Assignment 2b

Problem 10

A data scientist is building a logistic regression model to detect a rare disease, which only occurs in 0.5% of the population (a 1:200 class imbalance). After training, the model achieves 99.5% accuracy on the test set.

What is the most significant problem with using accuracy as the primary evaluation metric here?

(a) The model is clearly overfitting the training data.

(b) The model is clearly underfitting the training data.

(c) The 0.5% of data with the disease is likely just noise or outliers.

(d) A "dummy" model that always predicts "no disease" ((y=0)) would achieve the exact same accuracy, providing no real predictive value.

Briefly (1–2 sentences) justify your choice and name two other metrics that would be more appropriate for this task.

Answer (final): (d).

Why (short): With a 0.5% prevalence, a trivial classifier that always predicts "no disease" will be correct for 99.5% of cases, so high accuracy can be achieved while missing every single positive case. Therefore accuracy is useless for highly imbalanced, rare-event detection.

What to use instead (recommended): use metrics that focus on the minority class — e.g. **recall (sensitivity)** and **precision**, or their harmonic mean **F1-score**; also consider **ROC-AUC** and **PR-AUC** (precision–recall AUC) and inspect the **confusion matrix**.

(Practical note: for a disease detector you usually prioritize high recall/sensitivity so you catch most positives, then use precision to control false alarms.)

Problem 11

You are trying to find the best logistic regression model and are experimenting with the regularization parameter (C). (Recall that (C) is the **inverse** of the regularization strength (λ); a **small (C)** means **strong regularization**, and a **large (C)** means **weak regularization**.)

You plot the **Mean Squared Error (or Log Loss)** on both your **training set** and your **validation set** as you vary (C) from very small (e.g., (10^{-4})) to very large (e.g., (10^{4})).

Describe the expected shape of the two curves (training error and validation error) on this plot. How would you use this plot to choose the optimal (C)? What do the regions of "very small (C)" and "very large (C)" represent in terms of bias and variance?

---

1. Expected shape of the training error curve

Training error as a function of (C):

- **Very small (C)**
  - Strong regularization (large penalty).
  - Model coefficients shrink heavily → model is too simple.
  - **Training error is high.**
  - *(Why? Underfitting: the model cannot capture even the training patterns.)*
- **As (C) increases**
  - Regularization weakens → the model can fit the data better.
  - **Training error decreases monotonically.**
  - *(Why? A less constrained model always fits training data at least as well.)*
- **Very large (C)**
  - Almost no regularization.
  - Model fits the training data extremely well.
  - **Training error becomes very low** (possibly near zero).

- *(Why? Overfitting zone: model fits noise.)*

→ **Shape:** A **strictly decreasing curve** from high error to low error.

---

2. Expected shape of the validation error curve

Validation error as a function of (C):

- **Very small (C)**
  - Underfit model → poor generalization.
  - **Validation error is high.**
- **Moderate (C)**
  - Best balance between fitting the data and regularizing.
  - **Validation error reaches a minimum** at some optimal (C^*,) producing the "valley."
  - *(Why? Bias and variance are balanced.)*
- **Very large (C)**
  - Model overly flexible → overfits training noise.
  - **Validation error increases again.**
  - *(Why? High variance: model fails on unseen data.)*

→ **Shape:** A **U-shaped (or bowl-shaped) curve**.

---

3. Choosing the optimal value of (C)

- The optimal (C) is the one at **the minimum point of the validation error curve**.
- This is where the model has the **lowest generalization error**.
- If the curve is flat around the minimum, it is common to pick the **smaller (C)** in that "flat region" for a simpler, more stable model.

(Why? Smaller (C) = stronger regularization = less variance, better generalization.)

---

4. Interpretation of "very small (C)" and "very large (C)"

Very small (C)

- **Strong regularization** → large penalty
- **The model is overly simple**
- **High bias, low variance**
- **Underfitting region**

Very large (C)

- **Weak regularization** → small penalty
- **The model becomes overly complex**
- **Low bias, high variance**
- **Overfitting region**

Final Summary

- Training curve → **monotonically decreasing** as (C) increases.
- Validation curve → **U-shaped**, with a minimum at the optimal (C^*).
- Choose **(C^*)** where the **validation error is lowest**.
- **Small (C)** = high bias / low variance (underfitting).
- **Large (C)** = low bias / high variance (overfitting).

Problem 12

You are given the following dataset for a binary classification problem (target: “Buys Computer”):

Age	Income	Student	Credit Rating	Buys Computer
<30	High	No	Fair	No

Age	Income	Student	Credit Rating	Buys Computer
<30	High	No	Excellent	No
31–40	High	No	Fair	Yes
>40	Medium	No	Fair	Yes
>40	Low	Yes	Fair	Yes
>40	Low	Yes	Excellent	No
31–40	Low	Yes	Excellent	Yes
<30	Medium	No	Fair	No

(a) Compute the **information gain** for splitting on the attribute **Student**.

(Use entropy and information gain formulas step-by-step.)

(b) Based on your computed gain, discuss whether **Student** is a good attribute to split on first.

Solution — step-by-step (with explanations)

Step 1 — Count totals and class proportions (why: to compute base entropy)

Total examples (N = 8).

Count target labels:

- Number of **Yes** = 4 (rows 3, 4, 5, 7).
- Number of **No** = 4 (rows 1, 2, 6, 8).

So class probabilities:

$$P(\text{Yes}) = \frac{4}{8} = 0.5, \qquad P(\text{No}) = \frac{4}{8} = 0.5.$$

(Why? counts from the dataset.)

