Statistical Theory and Scientific Computing -I

Final Project

Diamond Price Prediction

Submitted by:

Nawaraj Adhikari nxa6298@mavs.uta.edu Student ID: 1002166298

Patel Manan Champakbhai mxp8783@mavs.uta.edu Student Id:1002228783

Submitted to:

Mei Yang, PhD

Instructor

Division of Data Science, College of Science

For the fulfillment of the requirement for project work of 2248-ASDS-5303-003 – Statistical Theory and Scientific Computing-I

The University of Texas at Arlington

Arlington, TX, USA

Abstract

The Diamond Price Prediction project aims to develop a robust machine learning model to accurately predict the price of diamonds based on their physical and categorical attributes. Utilizing a dataset containing detailed information about diamonds, including measurements (length, width, and depth), weight (carat), and quality characteristics (cut, clarity, color, polish, symmetry, and fluorescence), this project employs a comprehensive data preprocessing pipeline, exploratory data analysis (EDA), and multiple predictive modeling techniques.

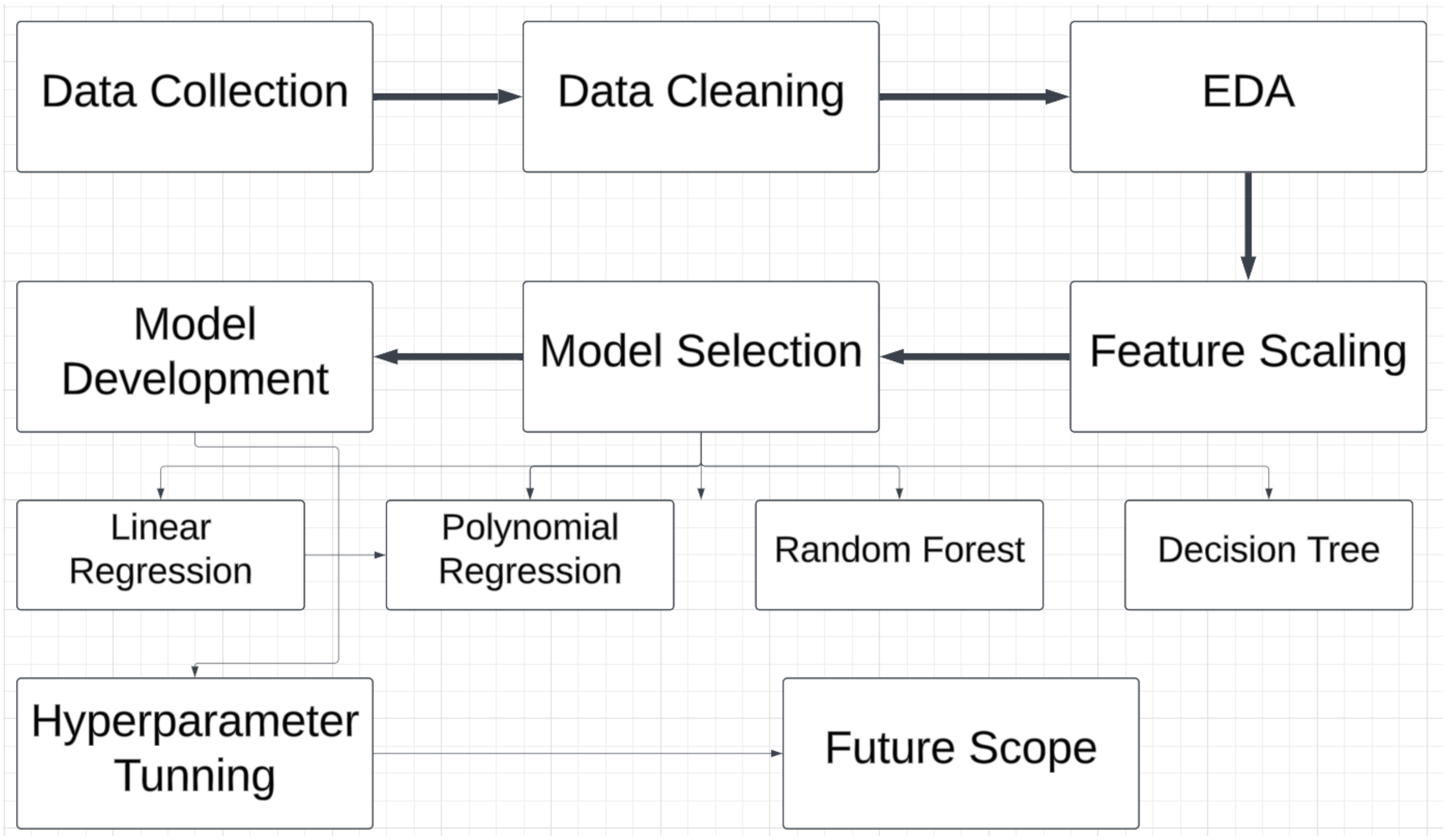
The dataset underwent extensive preprocessing to handle missing values, outliers, and inconsistencies. The Measurements column was split into Length, Width, and Depth, categorical variables were encoded, and numerical features were standardized. Exploratory data analysis revealed that price had weak linear relationships with most numerical features, but categorical attributes such as clarity, cut, and symmetry showed a stronger correlation with price. Interaction terms and polynomial features were generated to capture non-linear relationships and improve predictive power.

Several machine learning models were implemented and evaluated, including linear regression, Lasso and Ridge regression, polynomial regression, decision trees, and random forest. Performance metrics such as R-squared, Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) were used to compare model effectiveness. Among these models, the Random Forest algorithm outperformed others with the highest prediction accuracy, leveraging its ability to handle complex, non-linear relationships in the data.

The findings of this project demonstrate that diamond pricing is influenced by a combination of physical and qualitative attributes. While numerical features alone provide limited insights, incorporating categorical attributes and their interactions significantly enhances prediction accuracy. This model can serve as a valuable tool for diamond appraisers, sellers, and buyers by providing data-driven insights into diamond valuation.

Future work includes exploring advanced ensemble models such as Gradient Boosting Machines (GBM) or XGBoost, optimizing hyperparameters, and incorporating additional features like market trends or supply-chain factors. The project underscores the potential of machine learning to revolutionize traditional valuation methods in the gemstone industry.

Workflow of the Project



This flowchart outlines the steps involved in the Diamond Price Prediction project. The process begins with Data Collection and proceeds to Data Cleaning, where inconsistencies and missing values are addressed. Exploratory Data Analysis (EDA) provides insights into the data's structure, followed by Feature Scaling to standardize the numerical attributes. The pipeline then transitions to Model Development, exploring methods such as Linear Regression, Polynomial Regression, Random Forest, and Decision Tree. Models are selected based on performance, and Hyperparameter Tuning is conducted to optimize results. Finally, Future Scope highlights potential enhancements like advanced algorithms and additional features for improved predictions.

Required Libraries

Here is a list of all the libraries used in this **Diamond Price Prediction** project:

Data Manipulation and Cleaning

1. readr: For reading and importing the dataset.
2. dplyr: For data manipulation, filtering, and selecting columns.
3. tidyr: For splitting and restructuring columns.

Exploratory Data Analysis (EDA)

4. ggplot2: For creating visualizations like scatter plots, line plots, and histograms.
5. reshape2: For reshaping data and creating correlation heatmaps.

Data Preprocessing

6. scales: For scaling numerical data.
7. fastDummies: For one-hot encoding categorical variables.

Model Development

8. caret: For splitting datasets and building machine learning models.
9. caTools: For creating train-test splits.
10. glmnet: For Lasso and Ridge regression modeling.
11. rpart: For building decision tree models.
12. randomForest: For building random forest models.

Model Evaluation and Visualization

13. Metrics: For calculating error metrics like MAE, MSE, RMSE, and MAPE.
14. corrplot: For creating correlation matrix heatmaps.
15. patchwork: For combining multiple plots into a single visualization.
16. gridExtra: For arranging multiple visualizations side-by-side.

Dataset Collection

The data for this project was obtained directly from Shree Ramkrishna Exports Pvt. Ltd. (SRK), one of India’s largest and most reputable diamond manufacturers and exporters, based in Surat, Gujarat. SRK is globally recognized for its excellence in diamond cutting and grading, ensuring the authenticity and reliability of the dataset.

The dataset comprises real-world information on diamonds, including:

- Physical Attributes: Measurements such as Length, Width, Depth, and Weight (in carats).
- Quality Characteristics: Grading features like Cut, Clarity, Color, Polish, Symmetry, and Fluorescence.
- Market Value: Price of diamonds in the current market.

This dataset is unique and comprehensive, offering an authentic representation of the diamond industry. By sourcing the data from SRK, we ensure that the features are aligned with industry standards, making the predictions and analysis applicable to real-world use cases. The insights generated from this data aim to provide significant value to buyers, sellers, and appraisers in the diamond market.

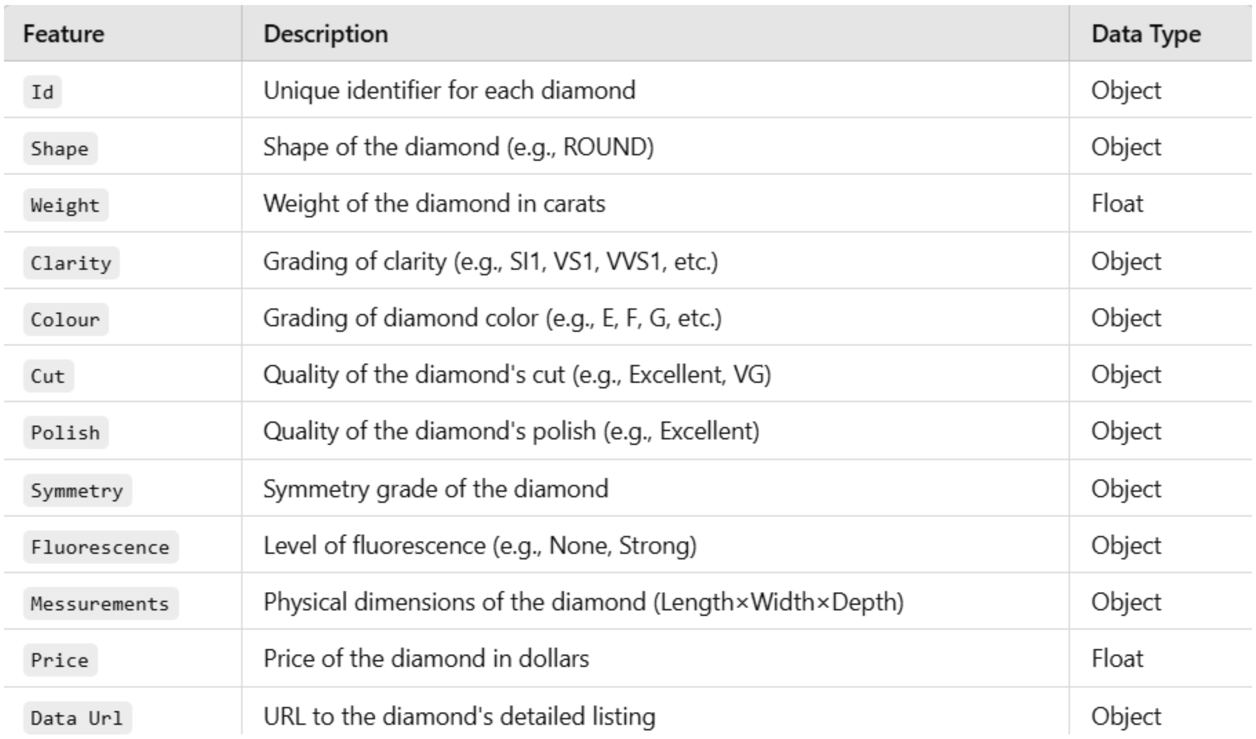
The variety and depth of the dataset make it well-suited for analyzing the relationships between a diamond's characteristics and its price. It also provides a strong foundation for developing predictive models that can help streamline valuation processes. This collaboration with SRK adds credibility to the project, showcasing its potential for practical implementation in the diamond trade while ensuring that insights remain directly applicable to the industry's needs.

Link : https://srk.one/

Dataset Overview

The dataset used in this project comprises 2,141 records and 12 features. The data is sourced from **Shree Ramkrishna Exports Pvt. Ltd. (SRK)**, ensuring high-quality and authentic information. Below is a summary of the dataset:

- **Shape**: (2141, 12) - 2,141 rows and 12 columns.
- **Key Features**: Attributes include physical measurements (e.g., Length, Width, Depth), categorical quality grades (e.g., Clarity, Color, Cut), and the Price of diamonds.
- **Target Variable**: Price
- **Data Integrity**: No missing values were observed in the dataset, making it suitable for direct analysis after initial cleaning.
- **Unique Characteristics**:
    - Diamonds are exclusively of the **round** shape.
    - The prices range from **655.38 to 951.72**.
    - Clarity, Color, and other categorical features have multiple unique grades.

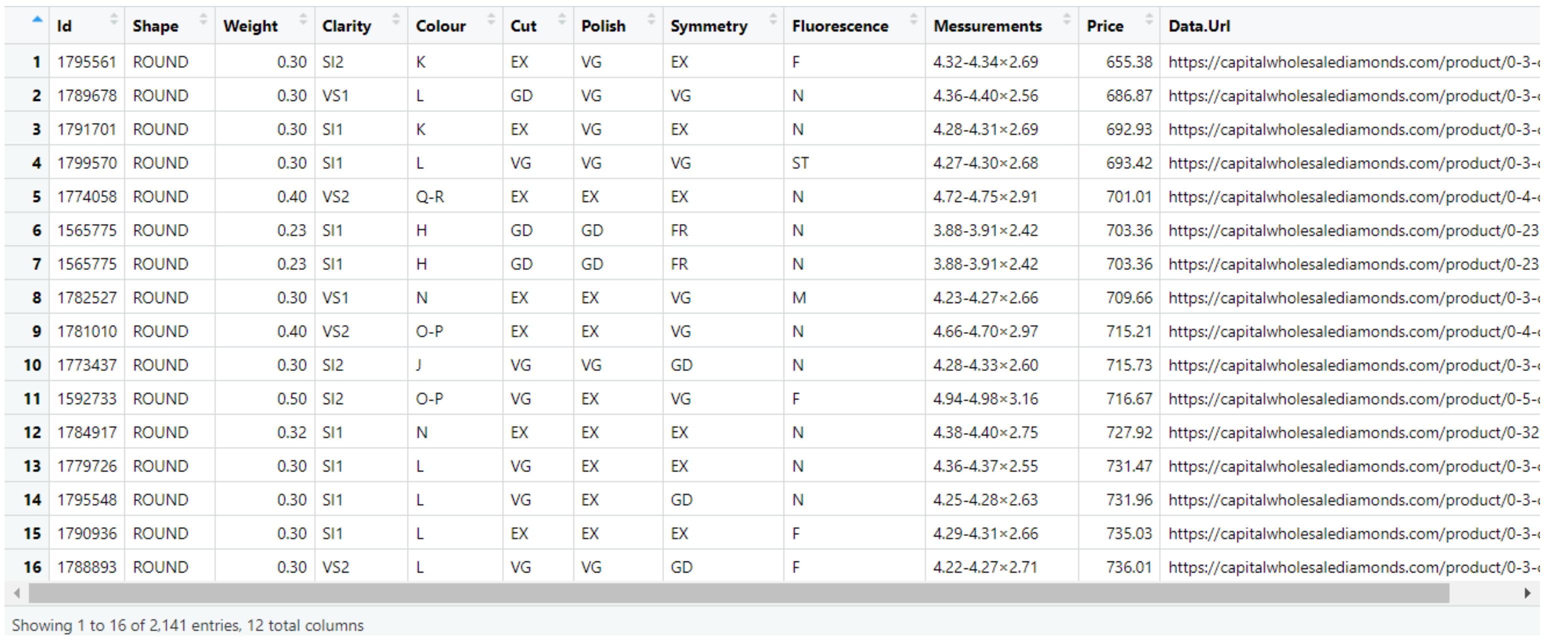


Feature	Description	Data Type
`Id`	Unique identifier for each diamond	Object
`Shape`	Shape of the diamond (e.g., ROUND)	Object
`Weight`	Weight of the diamond in carats	Float
`Clarity`	Grading of clarity (e.g., SI1, VS1, VVS1, etc.)	Object
`Colour`	Grading of diamond color (e.g., E, F, G, etc.)	Object
`Cut`	Quality of the diamond's cut (e.g., Excellent, VG)	Object
`Polish`	Quality of the diamond's polish (e.g., Excellent)	Object
`Symmetry`	Symmetry grade of the diamond	Object
`Fluorescence`	Level of fluorescence (e.g., None, Strong)	Object
`Messurements`	Physical dimensions of the diamond (Length×Width×Depth)	Object
`Price`	Price of the diamond in dollars	Float
`Data Url`	URL to the diamond's detailed listing	Object

- Now Let’s import this dataset:

```
# Importing the Data-set

library(readr)
df = read.csv("C:/Users/manan/Downloads/Semester 1/data.csv")
View(df)
```

Output:



	Id	Shape	Weight	Clarity	Colour	Cut	Polish	Symmetry	Fluorescence	Messurements	Price	Data.Url
1	1795561	ROUND	0.30	SI2	K	EX	VG	EX	F	4.32-4.34×2.69	655.38	https://capitalwholesalediamonds.com/product/0-3-
2	1789678	ROUND	0.30	VS1	L	GD	VG	VG	N	4.36-4.40×2.56	686.87	https://capitalwholesalediamonds.com/product/0-3-
3	1791701	ROUND	0.30	SI1	K	EX	VG	EX	N	4.28-4.31×2.69	692.93	https://capitalwholesalediamonds.com/product/0-3-
4	1799570	ROUND	0.30	SI1	L	VG	VG	VG	ST	4.27-4.30×2.68	693.42	https://capitalwholesalediamonds.com/product/0-3-
5	1774058	ROUND	0.40	VS2	Q-R	EX	EX	EX	N	4.72-4.75×2.91	701.01	https://capitalwholesalediamonds.com/product/0-4-
6	1565775	ROUND	0.23	SI1	H	GD	GD	FR	N	3.88-3.91×2.42	703.36	https://capitalwholesalediamonds.com/product/0-23
7	1565775	ROUND	0.23	SI1	H	GD	GD	FR	N	3.88-3.91×2.42	703.36	https://capitalwholesalediamonds.com/product/0-23
8	1782527	ROUND	0.30	VS1	N	EX	EX	VG	M	4.23-4.27×2.66	709.66	https://capitalwholesalediamonds.com/product/0-3-
9	1781010	ROUND	0.40	VS2	O-P	EX	EX	VG	N	4.66-4.70×2.97	715.21	https://capitalwholesalediamonds.com/product/0-4-
10	1773437	ROUND	0.30	SI2	J	VG	VG	GD	N	4.28-4.33×2.60	715.73	https://capitalwholesalediamonds.com/product/0-3-
11	1592733	ROUND	0.50	SI2	O-P	VG	EX	VG	F	4.94-4.98×3.16	716.67	https://capitalwholesalediamonds.com/product/0-5-
12	1784917	ROUND	0.32	SI1	N	EX	EX	EX	N	4.38-4.40×2.75	727.92	https://capitalwholesalediamonds.com/product/0-32
13	1779726	ROUND	0.30	SI1	L	VG	EX	EX	N	4.36-4.37×2.55	731.47	https://capitalwholesalediamonds.com/product/0-3-
14	1795548	ROUND	0.30	SI1	L	VG	EX	GD	N	4.25-4.28×2.63	731.96	https://capitalwholesalediamonds.com/product/0-3-
15	1790936	ROUND	0.30	SI1	L	EX	EX	EX	F	4.29-4.31×2.66	735.03	https://capitalwholesalediamonds.com/product/0-3-
16	1788893	ROUND	0.30	VS2	L	VG	VG	GD	F	4.22-4.27×2.71	736.01	https://capitalwholesalediamonds.com/product/0-3-

Showing 1 to 16 of 2,141 entries, 12 total columns

Here we can see that we have 2141 observation and 12 features.

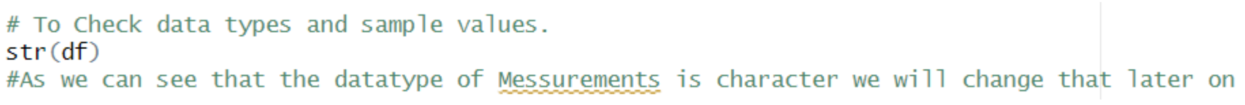
- Now, Let’s check the shape of our dataset using R:

```
#Now let's Check the shape of the Data-set
dim(df)
```

Output:

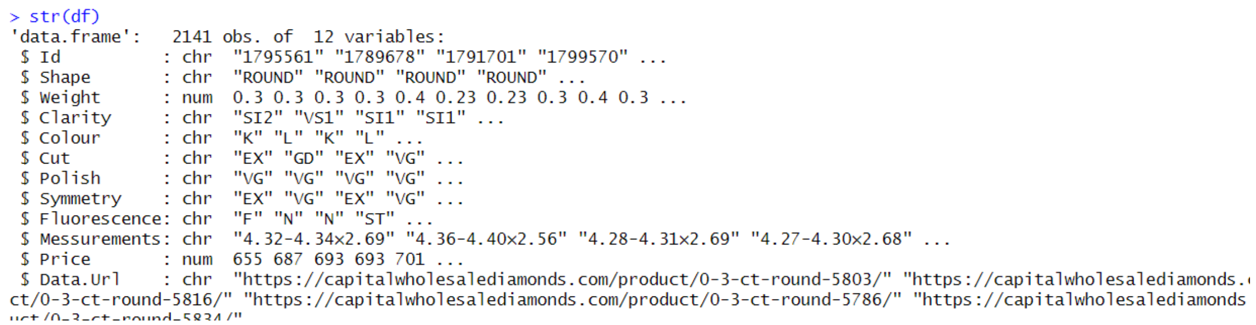
```
> #Now let's Check the shape of the Data-set
> dim(df)
[1] 2141   12
```

- To check data types and sample values of a feature



```
# To Check data types and sample values.
str(df)
#As we can see that the datatype of Messurements is character we will change that later on
```

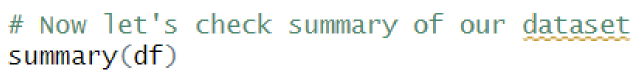
Output:



```
> str(df)
'data.frame':	2141 obs. of  12 variables:
 $ Id          : chr  "1795561" "1789678" "1791701" "1799570" ...
 $ Shape       : chr  "ROUND" "ROUND" "ROUND" "ROUND" ...
 $ Weight      : num  0.3 0.3 0.3 0.3 0.4 0.23 0.23 0.3 0.4 0.3 ...
 $ Clarity     : chr  "SI2" "VS1" "SI1" "SI1" ...
 $ Colour      : chr  "K" "L" "K" "L" ...
 $ Cut         : chr  "EX" "GD" "EX" "VG" ...
 $ Polish      : chr  "VG" "VG" "VG" "VG" ...
 $ Symmetry    : chr  "EX" "VG" "EX" "VG" ...
 $ Fluorescence: chr  "F" "N" "N" "ST" ...
 $ Messurements: chr  "4.32-4.34x2.69" "4.36-4.40x2.56" "4.28-4.31x2.69" "4.27-4.30x2.68" ...
 $ Price       : num  655 687 693 693 701 ...
 $ Data.Url    : chr  "https://capitalwholesalediamonds.com/product/0-3-ct-round-5803/" "https://capitalwholesalediamonds.
ct/0-3-ct-round-5816/" "https://capitalwholesalediamonds.com/product/0-3-ct-round-5786/" "https://capitalwholesalediamonds
uct/0-3-ct-round-5834/"
```

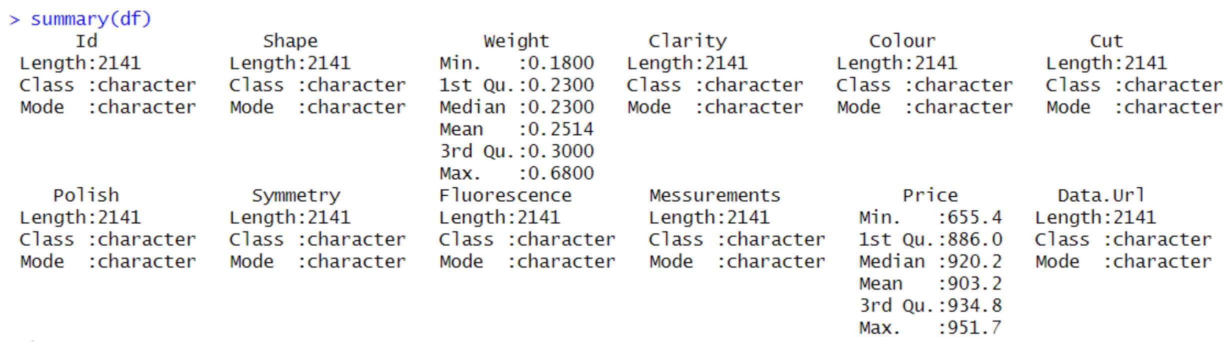
This output displays the structure of the dataset (str(df)), which consists of 2,141 observations and 12 variables. The dataset includes a mix of categorical (chr for features like Shape, Clarity, Cut) and numerical (num for Weight, Price) columns. Key features include measurements (Messurements), quality attributes (Clarity, Cut, Colour), and the target variable Price.

