Random Forest Challenge

The Power of Weak Learners

Random Forest Challenge - The Power of Weak Learners

! Challenge Requirements In Section Student Analysis Section

- Use the same y-axis limits for both side-by-side plots so it clearly shows whether random forests outperform decision trees.
- Do not `echo` the code that creates the visualization

! Note on Python Usage

You have not been coached through setting up a Python environment. **If using Python** You will need to set up a Python environment and install the necessary packages to run this code - takes about 15 minutes; see https://quarto.org/docs/projects/virtual-environments.html. Alternatively, delete the Python code and only leave the remaining R code that is provided. You can see the executed Python output at my GitHub pages site: https://flyaflya.github.io/randomForestChallenge/.

The Problem: Can Many Weak Learners Beat One Strong Learner?

Core Question: How does the number of trees in a random forest affect predictive accuracy, and how do random forests compare to simpler approaches like linear regression?

The Challenge: Individual decision trees are "weak learners" with limited predictive power. Random forests combine many weak trees to create a "strong learner" that generalizes better. But how many trees do we need? Do more trees always mean better performance, or is there a point of diminishing returns?

Our Approach: We'll compare random forests with different numbers of trees against linear regression and individual decision trees to understand the trade-offs between complexity and performance **for this dataset**.

⚠ 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.

Data and Methodology

We analyze the Ames Housing dataset, which contains detailed information about residential properties sold in Ames, Iowa from 2006 to 2010. This dataset is ideal for our analysis because:

- **Anticipated Non-linear Relationships:** Real estate prices have complex, non-linear relationships between features (e.g., square footage in wealthy vs. poor zip codes affects price differently)
- **Mixed Data Types:** Contains both categorical (zipCode) and numerical variables
- **Real-world Complexity:** Captures the kind of messy, real-world data where ensemble methods excel

Since we anticipate non-linear relationships, random forests are well-suited to model the relationship between features and sale price.

R

```r
# Load libraries
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(randomForest))

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

# Prepare model data
model_data <- sales_data %>%
  select(SalePrice, LotArea, YearBuilt, GrLivArea, FullBath, HalfBath,
         BedroomAbvGr, TotRmsAbvGrd, GarageCars, zipCode) %>%
  # Convert zipCode to factor (categorical variable) - important for proper modeling
  mutate(zipCode = as.factor(zipCode)) %>%
  na.omit()

cat("Data prepared with zipCode as categorical variable\n")
```

```
Data prepared with zipCode as categorical variable
```

```r
cat("Number of unique zip codes:", length(unique(model_data$zipCode)), "\n")
```

```
Number of unique zip codes: 25
```

```r
# 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 random forests with different numbers of trees (with corrected categorical zipCode)
rf_1 <- randomForest(SalePrice ~ ., data = train_data, ntree = 1, mtry = 3, seed = 123)
rf_5 <- randomForest(SalePrice ~ ., data = train_data, ntree = 5, mtry = 3, seed = 123)
rf_25 <- randomForest(SalePrice ~ ., data = train_data, ntree = 25, mtry = 3, seed = 123)
rf_100 <- randomForest(SalePrice ~ ., data = train_data, ntree = 100, mtry = 3, seed = 123)
rf_500 <- randomForest(SalePrice ~ ., data = train_data, ntree = 500, mtry = 3, seed = 123)
rf_1000 <- randomForest(SalePrice ~ ., data = train_data, ntree = 1000, mtry = 3, seed = 123
rf_2000 <- randomForest(SalePrice ~ ., data = train_data, ntree = 2000, mtry = 3, seed = 123
rf_5000 <- randomForest(SalePrice ~ ., data = train_data, ntree = 5000, mtry = 3, seed = 123
```

Python

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
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
model_vars = ['SalePrice', 'LotArea', 'YearBuilt', 'GrLivArea', 'FullBath',
              'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd', 'GarageCars', 'zipCode']
model_data = sales_data[model_vars].dropna()

# Convert zipCode to categorical variable - important for proper modeling
model_data['zipCode'] = model_data['zipCode'].astype('category')