Step 2 — Compute entropy of the full set (H(S))

Entropy formula (base-2):

$$H(S) = -\sum_c P(c)\log_2 P(c).$$

$$With(P(\text{Yes}) = P(\text{No}) = 0.5):$$

$$H(S) = -0.5\log_2(0.5) - 0.5\log_2(0.5).$$

Compute step by step:

- ($\log_2(0.5)$ = -1.)
- So each term ($-0.5 \times (-1) = 0.5$).
- Sum (H(S)=0.5+0.5=1.0) (bits).

$$\boxed{H(S) = 1.0 \text{ bit}.}$$

(Why? balanced 50/50 gives maximum entropy of 1 bit.)

Step 3 — Partition the dataset by Student and count class labels (why: needed for conditional entropies)

Attribute **Student** has two values: **Yes** and **No**.

- **Student = Yes**: which rows? rows 5, 6, 7 → (N_{Yes}=3). Their targets: Yes (row5), No (row6), Yes (row7) ⇒ **2 Yes**, **1 No**.
  - So ($P(\text{Yes} \mid \text{Student}=\text{Yes})$ = 2/3), ($P(\text{No} \mid \text{Student}=\text{Yes})$ = 1/3.)
- **Student = No**: remaining rows 1,2,3,4,8 → (N_{No}=5). Their targets: No, No, Yes, Yes, No ⇒ **2 Yes**, **3 No**.
  - So ($P(\text{Yes} \mid \text{Student}=\text{No})$ = 2/5 = 0.4), ($P(\text{No} \mid \text{Student}=\text{No})$ = 3/5 = 0.6.)

(Why? simply counting labels inside each subset.)

Step 4 — Compute entropy for each subset

Entropy formula again for each subset.

For Student = Yes ($(p_1 = 2/3,\ p_2 = 1/3)$):

$$H(\text{Student}=\text{Yes}) = -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right).$$

Compute numerically (digit-by-digit):

- ($\log_2(2/3) \approx -0.5849625007.$)
  Contribution: ($-\frac{2}{3} \times (-0.5849625007) = 0.389974999.$)
- ($\log_2(1/3) \approx -1.5849625007.$)
  Contribution: ($-\frac{1}{3} \times (-1.5849625007) = 0.5283208336.$)
- Sum: ($H(\text{Student}=\text{Yes}) \approx 0.389975 + 0.528321 = 0.9182958336.$)

$$\boxed{H(\text{Student}=\text{Yes}) \approx 0.91830 \text{ bits.}}$$

For Student = No ($(p_{\text{Yes}} = 0.4,\ p_{\text{No}} = 0.6)$):

$$H(\text{Student}=\text{No}) = -0.4 \log_2(0.4) - 0.6 \log_2(0.6).$$

Compute:

- ($\log_2(0.4) \approx -1.3219280949.$)
  $Contribution$: ($-0.4 \times (-1.3219280949) = 0.52877123796.$)
- ($\log_2(0.6) \approx -0.7369655942.$)
  $Contribution$: ($-0.6 \times (-0.7369655942) = 0.4421793565.$)
- Sum : ($H(\text{Student}=\text{No}) \approx 0.528771238 + 0.442179357 = 0.9709505955.$)

$$\boxed{H(\text{Student}=\text{No}) \approx 0.97095 \text{ bits.}}$$

(Why? entropy measures impurity inside each subset.)

Step 5 — Compute the weighted average (conditional) entropy after splitting on Student

Weighted entropy:

$$H(S \mid \text{Student}) = \frac{N_{\text{Yes}}}{N} H(\text{Student}=\text{Yes}) + \frac{N_{\text{No}}}{N} H(\text{Student}=\text{No}).$$

Plug numbers:

- $(N_{\text{Yes}} = 3,\ N_{\text{No}} = 5,\ N = 8.)$
- Weight for Yes subset = (3/8 = 0.375.)
  Contribution: $(0.375 \times 0.9182958336 \approx 0.3443609376.)$
- Weight for No subset = (5/8 = 0.625.)
  Contribution: $(0.625 \times 0.9709505955 \approx 0.6068441222.)$
- Sum: ($H(S \mid \text{Student}) \approx 0.3443609376 + 0.6068441222 = 0.9512050598.$
)

$$\boxed{H(S \mid \text{Student}) \approx 0.95121 \text{ bits.}}$$

(Why? the expected entropy after the split.)

Step 6 — Information Gain for splitting on Student

Information gain:

$$\text{IG}(S, \text{Student}) = H(S) - H(S \mid \text{Student}).$$

We have (H(S)=1.0) and ($H(S \mid \text{Student}) \approx 0.95120506$). So

$$\text{IG} \approx 1.0 - 0.95120506 = 0.04879494 \text{ bits.}$$

Round:

$$\boxed{\text{Information gain} \approx 0.0488 \text{ bits.}}$$

(Why? IG is reduction in entropy—how much uncertainty about the class is removed by splitting on Student.)

Step 7 — Interpret the result (Is Student a good first split?)

- The **information gain ≈ 0.0488 bits** is **very small** (base entropy is 1.0 bit).
- This means splitting on **Student** reduces class uncertainty only a little — the two child nodes remain fairly impure (entropies ≈ 0.918 and 0.971).
- **Conclusion: Student is not a strong attribute to split on first.** You would prefer an attribute that gives a much larger information gain (i.e., yields purer child nodes).

(Why? A good split should significantly reduce entropy; here the reduction is negligible.)

