Question-1

We will build the decision tree for the given dataset step by step, using **entropy** as the impurity measure. Finally, we'll compute the training error.

1. Compute Overall Entropy (Root Node)

The dataset consists of:

- 5 instances labeled **Approve**.
- 5 instances labeled **Reject**.

The total entropy $E(S)$ is given by:

$$E(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

where p_i is the proportion of instances of class i.

$$p(\text{Approve}) = \frac{5}{10} = 0.5, \quad p(\text{Reject}) = \frac{5}{10} = 0.5$$

$$E(S) = -\left(0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)\right)$$

$$E(S) = -\left(0.5 \cdot (-1) + 0.5 \cdot (-1)\right) = 1$$

The overall entropy of the root node is $E(S) = 1$.

2. Calculate Entropy and Information Gain for Each Attribute

a) Attribute: Long-Term Debt

Values: **Yes** and **No**

- **Long-Term Debt = Yes**: 5 instances, 1 Approve, 4 Reject

$$E(\text{Yes}) = -\left(\frac{1}{5} \cdot \log_2\left(\frac{1}{5}\right) + \frac{4}{5} \cdot \log_2\left(\frac{4}{5}\right)\right)$$

$$E(\text{Yes}) = -\left(0.2 \cdot (-2.32) + 0.8 \cdot (-0.32)\right) = 0.72$$

- **Long-Term Debt = No**: 5 instances, 4 Approve, 1 Reject

$$E(\text{No}) = -\left(\frac{4}{5} \cdot \log_2\left(\frac{4}{5}\right) + \frac{1}{5} \cdot \log_2\left(\frac{1}{5}\right)\right)$$

$$E(\text{No}) = -\left(0.8 \cdot (-0.32) + 0.2 \cdot (-2.32)\right) = 0.72$$

Weighted Entropy:

$$E(\text{Long-Term Debt}) = \frac{5}{10} \cdot 0.72 + \frac{5}{10} \cdot 0.72 = 0.72$$

Information Gain:

$$IG(\text{Long-Term Debt}) = E(S) - E(\text{Long-Term Debt}) = 1 - 0.72 = 0.28$$

b) Attribute: Unemployed

Values: **Yes** and **No**

- **Unemployed = Yes**: 2 instances, 1 Approve, 1 Reject

$$E(\text{Yes}) = -\left(\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) + \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right)\right) = 1$$

- **Unemployed = No**: 8 instances, 4 Approve, 4 Reject

$$E(\text{No}) = -\left(\frac{4}{8} \cdot \log_2\left(\frac{4}{8}\right) + \frac{4}{8} \cdot \log_2\left(\frac{4}{8}\right)\right) = 1$$

Weighted Entropy:

$$E(\text{Unemployed}) = \frac{2}{10} \cdot 1 + \frac{8}{10} \cdot 1 = 1$$

Information Gain:

$$IG(\text{Unemployed}) = E(S) - E(\text{Unemployed}) = 1 - 1 = 0$$

c) Attribute: Credit Rating

Values: **Good** and **Bad**

- **Credit Rating = Good**: 3 instances, 2 Approve, 1 Reject

$$E(\text{Good}) = -\left(\frac{2}{3}\cdot\log_2\left(\frac{2}{3}\right)+\frac{1}{3}\cdot\log_2\left(\frac{1}{3}\right)\right)$$

$$E(\text{Good}) = -\left(0.67\cdot(-0.58)+0.33\cdot(-1.58)\right) = 0.92$$

- **Credit Rating = Bad**: 7 instances, 3 Approve, 4 Reject

$$E(\text{Bad}) = -\left(\frac{3}{7}\cdot\log_2\left(\frac{3}{7}\right)+\frac{4}{7}\cdot\log_2\left(\frac{4}{7}\right)\right)$$

$$E(\text{Bad}) = -\left(0.43\cdot(-1.22)+0.57\cdot(-0.80)\right) = 0.98$$

Weighted Entropy:

$$E(\text{Credit Rating}) = \frac{3}{10}\cdot 0.92 + \frac{7}{10}\cdot 0.98 = 0.96$$