# 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 random forests with different numbers of trees (with corrected categorical zipCode)
rf_1 = RandomForestRegressor(n_estimators=1, max_features=3, random_state=123)
rf_5 = RandomForestRegressor(n_estimators=5, max_features=3, random_state=123)
rf_25 = RandomForestRegressor(n_estimators=25, max_features=3, random_state=123)
rf_100 = RandomForestRegressor(n_estimators=100, max_features=3, random_state=123)
rf_500 = RandomForestRegressor(n_estimators=500, max_features=3, random_state=123)
rf_1000 = RandomForestRegressor(n_estimators=1000, max_features=3, random_state=123)
rf_2000 = RandomForestRegressor(n_estimators=2000, max_features=3, random_state=123)
rf_5000 = RandomForestRegressor(n_estimators=5000, max_features=3, random_state=123)

# Fit all models
rf_1.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=1, random_state=123)
```

```
rf_5.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=5, random_state=123)
```

```
rf_25.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=25, random_state=123)
```

```
rf_100.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, random_state=123)
```

```
rf_500.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=500, random_state=123)
```

```
rf_1000.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=1000, random_state=123)
```

```
rf_2000.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=2000, random_state=123)
```

```
rf_5000.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=3, n_estimators=5000, random_state=123)
```

Results: The Power of Ensemble Learning

Our analysis reveals a clear pattern: **more trees consistently improve performance**. Let's examine the results and understand why this happens.

Performance Trends

R

```r
# Calculate predictions and performance metrics for test data
predictions_1_test <- predict(rf_1, test_data)
predictions_5_test <- predict(rf_5, test_data)
predictions_25_test <- predict(rf_25, test_data)
predictions_100_test <- predict(rf_100, test_data)
predictions_500_test <- predict(rf_500, test_data)
predictions_1000_test <- predict(rf_1000, test_data)
predictions_2000_test <- predict(rf_2000, test_data)
predictions_5000_test <- predict(rf_5000, test_data)

# Calculate predictions for training data
predictions_1_train <- predict(rf_1, train_data)
```

```r
predictions_5_train <- predict(rf_5, train_data)
predictions_25_train <- predict(rf_25, train_data)
predictions_100_train <- predict(rf_100, train_data)
predictions_500_train <- predict(rf_500, train_data)
predictions_1000_train <- predict(rf_1000, train_data)
predictions_2000_train <- predict(rf_2000, train_data)
predictions_5000_train <- predict(rf_5000, train_data)

# Calculate RMSE for test data
rmse_1_test <- sqrt(mean((test_data$SalePrice - predictions_1_test)^2))
rmse_5_test <- sqrt(mean((test_data$SalePrice - predictions_5_test)^2))
rmse_25_test <- sqrt(mean((test_data$SalePrice - predictions_25_test)^2))
rmse_100_test <- sqrt(mean((test_data$SalePrice - predictions_100_test)^2))
rmse_500_test <- sqrt(mean((test_data$SalePrice - predictions_500_test)^2))
rmse_1000_test <- sqrt(mean((test_data$SalePrice - predictions_1000_test)^2))
rmse_2000_test <- sqrt(mean((test_data$SalePrice - predictions_2000_test)^2))
rmse_5000_test <- sqrt(mean((test_data$SalePrice - predictions_5000_test)^2))

# Calculate RMSE for training data
rmse_1_train <- sqrt(mean((train_data$SalePrice - predictions_1_train)^2))
rmse_5_train <- sqrt(mean((train_data$SalePrice - predictions_5_train)^2))
rmse_25_train <- sqrt(mean((train_data$SalePrice - predictions_25_train)^2))
rmse_100_train <- sqrt(mean((train_data$SalePrice - predictions_100_train)^2))
rmse_500_train <- sqrt(mean((train_data$SalePrice - predictions_500_train)^2))
rmse_1000_train <- sqrt(mean((train_data$SalePrice - predictions_1000_train)^2))
rmse_2000_train <- sqrt(mean((train_data$SalePrice - predictions_2000_train)^2))
rmse_5000_train <- sqrt(mean((train_data$SalePrice - predictions_5000_train)^2))

