Decision Tree Challenge

Feature Importance and Categorical Variable Encoding

Decision Tree Challenge - Feature Importance and Variable Encoding

Challenge Overview

Decision Tree Challenge - Feature Importance and Variable Encoding

Challenge Overview

Your Mission: Create a comprehensive Quarto document that demonstrates how decision trees measure feature importance, analyzes the critical differences between categorical and numerical variable encoding, and presents compelling evidence of why proper data preprocessing matters for interpretable machine learning. Then render the document to HTML and deploy it via GitHub Pages using the starter repository workflow.

The Decision Tree Problem

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 understanding of feature importance.
> **Why this matters:** If a categorical variable like "Zip Code" (50010, 50011, 50012, 50013) is treated as a numerical variable, the tree might split on "Zip Code > 50012.5" instead of recognizing that this is really a categorical choice between discrete geographic areas where the order of numerical values has no meaningful interpretation.
> **The connection:** Proper encoding preserves the true nature of categorical variables and gives us accurate feature importance rankings.

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.

Mistaken Feature Importance: The Example of Zip Code

Here we load a dataset that has a categorical variable "Zip Code" and a numerical variable "Sale Price" along with a bunch of other truly numerical variables. In this section, we will explore how the decision tree treats the "Zip Code" when it is mistakenly considered a numerical variable.

Data Loading and Initial Exploration

R

```r
# Load required libraries
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(rpart))

# Try to load rpart.plot, install if not available
if (!require(rpart.plot, quietly = TRUE)) {
  install.packages("rpart.plot", repos = "https://cran.rstudio.com/")
  library(rpart.plot)
}

# Load the Sales Price dataset
# Note: This loads the data from the buad442Fall2025 repository
```

```r
sales_data <- read.csv("https://raw.githubusercontent.com/flyaflya/buad442Fall2025/refs/heads

# Display basic information about the dataset
cat("Dataset dimensions:", dim(sales_data), "\n")
```

```
Dataset dimensions: 1198 11
```

```r
cat("Number of variables:", ncol(sales_data), "\n")
```

```
Number of variables: 11
```

```r
cat("Number of observations:", nrow(sales_data), "\n\n")
```

```
Number of observations: 1198
```

```r
# Show first few rows
head(sales_data, 10)
```

```
   SalePrice LotArea YearBuilt GrLivArea FullBath HalfBath BedroomAbvGr
1     208500    8450      2003      1710        2        1            3
2     181500    9600      1976      1262        2        0            3
3     223500   11250      2001      1786        2        1            3
4     250000   14260      2000      2198        2        1            4
5     143000   14115      1993      1362        1        1            1
6     307000   10084      2004      1694        2        0            3
7     200000   10382      1973      2090        2        1            3
8     118000    7420      1939      1077        1        0            2
9     129500   11200      1965      1040        1        0            3
10    144000   12968      1962       912        1        0            2
   TotRmsAbvGrd GarageCars avgAreaIncome zipCode
1             8          2         97203   50045
2             6          2        106363   50049
3             6          2         97203   50045
4             9          3        150805   50026
5             5          2         77061   50039
6             7          2        104653   50046
7             7          2         91626   50043
8             5          1         65080   50015
9             5          1         62302   50016
10            4          1         62302   50016
```

```r
# Examine data types and structure
str(sales_data)
```

```
'data.frame':   1198 obs. of  11 variables:
 $ SalePrice    : int  208500 181500 223500 250000 143000 307000 200000 118000 129500 144000
 $ LotArea      : int  8450 9600 11250 14260 14115 10084 10382 7420 11200 12968 ...
 $ YearBuilt    : int  2003 1976 2001 2000 1993 2004 1973 1939 1965 1962 ...
 $ GrLivArea    : int  1710 1262 1786 2198 1362 1694 2090 1077 1040 912 ...
 $ FullBath     : int  2 2 2 2 1 2 2 1 1 1 ...
 $ HalfBath     : int  1 0 1 1 1 0 1 0 0 0 ...
 $ BedroomAbvGr : int  3 3 3 4 1 3 3 2 3 2 ...
 $ TotRmsAbvGrd : int  8 6 6 9 5 7 7 5 5 4 ...
 $ GarageCars   : int  2 2 2 3 2 2 2 1 1 1 ...
 $ avgAreaIncome: int  97203 106363 97203 150805 77061 104653 91626 65080 62302 62302 ...
 $ zipCode      : int  50045 50049 50045 50026 50039 50046 50043 50015 50016 50016 ...
```

```r
# Summary statistics for key variables
summary(sales_data)
```

```
   SalePrice         LotArea         YearBuilt      GrLivArea   
 Min.   : 39300   Min.   :  1300   Min.   :1872   Min.   : 334  
 1st Qu.:130000   1st Qu.:  7544   1st Qu.:1952   1st Qu.:1110  
 Median :160000   Median :  9468   Median :1971   Median :1456  
 Mean   :175202   Mean   : 10543   Mean   :1969   Mean   :1493  
 3rd Qu.:205000   3rd Qu.: 11452   3rd Qu.:1998   3rd Qu.:1766  
 Max.   :755000   Max.   :215245   Max.   :2009   Max.   :4316  
    FullBath        HalfBath       BedroomAbvGr    TotRmsAbvGrd   
 Min.   :0.000   Min.   :0.0000   Min.   :0.000   Min.   : 2.000  
 1st Qu.:1.000   1st Qu.:0.0000   1st Qu.:2.000   1st Qu.: 5.000  
 Median :2.000   Median :0.0000   Median :3.000   Median : 6.000  
 Mean   :1.537   Mean   :0.3856   Mean   :2.875   Mean   : 6.447  
 3rd Qu.:2.000   3rd Qu.:1.0000   3rd Qu.:3.000   3rd Qu.: 7.000  
 Max.   :3.000   Max.   :2.0000   Max.   :6.000   Max.   :12.000  
   GarageCars    avgAreaIncome       zipCode     
 Min.   :0.000   Min.   : 51360   Min.   :50010  
 1st Qu.:1.000   1st Qu.: 64351   1st Qu.:50026  
 Median :2.000   Median : 77061   Median :50037  
 Mean   :1.725   Mean   : 85413   Mean   :50035  
 3rd Qu.:2.000   3rd Qu.: 97203   3rd Qu.:50045  
 Max.   :4.000   Max.   :150805   Max.   :50051  
```