Information Gain:

$$IG(\text{Credit Rating}) = E(S) - E(\text{Credit Rating}) = 1 - 0.96 = 0.04$$

d) Attribute: Down Payment less than 20%

Values: *Yes* and *No*

- *Down Payment ¡ 20% = Yes*: 5 instances, 3 Approve, 2 Reject

$$E(\text{Yes}) = -\left(\frac{3}{5}\cdot\log_2\left(\frac{3}{5}\right)+\frac{2}{5}\cdot\log_2\left(\frac{2}{5}\right)\right) = -(0.6\cdot -0.737+0.4\cdot -1.322) = 0.97$$

- *Down Payment ¡ 20% = No*: 5 instances, 2 Approve, 3 Reject

$$E(\text{No}) = -\left(\frac{2}{5}\cdot\log_2\left(\frac{2}{5}\right)+\frac{3}{5}\cdot\log_2\left(\frac{3}{5}\right)\right) = 0.97$$

Weighted entropy:

$$E(\text{Down Payment}) = \frac{5}{10} \cdot 0.97 + \frac{5}{10} \cdot 0.97 = 0.97$$

Information gain:

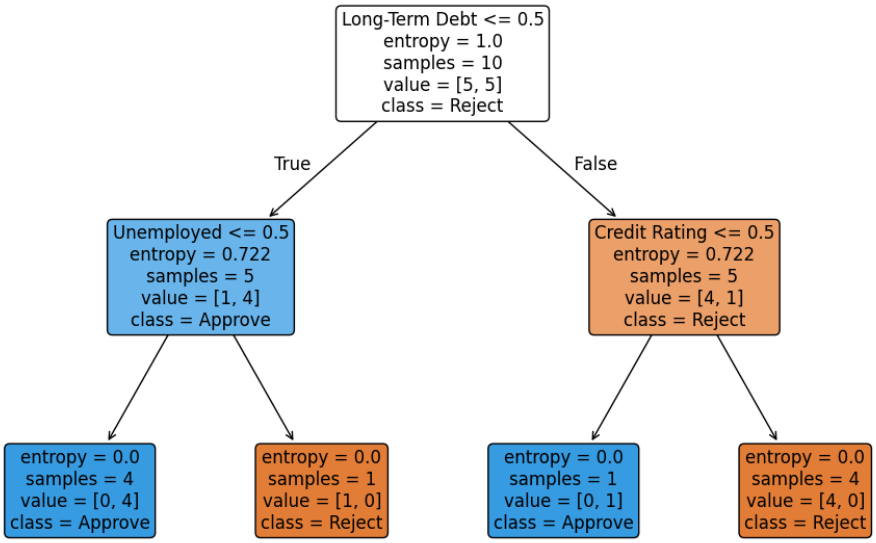
$$IG(\text{Down Payment}) = E(S) - E(\text{Down Payment}) = 1 - 0.97 = 0.03$$

3. Choose the Best Attribute for Splitting

The attribute with the highest *information gain* is *Long-Term Debt* (IG = 0.28).