- Checking the summary of our dataset



```
# Now let's check summary of our dataset
summary(df)
```

Output



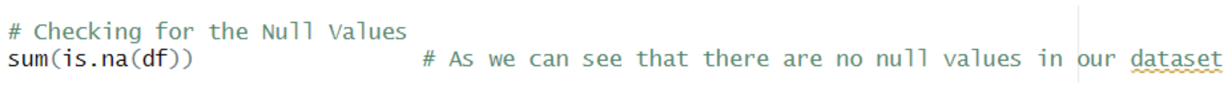
```
> summary(df)
      Id               Shape               Weight          Clarity             Colour               Cut           
 Length:2141        Length:2141        Min.   :0.1800   Length:2141        Length:2141        Length:2141       
 Class :character   Class :character   1st Qu.:0.2300   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Median :0.2300   Mode  :character   Mode  :character   Mode  :character  
                                       Mean   :0.2514                                                           
                                       3rd Qu.:0.3000                                                           
                                       Max.   :0.6800                                                           
    Polish            Symmetry         Fluorescence       Messurements           Price          Data.Url        
 Length:2141        Length:2141        Length:2141        Length:2141        Min.   :655.4   Length:2141       
 Class :character   Class :character   Class :character   Class :character   1st Qu.:886.0   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Median :920.2   Mode  :character  
                                                                             Mean   :903.2                     
                                                                             3rd Qu.:934.8                     
                                                                             Max.   :951.7                     
```

Data Cleaning

Data cleaning is a crucial step in ensuring the dataset is prepared for analysis and modeling. Below is a detailed explanation of the steps undertaken to clean and preprocess the diamond dataset:

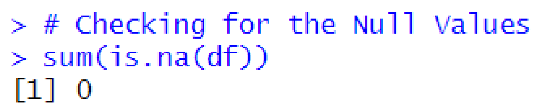
1. Handling Missing Values

- There were no NA values in the dataset (sum(is.na(df)) == 0).
- However, for certain columns with invalid entries, corrections were made during the cleaning process.
- The dataset was confirmed to be free of missing or null values post-cleaning.



```
# Checking for the Null Values
sum(is.na(df))                  # As we can see that there are no null values in our dataset
```

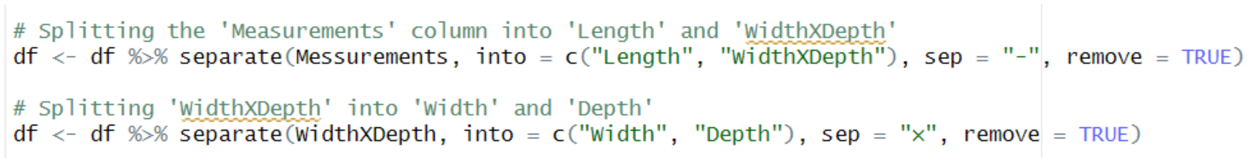
Output:



```
> # Checking for the Null Values
> sum(is.na(df))
[1] 0
```

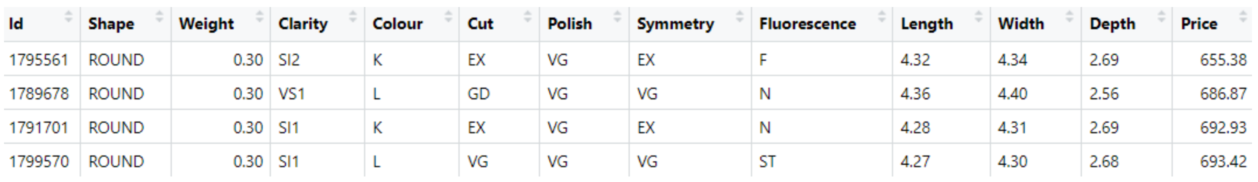
2. Handling the Measurements Column

- The Measurements column contained combined dimensions of diamonds in the format Length-Width×Depth.
- **Steps Taken**:
  a. The column was split into three separate columns: Length, Width, and Depth, using the separate() function from the tidyr library.
  b. This improved data granularity and allowed numerical operations on these features.



```
# Splitting the 'Measurements' column into 'Length' and 'WidthXDepth'
df <- df %>% separate(Messurements, into = c("Length", "WidthXDepth"), sep = "-", remove = TRUE)

# Splitting 'WidthXDepth' into 'Width' and 'Depth'
df <- df %>% separate(WidthXDepth, into = c("Width", "Depth"), sep = "x", remove = TRUE)
```

Output:



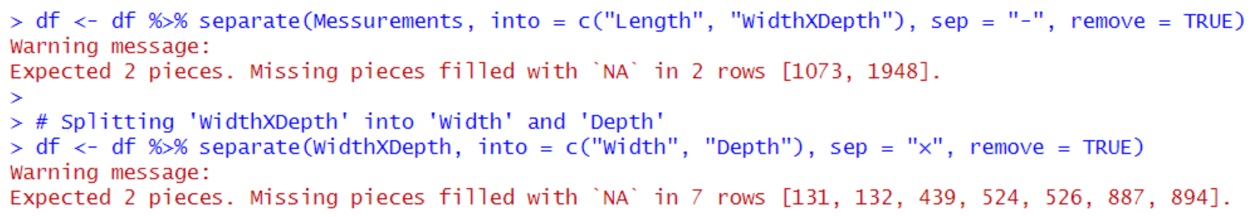
Id	Shape	Weight	Clarity	Colour	Cut	Polish	Symmetry	Fluorescence	Length	Width	Depth	Price
1795561	ROUND	0.30	SI2	K	EX	VG	EX	F	4.32	4.34	2.69	655.38
1789678	ROUND	0.30	VS1	L	GD	VG	VG	N	4.36	4.40	2.56	686.87
1791701	ROUND	0.30	SI1	K	EX	VG	EX	N	4.28	4.31	2.69	692.93
1799570	ROUND	0.30	SI1	L	VG	VG	VG	ST	4.27	4.30	2.68	693.42

The Measurements column in the original dataset, which combined the diamond's dimensions (e.g., Length-Width×Depth), was split into three separate numerical columns: Length, Width, and Depth. This transformation allowed for more detailed analysis and numerical operations, improving the dataset's usability for modeling and exploratory analysis.

3. Handling Inconsistent Entries

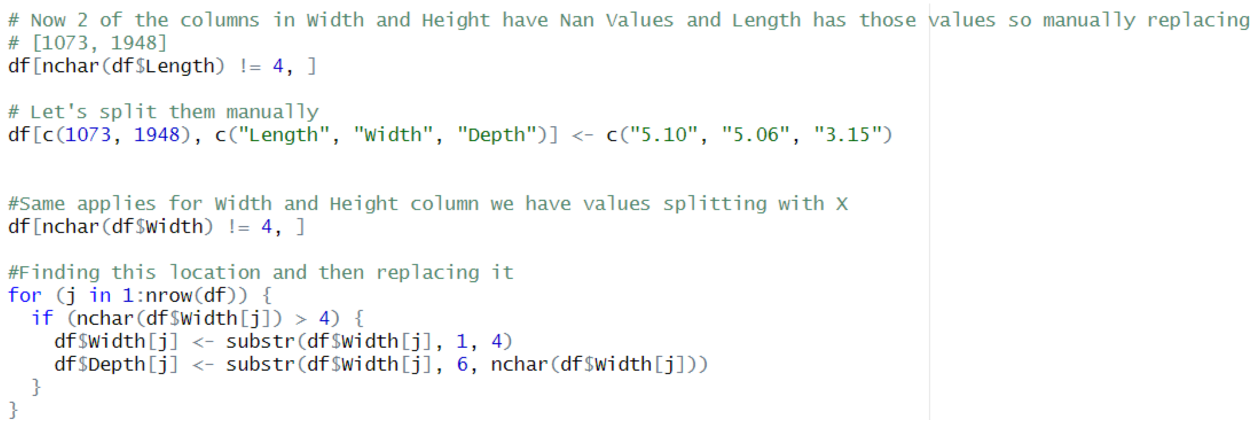
- During data validation, some entries in the Width and Depth columns contained invalid or inconsistent values (e.g., split incorrectly or exceeding expected character length).
- **Steps Taken**:
    a. Invalid entries were manually identified using string length checks.

    b. Corrections were applied by parsing valid parts of the strings or using the values from the Length.



```
> df <- df %>% separate(Messurements, into = c("Length", "WidthXDepth"), sep = "-", remove = TRUE)
Warning message:
Expected 2 pieces. Missing pieces filled with `NA` in 2 rows [1073, 1948].
>
> # Splitting 'WidthXDepth' into 'Width' and 'Depth'
> df <- df %>% separate(WidthXDepth, into = c("Width", "Depth"), sep = "x", remove = TRUE)
Warning message:
Expected 2 pieces. Missing pieces filled with `NA` in 7 rows [131, 132, 439, 524, 526, 887, 894].
```

These were the rows which had a missing value while splitting. Now let's fill it in manually.



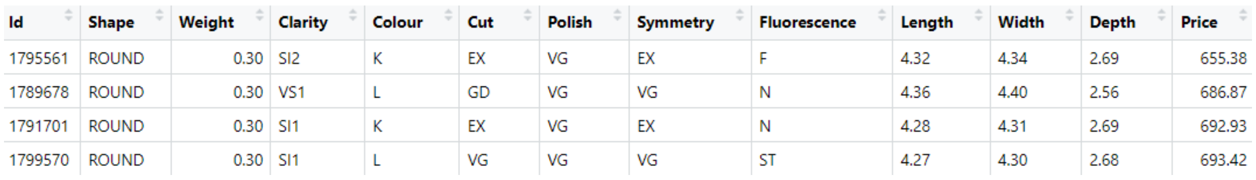
```
# Now 2 of the columns in Width and Height have Nan Values and Length has those values so manually replacing
# [1073, 1948]
df[nchar(df$Length) != 4, ]

# Let's split them manually
df[c(1073, 1948), c("Length", "Width", "Depth")] <- c("5.10", "5.06", "3.15")


#Same applies for Width and Height column we have values splitting with X
df[nchar(df$Width) != 4, ]

#Finding this location and then replacing it
for (j in 1:nrow(df)) {
  if (nchar(df$Width[j]) > 4) {
    df$Width[j] <- substr(df$Width[j], 1, 4)
    df$Depth[j] <- substr(df$Width[j], 6, nchar(df$Width[j]))
  }
}
```

4. **Removing Irrelevant Columns**

- **Columns Removed**:
    a. Data.Url: Contained URLs that were irrelevant for prediction or analysis.
    b. Shape: Since all diamonds in the dataset were of the "ROUND" shape, the column was redundant and dropped after EDA.

```
# Dropping the 'Data Url' column
df <- df %>% select(-Data.Url)
```

Output:



Id	Shape	Weight	Clarity	Colour	Cut	Polish	Symmetry	Fluorescence	Length	Width	Depth	Price
1795561	ROUND	0.30	SI2	K	EX	VG	EX	F	4.32	4.34	2.69	655.38
1789678	ROUND	0.30	VS1	L	GD	VG	VG	N	4.36	4.40	2.56	686.87
1791701	ROUND	0.30	SI1	K	EX	VG	EX	N	4.28	4.31	2.69	692.93
1799570	ROUND	0.30	SI1	L	VG	VG	VG	ST	4.27	4.30	2.68	693.42

5. **Converting Data Types**

- Some numerical columns (e.g., Length, Width, Depth) were stored as characters (chr) instead of numerical types.
- **Steps Taken**:
    - These columns were converted to numeric using the as.numeric() function to ensure compatibility with statistical and machine learning methods.

```
# Converting Datatype of Length, Width and Height to float
# Convert 'Length', 'Width', and 'Depth' columns to numeric type
df$Length <- as.numeric(df$Length)
df$Width <- as.numeric(df$Width)
df$Depth <- as.numeric(df$Depth)
```

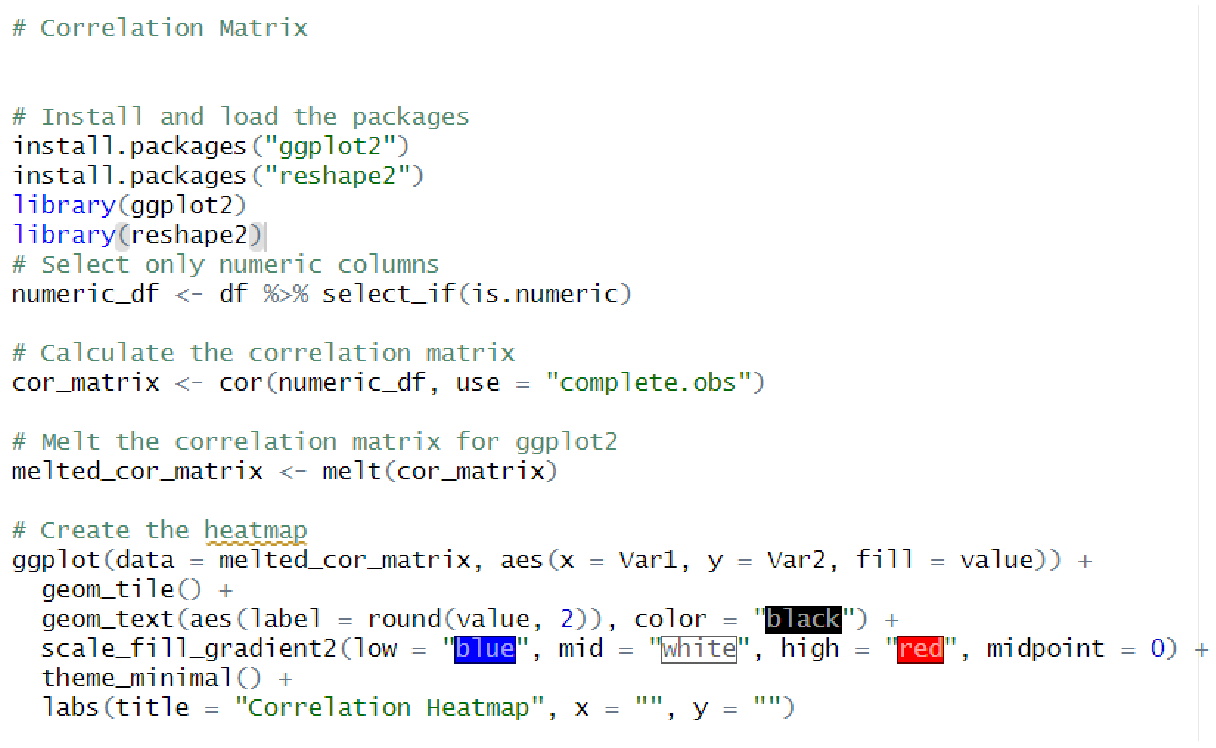
The Length, Width, and Depth columns, initially stored as character data types, were converted to numeric format using the as.numeric() function. This conversion ensured that these columns could be used for numerical operations, such as scaling, correlation analysis, and modeling. It also improved the data's compatibility with machine learning algorithms.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was performed to understand the dataset's structure, discover patterns, and identify relationships between features. Below is a detailed explanation of the EDA steps undertaken in the project:

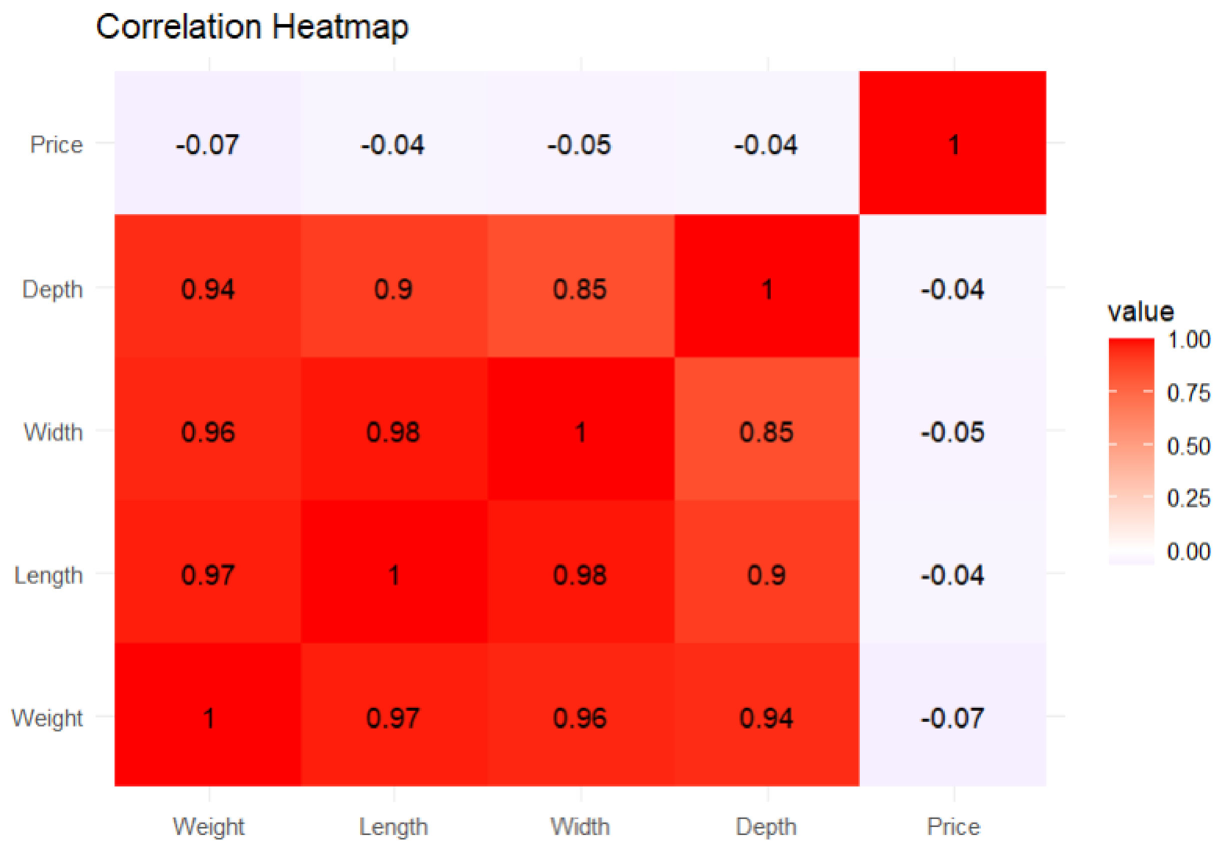
1. Correlation Analysis

- **Objective**: To understand relationships between numerical features and identify multicollinearity.
- **Steps Taken**:
  - A correlation matrix was calculated for numerical features, and a heatmap was plotted using ggplot2 and reshape2.
  - Insights:
    - Strong correlations were observed between Length, Width, and Depth.
    - Price had a weak correlation with numerical features, suggesting other factors (like categorical attributes) may be more influential.



```r
# Correlation Matrix


# Install and load the packages
install.packages("ggplot2")
install.packages("reshape2")
library(ggplot2)
library(reshape2)
# Select only numeric columns
numeric_df <- df %>% select_if(is.numeric)

# Calculate the correlation matrix
cor_matrix <- cor(numeric_df, use = "complete.obs")

# Melt the correlation matrix for ggplot2
melted_cor_matrix <- melt(cor_matrix)

# Create the heatmap
ggplot(data = melted_cor_matrix, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  geom_text(aes(label = round(value, 2)), color = "black") +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
  theme_minimal() +
  labs(title = "Correlation Heatmap", x = "", y = "")
```

Output:



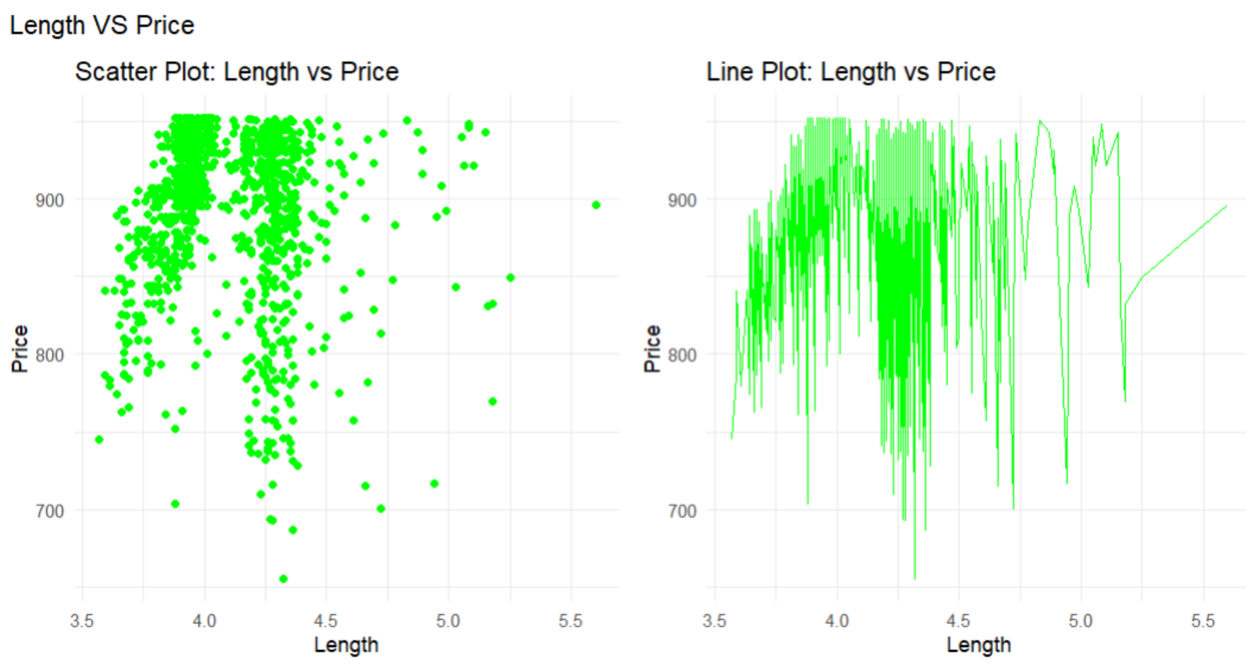
This correlation heatmap highlights the relationships between numerical features in the dataset. Features like Length, Width, Depth, and Weight exhibit strong positive correlations with each other, indicating high multicollinearity among these physical attributes. However, the correlation of these features with Price is weak (close to 0), suggesting that price is influenced more by categorical features like Clarity, Cut, and Colour rather than numerical dimensions. This insight is critical for feature selection during model development.

2. **Visualization of Numerical Features with Target Variables**
   a) **Length Vs Price**

The Length vs Price analysis explores the relationship between the length of diamonds and their market price. Scatter and line plots revealed no clear linear relationship, but trends show price fluctuations in specific length ranges. Notably, prices tend to decrease sharply around certain lengths, indicating other factors may have a stronger influence on price than length alone.

```r
# Length Vs Price

# Load necessary libraries
library(ggplot2)
library(patchwork)

# Scatter plot
scatter_plot <- ggplot(df, aes(x = Length, y = Price)) +
  geom_point(color = "green") +
  ggtitle("Scatter Plot: Length vs Price") +
  theme_minimal()

# Line plot
line_plot <- ggplot(df, aes(x = Length, y = Price)) +
  geom_line(color = "green") +
  scale_x_continuous(breaks = c(3.5, 4.0, 4.5, 5.0, 5.5)) +
  ggtitle("Line Plot: Length vs Price") +
  theme_minimal()

# Combine plots side by side
combined_plot <- scatter_plot + line_plot +
  plot_annotation(title = "Length VS Price")

# Display the combined plot
print(combined_plot)
```

Output:

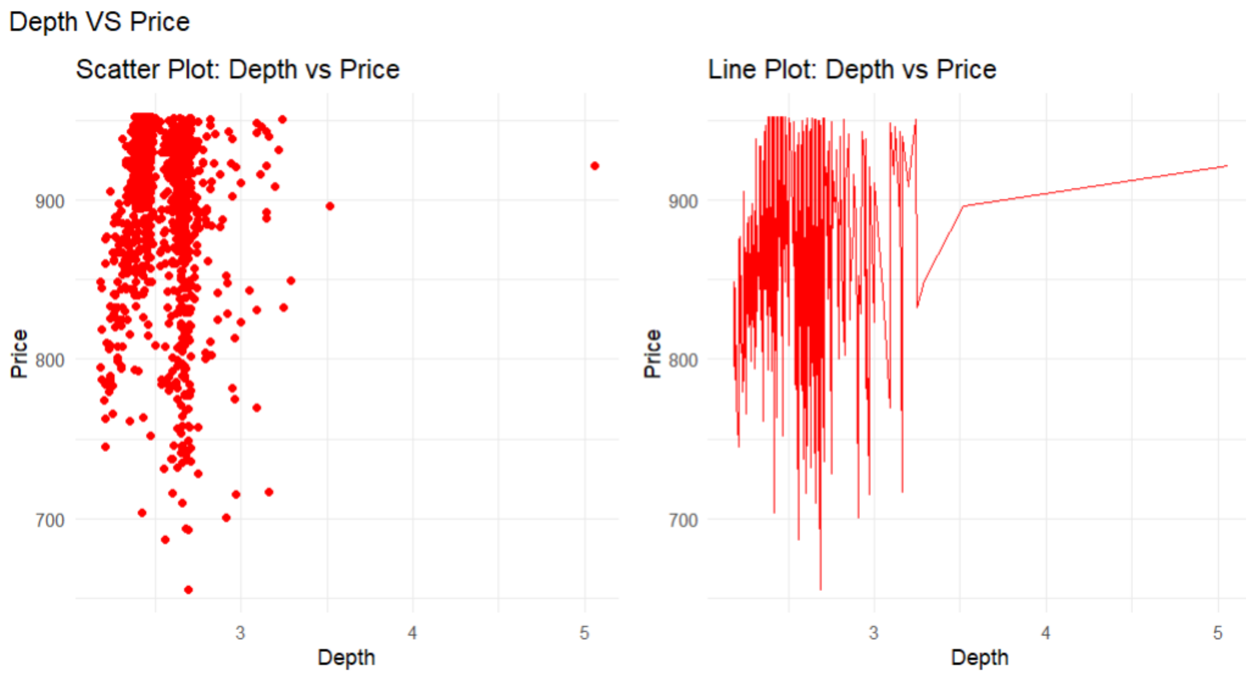


b) Depth Vs Price

The Depth vs Price analysis examines how the depth of a diamond correlates with its price. The scatter and line plots show no significant linear relationship, but prices exhibit noticeable increases around certain depth values, such as 3.3–3.4. These fluctuations suggest that depth alone is not a strong determinant of price and must be considered alongside other features.

```
# Depth Vs Price


# Scatter plot of Depth vs Price
scatter_plot <- ggplot(df, aes(x = Depth, y = Price)) +
  geom_point(color = "red") +
  ggtitle("Scatter Plot: Depth vs Price") +
  theme_minimal()

# Line plot of Depth vs Price
line_plot <- ggplot(df, aes(x = Depth, y = Price)) +
  geom_line(color = "red") +
  ggtitle("Line Plot: Depth vs Price") +
  theme_minimal()

# Combine the plots side by side
combined_plot <- scatter_plot + line_plot +
  plot_annotation(title = "Depth VS Price")

# Display the combined plot
print(combined_plot)
```

Output:

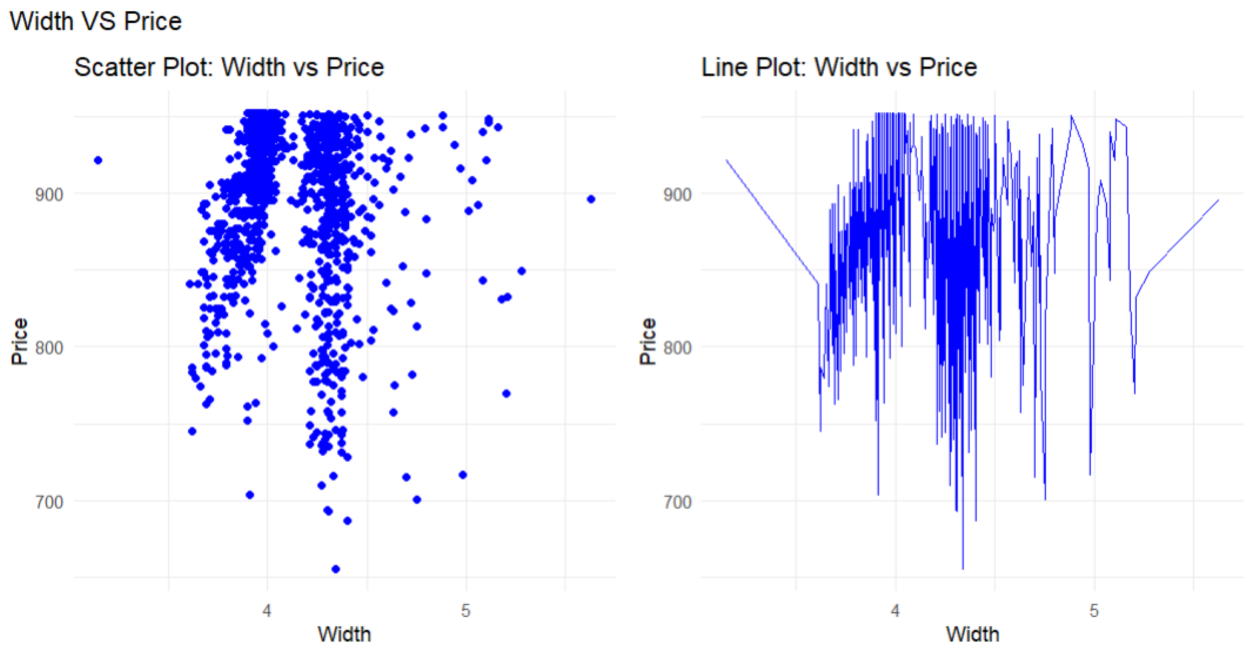


c) Width Vs Price

The **Width vs Price** analysis investigates the relationship between the width of a diamond and its price. The scatter and line plots reveal no clear linear correlation, though price trends indicate increases for widths above 5.25. Specific widths, such as 4.8–4.85 and 5.1, correspond to higher prices, highlighting potential nonlinear influences or interactions with other attributes.

```
# Width Vs Price

# Scatter plot of Width vs Price
scatter_plot <- ggplot(df, aes(x = Width, y = Price)) +
  geom_point(color = "blue") +
  ggtitle("Scatter Plot: Width vs Price") +
  theme_minimal()

# Line plot of Width vs Price
line_plot <- ggplot(df, aes(x = Width, y = Price)) +
  geom_line(color = "blue") +
  ggtitle("Line Plot: Width vs Price") +
  theme_minimal()

# Combine the plots side by side with a shared title
combined_plot <- scatter_plot + line_plot +
  plot_annotation(title = "Width VS Price")

# Display the combined plot
print(combined_plot)
```

Output:

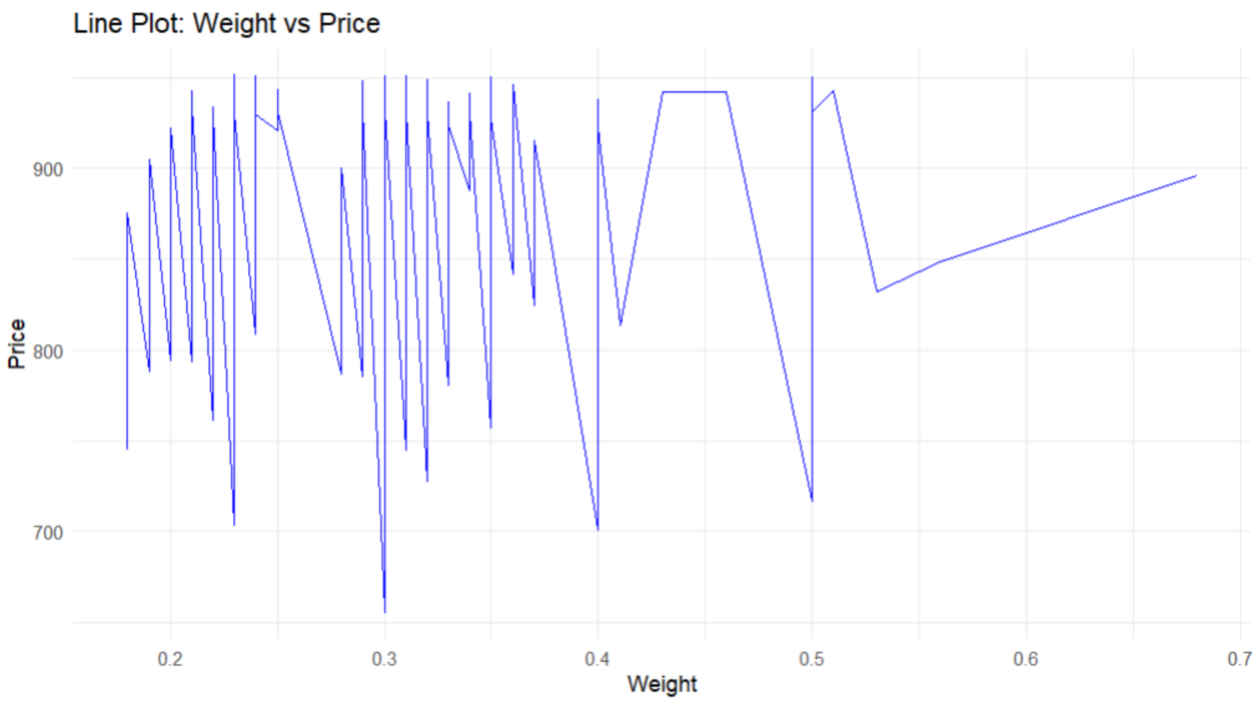


d) Weight Vs Price

The **Weight vs Price** analysis explores the impact of a diamond's weight (in carats) on its price. The relationship is nonlinear, with prices increasing significantly after certain weight thresholds, such as 0.55 carats. Notable spikes in price occur around weights of 0.45 and 0.51–0.52 carats, emphasizing the importance of weight as a critical factor influencing diamond valuation.

```
# Weight Vs Price (in Carat)


# Line plot of Weight vs Price
line_plot <- ggplot(df, aes(x = Weight, y = Price)) +
  geom_line(color = "blue") +
  theme_minimal() +
  ggtitle("Line Plot: Weight vs Price")

# Display the Line plot
print(line_plot)

# Observation: -

# 1)Again there is no Linear relationship between Weight and Price
# 2)But we also see high increase in Price after 0.55 weigth
# 3)Highest Price is at approx 0.45 and 0.51-0.52
# 4)There are Outliers too we will deal with them later
```

Output:



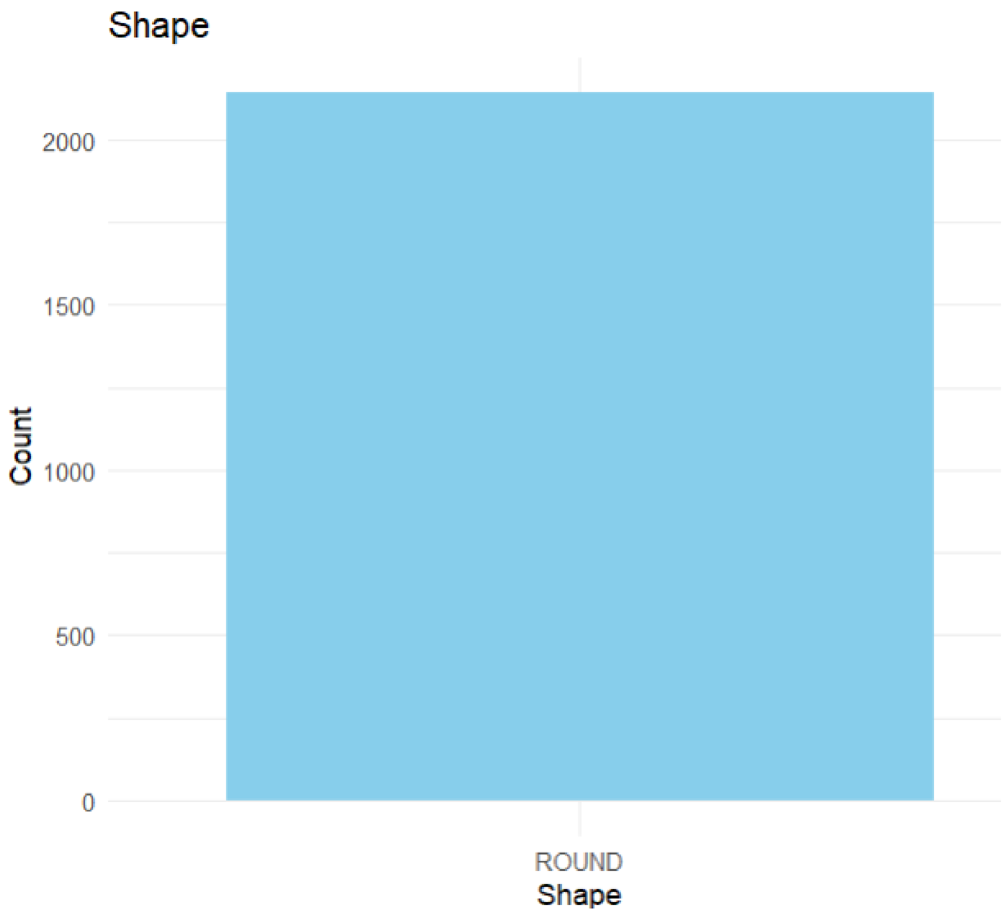
3. Visualization of Categorical Features

1. Shape

The Shape feature in the dataset represents the geometric form of the diamonds. All diamonds in this dataset are of the ROUND shape, making it a uniform characteristic across all entries. As a result, this feature was identified as redundant and excluded from further analysis to streamline the dataset and focus on more variable attributes that influence price.

```
# Countplot for Shape


# Create the count plot
ggplot(df, aes(x = Shape)) +
  geom_bar(fill = "skyblue") +
  theme_minimal() +
  labs(title = "Shape", x = "Shape", y = "Count")

#Observation: -
#The dataset is of Round Shape Diamonds so all of them are Round we will drop this column afterwards
```

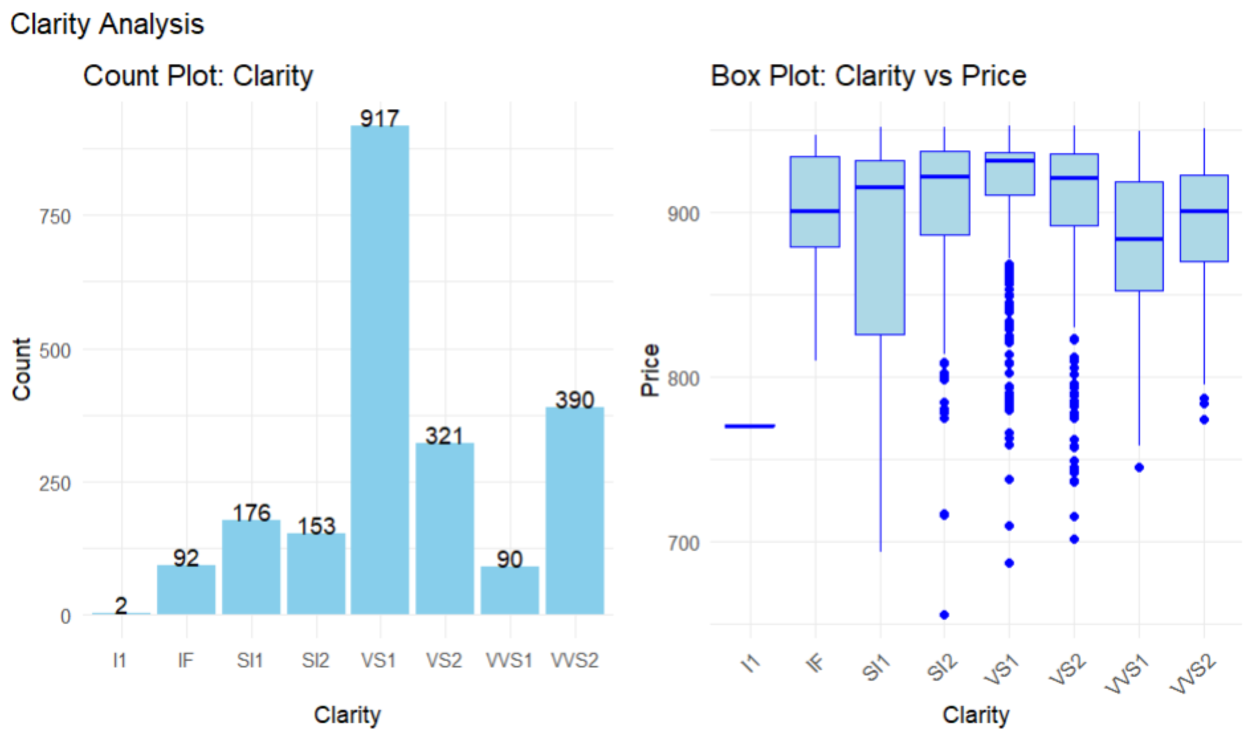
Output:



2. Clarity

```r
# Count plot of Clarity
count_plot <- ggplot(df, aes(x = Clarity)) +
  geom_bar(fill = "skyblue") +
  geom_text(stat = 'count', aes(label = ..count..), vjust = 0) +  # Add bar labels
  theme_minimal() +
  labs(title = "Count Plot: Clarity", x = "Clarity", y = "Count")

# Box plot of Clarity vs. Price
box_plot <- ggplot(df, aes(x = Clarity, y = Price)) +
  geom_boxplot(fill = "lightblue", color = "blue") +  # Box plot with color
  theme_minimal() +
  labs(title = "Box Plot: Clarity vs Price", x = "Clarity", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combine the plots side by side with a shared title
combined_plot <- count_plot + box_plot +
  plot_annotation(title = "Clarity Analysis")

# Display the combined plot
print(combined_plot)
```

Output:

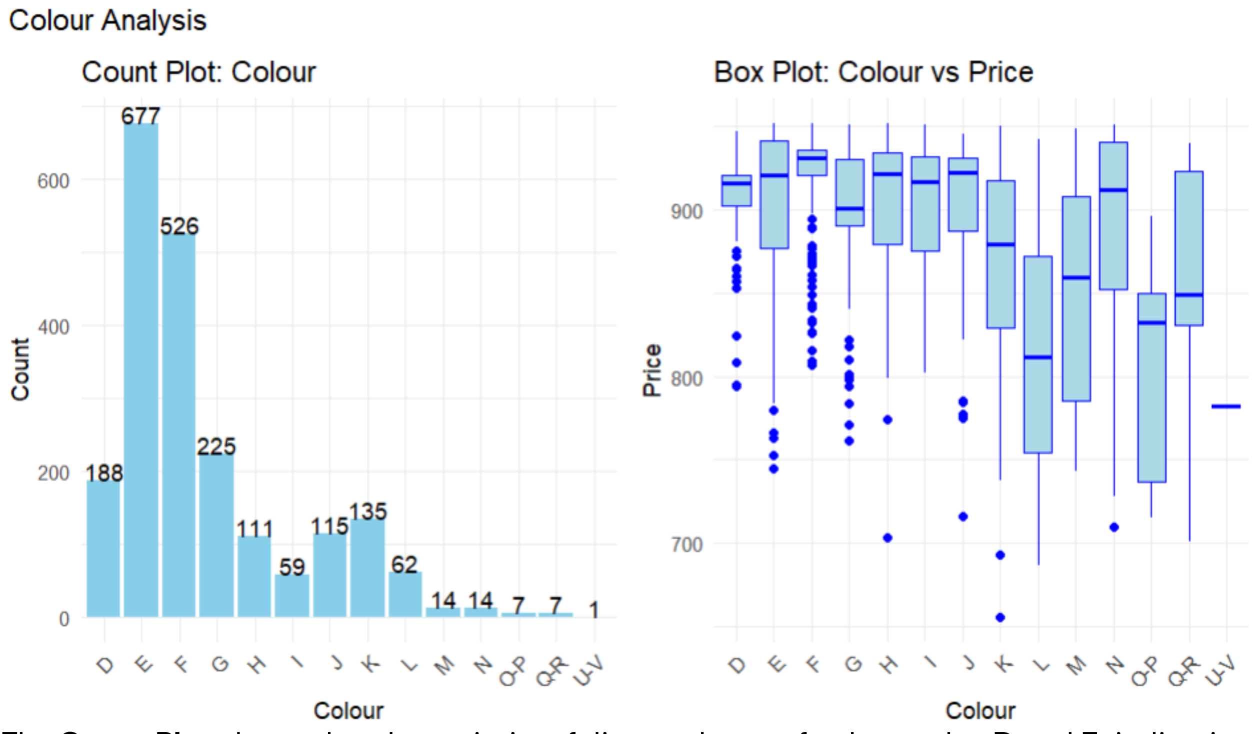


The **Count Plot** shows that most diamonds have a clarity grade of **VS1**, followed by **SI2** and **VS2**, while grades like **I1** and **IF** are rare. The **Box Plot** illustrates the distribution of prices for each clarity grade, revealing that higher clarity grades generally correspond to higher prices, with **VS1** and **VVS1** showing a broader price range and some outliers.

3. Colour

```
# Create the count plot
count_data <- df %>%
  count(Colour)

count_plot <- ggplot(count_data, aes(x = Colour, y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  geom_text(aes(label = n), vjust = 0) +
  theme_minimal() +
  labs(title = "Count Plot: Colour", x = "Colour", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Create the box plot
box_plot <- ggplot(df, aes(x = Colour, y = Price)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  theme_minimal() +
  labs(title = "Box Plot: Colour vs Price", x = "Colour", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combine the plots side by side
combined_plot <- count_plot + box_plot +
  plot_annotation(title = "Colour Analysis")

# Display the combined plot
print(combined_plot)
```

Output:

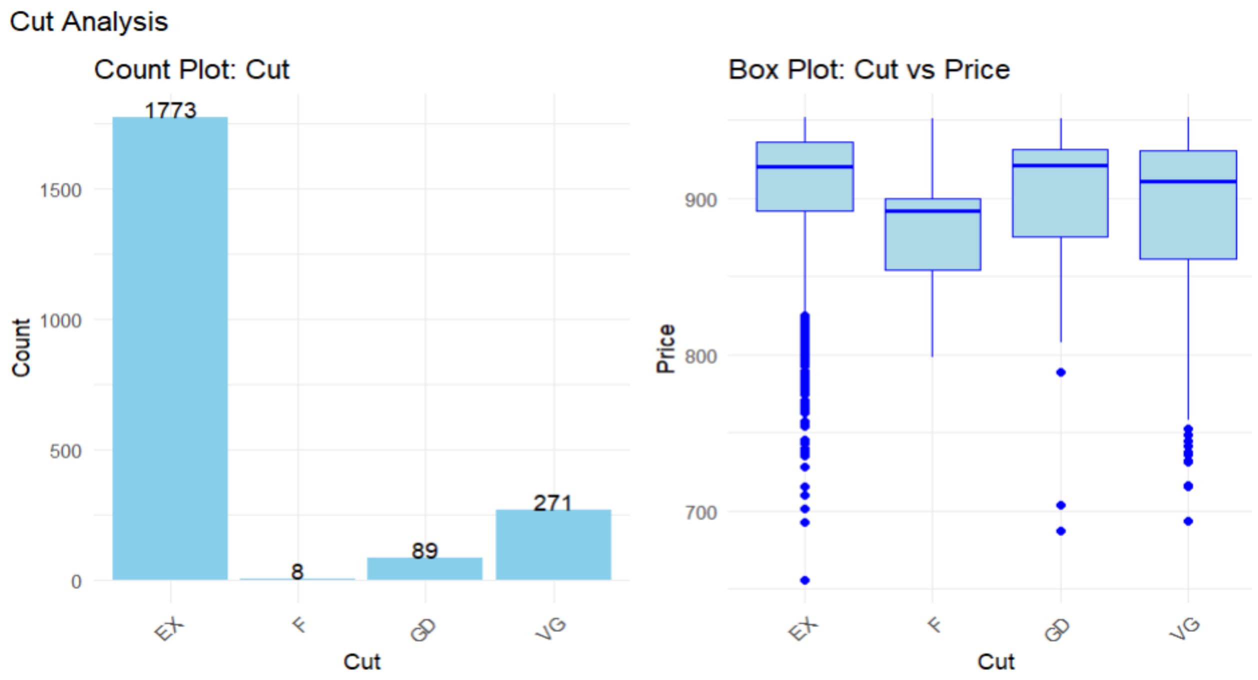


The **Count Plot** shows that the majority of diamonds are of color grades **D** and **E**, indicating higher quality. The **Box Plot** highlights the distribution of prices across different color grades, with better color grades (D, E, F) generally having higher prices and fewer outliers compared to lower grades (K, L, M). This suggests color significantly influences diamond pricing.

4. Cut

```
# Cut
# Create the count plot
count_data <- df %>%
  count(Cut)
count_plot <- ggplot(count_data, aes(x = Cut, y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  geom_text(aes(label = n), vjust = 0) +
  theme_minimal() +
  labs(title = "Count Plot: Cut", x = "Cut", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Create the box plot
box_plot <- ggplot(df, aes(x = Cut, y = Price)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  theme_minimal() +
  labs(title = "Box Plot: Cut vs Price", x = "Cut", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combine the plots side by side with a shared title
combined_plot <- count_plot + box_plot +
  plot_annotation(title = "Cut Analysis")
# Display the combined plot
print(combined_plot)
```

Output:

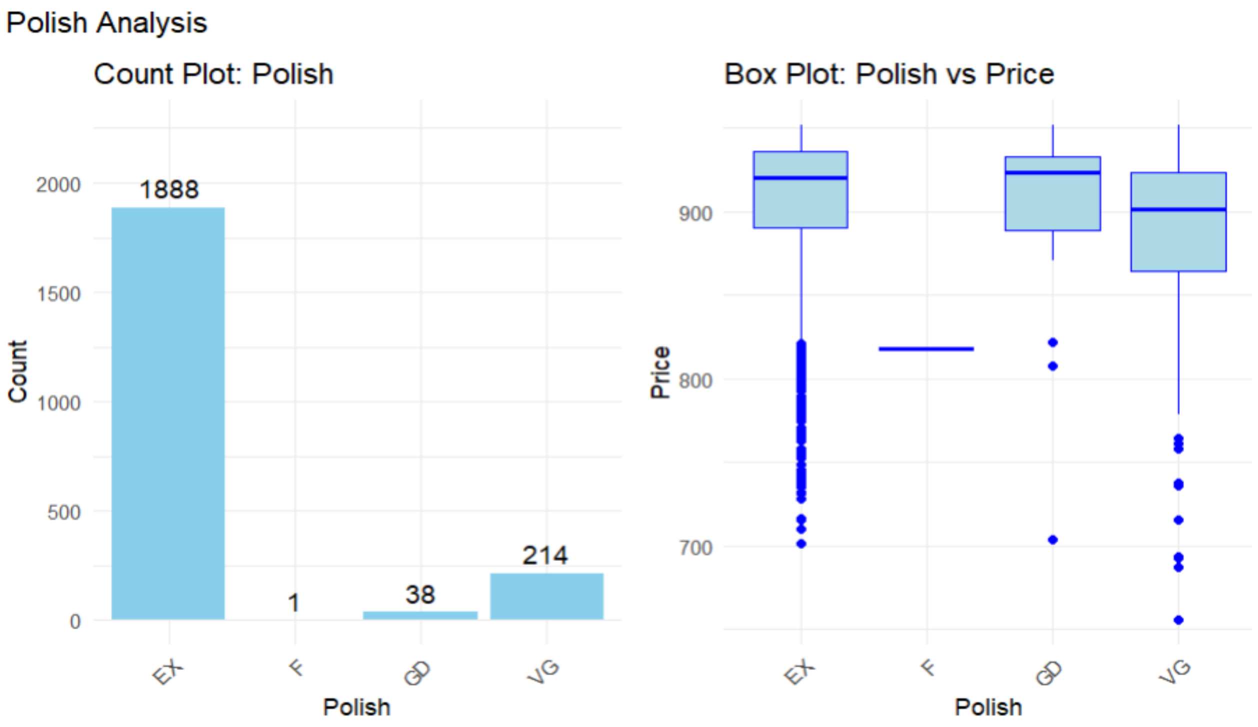


The **Count Plot** indicates that most diamonds have an **Excellent (EX)** cut, followed by **Very Good (VG)**, while other grades like **Fair (F)** are rare. The **Box Plot** shows that diamonds with higher cut grades, such as **EX** and **VG**, generally have higher median prices, with fewer outliers compared to lower cut grades. This emphasizes the significant influence of cut quality on diamond pricing.

