Data Mining Project

Yinan Zhou

October 26, 2024

1. Introduction

The diamond dataset diamonds.csv contains the 10 attributes of 53940 diamonds. We assume the data owners are diamond merchants or customers, who are eager to discover practical strategies of evaluating the diamond prices. Therefore, we select `price` as the response variable, and use the remaining 9 varaibles: `carat`, `cut`, `color`, `clarity`, `x`, `y`, `z`, `depth` and `table` as the features. The definition of each variable is available in Appendix A.

This report is divided into four key sections: (1) Exploratory Data Analysis (EDA), (2) Feature Engineering and Selection, (3) Classification Models, and (4) Regression Models. Each section explains the methodologies used, analyzes the results, interprets the findings. Finally, we offer recommendations for enhancing diamond price prediction and summarize our data mining efforts with this dataset.

2. Exploratory Data Analysis

2.1 Clean Dataset

We load the data using function `read.csv`, and store it in the variable `raw_data`. The raw data has 53940 rows and 11 columns. However, the first column 'X' is the useless index. We removed it. Therefore, the dataset has 53940 entries and 10 variables. Due to space constraints, details are printed in Appendix B1.

```
raw_data <- read.csv("diamonds.csv") # Reading the data set as a dataframe
diamonds <- raw_data[-1] # Remove the column of entry indices
dim(diamonds) # Print dimensions of the dataset
```

```
## [1] 53940    10
```

Fortunately, this dataset do not have any missing value. However, we catch 146 duplicated rows (not include their first appearance). These duplicated entry pairs have exactly the same price and all other attributes. After removing all duplications, 53794 entries remains in the dataset.

```
diamonds_clean <- diamonds[!duplicated(diamonds),]# Remove duplicated rows
```

The outliers of numerical variables are visualized using boxplots, shown in Appendix B2. The outliers are represented by dots in the box plots. There are many outliers in every numerical variable.

Some outliers are suspicious. For example, the entries with $x = 0$ or $y = 0$ or $z = 0$. Any one of these variables equals 0 should be wrong. There are two situations: (1) $x = 0$, $y = 0$ and $z = 0$; and (2) $x > 0$, $y > 0$ and $z = 0$. For the first situation, we directly remove the corresponding entries because they are

unable to be calculated. For the second situation, we calculate the z value using the known formula: depth = 200*z/(x+y). After our processing, only 7 suspicious outliers need be removed. Now there are 53787 entries in the dataset.

```
# Collect all suspicious outliers with x = 0 or y = 0 or z = 0
suspicious_outliers <- diamonds_clean[(diamonds_clean$x == 0)
              | (diamonds_clean$y == 0) | (diamonds_clean$z == 0),]
# Fix the suspicious outliers that can be fixed
outliers_can_fix <- which(diamonds_clean$x > 0
                          & diamonds_clean$y > 0 & diamonds_clean$z == 0)
diamonds_clean$z[outliers_can_fix] <- diamonds_clean$depth[outliers_can_fix] *
  (diamonds_clean$x[outliers_can_fix] + diamonds_clean$y[outliers_can_fix])/200
# Remove the 7 remaining suspicious outliers with x=0, y=0 and z=0
diamonds_clean <- diamonds_clean[!(diamonds_clean$x == 0 |
                  diamonds_clean$y == 0 | diamonds_clean$z == 0), ]
```

Until now, we have already cleaned the data by (1) removing the useless columns of the row index; (2) removing 146 repeated rows; (3) fixing 12 rows with z = 0, x, y > 0; (4) removing 7 rows with z = 0, x = 0 and y = 0, which are the suspicious outliers that is unable to fix.

Lastly, we verify that each entry conforms to the predefined equation: 100*depth = z/mean(x,y). Due to the `x`, `y`, and `z` values being recorded with only two decimal places, we allow a tolerance of 3% to account for rounding differences when verifying the accuracy of the calculated depth against the recorded depth. A total of 49 entries show a mismatch between the calculated and recorded depths. These entries are removed to ensure the reliability of the dataset. Observing the boxplots again in Appendix B2, we decide that larger outliers in `y` and `z` should also be removed. The final cleaned dataset contains 53738 rows and 10 columns.

```
# Verify the equation of calculating depth
diamonds_clean$calculated_depth <- with(diamonds_clean, round(100*z / ((x + y) / 2),1))
diamonds_clean$depth_error <- with(diamonds_clean,
                                   round(abs(depth - calculated_depth) /depth,2))
# Clean the dataset: remove entries with more than 3% depth error or larger outliers
diamonds_cleaned <- diamonds_clean[(diamonds_clean$depth_error <= 0.03)
                                   & (diamonds_clean$y<30 | diamonds_clean$z <30), ]
diamonds_cleaned$calculated_depth <- NULL; diamonds_cleaned$depth_error <- NULL
dim(diamonds_cleaned)
```

```
## [1] 53738    10
```

2.2 Data Distribution

We visualize the 7 continuous numerical variables by the histograms and visualize the 3 categorical variables by the bar plots, as shown in Appendix B3. The variables `depth` and `table` are normally distributed. The distributions of `carat`, `price`, `x`, `y`, and `z` are right-skewed, meaning most diamonds have lower carat, lower price and smaller dimensions. The distribution of `cut` is fairly uneven, with a large proportion of diamonds rated as "Very Good," followed by "Premium" and "Ideal." There are fewer "Fair" and "Good" diamonds. The color distribution is relatively balanced, with most diamonds having a color grade between D and G. Grades H to J appear less frequently. Most diamonds fall into the SI1 and VS2 clarity categories, indicating slight inclusions or very slight inclusions. The I1 and IF categories are much less common.

2.3 Variable Correlation

For the 7 numerical variables, we use `ggpairs()` to visualizes the correlation matrix, as shown in Appendix B4. To reduce the file size of the pdf file, we sampled 1000 observations from the original 53738. This

visualization helps in identifying potential relationships and patterns that could inform further analysis or modeling efforts related to diamond pricing.

Analysis based on the correlation visualization: (1) Carat, dimensions (x, y, z), and price are strongly related to each other and (2) Depth and table do not significantly affect the price of diamonds, and depth shows a weak inverse relationship with carat size.

3. Feature Engineering and Selection

3.1 Create new variables: average_size and price_per_carat

The scatter plots show positive relationships between price and the size-related variables (carat, x, y, z). These relationships are mostly linear with some dispersion. Such dispersion and how is it related to other variables could be of the data owner's interest. The strong linear relationships between x, y, and z indicate that these dimensions are highly proportional to each other. Therefore, we use a new variable `average_size` = (x+y+z)/3 to replace the original sizes in three directions.

