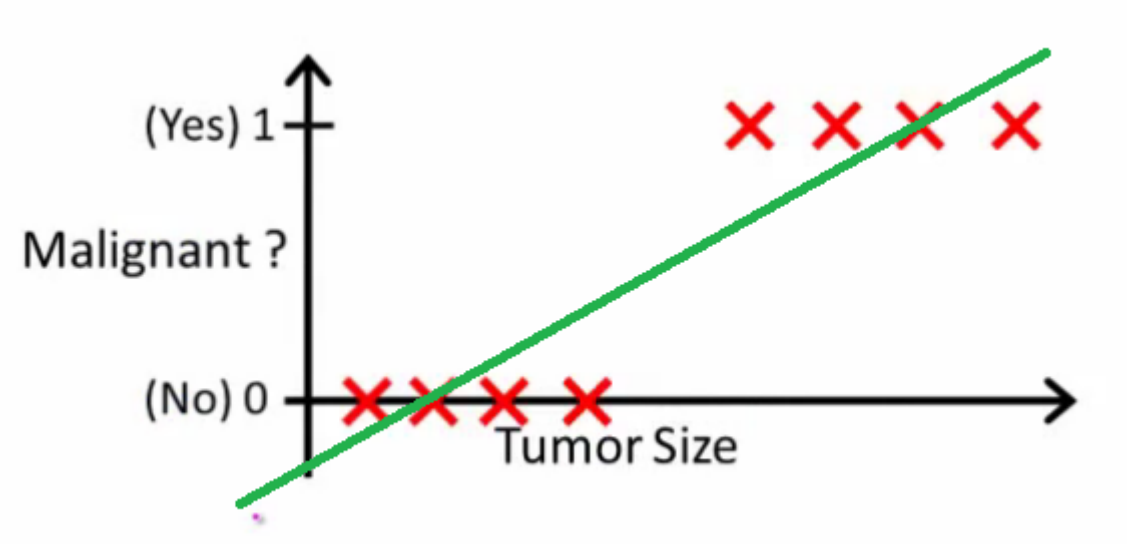
Supervised learning involves selecting a heuristic function using distinctly known feature/s to map labels. By observing the given data we can conclude that the input features and output labels are distinctly known, also the decision making process in a decision tree travels from the intermediate nodes(distinct features) to leaf nodes (labels). Hence satisfying the rules necessary for the use of supervised learning.

Why classification? (Bloom's Taxonomy: Evaluate)

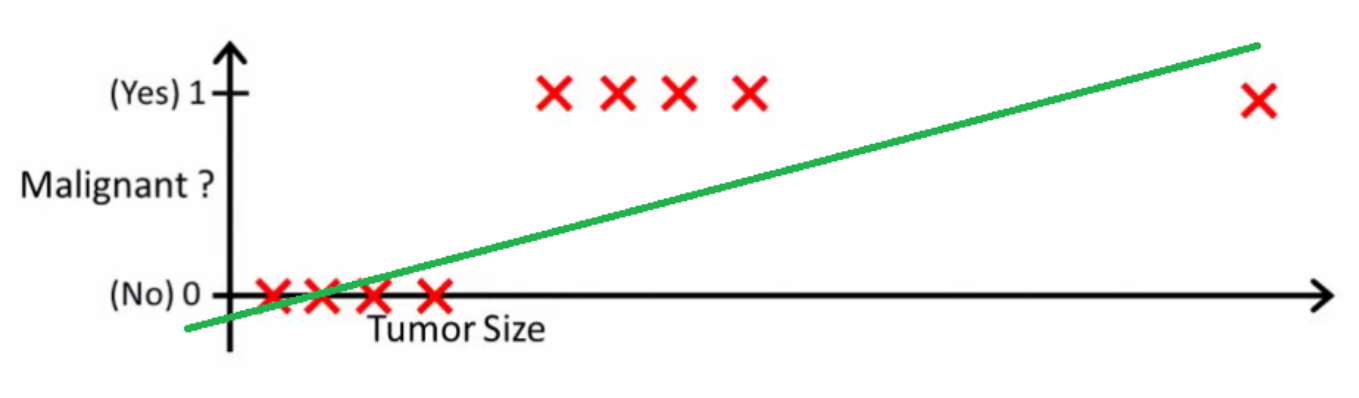
After analyzing the data it can be clearly observed that the ouput label "buys" is seperated into two distinct classes "Yes" or "No" i.e either the customer buys the lipstick or he/she doesn't buy the lipstick. As classification is the task of predicting such discrete class labels we can say that the given problem is classification problem.

Why not regression?

Regression models predict a continuous variable, in case of logistic regression they can be used to predict probabilities.



Consider the task of predicting malignancy of tumours. Let hypothesis h(x) represent malignant tumors with value 1 and non-malignant ones with value 0. Let tumor size be input feature x to make predictions for any given tumor size x, if h(x) > 0.5 the tumour is predicted to be malignant tumor, otherwise it is predict to be benign. Using this hypothesis it is possible to correctly predict every single training set sample, but addition of another sample with a huge tumor size renders our hypothesis to give inaccurate predictions. For regression to work here the hypothesis must be modified to something like h(x) > 0.2. Here lies the limitation of using regression algorithms for classification as the hypothesis cannot be changed each time a new sample arrives.



