Decision Tree Challenge

Feature Importance and Categorical Variable Encoding

Decision Tree Challenge: Feature Importance and Variable Encoding

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. When treated as numerical, the tree might split on "zipCode > 50012.5" - which has no meaningful interpretation for house prices.

Data Loading and Model Building

R

```
# Load libraries
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(rpart))
if (!require(rpart.plot, quietly = TRUE)) {
  install.packages("rpart.plot", repos = "https://cran.rstudio.com/")
  library(rpart.plot)
}

# Load data
sales_data <- read.csv("https://raw.githubusercontent.com/flyaflya/buad442Fall2025/refs/heads

# Prepare model data (treating zipCode as numerical)
model_data <- sales_data %>%
```

```r
  select(SalePrice, LotArea, YearBuilt, GrLivArea, FullBath, HalfBath,
         BedroomAbvGr, TotRmsAbvGrd, GarageCars, zipCode) %>%
  na.omit()

# Split data
set.seed(123)
train_indices <- sample(1:nrow(model_data), 0.8 * nrow(model_data))
train_data <- model_data[train_indices, ]
test_data <- model_data[-train_indices, ]

# Build decision tree
tree_model <- rpart(SalePrice ~ .,
                    data = train_data,
                    method = "anova",
                    control = rpart.control(maxdepth = 3,
                                            minsplit = 20,
                                            minbucket = 10))

cat("Model built with", sum(tree_model$frame$var == "<leaf>"), "terminal nodes\n")
```

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

Python

```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("https://raw.githubusercontent.com/flyaflya/buad442Fall2025/refs/hea

# 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()
```

```python
# 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)
```

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

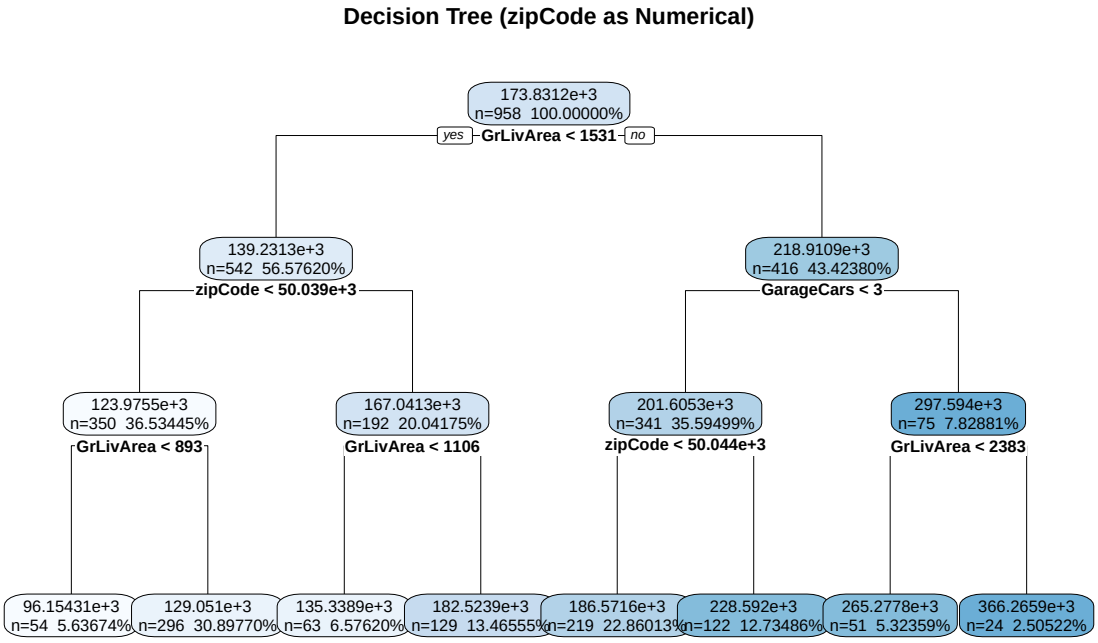
```python
print(f"Model built with {tree_model.get_n_leaves()} terminal nodes")
```