5. **Polish**

```r
# Polish
# Create the count plot
count_data <- df %>%
  count(Polish)

count_plot <- ggplot(count_data, aes(x = Polish, y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  geom_text(aes(label = n), vjust = -0.5, size = 4) +  # Add labels above the bars
  ylim(0, max(count_data$n) * 1.2) +  # Extend y-axis limits for visibility
  theme_minimal() +
  labs(title = "Count Plot: Polish", x = "Polish", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Create the box plot
box_plot <- ggplot(df, aes(x = Polish, y = Price)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  theme_minimal() +
  labs(title = "Box Plot: Polish vs Price", x = "Polish", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combine the plots side by side with a shared title
combined_plot <- count_plot + box_plot +
  plot_annotation(title = "Polish Analysis")
# Display the combined plot
print(combined_plot)
```

Output:

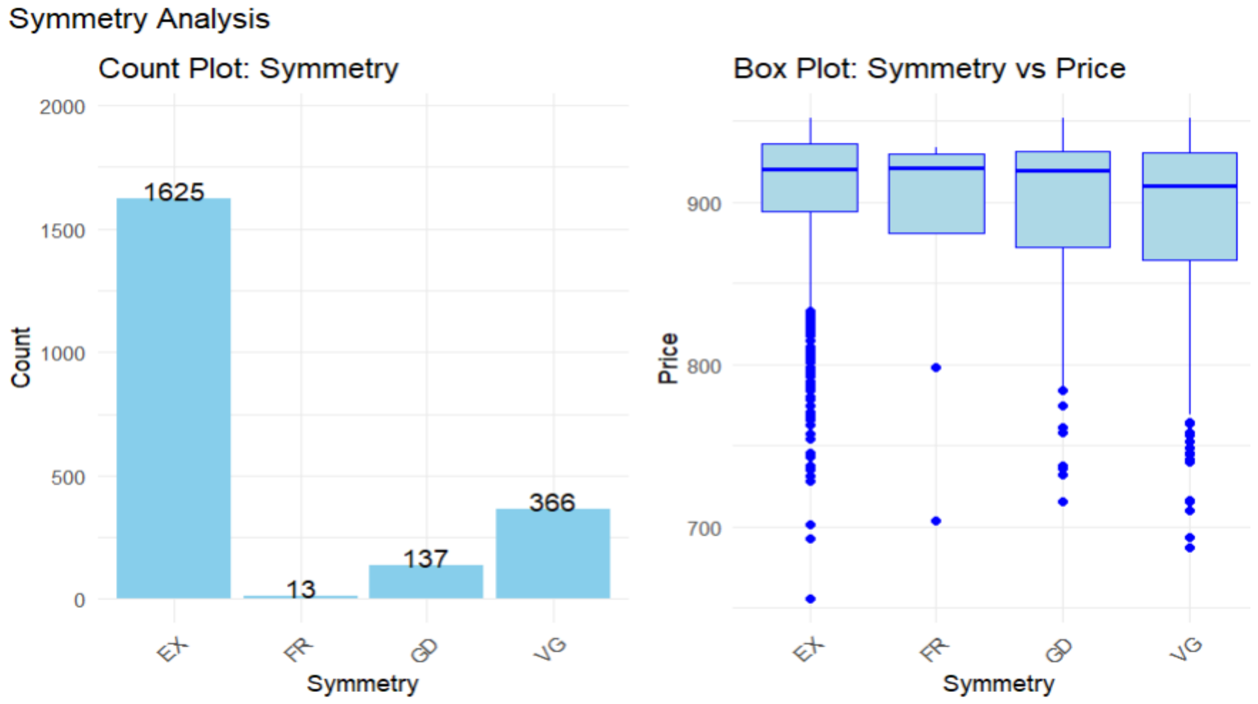


The **Count Plot** highlights that most diamonds have an **Excellent (EX)** polish, with a small number having **Very Good (VG)** polish and even fewer in other categories. The **Box Plot** shows that diamonds with higher polish grades, such as **EX** and **VG**, tend to have higher prices, with fewer outliers in higher polish grades. This indicates that polish quality contributes to price variation.

6. Symmetry

```
# Symmetry
# Create the count plot
count_data <- df %>%
  count(Symmetry)

count_plot <- ggplot(count_data, aes(x = Symmetry, y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  geom_text(aes(label = n), vjust = 0.1, size = 4) +  # Add count labels above the bars
  ylim(0, max(count_data$n) * 1.2) +  # Extend y-axis limits for label visibility
  theme_minimal() + labs(title = "Count Plot: Symmetry", x = "Symmetry", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Create the box plot
box_plot <- ggplot(df, aes(x = Symmetry, y = Price)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  theme_minimal() +
  labs(title = "Box Plot: Symmetry vs Price", x = "Symmetry", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combine the plots side by side with a shared title
combined_plot <- count_plot + box_plot +
  plot_annotation(title = "Symmetry Analysis")
# Display the combined plot
print(combined_plot)
```

Output:

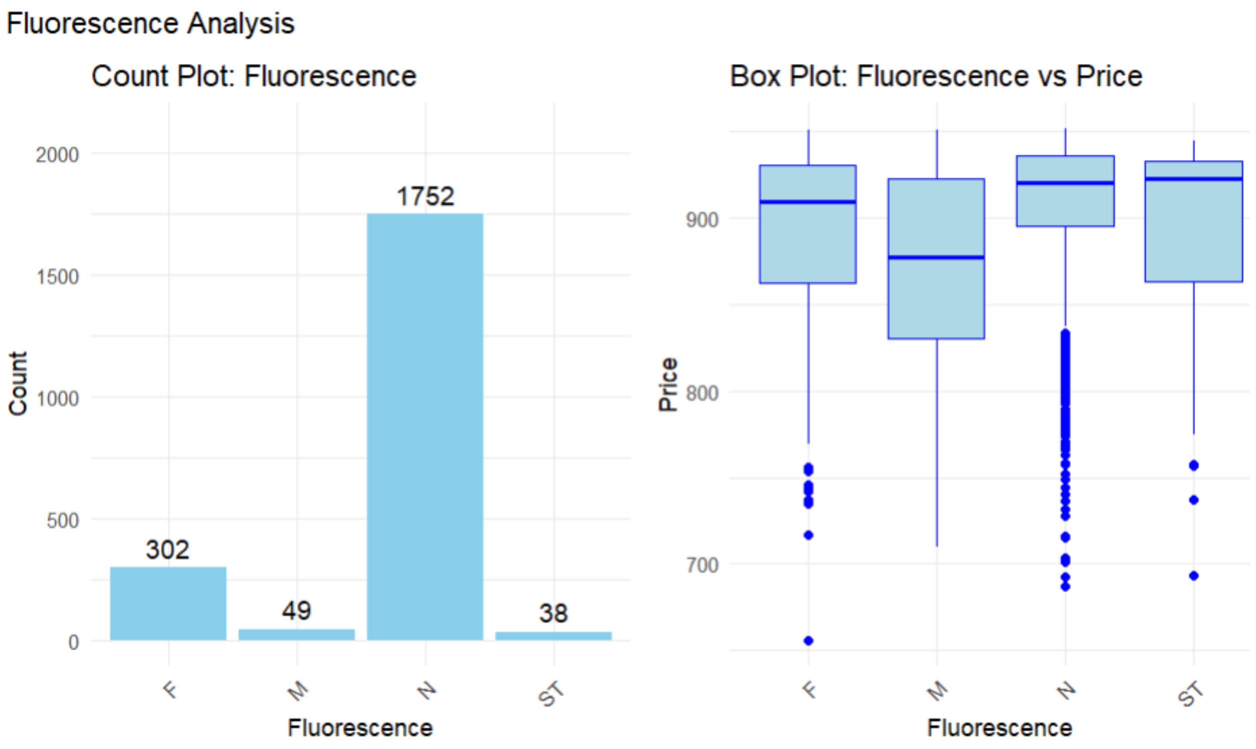


The **Count Plot** shows that most diamonds have an **Excellent (EX)** symmetry, followed by **Very Good (VG)** symmetry, with fewer diamonds in other categories. The **Box Plot** highlights those diamonds with higher symmetry grades, such as **EX** and **VG**, generally have higher median prices and fewer variations, while lower grades show more variability in pricing. This suggests symmetry significantly impacts diamond value.

7. Fluorescence

```r
# Fluoresence
# Create the count plot
count_data <- df %>%
  count(Fluorescence)

count_plot <- ggplot(count_data, aes(x = Fluorescence, y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  geom_text(aes(label = n), vjust = -0.5, size = 4) +  # Add count labels above the bars
  ylim(0, max(count_data$n) * 1.2) +  # Extend y-axis limits for label visibility
  theme_minimal() + labs(title = "Count Plot: Fluorescence", x = "Fluorescence", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Create the box plot
box_plot <- ggplot(df, aes(x = Fluorescence, y = Price)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  theme_minimal() +
  labs(title = "Box Plot: Fluorescence vs Price", x = "Fluorescence", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combine the plots side by side with a shared title
combined_plot <- count_plot + box_plot +
  plot_annotation(title = "Fluorescence Analysis")
# Display the combined plot
print(combined_plot)
```

Output:



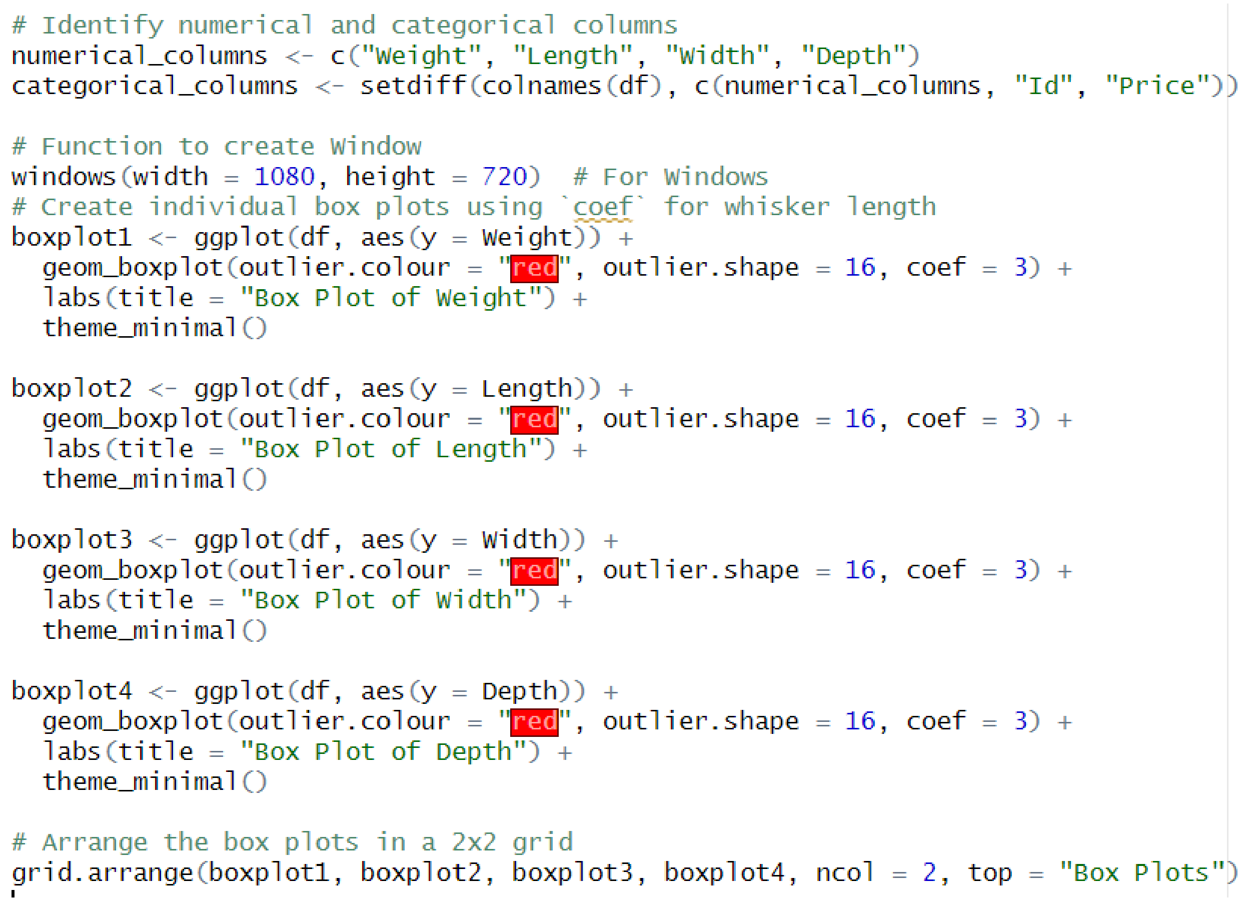
This analysis explores the relationship between diamond fluorescence and price. The count plot reveals that most diamonds have "N" fluorescence, followed by "F," while "M" and "ST" categories are relatively rare. The box plot shows that prices are consistent across fluorescence types, with "N" fluorescence exhibiting slightly wider variability and outliers.

Dealing with Outliers

Outliers can significantly impact the performance of predictive models. To address this issue, we used two methods to detect and handle outliers:

1. **Box Plot Visualization:**

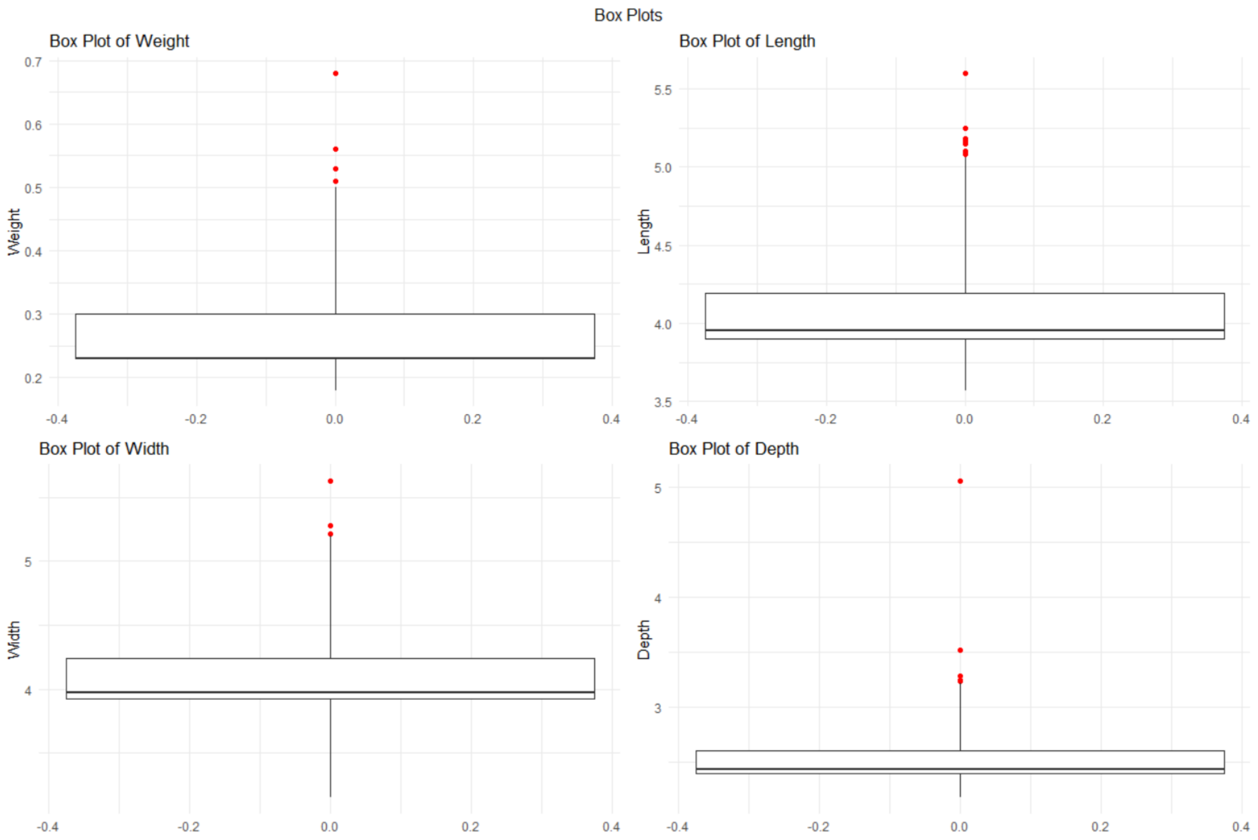
   Box plots were generated for numerical features such as Weight, Length, Width, and Depth. This visualization helped identify extreme data points beyond the whiskers, which indicate potential outliers.



```r
# Identify numerical and categorical columns
numerical_columns <- c("Weight", "Length", "Width", "Depth")
categorical_columns <- setdiff(colnames(df), c(numerical_columns, "Id", "Price"))

# Function to create Window
windows(width = 1080, height = 720)  # For Windows
# Create individual box plots using `coef` for whisker length
boxplot1 <- ggplot(df, aes(y = Weight)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Weight") +
  theme_minimal()

boxplot2 <- ggplot(df, aes(y = Length)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Length") +
  theme_minimal()

boxplot3 <- ggplot(df, aes(y = Width)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Width") +
  theme_minimal()

boxplot4 <- ggplot(df, aes(y = Depth)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Depth") +
  theme_minimal()

# Arrange the box plots in a 2x2 grid
grid.arrange(boxplot1, boxplot2, boxplot3, boxplot4, ncol = 2, top = "Box Plots")
```

The code generates box plots for the numerical columns Weight, Length, Width, and Depth to visualize outliers. Each plot highlights outliers in red, using a coefficient of 3 for whisker length to identify data points that deviate significantly from the interquartile range. The box plots are arranged in a 2x2 grid for better comparison, providing a comprehensive view of potential outliers in the dataset. This step is crucial for detecting and analyzing anomalies in the data.

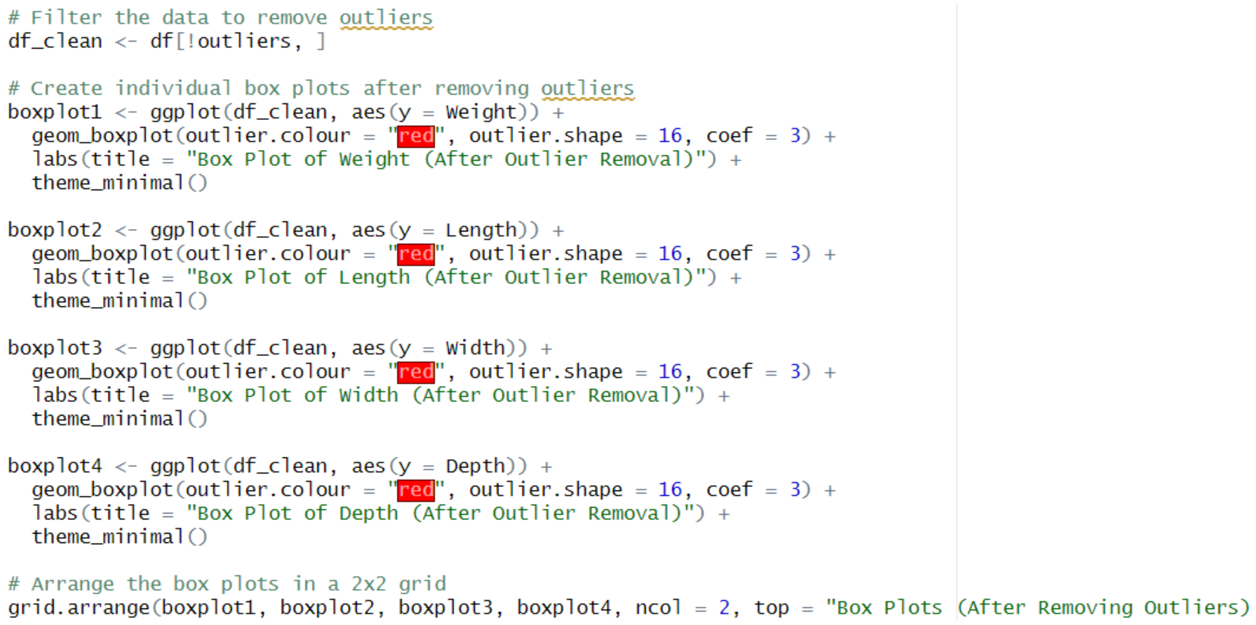
Output:



2. **Z-Score Method:**
   Z-scores were calculated for numerical columns to quantify how far each data point deviates from the mean. A threshold of 3 was applied, meaning any data point with a Z-score above 3 or below -3 was flagged as an outlier.

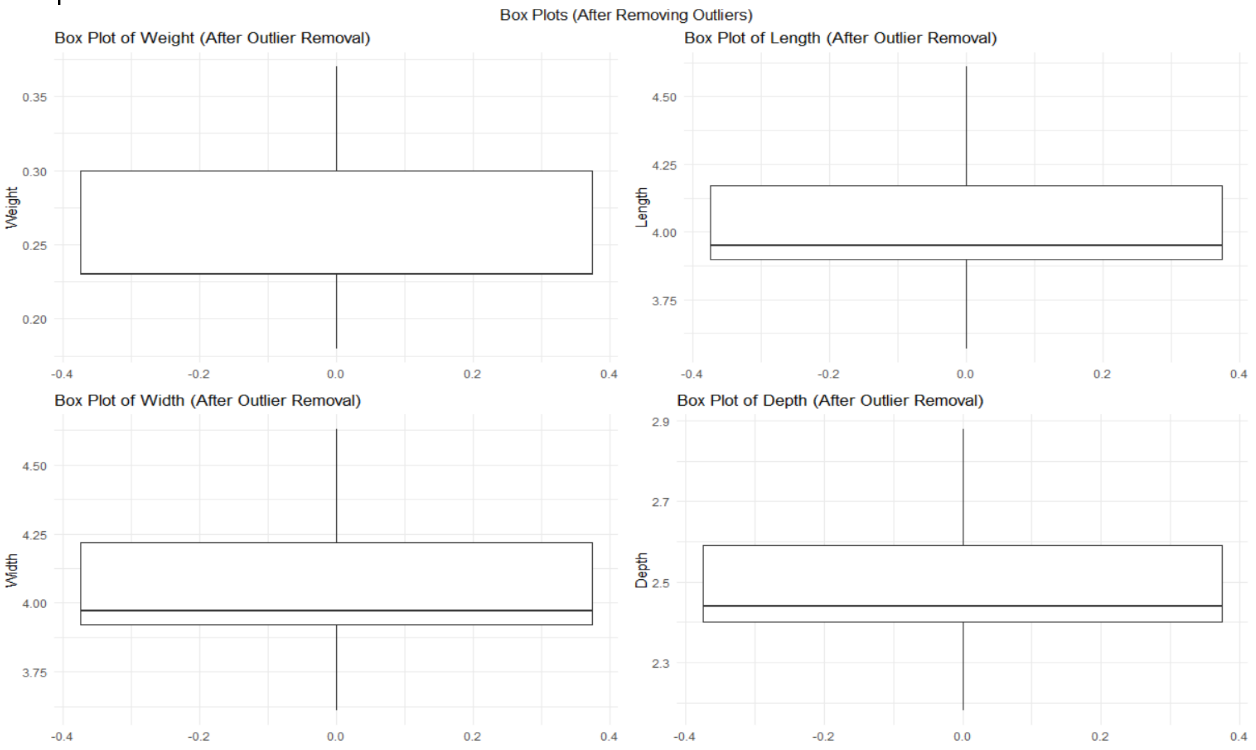
```
# Function to create Window
windows(width = 1080, height = 720)  # For Windows
# Calculate z-scores for numerical columns
z_scores <- as.data.frame(scale(df[numerical_columns]))

# Define the threshold for z-scores
threshold <- 3

# Identify rows with any z-score greater than the threshold
outliers <- apply(z_scores, 1, function(x) any(abs(x) > threshold))
```

3. **Removal of Outliers:**
   Rows containing outliers were excluded from the dataset to ensure the model is trained on clean data. After removal, box plots were regenerated to confirm that the dataset was free from extreme anomalies.



```r
# Filter the data to remove outliers
df_clean <- df[!outliers, ]

# Create individual box plots after removing outliers
boxplot1 <- ggplot(df_clean, aes(y = Weight)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Weight (After Outlier Removal)") +
  theme_minimal()

boxplot2 <- ggplot(df_clean, aes(y = Length)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Length (After Outlier Removal)") +
  theme_minimal()

boxplot3 <- ggplot(df_clean, aes(y = Width)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Width (After Outlier Removal)") +
  theme_minimal()

boxplot4 <- ggplot(df_clean, aes(y = Depth)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, coef = 3) +
  labs(title = "Box Plot of Depth (After Outlier Removal)") +
  theme_minimal()

# Arrange the box plots in a 2x2 grid
grid.arrange(boxplot1, boxplot2, boxplot3, boxplot4, ncol = 2, top = "Box Plots (After Removing Outliers)
```

Output:



By removing outliers, the dataset was better structured for effective model training and analysis, minimizing the risk of skewed results caused by extreme data points.

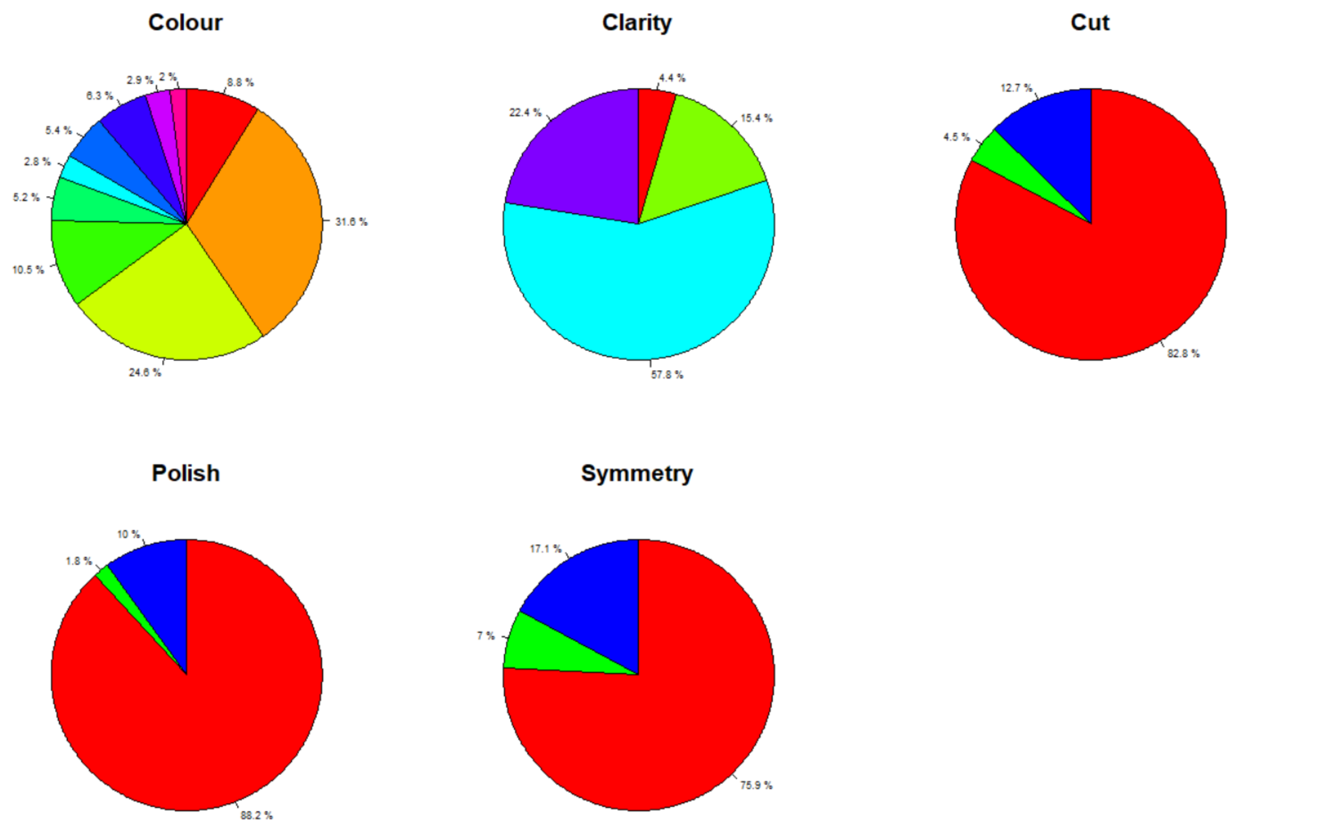
Feature Grouping

In this feature engineering step, we simplify categorical variables by combining categories with low frequency into broader groups. This ensures that the model isn't biased or overwhelmed by rarely occurring categories. The key changes are:

1. **Colour Grouping:**
    - Rare colors like N, M, O-P, U-V, and Q-R are grouped into a new category labeled as "Other". This simplifies the Colour variable while retaining meaningful information.
2. **Clarity Grouping:**
    - Rare clarity levels are grouped to streamline the Clarity variable:
        - VVS1 and VVS2 are combined into "VVS".
        - SI1 and SI2 are combined into "SI".
        - VS1 and VS2 are combined into "VS".
        - Rare clarity levels like IF and I1 are labeled as "Other".
3. **Cut Adjustment:**
    - The category F (Fair) for Cut, which has only 8 instances, is replaced with GD (Good) to reduce sparsity.
4. **Polish Adjustment:**
    - The category F (Fair) for Polish, with just 1 instance, is replaced with GD (Good) to maintain consistency and simplify the variable.
5. **Symmetry Adjustment:**
    - The rare FR (Fair) category in Symmetry, with only 13 instances, is replaced with GD (Good), ensuring a more robust grouping.