Programmer's Perspective:

Let S be the programmer's perspective of the linear regression, such that S = {s, e, X, Y, f_{me}, f_i^f | Φ}

s = Start State = { (X_i^j, y_i) | $X_i^j \in X$, $y_i \in y$, i < length of dataset and j < number of feature },

e = { Decision Tree for correctly mapping the non-deterministic data }

X = {X,y}

- X: Feature values = { (age, income, gender, marital status) |
  - age $\in$ { <21, 21-35, >35 },
  - income $\in$ { low, medium, high },
  - gender $\in$ { male, female},
  - marital status $\in$ { married, single}
- y: Output values = { y_i | y_i is the prediction for purchase decision of customer X_i, $y_i \in$ {yes, no}, $i \in \mathbb{N}$ },

f_{me} = {create_decision_tree(features, data, cur_depth, metric, max_depth) | X= (features, data, cur_depth, metric, max_depth), y=decision_tree}

- data: {age, income, gender, marital status}
- features: {list of all features}
- curr_depth: Current depth of the tree
- metric: {selection criteria}, metric ϵ {gini, entropy}
- max_depth: {Maximum depth of the tree}
- Method to build decision tree based on input data

f_i^f = friend functions: {f_1, f_2, f_3, f_4, f_5} For ever function f_i in f_i^f :

$$f_i : X \rightarrow y$$

1. f_1 : {entropy(feature, data) | X=(feature,data), y=entropy}

   - data: {age, income, gender, marital status}
   - feature: {selected feature}
   - y: entropy ϵ [0,1]
   - Method to find classification error of a specific feature

2. f_2 : {gini_index(feature, data) | X=(feature,data), y=gini_index}

   - data: {age, income, gender, marital status}
   - feature: {selected feature}
   - y: gini_index ϵ [0,1]
   - Method to find gini index of a specific feature

3. f_3 : {find_best_split(features, data, metric) | X=(features, data, metric), y=best_feature}

   - data: {age, income, gender, marital status}
   - features: {list of all features}
   - metric: {selection criteria}, metric ϵ {gini, entropy}
   - best_feature: {best feature selected based on the provided metrics}
   - Method to find best split out of all the features

4. f_4 : {create_leaf(data) | X=(data), y=leaf_node}

   - data: {age, income, gender, marital status}
   - leaf_node: {dictonary defining terminating condition of tree}
   - Method to create leaf node that predicts the output class

5. f_5 : {check_purity(data) | X=(data), y=is_pure}

   - data: {age, income, gender, marital status}
   - is_pure: {True, False}
   - Method to check if purity has been achieved

In [7]:

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
import threading
import warnings
warnings.filterwarnings('ignore')
```

In [8]:

```
# Generating dataset
data = {
    'age': ['<21', '<21', '21-35', '>35', '>35', '>35', '21-35', '<21', '<21', '>35', '<21'
    'income':['high','high','high','medium','low','low','low','medium','low','medium','medi
    'gender':['male','male','male','male','female','female','female','male','female','femal
    'marital_status':['single', 'married', 'single', 'single', 'single', 'married', 'marrie
    'buys':['no','no','yes','yes','yes','no','yes','no','yes','yes','yes','yes','yes','no']
}
df = pd.DataFrame.from_dict(data)
df
```

Out[8]:

	age	income	gender	marital_status	buys
0	<21	high	male	single	no
1	<21	high	male	married	no
2	21-35	high	male	single	yes
3	>35	medium	male	single	yes
4	>35	low	female	single	yes
5	>35	low	female	married	no
6	21-35	low	female	married	yes
7	<21	medium	male	single	no
8	<21	low	female	married	yes
9	>35	medium	female	single	yes
10	<21	medium	female	married	yes
11	21-35	medium	male	married	yes
12	21-35	high	female	single	yes
13	>35	medium	male	married	no

In [9]:

```
# Describing the dataset
df.describe()
```

Out[9]:

	age	income	gender	marital_status	buys
count	14	14	14	14	14
unique	3	3	2	2	2
top	>35	medium	male	single	yes
freq	5	6	7	7	9

In [10]:

```
# Calculating value counts for the dataset
for column in df.columns:
    print(column,":\n",df[column].value_counts(),"\n")
```