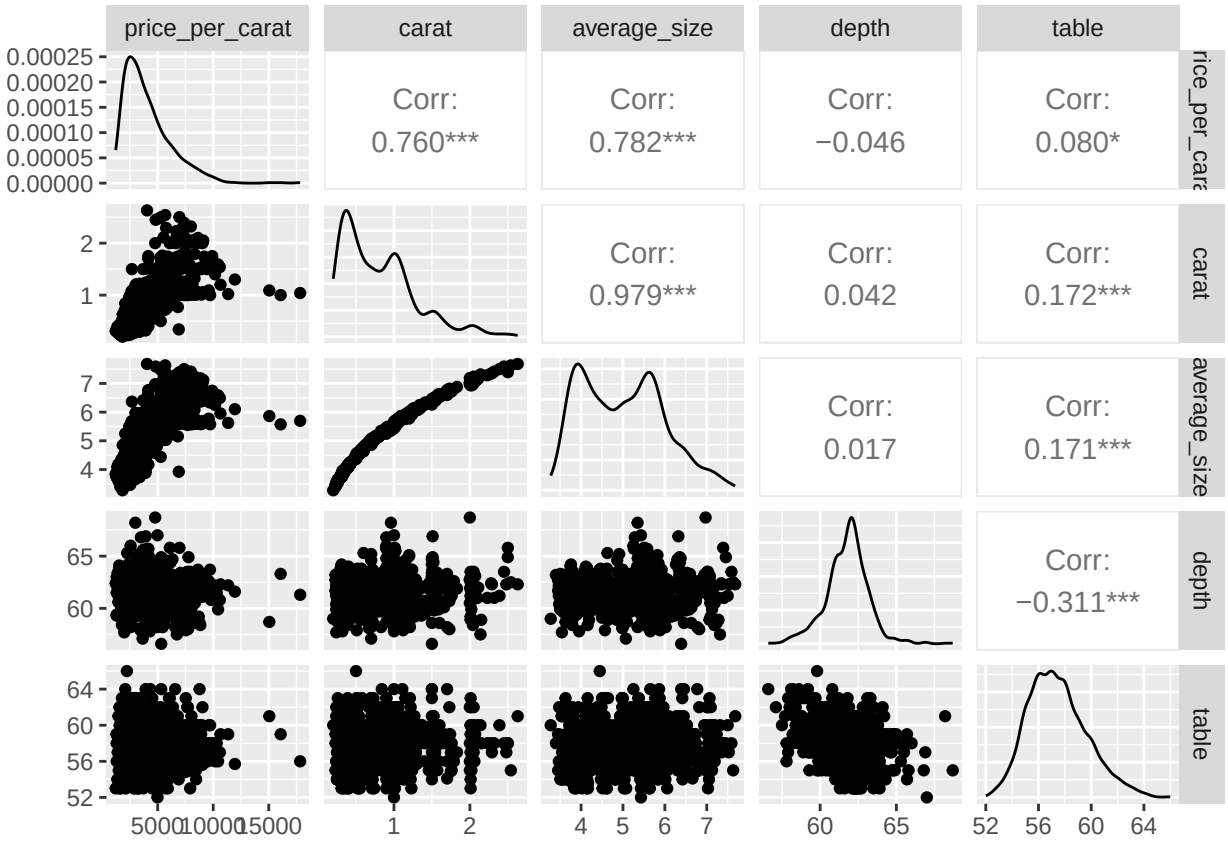
```
# Create the new variable 'average_size'
diamonds <- diamonds_cleaned
diamonds$average_size <- with(diamonds, (x + y + z) / 3)
```

Another new variable `price_per_carat` = price/carat could better reveal how does the other characteristics like cut, color, clarity affect the price in the following data mining procedure.

```
library(ggplot2)
library(GGally) # Load library
```

```
## Warning: package 'GGally' was built under R version 4.3.3
```

```
# Create the new variable 'price_per_carat'
diamonds$price_per_carat <- with(diamonds, price/carat)
# Select the right-skewed and related variables
vars_new <- diamonds[, c("price_per_carat", "carat", "average_size", "depth", "table")]
# Visualize pairwise relationships between the selected variables
ggpairs(vars_new[sample(53738, 1000), ])
```



Analysis of the new pairwise plots:

(1) carat and average_size still have a very strong positive correlation of 0.979, which makes sense because carat is highly related to the overall dimensions of the diamond, captured by average_size.

(2) price_per_carat still has a strong positive correlation with both carat (0.770) and average_size (0.792). This means that as either carat or average_size increases, price_per_carat tends to increase as well.

(3) depth and table still do not significantly affect the price_per_carat.

3.2 Create a new categorical response variable

Training a model to predict the exact price of diamonds could be challenging. Weplan to get a classification model at this stage: predict the price per carat is higher than the median value or not. A new binary response variable `expensive` is created by comparing the price_per_carat with its median. This can also guarantee the dataset is balanced.

The following code calculates the median value of the price_per_carat variable from the diamonds_cleaned dataset and creates a new binary variable named expensive, which assigns a value of 1 to diamonds with a price_per_carat greater than the median and 0 otherwise. It then constructs a new dataset called diamonds, which retains only the relevant columns, including the newly created expensive variable, along with other features such as carat, depth, table, cut, color, clarity, and average_size, along with price_per_carat.

```
# Calculate the median of price_per_carat
median_p <- median(diamonds$price_per_carat)
# Create a binary variable (1 if price_per_carat > median, 0 otherwise)
diamonds$expensive <- ifelse(diamonds$price_per_carat > median_p, 1, 0)
```

4. Classification Models

4.1 Split Data

Weprepare the diamonds dataset for analysis by converting the categorical variables cut, color, and clarity into factors, which is essential for accurate modeling and statistical analysis in R. Then Wesplit the dataset into training and testing sets, allocating 60% of the data for training and 40% for testing, ensuring that the results can be validated on a separate subset. The random seed is set to 12345 to ensure reproducibility of the random sampling process, allowing for consistent results across different runs of the code. The resulting train_data contains the sampled observations for model training, while test_data consists of the remaining observations for evaluating model performance.

```
# Convert categorical variables to factors
diamonds$cut <- as.factor(diamonds$cut)
diamonds$color <- as.factor(diamonds$color)
diamonds$clarity <- as.factor(diamonds$clarity)
# Split the data into training (60%) and testing (40%) sets
set.seed(12345)
train_index <- sample(seq_len(nrow(diamonds)), size = 0.6 * nrow(diamonds))
train_data <- diamonds[train_index, ]; test_data <- diamonds[-train_index, ]
```

