# Lab Assignment Report

**Assignment**

**CS503**: Machine Learning


| Name | Entry No. |
|---|---|
| Anshika | 2021CSB1069 |

Instructor: Dr. Santosh Vipparthi

Teaching Assistant: Nishchala Thakur


**Indian Institute of Technology Ropar**
**Punjab, India**
**April 5, 2025**

# Contents

float

# 1 | Developing a Random Forest

A Random Forest is a collection of Decision Trees (Just like a forest). The final output of the random forest is basically based on the output of its consisting decision trees.
**Attended the Classes**

## 1.1 | The Necessary EDA

### 1.1.1 | Dataset Loading and Naming Columns

- The dataset is loaded from CSV files: `adulttrain.csv` and `adulttest.csv`.
- Column names are manually provided for better readability.

### 1.1.2 | Dataset Structure and Overview

- `train.shape` and `test.shape` show the number of rows and columns in the training and test datasets.
- `train.head()` to display the **first 5 rows** of the training dataset.
- `train.info()` for a summary of:
  - Number of entries
  - Data types of each column
  - Memory usage
- `train.describe()` gives descriptive statistics (mean, std, min, max, etc.) **only for continuous/numeric columns**.

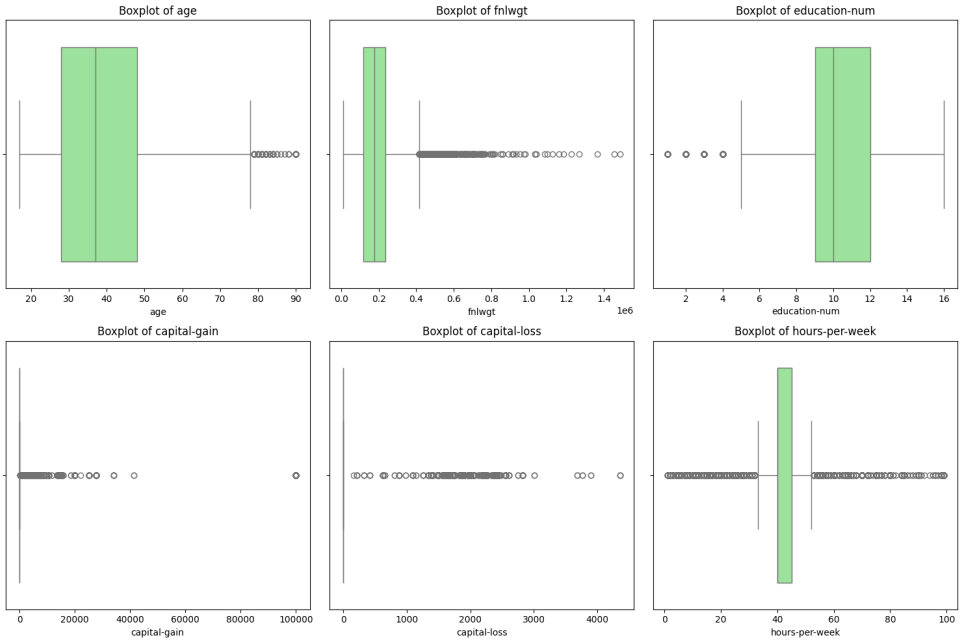
### 1.1.3 | Handling Categorical Features and Missing Values

- Identify all **categorical columns** using `object` data type.
- Count the number of `?` entries (used as placeholders for missing values) in each categorical column.
- Replace all `?` with `np.nan` in both train and test datasets for consistency and proper missing value treatment.