```
age :
 >35      5
<21       5
21-35     4
Name: age, dtype: int64

income :
 medium    6
high      4
low       4
Name: income, dtype: int64

gender :
 male      7
female    7
Name: gender, dtype: int64

marital_status :
 single     7
married    7
Name: marital_status, dtype: int64

buys :
 yes    9
no     5
Name: buys, dtype: int64
```

What is categorical data? (Bloom's Taxonomy: Analyze)

Categorical variables represent types of data which may be divided into groups such that the intersection of all these groups forms a null set. For example consider the feature income this feature can be seperated into three mutually exclusive sets: {low, medium, high} such that:

$$low \cap medium \cap high = \Phi$$

Feature	Type	Category	Preprocessing
age	Categorical	<21, 21 <= age <= 35, >35	T
income	Categorical	low, medium, high	T
gender	Categorical	male, female	F
marital status	Categorical	married, single	F
buys	Output Label(Binary)	yes, no	F

Categorical features with unique values > 2 need preprocessing which can be done using one hot encoding or label encoding.
Introduction of new features after preprocessing, using one hot encoding:

Feature	New Features after preprocessing
age:	age_<21 (for age <21)
	age_21-35 (for 21<= age <=35)
	age_>35 (for age >35)
income:	income_low (for low income)
	income_medium (for medium income)
	income_high (for high income)

In [11]:

```
# Function to encode data with unique values > 2
def encode_data(df, target):
    '''
    params: {df, exclude}
    df: Input data
    target: name of the output label

    returns: {df}
    df: One hot encoded data
    '''
    cat_cols = []
    for column in df.columns:
        if (df[column].dtype == "object") and (column != target): # Check if the column is
            if len(df[column].unique()) > 2: # Check if number of unique values is greater
                cat_cols.append(column)
                features = df[column].value_counts().index.tolist()
                # For every unique value create a new column
                for key in features:
                    col_name = f"{column}_{key}"
                    df[col_name] = 0
                    df.loc[df[column] == key, col_name] = 1
    df.drop(columns=cat_cols,inplace=True)
    return df
```

In [12]:

```
df_encoded = encode_data(df, 'buys')
df_encoded
```

Out[12]:

	gender	marital_status	buys	age_>35	age_<21	age_21-35	income_medium	income_high	ir
0	male	single	no	0	1	0	0	1	
1	male	married	no	0	1	0	0	1	
2	male	single	yes	0	0	1	0	1	
3	male	single	yes	1	0	0	1	0	
4	female	single	yes	1	0	0	0	0	
5	female	married	no	1	0	0	0	0	
6	female	married	yes	0	0	1	0	0	
7	male	single	no	0	1	0	1	0	
8	female	married	yes	0	1	0	0	0	
9	female	single	yes	1	0	0	1	0	
10	female	married	yes	0	1	0	1	0	
11	male	married	yes	0	0	1	1	0	
12	female	single	yes	0	0	1	0	1	
13	male	married	no	1	0	0	1	0	

In [13]:

```
# Function for preprocessing given data
def preprocess_data(df):
    '''
    params: {df}
    df: Input data

    returns: {df}
    df: Preprocessed data
    '''
    df_encoded = encode_data(df, 'buys') # One hot encode the given data
    for col in df.columns:
        if (df[col].dtype == "object") and (len(df[col].unique()) == 2):
            # Map binary values with (0,1)
            unique_values = df[col].unique()
            df_encoded[col]=df_encoded[col].map({f'{unique_values[0]}':0, f'{unique_values[
            print(f"For column {col} class {unique_values[0]} maps to 0 and class {unique_v
    return df_encoded
```

In [14]:

```
df_encoded = preprocess_data(df.copy())
```

```
For column gender class male maps to 0 and class female maps to 1
For column marital_status class single maps to 0 and class married maps to 1
For column buys class no maps to 0 and class yes maps to 1
```

In [15]:

```
print("\nPreprocessed data: ")
print(df_encoded)
```

```
Preprocessed data:
    gender  marital_status  buys  age_>35  age_<21  age_21-35  income_medium
\
0        0               0     0        0        1          0              0
1        0               1     0        0        1          0              0
2        0               0     1        0        0          1              0
3        0               0     1        1        0          0              1
4        1               0     1        1        0          0              0
5        1               1     0        1        0          0              0
6        1               1     1        0        0          1              0
7        0               0     0        0        1          0              1
8        1               1     1        0        1          0              0
9        1               0     1        1        0          0              1
10       1               1     1        0        1          0              1
11       0               1     1        0        0          1              1
12       1               0     1        0        0          1              0
13       0               1     0        1        0          0              1

