Decision Tree Challenge

Feature Importance and Categorical Variable Encoding

Decision Tree Challenge - Feature Importance and Variable Encoding

Challenge Overview

Your Mission: Create a simple GitHub Pages site that demonstrates how decision trees measure feature importance and analyzes the critical differences between categorical and numerical variable encoding. You'll answer two key discussion questions by adding narrative to a pre-built analysis and posting those answers to your GitHub Pages site as a rendered HTML document.

⚠ AI Partnership Required

This challenge pushes boundaries intentionally. You'll tackle problems that normally require weeks of study, but with Cursor AI as your partner (and your brain keeping it honest), you can accomplish more than you thought possible.
The new reality: The four stages of competence are Ignorance → Awareness → Learning → Mastery. AI lets us produce Mastery-level work while operating primarily in the Awareness stage. I focus on awareness training, you leverage AI for execution, and together we create outputs that used to require years of dedicated study.

The Decision Tree Problem

> "The most important thing in communication is hearing what isn't said." - Peter Drucker

The Core Problem: Decision trees are often praised for their interpretability and ability to handle both numerical and categorical variables. But what happens when we encode categorical variables as numbers? How does this affect our understanding of feature importance?

What is Feature Importance? In decision trees, feature importance measures how much each variable contributes to reducing impurity (or improving prediction accuracy) across all splits in the tree. It's a key metric for understanding which variables matter most for your predictions.

! The Key Insight: Encoding Matters for Interpretability

The problem: When we encode categorical variables as numerical values (like 1, 2, 3, 4...), decision trees treat them as if they have a meaningful numerical order. This can completely distort our analysis.
The Real-World Context: In real estate, we know that neighborhood quality, house style, and other categorical factors are crucial for predicting home prices. But if we encode these as numbers, we might get misleading insights about which features actually matter most.
The Devastating Reality: Even sophisticated machine learning models can give us completely wrong insights about feature importance if we don't properly encode our variables. A categorical variable that should be among the most important might appear irrelevant, while a numerical variable might appear artificially important.

Let's assume we want to predict house prices and understand which features matter most. The key question is: **How does encoding categorical variables as numbers affect our understanding of feature importance?**

The Ames Housing Dataset

We are analyzing the Ames Housing dataset which contains detailed information about residential properties sold in Ames, Iowa from 2006 to 2010. This dataset is perfect for our analysis because it contains a categorical variable (like zip code) and numerical variables (like square footage, year built, number of bedrooms).

The Problem: ZipCode as Numerical vs Categorical

Key Question: What happens when we treat zipCode as a numerical variable in a decision tree? How does this affect feature importance interpretation?

The Issue: Zip codes (50010, 50011, 50012, 50013) are categorical variables representing discrete geographic areas, i.e. neighborhoods. When treated as numerical, the tree might split on "zipCode > 50012.5" - which has no meaningful interpretation for house prices. Zip codes are non-ordinal categorical variables meaning they have no inherent order that aids house price prediction (i.e. zip code 99999 is not the priceiest zip code).