4. Split and Repeat Recursively

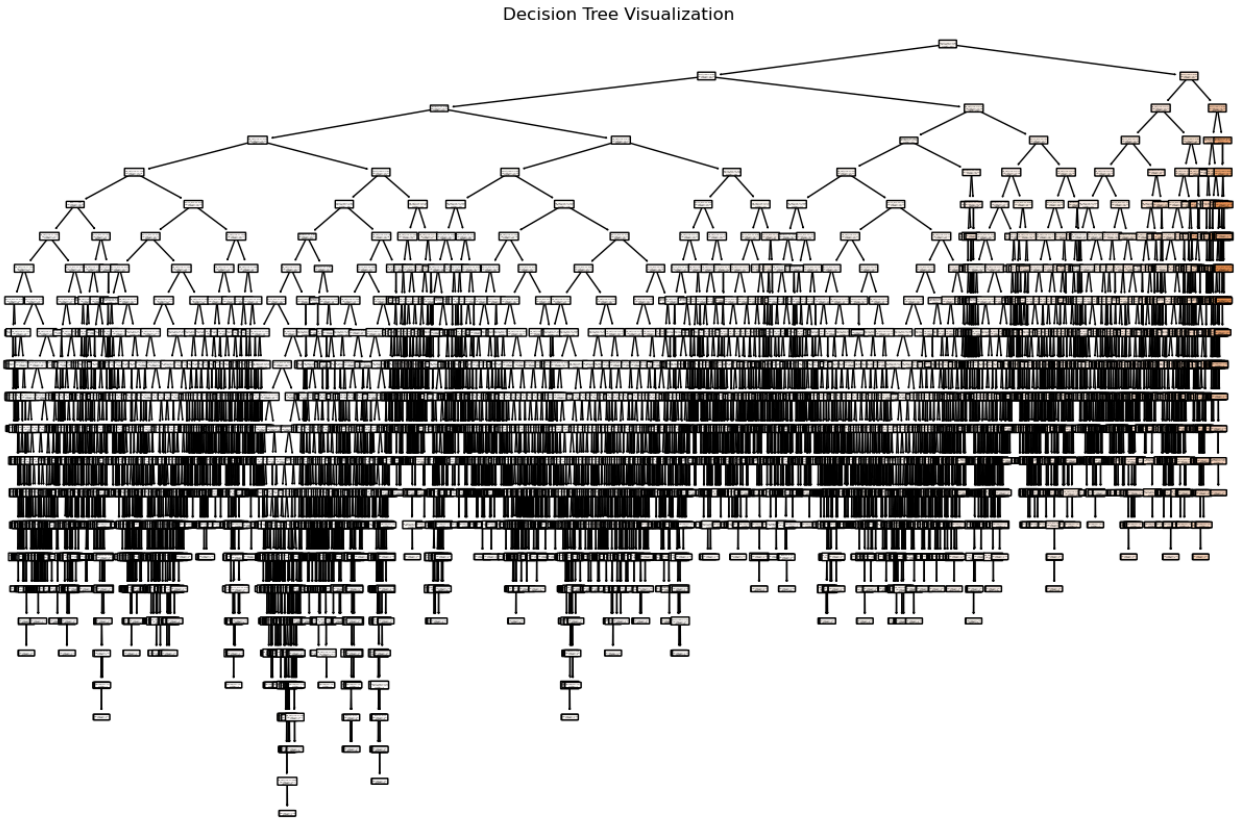
Using *Long-Term Debt* as the root, continue splitting on subsequent attributes for each branch ("Yes" and "No") until pure leaves are achieved.


Long-Term Debt <= 0.5
entropy = 1.0
samples = 10
value = [5, 5]
class = Reject
True
False
Unemployed <= 0.5
entropy = 0.722
samples = 5
value = [1, 4]
class = Approve
Credit Rating <= 0.5
entropy = 0.722
samples = 5
value = [4, 1]
class = Reject
entropy = 0.0
samples = 4
value = [0, 4]
class = Approve
entropy = 0.0
samples = 1
value = [1, 0]
class = Reject
entropy = 0.0
samples = 1
value = [0, 1]
class = Approve
entropy = 0.0
samples = 4
value = [4, 0]
class = Reject

Question-3)

Task-1:

(b)



Depth:

The depth of a decision tree refers to the maximum number of levels or decision nodes from the root node to a leaf node. In the image, the depth appears to be quite high, is exceeding 10 levels.

Nodes:

The number of nodes in a decision tree represents the total number of decision points and leaf nodes. From the visual, it seems that the tree has a large number of nodes, indicating a complex structure.

Depth's Impact on Complexity and Performance:

Depth significantly influences the complexity and performance of a decision tree model.

Complexity:

- **High Depth:** A deeper tree with many levels generally results in a more complex model. This is because it can capture intricate patterns within the data, potentially leading to overfitting. Overfitting occurs when the model becomes too tailored to the training data, and it struggles to generalize to new, unseen data.
- **Low Depth:** A shallower tree with fewer levels is simpler. While it might not capture all the nuances in the data, it is less prone to overfitting and can generalize better to new data.