Final concise takeaway

- Computed IG(Student) ≈ **0.0488 bits** (small).
- **Student is not a good first split** — it provides only a tiny reduction in uncertainty; look for attributes with larger information gain (e.g., check `Credit Rating`, `Income`, or `Age` to compare).

✅ 1. Entropy (measure of impurity/uncertainty)

Entropy is a measure of how "mixed" the classes are in a dataset. It is highest when the classes are equally mixed (50–50 for binary classification).

For a binary target:

$$H(S) = -p \log_2 p - (1-p) \log_2 (1-p)$$

where ($p = P(\text{positive})$).

Intuition:

- If all labels are the same (all Yes or all No) → **entropy = 0** (pure).
- If labels are evenly split → **entropy = 1** (maximal uncertainty).

Why do we use entropy?

Decision trees want to split data into groups that are **as pure as possible**.

Entropy tells the tree *how impure* the current group is.

✅ 2. Conditional Entropy (entropy after splitting)

When we split on an attribute (e.g., **Student**), we divide the dataset into subsets:

- Student = Yes subset
- Student = No subset

Each subset has its own entropy.

The **conditional entropy** is the weighted average of these entropies:

$$H(S \mid \text{Attribute}) = \sum_{v} \frac{|S_v|}{|S|} H(S_v)$$

Intuition:

- It measures **remaining uncertainty** after splitting on the attribute.
- Weighting is needed because larger subsets matter more.

✅ 3. Information Gain (IG)

Information Gain tells us **how helpful an attribute is** at reducing uncertainty.

$$\text{IG}(S, A) = H(S) - H(S \mid A)$$

Interpretation:

- High IG → the attribute produces **pure subsets** → good for splitting.
- Low IG → attribute does **not reduce uncertainty** → not useful for splitting.

Decision trees choose the attribute with **maximum IG** at each step (ID3 algorithm).

✅ 4. Why do we split on attributes?

The goal of a decision tree is to create branches where each branch has mostly one class.

A **good attribute**:

- divides the data in a meaningful way
- separates Yes from No as much as possible
- reduces impurity

If an attribute splits the data but both subsets remain mixed, then the split is not helpful.

✅ 5. Understanding the "Student" split specifically

In the dataset:

- Student = Yes subset

  → 2 Yes, 1 No (still mixed → entropy high)

- Student = No subset

  → 2 Yes, 3 No (also mixed → entropy high)

When both subsets are impure, **conditional entropy is high**, and IG becomes small.

Meaning:

Splitting on Student does **not** cleanly separate the classes.

✅ 6. What makes a "good" first attribute?

A good first attribute:

- produces subsets that are close to pure (entropy near 0)
- produces a **larger** reduction in overall entropy
- has the **highest IG**

Example:

- If an attribute leads to subsets like (5 Yes, 0 No) and (3 Yes, 0 No), entropy is 0 → IG is high → extremely good split.

✅ 7. Why do decision trees use these measures

Decision trees grow by making **greedy choices**:

- At each step, choose the attribute that maximizes information gain
- This locally reduces the uncertainty the most

Entropy + Information Gain allows a tree to approximate optimal splits without brute-force modeling.

Quick Summary of Concepts

Concept	Meaning	Why used?
Entropy	How mixed the class labels are	Trees want to make pure groups
Conditional Entropy	Remaining uncertainty after split	Measures usefulness of attribute
Information Gain	Reduction in entropy after splitting	Attribute selection criterion
Good split	Makes subsets purer	Improves classification
Bad split	Subsets still mixed	Adds no useful structure

Problem 13

Explain the following with examples:

(a) Decision Tree vs Random Forest in terms of the **bias–variance tradeoff**.

(b) **Gini Impurity vs Entropy** — when might one be preferred over the other?

(c) **Pruning vs Early Stopping** — How do both help avoid overfitting?

(a) Decision Tree vs Random Forest: Bias–Variance Tradeoff

Decision Tree

- Decision trees are **low-bias, high-variance** models.
- They can learn very flexible boundaries and perfectly fit the training data (especially deep trees).
- This often results in **overfitting**, meaning:
  - low training error,
  - high test error due to sensitivity to noise.

Example:

A single decision tree may produce very different splits if trained on slightly different datasets.

(High variance.)

Random Forest

- A Random Forest is an **ensemble of many decision trees**, each trained on a bootstrap sample and using random feature subsets.
- The predictions are averaged (for regression) or majority-voted (classification).

This:

- greatly **reduces variance** (averaging "smooths out" noisy trees),
- keeps bias roughly the same (bias of each tree is low).

Thus, RF has:

- **lower variance**
- **similar or slightly higher bias**,
- **much better generalization**.

Example:

If one tree makes a wrong split based on noise, the majority of trees won't, so the ensemble is stable.

Summary Table

Model	Bias	Variance
Decision Tree	Low bias	**High variance**
Random Forest	Slightly higher bias	**Much lower variance**

(b) Gini Impurity vs Entropy — When to use which?

Both measure node impurity, but behave slightly differently.

Entropy

$H = -\sum p_i \log_2 p_i$

- More sensitive to small probability changes.
- Has a stronger penalty for highly mixed nodes.
- Information Gain criterion uses entropy.

When useful:

- If you want splits that maximize pure information reduction.
- When comparing many-class problems (entropy captures imbalance well).

Gini Impurity

$G = 1 - \sum p_i^2$

- Easier to compute (no logarithm).
- Often leads to **very similar splits** to entropy.
- Favored in **CART trees** (Classification And Regression Trees).