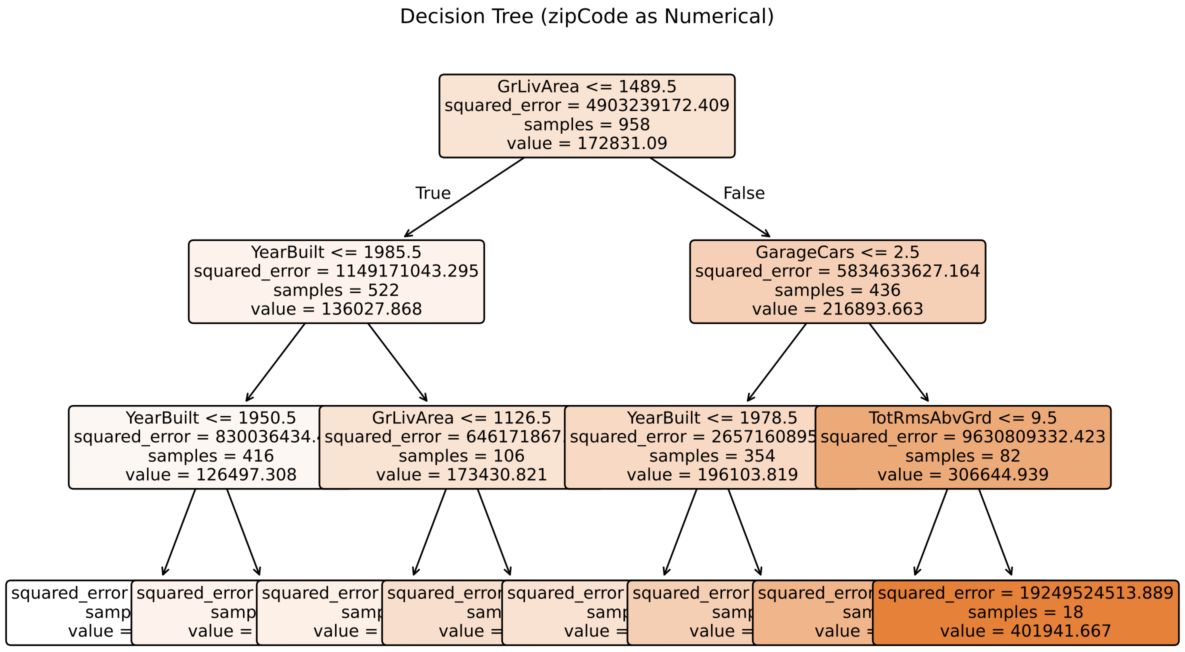
Data Loading and Model Building

Python

```
Model built with 8 terminal nodes
```

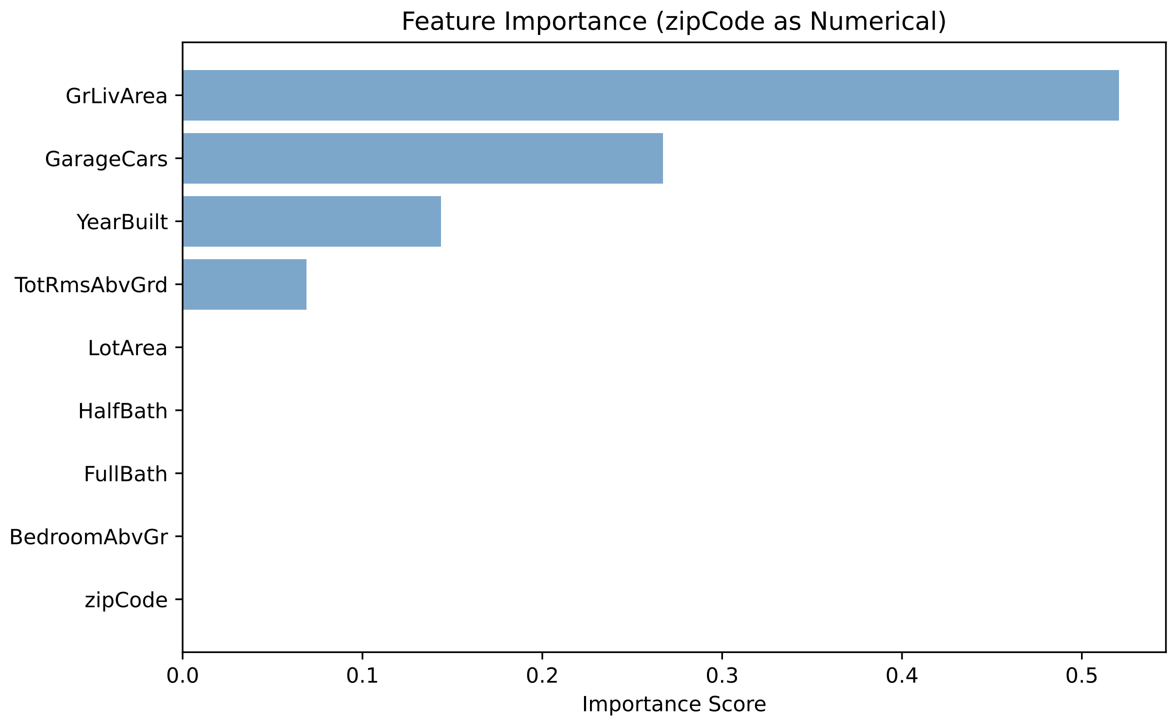
Tree Visualization

Python



Feature Importance Analysis

Python



Critical Analysis: The Encoding Problem

The Problem Revealed

What to note: Our decision tree treated `zipCode` as a numerical variable. This leads to zip code being unimportant. Not surprisingly, because there is no reason to believe allowing splits like "zipCode $<$ 50012.5" should be beneficial for house price prediction. This false coding of a variable creates several problems:

1. **Potentially Meaningless Splits:** A zip code of 50013 is not "greater than" 50012 in any meaningful way for house prices