4.2 Logistic Regression

The model is created using the glm() function with predictor variables such as carat, depth, table, cut, color, clarity, and average_size, and response varaiable expensive. After building the model, a summary is displayed to show the coefficients and statistical significance of each variable. By setting type = "response", the model then predicts the probabilities of diamonds being classified as expensive in the test dataset (test_data). These probabilities are converted into binary predictions (0 or 1) using a threshold of 0.5. A confusion matrix is generated to compare the predicted values with the actual expensive labels in the test data. Finally, the accuracy of the model is calculated by comparing the predictions to the actual values, and the result is printed.

```
# Build a logistic regression model
model_logit <- glm(expensive ~ carat + depth + table + cut + color
 + clarity + average_size, family = binomial, data = train_data)
# Summary of the logistic model
summary(model_logit)
```

```
##
## Call:
## glm(formula = expensive ~ carat + depth + table + cut + color +
##     clarity + average_size, family = binomial, data = train_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -83.30786    2.52677 -32.970   <2e-16 ***
## carat        -10.03084    0.56534 -17.743   <2e-16 ***
## depth          0.18473    0.02192   8.429   <2e-16 ***
## table          0.03383    0.01575   2.149   0.0317 *
## cutGood        1.56745    0.15921   9.845   <2e-16 ***
## cutIdeal       2.57174    0.16269  15.808   <2e-16 ***
## cutPremium     1.78882    0.15510  11.533   <2e-16 ***
```

```
## cutVery Good   1.95654    0.15298  12.789   <2e-16 ***
## colorE        -1.45120    0.09896 -14.664   <2e-16 ***
## colorF        -2.08422    0.10286 -20.263   <2e-16 ***
## colorG        -2.95487    0.10785 -27.399   <2e-16 ***
## colorH        -4.24094    0.12800 -33.131   <2e-16 ***
## colorI        -5.87320    0.14812 -39.653   <2e-16 ***
## colorJ        -7.39581    0.18292 -40.432   <2e-16 ***
## clarityIF     17.83217    0.40327  44.219   <2e-16 ***
## claritySI1     9.99393    0.30732  32.520   <2e-16 ***
## claritySI2     8.05151    0.29123  27.646   <2e-16 ***
## clarityVS1    12.87900    0.33757  38.152   <2e-16 ***
## clarityVS2    11.73159    0.32469  36.132   <2e-16 ***
## clarityVVS1   15.95943    0.37151  42.958   <2e-16 ***
## clarityVVS2   15.36371    0.35831  42.878   <2e-16 ***
## average_size  13.43248    0.31366  42.825   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 44696.6  on 32241  degrees of freedom
## Residual deviance:  9073.5  on 32220  degrees of freedom
## AIC: 9117.5
##
## Number of Fisher Scoring iterations: 8
```

```
# Predict on the test data
pred_logit <- predict(model_logit, newdata = test_data, type = "response")
# Convert probabilities to binary predictions (0 or 1)
pred_logit_class <- ifelse(pred_logit > 0.5, 1, 0)
# Confusion matrix for logistic regression
table(Predicted = pred_logit_class, Actual = test_data$expensive)
```

```
##          Actual
## Predicted     0     1
##         0 10072   621
##         1   628 10175
```

```
# Accuracy of the logistic model
accuracy_logit <- mean(pred_logit_class == test_data$expensive)
print(paste("Logistic Regression Accuracy:", accuracy_logit))
```

```
## [1] "Logistic Regression Accuracy: 0.941896166728694"
```

Interpretation:

The logistic regression model provides a good fit for predicting whether the price per carat is higher than the median. Most variables are statistically significant and contribute meaningfully to the prediction. Weplan to use logistic regression as the baseline to evaluate other models.

A p-value lower than 0.05 indicates that the variable is statistically significant. Most of the variables are highly significant (p-values<2e-16), meaning they strongly impact the classification of diamond prices. The coefficient of `table` is 0.03, indicating a slight increase in the odds of higher price per carat with an increase in

table. The coefficient of depthis 0.18, which is also 10x smaller than other coefficient magnitudes. Therefore, Wethink table and depth are the two less important variables when predicting the diamond is expensive or not.

4.3 Support Vector Machine

For this dataset, LDA gives worse accurancy than the logistic regression. The reason could be some of the LDA assumptions are not true in this dataset. Unlike LDA, SVM makes no assumptions about the distribution of the data.

Grid searching method is used to find a relative good SVM configurations. Among the selctions of cost (0.1,1,10) and gamma (0.1,1,10), cost = 10 and gamma = 0.1 can give us the best accuracy 0.95.

In the SVM model, 'C-classification' type and 'radial' kernal are specified. The features are scaled because they are of very different magnitudes. The hyperparameters are set with a cost of 10 and a gamma of 0.1. The accuracy of the SVM model is calculated and printed by comparing the predictions to the actual labels in the test data. The accuracy Weget from the initial SVM is 95%, which is slightly better than the logistic regression 94%.

```
library(e1071)
# Build a SVM model for classification model
model_svm <-svm(expensive ~ carat + depth + table + cut + color + clarity + average_size,
data=train_data, type='C-classification', kernel='radial', cost=10, gamma=0.1, scale=TRUE)
testPred <- predict(model_svm, test_data, type="response") # Predict on the test data
# Confusion matrix for logistic regression
table(Predicted = testPred, Actual = test_data$expensive)
```

```
##          Actual
## Predicted     0     1
##         0 10158   465
##         1   542 10331
```

```
accuracy_svm <- mean(testPred == test_data$expensive) # Accuracy
print(paste("SVM Classification Accuracy:", accuracy_svm))
```

```
## [1] "SVM Classification Accuracy: 0.953154075176777"
```

4.4 Decision Tree

This code trains a decision tree model to classify diamonds as "expensive" or not, using the variables carat, depth, table, cut, color, clarity, and average_size as predictors. The model is built on the train_data dataset using the rpart function with the method="class" option, which specifies a classification tree. The code then predicts the classifications for the test_data and calculates a confusion matrix (conf_matrix_tree) to compare the predicted labels to the actual labels in the test set. Finally, the accuracy of the model is computed by taking the mean of correct predictions and printed as the "Decision Tree Classification Accuracy."

```
library(rpart)
# Decision Tree Model
model_tree <- rpart(expensive ~ carat + depth + table + cut + color + clarity + average_size,
                    data=train_data, method="class")
# Predict on test data
```

```
testPred_tree <- predict(model_tree, test_data, type="class")
# Confusion Matrix for Decision Tree
conf_matrix_tree <- table(Predicted = testPred_tree, Actual = test_data$expensive)
print(conf_matrix_tree)
```

```
##          Actual
## Predicted     0     1
##         0  9195   596
##         1  1505 10200
```

```
accuracy_tree <- mean(testPred_tree == test_data$expensive)
print(paste("Decision Tree Classification Accuracy:", accuracy_tree))
```

```
## [1] "Decision Tree Classification Accuracy: 0.902260885746185"
```

