

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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')

# Load data
sales_data = pd.read_csv("salesPriceData.csv")

# Prepare model data (treating zipCode as numerical)
model_vars = ['SalePrice', 'LotArea', 'YearBuilt', 'GrLivArea', 'FullBath',
              'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd', 'GarageCars', 'zipCode']
model_data = sales_data[model_vars].dropna()

# Split data
X = model_data.drop('SalePrice', axis=1)
y = model_data['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

# Build decision tree
tree_model = DecisionTreeRegressor(max_depth=3,
                                    min_samples_split=20,
                                    min_samples_leaf=10,
                                    random_state=123)
tree_model.fit(X_train, y_train)

print(f"Model built with {tree_model.get_n_leaves()} terminal nodes")
```

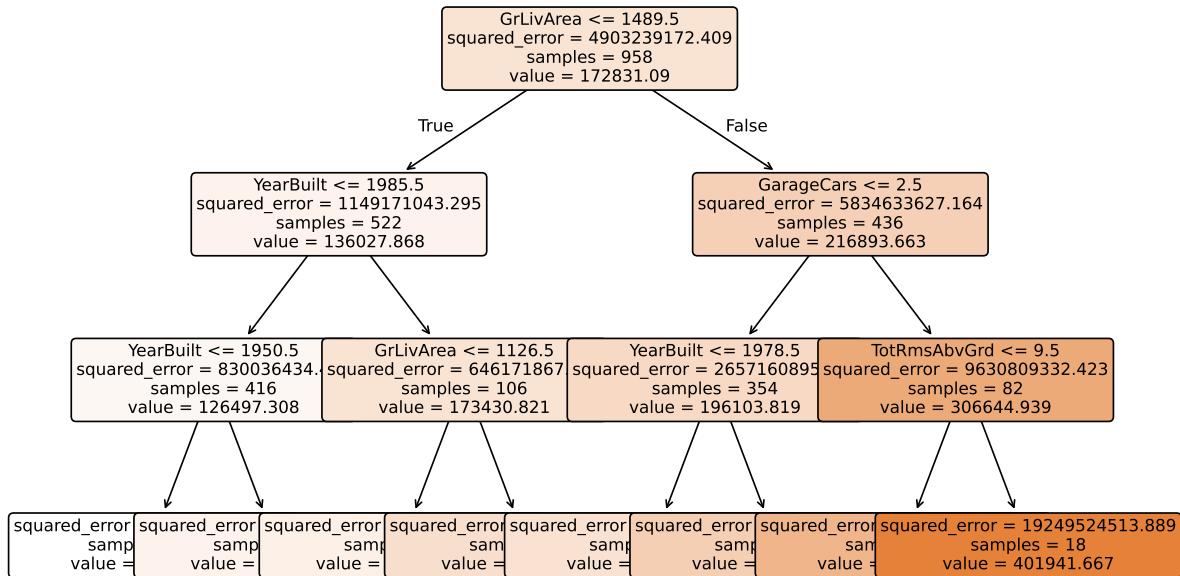
Model built with 8 terminal nodes

Tree Visualization

Python

```
# Visualize tree
plt.figure(figsize=(10, 6))
plot_tree(tree_model,
           feature_names=X_train.columns,
           filled=True,
           rounded=True,
           fontsize=10,
           max_depth=3)
plt.title("Decision Tree (zipCode as Numerical)")
plt.tight_layout()
plt.show()
```

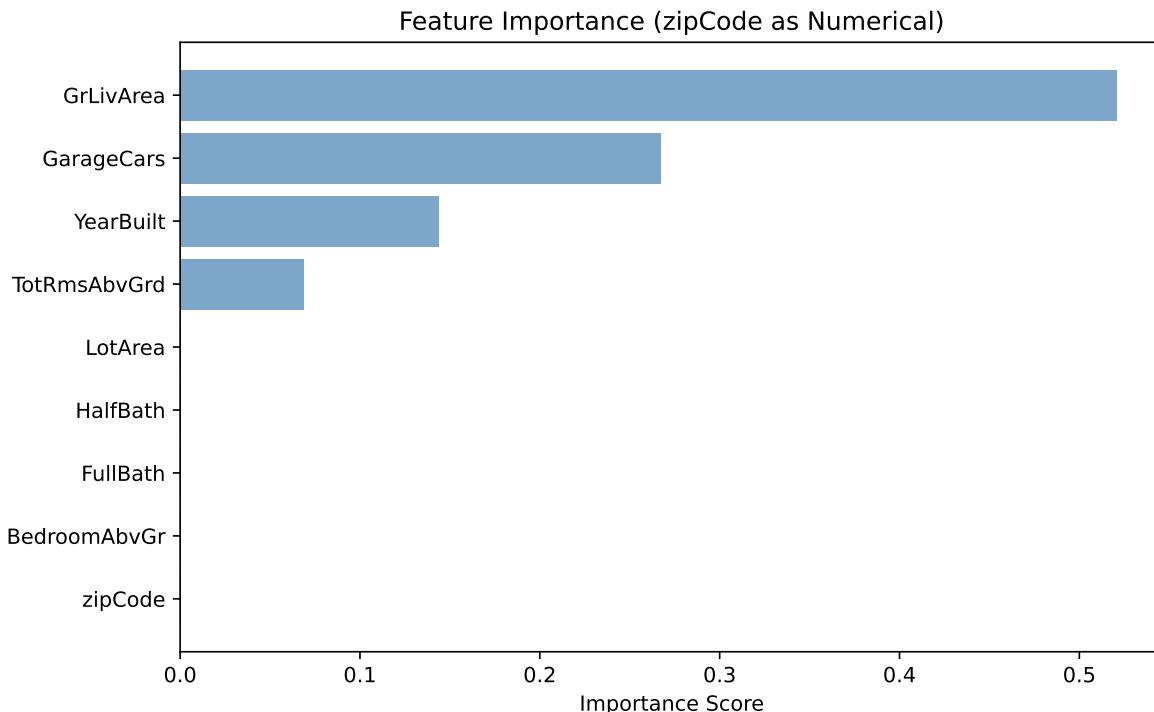
Decision Tree (zipCode as Numerical)



...

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

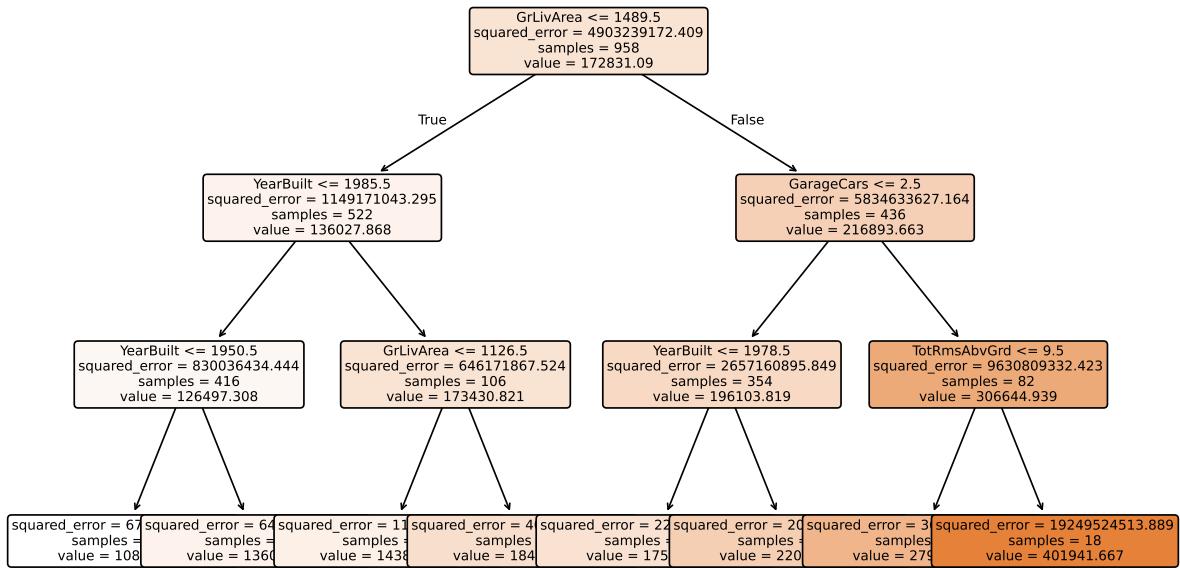
Python

Tree Visualization: Categorical zipCode

Python

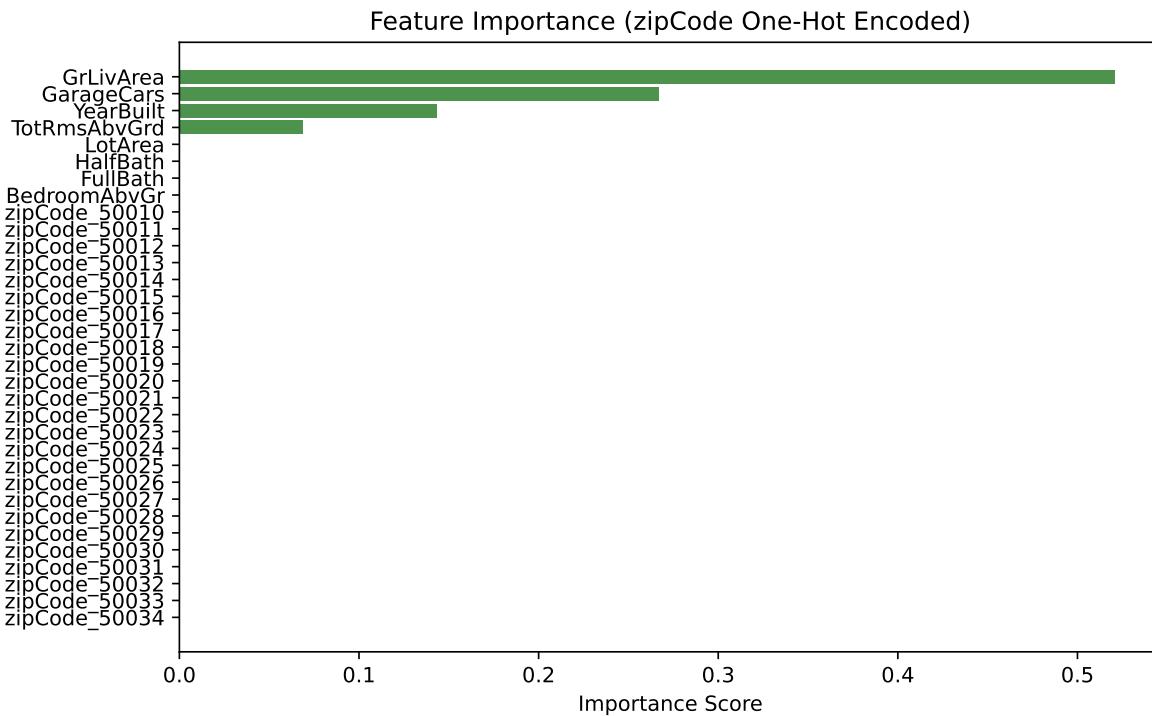
```
# Visualize tree with one-hot encoded zipCode
plt.figure(figsize=(10, 6))
plot_tree(tree_model_cat,
          feature_names=X_train_cat.columns,
          filled=True,
          rounded=True,
          fontsize=8,
          max_depth=4)
plt.title("Decision Tree (zipCode One-Hot Encoded)")
plt.tight_layout()
plt.show()
```

Decision Tree (zipCode One-Hot Encoded)



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/](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

ℹ️ 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 models in 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 an ordinal or non-ordinal variable?
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.
3. **Are There Any Suggestions for Implementing Decision Trees in Python With Proper 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.