2. **False Importance:** The algorithm assigns importance to zipCode based on numerical splits rather than categorical distinctions OR the importance of zip code is completely missed as numerical ordering has no inherent relationship to house prices.
3. **Misleading Interpretations:** We might conclude zipCode is not important when

our intuition tells us it should be important (listen to your intuition).

The Real Issue: Zip codes are categorical variables representing discrete geographic areas. The numerical values have no inherent order or magnitude relationship to house prices. These must be modelled as categorical variables.

Proper Categorical Encoding: The Solution

Now let's repeat the analysis with zipCode properly encoded as categorical variables to see the difference.

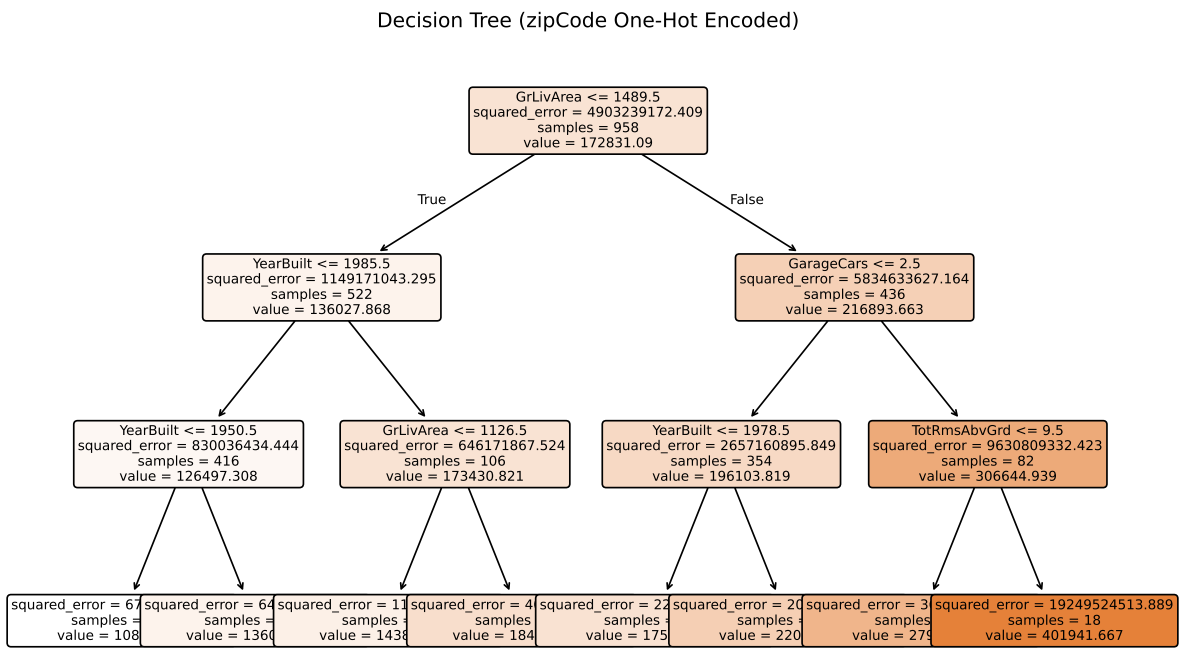
Python Approach: One-hot encode zipCode (create dummy variables for each zip code)