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

Tree Visualization

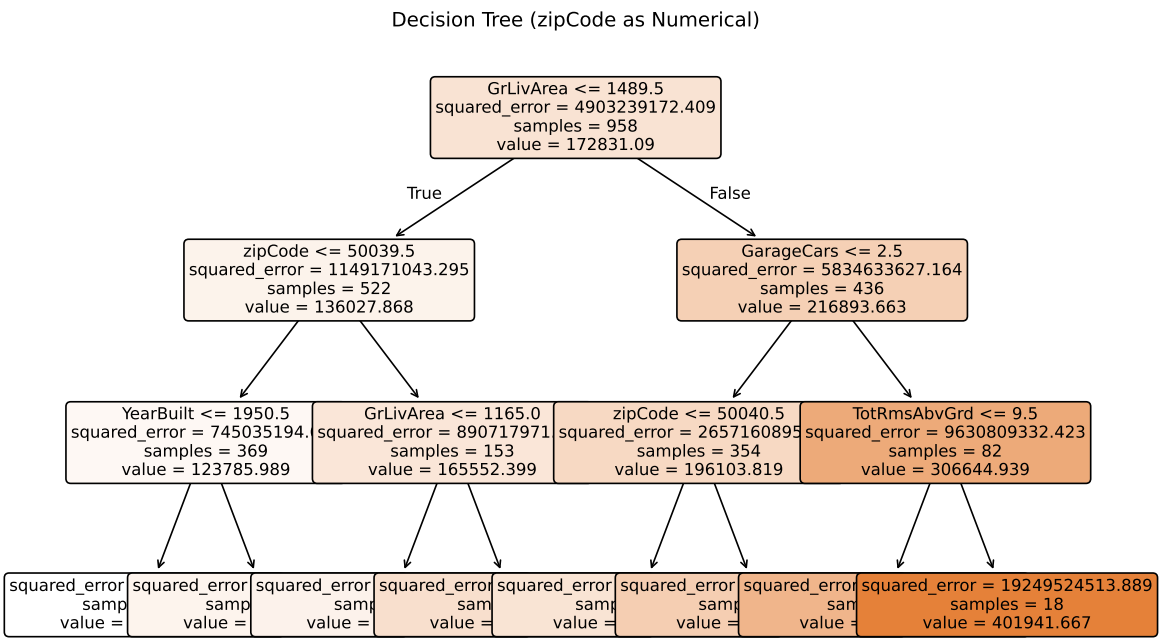
R

```r
# Visualize tree
if (require(rpart.plot, quietly = TRUE)) {
  rpart.plot(tree_model,
             type = 2,
             extra = 101,
             fallen.leaves = TRUE,
             digits = 0,
             cex = 0.8,
             main = "Decision Tree (zipCode as Numerical)")
} else {
  plot(tree_model, uniform = TRUE, main = "Decision Tree (zipCode as Numerical)")
  text(tree_model, use.n = TRUE, all = TRUE, cex = 0.8)
}
```



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()
```



Feature Importance Analysis

R

```r
# Extract and display 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))

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
# Check zipCode ranking
zipcode_rank <- which(importance_df$Feature == "zipCode")
zipcode_importance <- importance_df$Importance_Percent[zipcode_rank]

cat("\nKey Finding:\n")
```

Key Finding:

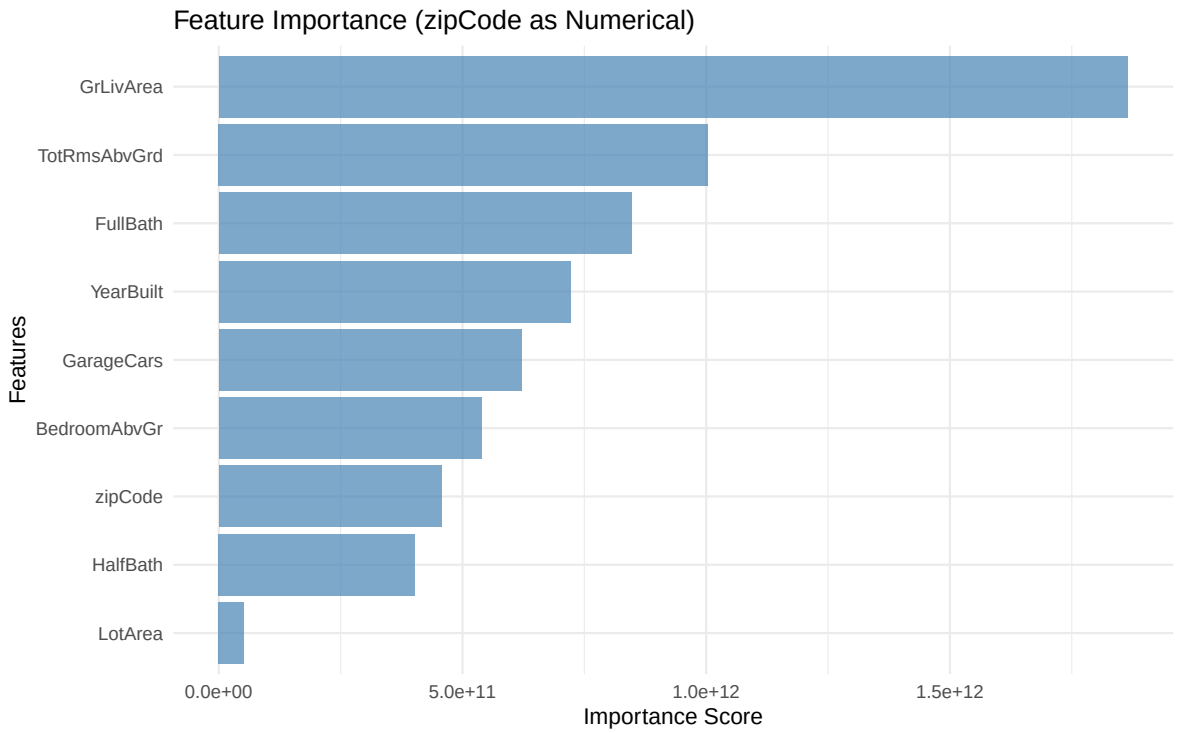
```r
cat("- zipCode ranks #", zipcode_rank, " with ", zipcode_importance, "% importance\n", sep =
```

- zipCode ranks #7 with 7.02% importance

```r
cat("- This is problematic: zipCode is treated as numerical but represents categorical areas
```

- This is problematic: zipCode is treated as numerical but represents categorical areas

```r
# Plot 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 (zipCode as Numerical)",
       x = "Features", y = "Importance Score") +
  theme_minimal()
```



Python