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')

# Try to import seaborn, but don't fail if it's not available
try:
    import seaborn as sns
    sns.set_style("whitegrid")
except ImportError:
    print("Note: seaborn not available, using matplotlib defaults")

# Load the Sales Price dataset
# Note: This loads the data from the buad442Fall2025 repository
sales_data = pd.read_csv("https://raw.githubusercontent.com/flyaflya/buad442Fall2025/refs/hea

print("Dataset loaded successfully!")
```

```
Dataset loaded successfully!
```

```
# Display basic information about the dataset
print(f"Dataset dimensions: {sales_data.shape}")
```

```
Dataset dimensions: (1198, 11)
```

```
print(f"Number of variables: {sales_data.shape[1]}")
```

```
Number of variables: 11
```

```
print(f"Number of observations: {sales_data.shape[0]}\n")
```

```
Number of observations: 1198
```

```python
# Show first few rows
print("First 10 rows:")
```

```
First 10 rows:
```

```python
sales_data.head(10)
```

```
   SalePrice  LotArea  YearBuilt  ...  GarageCars  avgAreaIncome  zipCode
0     208500     8450       2003  ...           2          97203    50045
1     181500     9600       1976  ...           2         106363    50049
2     223500    11250       2001  ...           2          97203    50045
3     250000    14260       2000  ...           3         150805    50026
4     143000    14115       1993  ...           2          77061    50039
5     307000    10084       2004  ...           2         104653    50046
6     200000    10382       1973  ...           2          91626    50043
7     118000     7420       1939  ...           1          65080    50015
8     129500    11200       1965  ...           1          62302    50016
9     144000    12968       1962  ...           1          62302    50016

[10 rows x 11 columns]
```

```python
# Examine data types and structure
print("Data types:")
```

```
Data types:
```

```python
print(sales_data.dtypes)
```

```
SalePrice        int64
LotArea          int64
YearBuilt        int64
GrLivArea        int64
FullBath         int64
HalfBath         int64
BedroomAbvGr     int64
TotRmsAbvGrd     int64
GarageCars       int64
avgAreaIncome    int64
zipCode          int64
dtype: object
```

```python
print("\nData structure:")
```

```
Data structure:
```

```python
print(sales_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1198 entries, 0 to 1197
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SalePrice      1198 non-null   int64
 1   LotArea        1198 non-null   int64
 2   YearBuilt      1198 non-null   int64
 3   GrLivArea      1198 non-null   int64
 4   FullBath       1198 non-null   int64
 5   HalfBath       1198 non-null   int64
 6   BedroomAbvGr   1198 non-null   int64
 7   TotRmsAbvGrd   1198 non-null   int64
 8   GarageCars     1198 non-null   int64
 9   avgAreaIncome  1198 non-null   int64
 10  zipCode        1198 non-null   int64
dtypes: int64(11)
memory usage: 103.1 KB
None
```

```python
# Summary statistics for key variables
print("\nSummary statistics for key variables:")
```

```
Summary statistics for key variables:
```

```python
sales_data.describe(include='all')
```

```
           SalePrice       LotArea  ...  avgAreaIncome       zipCode
count    1198.000000   1198.000000  ...    1198.000000   1198.000000
mean   175202.219533  10543.478297  ...   85413.114357  50034.983306
std     69713.636280  10681.016803  ...   22777.553312     12.077225
```

```
min     39300.000000    1300.000000  ...   51360.000000  50010.000000
25%    130000.000000    7544.500000  ...   64351.000000  50026.000000
50%    160000.000000    9468.500000  ...   77061.000000  50037.000000
75%    205000.000000   11451.500000  ...   97203.000000  50045.000000
max    755000.000000  215245.000000  ...  150805.000000  50051.000000

[8 rows x 11 columns]
```

Dataset Description

The Sales Price dataset contains real estate data with multiple variables describing various aspects of each property. Key variables include:

Target Variable: - `SalePrice`: The sale price of the house (our prediction target)

Key Variables: - `LotArea`: Lot size in square feet - `YearBuilt`: Year the house was originally built - `GrLivArea`: Above ground living area square feet - `FullBath`: Number of full bathrooms - `HalfBath`: Number of half bathrooms - `BedroomAbvGr`: Number of bedrooms above grade - `TotRmsAbvGrd`: Total rooms above grade - `GarageCars`: Size of garage in car capacity - `zipCode`: Zip code of the property

Decision Tree Model Building

Now we'll build a decision tree model to predict house sale prices. We'll treat all predictor variables as numerical (including zipCode) to demonstrate how this affects feature importance interpretation.

Data Preparation and Model Training

R

```r
# Select key variables for the model
# We'll treat zipCode as numerical (which is problematic for interpretation)
model_data <- sales_data %>%
  select(SalePrice, LotArea, YearBuilt, GrLivArea, FullBath, HalfBath,
         BedroomAbvGr, TotRmsAbvGrd, GarageCars, zipCode) %>%
  na.omit()  # Remove any missing values