### 1.1.4 | Target Variable Distribution

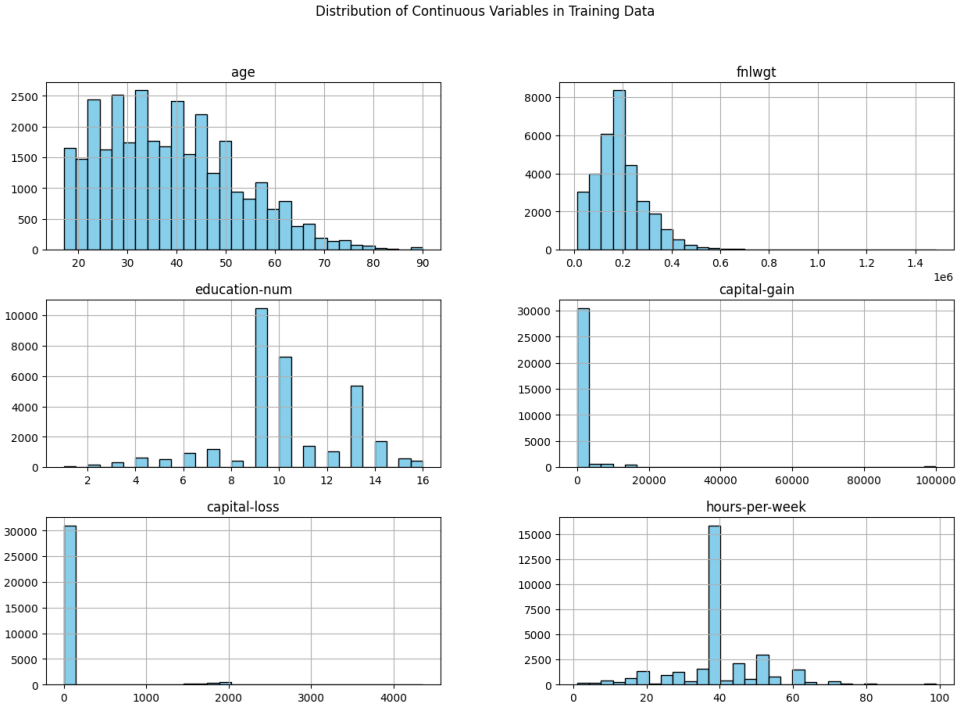
A **countplot** of the `income` column is drawn using `seaborn`, showing the distribution of the target classes:

- `<=50K` and `>50K`
- Helps check for class imbalance.