Performance:

- **High Depth:** Initially, a deeper tree might achieve high accuracy on the training data. However, as the depth increases, the risk of overfitting grows, leading to decreased performance on unseen data.
- **Low Depth:** A shallower tree might not achieve the same level of accuracy on the training data as a deeper tree, but it can often perform better on unseen data due to its simpler structure and reduced overfitting.

Trade-off:

There is a trade-off between complexity and performance when choosing the depth of a decision tree. The goal is to find the right balance that allows the model to capture the underlying patterns in the data without overfitting.

Task-2:

(b)

Why Certain Features Are More Important:

1. **Carpet Area:** This feature is likely the most important as it directly relates to the size and value of the property. A larger carpet area often indicates a larger and more valuable property.
2. **Per Sqft Price:** This feature provides information about the price per square foot of the property, which is a key factor in determining the overall value.
3. **Brokerage:** This feature could be important as it can indicate the seller's motivation to sell and the potential for negotiation.
4. **Buildup Area:** This feature is related to the total constructed area of the property, which can also impact its value.

Other Features:

The remaining features, such as bathrooms, bedrooms, floor, parking, furnishing, BHK, possession, and address, have relatively low importance. This could be due to several reasons:

- **Less Impact on Value:** These features might not have a significant impact on the property's value compared to carpet area and per sqft price.
- **Data Quality:** The data for these features might be noisy or incomplete, leading to less reliable predictions.
- **Correlation:** These features might be highly correlated with other features, reducing their individual importance.

Matching Expectations:

The feature importance analysis seems to match general expectations about property value. Carpet area and per sqft price are key factors that influence property value. Other features, while relevant, might have a lesser impact.

(c)

Hyperparameter Optimization

The provided hyperparameters are a good starting point for optimizing the decision tree model. Here's a breakdown of the optimized parameters:

- **max_depth:** None: This indicates that the tree can grow as deep as necessary to fit the data.
- **max_features:** None: This means that all features are considered at each split.
- **min_samples_leaf:** 2: This ensures that each leaf node has at least 2 data points.
- **min_samples_split:** 5: This requires at least 5 data points in a node for it to be split.

Comparing Tuned and Default Models

The tuned model with the above parameters resulted in a lower Mean Squared Error (MSE) of 0.007288 compared to the default model. This indicates that the tuned model has better performance on the training data.

Task-3:

(b)

Comparing Pruned vs. Unpruned Decision Trees

Pruned Decision Tree

A pruned decision tree is a tree that has been simplified by removing unnecessary branches and leaves. This process is known as pruning. Pruning is done to reduce the complexity of the tree and improve its generalization performance.

Unpruned Decision Tree

An unpruned decision tree is a tree that has not been simplified. It may contain many branches and leaves, which can lead to overfitting. Overfitting occurs when the model is too complex and fits the training data too closely, resulting in poor performance on new, unseen data.

Task-4)

(c)

The Role of Cross-Validation in Controlling Overfitting for Decision Trees

Cross-validation is a crucial technique in controlling overfitting in Decision Trees. Here's a detailed discussion:

1. Identifying Overfitting

- **Overfitting** occurs when a Decision Tree becomes overly complex, capturing noise or specifics of the training data, leading to poor generalization on unseen data.
- Cross-validation helps detect overfitting by splitting the dataset into multiple folds and evaluating the model on different subsets. If the model performs well on the training data but poorly on validation folds, it indicates overfitting.

2. Generalization Assessment

- By evaluating the model across different folds, cross-validation provides an estimate of how the Decision Tree will perform on unseen data.
- This prevents relying solely on training accuracy, which might be misleading due to overfitting.

3. Balancing Complexity

- Decision Trees tend to grow deep, with many splits, making them prone to overfitting.
- Cross-validation allows for experimentation with hyperparameters like **max_depth**, **min_samples_split**, and **min_samples_leaf**, ensuring the tree remains balanced:
    - **Shallow trees** risk underfitting (poor learning).
    - **Deep trees** risk overfitting (memorization of training data).

4. Hyperparameter Tuning

- During cross-validation, the model is trained and validated on different folds while testing combinations of hyperparameters (e.g., via Grid Search or Randomized Search).