# Check for missing values
cat("Missing values check:\n")
```

Missing values check:

```
sapply(model_data, function(x) sum(is.na(x)))
```

```
   SalePrice      LotArea    YearBuilt    GrLivArea     FullBath     HalfBath 
           0            0            0            0            0            0 
BedroomAbvGr TotRmsAbvGrd   GarageCars      zipCode 
           0            0            0            0 
```

```
# Split the data into training and testing sets (80/20 split)
set.seed(123)  # For reproducibility
train_indices <- sample(1:nrow(model_data), 0.8 * nrow(model_data))
train_data <- model_data[train_indices, ]
test_data <- model_data[-train_indices, ]

cat("\nTraining set size:", nrow(train_data), "\n")
```

```
Training set size: 958
```

```
cat("Testing set size:", nrow(test_data), "\n")
```

```
Testing set size: 240
```

```
# Build decision tree with maximum depth of 3
# Using rpart for regression tree
tree_model <- rpart(SalePrice ~ .,
                    data = train_data,
                    method = "anova",
                    control = rpart.control(maxdepth = 3,
                                            minsplit = 20,
                                            minbucket = 10))

# Display model summary
cat("Decision Tree Model Summary:\n")
```

```
Decision Tree Model Summary:
```

```r
print(tree_model)
```

```
n= 958

node), split, n, deviance, yval
      * denotes terminal node

 1) root 958 4.282049e+12 173831.20
   2) GrLivArea< 1531 542 6.788498e+11 139231.30
     4) zipCode< 50038.5 350 2.525015e+11 123975.50
       8) GrLivArea< 892.5 54 2.913175e+10  96154.31 *
       9) GrLivArea>=892.5 296 1.739476e+11 129051.00 *
     5) zipCode>=50038.5 192 1.963976e+11 167041.30
      10) GrLivArea< 1106 63 2.011493e+10 135338.90 *
      11) GrLivArea>=1106 129 8.204240e+10 182523.90 *
   3) GrLivArea>=1531 416 2.108954e+12 218910.90
     6) GarageCars< 2.5 341 8.864405e+11 201605.30
      12) zipCode< 50043.5 219 5.370196e+11 186571.60 *
      13) zipCode>=50043.5 122 2.110743e+11 228592.00 *
     7) GarageCars>=2.5 75 6.560621e+11 297594.00
      14) GrLivArea< 2383 51 1.879430e+11 265277.80 *
      15) GrLivArea>=2383 24 3.016782e+11 366265.90 *
```

```r
# Model complexity parameters
cat("\nModel complexity parameters:\n")
```

```

Model complexity parameters:
```

```r
cat("Number of splits:", length(unique(tree_model$where)), "\n")
```

```
Number of splits: 8
```

```r
cat("Number of terminal nodes:", sum(tree_model$frame$var == "<leaf>"), "\n")
```

```
Number of terminal nodes: 8
```

Python

```python
# Select key variables for the model
# We'll treat zipCode as numerical (which is problematic for interpretation)
model_vars = ['SalePrice', 'LotArea', 'YearBuilt', 'GrLivArea', 'FullBath',
              'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd', 'GarageCars', 'zipCode']

model_data = sales_data[model_vars].dropna()

# Check for missing values
print("Missing values check:")
```

```
Missing values check:
```

```python
print(model_data.isnull().sum())
```

```
SalePrice       0
LotArea         0
YearBuilt       0
GrLivArea       0
FullBath        0
HalfBath        0
BedroomAbvGr    0
TotRmsAbvGrd    0
GarageCars      0
zipCode         0
dtype: int64
```

```python
# Split the data into training and testing sets (80/20 split)
from sklearn.model_selection import train_test_split

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)

print(f"\nTraining set size: {X_train.shape[0]}")
```

```

Training set size: 958
```

```python
print(f"Testing set size: {X_test.shape[0]}")
```

```
Testing set size: 240
```

```python
print(f"Number of features: {X_train.shape[1]}")
```

```
Number of features: 9
```

```python
# Build decision tree with maximum depth of 3
tree_model = DecisionTreeRegressor(max_depth=3,
                                  min_samples_split=20,
                                  min_samples_leaf=10,
                                  random_state=123)

# Fit the model
tree_model.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=3, min_samples_leaf=10, min_samples_split=20,
                      random_state=123)
```

