1- preprocessing:

➔ First, I read the data using the pandas library with pd.read_csv(file_path). Then, using dataframe.info(), I checked whether any columns have null values and how many null values each column has.

➔

➔ Some columns in some rows were actually missing, but they were not saved as NaN; instead, they were labeled as ‘unknown’. To see each ‘worthless’ value in each column, I used the unique_column function, which prints each unique value of each column.

➔

➔ I found out these anomalies:

➔

➔ Age 999 and negative age like -1

➔ ‘Unknown’ value in the job column which means NaN

➔ ‘Unknown’ in education

➔ ‘Unknown’ value in contact

➔ Negative values in pdays

➔ ‘Unknown’ poutcome

➔ I replaced ‘unknown’ with NaN in every column. After analyzing the dataframe, I discovered that 37,369 rows contain at most one NaN value. Given the substantial number of affected rows, removing them entirely would not be prudent. Instead, I opted to eliminate columns with the highest frequency of NaN values, specifically ‘contact’ and ‘poutcome’. It’s important to note that these features may not significantly contribute to our predictive model, so their removal is unlikely to adversely impact the model’s performance.

➔

➔ Now, we need to address the categorical data within our dataframe. For the ‘default’, ‘housing’, ‘loan’, and ‘y’ columns, I mapped their values to 0 and 1. Some features are not sequential, and for these, I applied one-hot encoding to facilitate further analysis.

2-Decision tree class implementation:

I created two class Node and decision tree class. Our decision tree class has this methods:

_entropy:

Input: a list
Output: entropy of that list

Entropy is a measure of the disorder or randomness in a system. Formula: $\sum p \times log2\, p$

_gini:

Input: a list
Output: gini index of that list

the Gini index measures the impurity of D, a data partition or set of training tuples, as

Formula: $Gini(D) = 1 - \sum p_i^2$

where pi is the probability that a tuple in D belongs to class Ci.

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition.

*Information*_gain:

Input: `parent, left_child, right_child each one as a list`
Output: information gain or gini index

function do two works based on the criterion was gini or entropy:
terion was entropy this function calculates the information gain which is:

$$_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j). \qquad Gain(A) = Info(D) - Info_A(D).$$

_best_split:

Input: X which is dataframe whithout target column and y which is target column(type of this input is numpy array)
Output: output about best split of each feature including: best_split = {

```
'feature_index',
'threshold',
'df_left',
'df_right',
'Gain', / gini index
}
```

At each level, for every remaining feature and for each unique value of that feature, we split the dataset into left and right datasets based on whether they are greater than or less than the chosen

split value. Then, we calculate the information gain or Gini index for these subsets and update the best_info_gain and best_split values accordingly.

_build:

Input: X, y and depth
Output: Node or leaf node
Computing the best split at each depth, then recursively building the left and right children of the tree.

Fit:

Input: X, y and depth
Output: Node or leaf node
Building the tree from root node .

Predict:

Input: X: np.array, features
Output: np.array, predicted classes
Function used to classify new instances.

print_tree:

3-model validation:

Cross validation:

Procedure: The dataset is divided into ‘k’ equal-sized subsets or folds.
Training and Validation: The model is trained on ‘k-1’ folds and validated on the remaining fold.
Iteration: This process is repeated ‘k’ times, each time with a different fold as the validation set.
Result: The results from each fold are averaged to estimate the model’s generalization performance

In this code we used 5 folds.

Accuracy:

The accuracy function for classification is a metric that measures the proportion of correct predictions made by a model out of all predictions made. It’s calculated as the number of correct predictions divided by the total number of predictions.

Mse:

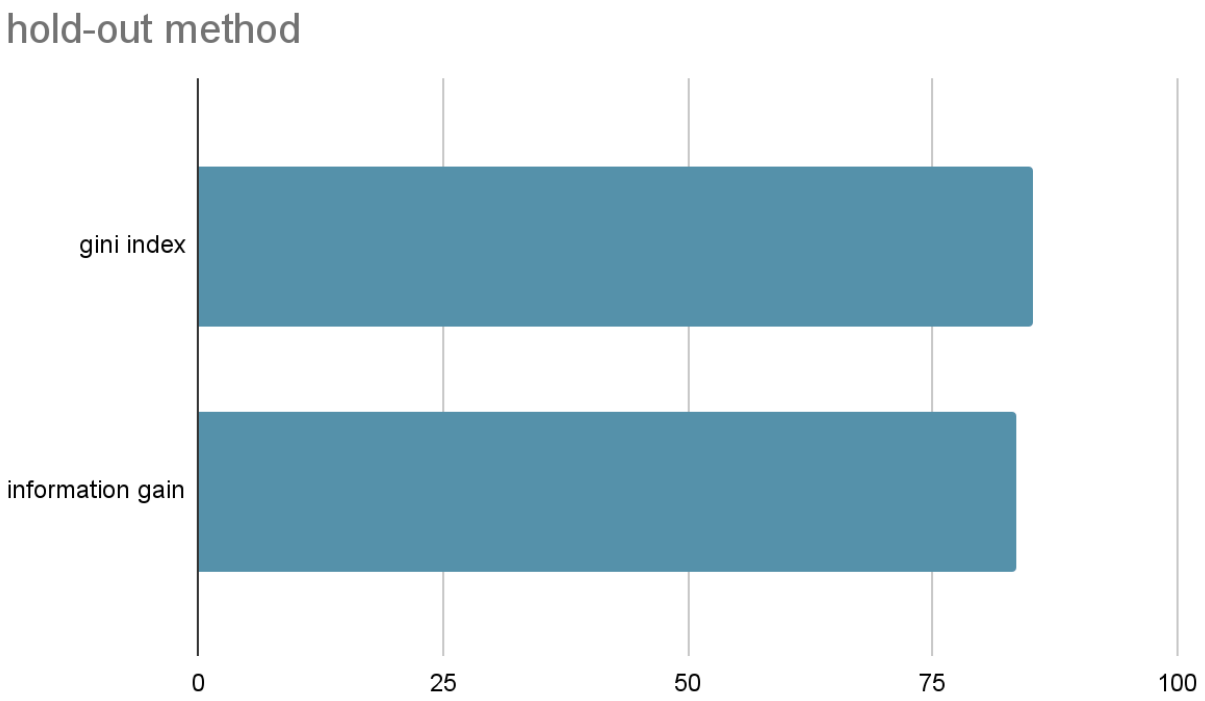
It calculates the average of the squares of the errors, which is the average squared difference between the estimated values and the actual value.