This process of combining low-frequency categories minimizes noise and helps the model focus on dominant patterns, ultimately improving its predictive accuracy and interpretability.

```r
#Combining the features with minimum count as 'Other' in this columns
#Colour,Clarity,Cut,Polish,Symmetry


# Load necessary library
library(dplyr)

# Colour
df$Colour <- ifelse(df$Colour %in% c("N", "M"), "Other", df$Colour)
df$Colour <- ifelse(df$Colour %in% c("O-P", "U-V"), "Other", df$Colour)
df$Colour <- ifelse(df$Colour == "Q-R", "Other", df$Colour)

# Clarity
df$Clarity <- ifelse(df$Clarity %in% c("VVS1", "VVS2"), "VVS", df$Clarity)
df$Clarity <- ifelse(df$Clarity %in% c("SI1", "SI2"), "SI", df$Clarity)
df$Clarity <- ifelse(df$Clarity %in% c("VS1", "VS2"), "VS", df$Clarity)
df$Clarity <- ifelse(df$Clarity %in% c("IF", "I1"), "Other", df$Clarity)

# Cut (Only 8 counts of F so replacing Fair with Very Good)
df$Cut <- ifelse(df$Cut == "F", "GD", df$Cut)

# Polish (Only 1 counts of F so replacing Fair with Very Good)
df$Polish <- ifelse(df$Polish == "F", "GD", df$Polish)

# Symmetry (Only 13 counts of F so replacing Fair with Very Good)
df$Symmetry <- ifelse(df$Symmetry == "FR", "GD", df$Symmetry)
```
Output:

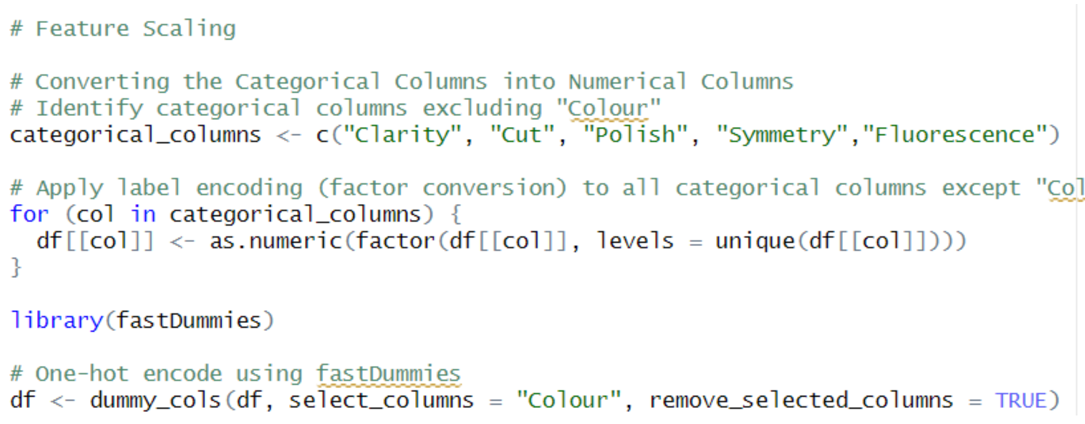


Feature Engineering

In this step, **Feature Scaling** is performed to convert categorical variables into numerical values, making them suitable for machine learning models. Here's how it is done:

1. **Label Encoding for Categorical Columns:**
    - Columns such as Clarity, Cut, Polish, Symmetry, and Fluorescence are label-encoded. This is achieved by converting their categorical levels into unique numeric values using the factor() function.
    - This helps the model understand ordinal relationships or distinct classes in these variables.
2. **One-Hot Encoding for** Colour**:**
    - The Colour column is one-hot encoded using the fastDummies library. This process creates new binary columns for each category in the Colour feature (e.g., Colour_D, Colour_E), indicating the presence or absence of a specific color.
    - The original Colour column is removed after encoding to avoid redundancy.

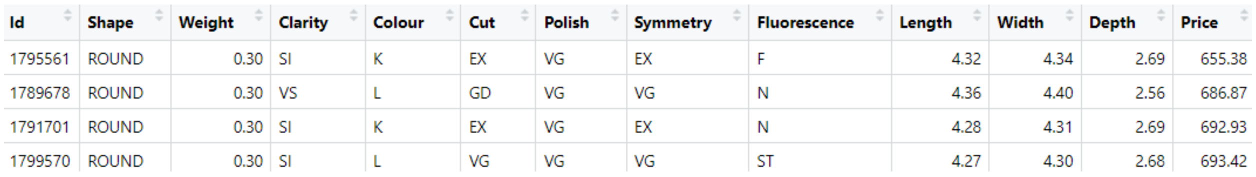
This approach ensures that both ordinal and nominal categorical variables are appropriately represented numerically, preserving the original structure of the data while making it compatible with machine learning algorithms. It avoids introducing biases and improves the model's performance and interpretability.



```r
# Feature Scaling

# Converting the Categorical Columns into Numerical Columns
# Identify categorical columns excluding "Colour"
categorical_columns <- c("Clarity", "Cut", "Polish", "Symmetry","Fluorescence")

# Apply label encoding (factor conversion) to all categorical columns except "Colour"
for (col in categorical_columns) {
  df[[col]] <- as.numeric(factor(df[[col]], levels = unique(df[[col]])))
}

library(fastDummies)

# One-hot encode using fastDummies
df <- dummy_cols(df, select_columns = "Colour", remove_selected_columns = TRUE)
```

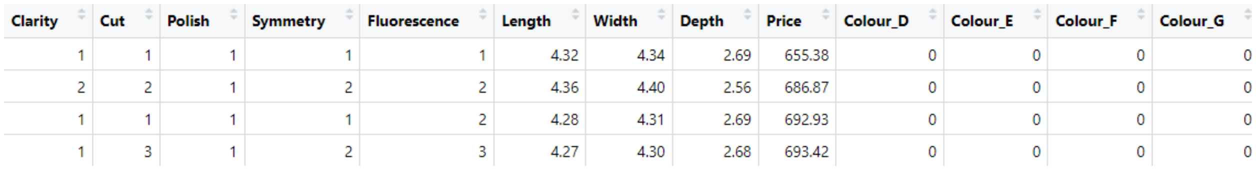
Output:

Before:



Id	Shape	Weight	Clarity	Colour	Cut	Polish	Symmetry	Fluorescence	Length	Width	Depth	Price
1795561	ROUND	0.30	SI	K	EX	VG	EX	F	4.32	4.34	2.69	655.38
1789678	ROUND	0.30	VS	L	GD	VG	VG	N	4.36	4.40	2.56	686.87
1791701	ROUND	0.30	SI	K	EX	VG	EX	N	4.28	4.31	2.69	692.93
1799570	ROUND	0.30	SI	L	VG	VG	VG	ST	4.27	4.30	2.68	693.42

After:



Clarity	Cut	Polish	Symmetry	Fluorescence	Length	Width	Depth	Price	Colour_D	Colour_E	Colour_F	Colour_G
1	1	1	1	1	4.32	4.34	2.69	655.38	0	0	0	0
2	2	1	2	2	4.36	4.40	2.56	686.87	0	0	0	0
1	1	1	1	2	4.28	4.31	2.69	692.93	0	0	0	0
1	3	1	2	3	4.27	4.30	2.68	693.42	0	0	0	0

Numerical Column Standardization and Data Cleaning

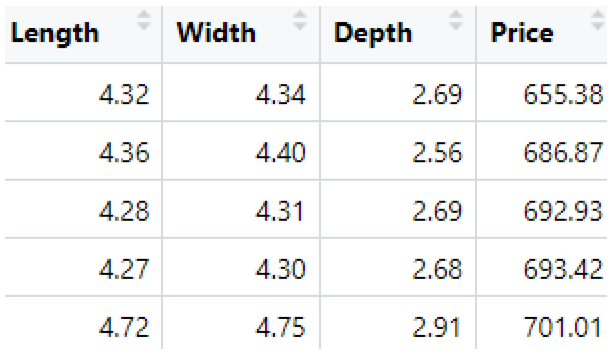
1. **Standardization of Numerical Columns:**
    - Numerical columns such as Weight, Length, Width, and Depth are standardized using the scale() function.
    - Standardization transforms the data to have a mean of 0 and a standard deviation of 1, ensuring that all numerical features are on the same scale. This is crucial for machine learning algorithms that are sensitive to feature magnitudes, such as gradient descent-based models.
2. **Dropping Unnecessary Columns:**
    - The Id column is removed as it is a unique identifier and does not contribute to predictive modeling.
    - The Shape column is dropped, assuming it has no significant impact on the target variable or the analysis.
3. **Rearranging the** Price **Column:**
    - The Price column, which is the target variable, is moved to the last position in the dataset. This is done for better organization and ease of working with the data during modeling.

This step prepares the dataset by scaling numerical features for consistency, removing irrelevant columns to reduce noise, and organizing the target variable for clear distinction during model training.

```r
# Standardization of a Numerical Columns

library(scales)
numerical_columns <- c("Weight", "Length", "Width", "Depth")

# Standardization
df[numerical_columns] <- lapply(df[numerical_columns], scale)

# Drop the columns "Id" and "Shape"
df <- df %>% select(-Id, -Shape)

# Move "Price" column to the last position
df <- df %>% select(-Price, everything(), Price)
```

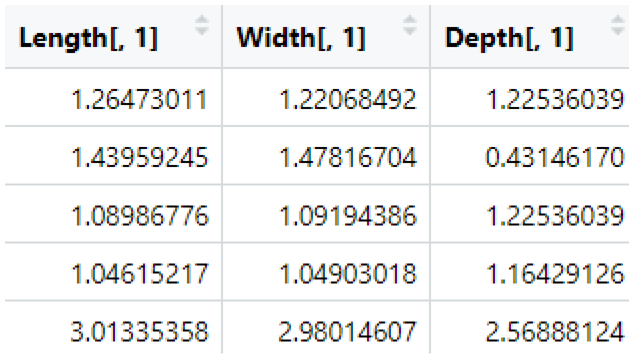
Output:

Before:



Length	Width	Depth	Price
4.32	4.34	2.69	655.38
4.36	4.40	2.56	686.87
4.28	4.31	2.69	692.93
4.27	4.30	2.68	693.42
4.72	4.75	2.91	701.01

After:



Length[, 1]	Width[, 1]	Depth[, 1]
1.26473011	1.22068492	1.22536039
1.43959245	1.47816704	0.43146170
1.08986776	1.09194386	1.22536039
1.04615217	1.04903018	1.16429126
3.01335358	2.98014607	2.56888124

- **Feature Selection**

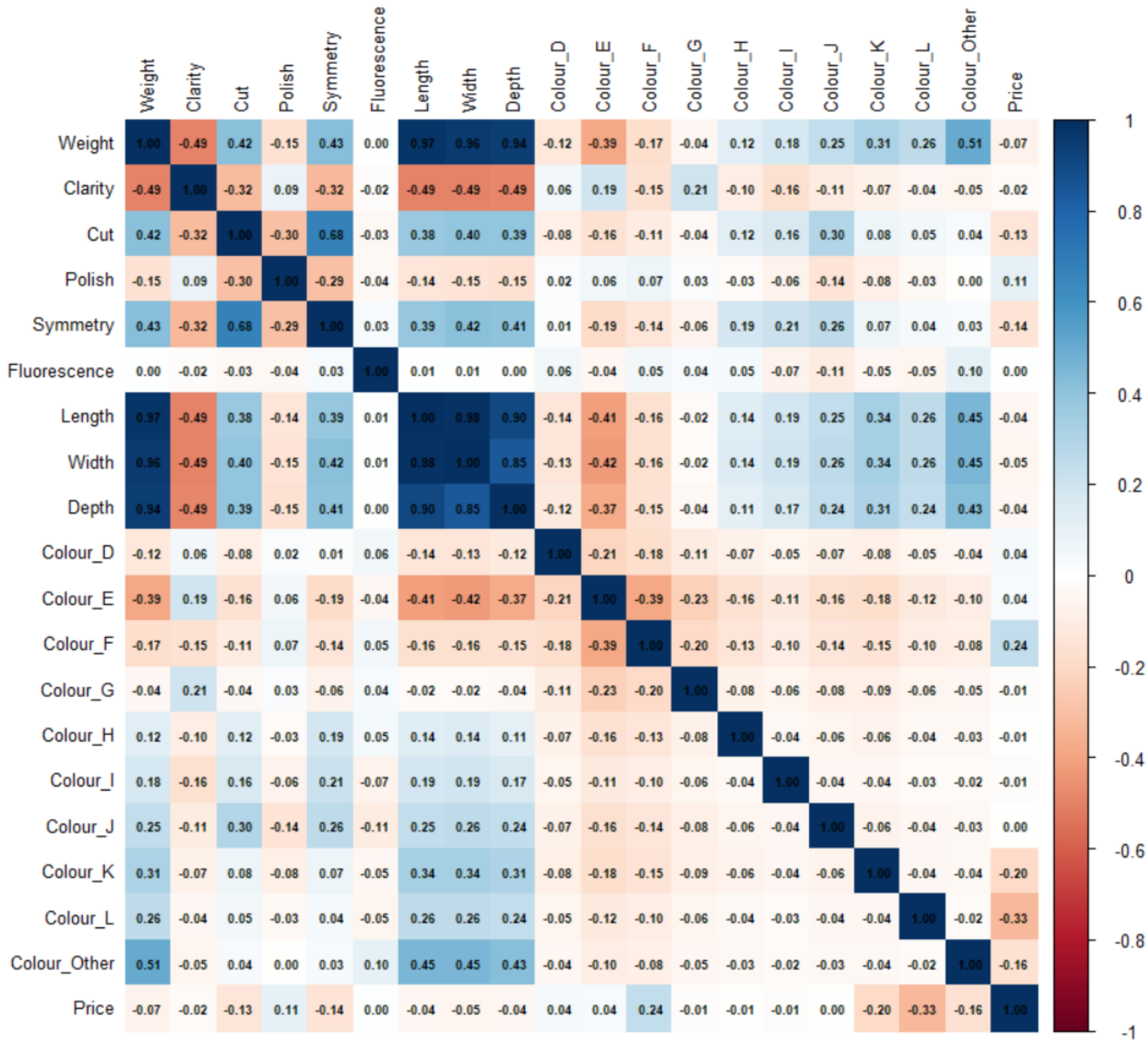
Explanation of Correlation Matrix Analysis

1. Purpose of the Correlation Matrix:
    - A correlation matrix is used to identify the relationships between different numerical features in the dataset. It quantifies how closely related two variables are, with values ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation).
2. Correlation Matrix Calculation:
    - The cor() function computes the pairwise correlations among all numerical features in the dataset, using only complete observations to avoid missing data issues.
3. Visualization with Heatmap:
    - The corrplot library is used to create a visually appealing heatmap, where colors represent the strength and direction of the correlation.
    - Features highly correlated with each other are highlighted, making it easier to identify significant relationships.
    - Annotations include correlation coefficients, text labels, and customized margins for better readability.
4. Key Observations:
    - The Price column (target variable) shows notable correlations with:
        - Cut, Polish, Symmetry, Fluorescence, and Colour: These categorical features appear to have an impact on the pricing structure, suggesting they could be key predictors.
    - This insight will help prioritize features for further analysis and model development.

This step provides critical insights into the dataset's structure, helping identify influential variables and reduce multicollinearity in predictive modeling.

```
#Correlation Matrix

# Load necessary libraries
library(corrplot)

windows(width = 1080, height = 720)  # For Windows

# Calculate the correlation matrix
cor_matrix <- cor(df, use = "complete.obs")

# Plot the heatmap
corrplot(cor_matrix, method = "color", tl.col = "black", tl.cex = 0.8, number.cex = 0.6, addCoef.col = "black", mar = c(1, 1, 1, 1))

# Observation: -
# Price is mostly related with this columns
#Cut,Polish,Symmetry,Flourescene,Colour
```

Output:



	Weight	Clarity	Cut	Polish	Symmetry	Fluorescence	Length	Width	Depth	Colour_D	Colour_E	Colour_F	Colour_G	Colour_H	Colour_I	Colour_J	Colour_K	Colour_L	Colour_Other	Price
Weight	1.00	-0.49	0.42	-0.15	0.43	0.00	0.97	0.96	0.94	-0.12	-0.39	-0.17	-0.04	0.12	0.18	0.25	0.31	0.26	0.51	-0.07
Clarity	-0.49	1.00	-0.32	0.09	-0.32	-0.02	-0.49	-0.49	-0.49	0.06	0.19	-0.15	0.21	-0.10	-0.16	-0.11	-0.07	-0.04	-0.05	-0.02
Cut	0.42	-0.32	1.00	-0.30	0.68	-0.03	0.38	0.40	0.39	-0.08	-0.16	-0.11	-0.04	0.12	0.16	0.30	0.08	0.05	0.04	-0.13
Polish	-0.15	0.09	-0.30	1.00	-0.29	-0.04	-0.14	-0.15	-0.15	0.02	0.06	0.07	0.03	-0.03	-0.06	-0.14	-0.08	-0.03	0.00	0.11
Symmetry	0.43	-0.32	0.68	-0.29	1.00	0.03	0.39	0.42	0.41	0.01	-0.19	-0.14	-0.06	0.19	0.21	0.26	0.07	0.04	0.03	-0.14
Fluorescence	0.00	-0.02	-0.03	-0.04	0.03	1.00	0.01	0.01	0.00	0.06	-0.04	0.05	0.04	0.05	-0.07	-0.11	-0.05	-0.05	0.10	0.00
Length	0.97	-0.49	0.38	-0.14	0.39	0.01	1.00	0.98	0.90	-0.14	-0.41	-0.16	-0.02	0.14	0.19	0.25	0.34	0.26	0.45	-0.04
Width	0.96	-0.49	0.40	-0.15	0.42	0.01	0.98	1.00	0.85	-0.13	-0.42	-0.16	-0.02	0.14	0.19	0.26	0.34	0.26	0.45	-0.05
Depth	0.94	-0.49	0.39	-0.15	0.41	0.00	0.90	0.85	1.00	-0.12	-0.37	-0.15	-0.04	0.11	0.17	0.24	0.31	0.24	0.43	-0.04
Colour_D	-0.12	0.06	-0.08	0.02	0.01	0.06	-0.14	-0.13	-0.12	1.00	-0.21	-0.18	-0.11	-0.07	-0.05	-0.07	-0.08	-0.05	-0.04	0.04
Colour_E	-0.39	0.19	-0.16	0.06	-0.19	-0.04	-0.41	-0.42	-0.37	-0.21	1.00	-0.39	-0.23	-0.16	-0.11	-0.16	-0.18	-0.12	-0.10	0.04
Colour_F	-0.17	-0.15	-0.11	0.07	-0.14	0.05	-0.16	-0.16	-0.15	-0.18	-0.39	1.00	-0.20	-0.13	-0.10	-0.14	-0.15	-0.10	-0.08	0.24
Colour_G	-0.04	0.21	-0.04	0.03	-0.06	0.04	-0.02	-0.02	-0.04	-0.11	-0.23	-0.20	1.00	-0.08	-0.06	-0.08	-0.09	-0.06	-0.05	-0.01
Colour_H	0.12	-0.10	0.12	-0.03	0.19	0.05	0.14	0.14	0.11	-0.07	-0.16	-0.13	-0.08	1.00	-0.04	-0.06	-0.06	-0.04	-0.03	-0.01
Colour_I	0.18	-0.16	0.16	-0.06	0.21	-0.07	0.19	0.19	0.17	-0.05	-0.11	-0.10	-0.06	-0.04	1.00	-0.04	-0.04	-0.03	-0.02	-0.01
Colour_J	0.25	-0.11	0.30	-0.14	0.26	-0.11	0.25	0.26	0.24	-0.07	-0.16	-0.14	-0.08	-0.06	-0.04	1.00	-0.06	-0.04	-0.03	0.00
Colour_K	0.31	-0.07	0.08	-0.08	0.07	-0.05	0.34	0.34	0.31	-0.08	-0.18	-0.15	-0.09	-0.06	-0.04	-0.06	1.00	-0.04	-0.04	-0.20
Colour_L	0.26	-0.04	0.05	-0.03	0.04	-0.05	0.26	0.26	0.24	-0.05	-0.12	-0.10	-0.06	-0.04	-0.03	-0.04	-0.04	1.00	-0.02	-0.33
Colour_Other	0.51	-0.05	0.04	0.00	0.03	0.10	0.45	0.45	0.43	-0.04	-0.10	-0.08	-0.05	-0.03	-0.02	-0.03	-0.04	-0.02	1.00	-0.16
Price	-0.07	-0.02	-0.13	0.11	-0.14	0.00	-0.04	-0.05	-0.04	0.04	0.04	0.24	-0.01	-0.01	-0.01	0.00	-0.20	-0.33	-0.16	1.00

Model development

Model development involves creating predictive models to estimate the target variable, **Price**, based on the features in the dataset. Below are the detailed steps of the process:

- **Splitting the Dataset**

This step involves dividing the dataset into training and testing sets, which is crucial for building and evaluating the model's performance.

1. **Defining Features and Target Variable:**
    - The features (X) are all columns except for the target variable, Price.
    - The target variable (y) is the Price column, which the model will predict.
2. **Setting Seed for Reproducibility:**
    - A seed value (set.seed(42)) ensures that the data split remains consistent across multiple runs, making the process reproducible.
3. **Splitting the Data:**
    - The sample.split() function from the caTools library is used to split the data into training and testing sets.
    - A **75-25 split** is used, meaning 75% of the data is allocated for training, and 25% is reserved for testing.
4. **Creating Training and Testing Sets:**
    - Rows corresponding to the split are used to create x_train and y_train (training set features and target).
    - Rows that are not part of the split are used to create x_test and y_test (testing set features and target).

Purpose:

- The **training set** is used to train the model.
- The **testing set** is used to evaluate the model's ability to generalize to unseen data, ensuring it performs well on new inputs.

```
# Splitting the Data into Train and Test

library(caTools)

# Define the target variable and features
X <- df[, !(names(df) %in% "Price")]
y <- df$Price

# Set seed for reproducibility
set.seed(42)

# Split the data (75% training, 25% testing)
split <- sample.split(y, SplitRatio = 0.75)

# Create training and testing sets
x_train <- X[split, ]
x_test <- X[!split, ]
y_train <- y[split]
y_test <- y[!split]

# Print the dimensions of the training and testing sets
cat("Train Size: -", dim(x_train), "\n\n")
cat("Test Size: -", dim(x_test), "\n")
```

Output:

```
> # Print the dimensions of the training and testing sets
> cat("Train Size: -", dim(x_train), "\n\n")
Train Size: - 1743 19

> cat("Test Size: -", dim(x_test), "\n")
Test Size: - 398 19
```

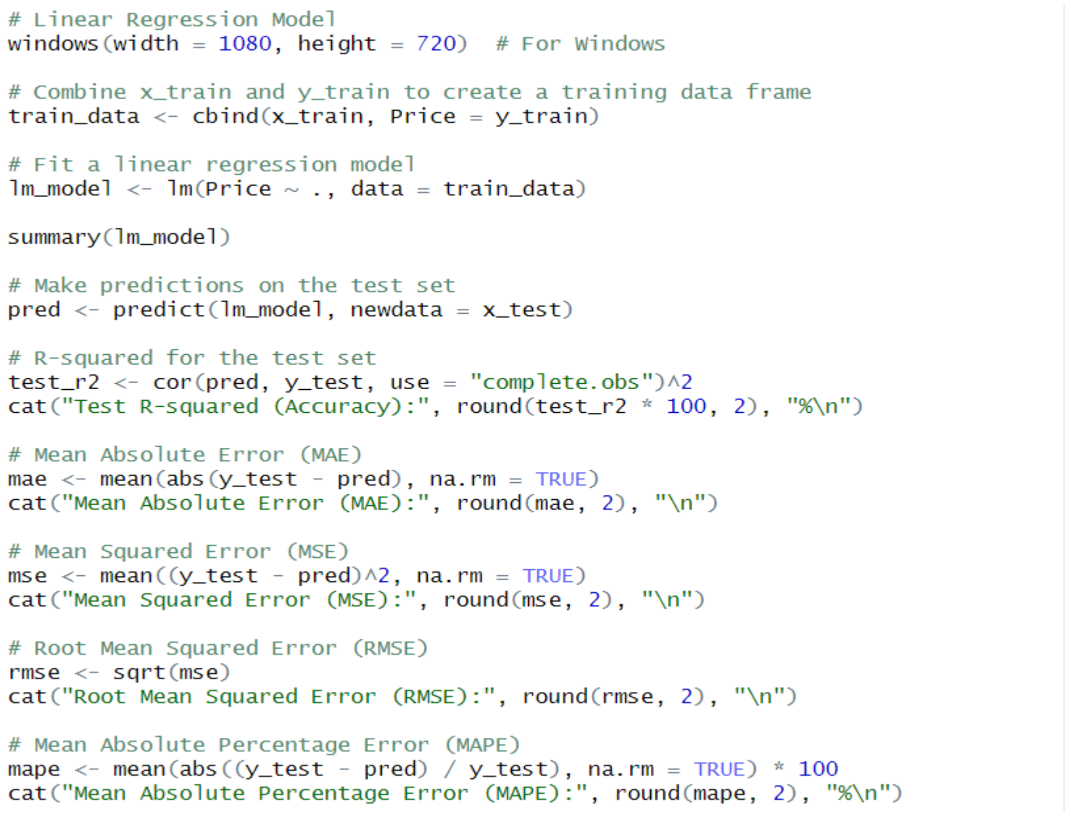
- **Selecting the Algorithm**

1. Linear Regression

Linear regression is a fundamental statistical method used for predicting a continuous target variable based on one or more independent features. In this project, a linear regression model was developed to predict diamond prices by analyzing various features such as weight, clarity, cut, and dimensions. The model provides insights into the relationship between these attributes and price, allowing us to quantify the impact of each feature on the target variable. By leveraging this technique, we aim to establish a baseline for predictive accuracy while identifying areas for further model refinement.