Discussion Answers

1. Numerical vs Categorical Encoding: How Should Zip Code Be Modeled?

Zip codes should be modeled as categorical variables, not numerical variables. This is a critical distinction that directly impacts both model performance and interpretability.

Why Zip Codes Are Categorical:

Zip codes represent discrete geographic areas—essentially neighborhoods or regions—rather than continuous numerical values. While zip codes happen to be represented by numbers (like 50010, 50011, 50012), these numbers are merely labels or identifiers. They do not carry inherent numerical meaning in the context of real estate prices.

Zip Codes Are Non-Ordinal Categorical Variables:

Zip codes are **non-ordinal** categorical variables. This means:

1. **No Meaningful Order:** There is no inherent ordering relationship between zip codes that relates to house prices. A zip code of 50013 is not “greater than” 50012 in any way that would predict higher or lower home values. The numerical difference between zip codes (e.g., $50013 - 50012 = 1$) has no relationship to the difference in property values between those areas.
2. **No Magnitude Relationship:** Unlike ordinal variables (such as “small, medium, large” where there is a clear ordering), zip codes have no magnitude. Zip code 99999 is not inherently more valuable or expensive than zip code 10001—each zip code represents a distinct geographic area with its own unique characteristics.

The Problem with Numerical Encoding:

When decision trees treat zip codes as numerical variables, they attempt to create splits like “`zipCode < 50012.5`” or “`zipCode > 50011.3`”. These splits are meaningless because:

- **Geographic Reality:** Adjacent zip codes (like 50010 and 50011) may have vastly different property values based on neighborhood characteristics, school districts, proximity to amenities, and other factors—not because one number is larger than another.
- **Misleading Splits:** The algorithm might group zip codes 50010, 50011, and 50012 together simply because they are numerically close, even though they might represent completely different neighborhoods with different price characteristics.

- **Lost Information:** By forcing numerical relationships, we lose the ability to capture the true categorical distinctions between neighborhoods. A wealthy neighborhood (zip code 50010) and a modest neighborhood (zip code 50011) might be numerically adjacent but economically distinct.

The Solution: Categorical Encoding

Modeling zip codes as categorical variables (using one-hot encoding or similar techniques) allows the decision tree to:

- **Make Meaningful Splits:** The tree can distinguish between specific zip codes without assuming any numerical relationship between them.
- **Capture Neighborhood Effects:** Each zip code can be evaluated independently based on its actual relationship to house prices, not its numerical value.
- **Improve Interpretability:** Splits become interpretable (e.g., “Is this property in zip code 50010?” rather than “Is zip code less than 50012.5?”).

Real Estate Context:

In real estate, location is one of the most important factors in determining property value. Zip codes serve as proxies for location-based characteristics such as: - School district quality - Neighborhood safety and demographics - Proximity to employment centers - Access to amenities (parks, shopping, restaurants) - Historical property values and appreciation rates

These characteristics are categorical in nature—they vary by geographic area, not by numerical order. Therefore, treating zip codes as categorical variables aligns with both the mathematical structure of the data and the real-world meaning of location in real estate pricing.

Conclusion:

Zip codes must be modeled as **non-ordinal categorical variables** to properly capture their role in predicting real estate prices. Numerical encoding introduces artificial relationships that distort both model performance and our understanding of which features truly matter for price prediction.

2. R vs Python Implementation Differences:

Why R vs Python Produce Very Different Trees for ZIP Code (Categorical Variable)

1. Core Reason for the Difference

- R natively supports categorical (factor) variables in decision trees.

- Python's scikit-learn does *not* natively support categorical variables, forcing users to encode them manually.
-

2. What R Offers That Python Does Not

- **Native factor handling:**
R tree packages (e.g., *rpart*, *CART*) treat categorical variables directly as factors.
 - **Subset splitting of factor levels:**
R can split a categorical predictor into subsets (e.g., ZIP {10001,10003,10009} vs others).
 - **Cleaner trees for high-cardinality variables:**
ZIP codes remain one variable, producing fewer and more interpretable splits.