When useful:

- When computation must be fast (large datasets).
- When you want slightly more “greedy” splits that tend to isolate the majority class quickly.

Practical rule of thumb

- **Gini** is faster and usually behaves almost the same.
- **Entropy** is more theoretically grounded (information theory).
- In real datasets, **differences are usually minimal**.

Example:

If a node has class distribution:

- Yes = 90%, No = 10%

- Gini = (1 - ($0.9^2 + 0.1^2$) = 0.18)
- Entropy = ($-0.9 \log_2 0.9 - 0.1 \log_2 0.1 \approx 0.469$)

Both indicate “low impurity,” but entropy penalizes impurity more strongly.

(c) Pruning vs Early Stopping — How they prevent overfitting

Early Stopping (pre-pruning)

You **stop the tree from growing too large** from the beginning.

Examples:

- Set a maximum depth (e.g., depth ≤ 5).
- Minimum samples per split.
- Minimum samples per leaf.
- Maximum number of leaf nodes.

Effect:

- Tree stays small and simple → **higher bias but lower variance**.
- Prevents overly specific rules that memorize the training data.

Analogy:

Don’t let the tree “memorize.”

Pruning (post-pruning)

You **grow the full tree first**, then cut off branches that do not improve validation performance.

Two types:

- **Cost-complexity pruning** (CART)
- **Reduced-error pruning**

Effect:

- Removes branches that are too specific or noisy.
- Shrinks tree size → reduces variance.
- Keeps only useful, generalizable splits.

Analogy:

Let the tree learn everything, then “trim” the overfit parts.

Final Summary

- **Decision tree** = low bias, high variance.
- **Random forest** = reduces variance via ensembling → better generalization.
- **Entropy** and **Gini** are both impurity measures; entropy penalizes uncertainty more heavily, Gini is faster and gives similar results.
- **Early stopping** prevents complex trees from growing (pre-pruning).
- **Pruning** removes unhelpful branches after the tree has grown (post-pruning).

  Both combat overfitting by reducing complexity.

Problem 14

You are building a decision tree for a bank to classify whether a loan applicant is likely to default. Discuss how you would handle the following:

(a) Missing data in categorical and numerical features.

(b) Highly imbalanced data (e.g., 95% non-defaulters, 5% defaulters).

(c) Continuous attributes (like income, age) — how does the algorithm find the best split?

(Explain both conceptually and practically how these are handled in implementations like sklearn.)

(a) Handling Missing Data in Categorical and Numerical Features

Decision trees can handle missing values in several practical ways. The general goal is: **don't discard data; route missing values sensibly**.

---

1. Numerical features with missing values

Approach A: Imputation

- Replace missing values with:
  - **Median** (most common for numerical data)
  - **Mean** (less robust)
  - **Model-based imputation** (KNN, regression, etc.)

Why median?

Robust to outliers and preserves the distribution better.

Approach B: Add a "missing-value indicator" feature

- Replace missing values with the median
- Add a binary feature: `age_missing = 1 if age is missing else 0`.

Why this works:

Decision trees can then learn separate paths for missing vs. non-missing values.

---

2. Categorical features with missing values

Approach A: Replace missing categories with a special label ("Unknown")

- Treat missing as its own category:
  - `Income = {High, Medium, Low, Unknown}`

Why?

Decision trees can branch on Unknown and treat it as a meaningful pattern.

Approach B: Mode imputation

- Replace the missing category with the most frequent category.

Why?

Simple and maintains class proportions.

---

3. How sklearn handles missing values

- **sklearn DecisionTreeClassifier DOES NOT natively support** `NaN` → you must impute first.
- Common choice: `SimpleImputer(strategy="median" or "most_frequent")`.

Tree-based models in **LightGBM, XGBoost, CatBoost** *do* support missing values by learning default directions ("go left when a feature is missing").

(b) Handling Highly Imbalanced Data (95% non-defaulters, 5% defaulters)

When classes are very imbalanced, a tree can get **biased toward predicting the majority class**.

---

1. Resampling Techniques

Oversampling the minority class

- Duplicate positive (defaulter) examples.
- Or use **SMOTE** to synthesize new minority examples.

Undersampling the majority class

- Randomly remove some majority-class samples.
- Risk: losing information.

Balanced batch sampling (in ensemble methods)

- Make sure each tree sees balanced subsets.

---

2. Cost-sensitive Learning

Make misclassifying a **defaulter** far more expensive.

In **sklearn**:

`DecisionTreeClassifier(class_weight="balanced")`

or

`class_weight={0:1, 1:10}`

Why use this?

The tree will try harder to split in a way that correctly identifies the minority class.

---

3. Evaluation Metrics

Use metrics appropriate for imbalance:

- Precision, Recall
- F1-score
- ROC-AUC
- PR-AUC
- Confusion matrix

Accuracy is misleading (95% accuracy = predict "non-default" every time).

---

4. Ensemble Approaches

RandomForest, XGBoost, and Gradient Boosting handle imbalance better, especially with:

`scale_pos_weight` or `class_weight`

(c) Handling Continuous Attributes — How Decision Trees Find the Best Split

Decision trees **do NOT** treat continuous features like categories.

Instead, they search for the **best threshold** (t) that splits the feature into:

$$x \leq t \quad \text{vs.} \quad x > t.$$

Conceptual Explanation

1. **Sort the values** of the feature (e.g., incomes).
2. Consider **potential split points** between adjacent unique values.
3. For each candidate threshold (t):

- split data into Left = (x ≤ t), Right = (x > t)
- compute impurity (Gini or Entropy) for both sides
- compute weighted impurity after split

4. Choose the threshold (t) that **reduces impurity the most** (maximum Information Gain or Gini reduction).

Example:

For Income = [40k, 45k, 50k, 60k, 80k]

Possible split thresholds:

- 42.5 (between 40 and 45)
- 47.5 (between 45 and 50)
- 55 (between 50 and 60)
- 70 (between 60 and 80)

Each threshold is tested.