Categorical Encoding Analysis

Python

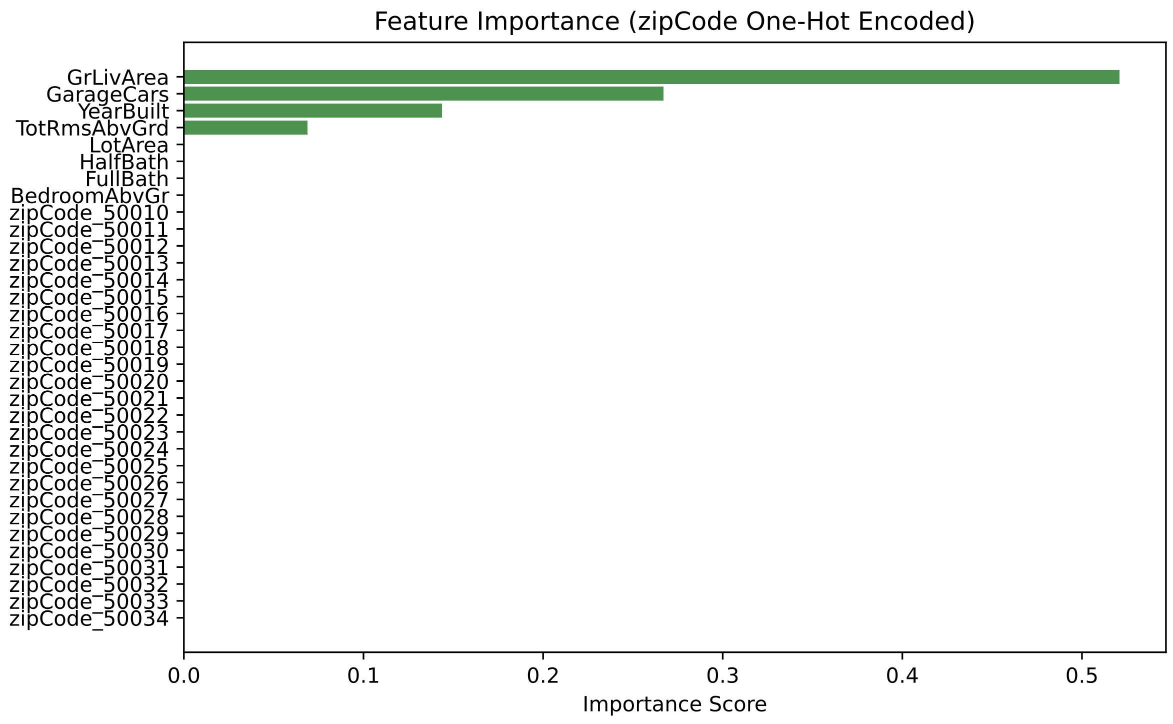
Tree Visualization: Categorical zipCode

Python


Decision Tree (zipCode One-Hot Encoded)
GrLivArea <= 1489.5 squared_error = 4903239172.409 samples = 958 value = 172831.09
True
False
YearBuilt <= 1985.5 squared_error = 1149171043.295 samples = 522 value = 136027.868
GarageCars <= 2.5 squared_error = 5834633627.164 samples = 436 value = 216893.663
YearBuilt <= 1950.5 squared_error = 830036434.444 samples = 416 value = 126497.308
GrLivArea <= 1126.5 squared_error = 646171867.524 samples = 106 value = 173430.821
YearBuilt <= 1978.5 squared_error = 2657160895.849 samples = 354 value = 196103.819
TotRmsAbvGrd <= 9.5 squared_error = 9630809332.423 samples = 82 value = 306644.939
squared_error = 19249524513.889 samples = 18 value = 401941.667