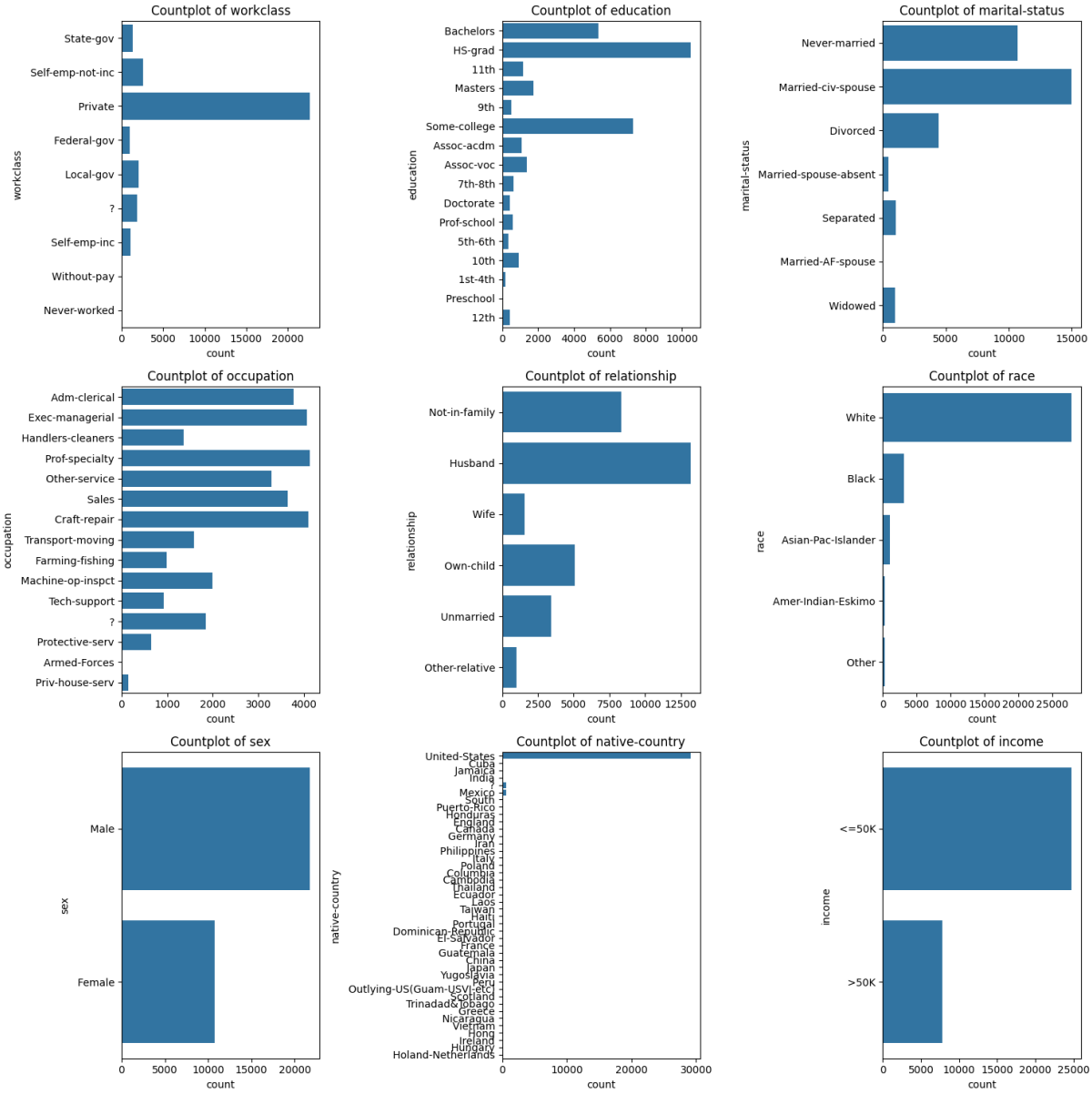

### 1.1.5 | Univariate Analysis of Continuous Variables

- **Histograms** for continuous features:
  - Age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week
  - Gives insights into skewness, modality, and range of values.
- **Boxplots** for the same continuous variables: Useful for identifying **outliers**, median, interquartile range, and spread.




### 1.1.6 | Univariate Analysis of Categorical Variables

- Uses **countplots** for every categorical column.




- Helps in understanding the frequency distribution of each category.
- Plotted in a grid layout using subplots for better visual analysis.

## 2 | Implement a decision tree for the above classification problem

- The decision tree is implemented from scratch using concepts of **entropy** and **information gain** to determine the best splits.
- **Entropy** is calculated as a measure of impurity in the dataset:

$$H(y) = -\sum p_i \log_2(p_i)$$

where $p_i$ are the class probabilities in the label vector $y$.

- **Information Gain** is used to decide the effectiveness of a split:

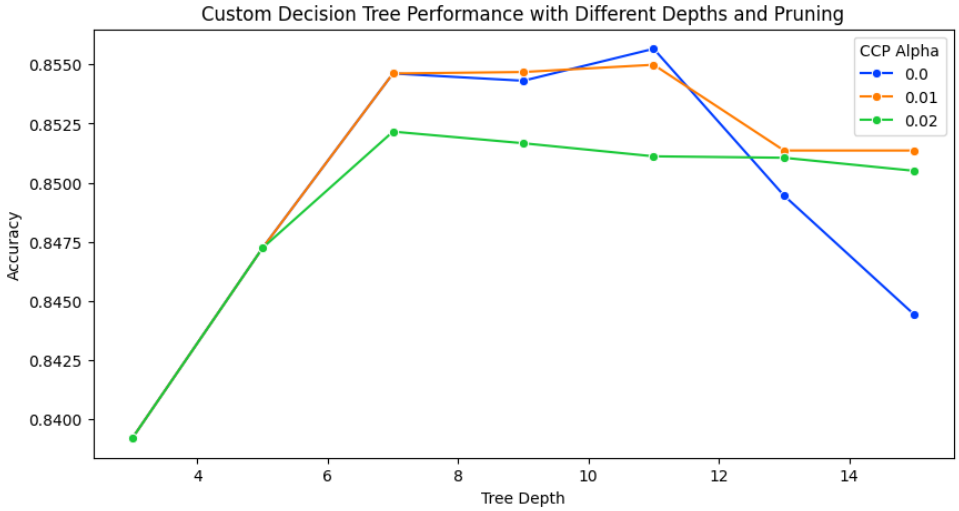
$$IG = H(y) - \left(\frac{n_{left}}{n} H(y_{left}) + \frac{n_{right}}{n} H(y_{right})\right)$$

where $n$ is the total number of samples, and $H(y_{left})$ and $H(y_{right})$ are the entropies of the child nodes.