4.5 Random Forest

To improve the accuracy of the decision tree model, we train a random forest model containing 100 trees. `randomForest` is used. importance=TRUE parameter allows for evaluating the significance of each predictor, helping to identify which features most influence the classification.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3
```

```
set.seed(123) # Set seed for reproducibility
train_data$expensive <- as.factor(train_data$expensive)
test_data$expensive <- as.factor(test_data$expensive)
model_randomForest <- randomForest(expensive ~ carat + depth + table + cut + color
+ clarity + average_size, data=train_data, ntree=100, mtry=3, importance=TRUE)
testPred_rf <- predict(model_randomForest, test_data) # Predict on test data
accuracy_rf <- mean(testPred_rf == test_data$expensive) # Calculate Accuracy
print(paste("Random Forest Classification Accuracy:", accuracy_rf))
```

```
## [1] "Random Forest Classification Accuracy: 0.954828805359137"
```

5. Interpretations

From the EDA, we gained insights into the distribution and relationships between variables. The correlation visualization in Appendix B4 informed our feature selection process. For instance, the high covariance values (0.99) between variables `x`, `y`, and `z` suggest a degree of redundancy among them. Additionally, the strong correlation between `price` and `carat` may hinder the discovery of relationships between `price` and other features.

The next step involved feature engineering and selection. We created new variables, such as `average_size` and `price_per_carat`, and derived a categorical response variable `expensive`, based on `price_per_carat`. In the updated pairwise plots, the covariance between `price_per_carat` and `carat` decreased to 0.79, compared to the original 0.93 covariance between `price` and `carat`. This reduction is encouraging, as it

may allow us to better uncover the contributions of other features to the price. Furthermore, the `expensive` variable is well-balanced, with an equal distribution of TRUE and FALSE values.

Using the categorical response `expensive`, we applied classification models to predict whether a diamond is expensive. Logistic regression achieved 94% accuracy, while SVM reached 95% accuracy. Grid search was used to optimize the hyperparameters for the SVM model. The logistic regression model revealed that `table` and `depth` were less important in predicting whether a diamond is expensive, as indicated by their higher p-values (0.03 and 0.18, respectively). The SVM model, with its higher accuracy, suggests that this approach might be promising for price prediction as well. Decision tree achieved 90% accuracy, while random forest reached 95% accuracy. Random forest is the most effective model for this classification task. We can also interpret the feature importance from the random forest.

5.1 Models Comparison

Best Overall Model: The random forest has the highest accuracy (95.48%), suggesting it is the most effective model for this classification task. It's robust and captures complex patterns, making it suitable for production use. Random forests reduce the risk of overfitting by averaging across multiple trees, and they capture non-linear relationships better than logistic regression and individual decision trees. They also provide insights into feature importance. However, random forests can be less interpretable than a single decision tree. The ensemble nature of the model also requires more computation, which may impact performance with very large datasets.

Second-Best: The SVM model is close to random forest in accuracy (95.32%) and can be a strong choice if interpretability is less of a concern. SVM is effective in handling high-dimensional data and complex boundaries. With kernel functions, it can capture non-linear relationships well. However, SVMs can be computationally expensive, especially with large datasets, and they are less interpretable than logistic regression.

Decision Tree: accuracy is 90.23%. It is highly interpretable, providing a clear decision-making path and insights into which features are most influential for each classification. However, it is prone to overfitting, which may reduce their performance on test data compared to more robust ensemble methods.

Logistic Regression: accuracy is 94.19%. It is easy to interpret and quick to train. It provides insights into the importance of individual variables through coefficients. However, it is limited in handling non-linear relationships and may underperform if the data contains complex interactions between features.

In summary, the random forest is the best model if accuracy is the priority, while logistic regression or a decision tree might be preferred if interpretability or computational efficiency are more important.

5.2 Strategies Suggested By Random Forest

Because decision tree can provide a clear decision-making path. We visualize the first tree in the random forest using `getTree()`.

```
# Replace 1 with the index of any tree within the range of `ntree`
single_tree <- getTree(model_randomForest, k = 1, labelVar = TRUE)

# Convert single_tree to a format suitable for plotting
library(partykit)
```

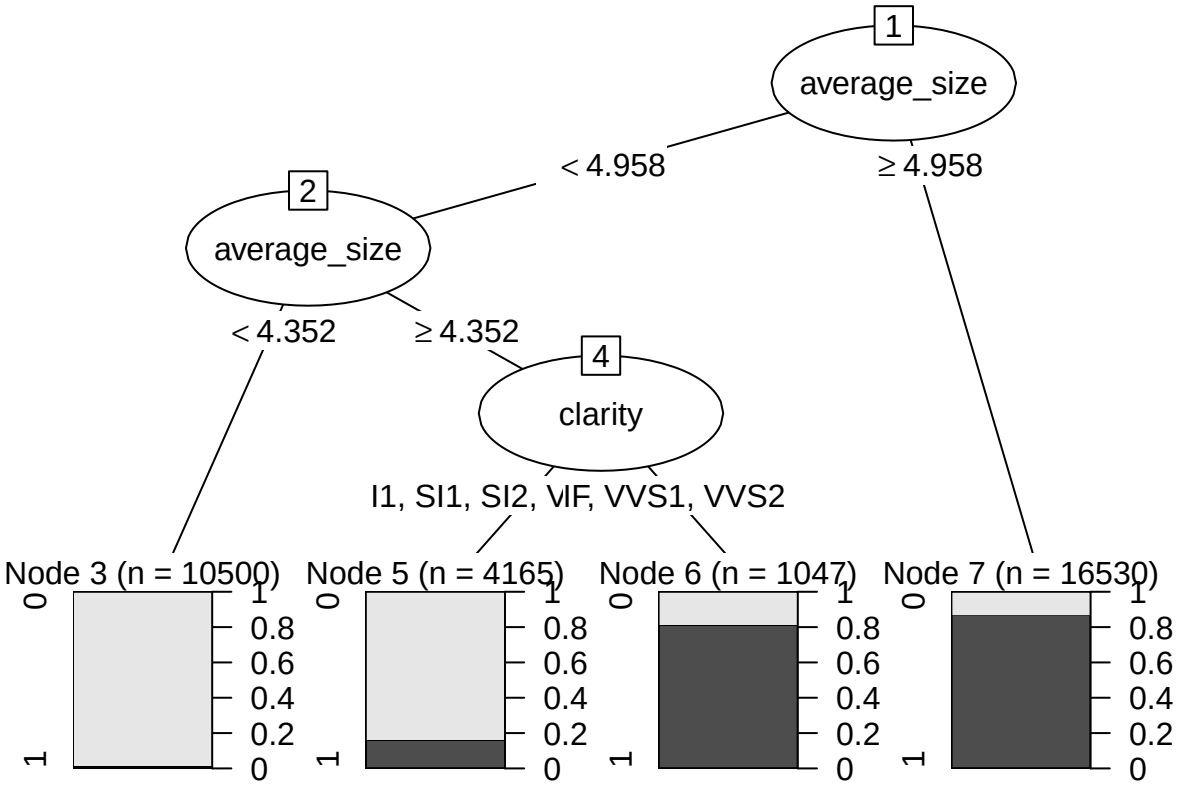
```
## Warning: package 'partykit' was built under R version 4.3.3
```