- This ensures the selected hyperparameters lead to an optimal balance between bias (underfitting) and variance (overfitting).

5. Reducing Variance in Model Performance

- Cross-validation averages the performance across all folds, reducing the impact of any single train-test split.
- This provides a more reliable estimate of model performance and ensures that the Decision Tree is not tailored to a specific subset of data.

6. Using Learning Curves

- Cross-validation is often combined with **learning curves** to observe training and validation errors.
- A large gap between training and validation errors typically indicates overfitting. Cross-validation helps identify the point where additional complexity no longer improves validation performance.

7. Pruning Trees

- Cross-validation helps determine the optimal pruning level by evaluating different values of ccp_alpha (cost-complexity pruning). This reduces overfitting by removing branches that contribute minimally to performance.

Q.4)

Task-1:
(b)

Model Performance Report and Interpretation

The model's performance metrics for both the training and test datasets are as follows:

Training Dataset Performance:

- **Mean Squared Error (MSE):** 0.0073
- **R-squared (R^2):** 0.9927

Interpretation:

- A very low **MSE** indicates that the model has a small average squared error in predicting the training dataset outcomes, which suggests high accuracy on the training data.

- The high **R^2 (99.27%)** shows that the model explains a significant proportion of the variance in the target variable in the training data. The model has learned the patterns in the data effectively.

Test Dataset Performance:

- **Mean Squared Error (MSE):** 0.0060
- **Mean Absolute Error (MAE):** 0.0272
- **R-squared (R^2):** 0.9926

Interpretation:

- The **MSE** is even slightly lower than the training MSE, and the **MAE** confirms a very small absolute error in predictions. This indicates that the model performs exceptionally well on unseen data.
- The **R^2 (99.26%)** is nearly identical to the training R^2, demonstrating that the model generalizes well and maintains high accuracy when applied to the test data.

Comparison of Training and Test Performance

- The training and test **MSE** values are very close, and the **R^2** scores are nearly identical.
- The slight difference between the metrics suggests that the model is neither **overfitting** (memorizing training data) nor **underfitting** (failing to capture patterns in the data). Instead, it achieves a good balance between bias and variance.

Task-2:

(b)

Yes, there are certain groups of data where the model consistently underperforms, as indicated by the residuals plot.

Observations from the Residuals Plot:

1. **Non-Randomness:** The residuals do not appear to be randomly distributed around the zero line. There are patterns in the residuals, suggesting that the model might not be capturing all the underlying patterns in the data.
2. **Heteroscedasticity:** The spread of residuals seems to vary across different ranges of actual prices. This indicates that the model's error is not constant across all price ranges.
3. **Outliers:** There are some data points with large residuals, indicating that the model is making significant errors for these particular observations.

Possible Improvements Based on Error Analysis:

1. **Model Complexity:**

    - Consider increasing the complexity of the model by adding more features or using a more flexible model architecture.
    - However, be cautious of overfitting, which can lead to poor performance on unseen data.
2. **Feature Engineering:**
    - Create new features that might capture the underlying patterns in the data better.
    - For example, you could create interaction terms between existing features or transform numerical features using techniques like log transformation or polynomial features.
3. **Outlier Treatment:**
    - Identify and handle outliers appropriately. This could involve removing outliers, imputing missing values, or using robust regression techniques.
4. **Regularization:**
    - Apply regularization techniques like L1 or L2 regularization to prevent overfitting and improve the model's generalization performance.
5. **Ensemble Methods:**
    - Combine multiple models (e.g., using bagging or boosting) to improve predictive accuracy and reduce variance.
6. **Non-Linear Transformations:**
    - Consider using non-linear transformations of the features, such as polynomial or exponential transformations, to capture complex relationships between features and the target variable.

Additional Considerations:

- **Data Quality:** Ensure that the data is clean and free of errors or inconsistencies.
- **Feature Importance:** Analyze the importance of different features to identify which features contribute most to the model's predictions.
- **Hyperparameter Tuning:** Experiment with different hyperparameters to find the optimal configuration for your model.