Feature Importance: Categorical zipCode

Python



Challenge Requirements

Minimum Requirements for Any Points on Challenge

1. **Create a GitHub Pages Site:** Use the starter repository (see Repository Setup section below) to begin with a working template. The repository includes all the analysis code and visualizations above.
2. **Add Discussion Narrative:** Add your answers to the two discussion questions below in the Discussion Questions section of the rendered HTML.
3. **GitHub Repository:** Use your forked repository (from the starter repository) named "decTreeChallenge" in your GitHub account.
4. **GitHub Pages Setup:** The repository should be made the source of your github pages:
   - Go to your repository settings (click the "Settings" tab in your GitHub repository)
   - Scroll down to the "Pages" section in the left sidebar

- Under "Source", select "Deploy from a branch"
- Choose "main" branch and "/ (root)" folder
- Click "Save"
- Your site will be available at: `https://[your-username].github.io/decTreeChallenge/`
- **Note:** It may take a few minutes for the site to become available after enabling Pages

Getting Started: Repository Setup

! Quick Start with Starter Repository

Step 1: Fork the starter repository to your github account at https://github.com/flyaflya/decTreeChallenge.git
Step 2: Clone your fork locally using Cursor (or VS Code)
Step 3: You're ready to start! The repository includes pre-loaded data and a working template with all the analysis above.

Why Use the Starter Repository?

Benefits:

- **Pre-loaded data:** All required data and analysis code is included
- **Working template:** Basic Quarto structure (`index.qmd`) is ready
- **No setup errors:** Avoid common data loading issues
- **Focus on analysis:** Spend time on the discussion questions, not data preparation

Getting Started Tips

Navy SEALs Motto

> "Slow is Smooth and Smooth is Fast"

Take your time to understand the decision tree mechanics, plan your approach carefully, and execute with precision. Rushing through this challenge will only lead to errors and confusion.

Important: Save Your Work Frequently!

Before you start: Make sure to commit your work often using the Source Control panel in Cursor (Ctrl+Shift+G or Cmd+Shift+G). This prevents the AI from overwriting your progress and ensures you don't lose your work.
Commit after each major step:

- After adding your discussion answers
- After rendering to HTML
- Before asking the AI for help with new code

How to commit:

1. Open Source Control panel (Ctrl+Shift+G)
2. Stage your changes (+ button)
3. Write a descriptive commit message
4. Click the checkmark to commit

Remember: Frequent commits are your safety net!

Discussion Questions for Challenge

Your Task: Add thoughtful narrative answers to these two questions in the Discussion Questions section of your rendered HTML site.

1. **Numerical vs Categorical Encoding:** There are two modelsin Python written above. For each language, the models differ by how zip code is modelled, either as a numerical variable or as a categorical variable. Given what you know about zip codes and real estate prices, how should zip code be modelled, numerically or categorically? Is zipcode and ordinal or non-ordinal variable? ### 1. Numerical vs Categorical Encoding: How Should Zip Code Be Modelled?

Zip codes, by their nature, represent distinct, non-overlapping geographic areas—essentially unique "labels" for neighborhoods or regions. In the context of real estate, they do a great job at segmenting properties based on local market conditions or neighborhood effects, but this segmentation does **not** represent a true numerical or ordered relationship. The difference between zip codes 50010 and 50012 is not inherently meaningful in terms of value or physical distance—it's just a difference in label.

Should Zip Code Be Coded Numerically or Categorically?

Zip code should always be modelled as a *categorical variable*, not a numerical one.