# Calculate R-squared
r2_1 <- 1 - sum((test_data$SalePrice - predictions_1_test)^2) / sum((test_data$SalePrice - me
r2_5 <- 1 - sum((test_data$SalePrice - predictions_5_test)^2) / sum((test_data$SalePrice - me
r2_25 <- 1 - sum((test_data$SalePrice - predictions_25_test)^2) / sum((test_data$SalePrice -
r2_100 <- 1 - sum((test_data$SalePrice - predictions_100_test)^2) / sum((test_data$SalePrice
r2_500 <- 1 - sum((test_data$SalePrice - predictions_500_test)^2) / sum((test_data$SalePrice
r2_1000 <- 1 - sum((test_data$SalePrice - predictions_1000_test)^2) / sum((test_data$SalePric
r2_2000 <- 1 - sum((test_data$SalePrice - predictions_2000_test)^2) / sum((test_data$SalePric
r2_5000 <- 1 - sum((test_data$SalePrice - predictions_5000_test)^2) / sum((test_data$SalePric

# Create performance comparison
performance_df <- data.frame(
  Trees = c(1, 5, 25, 100, 500, 1000, 2000, 5000),
  RMSE_Test = c(rmse_1_test, rmse_5_test, rmse_25_test, rmse_100_test, rmse_500_test, rmse_1
  RMSE_Train = c(rmse_1_train, rmse_5_train, rmse_25_train, rmse_100_train, rmse_500_train, r
  R_squared = c(r2_1, r2_5, r2_25, r2_100, r2_500, r2_1000, r2_2000, r2_5000)
)

print(performance_df)
```

```
  Trees RMSE_Test RMSE_Train R_squared
1     1  42548.89   28137.39 0.7153393
2     5  35480.69   18667.02 0.8020593
3    25  34199.72   14460.33 0.8160939
4   100  34011.56   13894.56 0.8181119
5   500  33383.01   13881.62 0.8247726
6  1000  33634.04   13764.16 0.8221274
7  2000  33659.16   13738.77 0.8218616
8  5000  33523.12   13838.62 0.8232987
```

Python

```
# Calculate predictions for test data
predictions_1_test = rf_1.predict(X_test)
predictions_5_test = rf_5.predict(X_test)
predictions_25_test = rf_25.predict(X_test)
predictions_100_test = rf_100.predict(X_test)
predictions_500_test = rf_500.predict(X_test)
predictions_1000_test = rf_1000.predict(X_test)
predictions_2000_test = rf_2000.predict(X_test)
predictions_5000_test = rf_5000.predict(X_test)

# Calculate predictions for training data
predictions_1_train = rf_1.predict(X_train)
predictions_5_train = rf_5.predict(X_train)
predictions_25_train = rf_25.predict(X_train)
predictions_100_train = rf_100.predict(X_train)
predictions_500_train = rf_500.predict(X_train)
predictions_1000_train = rf_1000.predict(X_train)
predictions_2000_train = rf_2000.predict(X_train)
predictions_5000_train = rf_5000.predict(X_train)

# Calculate performance metrics for test data
rmse_1_test = np.sqrt(mean_squared_error(y_test, predictions_1_test))
```

```
rmse_5_test = np.sqrt(mean_squared_error(y_test, predictions_5_test))
rmse_25_test = np.sqrt(mean_squared_error(y_test, predictions_25_test))
rmse_100_test = np.sqrt(mean_squared_error(y_test, predictions_100_test))
rmse_500_test = np.sqrt(mean_squared_error(y_test, predictions_500_test))
rmse_1000_test = np.sqrt(mean_squared_error(y_test, predictions_1000_test))
rmse_2000_test = np.sqrt(mean_squared_error(y_test, predictions_2000_test))
rmse_5000_test = np.sqrt(mean_squared_error(y_test, predictions_5000_test))

