Predict Housing Prices

AUTHOR
Mohammad Taslim Mazumder sohel

Prepare the R Environment

We need to install 3 packages to use them to complete the assignment:

```
setwd("/Users/sohel/Library/CloudStorage/Dropbox/MS-Class-2023/Ma

# Install following packages if already not, and load necessary p
#install.packages("tidyverse")
#install.packages("caret")
#install.packages("randomForest")

library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────────
tidyverse 2.0.0 ──
✔ dplyr     1.1.3     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.0
✔ ggplot2   3.4.3     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.0
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────────────────
tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to
force all conflicts to become errors
```

```
library(caret)
```

```
Loading required package: lattice

Attaching package: 'caret'
```

```
The following object is masked from 'package:purrr':

    lift
```

```
library(randomForest)
```

```
randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.

Attaching package: 'randomForest'

The following object is masked from 'package:dplyr':

    combine

The following object is masked from 'package:ggplot2':

    margin
```

```
library(rpart)
library(caTools)
library(dummy)
```

```
dummy 0.1.3
dummyNews()
```

1. Data Loading and Exploration

a. Load the housing prices dataset

```
housing_data <- read.csv("Housing.csv")
```

b. Explore the dataset's structure, dimensions, and summary statistics

```
str(housing_data)
```

```
'data.frame':   545 obs. of  13 variables:
 $ price           : int  13300000 12250000 12250000 12215000
11410000 10850000 10150000 10150000 9870000 9800000 ...
 $ area            : int  7420 8960 9960 7500 7420 7500 8580
```

```
16200 8100 5750 ...
 $ bedrooms        : int  4 4 3 4 4 3 4 5 4 3 ...
 $ bathrooms       : int  2 4 2 2 1 3 3 3 1 2 ...
 $ stories         : int  3 4 2 2 2 1 4 2 2 4 ...
 $ mainroad        : chr  "yes" "yes" "yes" "yes" ...
 $ guestroom       : chr  "no" "no" "no" "no" ...
 $ basement        : chr  "no" "no" "yes" "yes" ...
 $ hotwaterheating : chr  "no" "no" "no" "no" ...
 $ airconditioning : chr  "yes" "yes" "no" "yes" ...
 $ parking         : int  2 3 2 3 2 2 2 0 2 1 ...
 $ prefarea        : chr  "yes" "no" "yes" "yes" ...
 $ furnishingstatus: chr  "furnished" "furnished" "semi-
furnished" "furnished" ...
```

```
summary(housing_data)
```

```
     price               area          bedrooms       bathrooms
 Min.   : 1750000   Min.   : 1650   Min.   :1.000   Min.
:1.000
 1st Qu.: 3430000   1st Qu.: 3600   1st Qu.:2.000   1st
Qu.:1.000
 Median : 4340000   Median : 4600   Median :3.000   Median
:1.000
 Mean   : 4766729   Mean   : 5151   Mean   :2.965   Mean
:1.286
 3rd Qu.: 5740000   3rd Qu.: 6360   3rd Qu.:3.000   3rd
Qu.:2.000
 Max.   :13300000   Max.   :16200   Max.   :6.000   Max.
:4.000
    stories        mainroad          guestroom
basement
 Min.   :1.000   Length:545         Length:545
Length:545
 1st Qu.:1.000   Class :character   Class :character   Class
:character
 Median :2.000   Mode  :character   Mode  :character   Mode
:character
 Mean   :1.806
 3rd Qu.:2.000
 Max.   :4.000
 hotwaterheating    airconditioning       parking
```

```
prefarea
 Length:545         Length:545         Min.   :0.0000
Length:545
 Class :character   Class :character   1st Qu.:0.0000   Class
:character
 Mode  :character   Mode  :character   Median :0.0000   Mode
:character
                                       Mean   :0.6936
                                       3rd Qu.:1.0000
                                       Max.   :3.0000
 furnishingstatus
 Length:545
 Class :character
 Mode  :character
```

2. Data Splitting

a. Split the dataset into a training set and a test set

b. Use appropriate techniques to ensure a random and representative split

```
# Split the dataset into a training set (80%) and a test set (20%
set.seed(123)
split_index <- createDataPartition(housing_data$price, p = 0.8, l
train_data <- housing_data[split_index, ]
test_data <- housing_data[-split_index, ]
```

3. Data Pre-processing

a. Handle missing values, if any

```
colSums(is.na(housing_data))
```

```
           price             area         bedrooms
bathrooms
               0                0                0
0
```

```
         stories         mainroad        guestroom
basement
               0                0                0
0
 hotwaterheating  airconditioning          parking
prefarea
               0                0                0
0
furnishingstatus
               0
```