```
## Warning: package 'libcoin' was built under R version 4.3.3
```

```
## Warning: package 'mvtnorm' was built under R version 4.3.3
```

```
single_tree_party <- as.party(rpart(expensive ~ carat + depth + table + cut
                      + color + clarity + average_size, data = train_data))

plot(single_tree_party) # Plot the single tree
```



The decision tree visualization provides a clear, interpretable decision-making path based on the feature average_size and clarity.

Root Node (Node 1): The first decision point splits the data based on average_size < 4.958.

If average_size is less than 4.958, we move to the left child node (Node 2).

If average_size is greater than or equal to 4.958, we move to the right child node (Node 7).

Left Branch:

Node 2: Here, the data is further split based on average_size < 4.352.

If average_size is less than 4.352, we reach Node 3, which is a terminal node (leaf).

If average_size is greater than or equal to 4.352, we proceed to Node 4.

Node 4: This node splits the data based on clarity, categorizing diamonds into groups (I1, SI1, SI2 vs. VSIF, VVS1, VVS2).

If the diamond clarity is in the first group (I1, SI1, SI2), we reach Node 5 (a terminal node).

If the clarity is in the second group (VSIF, VVS1, VVS2), we reach Node 6 (another terminal node).

Right Branch:

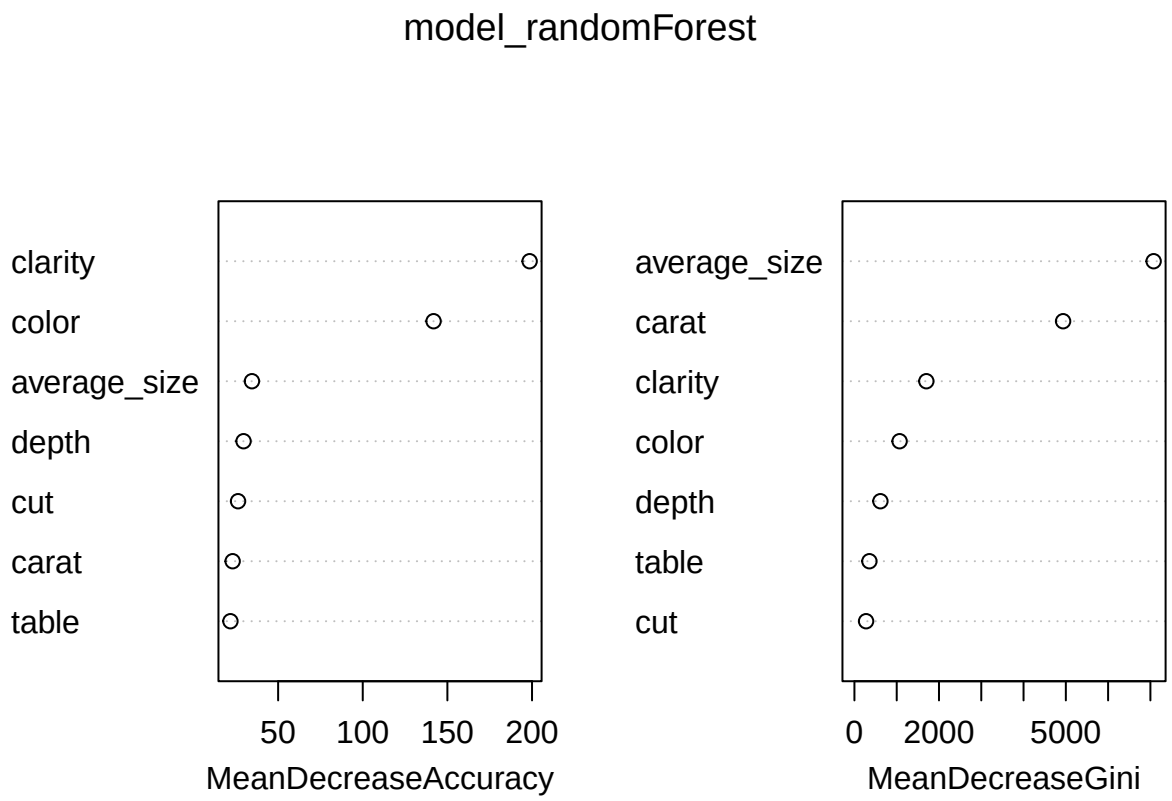
Node 7: This node represents cases where average_size is greater than or equal to 4.958. It is a terminal node, as there are no further splits.

Each terminal node (3, 5, 6, and 7) has a bar chart indicating the proportion of the class predictions at that node. The dark bar represents instances predicted as expensive, while the light bar represents instances predicted as not expensive.

This tree provides an interpretable decision rule that the model uses to classify diamonds based on average_size and clarity. This information can help in formulating a strategy where, for example, diamonds with larger average_size or higher clarity are more likely to be expensive.

To understand the general decision-making pattern across all trees, visualize feature importance, which shows the variables that most influence classification. We use `varImpPlot()` to display the 'Mean Decrease Accuracy' and 'Mean Decrease Gini' plots.