# Calculate performance metrics for training data
rmse_1_train = np.sqrt(mean_squared_error(y_train, predictions_1_train))
rmse_5_train = np.sqrt(mean_squared_error(y_train, predictions_5_train))
rmse_25_train = np.sqrt(mean_squared_error(y_train, predictions_25_train))
rmse_100_train = np.sqrt(mean_squared_error(y_train, predictions_100_train))
rmse_500_train = np.sqrt(mean_squared_error(y_train, predictions_500_train))
rmse_1000_train = np.sqrt(mean_squared_error(y_train, predictions_1000_train))
rmse_2000_train = np.sqrt(mean_squared_error(y_train, predictions_2000_train))
rmse_5000_train = np.sqrt(mean_squared_error(y_train, predictions_5000_train))

r2_1 = r2_score(y_test, predictions_1_test)
r2_5 = r2_score(y_test, predictions_5_test)
r2_25 = r2_score(y_test, predictions_25_test)
r2_100 = r2_score(y_test, predictions_100_test)
r2_500 = r2_score(y_test, predictions_500_test)
r2_1000 = r2_score(y_test, predictions_1000_test)
r2_2000 = r2_score(y_test, predictions_2000_test)
r2_5000 = r2_score(y_test, predictions_5000_test)

# Create performance comparison
performance_data = {
    'Trees': [1, 5, 25, 100, 500, 1000, 2000, 5000],
    'RMSE_Test': [rmse_1_test, rmse_5_test, rmse_25_test, rmse_100_test, rmse_500_test, rmse_
    'RMSE_Train': [rmse_1_train, rmse_5_train, rmse_25_train, rmse_100_train, rmse_500_train
    'R_squared': [r2_1, r2_5, r2_25, r2_100, r2_500, r2_1000, r2_2000, r2_5000]
}

performance_df = pd.DataFrame(performance_data)
print(performance_df)
```

```
   Trees     RMSE_Test    RMSE_Train  R_squared
0      1  48915.173827  25868.676052   0.474720
1      5  36713.731261  14970.364692   0.704089
2     25  31703.877433  11732.517136   0.779338
```