    income_high  income_low
0             1           0
1             1           0
2             1           0
3             0           0
4             0           1
5             0           1
6             0           1
7             0           0
8             0           1
9             0           0
10            0           0
11            0           0
12            1           0
13            0           0
```

Decision Tree:

A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition based on the attribute value. Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example, this approach is called a Top-Down approach. Each node in the tree acts as a test

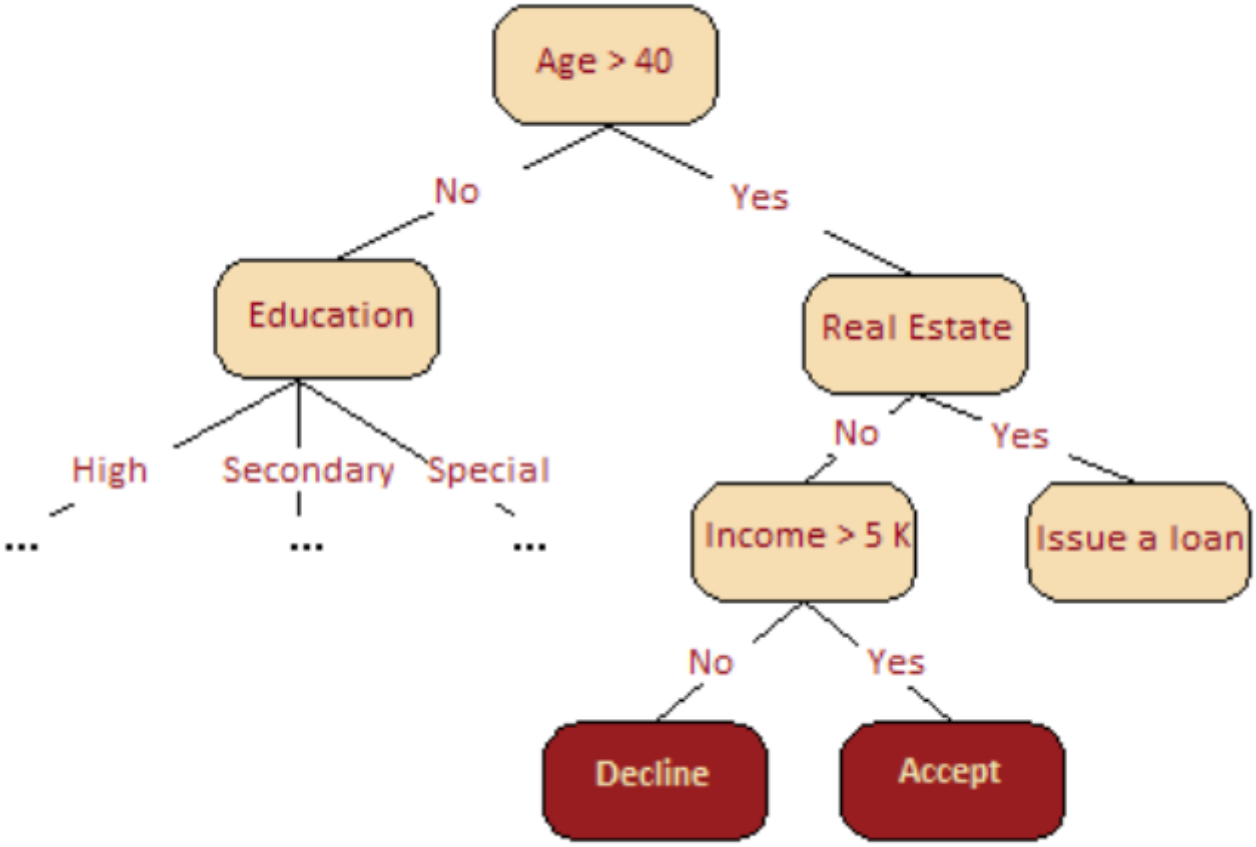
case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive and is repeated for every subtree rooted at the new nodes, as the tree is partitioned in a recursive manner the process is called recursive partitioning.

Can B trees or B+ trees be used to make decision trees:

- It is possible to implement decision trees as B or B+ trees using a criteria like Chi square which performs multiway splits.
- Entropy and gini index cannot be used in such cases as they perform binary splits.

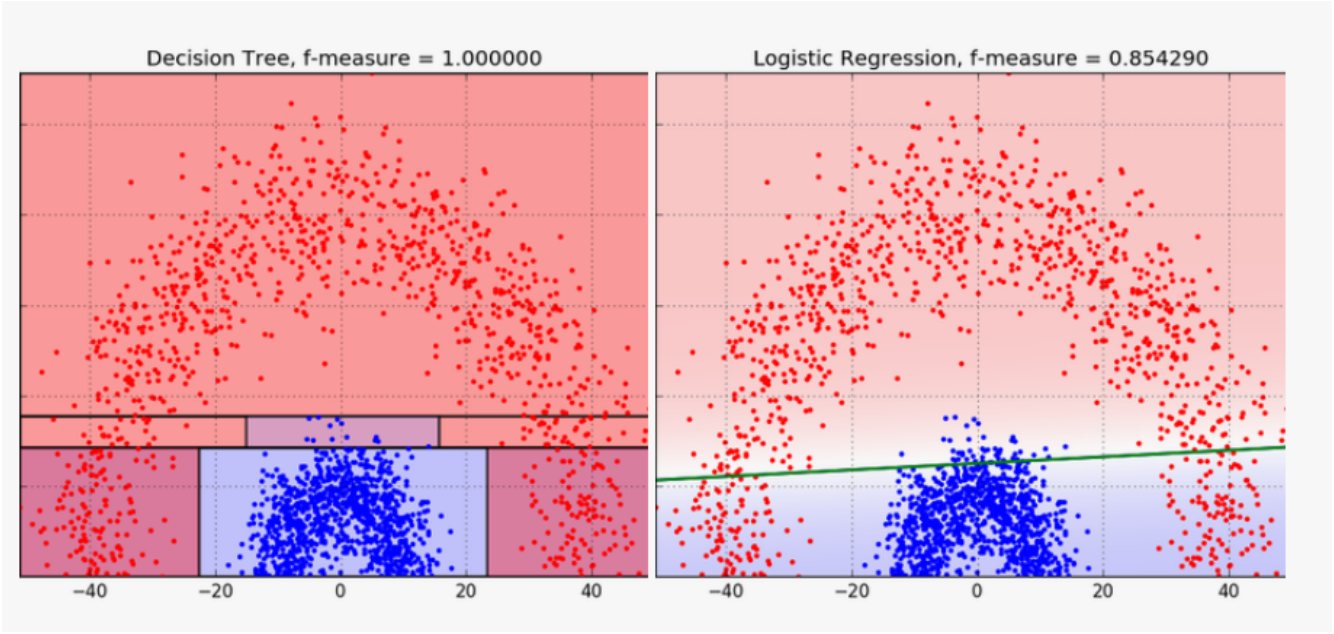
Why decision tree? (Bloom's Taxonomy: Analyze)

- A decision tree is often a generalization of the experts' experience, a means of sharing knowledge of a particular process. For example, before the introduction of scalable machine learning algorithms, the credit scoring task in the banking sector was solved by experts. The decision to grant a loan was made on the basis of some intuitively (or empirically) derived rules that could be represented as a decision tree.



- Decision Trees are simple to understand and interpret. For example, using the above scheme, the bank can explain to the client why they were denied for a loan: e.g the client does not own a house and her income is less than 5,000. Many other models, although more accurate, do not have this property and can be regarded as more of a "black box" approach, where it is harder to interpret how the input data was transformed into the output. Due to this "understandability" and similarity to human decision-making.