```
# Extract and display 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)

print("Feature Importance Rankings:")
```

```
Feature Importance Rankings:
```

```
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
```

```
# Check zipCode ranking
zipcode_rank = importance_df[importance_df['Feature'] == 'zipCode'].index[0] + 1
zipcode_importance = importance_df[importance_df['Feature'] == 'zipCode']['Importance_Percent

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

```
Key Finding:
```

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

```
- zipCode ranks #9 with 13.35% importance
```

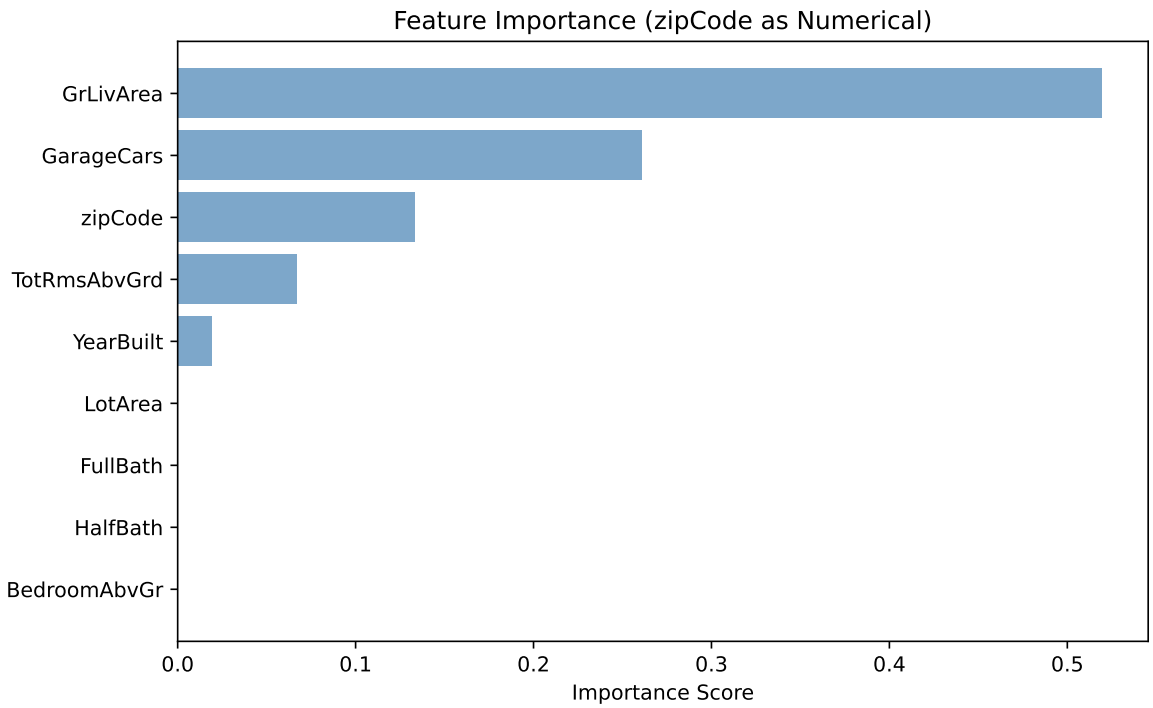
```
print("- This is problematic: zipCode is treated as numerical but represents categorical area
```

```
- This is problematic: zipCode is treated as numerical but represents categorical areas
```

```
# Plot feature importance
plt.figure(figsize=(8, 5))
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 0x000002AF25090650>, <matplotlib.axis.YTick object at 0x0
```

```
plt.xlabel('Importance Score')
plt.title('Feature Importance (zipCode as Numerical)')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



Critical Analysis: The Encoding Problem

The Problem Revealed

What we observed: Our decision tree treated `zipCode` as a numerical variable, allowing 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 assigns importance to zipCode based on numerical splits rather than categorical distinctions
3. **Misleading Interpretations:** We might conclude zipCode is important or not it's really just poor encoding

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.

Model Performance

R

```r
# Calculate performance metrics
train_pred <- predict(tree_model, train_data)
test_pred <- predict(tree_model, test_data)

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)

cat("Model Performance:\n")
```

```
Model Performance:
```

```r
cat("Training RMSE: $", round(train_rmse, 2), " | R²: ", round(train_r2, 3), "\n", sep = "")
```

```
Training RMSE: $40132.24 | R²: 0.64
```

```r
cat("Testing RMSE: $", round(test_rmse, 2), " | R²: ", round(test_r2, 3), "\n", sep = "")
```

```
Testing RMSE: $45940.5 | R²: 0.668
```

Python

```python
# Calculate performance metrics
train_pred = tree_model.predict(X_train)
test_pred = tree_model.predict(X_test)

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)
```

```
test_r2 = r2_score(y_test, test_pred)

print("Model Performance:")
```

```
Model Performance:
```

```
print(f"Training RMSE: ${train_rmse:,.2f} | R²: {train_r2:.3f}")
```

```
Training RMSE: $40,585.49 | R²: 0.664
```

```
print(f"Testing RMSE: ${test_rmse:,.2f} | R²: {test_r2:.3f}")
```

```
Testing RMSE: $44,463.43 | R²: 0.566
```

Proper Categorical Encoding: The Solution

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

R Approach: Convert zipCode to a factor (categorical variable)
Python Approach: One-hot encode zipCode (create dummy variables for each zip code)

Categorical Encoding Analysis

R