```python
# Display model summary
print("Decision Tree Model Summary:")
```

```
Decision Tree Model Summary:
```

```python
print(f"Number of features: {tree_model.n_features_in_}")
```

```
Number of features: 9
```

```python
print(f"Number of leaves: {tree_model.get_n_leaves()}")
```

```
Number of leaves: 8
```

```python
print(f"Tree depth: {tree_model.get_depth()}")
```

```
Tree depth: 3
```

```
print(f"Number of nodes: {tree_model.tree_.node_count}")
```

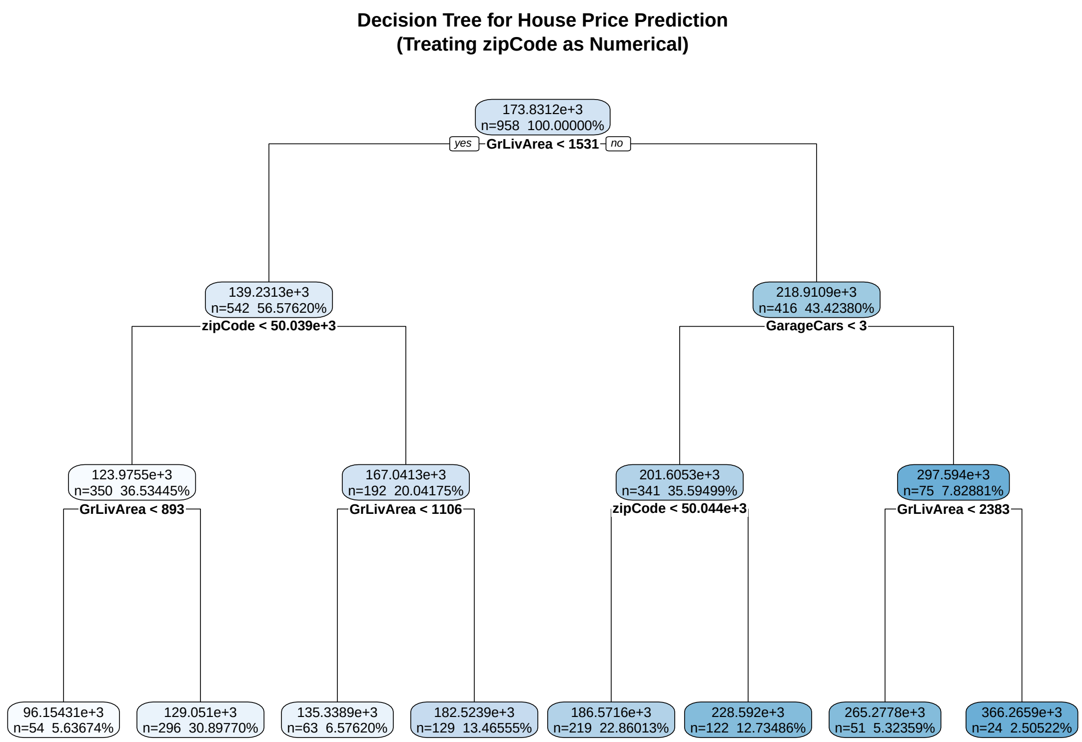
```
Number of nodes: 15
```

Tree Visualization

Let's visualize our decision tree to understand how it makes predictions and which variables it considers most important.

R

```
# Create a more detailed tree plot
par(mfrow = c(1, 1))

# Try rpart.plot first, fallback to plot if not available
if (require(rpart.plot, quietly = TRUE)) {
  rpart.plot(tree_model,
             type = 2,  # Show split labels
             extra = 101,  # Show number of observations and percentage
             fallen.leaves = TRUE,  # Position leaf nodes at bottom
             digits = 0,  # Round numbers
             cex = 0.8,  # Text size
             main = "Decision Tree for House Price Prediction\n(Treating zipCode as Numerical
} else {
  # Fallback to basic plot
  plot(tree_model, uniform = TRUE, main = "Decision Tree for House Price Prediction\n(Treatin
  text(tree_model, use.n = TRUE, all = TRUE, cex = 0.8)
}
```




```r
# Print the tree structure in text format
cat("Tree Structure (Text Format):\n")
```

```
Tree Structure (Text Format):
```

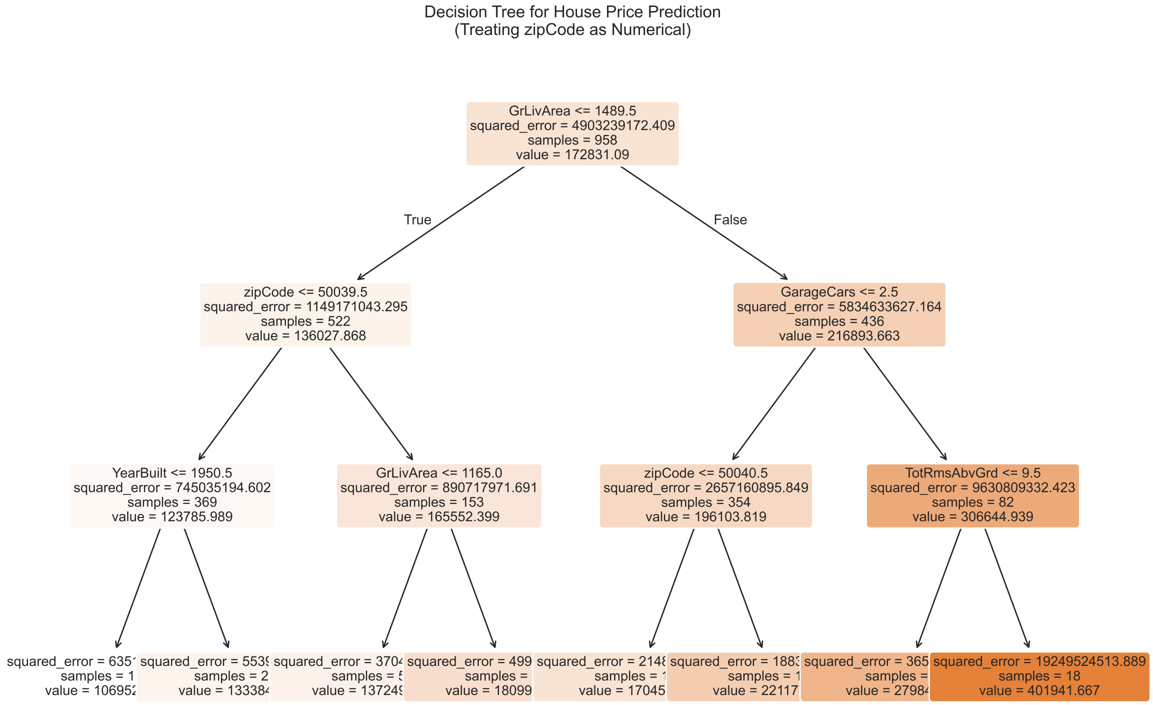
```r
print(tree_model)
```