```
varImpPlot(model_randomForest) # Plot feature importance
```



Mean Decrease Accuracy (left plot): This indicates how much accuracy decreases when a given variable is excluded from the model. Higher values mean that the variable is more important for predictive accuracy. If removing a variable leads to a large drop in accuracy, it suggests that this variable plays an important role in distinguishing between the classes (expensive or not).

Mean Decrease Gini (right plot): This measures the total decrease in node impurity (or Gini impurity) that the variable contributes across all trees in the forest. Higher values mean the variable is more effective at splitting data into distinct classes. In other words, it captures the contribution of each variable to creating "pure" nodes where most samples belong to a single class.

Clarity and Average Size: These variables appear to be the most important features, as they have the highest values for both MeanDecreaseAccuracy and MeanDecreaseGini. This suggests that they are key features in predicting whether a diamond is expensive.

Color and Carat: These also have relatively high importance, indicating that they provide valuable information for classification.

Depth, Table, and Cut: These features show lower importance, meaning they contribute less to the model's accuracy and classification decisions.

Focus on High-Importance Features: Variables like clarity, average_size, and carat should be central to any decision-making strategy, as they significantly influence the model's predictions.

Lower-Importance Features May Be Optional: Features like table, depth, and cut are less critical and could potentially be excluded to simplify the model, as they contribute less to accuracy and purity.

6. Recommendations & Conclusions

For the data owner, this analysis provides valuable insights into predicting diamond prices based on key features. Our classification model achieves a high accuracy of at least 95% in identifying whether a diamond can be classified as expensive, supporting the effectiveness of these features in price categorization. To further improve price prediction, we recommend focusing on advanced regression models and exploring dimensionality reduction techniques such as Principal Component Analysis (PCA) to refine feature selection.

Our findings highlight that `carat`, `average_size`, `color`, and `clarity` significantly influence diamond pricing, whereas `cut`, `depth`, and `table` are comparatively less impactful. We suggest that the data owner prioritize these influential features when developing pricing strategies and consider deprioritizing the less critical ones to streamline analyses and enhance focus on the most impactful predictors.

Additionally, we recommend refining the definition of diamond size. Using x, y, and z dimensions introduces variability due to rotation, which may affect pricing accuracy. A standardized metric—such as diamond volume or surface area in combination with carat weight—may provide a more consistent measure. Including factors like shape and cut quality in this metric could further enhance the model's predictive accuracy.

Furthermore, we propose the following strategies to enrich the analysis:

1. **Geographical Market Data**: If available, incorporate market location information (e.g., region or market where the diamond is sold). This data could help capture regional pricing fluctuations and improve prediction accuracy.
2. **Temporal Analysis**: Adding time-related variables (such as the month or year of sale) could account for seasonal trends and market changes, making the model more adaptable to temporal variations.
3. **Price Elasticity**: Evaluate price elasticity by introducing data on comparable products or alternatives. Understanding the impact of changes in attributes (e.g., cut or clarity) on price sensitivity can help in constructing a more realistic and responsive pricing model.

In conclusion, our analysis identifies the core features that drive diamond pricing and provides a predictive framework for the data owner. By addressing variability in predicting prices for high-end diamonds and implementing these recommendations, the data owner can further improve pricing strategies and maximize predictive accuracy.

Our answer to the assignment questions:

1. What tools and techniques were used and why?

In our analysis, we used R for data processing, visualization, and model building due to its rich libraries. EDA was performed to understand the distribution and relationships between variables, using techniques like correlation matrices and pairwise plots to identify patterns and redundancies. For instance, the strong

covariance between `x`, `y`, and `z` indicated potential multicollinearity, while high correlation between `price` and `carat` guided us to focus on alternative variables. Feature engineering involved creating new variables such as `average_size` and `price_per_carat` to reduce multicollinearity and better capture the underlying relationships in the data. Classification techniques were then applied, including logistic regression and SVM, deciosion tree and random forest, selected for their performance and interpretability in the binary classification.

2. What were the results of the techniques? What were the new insights?

The EDA revealed key insights about the relationships between variables, such as the redundancy among dimensions `x`, `y`, and `z` and the dominance of `carat` in relation to `price`. By engineering features like `price_per_carat`, we reduced the covariance between `carat` and the new target variable to 0.79, compared to 0.93 with `price`, which allowed us to explore other features' contributions to diamond pricing. In terms of classification, both SVM and random forest models performed well, achieving 95.32% and 95.48% accuracy, respectively. The random Forest Model also provides insights into feature importance: Variables like clarity, average_size, and carat should be central to any decision-making strategy, as they significantly influence the model's predictions. Lower-Importance Features May Be Optional: Features like table, depth, and cut are less critical and could potentially be excluded to simplify the model, as they contribute less to accuracy and purity.

3. How did this help answer your questions? If your questions changed, why?

Our initial question was how to improve the prediction of diamond prices, and the analysis helped clarify that focusing on `price_per_carat` rather than `price` yielded better insights and predictions. The discovery that `price_per_carat` had less covariance with other features enabled us to better model relationships within the dataset, particularly through feature engineering. Additionally, the well-balanced `expensive` categorical variable allowed us to apply classification models effectively. The success of SVM and random forest models in classification confirmed that feature engineering improved our predictive power. As a result, our approach shifted towards optimizing the performance of models on the categorical variable `expensive` derived from `price_per_carat`, rather than continuing to work with `price` directly.

Appendix A: Dataset Information

Definition of all variables in the dataset (ranges are listed in the parenthesis):

- price: price in US dollars ($326–$18,823)
- carat: weight of the diamond (0.2–5.01)
- cut: quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- color: diamond color, from J (worst) to D (best)
- clarity: a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- x: length in mm (0–10.74)
- y: width in mm (0–58.9)
- z: depth in mm (0–31.8)
- depth: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- table: width of top of diamond relative to widest point (43–95)

Appendix B : Code Results Occupying Space

B1. Subsection 2.1: Variable types and statistics