- Requires little data preparation: Other techniques often require data normalization. Since trees can handle qualitative predictors, there is no need to create dummy variables.

- Decision Trees bisect the space into smaller and smaller regions, whereas Logistic Regression fits a single line to divide the space exactly into two.



How is the root of a decision tree selected? (Bloom's Taxonomy: Apply)

- Recursively partition the input space and define a local split condition in each resulting region of input space by selecting the best attribute i.e the attribute which has the most information gain, a measure that expresses how well an attribute splits the data into groups based on classification.

Attribute Selection Measures

Entropy

Shannon's entropy is defined for a system with N possible states as follows:

$$S = -\sum_{i=1}^{N} p_i \log_2 p_i,$$

where p_i is the probability of finding the system in the i-th state. Entropy can be described as the degree of chaos in the system. The higher the entropy, the less ordered the system and vice versa. Since entropy is, in fact, the degree of chaos (or uncertainty) in the system, the reduction in entropy is called **information gain (IG)**. Formally, the information gain for a split based on the variable Q is defined as

$$IG(Q) = S_O - \sum_{i=1}^{q} \frac{N_i}{N} S_i,$$

where q is the number of groups after the split, N_i is number of objects from the sample in which variable Q is equal to the i-th value.

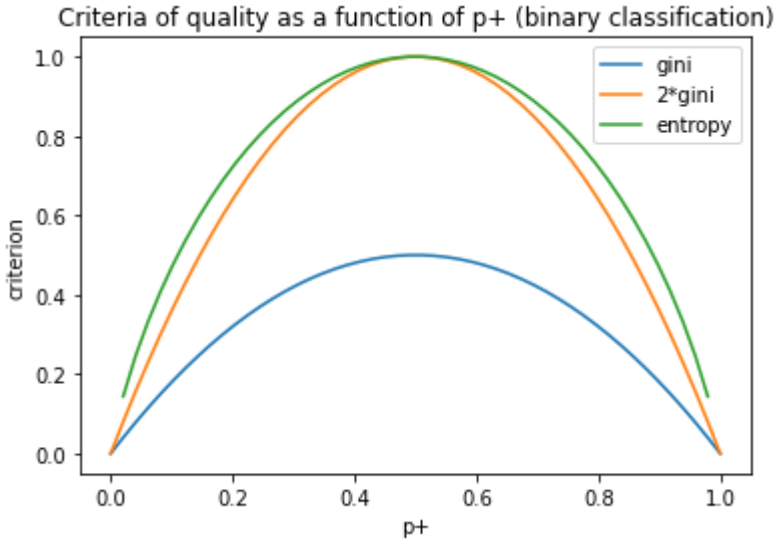
Gini uncertaintv:

$$G = 1 - \sum_k (p_k)^2$$

Maximizing this criterion can be interpreted as the maximization of the number of pairs of objects of the same class that are in the same subtree. In practice Gini uncertainty and information gain work similarly.

In [16]:

```
plt.figure(figsize=(6, 4))
xx = np.linspace(0,1,50)
plt.plot(xx, [2 * x * (1-x) for x in xx], label='gini')
plt.plot(xx, [4 * x * (1-x) for x in xx], label='2*gini')
plt.plot(xx, [-x * np.log2(x) - (1-x) * np.log2(1 - x)  for x in xx], label='entropy')
plt.xlabel('p+')
plt.ylabel('criterion')
plt.title('Criteria of quality as a function of p+ (binary classification)')
plt.legend();
```



In [17]:

```
# Function to calculate entropy of selected feature
def entropy(feature, data):
    '''
    params: {feature, data}
    feature: Name of feature
    data: Input data

    returns: {entropy}
    entropy: Entropy of the given feature
    '''
    weighted_entropy = 0
    if len(data) == 0:
        return 0
    else: total_count = len(data)

    unique_target_values = data['buys'].unique()
    unique_feature_values = data[feature].unique()
    for i in unique_feature_values:
        # For each unique value in selected feature
        entropy = 0
        feature_value_count = data[feature].value_counts()[i]
        for j in unique_target_values:
            # For each unique value in target label
            count = len(data[(data[feature] == i) & (data['buys'] == j)])
            # Calculate probability of getting feature as i and target label as j
            if count == 0: continue
            probability = count/feature_value_count
            entropy = entropy + probability*np.log2(probability)
        feature_probability = feature_value_count/total_count
        weighted_entropy = weighted_entropy + feature_probability*entropy
    weighted_entropy = -weighted_entropy
    print(f'feature = {feature} - {weighted_entropy}')
    return weighted_entropy
```

In [18]:

```
# Function to calculate gini index of selected feature
def gini_index(feature, data):
    '''
    params: {feature, data}
    feature: Name of feature
    data: Input data

    returns: {gini_index}
    gini_index: Weighted gini index of the given feature
    '''
    gini_index = 0
    if len(data) == 0:
        return 0
    else: total_count = len(data)

    unique_target_values = data['buys'].unique()
    unique_feature_values = data[feature].unique()
    for j in unique_target_values:
        # For each unique value in target label
        target_value_count = data['buys'].value_counts()[j]
        probability2_sum = 0
        for i in unique_feature_values:
            # For each unique value in selected feature
            count = len(data[(data[feature] == i) & (data['buys'] == j)])
            # Calculate probability of getting feature as i and target label as j
            probability = count/target_value_count
            probability2_sum = probability2_sum + probability**2
        # Calculate probability of getting target variable i
        target_probability = target_value_count/total_count
        # Calculate weighted gini index
        gini_index = gini_index + (1 - probability2_sum)*(target_probability)
    print(f'feature = {feature} - {gini_index}')
    return gini_index
```

In [19]:

```
# Function to determing the best split
def find_best_split(features, data, metric):
    '''
    params: {feature, data, metric}
    feature: Name of feature
    data: Input data
    metric: Criteria for attribute selection

    returns: {best_feature}
    best_feature: Name of feature selected according to the metric
    '''

    print('Available features =', len(features))
    print('Number of datapoints =', len(data))
    best_feature = ''
    best_metric = 1
    # For each feature calculate the best feature by minimizing the selected metric value
    for feature in features:
        if metric=='gini':
            curr_criteria = gini(feature, data)
        elif metric=='entropy':
            curr_criteria = entropy(feature, data)
        if curr_criteria < best_metric:
            best_feature=feature
            best_metric=curr_criteria

    print(f'Best feature: {best_feature} with {metric} = {best_metric}')
    return best_feature
```

In [20]:

```
# Function to create a leaf node
def create_leaf(data):
    '''
    params: {data}
    data: Input data

    returns: {leaf}
    leaf: dictonary defining terminating condition of tree
    '''
    # Initializing the leaf
    leaf = {'splitting_feature' : None,
            'left' : None,
            'right' : None,
            'is_leaf': True,
            'size': len(data)}
    # Setting leaf value
    target_value_counts = data['buys'].value_counts()
    target_unique_values = data['buys'].unique()
    predicted_value = target_unique_values[0]
    for i in target_unique_values:
        if target_value_counts[i] > target_value_counts[predicted_value]:
            predicted_value = i
    leaf['prediction'] = predicted_value
    return leaf
```

In [21]:

```
# Function to check purity of the data
def check_purity(data):
    '''
    params: {data}
    data: Input data

    returns: {is_pure}
    is_pure: {True if purity has been achieved, else False}
    '''
    # A node having multiple classes is impure whereas a node having only one class is pure
    target_value_counts = data['buys'].value_counts()
    target_unique_values = data['buys'].unique()
    for i in target_unique_values:
        if target_value_counts[i] == 0:
            return True
    return False
```

In [22]:

```
# Recursive function to create the decision tree based on input data and parameters
def create_decision_tree(rem_features, data, curr_depth, metric, max_depth):
    '''
    params: {feature, data, curr_depth, metric, max_depth}
    feature: Name of feature
    data: Input data
    curr_depth: Current depth of the tree
    metric: Criteria for attribute selection
    max_depth: Maximum depth of the tree

    returns: {node}
    node: Dictionary representing the root node in the decision tree
    '''
    if rem_features == []:
        return create_leaf(data)

    if check_purity(data):
        return create_leaf(data)

    if curr_depth>=max_depth:
        return create_leaf(data)

    if len(data)<3:
        return create_leaf(data)