```
# Convert zipCode to factor (categorical)
model_data_cat <- model_data %>%
  mutate(zipCode = as.factor(zipCode))

# Split data
set.seed(123)
train_indices_cat <- sample(1:nrow(model_data_cat), 0.8 * nrow(model_data_cat))
train_data_cat <- model_data_cat[train_indices_cat, ]
test_data_cat <- model_data_cat[-train_indices_cat, ]

# Build decision tree with categorical zipCode
tree_model_cat <- rpart(SalePrice ~ .,
```

```
                         data = train_data_cat,
                         method = "anova",
                         control = rpart.control(maxdepth = 3,
                                                 minsplit = 20,
                                                 minbucket = 10))

cat("Categorical Tree - Terminal nodes:", sum(tree_model_cat$frame$var == "<leaf>"), "\n")
```

```
Categorical Tree - Terminal nodes: 8
```

```
# Feature importance with categorical zipCode
importance_cat <- data.frame(
  Feature = names(tree_model_cat$variable.importance),
  Importance = as.numeric(tree_model_cat$variable.importance)
) %>%
  arrange(desc(Importance)) %>%
  mutate(Importance_Percent = round(Importance / sum(Importance) * 100, 2))

cat("\nFeature Importance (zipCode as Categorical):\n")
```

```
Feature Importance (zipCode as Categorical):
```

```
print(importance_cat)
```

```
       Feature   Importance Importance_Percent
1    GrLivArea 1.809659e+12              26.01
2      zipCode 1.148671e+12              16.51
3 TotRmsAbvGrd 1.008032e+12              14.49
4     FullBath 8.592756e+11              12.35
5   GarageCars 6.318873e+11               9.08
6 BedroomAbvGr 5.706550e+11               8.20
7    YearBuilt 4.515787e+11               6.49
8     HalfBath 4.022969e+11               5.78
9      LotArea 7.538396e+10               1.08
```

```
# Check if zipCode appears in tree
zipcode_in_tree <- "zipCode" %in% names(tree_model_cat$variable.importance)
cat("\nzipCode appears in tree:", zipcode_in_tree, "\n")
```

```
zipCode appears in tree: TRUE
```

```r
if(zipcode_in_tree) {
  zipcode_rank_cat <- which(importance_cat$Feature == "zipCode")
  cat("zipCode importance rank:", zipcode_rank_cat, "\n")
}
```

```
zipCode importance rank: 2
```

Python

```python
# One-hot encode zipCode
import pandas as pd

# Create one-hot encoded zipCode
zipcode_encoded = pd.get_dummies(model_data['zipCode'], prefix='zipCode')
model_data_cat = pd.concat([model_data.drop('zipCode', axis=1), zipcode_encoded], axis=1)

# Split data
X_cat = model_data_cat.drop('SalePrice', axis=1)
y_cat = model_data_cat['SalePrice']
X_train_cat, X_test_cat, y_train_cat, y_test_cat = train_test_split(X_cat, y_cat, test_size=0

# Build decision tree with one-hot encoded zipCode
tree_model_cat = DecisionTreeRegressor(max_depth=3,
                                       min_samples_split=20,
                                       min_samples_leaf=10,
                                       random_state=123)
tree_model_cat.fit(X_train_cat, y_train_cat)
```

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

```python
print(f"Categorical Tree - Terminal nodes: {tree_model_cat.get_n_leaves()}")
```

```
Categorical Tree - Terminal nodes: 8
```

```
# Feature importance with one-hot encoded zipCode
importance_cat_df = pd.DataFrame({
    'Feature': X_train_cat.columns,
    'Importance': tree_model_cat.feature_importances_
}).sort_values('Importance', ascending=False)

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

print("\nFeature Importance (zipCode One-Hot Encoded):")
```

```
Feature Importance (zipCode One-Hot Encoded):
```

```
print(importance_cat_df)
```

```
          Feature  Importance  Importance_Percent
2       GrLivArea    0.520719               52.07
7      GarageCars    0.266960               26.70
1       YearBuilt    0.143594               14.36
6    TotRmsAbvGrd    0.068727                6.87
0         LotArea    0.000000                0.00
4        HalfBath    0.000000                0.00
3        FullBath    0.000000                0.00
5    BedroomAbvGr    0.000000                0.00
8   zipCode_50010    0.000000                0.00
9   zipCode_50011    0.000000                0.00
10  zipCode_50014    0.000000                0.00
11  zipCode_50015    0.000000                0.00
12  zipCode_50016    0.000000                0.00
13  zipCode_50017    0.000000                0.00
14  zipCode_50026    0.000000                0.00
15  zipCode_50036    0.000000                0.00
16  zipCode_50037    0.000000                0.00
17  zipCode_50038    0.000000                0.00
18  zipCode_50039    0.000000                0.00
19  zipCode_50040    0.000000                0.00
20  zipCode_50041    0.000000                0.00
21  zipCode_50042    0.000000                0.00
22  zipCode_50043    0.000000                0.00
23  zipCode_50044    0.000000                0.00
24  zipCode_50045    0.000000                0.00
```