```
# Print variable types and statistics
summary(diamonds)
```

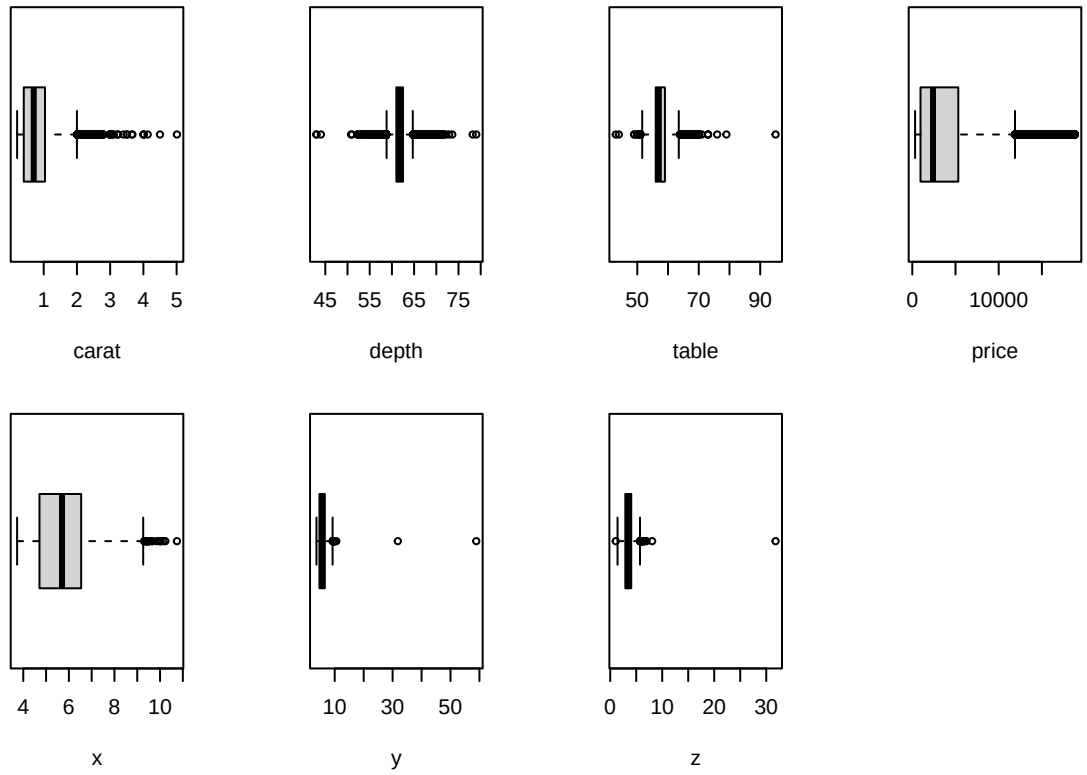
```
##      carat               cut        color        clarity          depth
##  Min.   :0.2000   Fair      : 1585   D: 6752   SI1    :13018   Min.   :50.80
##  1st Qu.:0.4000   Good      : 4889   E: 9765   VS2    :12220   1st Qu.:61.00
##  Median :0.7000   Ideal     :21475   F: 9511   SI2    : 9134   Median :61.80
##  Mean   :0.7975   Premium   :13725   G:11251   VS1    : 8147   Mean   :61.75
##  3rd Qu.:1.0400   Very Good:12064   H: 8256   VVS2   : 5054   3rd Qu.:62.50
##  Max.   :5.0100                     I: 5405   VVS1   : 3643   Max.   :79.00
##                                     J: 2798   (Other): 2522
##      table           price             x                y
##  Min.   :43.00   Min.   :  326   Min.   : 3.730   Min.   : 3.680
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710
##  Mean   :57.46   Mean   : 3931   Mean   : 5.731   Mean   : 5.734
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :10.540
##
##        z          average_size   price_per_carat   expensive
##  Min.   :2.240   Min.   :3.240   Min.   : 1051   Min.   :0.0
##  1st Qu.:2.910   1st Qu.:4.123   1st Qu.: 2478   1st Qu.:0.0
##  Median :3.530   Median :4.980   Median : 3495   Median :0.5
##  Mean   :3.539   Mean   :5.002   Mean   : 4009   Mean   :0.5
##  3rd Qu.:4.030   3rd Qu.:5.690   3rd Qu.: 4950   3rd Qu.:1.0
##  Max.   :6.980   Max.   :9.420   Max.   :17829   Max.   :1.0
##
```