```
n= 958

node), split, n, deviance, yval
      * denotes terminal node

 1) root 958 4.282049e+12 173831.20
   2) GrLivArea< 1531 542 6.788498e+11 139231.30
     4) zipCode< 50038.5 350 2.525015e+11 123975.50
       8) GrLivArea< 892.5 54 2.913175e+10  96154.31 *
       9) GrLivArea>=892.5 296 1.739476e+11 129051.00 *
     5) zipCode>=50038.5 192 1.963976e+11 167041.30
```

```
    10) GrLivArea< 1106 63 2.011493e+10 135338.90 *
    11) GrLivArea>=1106 129 8.204240e+10 182523.90 *
 3) GrLivArea>=1531 416 2.108954e+12 218910.90
   6) GarageCars< 2.5 341 8.864405e+11 201605.30
    12) zipCode< 50043.5 219 5.370196e+11 186571.60 *
    13) zipCode>=50043.5 122 2.110743e+11 228592.00 *
   7) GarageCars>=2.5 75 6.560621e+11 297594.00
    14) GrLivArea< 2383 51 1.879430e+11 265277.80 *
    15) GrLivArea>=2383 24 3.016782e+11 366265.90 *
```

Python

```python
# Create tree visualization
plt.figure(figsize=(12, 8))
plot_tree(tree_model,
          feature_names=X_train.columns,
          filled=True,
          rounded=True,
          fontsize=10,
          max_depth=3)
plt.title("Decision Tree for House Price Prediction\n(Treating zipCode as Numerical)")
plt.tight_layout()
plt.show()
```




```
# Print tree structure in text format
from sklearn.tree import export_text

tree_rules = export_text(tree_model, feature_names=list(X_train.columns))
print("Tree Structure (Text Format):")
```

```
Tree Structure (Text Format):
```

```
print(tree_rules)
```

```
|--- GrLivArea <= 1489.50
|   |--- zipCode <= 50039.50
|   |   |--- YearBuilt <= 1950.50
|   |   |   |--- value: [106952.64]
|   |   |--- YearBuilt >  1950.50
|   |   |   |--- value: [133384.58]
|   |--- zipCode >  50039.50
|   |   |--- GrLivArea <= 1165.00
```

```
|   |   |   |--- value: [137249.07]
|   |   |--- GrLivArea >  1165.00
|   |   |   |--- value: [180990.58]
|--- GrLivArea >  1489.50
|   |--- GarageCars <= 2.50
|   |   |--- zipCode <= 50040.50
|   |   |   |--- value: [170457.09]
|   |   |--- zipCode >  50040.50
|   |   |   |--- value: [221177.44]
|   |--- GarageCars >  2.50
|   |   |--- TotRmsAbvGrd <= 9.50
|   |   |   |--- value: [279842.73]
|   |   |--- TotRmsAbvGrd >  9.50
|   |   |   |--- value: [401941.67]
```

Model Performance Evaluation

Let's evaluate how well our decision tree performs on both training and testing data.

R

```r
# Make predictions
train_pred <- predict(tree_model, train_data)
test_pred <- predict(tree_model, test_data)

# Calculate performance metrics
train_rmse <- sqrt(mean((train_data$SalePrice - train_pred)^2))
test_rmse <- sqrt(mean((test_data$SalePrice - test_pred)^2))

train_r2 <- 1 - sum((train_data$SalePrice - train_pred)^2) /
            sum((train_data$SalePrice - mean(train_data$SalePrice))^2)
test_r2 <- 1 - sum((test_data$SalePrice - test_pred)^2) /
           sum((test_data$SalePrice - mean(test_data$SalePrice))^2)

# Display performance metrics
cat("Model Performance Metrics:\n")
```

```
Model Performance Metrics:
```

```r
cat("Training RMSE: $", round(train_rmse, 2), "\n")
```

```
Training RMSE: $ 40132.24
```

```r
cat("Testing RMSE: $", round(test_rmse, 2), "\n")
```

```
Testing RMSE: $ 45940.5
```

```r
cat("Training R²: ", round(train_r2, 4), "\n")
```

```
Training R²:  0.6397
```

```r
cat("Testing R²: ", round(test_r2, 4), "\n")
```

```
Testing R²:  0.6681
```

```r
# Calculate mean absolute error
train_mae <- mean(abs(train_data$SalePrice - train_pred))
test_mae <- mean(abs(test_data$SalePrice - test_pred))

cat("Training MAE: $", round(train_mae, 2), "\n")
```

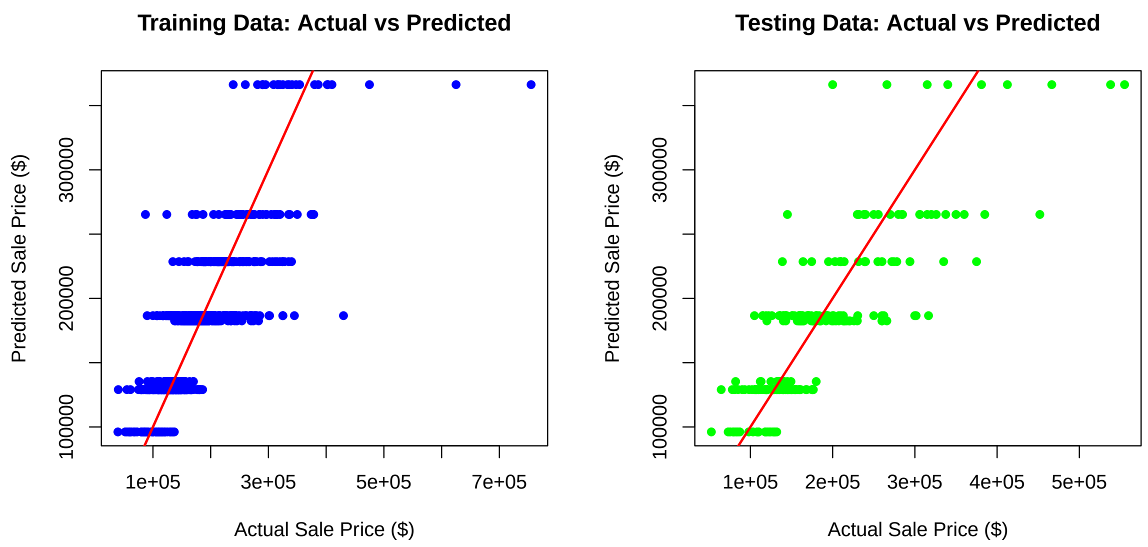
```
Training MAE: $ 27435.82
```