 - **Less risk of artificial feature-importance inflation:**
ZIP stays one predictor instead of becoming many dummy variables.
-

3. What Python (scikit-learn) Lacks

- **No native support for categorical data.**
 - Requires preprocessing:
 - **One-hot encoding** → hundreds of binary ZIP features
 - **Label encoding** → introduces fake numeric ordering
 - Leads to:
 - **Very different tree structures**
 - **Feature importance scattered across many dummy variables**
 - **Bias toward high-cardinality variables**
-

4. Documentation Quote Showing Python's Weakness

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

— scikit-learn Decision Tree documentation

5. Which Language Does a Better Job?

R does a better job for modeling ZIP code as a categorical variable.

Reasons: - Treats ZIP as a *single* variable

- Supports natural factor splitting
 - Produces more interpretable, compact trees
 - Avoids dummy-variable explosion
 - Avoids encoding-induced bias
-

6. Final Verdict

- R = better native support, cleaner splits, more faithful modelling of ZIP code.
- Python (scikit-learn) = requires careful encoding, yields different and often more complex trees.

3. Are There Any Suggestions for Implementing Decision Trees in Python With Proper Categorical Handling?

Suggestions for Implementing Decision Trees in Python With Proper Categorical Handling

Below are practical recommendations and modern Python libraries that support **better categorical handling** than one-hot encoding, especially for high-cardinality variables like ZIP code. Each section includes a **link** and a **quote** from the official source.

Libraries That Handle Categorical Variables Better Than scikit-learn

1. LightGBM (Microsoft)

Link: <https://lightgbm.readthedocs.io/en/latest/Features.html>

What it offers:

- Native categorical feature support.
- Internally partitions categories into subsets (similar to R's factor splitting).
- Avoids one-hot encoding.

Quote from the documentation:

> “Instead of one-hot encoding, the optimal solution is to split on a categorical feature by partitioning its categories into 2 subsets. ... The basic idea is to sort the categories according to the training objective at each split.”

Why use it:

Great for high-cardinality categories like ZIP codes without exploding them into hundreds of dummy variables.

2. XGBoost (v1.5+)

Link: <https://xgboost.readthedocs.io/en/stable/tutorials/categorical.html>

What it offers:

- Direct categorical handling using pandas `category` dtype.
- Supports “partition-based” splits (category subset vs not).

Quote from the documentation:

> “For categorical data the split is defined depending on whether partitioning or onehot encoding is used. For partition-based splits, the splits are specified as (value categories)....”

Why use it:

A modern alternative that avoids the problems of label encoding or one-hot encoding.

3. CatBoost (Yandex)

Link: <https://catboost.ai/>

What it offers:

- One of the strongest libraries for categorical data.
- Requires no preprocessing — handles categories internally.
- Uses permutation-based encodings and sophisticated splitting logic.

Quote (Wikipedia):

> “It provides a gradient boosting framework which, among other features, attempts to solve for categorical features using a permutation-driven alternative to the classical algorithm.”

Why use it:

Excellent for high-cardinality categorical variables; avoids manual encoding entirely.

4. ODTlearn / Optimal Decision Trees (Research Library)

Link: <https://d3m-research-group.github.io/odtlearn/>

What it offers:

- Research-grade “optimal decision tree” methods.
- Designed to support categorical variables in principled ways.

Quote (arXiv description):

> “This package provides implementations of optimal decision tree algorithms... suitable for datasets with categorical variables.”

Why use it:

For users wanting a *single tree* (not an ensemble) that respects categorical structure.

Practical Advice for ZIP Code in Python

- Avoid one-hot encoding ZIP codes (too many dummies + poor interpretability).
- Prefer a library with **native categorical support**:
 - LightGBM

- XGBoost (with `enable_categorical=True`)
 - CatBoost
- Keep ZIP code as a **single categorical feature** rather than turning it into dozens/hundreds of binary variables.