The one with the highest information gain becomes the split.

Practical Explanation (sklearn implementation)

sklearn's `DecisionTreeClassifier`:

- Automatically sorts values internally.
- Considers all midpoints between sorted unique values.
- Computes impurity (`criterion="gini"` or `criterion="entropy"`).
- Selects the split maximizing impurity reduction.

You don't need to manually discretize or bin continuous variables — the algorithm handles it optimally.

Final Summary

(a) Missing data

- Numerical: median imputation; add missing-indicator feature.
- Categorical: use “Unknown” category or mode imputation.
- sklearn requires imputation first.

(b) Imbalanced data

- Use resampling (SMOTE, oversampling, undersampling).
- Use class weights (`class_weight="balanced"`).
- Use proper metrics (Recall, F1, PR-AUC).
- Ensemble methods often perform better.

(c) Continuous attributes

- Trees choose splits by scanning all possible thresholds.
- Measure impurity reduction at each candidate.
- sklearn does this automatically by sorting feature values and testing midpoints.

Problem 15

A student claims: “Decision Trees are always better than Logistic Regression because they are more interpretable and can model nonlinear data.” Critically analyze this statement — in what cases is this true, and when might Logistic Regression still perform better? Provide reasoning supported by examples or intuitions.

Short answer (thesis).
The claim is too strong. Decision trees can be more interpretable in some settings and can model nonlinear interactions, but they also have important weaknesses (high variance, poor probability calibration, data-hungry when deep). Logistic regression is simpler and often better when the true relationship is (approximately) linear in the log-odds, when you have limited data, or when you need well-calibrated probabilities. Which is “better” depends on the problem, data, and what you mean by “interpretable.”

Detailed analysis, point-by-point.

A. Interpretability — nuance and context

- **Decision Tree (single shallow tree):**
  - **Pros:** A small tree (depth 2–4) yields human-readable rules ("if Income > 50k and Age < 30 then ..."). This is often what people mean by "interpretable."
  - **Cons:** Trees are *unstable*: small changes in data can flip splits, producing quite different trees. Ensembles (Random Forest, Gradient Boosting) remove instability but become **much less interpretable**. Also deep trees produce complex rule-sets that are hard to reason about.
- **Logistic Regression:**
  - **Pros:** Coefficients give a **global**, consistent measure of feature influence (log-odds per unit change). Easy to explain: "holding other features constant, an extra year of age multiplies the odds by e^{β}." Very stable and reproducible.
  - **Cons:** For interactions or thresholds, coefficients may be harder to interpret without engineered features (e.g., an interaction term or binning).

Takeaway: *A small decision tree is interpretable in a different sense than logistic regression.* If you require a **stable, global** explanation, logistic regression often wins. If you need **rule-based** explanations and are okay with local rules, a shallow tree can be preferable.

B. Ability to model nonlinearity and interactions

- **Decision Trees:**
  - Naturally model **nonlinearities** and **feature interactions** without feature engineering. They partition feature space into axis-aligned boxes; they can capture threshold effects like "if income > 50k then...".
  - Example: XOR-type patterns or rules where only combinations matter — a tree can represent them compactly.

- **Logistic Regression:**
  - Models a **linear decision boundary in feature space** (linear in the parameters for log-odds). To capture nonlinearity or interactions you must add engineered features (polynomials, interactions, splines).
  - Example: if the true boundary is linearly separable in the original features → logistic is perfect and simple. If the true decision surface is highly nonlinear, logistic needs feature transforms.

Takeaway: Trees win *when you want to capture complex interactions without feature engineering.* Logistic wins *when the log-odds are (approximately) linear or you can reliably engineer features.*

C. Generalization, variance, and sample efficiency

- **Decision Trees (single):** low bias but **high variance** — easy to overfit especially when deep and with limited data. Need pruning, constraints, or ensembling to generalize.
- **Logistic Regression:** convex optimization, **low variance** (especially with regularization), generally **more sample-efficient** for low-to-moderate complexity problems.

Example: Small dataset (few hundred examples) where signal is approximately linear → logistic with L2 regularization likely outperforms a deep tree. Conversely, very large dataset with complex interactions → tree-based ensembles often outperform.

D. Probability estimates & calibration

- **Logistic regression** directly models P(y=1 ∣ x) (well-calibrated probabilities when model fits).
- **Single decision tree** often produces poor probability estimates (leaf empirical frequencies), and **ensembles** (Random Forest) give better but sometimes still miscalibrated probabilities. Calibration matters for risk scoring (medicine, finance).

Implication: If you require **well-calibrated probabilities** (not just classification), logistic regression is often preferable (or you must calibrate tree outputs).

E. Robustness to feature types, missing data, and preprocessing

- **Decision Trees:** handle mixed numerical/categorical features naturally (no need to one-hot), robust to monotone transformations; some tree implementations handle missing values natively.
- **Logistic Regression:** usually requires numeric features; categorical features need encoding; missing values must be imputed.

Implication: When features are messy or mixed, trees are convenient; when data are numeric and clean, logistic is straightforward.