This evaluation metric is not used for classification problems, so I don't know why the TAs asked us to implement this metric! In any case, there was no need for such a metric.

4-Analysis and Review of Results:

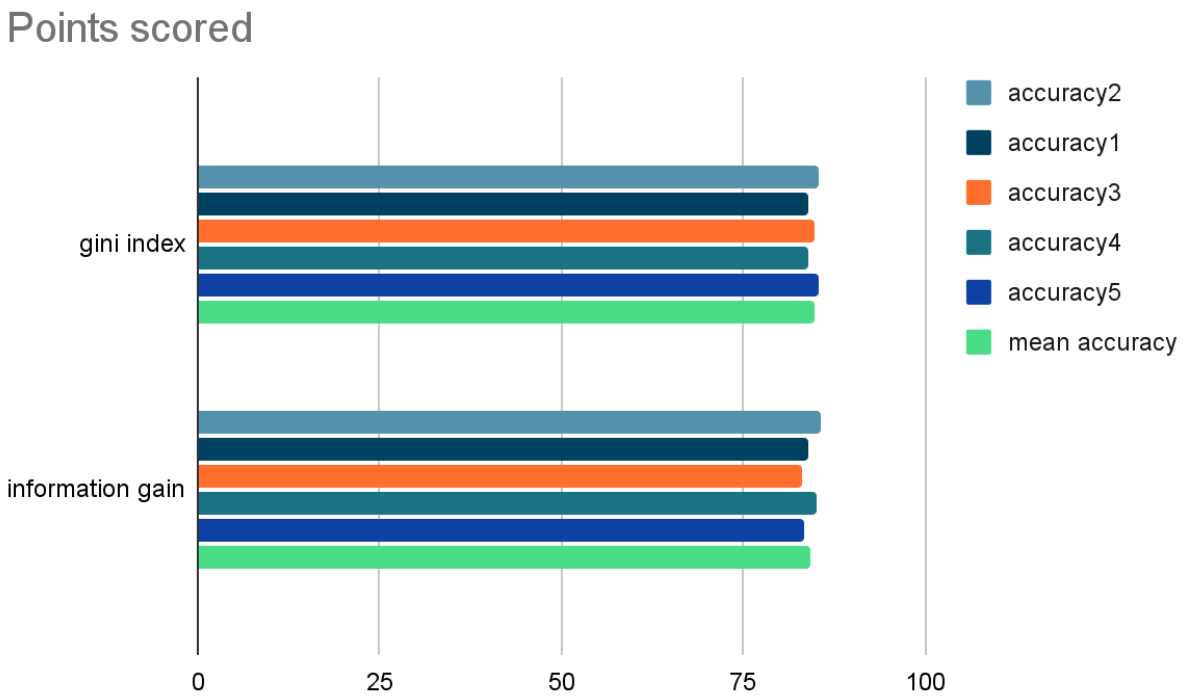
After preprocessing the data as explained, we divided the data into two sets: train and test. We set aside 20% of the data as training data. Then, we classified this data once using the information gain criterion and once with the gini index. The evaluation of the output of these two trees was as follows:

	accuracy	mse
gini index	85.15476943	235
information gain	83.57548958	260



We trained the model again, this time using the cross-validation function, dividing the data into 5 folds. The output was as follows:

	accuracy1	accuracy2	accuracy3	accuracy4	accuracy5	mean accuracy
gini index	83.82817435	85.33501896	84.82932996	84.00758534	85.33501896	84.66702552
information gain	83.95451674	85.52465234	83.05941846	85.08217446	83.31226296	84.18660499



By the way, we can easily conclude that we don't have overfitting. This is because, according to the results of hold-out and cross-validation methods, we observe a high accuracy for predicting unseen data by the model.

To increase the accuracy of the model, the following steps can be applied:

- Investigate the correlation of features present in the initial data and remove those that have little impact on the final label.
- Add columns derived from existing columns.
- Increase the existing data or do not remove them in a way that we fill the missing data with suggested values instead of deleting them.
- Prune the decision tree.

Each of the above items will not necessarily lead to an increase in accuracy.!!