    # Select best feature for each level
    print('\nFor level', curr_depth)
    selected_feature = find_best_split(rem_features, data, metric)

    # Remove the selected best feature from the remaining features list
    rem_features.remove(selected_feature)
    left_split=data[data[selected_feature]==0]
    right_split=data[data[selected_feature]==1]

    # Check if all data points belong to the same class
    # Create a leaf node if the condition is true
    if len(left_split) == len(data):
        return create_leaf(left_split)

    if len(right_split) == len(data):
        return create_leaf(right_split)

    # Create decision tree for left split and right split
    left=create_decision_tree(rem_features, left_split, curr_depth+1, metric, max_depth)
    right=create_decision_tree(rem_features, right_split, curr_depth+1, metric, max_depth)

    return {'is_leaf'          : False,
            'prediction'       : None,
            'splitting_feature': selected_feature,
            'left'             : left,
            'right'            : right,
            'size'             : len(data)}
```

In [23]:

```
# Function to print the decision tree
def print_tree(tree, level, store):
    '''
    params: {tree, level, store}
    tree: Decision tree to be printed
    level: Starting level of the decision tree
    store: Dictionary storing traversed decision tree

    returns: None
    '''
    if not tree:
        return
    split_name = tree['splitting_feature']
    dashes='-'*level*3

    string=''
    if tree['is_leaf']==False:
        print(dashes, store)
    else:
        if tree['size']>0:
            print(dashes, store, " PREDICTION: ", tree['prediction'], 'size=', tree['size']

    left_store=store.copy()
    left_store[split_name]=0
    print_tree(tree['left'], level+1, left_store)

    right_store=store.copy()
    right_store[split_name]=1
    print_tree(tree['right'], level+1, right_store)
```

In [24]:

```
def build_tree(df_encoded, max_depth, metric):
    '''
    params: {df_encoded, max_depth, metric}
    df_encoded: Encoded and preprocessed input data
    metric: Criteria for attribute selection
    max_depth: Maximum depth of the tree

    returns: None
    '''
    features = list(df_encoded.columns)
    features.remove('buys')
    decision_tree=create_decision_tree(features, df_encoded, 0, metric, max_depth)
    print(f"\nDecision Tree:\nSelected criteria: {metric}\nMax depth: {max_depth}\n--------
    print_tree(decision_tree, 0, {})
    return decision_tree
```

In [25]:

```
metric =  'entropy'
max_depth = 3
decision_tree = build_tree(df_encoded, max_depth, metric)
```

```
For level 0
Available features = 8
Number of datapoints = 14
feature = gender - 0.7884504573082896
feature = marital_status - 0.9241743523004413
feature = age_>35 - 0.9371011056259821
feature = age_<21 - 0.8380423950607804
feature = age_21-35 - 0.7142857142857143
feature = income_medium - 0.9389462162661898
feature = income_high - 0.9152077851647805
feature = income_low - 0.9253298887416583
Best feature: age_21-35 with entropy = 0.7142857142857143

For level 1
Available features = 7
Number of datapoints = 10
feature = gender - 0.7219280948873623
feature = marital_status - 0.9709505944546686
feature = age_>35 - 0.9709505944546686
feature = age_<21 - 0.9709505944546686
feature = income_medium - 0.9709505944546686
feature = income_high - 0.7635472023399721
feature = income_low - 0.965148445440323
Best feature: gender with entropy = 0.7219280948873623

For level 2
Available features = 6
Number of datapoints = 5
feature = marital_status - 0.5509775004326937
feature = age_>35 - 0.4
feature = age_<21 - 0.4
feature = income_medium - 0.5509775004326937
feature = income_high - 0.5509775004326937
feature = income_low - 0.7219280948873623
Best feature: age_>35 with entropy = 0.4

For level 2
Available features = 5
Number of datapoints = 5
feature = marital_status - 0.5509775004326937
feature = age_<21 - 0.5509775004326937
feature = income_medium - 0.5509775004326937
feature = income_high - 0.7219280948873623
feature = income_low - 0.5509775004326937
Best feature: marital_status with entropy = 0.5509775004326937

For level 1
Available features = 4
Number of datapoints = 4
feature = age_<21 - -0.0
feature = income_medium - -0.0
feature = income_high - -0.0
feature = income_low - -0.0
Best feature: age_<21 with entropy = -0.0
```

```
Decision Tree:
Selected criteria: entropy
Max depth: 3
---------
 {}
--- {'age_21-35': 0}
------ {'age_21-35': 0, 'gender': 0}
--------- {'age_21-35': 0, 'gender': 0, 'age_>35': 0}  PREDICTION:  0 size=
3
--------- {'age_21-35': 0, 'gender': 0, 'age_>35': 1}  PREDICTION:  1 size=
2
------ {'age_21-35': 0, 'gender': 1}
--------- {'age_21-35': 0, 'gender': 1, 'marital_status': 0}  PREDICTION:  1
size= 2
--------- {'age_21-35': 0, 'gender': 1, 'marital_status': 1}  PREDICTION:  1
size= 3
--- {'age_21-35': 1}  PREDICTION:  1 size= 4
```

In [26]:

```python
# Function to perform prediction using decision tree on input values
def predict(tree, X):
    '''
    params: {tree, X}
    tree: Decision tree using which prediction is to be performed
    X: Input value for which prediction is to be performed

    returns: {prediction}
    prediction: {yes, no}
    '''
    if tree['is_leaf']:
        if (tree['prediction']==0):
            return "no"
        return "yes"
    else:
        split_feature_value = X[tree['splitting_feature']]
        if split_feature_value == 0:
            return predict(tree['left'], X)
        else:
            return predict(tree['right'], X)
```

In [27]:

```
# Function to take input values from the user
def get_data(input_columns):
    input_dict = {}
    flag = True
    while flag:
        age = int(input('''
Select age category:
1. age < 21
2. 21 <= age <=35
3. age > 35
        '''))
        if age not in [1,2,3]:
            print("Please select a valid age category!")
        else:
            if age == 1:
                input_dict['age_<21'] = 1
            elif age == 2:
                input_dict['age_21-35'] = 1
            elif age == 3:
                input_dict['age_>35'] = 1
            flag = False

    flag = True
    while flag:
        income = int(input('''
Select income category:
1. low
2. medium
3. high
        '''))
        if income not in [1,2,3]:
            print("Please select a valid income category!")
        else:
            if income == 1:
                input_dict['income_low'] = 1
            elif income == 2:
                input_dict['income_medium'] = 1
            elif income == 3:
                input_dict['income_high'] = 1
            flag = False

    flag = True
    while flag:
        marital_status = input('Are you married(y,n): ')
        if marital_status.lower() not in ['y','n']:
            print("Please answer the question with: (y/n)!")
        else:
            if marital_status.lower() == 'y':
                input_dict['marital_status'] = 1
            else:
                input_dict['marital_status'] = 0
            flag = False

    flag = True
    while flag:
        gender = int(input('''
Select gender category:
1. male
2. female
```

```
        '''))
        if gender not in [1,2]:
            print("Please select a valid gender category!")
        else:
            if gender == 1:
                input_dict['gender'] = 0
            elif gender == 2:
                input_dict['gender'] = 1
            flag = False

    print("Provided input: ", input_dict)
    for col in input_columns:
        if col not in input_dict:
            input_dict[col] = 0
    return input_dict
```

In [28]:

```
columns = df_encoded.drop(columns=['buys']).columns
x = get_data(columns)
input_val = x.copy()
x = pd.Series(x)
print("Prediction: ",predict(decision_tree, x))
```

```
Select age category:
1. age < 21
2. 21 <= age <=35
3. age > 35
        1

Select income category:
1. low
2. medium
3. high
        2
Are you married(y,n): n

Select gender category:
1. male
2. female
        2
Provided input:  {'age_<21': 1, 'income_medium': 1, 'marital_status': 0, 'ge
nder': 1}
Prediction:  yes
```

In [29]:

```
print(input_val)
```

```
{'age_<21': 1, 'income_medium': 1, 'marital_status': 0, 'gender': 1, 'age_>3
5': 0, 'age_21-35': 0, 'income_high': 0, 'income_low': 0}
```

Sklearn Implementation

In [30]:

```
X = df_encoded.drop(columns=['buys']).to_numpy()
y = df_encoded[['buys']].to_numpy()
```

In [31]:

```
classifier = DecisionTreeClassifier()
classifier.fit(X,y)
y_pred = classifier.predict(X)
y_pred = ['yes' if y==1 else 'no' for y in y_pred]
print(y_pred)
```

```
['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no', 'yes', 'yes', 'yes', 'y
es', 'yes', 'no']
```

In [32]:

```
def prep_data(input_val, columns):
    print("Provided input: ", input_val)
    df = pd.DataFrame(input_val, index=[0])
    df = df[columns]
    return df.to_numpy()
```

In [33]:

```
input_x = prep_data(input_val, columns)
print(input_x)
```

```
Provided input:  {'age_<21': 1, 'income_medium': 1, 'marital_status': 0, 'ge
nder': 1, 'age_>35': 0, 'age_21-35': 0, 'income_high': 0, 'income_low': 0}
[[1 0 0 1 0 1 0 0]]
```

In [208]:

```
if classifier.predict(input_x)[0] == 0: print("\nPrediction: no")
else: print("\nPrediction: yes")
```

```
Provided input:  {'age_<21': 1, 'income_low': 1, 'marital_status': 1, 'gende
r': 1, 'age_>35': 0, 'age_21-35': 0, 'income_medium': 0, 'income_high': 0}

Prediction: yes
```

Conclusion:

Decision tree classification was used to find the optimum fit for the given dataset.