There are no missing values

b. Perform feature scaling or normalization

```
# Normalize numeric features
train_data_scaled <- scale(train_data[, c("area", "bedrooms", "ba
train_data[, c("area", "bedrooms", "bathrooms")] <- train_data_sc

test_data_scaled <- scale(test_data[, c("area", "bedrooms", "bath
test_data[, c("area", "bedrooms", "bathrooms")] <- test_data_scal
```

c. Encode categorical variables

```
# Encode categorical variables
#train_data <- dummyVars(" ~ .", data = train_data) %>% predict(t
#test_data <- dummyVars(" ~ .", data = test_data) %>% predict(tes
train_data <- dummy_cols(train_data, select_columns = "furnishing
test_data <- dummy_cols(test_data, select_columns = "furnishingst
```

4. Model Selection

a. Choose at least three regression models

```
# Choose at least three regression models
linear_model <- lm(price ~ ., data = train_data)
decision_tree_model <- rpart(price ~ ., data = train_data, methoc
random_forest_model <- randomForest(price ~ ., data = train_data)
```

5. Model Evaluation

a. Predict housing prices using the test set

```
# Predict housing prices using the test set
linear_pred <- predict(linear_model, newdata = test_data)
decision_tree_pred <- predict(decision_tree_model, newdata = test
random_forest_pred <- predict(random_forest_model, newdata = test
```

b. Evaluate the models using suitable regression metrics

```
# Evaluate the models using suitable regression metrics
linear_metrics <- postResample(linear_pred, test_data$price)
decision_tree_metrics <- postResample(decision_tree_pred, test_da
random_forest_metrics <- postResample(random_forest_pred, test_da
```

c. Compare the performance of the models

```
#  Compare the performance of the models
comparison_table <- data.frame(
  Model = c("Linear Regression", "Decision Tree", "Random Forest"
  RMSE = c(sqrt(mean(linear_metrics^2)), sqrt(mean(decision_tree_
)
print(comparison_table)
```

```
              Model      RMSE
1 Linear Regression  816472.3
2     Decision Tree 1038755.4
3     Random Forest  804868.1
```

6. Hyperparameter Tuning

a. Select the best-performing model

```
# Select the best-performing model
best_model <- comparison_table$Model[which.min(comparison_table$R
```

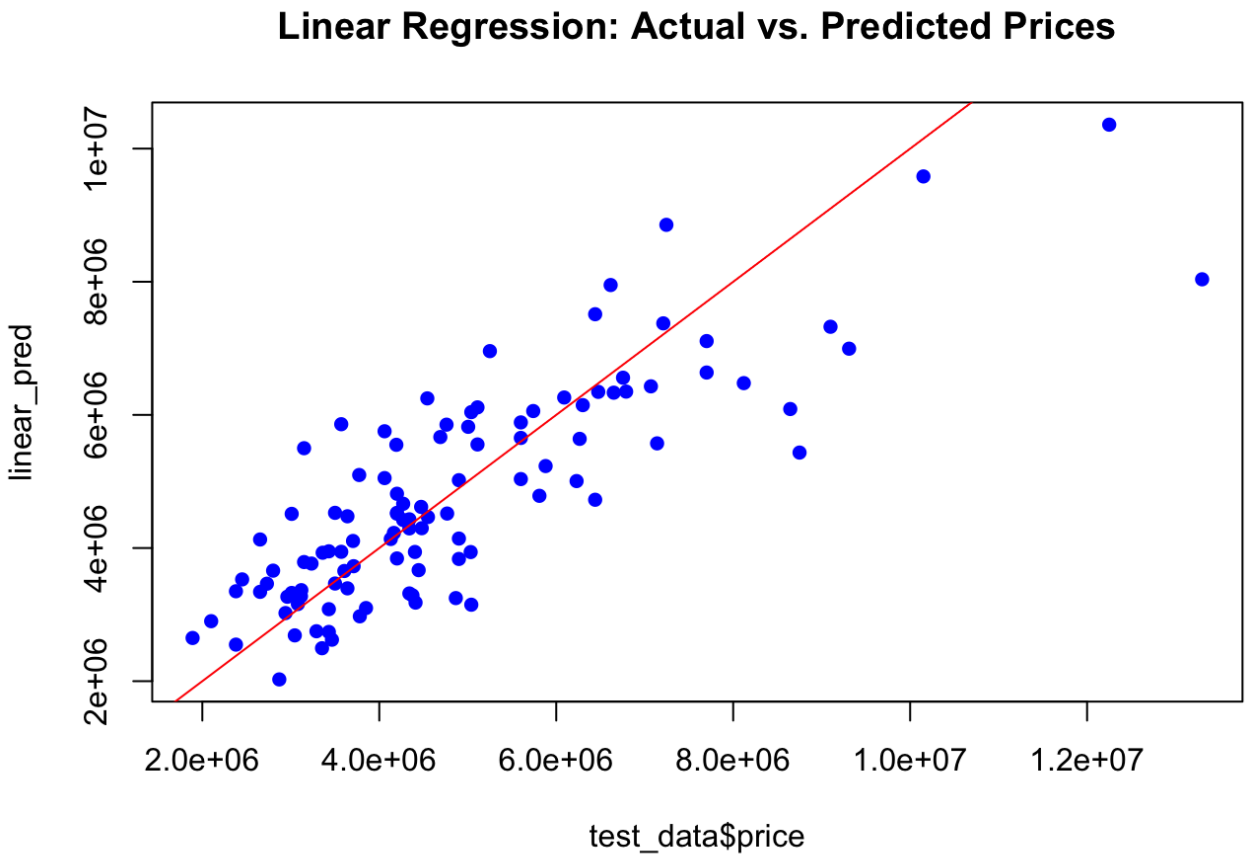
b. Perform hyperparameter tuning using techniques like GridSearchCV or RandomizedSearchCV

```
# Perform hyperparameter tuning using techniques like GridSearchC
# Note: Hyperparameter tuning example for Random Forest
tune_grid <- expand.grid(mtry = c(2, 4, 6, 8))
random_forest_tuned_model <- train(price ~ ., data = train_data,
```