F. Computational and practical considerations

- **Training:** Logistic regression (convex) is fast and predictable; trees can be fast but require tuning (depth, min samples).
- **Ensembling:** Random Forests / Boosting typically beat a single tree but lose interpretability. They are state-of-the-art for many tabular tasks.

Concrete examples to illustrate when each is better

- **When Decision Trees (or tree ensembles) are better**
  - Problem has threshold rules or complex interactions (e.g., “if feature A > t and B in {x,y} then outcome”), and you lack time for feature engineering.
  - Large dataset with many categorical features (no costly encoding) — ensemble methods (XGBoost/LightGBM) often win Kaggle-style tabular tasks.
- **When Logistic Regression is better**
  - Relationship is (approximately) linear in the log-odds, or you can engineer good features (e.g., transformed predictors).

- Small dataset where high variance models overfit.
- You need stable, **global explanations** and calibrated probabilities (credit scoring, clinical risk scores).

Practical advice / decision checklist

- If you need **simple, stable explanations** and data likely linear → use **logistic regression** (with regularization).
- If you need **rule-style explanations** and can keep tree shallow, consider a **single decision tree**.
- If predictive performance on complex tabular data is the goal and interpretability is secondary → use **tree ensembles** (Random Forest / Boosted Trees). Then use model-agnostic explanation tools (SHAP, LIME) for interpretability.
- Always check **calibration**, **cross-validated performance**, and **sensitivity** to hyperparameters.

Final concise verdict

The student's statement is **overgeneralized**. Decision trees are powerful and flexible, but **not always better** — choice depends on data size, true relationship, need for stable calibrated probabilities, and interpretability style required. Use the method whose **inductive biases** best match the problem.

Problem 16

You are a data scientist at 'Zwiggato' trying to build a Decision Tree model to predict **delivery time in minutes (Y)**. You have the following small dataset from recent orders:

ID	Distance	Order Time	Items	Delivery Time
1	3.5 km	06:00 PM	2	25 min
2	1.2 km	07:00 PM	5	30 min

ID	Distance	Order Time	Items	Delivery Time
3	5.0 km	12:00 PM	1	20 min
4	2.1 km	06:30 PM	3	35 min
5	4.8 km	01:00 PM	6	40 min
6	1.5 km	08:00 PM	2	22 min

Your algorithm is considering two **potential first splits** for the root node:

- **Split A:** Distance ≤ 3.0 km
- **Split B:** Items ≤ 2

(a) For Split A and B, calculate the **MSE for each child node**, and then calculate the final **weighted average MSE** for Delivery Time.

(b) Based on your calculations, which split (A or B) would the regression tree choose, and why?

---

1. **Solution — step-by-step with explanations (why each step).**

Step 1 — Method / remind why we compute MSE

For a regression tree, each leaf predicts the **mean** of the target values in that leaf. The impurity (loss) used here is the **Mean Squared Error (MSE)** of that leaf:

$$\text{MSEnode} = \frac{1}{n} \sum i = 1^{n} (y_i - \bar{y})^2,$$

where ($\bar{y}$) is the mean delivery time in that node.

The **overall split score** is the weighted average of child-node MSEs (weights = proportion of samples). The algorithm chooses the split with the **smaller** weighted MSE (i.e., larger reduction in MSE).

Split A: Distance ≤ 3.0 km

Partition the data (why): separate rows with Distance ≤ 3.0 (left) and > 3.0 (right).

- Left group (Distance ≤ 3.0): IDs 2 (1.2 km, 30), 4 (2.1 km, 35), 6 (1.5 km, 22). Delivery times = ([30,35,22]). ($n_L = 3$.)
- Right group (Distance > 3.0): IDs 1 (3.5 km,25), 3 (5.0 km,20), 5 (4.8 km,40).

Delivery times = ([25,20,40]). ($n_R = 3$.)

Compute left-node mean and MSE (why: prediction = mean; MSE quantifies residuals).

- Mean left ($\bar{y}_L$ = (30+35+22)/3 = 87/3 = 29.0) minutes.
- Squared errors left:

  $((30 - 29)^2$ = 1)

  $((35 - 29)^2$ = 36)

  $((22 - 29)^2$ = 49)

  Sum SSE_L = (1+36+49 = 86).
- $MSE_L = SSE_L/3$ = ($86/3 \approx 28.6667$).

Compute right-node mean and MSE.

- Mean right ($\bar{y}_R = (25 + 20 + 40)/3 = 85/3 \approx 28.3333$) minutes.
- Squared errors right:

  $((25 - 28.3333)^2 \approx 11.1111)$

  $((20 - 28.3333)^2 \approx 69.4444)$

  $((40 - 28.3333)^2 \approx 136.1111)$

  Sum $SSE_R \approx (11.1111 + 69.4444 + 136.1111 = 216.6666)$.
- $MSE_R = SSE_R/3 \approx (216.6666/3 = 72.2222)$.

Weighted average MSE for Split A (weights = 3/6 each, so simple average):

$$\text{MSE}_A = \tfrac{3}{6} \cdot 28.6667 + \tfrac{3}{6} \cdot 72.2222 = \tfrac{86+216.6667}{6} = \tfrac{302.6667}{6} \approx 50.4444.$$

Summary Split A results

- Left MSE ≈ **28.6667** (n=3)
- Right MSE ≈ **72.2222** (n=3)
- Weighted MSE_A ≈ **50.4444**

Split B: Items ≤ 2

Partition the data (why): Items ≤ 2 left, Items > 2 right.

- Left group (Items ≤ 2): IDs 1 (Items2, 25), 3 (Items1, 20), 6 (Items2, 22).