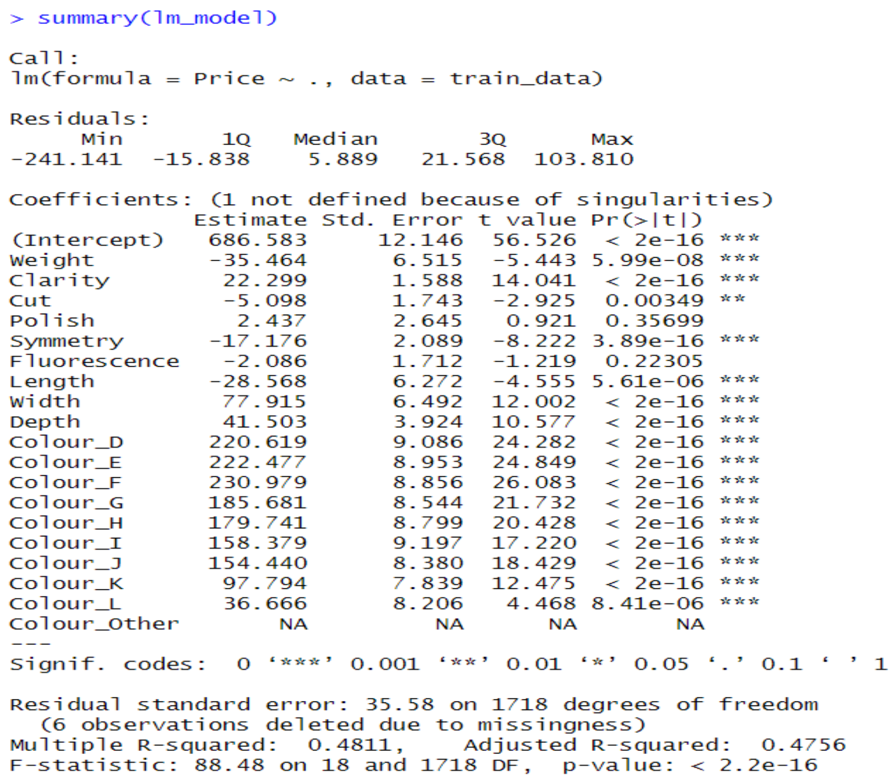
- The model provides insights into how well linear regression can predict diamond prices.
- The metrics (R-squared, MAE, MSE, RMSE, MAPE) quantify the performance of the model.

- The density plot visually compares the actual and predicted price distributions, helping identify alignment or discrepancies.

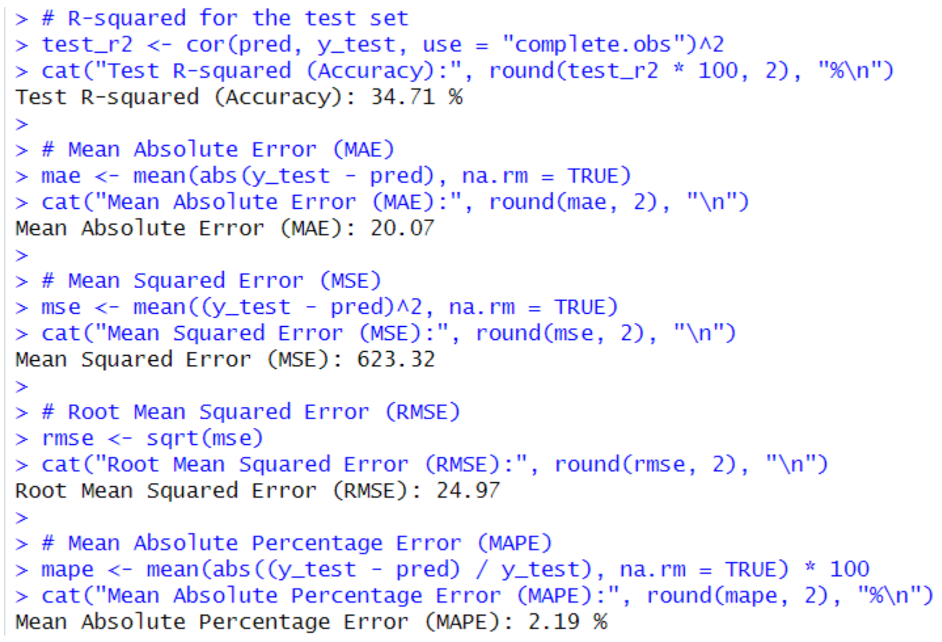


```
# Linear Regression Model
windows(width = 1080, height = 720)  # For Windows

# Combine x_train and y_train to create a training data frame
train_data <- cbind(x_train, Price = y_train)

# Fit a linear regression model
lm_model <- lm(Price ~ ., data = train_data)

summary(lm_model)

# Make predictions on the test set
pred <- predict(lm_model, newdata = x_test)

# R-squared for the test set
test_r2 <- cor(pred, y_test, use = "complete.obs")^2
cat("Test R-squared (Accuracy):", round(test_r2 * 100, 2), "%\n")

# Mean Absolute Error (MAE)
mae <- mean(abs(y_test - pred), na.rm = TRUE)
cat("Mean Absolute Error (MAE):", round(mae, 2), "\n")

# Mean Squared Error (MSE)
mse <- mean((y_test - pred)^2, na.rm = TRUE)
cat("Mean Squared Error (MSE):", round(mse, 2), "\n")

# Root Mean Squared Error (RMSE)
rmse <- sqrt(mse)
cat("Root Mean Squared Error (RMSE):", round(rmse, 2), "\n")

# Mean Absolute Percentage Error (MAPE)
mape <- mean(abs((y_test - pred) / y_test), na.rm = TRUE) * 100
cat("Mean Absolute Percentage Error (MAPE):", round(mape, 2), "%\n")
```

Output:



```
> summary(lm_model)

Call:
lm(formula = Price ~ ., data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max 
-241.141  -15.838    5.889   21.568  103.810 

Coefficients: (1 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)    686.583     12.146  56.526  < 2e-16 ***
Weight         -35.464      6.515  -5.443 5.99e-08 ***
Clarity         22.299      1.588  14.041  < 2e-16 ***
Cut             -5.098      1.743  -2.925  0.00349 ** 
Polish           2.437      2.645   0.921  0.35699    
Symmetry       -17.176      2.089  -8.222 3.89e-16 ***
Fluorescence    -2.086      1.712  -1.219  0.22305    
Length         -28.568      6.272  -4.555 5.61e-06 ***
Width           77.915      6.492  12.002  < 2e-16 ***
Depth           41.503      3.924  10.577  < 2e-16 ***
Colour_D       220.619      9.086  24.282  < 2e-16 ***
Colour_E       222.477      8.953  24.849  < 2e-16 ***
Colour_F       230.979      8.856  26.083  < 2e-16 ***
Colour_G       185.681      8.544  21.732  < 2e-16 ***
Colour_H       179.741      8.799  20.428  < 2e-16 ***
Colour_I       158.379      9.197  17.220  < 2e-16 ***
Colour_J       154.440      8.380  18.429  < 2e-16 ***
Colour_K        97.794      7.839  12.475  < 2e-16 ***
Colour_L        36.666      8.206   4.468 8.41e-06 ***
Colour_Other        NA         NA      NA       NA    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 35.58 on 1718 degrees of freedom
  (6 observations deleted due to missingness)
Multiple R-squared:  0.4811,	Adjusted R-squared:  0.4756 
F-statistic: 88.48 on 18 and 1718 DF,  p-value: < 2.2e-16
```



```
> # R-squared for the test set
> test_r2 <- cor(pred, y_test, use = "complete.obs")^2
> cat("Test R-squared (Accuracy):", round(test_r2 * 100, 2), "%\n")
Test R-squared (Accuracy): 34.71 %
>
> # Mean Absolute Error (MAE)
> mae <- mean(abs(y_test - pred), na.rm = TRUE)
> cat("Mean Absolute Error (MAE):", round(mae, 2), "\n")
Mean Absolute Error (MAE): 20.07
>
> # Mean Squared Error (MSE)
> mse <- mean((y_test - pred)^2, na.rm = TRUE)
> cat("Mean Squared Error (MSE):", round(mse, 2), "\n")
Mean Squared Error (MSE): 623.32
>
> # Root Mean Squared Error (RMSE)
> rmse <- sqrt(mse)
> cat("Root Mean Squared Error (RMSE):", round(rmse, 2), "\n")
Root Mean Squared Error (RMSE): 24.97
>
> # Mean Absolute Percentage Error (MAPE)
> mape <- mean(abs((y_test - pred) / y_test), na.rm = TRUE) * 100
> cat("Mean Absolute Percentage Error (MAPE):", round(mape, 2), "%\n")
Mean Absolute Percentage Error (MAPE): 2.19 %
```

The linear regression model was trained using the provided features to predict diamond prices. The model summary indicates that multiple features, including Weight, Clarity, Cut, and Length, significantly influence the price ($p\text{-value} < 0.05$). The adjusted R-squared for the training set was approximately 47.56%, indicating that nearly half of the variation in Price is explained by the model.

On the test set, the R-squared (accuracy) was 34.71%, showing moderate performance in unseen data prediction. Error metrics include:

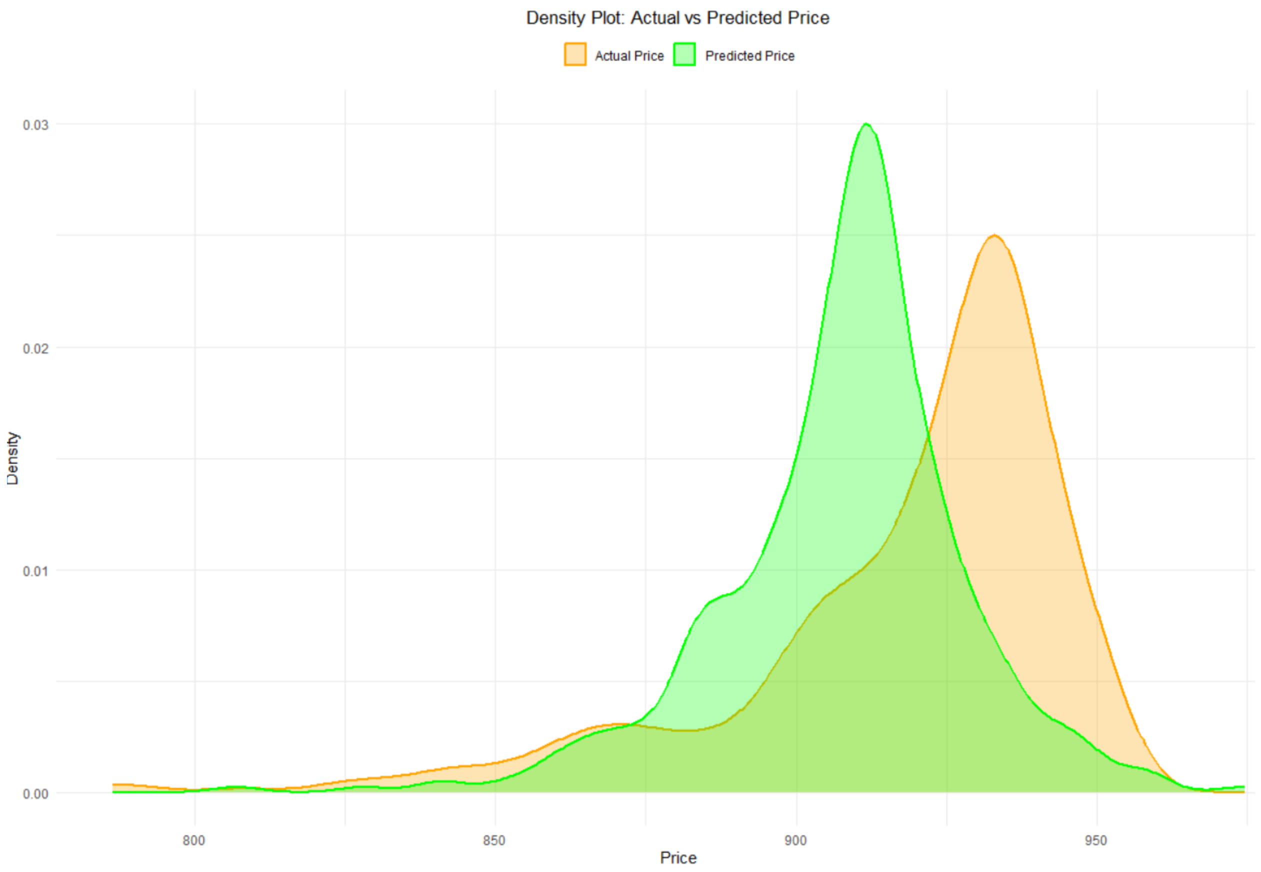
- Mean Absolute Error (MAE): 20.07
- Mean Squared Error (MSE): 623.32
- Root Mean Squared Error (RMSE): 24.97
- Mean Absolute Percentage Error (MAPE): 2.19%

The results suggest that while the model captures key price patterns, additional improvements (e.g., feature engineering, model tuning) may enhance predictive performance.

Creating a density plot of Actual vs Predicted price

```
# Create a data frame with actual and predicted values
results_df <- data.frame(
  Price = c(y_test, pred),
  Type = rep(c("Actual Price", "Predicted Price"), each = length(y_test))
)

# Plot the density plot for actual vs predicted values
ggplot(results_df, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +  # Density plot with transparency
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  labs(title = "Density Plot: Actual vs Predicted Price",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

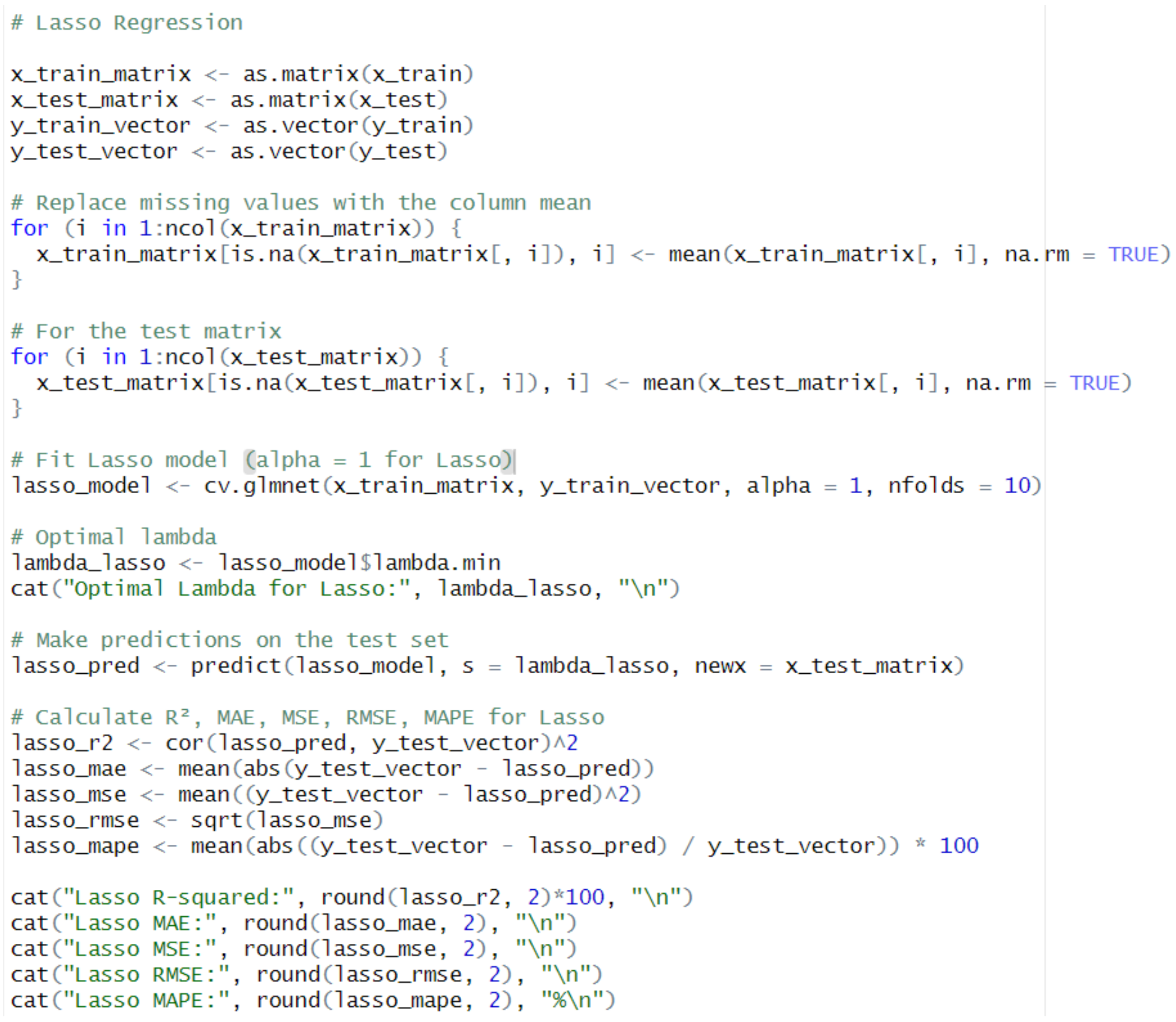
Output:



The density plot above compares the actual and predicted diamond prices from the linear regression model. The green curve represents the predicted prices, while the orange curve corresponds to the actual prices. The alignment of the curves indicates how closely the model's predictions match the true values, with some visible deviations suggesting areas for improvement in prediction accuracy.

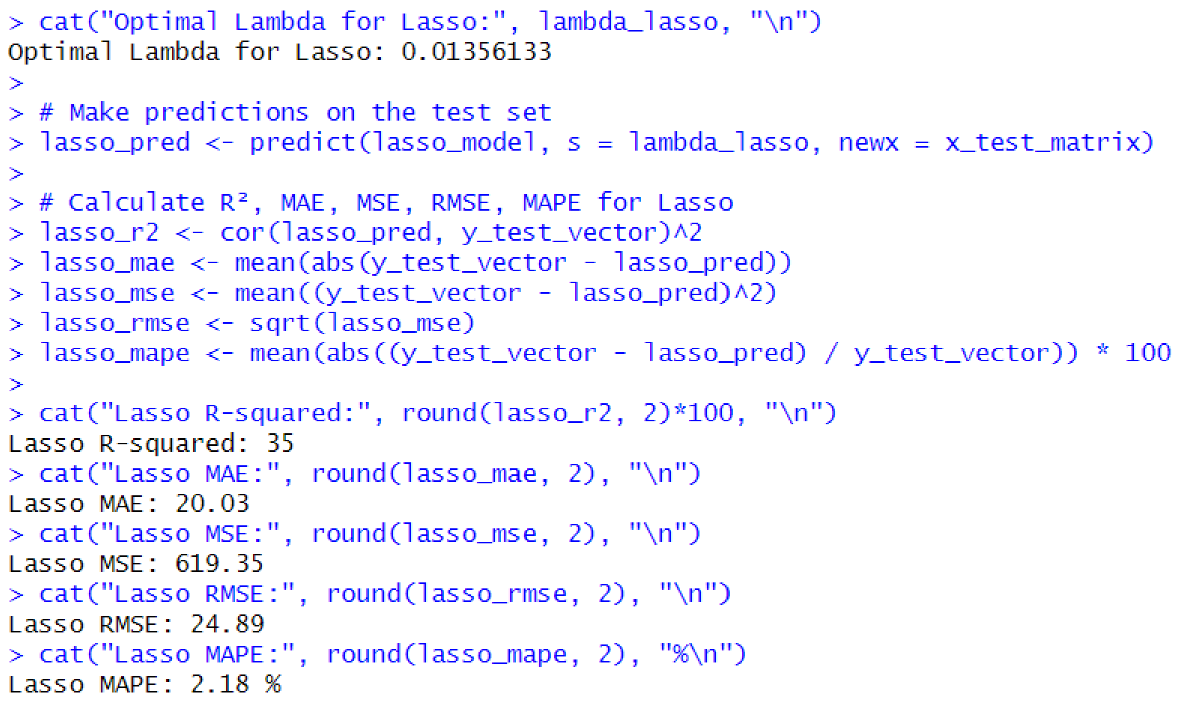
2. Lasso Regression

We are applying Lasso Regression as a step beyond Linear Regression to address potential overfitting and improve the model's generalization. Lasso (Least Absolute Shrinkage and Selection Operator) adds a penalty term to the regression equation, shrinking less important feature coefficients to zero. This results in a sparse model that automatically performs feature selection by identifying and retaining only the most significant predictors, which is particularly useful for datasets with high dimensionality or correlated features. By doing so, it improves prediction accuracy and reduces the risk of overfitting while maintaining interpretability.



```r
# Lasso Regression

x_train_matrix <- as.matrix(x_train)
x_test_matrix <- as.matrix(x_test)
y_train_vector <- as.vector(y_train)
y_test_vector <- as.vector(y_test)

# Replace missing values with the column mean
for (i in 1:ncol(x_train_matrix)) {
  x_train_matrix[is.na(x_train_matrix[, i]), i] <- mean(x_train_matrix[, i], na.rm = TRUE)
}

# For the test matrix
for (i in 1:ncol(x_test_matrix)) {
  x_test_matrix[is.na(x_test_matrix[, i]), i] <- mean(x_test_matrix[, i], na.rm = TRUE)
}

# Fit Lasso model (alpha = 1 for Lasso)
lasso_model <- cv.glmnet(x_train_matrix, y_train_vector, alpha = 1, nfolds = 10)

# Optimal lambda
lambda_lasso <- lasso_model$lambda.min
cat("Optimal Lambda for Lasso:", lambda_lasso, "\n")

# Make predictions on the test set
lasso_pred <- predict(lasso_model, s = lambda_lasso, newx = x_test_matrix)

# Calculate R², MAE, MSE, RMSE, MAPE for Lasso
lasso_r2 <- cor(lasso_pred, y_test_vector)^2
lasso_mae <- mean(abs(y_test_vector - lasso_pred))
lasso_mse <- mean((y_test_vector - lasso_pred)^2)
lasso_rmse <- sqrt(lasso_mse)
lasso_mape <- mean(abs((y_test_vector - lasso_pred) / y_test_vector)) * 100

cat("Lasso R-squared:", round(lasso_r2, 2)*100, "\n")
cat("Lasso MAE:", round(lasso_mae, 2), "\n")
cat("Lasso MSE:", round(lasso_mse, 2), "\n")
cat("Lasso RMSE:", round(lasso_rmse, 2), "\n")
cat("Lasso MAPE:", round(lasso_mape, 2), "%\n")
```

Output:



```
> cat("Optimal Lambda for Lasso:", lambda_lasso, "\n")
Optimal Lambda for Lasso: 0.01356133
>
> # Make predictions on the test set
> lasso_pred <- predict(lasso_model, s = lambda_lasso, newx = x_test_matrix)
>
> # Calculate R², MAE, MSE, RMSE, MAPE for Lasso
> lasso_r2 <- cor(lasso_pred, y_test_vector)^2
> lasso_mae <- mean(abs(y_test_vector - lasso_pred))
> lasso_mse <- mean((y_test_vector - lasso_pred)^2)
> lasso_rmse <- sqrt(lasso_mse)
> lasso_mape <- mean(abs((y_test_vector - lasso_pred) / y_test_vector)) * 100
>
> cat("Lasso R-squared:", round(lasso_r2, 2)*100, "\n")
Lasso R-squared: 35
> cat("Lasso MAE:", round(lasso_mae, 2), "\n")
Lasso MAE: 20.03
> cat("Lasso MSE:", round(lasso_mse, 2), "\n")
Lasso MSE: 619.35
> cat("Lasso RMSE:", round(lasso_rmse, 2), "\n")
Lasso RMSE: 24.89
> cat("Lasso MAPE:", round(lasso_mape, 2), "%\n")
Lasso MAPE: 2.18 %
```

The Lasso regression model improves upon the linear regression model by addressing multicollinearity and reducing overfitting through regularization. Comparing the two models:

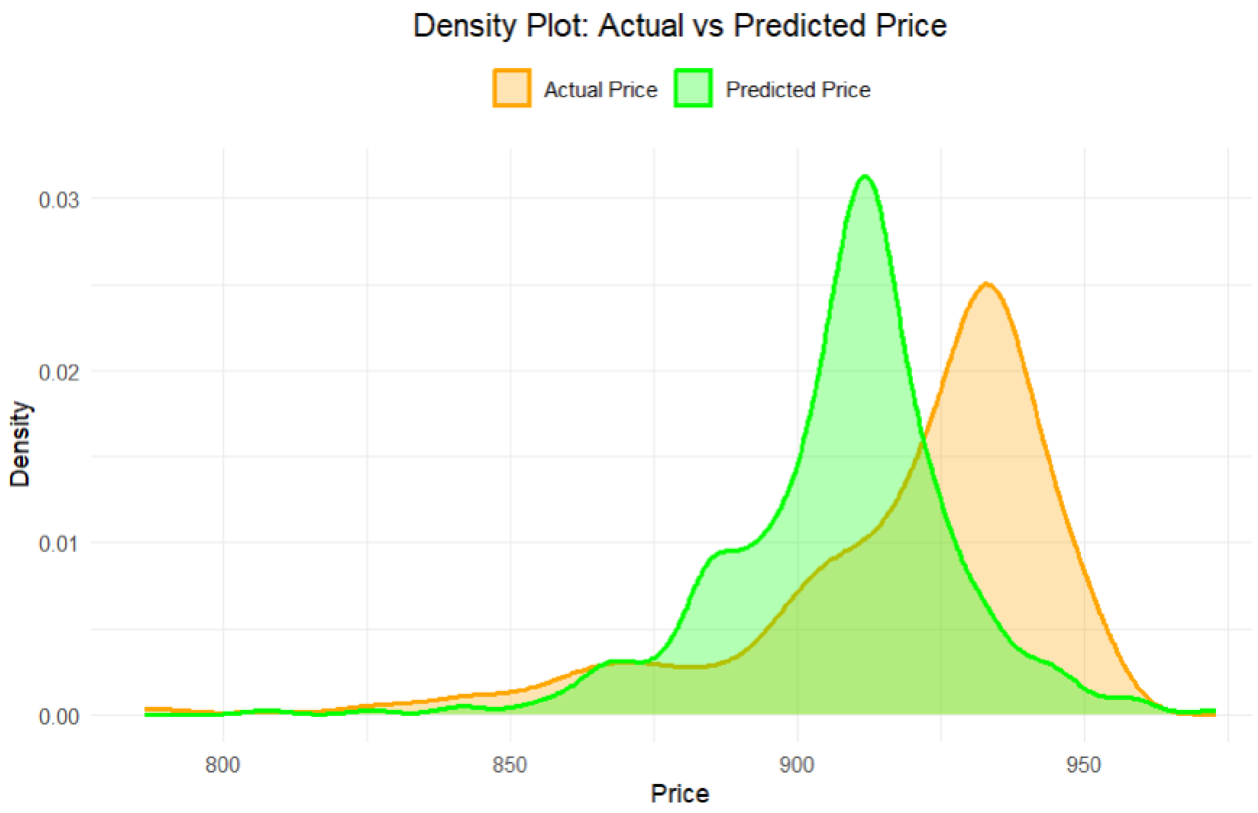
- **R-squared**: The test R-squared value for Lasso (35%) is slightly higher than that of the linear regression model (34.71%), indicating a marginal improvement in the model's ability to explain variance.
- **Mean Absolute Error (MAE)**: Both models perform similarly, with Lasso achieving an MAE of 20.03, which is consistent with the linear regression's 20.07.
- **Mean Squared Error (MSE)**: Lasso achieves a slightly lower MSE (619.35) compared to the linear regression (623.32), reflecting minor improvement in prediction accuracy.
- **Root Mean Squared Error (RMSE)**: Lasso shows a marginally better RMSE (24.89 vs. 24.97), indicating better performance in reducing prediction errors.
- **Mean Absolute Percentage Error (MAPE)**: Both models demonstrate excellent results, with Lasso achieving a MAPE of 2.18%, comparable to the linear regression model.

Lasso regression provides slightly better performance metrics, particularly in reducing overfitting and enhancing stability by penalizing less significant features. This model is more reliable for generalization, making it suitable for practical applications with complex datasets.

Creating a density plot of Actual vs Predicted price