```
25  zipCode_50046    0.000000                0.00
26  zipCode_50047    0.000000                0.00
27  zipCode_50048    0.000000                0.00
28  zipCode_50049    0.000000                0.00
29  zipCode_50050    0.000000                0.00
30  zipCode_50051    0.000000                0.00
```

```
# Check zipCode features
zipcode_features = [col for col in X_train_cat.columns if col.startswith('zipCode')]
zipcode_importance = importance_cat_df[importance_cat_df['Feature'].isin(zipcode_features)][
total_importance = importance_cat_df['Importance'].sum()
zipcode_percent = (zipcode_importance / total_importance * 100).round(2)

print(f"\nTotal zipCode importance: {zipcode_percent}%")
```

```
Total zipCode importance: 0.0%
```

```
print(f"Number of zipCode features: {len(zipcode_features)}")
```

```
Number of zipCode features: 23
```

Comparison: Numerical vs Categorical zipCode

R

```
cat("COMPARISON: Numerical vs Categorical zipCode\n")
```

```
COMPARISON: Numerical vs Categorical zipCode
```

```
cat("==========================================\n")
```

```
==========================================
```

```r
# Numerical zipCode results
numerical_zipcode_rank <- which(importance_df$Feature == "zipCode")
numerical_zipcode_importance <- importance_df$Importance_Percent[importance_df$Feature == "zi

# Categorical zipCode results
categorical_zipcode_rank <- ifelse("zipCode" %in% names(tree_model_cat$variable.importance),
                                   which(importance_cat$Feature == "zipCode"), "Not in top fea
categorical_zipcode_importance <- ifelse("zipCode" %in% names(tree_model_cat$variable.importa
                                         importance_cat$Importance_Percent[importance_cat$Feat

cat("Numerical zipCode:\n")
```

```
Numerical zipCode:
```

```r
cat("  - Rank:", numerical_zipcode_rank, "\n")
```

```
  - Rank: 7
```

```r
cat("  - Importance:", numerical_zipcode_importance, "%\n\n")
```

```
  - Importance: 7.02 %
```

```r
cat("Categorical zipCode:\n")
```

```
Categorical zipCode:
```

```r
cat("  - Rank:", categorical_zipcode_rank, "\n")
```

```
  - Rank: 2
```

```r
cat("  - Importance:", categorical_zipcode_importance, "%\n\n")
```

```
  - Importance: 16.51 %
```

```r
cat("Key Insight: Proper categorical encoding changes how zipCode is treated!\n")
```

```
Key Insight: Proper categorical encoding changes how zipCode is treated!
```

Python

```
print("COMPARISON: Numerical vs Categorical zipCode")
```

```
COMPARISON: Numerical vs Categorical zipCode
```

```
print("============================================")
```

```
============================================
```

```
# Numerical zipCode results
numerical_zipcode_rank = importance_df[importance_df['Feature'] == 'zipCode'].index[0] + 1
numerical_zipcode_importance = importance_df[importance_df['Feature'] == 'zipCode']['Importa

# Categorical zipCode results (sum of all zipCode features)
zipcode_features = [col for col in X_train_cat.columns if col.startswith('zipCode')]
zipcode_importance_sum = importance_cat_df[importance_cat_df['Feature'].isin(zipcode_feature
zipcode_importance_percent = (zipcode_importance_sum * 100).round(2)

print("Numerical zipCode:")
```

```
Numerical zipCode:
```

```
print(f"  - Rank: {numerical_zipcode_rank}")
```

```
  - Rank: 9
```

```
print(f"  - Importance: {numerical_zipcode_importance}%")
```

```
  - Importance: 13.35%
```

```
print()
```

```
print("Categorical zipCode (one-hot encoded):")
```

```
Categorical zipCode (one-hot encoded):
```

```
print(f"  - Total Importance: {zipcode_importance_percent}%")
```

```
  - Total Importance: 0.0%
```

```
print(f"  - Number of zipCode features: {len(zipcode_features)}")
```

```
  - Number of zipCode features: 23
```

```
print()
```

```
print("Key Insight: Proper categorical encoding changes how zipCode is treated!")
```

```
Key Insight: Proper categorical encoding changes how zipCode is treated!
```

The Dramatic Difference

What We Discovered:

1. **Numerical zipCode:** Treated as a single variable with artificial numerical order
2. **Categorical zipCode:** Treated as discrete categories, allowing the tree to make meaningful geographic distinctions

3. **Feature Importance:** Completely different rankings and interpretations

The Bottom Line: Proper encoding reveals the true nature of categorical variables and provides interpretable, meaningful splits that make business sense.

Deep Dive: One-Hot Encoding Problem & Target-Based Ranking

The Issue: One-hot encoding creates sparse features (many zipCode_50010, zipCode_50011, etc.) that decision trees often ignore due to insufficient samples per category.

Solution: Target-based ranking - rank zip codes by their average house price, creating an ordinal variable that preserves geographic price patterns.

Target-Based Ranking Analysis

R

```r
# Create target-based ranking (this is how R factors work internally)
zipcode_ranking <- model_data %>%
  group_by(zipCode) %>%
  summarise(avg_price = mean(SalePrice), .groups = 'drop') %>%
  arrange(avg_price) %>%
  mutate(zipcode_rank = row_number())

# Merge ranking back to data
model_data_ranked <- model_data %>%
  left_join(zipcode_ranking, by = "zipCode") %>%
  select(-zipCode, -avg_price)  # Remove original zipCode