```
str(diamonds)
```

```
## 'data.frame':    53738 obs. of  13 variables:
##  $ carat          : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut            : Factor w/ 5 levels "Fair","Good",..: 3 4 2 4 2 5 5 5 1 5 ...
##  $ color          : Factor w/ 7 levels "D","E","F","G",..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity        : Factor w/ 8 levels "I1","IF","SI1",..: 4 3 5 6 4 8 7 3 6 5 ...
##  $ depth          : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table          : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price          : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x              : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y              : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z              : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
##  $ average_size   : num  3.45 3.35 3.48 3.69 3.81 ...
##  $ price_per_carat: num  1417 1552 1422 1152 1081 ...
##  $ expensive      : num  0 0 0 0 0 0 0 0 0 0 ...
```

B2. Subsection 2.1: Boxplots of 7 numerical variables to find outliers

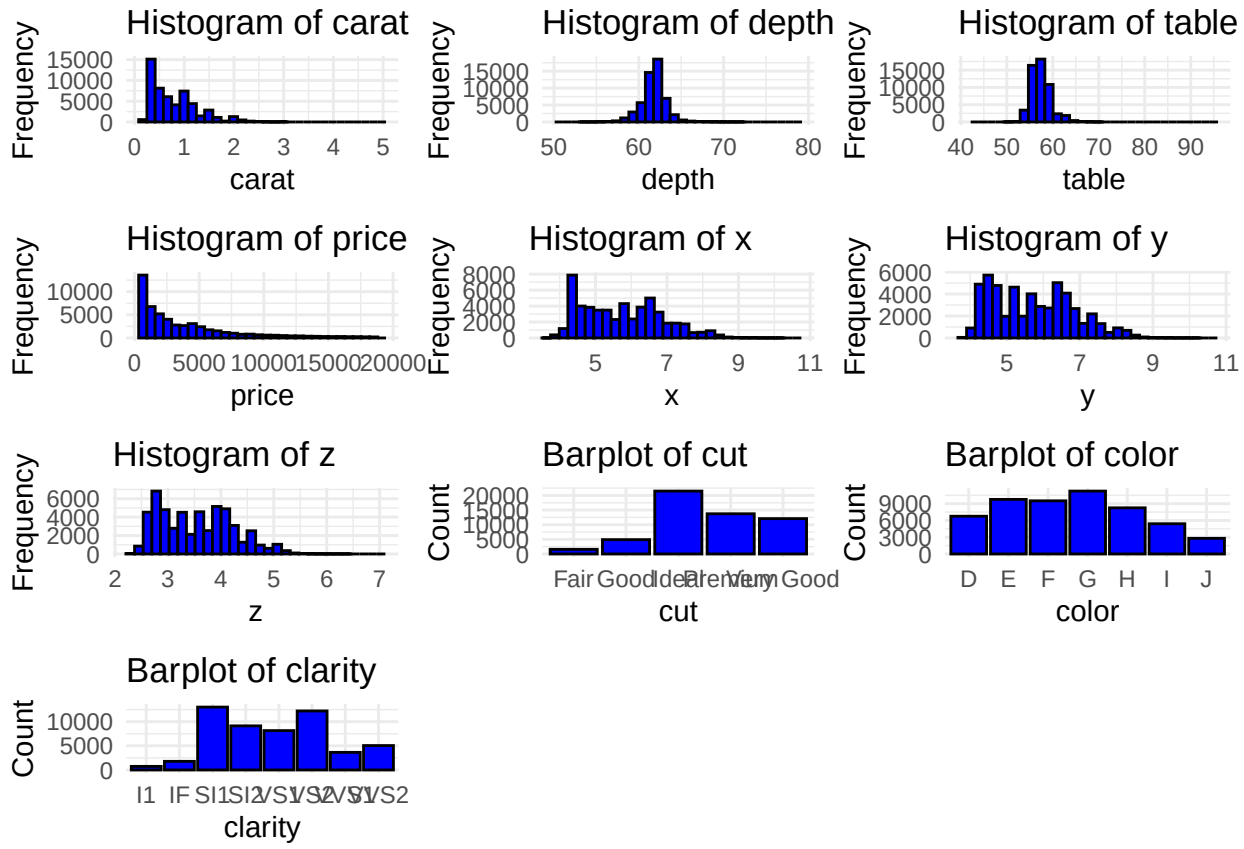
```
numeric_data <- diamonds_clean[,sapply(diamonds_clean,is.numeric)]
par(mfrow = c(2, 4), mar = c(4, 4, 2, 1), oma = c(1, 1, 1, 1))
for (i in 1:7){boxplot(numeric_data[, i],
                       xlab=colnames(numeric_data)[i],horizontal = TRUE)}
```



B3. Subsection 2.2: Histograms and bar plots

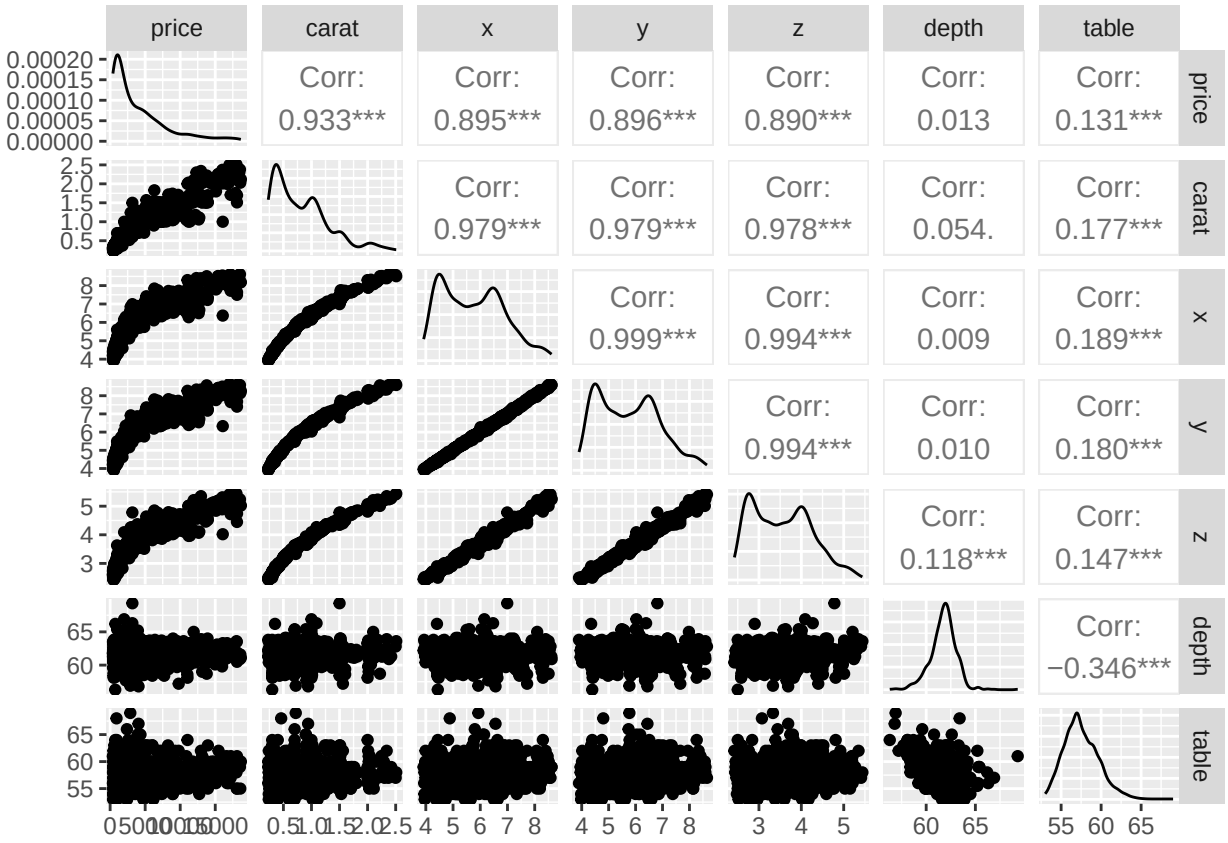
```
# Load necessary libraries
library(ggplot2)
library(gridExtra)
# Separate numeric and categorical variables
numeric_vars <- names(diamonds_cleaned)[sapply(diamonds_cleaned, is.numeric)]
categorical_vars <- names(diamonds_cleaned)[sapply(diamonds_cleaned, is.factor)
                    | sapply(diamonds_cleaned, is.character)]
# Create an empty list to store plots
plot_list <- list()
# Visualize numerical variables using histograms
for (var in numeric_vars) {
  p <- ggplot(diamonds_cleaned, aes(x = .data[[var]])) +
    geom_histogram(fill = "blue", color = "black") +
```

```
    ggtitle(paste("Histogram of", var)) +
    theme_minimal() + xlab(var) + ylab("Frequency")
  plot_list[[length(plot_list) + 1]] <- p
}
# Visualize categorical variables using bar plots
for (var in categorical_vars) {
  p <- ggplot(diamonds_cleaned, aes(x = .data[[var]])) +
    geom_bar(fill = "blue", color = "black") +
    ggtitle(paste("Barplot of", var)) +
    theme_minimal() + xlab(var) + ylab("Count")
  plot_list[[length(plot_list) + 1]] <- p
}
# Arrange the plots in a 2 by 5 grid
grid.arrange(grobs = plot_list, ncol = 3, nrow = 4)
```



B4. Subsection 2.3: Correlation matrix {#appendix-b4}

```
# Select the right-skewed and related variables
selected_vars <- diamonds_cleaned[, c("price", "carat", "x", "y", "z", "depth", "table")]
# Visualize pairwise relationships between the selected variables
ggpairs(selected_vars[sample(53738, 1000), ])
```


price
carat
x
y
z
depth
table
Corr: 0.933***
Corr: 0.895***
Corr: 0.896***
Corr: 0.890***
Corr: 0.013
Corr: 0.131***
Corr: 0.979***
Corr: 0.979***
Corr: 0.978***
Corr: 0.054.
Corr: 0.177***
Corr: 0.999***
Corr: 0.994***
Corr: 0.009
Corr: 0.189***
Corr: 0.994***
Corr: 0.010
Corr: 0.180***
Corr: 0.118***
Corr: 0.147***
Corr: −0.346***