```r
cat("Testing MAE: $", round(test_mae, 2), "\n")
```

```
Testing MAE: $ 32388.83
```

```r
# Create prediction vs actual plots
par(mfrow = c(1, 2))

# Training data
plot(train_data$SalePrice, train_pred,
     main = "Training Data: Actual vs Predicted",
     xlab = "Actual Sale Price ($)",
     ylab = "Predicted Sale Price ($)",
     pch = 16, col = "blue", alpha = 0.6)
```

```r
abline(0, 1, col = "red", lwd = 2)

# Testing data
plot(test_data$SalePrice, test_pred,
     main = "Testing Data: Actual vs Predicted",
     xlab = "Actual Sale Price ($)",
     ylab = "Predicted Sale Price ($)",
     pch = 16, col = "green", alpha = 0.6)
abline(0, 1, col = "red", lwd = 2)
```




```r
par(mfrow = c(1, 1))
```

Python

```python
# Make predictions
train_pred = tree_model.predict(X_train)
test_pred = tree_model.predict(X_test)

# Calculate performance metrics
train_rmse = np.sqrt(mean_squared_error(y_train, train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, test_pred))

train_r2 = r2_score(y_train, train_pred)
```

```python
test_r2 = r2_score(y_test, test_pred)

# Display performance metrics
print("Model Performance Metrics:")
```

```
Model Performance Metrics:
```

```python
print(f"Training RMSE: ${train_rmse:,.2f}")
```

```
Training RMSE: $40,585.49
```

```python
print(f"Testing RMSE: ${test_rmse:,.2f}")
```

```
Testing RMSE: $44,463.43
```

```python
print(f"Training R²: {train_r2:.4f}")
```

```
Training R²: 0.6641
```

```python
print(f"Testing R²: {test_r2:.4f}")
```

```
Testing R²: 0.5660
```

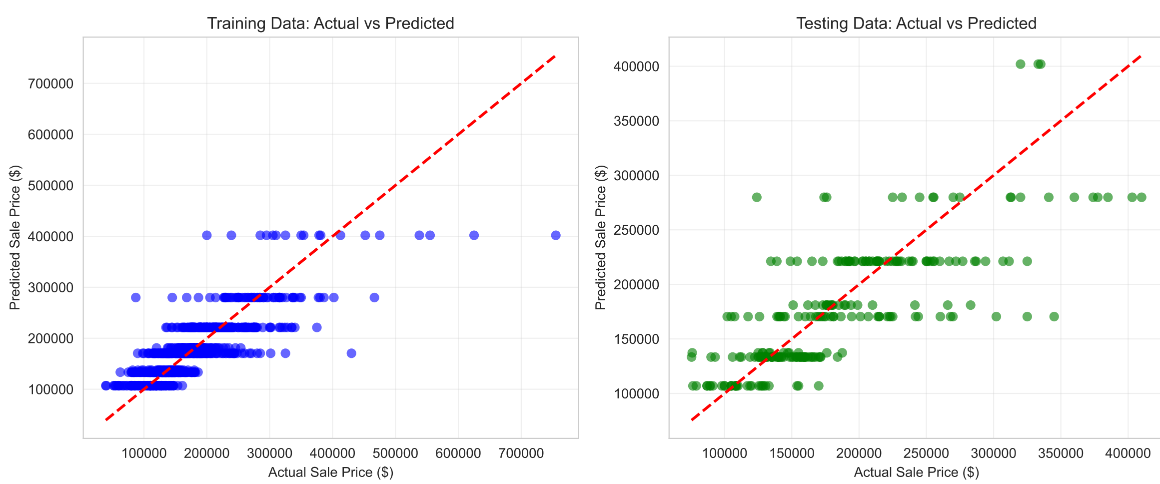
```python
# Calculate mean absolute error
from sklearn.metrics import mean_absolute_error

train_mae = mean_absolute_error(y_train, train_pred)
test_mae = mean_absolute_error(y_test, test_pred)

print(f"Training MAE: ${train_mae:,.2f}")
```

```
Training MAE: $27,558.19
```

```python
print(f"Testing MAE: ${test_mae:,.2f}")
```

```
Testing MAE: $31,272.33
```

```python
# Create prediction vs actual plots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Training data
ax1.scatter(y_train, train_pred, alpha=0.6, color='blue')
ax1.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--', lw=2)
ax1.set_xlabel('Actual Sale Price ($)')
ax1.set_ylabel('Predicted Sale Price ($)')
ax1.set_title('Training Data: Actual vs Predicted')
ax1.grid(True, alpha=0.3)

# Testing data
ax2.scatter(y_test, test_pred, alpha=0.6, color='green')
ax2.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
ax2.set_xlabel('Actual Sale Price ($)')
ax2.set_ylabel('Predicted Sale Price ($)')
ax2.set_title('Testing Data: Actual vs Predicted')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



Feature Importance Analysis

Now let's examine which features the decision tree considers most important. This is where we'll see the impact of treating zipCode as a numerical variable.