```
3    100  29883.435196  10794.958719   0.803951
4    500  29480.013577  10543.752687   0.809209
5   1000  29485.398882  10476.382036   0.809139
6   2000  29457.539635  10550.708743   0.809499
7   5000  29369.918560  10520.683655   0.810631
```

Student Analysis Section: The Power of More Trees

Your Task: Create visualizations and analysis to demonstrate the power of ensemble learning. You'll need to create three key components:

1. The Power of More Trees Visualization

Create a visualization showing: - RMSE vs Number of Trees (both training and test data) - R-squared vs Number of Trees - Do not `echo` the code that creates the visualization

Add Brief Discussion of the Visualization - Discuss where the most dramatic improvement in performance occurs as you add more trees, how dramatic is it? - Discuss diminishing returns as you add more trees

! Visualization Requirements

Create two plots: 1. **RMSE Plot:** Show how RMSE decreases with more trees (both training and test) 2. **R-squared Plot:** Show how R-squared increases with more trees Use log scale on x-axis to better show the relationship across the range of tree counts.

2. Overfitting Visualization and Analysis

Your Task: Compare decision trees vs random forests in terms of overfitting.

Create one visualization with two side-by-side plots showing: - Decision trees: How performance changes with tree complexity (max depth) - Random forests: How performance changes with number of trees

Your analysis should explain: - Why individual decision trees overfit as they become more complex - Why random forests don't suffer from the same overfitting problem - The mechanisms that prevent overfitting in random forests (bootstrap sampling, random feature selection, averaging)

! Overfitting Analysis Requirements

Create a side-by-side comparison showing: 1. **Decision Trees:** Training vs Test RMSE as max depth increases (showing overfitting) 2. **Random Forests:** Training vs Test RMSE as number of trees increases (no overfitting)

- Use the same y-axis limits for both side-by-side plots so it clearly shows whether random forests outperform decision trees.
- Do not `echo` the code that creates the visualization

3. Linear Regression vs Random Forest Comparison

Your Task: Compare random forests to linear regression baseline.

Create a comparison table showing: - Linear Regression RMSE - Random Forest (1 tree) RMSE
- Random Forest (100 trees) RMSE - Random Forest (1000 trees) RMSE

Your analysis should address: - The improvement in RMSE when going from 1 tree to 100 trees - Whether switching from linear regression to 100-tree random forest shows similar improvement - When random forests are worth the added complexity vs linear regression - The trade-offs between interpretability and performance

! Comparison Requirements

Create a clear table comparing: - Linear Regression - Random Forest (1 tree) - Random Forest (100 trees) - Random Forest (1000 trees)
Include percentage improvements over linear regression for each random forest model.

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. Use just one language for the analysis and visualizations, delete the other language and omit the panel tabsets.
2. **Add Analysis and Visualizations:** Complete the three analysis sections above with your own code and insights.
3. **GitHub Repository:** Use your forked repository (from the starter repository) named "randomForestChallenge" 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/randomForestChallenge/`
    - **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/randomForestChallenge.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 visualizations and analysis, not data preparation

Getting Started Tips

Navy SEALs Motto

"Slow is Smooth and Smooth is Fast"

Take your time to understand the random forest 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 visualizations
- After adding your analysis
- 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!

Grading Rubric

❗ What You're Really Being Graded On

This is an investigative report, not a coding exercise. You're analyzing random forest 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 the power of ensemble learning and when to use random forests vs simpler approaches.
What makes a great report:

- **Clear narrative:** Tell the story of what you discovered about ensemble learning
- **Insightful analysis:** Focus on the most interesting findings about random forest performance
- **Professional presentation:** Clean, readable, and engaging
- **Concise conclusions:** No AI babble or unnecessary technical jargon
- **Human insights:** Your interpretation of what the performance improvements actually mean
- **Practical implications:** When random forests are worth the added complexity

What we're looking for: A compelling 2-3 minute read that demonstrates both the power of ensemble learning and the importance of choosing the right tool for the job.

Questions to Answer for 75% Grade on Challenge

1. **Power of More Trees Analysis:** Provide a clear, well-reasoned analysis of how random forest performance improves with more trees. Your analysis should demonstrate understanding of ensemble learning principles and diminishing returns.

Questions to Answer for 85% Grade on Challenge

2. **Overfitting Analysis:** Provide a thorough analysis comparing decision trees vs random forests in terms of overfitting. Your analysis should explain why individual trees overfit while random forests don't, and the mechanisms that prevent overfitting in ensemble methods.

Questions to Answer for 95% Grade on Challenge

3. **Linear Regression Comparison:** Your analysis should include a clear comparison table and discussion of when random forests are worth the added complexity vs linear regression. Focus on practical implications for real-world applications.

Questions to Answer for 100% Grade on Challenge

4. **Professional Presentation:** Your analysis should be written in a professional, engaging style that would be appropriate for a business audience. Use clear visualizations and focus on practical insights rather than technical jargon.

Submission Checklist

Minimum Requirements (Required for Any Points):

- ☐ Forked starter repository from https://github.com/flyaflya/randomForestChallenge.git
- ☐ Cloned repository locally using Cursor (or VS Code)
- ☐ Completed all three analysis sections with visualizations
- ☐ 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/randomForestChallenge/`

75% Grade Requirements:

- ☐ Clear analysis of how random forest performance improves with more trees
- ☐ Discussion of diminishing returns in ensemble learning

85% Grade Requirements:

- ☐ Thorough overfitting analysis comparing decision trees vs random forests
- ☐ Explanation of mechanisms that prevent overfitting in random forests

95% Grade Requirements:

- ☐ Complete linear regression comparison with clear table
- ☐ Discussion of when random forests are worth the complexity

100% Grade Requirements:

- ☐ Professional presentation style appropriate for business audience
- ☐ Clear, engaging narrative that tells a compelling story
- ☐ Practical insights that would help a real data scientist

Report Quality (Critical for Higher Grades):

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

☐ Practical insights that would help a real data scientist
☐ Well-designed visualizations that support your analysis