- The algorithm searches for the **best feature** and **threshold** to split the data by iterating over all features and possible thresholds (unique values of features).
- A **Node class** is used to represent each node in the tree, with attributes indicating whether it is a leaf, its prediction (if any), the splitting feature, threshold, and links to its left and right child nodes.
- The **DecisionTree class** manages the training and prediction. It uses recursive splitting in the `_build_tree` method.
- **Stopping conditions** for the recursion include:
  - All labels in a node are the same.
  - The maximum depth (controlled by `max_depth`) is reached.
  - The information gain of the best split is less than a `pruning_threshold`.
- If a node is terminal (a leaf), it stores the most common class in the node as its prediction.
- During prediction, a new sample is passed through the tree using the learned splits until a leaf is reached, and the leaf's prediction is returned.
- The tree supports **basic post-pruning** by avoiding splits with very low information gain, controlled by the `pruning_threshold` parameter.
- Overall, the implementation supports:
  - Binary classification
  - Recursive tree construction with depth and gain-based stopping
  - Custom splitting criteria
  - Simple post-pruning mechanism

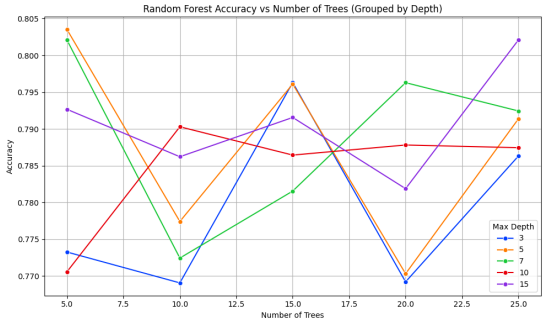
| Tree | Depth | CCP Alpha | Accuracy | Precision | Recall | F1-Score (Above 50K) |
|---|---|---|---|---|---|---|
| 0 | 3.0 | 0.01 | 0.839199 | 0.760611 | 0.465939 | 0.577878 |
| 1 | 3.0 | 0.02 | 0.839199 | 0.760611 | 0.465939 | 0.577878 |
| 2 | 3.0 | 0.00 | 0.839199 | 0.760611 | 0.465939 | 0.577878 |
| 3 | 5.0 | 0.01 | 0.847245 | 0.769750 | 0.504160 | 0.609269 |
| 4 | 5.0 | 0.02 | 0.847245 | 0.769750 | 0.504160 | 0.609269 |
| 5 | 5.0 | 0.00 | 0.847245 | 0.769750 | 0.504160 | 0.609269 |
| 6 | 7.0 | 0.01 | 0.854616 | 0.792639 | 0.520801 | 0.628589 |
| 7 | 7.0 | 0.02 | 0.852159 | 0.780288 | 0.520801 | 0.624669 |
| 8 | 7.0 | 0.00 | 0.854616 | 0.792639 | 0.520801 | 0.628589 |
| 9 | 9.0 | 0.01 | 0.854677 | 0.772459 | 0.545502 | 0.639439 |
| 10 | 9.0 | 0.02 | 0.851668 | 0.761609 | 0.541602 | 0.633034 |
| 11 | 9.0 | 0.00 | 0.854309 | 0.770558 | 0.545762 | 0.638965 |
| 12 | 11.0 | 0.01 | 0.854984 | 0.715280 | 0.641446 | 0.676354 |
| 13 | 11.0 | 0.02 | 0.851115 | 0.755755 | 0.546282 | 0.634168 |
| 14 | 11.0 | 0.00 | 0.855660 | 0.760083 | 0.568383 | 0.650402 |
| 15 | 13.0 | 0.01 | 0.851360 | 0.694702 | 0.661466 | 0.677677 |
| 16 | 13.0 | 0.02 | 0.851053 | 0.755300 | 0.546542 | 0.634183 |
| 17 | 13.0 | 0.00 | 0.849456 | 0.715611 | 0.601924 | 0.653862 |
| 18 | 15.0 | 0.01 | 0.851360 | 0.696527 | 0.657046 | 0.676211 |
| 19 | 15.0 | 0.02 | 0.850501 | 0.752865 | 0.546542 | 0.633323 |
| 20 | 15.0 | 0.00 | 0.844420 | 0.695097 | 0.608164 | 0.648731 |
| 21 | None | 0.01 | 0.845341 | 0.676972 | 0.660426 | 0.668597 |
| 22 | None | 0.02 | 0.849641 | 0.749465 | 0.546022 | 0.631769 |
| 23 | None | 0.00 | 0.811252 | 0.599742 | 0.604264 | 0.601995 |