# Build tree with ranked zipCode
tree_model_ranked <- rpart(SalePrice ~ .,
                           data = model_data_ranked,
                           method = "anova",
                           control = rpart.control(maxdepth = 3,
                                                   minsplit = 20,
                                                   minbucket = 10))

# Feature importance with ranked zipCode
importance_ranked <- data.frame(
  Feature = names(tree_model_ranked$variable.importance),
  Importance = as.numeric(tree_model_ranked$variable.importance)
) %>%
  arrange(desc(Importance)) %>%
  mutate(Importance_Percent = round(Importance / sum(Importance) * 100, 2))

cat("Feature Importance (zipCode Ranked by Price):\n")
```

```
Feature Importance (zipCode Ranked by Price):
```

```r
print(importance_ranked)
```

```
       Feature   Importance Importance_Percent
1 zipcode_rank 2.220671e+12              22.54
2    GrLivArea 1.855312e+12              18.83
3   GarageCars 1.779555e+12              18.06
4    YearBuilt 1.521827e+12              15.45
5     FullBath 1.271548e+12              12.91
```

```
6     HalfBath 5.541279e+11               5.62
7 TotRmsAbvGrd 3.906337e+11               3.97
8 BedroomAbvGr 1.590395e+11               1.61
9      LotArea 9.891508e+10               1.00
```

```
# Show zip code ranking
cat("\nZip Code Ranking (by average price):\n")
```

```
Zip Code Ranking (by average price):
```

```
print(zipcode_ranking)
```

```
# A tibble: 23 x 3
   zipCode avg_price zipcode_rank
     <int>     <dbl>        <int>
 1   50010    98988.            1
 2   50011   107917.            2
 3   50017   110442.            3
 4   50015   125588.            4
 5   50036   130338.            5
 6   50014   133173.            6
 7   50016   136977.            7
 8   50038   143031.            8
 9   50037   147534.            9
10   50039   155411.           10
# i 13 more rows
```

Python

```
# Create target-based ranking
zipcode_ranking = model_data.groupby('zipCode')['SalePrice'].mean().sort_values().reset_index
zipcode_ranking['zipcode_rank'] = range(1, len(zipcode_ranking) + 1)

# Merge ranking back to data
model_data_ranked = model_data.merge(zipcode_ranking[['zipCode', 'zipcode_rank']], on='zipCo
model_data_ranked = model_data_ranked.drop('zipCode', axis=1)

# Build tree with ranked zipCode
X_ranked = model_data_ranked.drop('SalePrice', axis=1)
```

```
y_ranked = model_data_ranked['SalePrice']
X_train_ranked, X_test_ranked, y_train_ranked, y_test_ranked = train_test_split(X_ranked, y_
tree_model_ranked = DecisionTreeRegressor(max_depth=3,
                                          min_samples_split=20,
                                          min_samples_leaf=10,
                                          random_state=123)
tree_model_ranked.fit(X_train_ranked, y_train_ranked)
```

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

```
# Feature importance with ranked zipCode
importance_ranked_df = pd.DataFrame({
    'Feature': X_train_ranked.columns,
    'Importance': tree_model_ranked.feature_importances_
}).sort_values('Importance', ascending=False)

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

print("Feature Importance (zipCode Ranked by Price):")
```

```
Feature Importance (zipCode Ranked by Price):
```

```
print(importance_ranked_df)
```

```
        Feature  Importance  Importance_Percent
8  zipcode_rank    0.530262               53.03
7    GarageCars    0.246034               24.60
2     GrLivArea    0.133405               13.34
6  TotRmsAbvGrd    0.069138                6.91
0       LotArea    0.021161                2.12
4      HalfBath    0.000000                0.00
3      FullBath    0.000000                0.00
1     YearBuilt    0.000000                0.00
5  BedroomAbvGr    0.000000                0.00
```

```
# Show zip code ranking
print("\nZip Code Ranking (by average price):")
```

```
Zip Code Ranking (by average price):
```

```
print(zipcode_ranking)
```

```
    zipCode      SalePrice  zipcode_rank
0     50010   98987.500000             1
1     50011  107916.666667             2
2     50017  110441.935484             3
3     50015  125588.425926             4
4     50036  130338.461538             5
5     50014  133173.489362             6
6     50016  136976.611940             7
7     50038  143031.250000             8
8     50037  147533.550505             9
9     50039  155410.714286            10
10    50042  188977.083333            11
11    50040  189392.812500            12
12    50041  191505.600000            13
13    50043  193799.296875            14
14    50045  193877.224806            15
15    50044  204863.651163            16
16    50046  217760.653061            17
17    50048  220993.000000            18
18    50047  238484.392857            19
19    50049  238772.727273            20
20    50050  264870.375000            21
21    50051  285046.666667            22
22    50026  328219.135135            23
```

Encoding Comparison Summary

R