  Delivery times = ([25,20,22]). ($n_L = 3$.)
- Right group (Items > 2): IDs 2 (5,30), 4 (3,35), 5 (6,40).

  Delivery times = ([30,35,40]). ($n_R = 3$.)

Compute left-node mean and MSE.

- Mean left ($\bar{y}_L$ = (25+20+22)/3 = 67/3 ≈ 22.3333).
- Squared errors left:

  $((25 - 22.3333)^2 \approx 7.1111)$

  $((20 - 22.3333)^2 \approx 5.4444)$

  $((22 - 22.3333)^2 \approx 0.1111)$

  $SumSSE_L \approx (7.1111 + 5.4444 + 0.1111 = 12.6666)$.
- $MSE_L = SSE_L/3 \approx (12.6666/3 = 4.2222)$.

Compute right-node mean and MSE.

- Mean right ($\bar{y}_R$ = (30+35+40)/3 = 105/3 = 35.0).
- Squared errors right:

  $((30 - 35)^2 = 25)$

  $((35 - 35)^2 = 0)$

  $((40 - 35)^2 = 25)$

  Sum $SSE_R = (25 + 0 + 25 = 50)$.
- $MSE_R = SSE_R/3 = (50/3 \approx 16.6667)$.

Weighted average MSE for Split B:

$$\mathrm{MSE}_B = \frac{12.6666 + 50}{6} = \frac{62.6666}{6} \approx 10.4444.$$

Summary Split B results

- Left MSE ≈ **4.2222** (n=3)

- Right MSE ≈ **16.6667** (n=3)
- Weighted MSE_B ≈ **10.4444**

Step 2 — Compare splits and explain the choice (why)

The regression tree algorithm chooses the split that **minimizes the weighted child-node MSE** (equivalently maximizes reduction in SSE/MSE). We computed:

- Weighted MSE for **Split A** ≈ **50.4444**
- Weighted MSE for **Split B** ≈ **10.4444**

Since **MSE(_B) ≪ MSE(_A)**, Split B (Items ≤ 2) achieves a much larger reduction in prediction error and therefore is the better split.

---

1. **Final concise answers (boxed)**

- **Split A (Distance ≤ 3.0 km):**

  Left MSE ≈ **28.6667**, Right MSE ≈ **72.2222**, Weighted MSE ≈ **50.4444**.

- **Split B (Items ≤ 2):**

  Left MSE ≈ **4.2222**, Right MSE ≈ **16.6667**, Weighted MSE ≈ **10.4444**.

- **Decision:** The tree would choose **Split B (Items ≤ 2)** as the root split because it produces the **lower weighted MSE** (≈10.44 vs 50.44), i.e., it yields much purer child nodes and better predictive performance.

These concepts are extremely important for understanding how **Decision Trees for regression** work internally.

✅ 1. Regression Trees Predict Using the Mean of Y

In **regression trees**, each leaf node predicts a **numeric value** (e.g., delivery time).

The prediction at a leaf is always:

$$\hat{y}_{\text{leaf}} = \text{mean of all target values in that leaf.}$$

Why mean?

It minimizes the squared error within that leaf.

For example, if a leaf has delivery times `[30, 35, 22]`, the leaf predicts:

$$\bar{y} = \frac{30 + 35 + 22}{3} = 29.$$

✅ 2. Impurity for Regression Trees = MSE (or SSE)

Decision trees need a measure of "impurity" or "error" to decide splits.

For classification

→ impurity = entropy or Gini.

For regression

→ impurity = MSE, defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2.$$

Intuition:

If numbers in the node are very close to each other, MSE is **small** → good leaf.

If numbers are scattered, MSE is **large** → bad leaf.

✅ 3. Why Decision Trees Try Many Possible Splits

A tree tries different ways to divide the dataset so that the resulting subsets have **low MSE**.

In this question, the algorithm tries:

- Split A: `Distance ≤ 3.0`
- Split B: `Items ≤ 2`

The tree checks **which split makes leaf nodes more pure / lower-variance**.

✅ 4. Splitting Creates Two Child Nodes

Splits create:

- Left node (matching the condition)
- Right node (not matching the condition)

Each child node has its own:

- mean,
- squared errors,
- MSE.

The tree evaluates both child nodes.

✅ 5. Weighted Average MSE After the Split

The algorithm computes:

$$\text{Split MSE} = \frac{n_L}{n} MSE_L + \frac{n_R}{n} MSE_R$$

where:

- (n_L) = number of samples in the left child
- (n_R) = number of samples in the right child
- (n) = total samples (here 6)

Why weighted?

Because a large node should matter more than a small node.

✅ 6. The Best Split Minimizes MSE

A regression tree ALWAYS chooses the split with the **lowest weighted MSE** (highest reduction in error).

This is the exact analogue of:

- Information Gain in Classification

- Reduction in variance in mathematics

In your example:

- Weighted MSE(Split A) ≈ **50.44**
- Weighted MSE(Split B) ≈ **10.44**

Since **10.44 < 50.44**, Split B produces leaf nodes with much lower internal variation, so it's preferred.

✅ 7. Why MSE Reduction Works

MSE reduction = leaf values become more similar.

Trees aim to create regions of the input space where:

- delivery times are predictable,
- variance is low,
- the target values cluster tightly.

That's what helps the tree predict better.

🔍 Putting It All Together (Concept Flow)

1. A split divides data into two subsets.
2. Each subset gets its own prediction = mean target value.
3. MSE measures how good each subset is.
4. Weighted MSE combines them.
5. The tree picks the split that reduces MSE the most.