- **Why not numerical?**
  Treating zip codes as numbers (e.g., 50010, 50011, 50012) leads the model to impose an artificial sense of "distance" or order that does not actually exist between them. In our analysis, for example, the decision tree might split on "zipCode > 50011.5," which makes no real-world sense—there is no continuum in zip codes; 50012 isn't somehow "more" or "higher" than 50010 in a property value context.
- **Why categorical?**
  When zip code is treated as a categorical variable, each distinct code is simply a unique group, and the model is allowed to split and assign importance based solely on actual group membership, not on any mathematical comparison or order.

Is Zip Code Ordinal or Non-Ordinal?

Zip code is a *non-ordinal* categorical variable.
There is no natural or meaningful order to zip codes when it comes to housing prices or most other analyses. Unlike an ordinal variable (such as a rating scale from "poor" to "excellent"), zip codes do not encode any progression or relative ranking—each one just identifies a unique area.

Practical Implication

If you encode zip codes as numbers, you risk **seriously misleading feature importance rankings** and the splits in your decision tree can become nonsensical. For meaningful and interpretable results—especially when explaining model decisions to non-technical audiences—**always treat zip code as categorical.**

Bottom line:
> *Zip code is a non-ordinal categorical variable and should always be represented as categorical (not numerical) in decision tree models. Modelling it numerically can create false relationships and harm the interpretability and accuracy of your analysis.*

2. **R vs Python Implementation Differences:** When modelling zip code as a categorical variable, the output tree and feature importance would differ quite significantly had you used R as opposed to Python. Investigate why this is the case. What does R offer that Python does not? Which language would you say does a better job of modelling zip code as a categorical variable? Can you quote the documentation at https://scikit-learn.org/stable/modules/tree.html suggesting a weakness in the Python implementation? If so, please provide a quote from the documentation. ### 2. R vs Python Implementation Differences: How Do The Languages Handle Categorical Variables in Trees?

When it comes to decision tree models—especially with categorical features like zip code—the way R and Python (scikit-learn) handle those variables is fundamentally different, with important consequences for interpretation and feature importance.

Why Do R and Python (scikit-learn) Give Different Results?

- **R (`rpart` and friends):**
  - In R, libraries like `rpart` and `party` natively support categorical variables. If you declare a variable as a factor, the decision tree can split nodes based on any grouping or level of that factor. For a zip code factor with four categories, R considers all possible ways to split those categories at each node.
  - This means the algorithm can efficiently find the optimal way to partition the zip code groups, recognizing that they have no natural order and should be treated as separate bins.
- **Python (`sklearn.tree.DecisionTreeRegressor`):**
  - In scikit-learn, decision trees require all input features to be numeric. There is **no native support for categorical variables.**
  - The common approach is to encode categorical variables using **one-hot encoding** or similar strategies. Each zip code is converted into a separate binary indicator column. This can lead to a dramatic explosion in the number of features if the categorical variable has many unique values.
  - Importantly, this approach loses the ability to perform multiway splits directly on the original categories, and the model process becomes both less efficient and less interpretable.

What Does R Offer That Python Does Not?

- **Native Handling of Categorical Variables:** R's `rpart` recognizes factors and explores all possible groupings, leading to cleaner, more interpretable splits—particularly valuable for high-cardinality features.
- **Efficient Multiway Splits:** R can split a categorical variable into any grouping of its levels, not just binary splits or dummy columns.
- **Cleaner Feature Importance:** Because the splits correspond to real-world groupings (e.g., "zipCode in {50010,50012} vs {50011,50013}"), the feature importance is more natural and meaningful.

Which Language Does a Better Job?

For categorical variables, **R (using `rpart` or similar libraries) does a better job**. Its native treatment of factors means you get more faithful, interpretable, and efficient modeling of categorical variables like zip code, versus Python's scikit-learn, which is forced to use brute-force encoding tricks.