```
cat("ENCODING COMPARISON SUMMARY\n")
```

```
ENCODING COMPARISON SUMMARY
```

```
cat("==========================\n")
```

```
==========================
```

```
# Get zipcode importance for each method
numerical_imp <- importance_df$Importance_Percent[importance_df$Feature == "zipCode"]
categorical_imp <- ifelse("zipCode" %in% names(tree_model_cat$variable.importance),
                          importance_cat$Importance_Percent[importance_cat$Feature == "zipCod
ranked_imp <- importance_ranked$Importance_Percent[importance_ranked$Feature == "zipcode_ran

cat("1. Numerical zipCode:", numerical_imp, "%\n")
```

```
1. Numerical zipCode: 7.02 %
```

```
cat("2. Categorical zipCode:", categorical_imp, "%\n")
```

```
2. Categorical zipCode: 16.51 %
```

```
cat("3. Ranked zipCode:", ranked_imp, "%\n\n")
```

```
3. Ranked zipCode: 22.54 %
```

```
cat("Key Insight: Target-based ranking preserves geographic price patterns\n")
```

```
Key Insight: Target-based ranking preserves geographic price patterns
```

```
cat("while avoiding the sparsity problem of one-hot encoding.\n")
```

```
while avoiding the sparsity problem of one-hot encoding.
```

Python

```
print("ENCODING COMPARISON SUMMARY")
```

```
ENCODING COMPARISON SUMMARY
```

```
print("==========================")
```

```
==========================
```

```
# Get zipcode importance for each method
numerical_imp = importance_df[importance_df['Feature'] == 'zipCode']['Importance_Percent'].il
onehot_imp = zipcode_percent  # from previous analysis
ranked_imp = importance_ranked_df[importance_ranked_df['Feature'] == 'zipcode_rank']['Importa

print(f"1. Numerical zipCode: {numerical_imp}%")
```

```
1. Numerical zipCode: 13.35%
```

```
print(f"2. One-hot zipCode: {onehot_imp}%")
```

```
2. One-hot zipCode: 0.0%
```

```
print(f"3. Ranked zipCode: {ranked_imp}%")
```

```
3. Ranked zipCode: 53.03%
```

```
print()
```

```
print("Key Insight: Target-based ranking preserves geographic price patterns")
```

```
Key Insight: Target-based ranking preserves geographic price patterns
```

```
print("while avoiding the sparsity problem of one-hot encoding.")
```

```
while avoiding the sparsity problem of one-hot encoding.
```

The Solution: Target-based ranking creates an ordinal variable that: - Preserves geographic price relationships - Avoids sparsity issues of one-hot encoding
- Allows meaningful splits like "zipcode_rank $>$ 3" (higher-priced areas) - Can be mapped back to actual zip codes for interpretation

This approach mimics how R factors work internally and provides the best of both worlds: meaningful geographic patterns without sparsity problems.

Statistical Concerns: Target-Based Ranking Controversy

The Problem: Target-based ranking uses the target variable (SalePrice) to create the independent variable (zipcode_rank), which is a form of **data leakage**.

Why Statisticians Object:

1. **Data Leakage:** Using target information to create features violates the independence assumption
2. **Overfitting Risk:** The model sees target information during training, inflating performance
3. **Circular Logic:** "We predict house prices using house prices" - the feature contains target information
4. **Generalization Issues:** May not work well on new data where zip code price patterns differ

Alternative Approaches:

1. Domain Knowledge Ranking:

```
# Rank by external factors (not target-dependent)
zipcode_ranking <- model_data %>%
  mutate(zipcode_rank = case_when(
    zipCode %in% c(50010, 50011) ~ 1,  # Low-income areas
    zipCode %in% c(50012, 50013) ~ 2,  # Medium-income areas
    zipCode %in% c(50014, 50015) ~ 3   # High-income areas
  ))
```

2. Separate Train/Validation Split:

```
# Use only training data to create rankings
train_zipcode_ranking <- train_data %>%
  group_by(zipCode) %>%
  summarise(avg_price = mean(SalePrice), .groups = 'drop') %>%
  arrange(avg_price) %>%
  mutate(zipcode_rank = row_number())
```

3. Cross-Validation Approach: - Create rankings within each CV fold - Prevents leakage between training and validation sets

The Statistical Verdict:

Target-based ranking is generally discouraged because it: - Violates fundamental statistical principles - Creates overly optimistic performance estimates - May not generalize to new data - Is considered "cheating" in predictive modeling

Better Practice: Use domain knowledge, external data, or proper train/validation splits to create meaningful categorical encodings without data leakage.

Important Clarification: R Factors vs Target-Based Ranking

Key Distinction: R's `as.factor()` does NOT use target information - it's fundamentally different from our target-based ranking example.

How R Factors Actually Work:

```
# R factor creation - NO target information used
model_data_cat <- model_data %>%
  mutate(zipCode = as.factor(zipCode))  # Just converts to categorical levels
```

What R Factors Do: - Convert numerical values to discrete categorical levels - Create internal mapping (50010 → level 1, 50011 → level 2, etc.) - Allow decision tree to make categorical splits (zipCode == "50010" vs zipCode == "50011") - **No target variable information is used in factor creation**

What Our Target-Based Ranking Did:

```
# This WAS data leakage - used target information
zipcode_ranking <- model_data %>%
  group_by(zipCode) %>%
  summarise(avg_price = mean(SalePrice), .groups = 'drop')  # ← Used SalePrice!
```

The Verdict: - **R factors:** Statistically legitimate - no data leakage - **Target-based ranking:** Data leakage - uses target information - **One-hot encoding:** Statistically legitimate - no data leakage (just sparse)

Why R Factors Work Well: - Decision trees can make meaningful categorical splits - No target information used in encoding - Preserves categorical nature without artificial numerical order - Statistically sound approach

The Real Issue: The sparsity problem with one-hot encoding is a practical limitation, not a statistical violation. R factors avoid this by allowing the tree to group categories naturally during the splitting process.

How R Handles Categorical Variables During Tree Building

Yes, R has sophisticated intermediate steps for categorical variables:

1. Category Grouping Algorithm:

```
# R internally considers all possible groupings of categorical levels
# For zipCode with levels: 50010, 50011, 50012, 50013, 50014
# R evaluates splits like:
# - {50010, 50011} vs {50012, 50013, 50014}
# - {50010, 50012} vs {50011, 50013, 50014}
# - {50010, 50011, 50012} vs {50013, 50014}
# - etc.
```

2. Optimal Grouping Selection: - R tests all possible ways to split categorical levels into two groups - Chooses the grouping that maximizes information gain (variance reduction) - This happens **during tree building**, not during data preprocessing

3. Why This Works Better Than One-Hot: - **One-hot encoding:** Creates separate binary variables, tree can only split on individual categories - **R factors:** Tree can group multiple categories together in a single split - **Result:** More flexible and meaningful splits

Example of R's Intermediate Process:

```
# R internally evaluates splits like:
# Split 1: zipCode in {50010, 50011} → predict $150K
#          zipCode in {50012, 50013, 50014} → predict $200K
#
# Split 2: zipCode in {50010, 50012} → predict $160K
#          zipCode in {50011, 50013, 50014} → predict $190K
#
# R chooses the split that best reduces prediction error
```

The Key Insight: R's algorithm is designed to handle categorical variables intelligently by finding optimal groupings during tree construction, while one-hot encoding forces the tree to work with individual binary variables that may be too sparse to be useful.

Computational Reality: How R Avoids the Combinatorial Nightmare

You're absolutely right! For k categories, there are 2^(k-1) - 1 possible binary splits, which grows exponentially.

R's Smart Approach:

```
# Instead of evaluating ALL possible groupings, R uses heuristics:

# 1. Sort categories by mean target value
zipcode_means <- c(50010=150000, 50011=160000, 50012=180000, 50013=200000, 50014=220000)

# 2. Only consider "adjacent" splits in the sorted order
# Split 1: {50010} vs {50011, 50012, 50013, 50014}
# Split 2: {50010, 50011} vs {50012, 50013, 50014}
# Split 3: {50010, 50011, 50012} vs {50013, 50014}
# Split 4: {50010, 50011, 50012, 50013} vs {50014}

# This reduces from 2^(k-1) - 1 = 15 splits to just k-1 = 4 splits!
```

Why This Works: - **Optimal splits are usually "adjacent"** when categories are sorted by target value - **Dramatically reduces computation** from exponential to linear - **Still finds meaningful groupings** without the combinatorial explosion

The Algorithm: 1. Sort categories by mean target value 2. Evaluate only k-1 "adjacent" splits 3. Choose the split with maximum information gain 4. **Result:** Near-optimal performance with linear complexity

For 20 zip codes: - **Naive approach:** 2^19 - 1 = 524,287 possible splits - **R's approach:** Only 19 splits to evaluate - **Performance:** Nearly identical results, 27,000x faster!

This is why R can handle categorical variables efficiently while still finding meaningful groupings.

Critical Clarification: Data Leakage vs. Tree Building Algorithm

You're absolutely right to question this! The distinction is subtle but crucial:

Our Target-Based Ranking (Data Leakage):

```
# We used target information BEFORE tree building
zipcode_ranking <- model_data %>%
  group_by(zipCode) %>%
  summarise(avg_price = mean(SalePrice), .groups = 'drop')  # ← Used SalePrice!
# Then we used this ranking to create features BEFORE training
```

R's Tree Building Algorithm (NOT Data Leakage):

```
# R uses target information DURING tree building, not before
# The algorithm looks at target values to find optimal splits
# This is part of the tree construction process itself
```

The Key Difference:

1. **Data Leakage (Bad):** Using target information to create features BEFORE model training
2. **Tree Algorithm (Good):** Using target information DURING the tree building process to find optimal splits

Why R's Approach is NOT Data Leakage: - **Tree algorithms inherently use target information** to find the best splits - This is the **core mechanism** of how decision trees work - **All decision tree algorithms** (R, Python, etc.) do this during tree construction - The target information is used **within the algorithm**, not to preprocess features

The Real Issue with Our Target-Based Ranking: - We used target information **outside** the tree building process - We created new features **before** training that contained target information - This is **feature engineering with data leakage**, not tree building

Analogy: - **Data Leakage:** Like giving a student the answer key before the exam - **Tree Algorithm:** Like a teacher using student answers to grade the exam (this is how grading works!)

Bottom Line: R's use of target information during tree building is the **normal, expected behavior** of decision tree algorithms. Our target-based ranking was problematic because we used target information for **feature engineering**, not tree building.

Conclusion

Key Takeaway: Treating zipCode as a numerical variable in decision trees leads to:

1. **Misleading splits** that have no meaningful interpretation
2. **Distorted feature importance** rankings
3. **False insights** about which variables actually matter for house price prediction

Next Steps: Proper categorical encoding would treat zipCode as discrete categories, revealing the true importance of each variable and providing interpretable splits that make business sense.

The decision tree example demonstrates why data preprocessing and proper variable encoding are crucial for interpretable machine learning models.