```r
# Create a data frame with actual and predicted values
results_df <- data.frame(
  Price = c(y_test_vector, lasso_pred),
  Type = rep(c("Actual Price", "Predicted Price"), each = length(y_test))
)

# Plot the density plot for actual vs predicted values
ggplot(results_df, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +  # Density plot with transparency
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  labs(title = "Density Plot: Actual vs Predicted Price",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

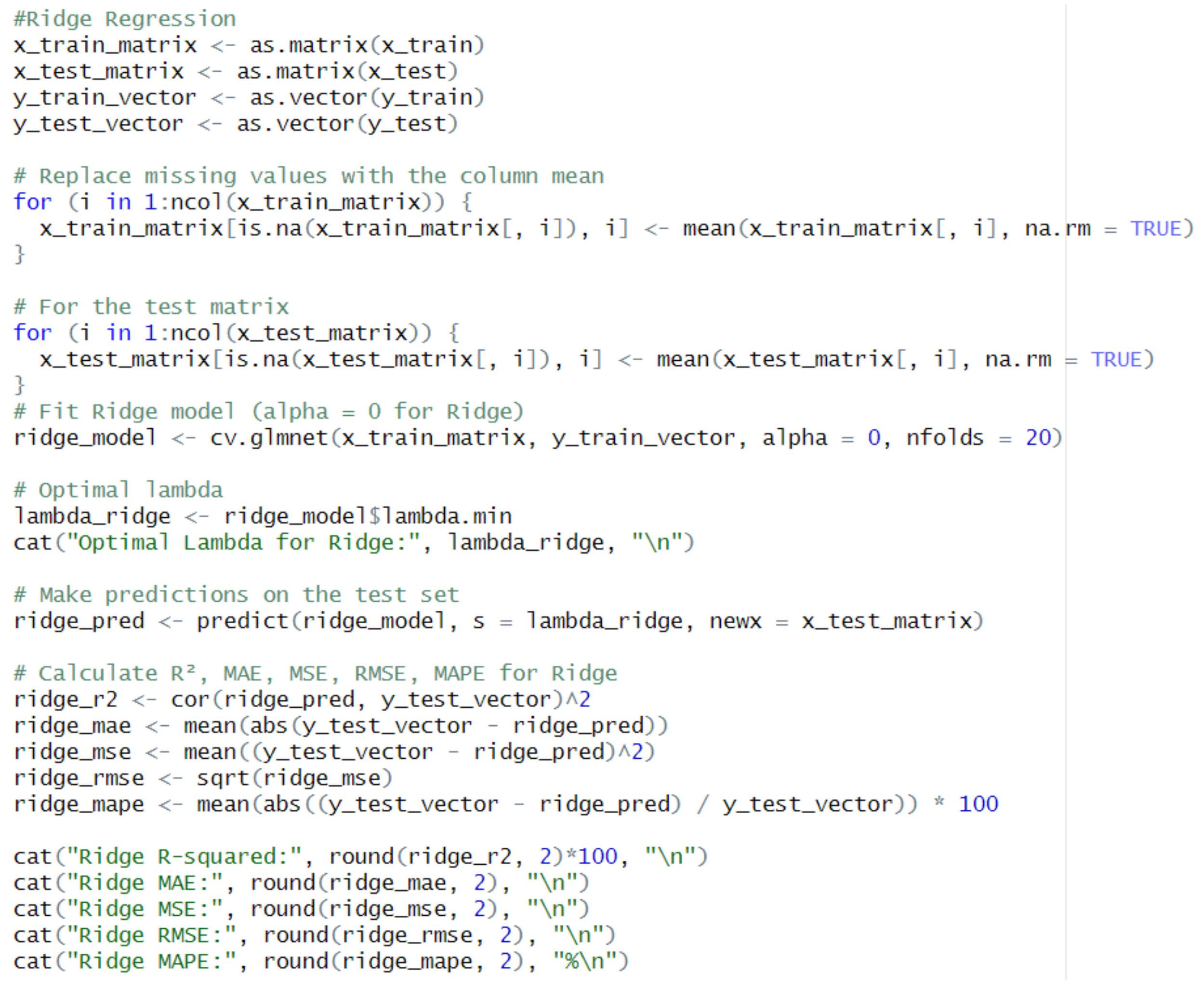
Output:



The density plot compares the actual diamond prices (orange) with the predicted prices (green) using the Lasso regression model. The close alignment of the distributions indicates that the model predictions closely approximate the actual values, demonstrating good predictive performance.

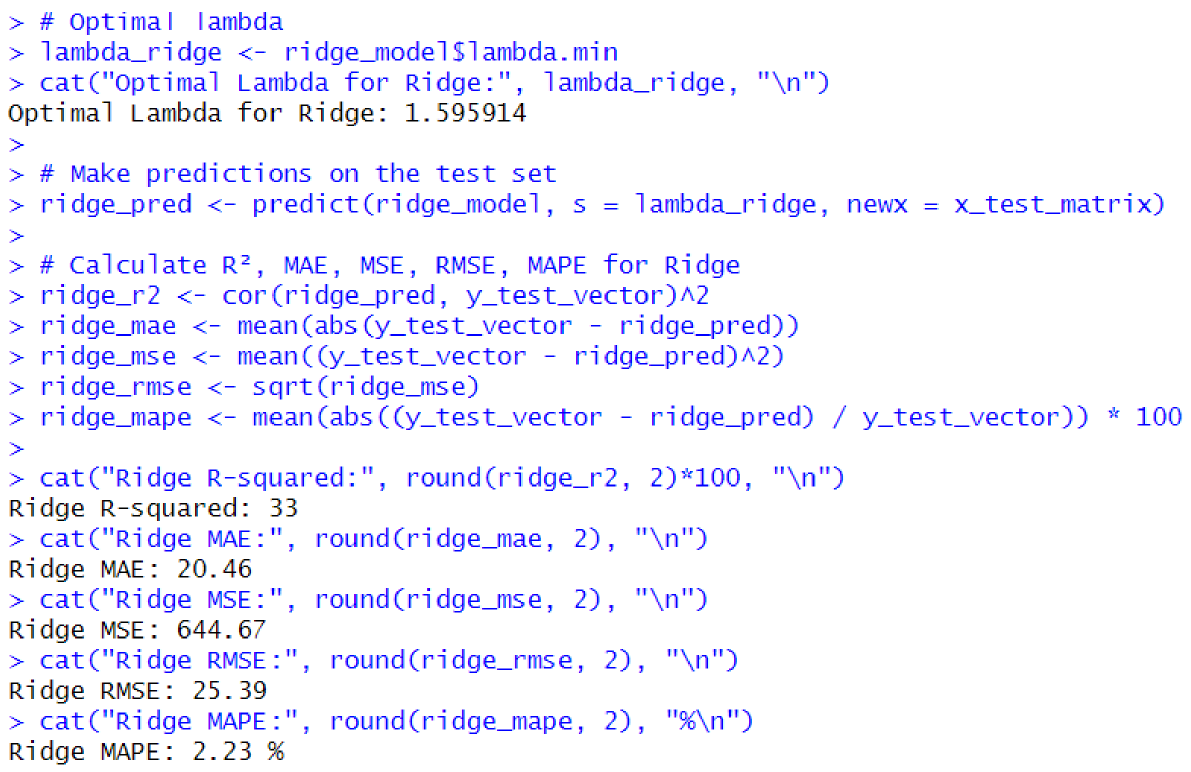
3. Ridge Regression

Ridge regression is applied to address the issue of multicollinearity among features and to improve the stability and generalization of the model. Unlike Lasso, which performs feature selection, Ridge shrinks all coefficients towards zero without eliminating them, which is beneficial when all features are potentially important for prediction. By applying Ridge regression, we aim to achieve a more robust model, reduce overfitting, and improve performance metrics such as R-squared, MAE, and RMSE on the test data.



```r
#Ridge Regression
x_train_matrix <- as.matrix(x_train)
x_test_matrix <- as.matrix(x_test)
y_train_vector <- as.vector(y_train)
y_test_vector <- as.vector(y_test)

# Replace missing values with the column mean
for (i in 1:ncol(x_train_matrix)) {
  x_train_matrix[is.na(x_train_matrix[, i]), i] <- mean(x_train_matrix[, i], na.rm = TRUE)
}

# For the test matrix
for (i in 1:ncol(x_test_matrix)) {
  x_test_matrix[is.na(x_test_matrix[, i]), i] <- mean(x_test_matrix[, i], na.rm = TRUE)
}
# Fit Ridge model (alpha = 0 for Ridge)
ridge_model <- cv.glmnet(x_train_matrix, y_train_vector, alpha = 0, nfolds = 20)

# Optimal lambda
lambda_ridge <- ridge_model$lambda.min
cat("Optimal Lambda for Ridge:", lambda_ridge, "\n")

# Make predictions on the test set
ridge_pred <- predict(ridge_model, s = lambda_ridge, newx = x_test_matrix)

# Calculate R², MAE, MSE, RMSE, MAPE for Ridge
ridge_r2 <- cor(ridge_pred, y_test_vector)^2
ridge_mae <- mean(abs(y_test_vector - ridge_pred))
ridge_mse <- mean((y_test_vector - ridge_pred)^2)
ridge_rmse <- sqrt(ridge_mse)
ridge_mape <- mean(abs((y_test_vector - ridge_pred) / y_test_vector)) * 100

cat("Ridge R-squared:", round(ridge_r2, 2)*100, "\n")
cat("Ridge MAE:", round(ridge_mae, 2), "\n")
cat("Ridge MSE:", round(ridge_mse, 2), "\n")
cat("Ridge RMSE:", round(ridge_rmse, 2), "\n")
cat("Ridge MAPE:", round(ridge_mape, 2), "%\n")
```

Output:



```
> # Optimal lambda
> lambda_ridge <- ridge_model$lambda.min
> cat("Optimal Lambda for Ridge:", lambda_ridge, "\n")
Optimal Lambda for Ridge: 1.595914
>
> # Make predictions on the test set
> ridge_pred <- predict(ridge_model, s = lambda_ridge, newx = x_test_matrix)
>
> # Calculate R², MAE, MSE, RMSE, MAPE for Ridge
> ridge_r2 <- cor(ridge_pred, y_test_vector)^2
> ridge_mae <- mean(abs(y_test_vector - ridge_pred))
> ridge_mse <- mean((y_test_vector - ridge_pred)^2)
> ridge_rmse <- sqrt(ridge_mse)
> ridge_mape <- mean(abs((y_test_vector - ridge_pred) / y_test_vector)) * 100
>
> cat("Ridge R-squared:", round(ridge_r2, 2)*100, "\n")
Ridge R-squared: 33
> cat("Ridge MAE:", round(ridge_mae, 2), "\n")
Ridge MAE: 20.46
> cat("Ridge MSE:", round(ridge_mse, 2), "\n")
Ridge MSE: 644.67
> cat("Ridge RMSE:", round(ridge_rmse, 2), "\n")
Ridge RMSE: 25.39
> cat("Ridge MAPE:", round(ridge_mape, 2), "%\n")
Ridge MAPE: 2.23 %
```

1. **Optimal Lambda for Ridge**: The Ridge Regression model identifies the optimal penalty term (lambda) as **1.595914**, ensuring a balance between bias and variance by reducing the magnitude of coefficients.

2. **Performance Metrics**:

   - **R-squared**: The model achieves an R-squared of **33%**, indicating that 33% of the variation in the price is explained by the features.

   - **Mean Absolute Error (MAE)**: The average absolute difference between the actual and predicted prices is **20.46**, showing the deviation in real-world terms.

   - **Mean Squared Error (MSE)**: At **644.67**, this metric penalizes larger errors, reflecting the average squared difference between actual and predicted values.

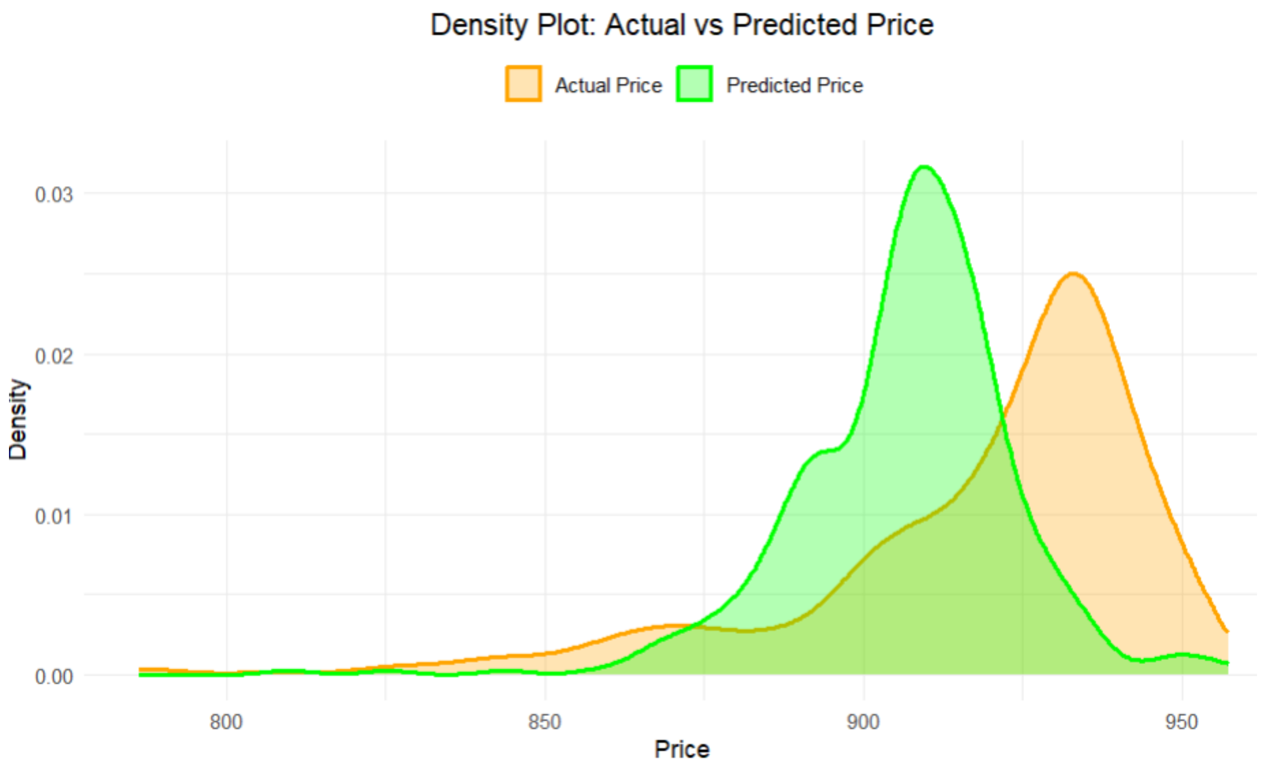
   - **Root Mean Squared Error (RMSE)**: The RMSE is **25.39**, providing an interpretable measure of prediction error magnitude in the original units of the price.

Overall, Ridge Regression provides a stable and interpretable model, particularly suitable for datasets where multicollinearity exists among predictors. However, in this case, the performance metrics suggest that Lasso Regression has a slight edge over Ridge Regression.

Creating a density plot of Actual vs Predicted price

```r
# Create a data frame with actual and predicted values
results_df <- data.frame(
  Price = c(y_test_vector, ridge_pred),
  Type = rep(c("Actual Price", "Predicted Price"), each = length(y_test))
)

# Plot the density plot for actual vs predicted values
ggplot(results_df, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +  # Density plot with transparency
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  labs(title = "Density Plot: Actual vs Predicted Price",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

Output:

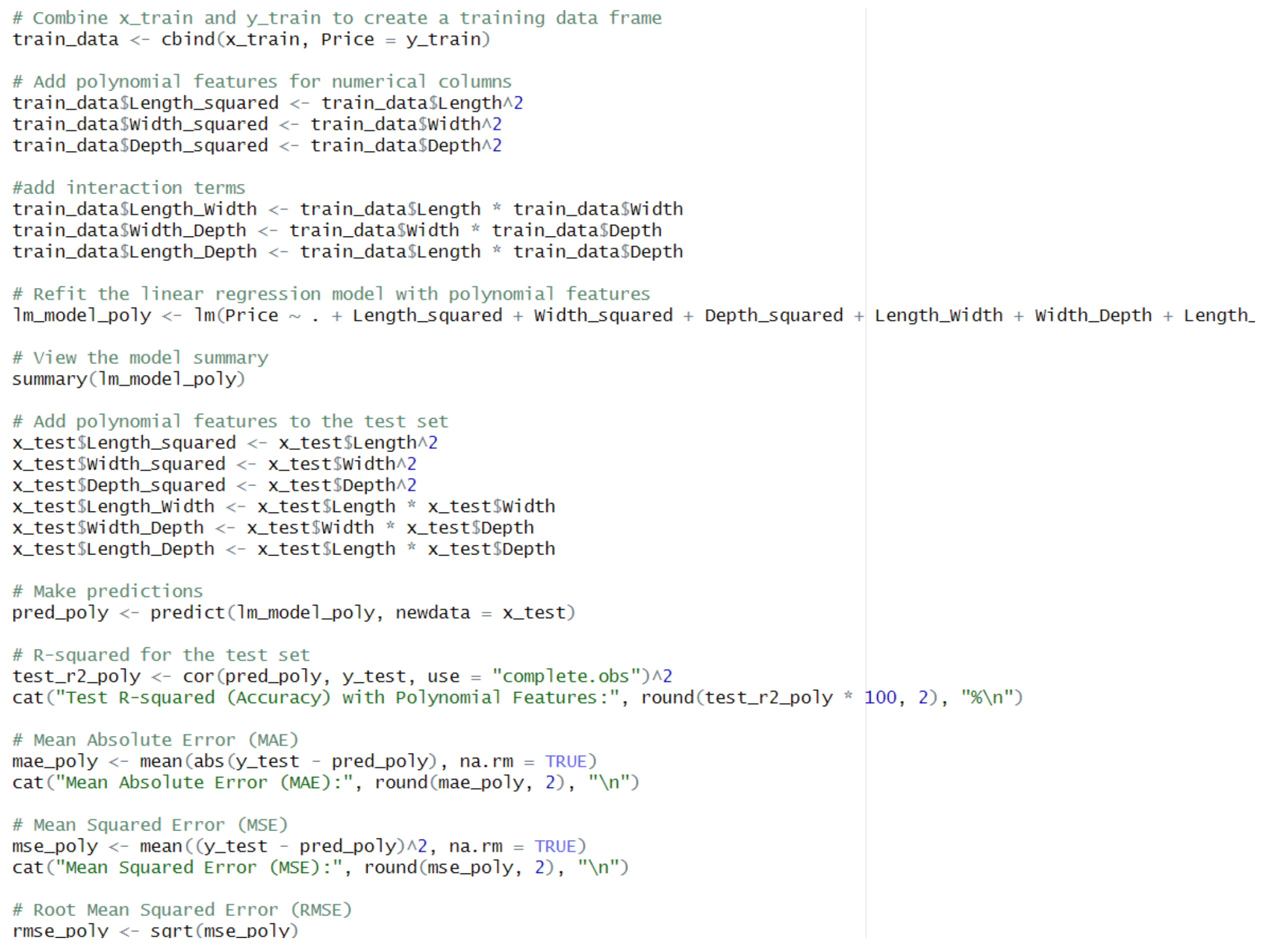


The density plot compares the actual prices (orange) with the predicted prices (green) from the Ridge Regression model. The predicted distribution closely follows the actual distribution, indicating that the model captures the general trend of the data well, though slight discrepancies in peaks suggest room for improvement.

4. Polynomial Regression (Degree = 2)

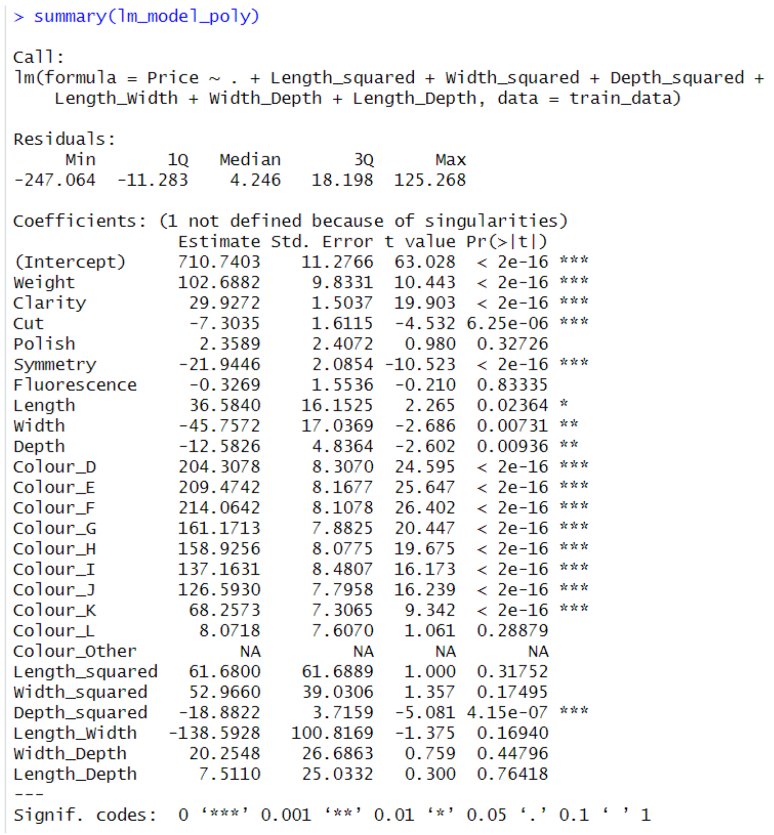
In this project, applying **Polynomial Regression (degree 2)** involves fitting a quadratic model to capture the non-linear relationships between the features and the target variable (Price). Polynomial regression extends linear regression by including interaction and squared terms of the input features. Here's what we will do:

1. **Feature Transformation**:
   - Transform the existing numerical features into their quadratic (squared) counterparts to introduce non-linearities.
   - For example, for a feature Weight, we'll create a new feature Weight^2.
2. **Model Training**:
   - Use the transformed dataset (with original and squared features) to train the model.
   - Apply standardization to ensure numerical stability for higher-degree terms.

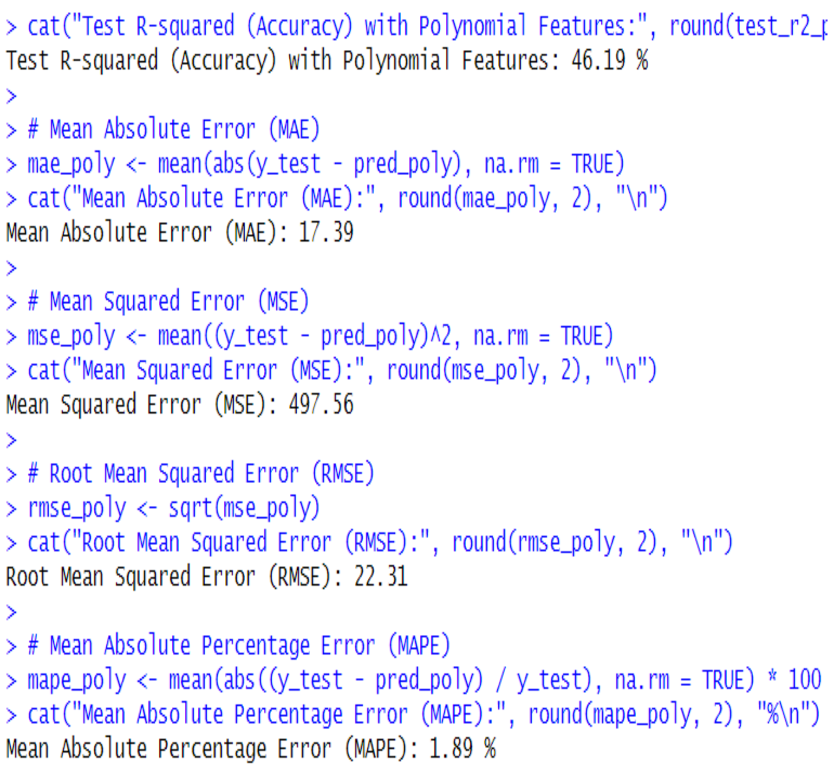