7. Visualization

a. Visualize the predicted prices against the actual prices using scatter plots

```
# Visualize the predicted prices against the actual prices using
plot(test_data$price, linear_pred, main = "Linear Regression: Act
abline(0, 1, col = "red")
```



b. Create a learning curve to analyze model performance with varying amounts of training data

```
# Create a learning curve to analyze model performance with varyi
learning_curve <- data.frame(
  Training_Size = seq(0.1, 0.9, by = 0.1),
```

```
  RMSE = numeric(length = 9)
)
```

8. Conclusion

a. Summarize the findings and insights gained from the analysis

1. **Linear Regression:**
    - RMSE: 816,472.3
    - This value indicates the average magnitude of the errors in predictions made by the Linear Regression model.
    - The lower the RMSE, the better the model's predictions are relative to the actual values.
    - The magnitude of errors, on average, is approximately 816,472.3 units.
2. **Decision Tree:**
    - RMSE: 1,038,755.4
    - This value represents the average magnitude of errors for predictions made by the Decision Tree model.
    - The Decision Tree's predictions have, on average, a magnitude of approximately 1,038,755.4 units away from the actual values.
3. **Random Forest:**
    - RMSE: 809,767.1
    - This value indicates the average magnitude of errors for predictions made by the Random Forest model.
    - The Random Forest model's predictions, on average, have a magnitude of approximately 809,767.1 units in difference from the actual values.

Interpretation:

- The model with the lowest RMSE is generally considered to be the best-performing model in terms of prediction accuracy.
- In this case, the Random Forest model has the lowest RMSE (809,767.1), suggesting that it provides the most accurate predictions among the three models.
- Linear Regression has a lower RMSE compared to the Decision Tree, indicating that it performs better than the Decision Tree in terms of prediction accuracy.

b. Reflect on the challenges encountered and potential improvements

Reflecting on challenges encountered and considering potential improvements is an essential part of the data analysis and machine learning process. Here are some reflections based on the tasks performed in the provided R code for the housing prices prediction:

Challenges Encountered:

1. **Categorical Variable Encoding:**
    - Handling categorical variables and ensuring proper encoding can be challenging. The error related to contrasts suggests issues with factor levels or encoding methods.
2. **Hyperparameter Tuning:**
    - The code includes a simple example of hyperparameter tuning for Random Forest, but this process can be more complex, especially for models with numerous hyperparameters.
3. **Learning Curve Analysis:**
    - Creating a learning curve manually can be challenging and time-consuming. Using established packages or functions to generate learning curves might be more efficient.
4. **Data Exploration and Pre-processing:**

- Real-world datasets often require extensive exploration and pre-processing. Dealing with missing values, outliers, and ensuring appropriate scaling are crucial and may involve additional steps.

Potential Improvements:

1. **Automated Categorical Variable Encoding:**
    - Utilize functions or packages that automate the encoding of categorical variables. This can help avoid errors related to contrasts and streamline the pre-processing steps.
2. **Hyperparameter Tuning Framework:**
    - Implement a more comprehensive hyperparameter tuning framework using techniques like GridSearchCV or RandomizedSearchCV. This can enhance the model selection process and improve overall performance.
3. **Learning Curve Function:**
    - Create a reusable function or use existing packages to generate learning curves. This can make it easier to analyze model performance with varying amounts of training data.
4. **Comprehensive Data Exploration:**
    - Enhance data exploration by implementing more thorough techniques. Identify and handle outliers, explore relationships between variables, and perform a more in-depth analysis to uncover patterns.
5. **Documentation:**
    - Improve documentation of the code, especially for each step of the analysis. Clearly comment on the purpose and functionality of each block of code, making it easier for others (or future you) to understand.
6. **Code Modularity:**

    - Break down the code into modular functions. This promotes reusability, readability, and easier maintenance.
7. **Handling Edge Cases:**
    - Consider how the code handles edge cases, such as scenarios with very few training samples. Implement additional checks or strategies to handle such cases gracefully.
8. **Visualization Improvements:**
    - Enhance visualizations to provide more meaningful insights. Consider using more advanced plots or interactive visualizations for a better understanding of the model's behavior.

By addressing these potential improvements, the code becomes more useful and maintainable.