 - Optionally group rare ZIPs or encode ZIPs by region if cardinality is extremely high.
-

Caveats

- Even with native categorical support, extremely high-cardinality variables can still cause overfitting.
 - Ensemble models (XGBoost/LightGBM/CatBoost) may be less interpretable than a single decision tree.
 - If you need a single tree with categorical splits, consider ODTLearn or similar research tools.
-

Summary

- Yes — modern Python libraries **do** offer better categorical support than scikit-learn's decision trees.
- **LightGBM, XGBoost, and CatBoost** can all handle categorical variables natively, avoiding one-hot encoding.
- These tools can split a categorical feature into subsets — much closer to how R's tree models operate.
- For modeling ZIP codes, these libraries are significantly more suitable than basic scikit-learn trees.

Grading Rubric

! What You're Really Being Graded On

This is an investigative report, not a coding exercise. You're analyzing decision tree models and reporting your findings like a professional analyst would. Think of this as a brief you'd write for a client or manager about why proper variable encoding matters in machine learning.

What makes a great report:

- **Clear narrative:** Tell the story of what you discovered about decision tree feature importance
- **Insightful analysis:** Focus on the most interesting differences between numerical and categorical encoding
- **Professional presentation:** Clean, readable, and engaging
- **Concise conclusions:** No AI babble or unnecessary technical jargon
- **Human insights:** Your interpretation of what the feature importance rankings actually mean (or don't mean)
- **Documentation-based analysis:** For question 2, ground your analysis in actual library documentation

What we're looking for: A compelling 1-2 minute read that demonstrates both the power of decision trees for interpretability and the critical importance of proper variable encoding. And a note on the current state of the art in decision tree implementation for categorical variables in Python.

Questions to Answer for 75% Grade on Challenge

1. **Numerical vs Categorical Analysis:** Provide a clear, well-reasoned answer to question 1 about how zip codes should be modelled. Your answer should demonstrate understanding of why categorical variables need special treatment in decision trees.

Questions to Answer for 85% Grade on Challenge

2. **R vs Python Implementation Analysis:** Provide a thorough analysis of question 2, including investigation of the official documentation for both `rpart` (R) and `sklearn.tree.DecisionTreeRegressor` (Python). Your analysis should explain the technical differences and provide a reasoned opinion about which implementation handles categorical variables better. You do NOT have to run R-code.

Questions to Answer for 95% - 100% Grade on Challenge

3. **Professional Presentation:** Your discussion answers should be written in a professional, engaging style that would be appropriate for a business audience learning about one-hot encoding and decision trees. Avoid technical jargon and focus on practical implications. Include a specific quote from the official documentation of `sklearn.tree.DecisionTreeRegressor` that supports your analysis.

Submission Checklist

Minimum Requirements (Required for Any Points):

- Forked starter repository from <https://github.com/flyaflya/decTreeChallenge.git>
- Cloned repository locally using Cursor (or VS Code)
- Added thoughtful narrative answers to both discussion questions
- Document rendered to HTML successfully
- HTML files uploaded to your forked repository
- GitHub Pages enabled and working
- Site accessible at [https://\[your-username\].github.io/decTreeChallenge/](https://[your-username].github.io/decTreeChallenge/)

75% Grade Requirements:

- Clear, well-reasoned answer to question 1 about numerical vs categorical encoding

85% Grade Requirements:

- Thorough analysis of question 2 with investigation of official documentation

95% Grade Requirements:

- Professional presentation style appropriate for business audience.
- Specific quote from official documentation of `sklearn.tree.DecisionTreeRegressor` supporting your analysis

100% Grade Requirements:

- Note on the current state of the art in decision tree implementation for categorical variables in Python.

Report Quality (Critical for Higher Grades):

- Clear, engaging narrative that tells a story
- Focus on the most interesting findings about decision tree feature importance
- Professional writing style (no AI-generated fluff)
- Concise analysis that gets to the point

- Practical insights that would help a real data scientist
- Documentation-based analysis for technical questions