R

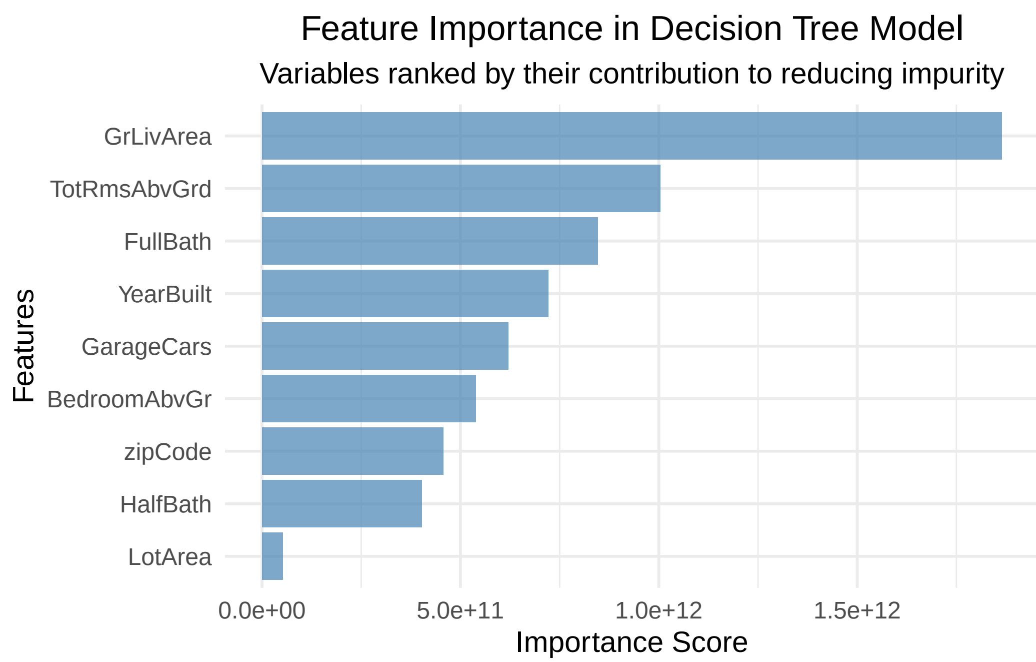
```r
# Extract feature importance
importance_df <- data.frame(
  Feature = names(tree_model$variable.importance),
  Importance = as.numeric(tree_model$variable.importance)
) %>%
  arrange(desc(Importance)) %>%
  mutate(Importance_Percent = round(Importance / sum(Importance) * 100, 2))

# Display feature importance
cat("Feature Importance Rankings:\n")
```

```
Feature Importance Rankings:
```

```r
print(importance_df)
```

```
       Feature   Importance Importance_Percent
1    GrLivArea 1.864741e+12              28.65
2 TotRmsAbvGrd 1.003386e+12              15.42
3     FullBath 8.468023e+11              13.01
4    YearBuilt 7.219815e+11              11.09
5   GarageCars 6.209281e+11               9.54
6 BedroomAbvGr 5.387957e+11               8.28
7      zipCode 4.569801e+11               7.02
8     HalfBath 4.022969e+11               6.18
9      LotArea 5.178462e+10               0.80
```

```r
# Create a bar plot of feature importance
library(ggplot2)

ggplot(importance_df, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_col(fill = "steelblue", alpha = 0.7) +
  coord_flip() +
  labs(title = "Feature Importance in Decision Tree Model",
       subtitle = "Variables ranked by their contribution to reducing impurity",
       x = "Features",
       y = "Importance Score") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))
```




```r
# Analyze the top features
cat("Top 3 Most Important Features:\n")
```

```
Top 3 Most Important Features:
```

```r
top_features <- head(importance_df, 3)
for(i in 1:nrow(top_features)) {
  cat(i, ". ", top_features$Feature[i],
      " (", top_features$Importance_Percent[i], "% of total importance)\n", sep = "")
}
```

```
1. GrLivArea (28.65% of total importance)
2. TotRmsAbvGrd (15.42% of total importance)
3. FullBath (13.01% of total importance)
```

```r
cat("\nKey Observations:\n")
```

```

Key Observations:
```

```r
cat("- zipCode appears as feature #", which(importance_df$Feature == "zipCode"),
    " with ", importance_df$Importance_Percent[importance_df$Feature == "zipCode"],
    "% importance\n", sep = "")
```

```
- zipCode appears as feature #7 with 7.02% importance
```

```r
cat("- This is problematic because zipCode is being treated as a numerical variable\n")
```

```
- This is problematic because zipCode is being treated as a numerical variable
```

```r
cat("- The tree might split on 'zipCode > 50012.5' which has no meaningful interpretation\n")
```

```
- The tree might split on 'zipCode > 50012.5' which has no meaningful interpretation
```

Python

```python
# Extract feature importance
importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': tree_model.feature_importances_
}).sort_values('Importance', ascending=False)

importance_df['Importance_Percent'] = (importance_df['Importance'] * 100).round(2)

# Display feature importance
print("Feature Importance Rankings:")
```

```
Feature Importance Rankings:
```

```python
print(importance_df)
```