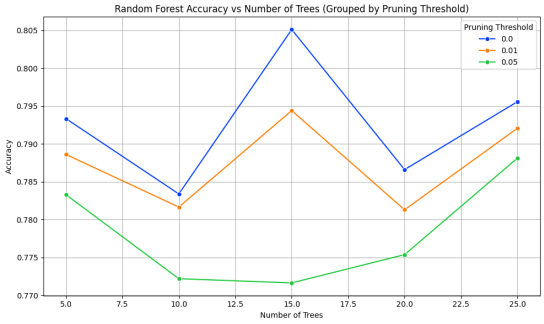
**Table 2.1:** Performance metrics for different decision trees with varying Depth and CCP Alpha




## 3 | Make a Random Forest

- **Class Definition:** The `RandomForest` class is a custom ensemble learning implementation that builds multiple decision trees and aggregates their predictions using majority voting.
- **Constructor Parameters:**
  - `n_trees`: Number of decision trees in the forest (default = 11).
  - `max_depth`: Maximum depth for each decision tree (default = 5).
  - `pruning_threshold`: Threshold used for pruning in the decision tree (default = 0.01).
  - `max_features`: Number of features to consider when looking for the best split. Can be `None`, `'sqrt'`, or an integer.
- **Bootstrap Sampling:** The `_bootstrap_sample` method generates a random sample of the training data (with replacement), used to train each tree.
- **Feature Sampling:** The `_sample_features` method randomly selects a subset of features for each tree:
  - If `max_features = 'sqrt'`, it selects $\sqrt{n\_features}$ features.
  - If it is an integer, it selects that many features.
  - If `None`, all features are used.
- **Training:** The `fit` method:
  1. Bootstraps the dataset.
  2. Samples a subset of features.
  3. Trains a `DecisionTree` instance using the sampled data and features.
  4. Stores the trained tree in the forest.
- **Prediction:** The `predict` method:
  1. Collects predictions from all trees on the input data (each using only its own feature subset).
  2. Uses majority voting to determine the final predicted label for each sample.




## 3.1 | Comparison of Best Decision Tree and Random Forest

| Parameter | Value |
|---|---|
| Depth | 11 |
| CCP Alpha | 0.0 |
| Accuracy | 0.8557 |
| Precision (>50K) | 0.7601 |
| Recall (>50K) | 0.5684 |
| F1-Score (>50K) | 0.6504 |
| Confusion Matrix | $\begin{bmatrix} 11745 & 690 \\ 1660 & 2186 \end{bmatrix}$ |

**Table 3.1:** Performance metrics of the best decision tree

| Model | Accuracy |
|---|---|
| Best Decision Tree | 0.8557 |
| Random Forest | 0.7801 |

**Table 3.2:** Accuracy comparison between best decision tree and random forest

# 4 | Compare the Performance of your Implemented version of Random Forest with the inbuilt classifier

| Model | Accuracy | F1-Score |
|---|---|---|
| Custom Random Forest | 0.8050 | 0.3267 |
| Sklearn Random Forest | 0.8506 | 0.6052 |
| Logistic Regression | 0.8049 | 0.4708 |

**Table 4.1:** Comparison of different models based on Accuracy and F1-Score

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.8581 | 0.9637 | 0.9078 | 12435 |
| 1 | 0.8049 | 0.4849 | 0.6052 | 3846 |
| **Accuracy** | | 0.8506 | | |
| **Macro Avg** | 0.8315 | 0.7243 | 0.7565 | 16281 |
| **Weighted Avg** | 0.8456 | 0.8506 | 0.8364 | 16281 |

**Table 4.2:** Classification performance metrics for for **SkLearn Random Forest**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.8277 | 0.9402 | 0.8804 | 12435 |
| 1 | 0.6551 | 0.3674 | 0.4708 | 3846 |
| **Accuracy** | | 0.8049 | | |
| **Macro Avg** | 0.7414 | 0.6538 | 0.6756 | 16281 |
| **Weighted Avg** | 0.7870 | 0.8049 | 0.7836 | 16281 |

**Table 4.3:** Classification performance metrics for SkLearn Logistic Regression

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.8004 | 0.9921 | 0.8860 | 12435 |
| 1 | 0.8871 | 0.2002 | 0.3267 | 3846 |
| **Accuracy** | | 0.8050 | | |
| **Macro Avg** | 0.8438 | 0.5962 | 0.6064 | 16281 |
| **Weighted Avg** | 0.8209 | 0.8050 | 0.7539 | 16281 |

**Table 4.4:** Classification performance metrics for Custom Random Forest