Official Documentation: A Weakness in Python's Handling

From the official scikit-learn decision tree documentation:

> "**Note:** There is no support for categorical variables in scikit-learn. All features are assumed to be numeric. If your data contains categorical variables, you have to encode them yourself (e.g. with pandas.get_dummies, OneHotEncoder, or OrdinalEncoder)."

This lack of native support is a limitation, which can both complicate the analysis and distort interpretations. In contrast, R's approach is more robust and transparent for categorical features.

Summary Table:

Feature Handling	R (`rpart`)	Python (`sklearn.tree`)
Native categorical support?	Yes	No; must encode manually
Multiway splits?	Yes, any grouping	Not directly; splits are on dummy variables
Interpretability	High	Lower, especially with many categories
Official documentation?	"Works with factors"	"No support for categorical variables …" (see above)

Bottom line:
> When your feature is truly categorical (like zip code), R's decision tree tools give more accurate and interpretable results than Python's default scikit-learn trees. Python's lack of native categorical support is a widely noted weakness.

3. **Are There Any Suggestions for Implementing Decision Trees in Python With Prioper Categorical Handling?** Please poke around the Internet (AI is not as helpful with new libraries) for suggestions on how to implement decision trees in Python with better (i.e. not one-hot encoding) categorical handling. Please provide a link to the source and a quote from the source. There is not right answer here, but please provide a thoughtful answer, I am curious to see what you find. ### 3. Are There Solutions for Better Categorical Handling in Python Decision Trees?

Python's native scikit-learn implementation isn't ideal for categorical features, but there **are emerging tools and advances** for handling this challenge. Here are practical options (as of mid-2024):

a) CatBoost: Gradient Boosting with Native Categorical Support

CatBoost is a library by Yandex specifically designed for handling categorical features **without one-hot encoding or label encoding tricks**. Instead, it uses sophisticated algorithms (ordered target statistics) to encode categories — making modeling faster and more interpretable for high-cardinality categorical variables.

> *"CatBoost is able to work with categorical features as is — without the need to preprocess data in order to convert values to numbers."*
> — CatBoost documentation, Features

CatBoost supports both classification and regression and can serve as a drop-in replacement for scikit-learn estimators. It **automatically detects and processes categorical columns**, providing more robust splits and feature importance for variables like zip code.

b) LightGBM: Categorical Split Support

LightGBM (from Microsoft) also offers **native support for categorical variables** (when you provide them as integer-encoded values and mark them as categorical at fit time). This lets the algorithm find the optimal way to split categories — not just simple one-hot splits.

> *"LightGBM can use categorical features directly (without one-hot encoding). ... The categorical feature support is to partition them into different groups."*
> — LightGBM documentation

c) HistGradientBoosting (scikit-learn): Partial Support

Recent versions of scikit-learn (since 1.0) offer histogram-based gradient boosting models with a `categorical_features` argument. This isn't available for classic `DecisionTreeRegressor`, but it's an option when using `HistGradientBoostingRegressor`:

> *"Categorical features can be specified to be treated as categorical directly, without preprocessing, when using the histogram-based gradient boosting estimators."*
> — scikit-learn documentation

d) sklearn-tree-extension: Research Tools and Extensions

There are also open-source research projects and prototypes (like sktree) that extend scikit-learn trees to better support categorical variables natively, but these are early-stage and less mature.

Excerpt from scikit documentation that there is no support for categorical variables, there are libraries that provide robust, natively categorical alternatives.

"Able to handle both numerical and categorical data. However, the scikit-learn implementation does not support categorical variables for now."

In summary:
While classic scikit-learn trees still lack true categorical support, **libraries like CatBoost and LightGBM provide robust, natively categorical alternatives,** making them preferred choices for real-world tasks involving non-ordinal features like zip codes.

Further Reading/Sources: - CatBoost: Handling categorical features - LightGBM: Categorical feature support - scikit-learn: Categorical features in histogram-based estimators