This concept is called:

"Variance Reduction"

and is the core criterion for regression trees.

⭐ Final Summary of Concepts

- Regression trees use **MSE** (or SSE) as impurity.
- Each leaf predicts the **mean** of its target values.
- Splitting the data creates two nodes with their own MSE.
- The split is evaluated using **weighted MSE** of children.
- The best split = **lowest weighted MSE** = largest reduction in variance.

Problem 17

In which of the following scenarios would a Linear Regression model be strongly preferred over a Decision Tree model?

(a) The relationship between features and the target is highly non-linear.

(b) Feature values are significantly outside the range of the training data.

(c) The dataset contains a mix of categorical and numerical features, and you want to avoid manual one-hot encoding.

(d) The dataset is noisy, and needs the interpretability of the coefficients of the model.

We evaluate each option based on when **Linear Regression** is clearly better than a **Decision Tree**.

The key ideas:

- Linear Regression → **low variance, good extrapolation, globally interpretable**
- Decision Trees → **high variance, poor extrapolation, piecewise-constant predictions**

(a) The relationship is highly non-linear

- Linear Regression assumes a **linear relationship**.
- A Decision Tree naturally models **non-linear** changes and interactions.

Conclusion:

Linear Regression is **not** preferred here.

❌ **Not a correct choice.**

(b) Feature values are far outside the training range

Important conceptual difference:

- **Decision Trees cannot extrapolate.**

  They always predict **within the range** of training targets because leaves output a constant (mean of that leaf).

- **Linear Regression extrapolates smoothly.**

  Its prediction ($\hat{y} = w^T x + b$) continues the line beyond seen values.

Example:

Training data only has incomes from 20k to 80k. A new sample has 300k income.

- Tree → outputs some value inside the 20k–80k region.
- LR → outputs a reasonable extension of the trend.

Conclusion:

Linear Regression is clearly preferred.

✅ **Correct choice.**

(c) The dataset has mixed categorical + numerical features, and you want to avoid one-hot encoding

- Linear Regression **requires** one-hot encoding for categorical features.
- Decision Trees can naturally handle categorical attributes (after simple encoding).

Conclusion:

This situation favors **Decision Trees**, not Linear Regression.

❌ **Not a correct choice.**

(d) The dataset is noisy, and you want the interpretability of the coefficients.

- Decision Trees are **high-variance** → very sensitive to noise → unstable splits.
- Linear Regression is **low-variance** and much more robust to noise.
- LR provides **global interpretability** through coefficients.

Conclusion:

Linear Regression is strongly preferred for:

- stability
- noise-resistance
- clean interpretability

✅ **Correct choice.**

Final Answer

$$\boxed{\text{The correct scenarios are: (b) and (d).}}$$

- **(b)** LR can extrapolate; trees cannot.
- **(d)** LR is more stable under noise and provides interpretable coefficients.

Problem 18

Based on the two components given below, explain what are the differences between a Decision Tree Regressor and a Decision Tree Classifier.

(a) **Metrics:** Which one and how are they used by the model?

(b) **Leaf Node Prediction:** How are they calculated by the model?

(a) Metrics — which metrics each model uses and how they are used

Decision Tree Classifier (classification tree)

Uses **impurity measures** to decide splits:

1. **Gini Impurity**

   $G = 1 - \sum p_i^2$

2. **Entropy (Information Gain)**

   $H = -\sum p_i \log_2 p_i$

How used:

At each candidate split, the classifier computes:

- impurity of left child
- impurity of right child
- **weighted impurity after the split**

The model chooses the split that **minimizes impurity** (or maximizes Information Gain).

Intuition:

Classifier trees try to create child nodes that are **pure**, i.e., dominated by one class.

Decision Tree Regressor (regression tree)

Uses **variance-based metrics** instead of class impurity:

1. **Mean Squared Error (MSE)**

   $\text{MSE} = \frac{1}{n}\sum (y_i - \bar{y})^2$

2. **Sum of Squared Errors (SSE)**

   $\text{SSE} = \sum (y_i - \bar{y})^2$

How used:

At each candidate split, the regressor computes:

- mean target value in each child
- squared errors around that mean
- **weighted MSE of the two child nodes**

It chooses the split that **reduces MSE the most** (i.e., largest variance reduction).

Intuition:

Regression trees try to create child nodes where the **target values are close together**.

(b) Leaf Node Prediction — how predictions are made

Decision Tree Classifier

Prediction at a leaf = **the majority class** among the samples in that leaf.

$\hat{y}_{\text{leaf}} = \arg\max_c ; \text{count}(c)$

Optionally, for probabilities:

$$P(y = c \mid \text{leaf}) = \frac{\text{count of class } c}{\text{total samples in that leaf}}$$

Interpretation:

The leaf "votes" by taking the class that appears most often.

Decision Tree Regressor

Prediction at a leaf = **the mean of target values** in that leaf.

$$\hat{y}_{\text{leaf}} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Why mean?

The mean minimizes the squared error, which is the loss function used by regression trees.

Interpretation:

The leaf estimates the "average" value of all samples grouped there.

Final Summary (concise)

(a) Metrics

- **Classifier:** Uses *Gini impurity* or *Entropy* to measure how mixed the classes are.
- **Regressor:** Uses *MSE* or *SSE* to measure how spread out the numeric targets are.

(b) Leaf Node Prediction

- **Classifier:** Predicts the **majority class** (or class probabilities).
- **Regressor:** Predicts the **mean target value** of the samples in that leaf.