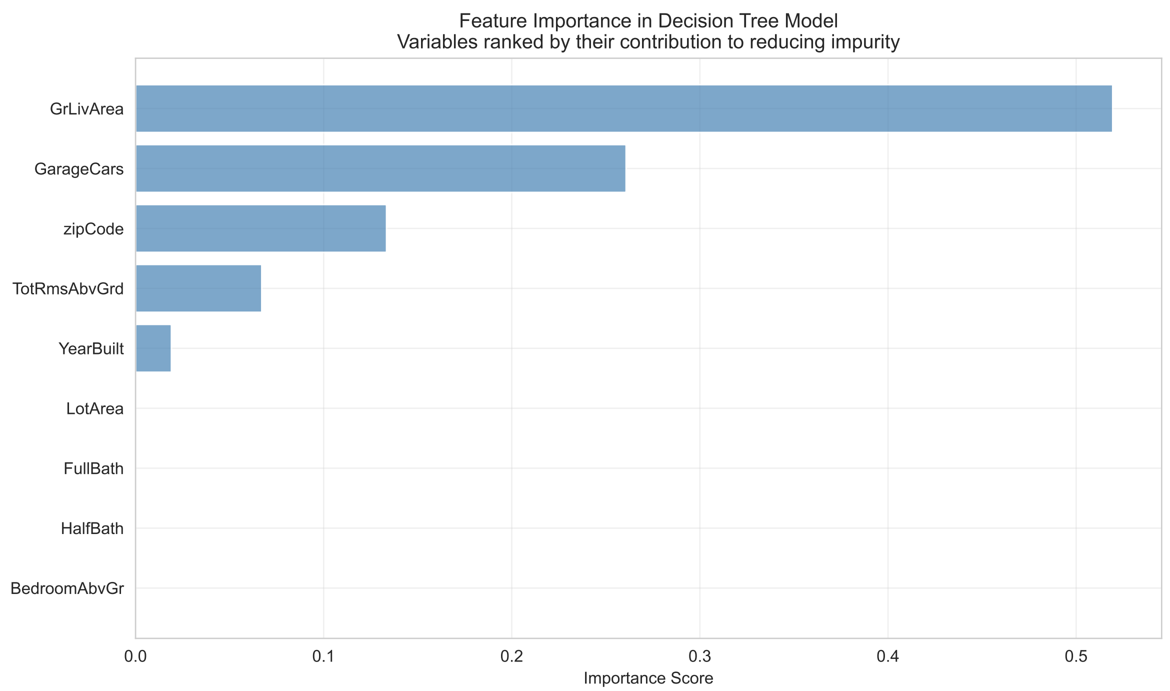
```
        Feature  Importance  Importance_Percent
2     GrLivArea    0.519472               51.95
7    GarageCars    0.260808               26.08
8       zipCode    0.133463               13.35
6  TotRmsAbvGrd    0.067144                6.71
1     YearBuilt    0.019114                1.91
```

```
0       LotArea    0.000000                0.00
3      FullBath    0.000000                0.00
4      HalfBath    0.000000                0.00
5   BedroomAbvGr    0.000000                0.00
```

```python
# Create a bar plot of feature importance
plt.figure(figsize=(10, 6))
plt.barh(range(len(importance_df)), importance_df['Importance'],
         color='steelblue', alpha=0.7)
plt.yticks(range(len(importance_df)), importance_df['Feature'])
```

```
([<matplotlib.axis.YTick object at 0x000002BD63F1FC50>, <matplotlib.axis.YTick object at 0x00
```

```python
plt.xlabel('Importance Score')
plt.title('Feature Importance in Decision Tree Model\nVariables ranked by their contribution
plt.gca().invert_yaxis()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



```
# Analyze the top features
print("Top 3 Most Important Features:")
```

```
Top 3 Most Important Features:
```

```
top_features = importance_df.head(3)
for i, (_, row) in enumerate(top_features.iterrows(), 1):
    print(f"{i}. {row['Feature']} ({row['Importance_Percent']}% of total importance)")
```

```
1. GrLivArea (51.95% of total importance)
2. GarageCars (26.08% of total importance)
3. zipCode (13.35% of total importance)
```

```
zipcode_rank = importance_df[importance_df['Feature'] == 'zipCode'].index[0] + 1
zipcode_importance = importance_df[importance_df['Feature'] == 'zipCode']['Importance_Percen

print(f"\nKey Observations:")
```

```
Key Observations:
```

```
print(f"- zipCode appears as feature #{zipcode_rank} with {zipcode_importance}% importance")
```

```
- zipCode appears as feature #9 with 13.35% importance
```

```
print("- This is problematic because zipCode is being treated as a numerical variable")
```

```
- This is problematic because zipCode is being treated as a numerical variable
```

```
print("- The tree might split on 'zipCode > 50012.5' which has no meaningful interpretation"
```

```
- The tree might split on 'zipCode > 50012.5' which has no meaningful interpretation
```

Critical Analysis: The Problem with Numerical Encoding

> ⚠ The Encoding Problem Revealed
>
> **What we just observed:** Our decision tree treated `zipCode` as a numerical variable, allowing it to make splits like "zipCode > 50012.5". This creates several problems:
>
> 1. **Meaningless Splits:** A zip code of 50013 is not "greater than" 50012 in any meaningful way for house prices
> 2. **False Importance:** The algorithm might assign high importance to zipCode simply because it can create many splits
> 3. **Misleading Interpretations:** We might conclude that zipCode is very important when it's really just an artifact of poor encoding
>
> **The Real Issue:** Zip codes are categorical variables that represent discrete geographic areas. The numerical values (50010, 50011, 50012, etc.) have no inherent order or magnitude relationship to house prices.

Next Steps: In the following sections, we'll demonstrate how proper categorical encoding changes both the tree structure and feature importance rankings, revealing the true importance of each variable for predicting house prices.