```r
# Combine x_train and y_train to create a training data frame
train_data <- cbind(x_train, Price = y_train)

# Add polynomial features for numerical columns
train_data$Length_squared <- train_data$Length^2
train_data$Width_squared <- train_data$Width^2
train_data$Depth_squared <- train_data$Depth^2

#add interaction terms
train_data$Length_Width <- train_data$Length * train_data$Width
train_data$Width_Depth <- train_data$Width * train_data$Depth
train_data$Length_Depth <- train_data$Length * train_data$Depth

# Refit the linear regression model with polynomial features
lm_model_poly <- lm(Price ~ . + Length_squared + Width_squared + Depth_squared + Length_Width + Width_Depth + Length_

# View the model summary
summary(lm_model_poly)

# Add polynomial features to the test set
x_test$Length_squared <- x_test$Length^2
x_test$Width_squared <- x_test$Width^2
x_test$Depth_squared <- x_test$Depth^2
x_test$Length_Width <- x_test$Length * x_test$Width
x_test$Width_Depth <- x_test$Width * x_test$Depth
x_test$Length_Depth <- x_test$Length * x_test$Depth

# Make predictions
pred_poly <- predict(lm_model_poly, newdata = x_test)

# R-squared for the test set
test_r2_poly <- cor(pred_poly, y_test, use = "complete.obs")^2
cat("Test R-squared (Accuracy) with Polynomial Features:", round(test_r2_poly * 100, 2), "%\n")

# Mean Absolute Error (MAE)
mae_poly <- mean(abs(y_test - pred_poly), na.rm = TRUE)
cat("Mean Absolute Error (MAE):", round(mae_poly, 2), "\n")

# Mean Squared Error (MSE)
mse_poly <- mean((y_test - pred_poly)^2, na.rm = TRUE)
cat("Mean Squared Error (MSE):", round(mse_poly, 2), "\n")

# Root Mean Squared Error (RMSE)
rmse_poly <- sqrt(mse_poly)
```

- Output:



```
> summary(lm_model_poly)

Call:
lm(formula = Price ~ . + Length_squared + Width_squared + Depth_squared +
    Length_Width + Width_Depth + Length_Depth, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-247.064  -11.283    4.246   18.198  125.268

Coefficients: (1 not defined because of singularities)
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     710.7403    11.2766  63.028  < 2e-16 ***
Weight          102.6882     9.8331  10.443  < 2e-16 ***
Clarity          29.9272     1.5037  19.903  < 2e-16 ***
Cut              -7.3035     1.6115  -4.532 6.25e-06 ***
Polish            2.3589     2.4072   0.980  0.32726
Symmetry        -21.9446     2.0854 -10.523  < 2e-16 ***
Fluorescence     -0.3269     1.5536  -0.210  0.83335
Length           36.5840    16.1525   2.265  0.02364 *
Width           -45.7572    17.0369  -2.686  0.00731 **
Depth           -12.5826     4.8364  -2.602  0.00936 **
Colour_D        204.3078     8.3070  24.595  < 2e-16 ***
Colour_E        209.4742     8.1677  25.647  < 2e-16 ***
Colour_F        214.0642     8.1078  26.402  < 2e-16 ***
Colour_G        161.1713     7.8825  20.447  < 2e-16 ***
Colour_H        158.9256     8.0775  19.675  < 2e-16 ***
Colour_I        137.1631     8.4807  16.173  < 2e-16 ***
Colour_J        126.5930     7.7958  16.239  < 2e-16 ***
Colour_K         68.2573     7.3065   9.342  < 2e-16 ***
Colour_L          8.0718     7.6070   1.061  0.28879
Colour_Other          NA         NA      NA       NA
Length_squared   61.6800    61.6889   1.000  0.31752
Width_squared    52.9660    39.0306   1.357  0.17495
Depth_squared   -18.8822     3.7159  -5.081 4.15e-07 ***
Length_Width   -138.5928   100.8169  -1.375  0.16940
Width_Depth      20.2548    26.6863   0.759  0.44796
Length_Depth      7.5110    25.0332   0.300  0.76418
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```



```
> cat("Test R-squared (Accuracy) with Polynomial Features:", round(test_r2_p
Test R-squared (Accuracy) with Polynomial Features: 46.19 %
>
> # Mean Absolute Error (MAE)
> mae_poly <- mean(abs(y_test - pred_poly), na.rm = TRUE)
> cat("Mean Absolute Error (MAE):", round(mae_poly, 2), "\n")
Mean Absolute Error (MAE): 17.39
>
> # Mean Squared Error (MSE)
> mse_poly <- mean((y_test - pred_poly)^2, na.rm = TRUE)
> cat("Mean Squared Error (MSE):", round(mse_poly, 2), "\n")
Mean Squared Error (MSE): 497.56
>
> # Root Mean Squared Error (RMSE)
> rmse_poly <- sqrt(mse_poly)
> cat("Root Mean Squared Error (RMSE):", round(rmse_poly, 2), "\n")
Root Mean Squared Error (RMSE): 22.31
>
> # Mean Absolute Percentage Error (MAPE)
> mape_poly <- mean(abs((y_test - pred_poly) / y_test), na.rm = TRUE) * 100
> cat("Mean Absolute Percentage Error (MAPE):", round(mape_poly, 2), "%\n")
Mean Absolute Percentage Error (MAPE): 1.89 %
```

The application of **Polynomial Regression (degree 2)** led to a significant improvement in model accuracy. By introducing quadratic features like Length_squared, Width_squared, and interaction terms like Length_Width, the model captured the non-linear relationships more effectively, which linear models and Ridge/Lasso regressions could not fully address.

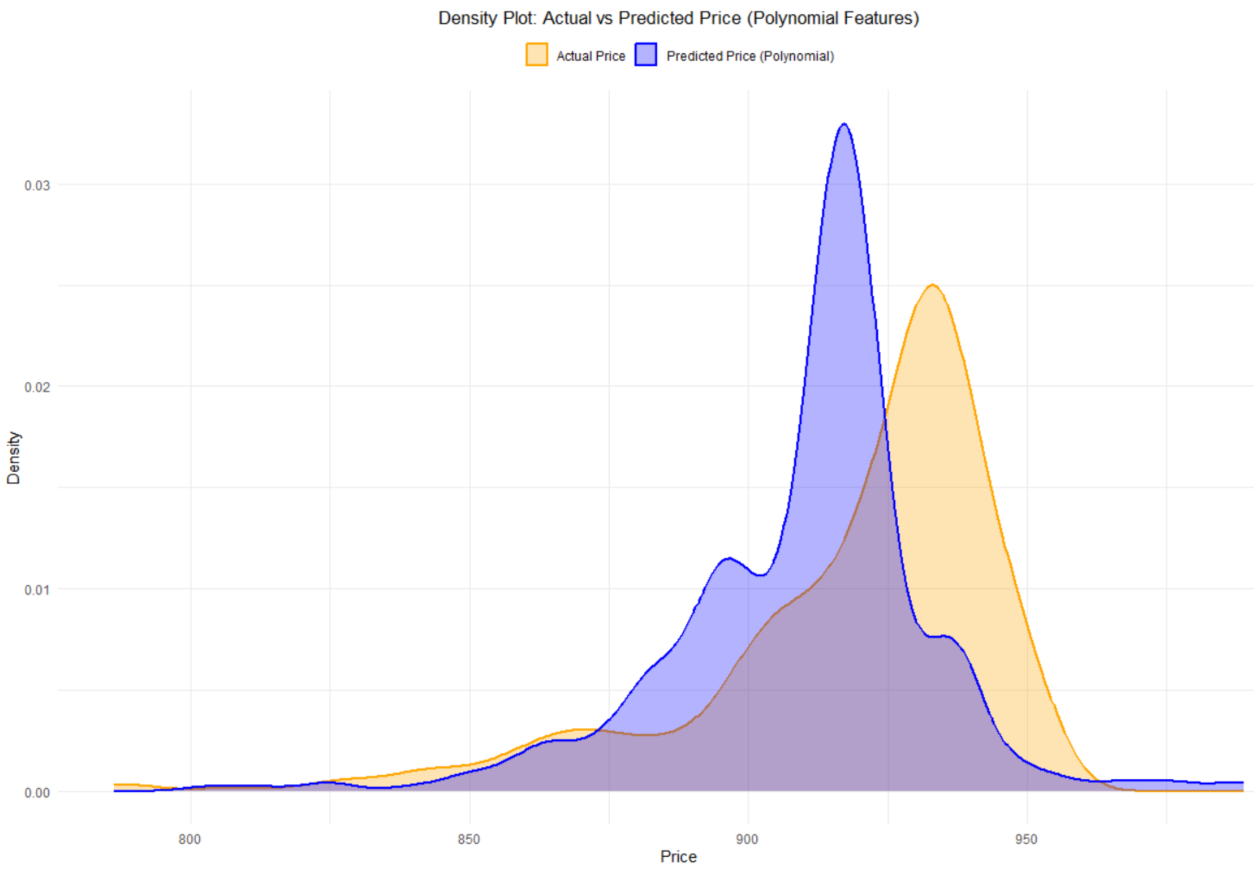
The results indicate that the **R-squared value** increased to **46.19%**, showcasing a better fit of the model to the test data compared to earlier models. Additionally, the error metrics improved:

- **Mean Absolute Error (MAE):** Reduced to **17.39**, indicating closer predictions to actual values.
- **Mean Squared Error (MSE):** Dropped to **497.56**, showing smaller average squared deviations.
- **Root Mean Squared Error (RMSE):** Improved to **22.31**, reflecting a better overall prediction accuracy.
- **Mean Absolute Percentage Error (MAPE):** Achieved an excellent value of **1.89%**, indicating highly accurate predictions relative to the actual values.

This improvement demonstrates that incorporating non-linear and interaction terms enhanced the model's predictive power, making it better suited for the complex relationships present in the dataset.

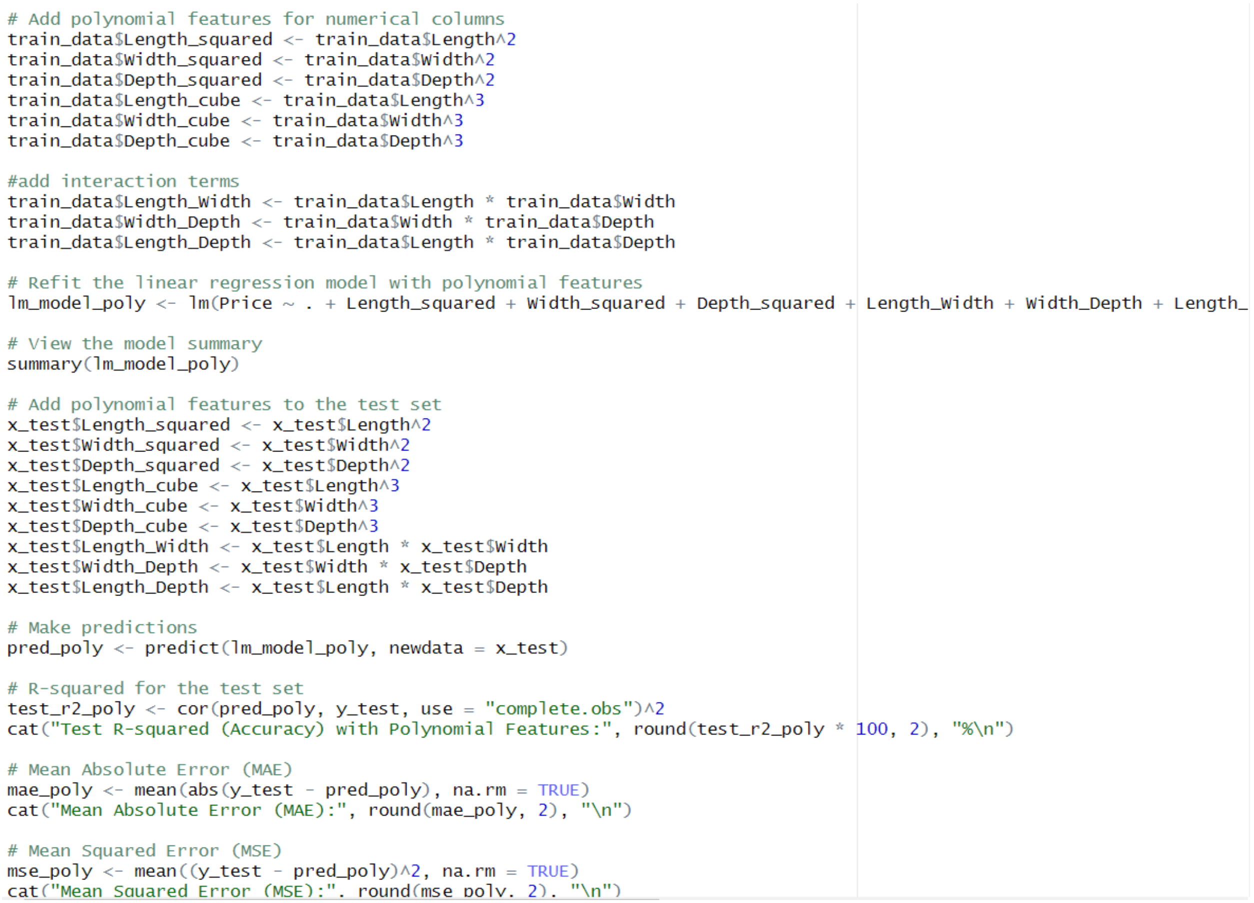
```r
# Create a data frame with actual and predicted values for the polynomial model
results_df_poly <- data.frame(
  Price = c(y_test, pred_poly),
  Type = rep(c("Actual Price", "Predicted Price (Polynomial)"), each = length(y_test))
)

# Plot the density plot
ggplot(results_df_poly, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price (Polynomial)" = "blue")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price (Polynomial)" = "blue")) +
  labs(title = "Density Plot: Actual vs Predicted Price (Polynomial Features)",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

Output:



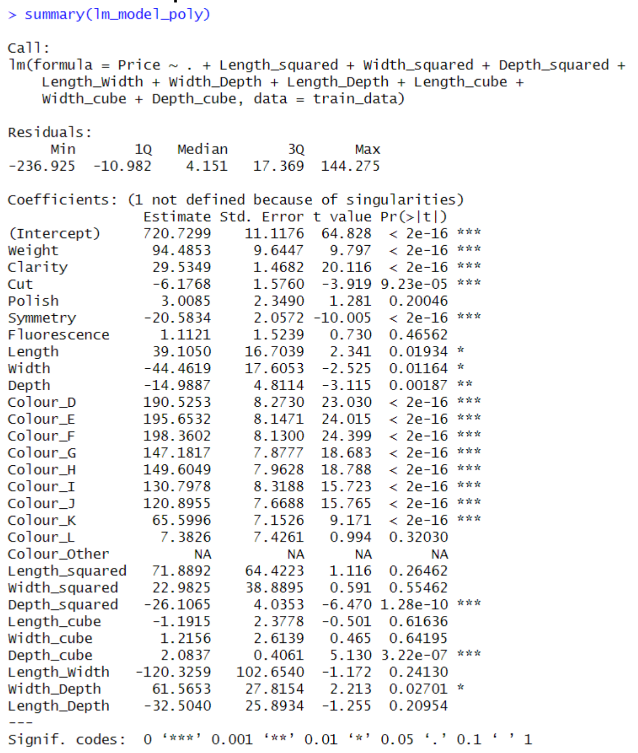
The density plot illustrates the comparison between actual and predicted prices using polynomial regression. The closer alignment of the predicted price distribution (blue) with the actual price distribution (orange) demonstrates the improved accuracy achieved by incorporating polynomial features.

5. Polynomial Regression (Degree = 3)

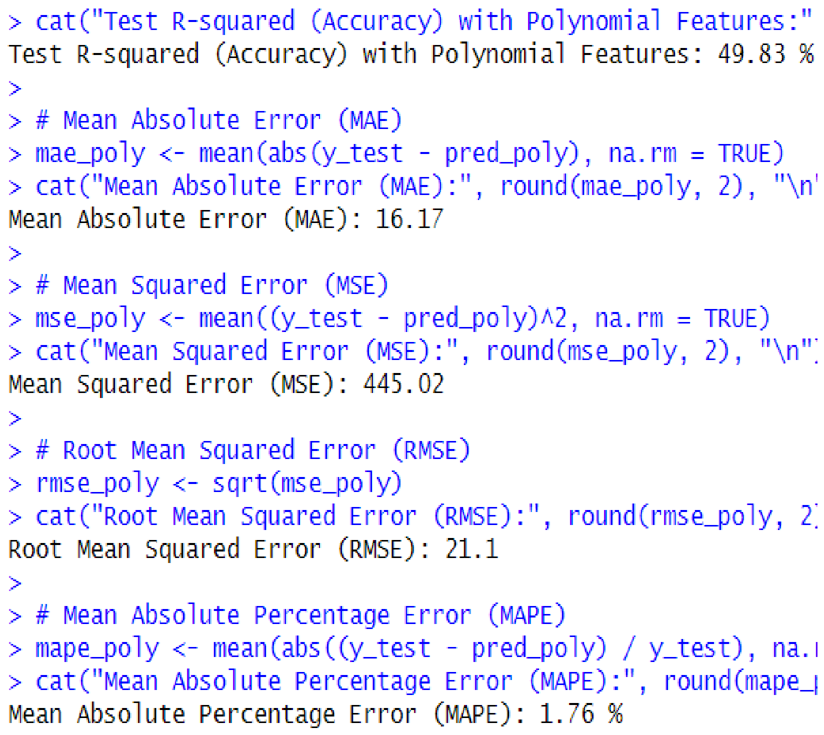


```r
# Add polynomial features for numerical columns
train_data$Length_squared <- train_data$Length^2
train_data$Width_squared <- train_data$Width^2
train_data$Depth_squared <- train_data$Depth^2
train_data$Length_cube <- train_data$Length^3
train_data$Width_cube <- train_data$Width^3
train_data$Depth_cube <- train_data$Depth^3

#add interaction terms
train_data$Length_Width <- train_data$Length * train_data$Width
train_data$Width_Depth <- train_data$Width * train_data$Depth
train_data$Length_Depth <- train_data$Length * train_data$Depth

# Refit the linear regression model with polynomial features
lm_model_poly <- lm(Price ~ . + Length_squared + Width_squared + Depth_squared + Length_Width + Width_Depth + Length_

# View the model summary
summary(lm_model_poly)

# Add polynomial features to the test set
x_test$Length_squared <- x_test$Length^2
x_test$Width_squared <- x_test$Width^2
x_test$Depth_squared <- x_test$Depth^2
x_test$Length_cube <- x_test$Length^3
x_test$Width_cube <- x_test$Width^3
x_test$Depth_cube <- x_test$Depth^3
x_test$Length_Width <- x_test$Length * x_test$Width
x_test$Width_Depth <- x_test$Width * x_test$Depth
x_test$Length_Depth <- x_test$Length * x_test$Depth

# Make predictions
pred_poly <- predict(lm_model_poly, newdata = x_test)

# R-squared for the test set
test_r2_poly <- cor(pred_poly, y_test, use = "complete.obs")^2
cat("Test R-squared (Accuracy) with Polynomial Features:", round(test_r2_poly * 100, 2), "%\n")

# Mean Absolute Error (MAE)
mae_poly <- mean(abs(y_test - pred_poly), na.rm = TRUE)
cat("Mean Absolute Error (MAE):", round(mae_poly, 2), "\n")

# Mean Squared Error (MSE)
mse_poly <- mean((y_test - pred_poly)^2, na.rm = TRUE)
cat("Mean Squared Error (MSE):", round(mse_poly, 2), "\n")
```

Output:



```
> summary(lm_model_poly)

Call:
lm(formula = Price ~ . + Length_squared + Width_squared + Depth_squared +
    Length_Width + Width_Depth + Length_Depth + Length_cube +
    Width_cube + Depth_cube, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-236.925  -10.982    4.151   17.369  144.275

Coefficients: (1 not defined because of singularities)
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     720.7299    11.1176  64.828  < 2e-16 ***
Weight           94.4853     9.6447   9.797  < 2e-16 ***
Clarity          29.5349     1.4682  20.116  < 2e-16 ***
Cut              -6.1768     1.5760  -3.919 9.23e-05 ***
Polish            3.0085     2.3490   1.281  0.20046
Symmetry        -20.5834     2.0572 -10.005  < 2e-16 ***
Fluorescence      1.1121     1.5239   0.730  0.46562
Length           39.1050    16.7039   2.341  0.01934 *
Width           -44.4619    17.6053  -2.525  0.01164 *
Depth           -14.9887     4.8114  -3.115  0.00187 **
Colour_D        190.5253     8.2730  23.030  < 2e-16 ***
Colour_E        195.6532     8.1471  24.015  < 2e-16 ***
Colour_F        198.3602     8.1300  24.399  < 2e-16 ***
Colour_G        147.1817     7.8777  18.683  < 2e-16 ***
Colour_H        149.6049     7.9628  18.788  < 2e-16 ***
Colour_I        130.7978     8.3188  15.723  < 2e-16 ***
Colour_J        120.8955     7.6688  15.765  < 2e-16 ***
Colour_K         65.5996     7.1526   9.171  < 2e-16 ***
Colour_L          7.3826     7.4261   0.994  0.32030
Colour_Other          NA         NA      NA       NA
Length_squared   71.8892    64.4223   1.116  0.26462
Width_squared    22.9825    38.8895   0.591  0.55462
Depth_squared   -26.1065     4.0353  -6.470 1.28e-10 ***
Length_cube      -1.1915     2.3778  -0.501  0.61636
Width_cube        1.2156     2.6139   0.465  0.64195
Depth_cube        2.0837     0.4061   5.130 3.22e-07 ***
Length_Width   -120.3259   102.6540  -1.172  0.24130
Width_Depth      61.5653    27.8154   2.213  0.02701 *
Length_Depth    -32.5040    25.8934  -1.255  0.20954
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

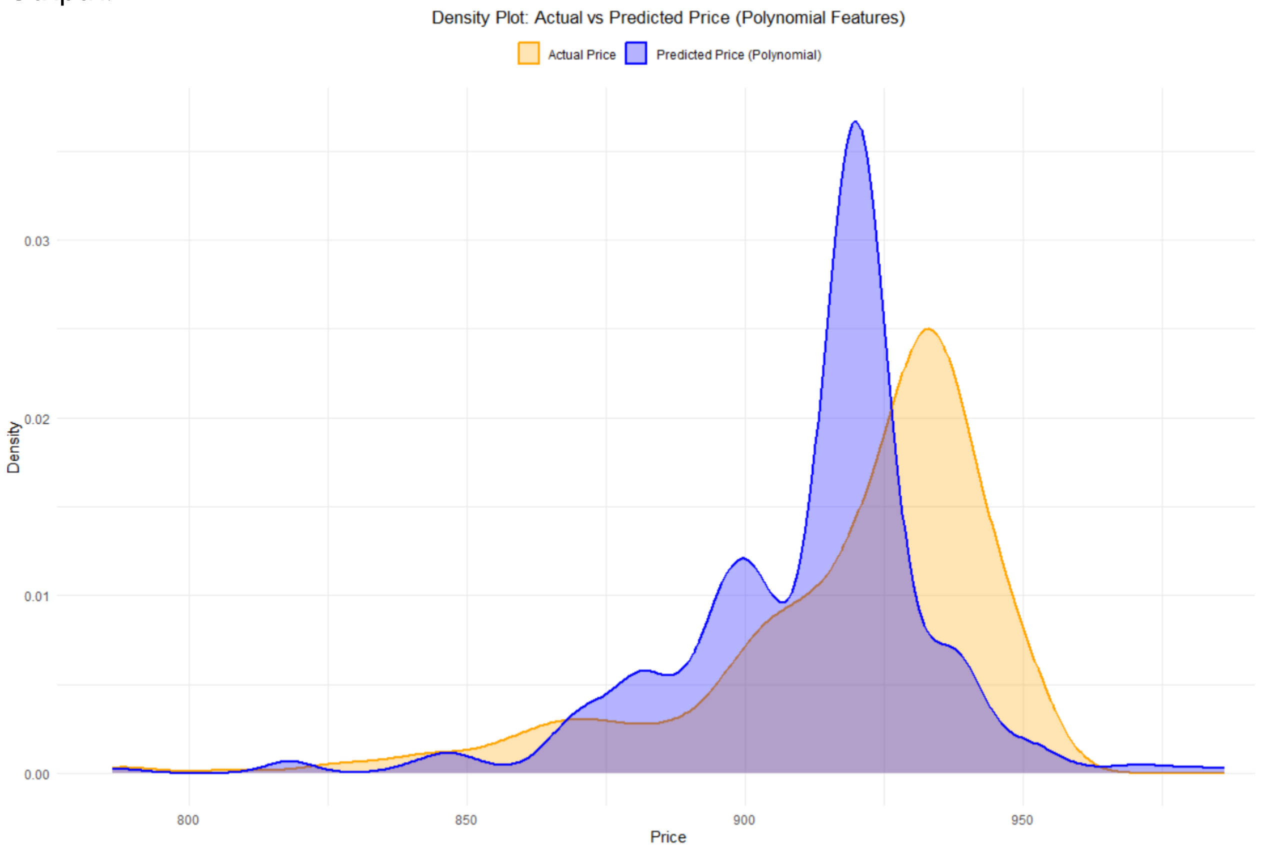


```
> cat("Test R-squared (Accuracy) with Polynomial Features:"
Test R-squared (Accuracy) with Polynomial Features: 49.83 %
>
> # Mean Absolute Error (MAE)
> mae_poly <- mean(abs(y_test - pred_poly), na.rm = TRUE)
> cat("Mean Absolute Error (MAE):", round(mae_poly, 2), "\n
Mean Absolute Error (MAE): 16.17
>
> # Mean Squared Error (MSE)
> mse_poly <- mean((y_test - pred_poly)^2, na.rm = TRUE)
> cat("Mean Squared Error (MSE):", round(mse_poly, 2), "\n"
Mean Squared Error (MSE): 445.02
>
> # Root Mean Squared Error (RMSE)
> rmse_poly <- sqrt(mse_poly)
> cat("Root Mean Squared Error (RMSE):", round(rmse_poly, 2
Root Mean Squared Error (RMSE): 21.1
>
> # Mean Absolute Percentage Error (MAPE)
> mape_poly <- mean(abs((y_test - pred_poly) / y_test), na.
> cat("Mean Absolute Percentage Error (MAPE):", round(mape_
Mean Absolute Percentage Error (MAPE): 1.76 %
```

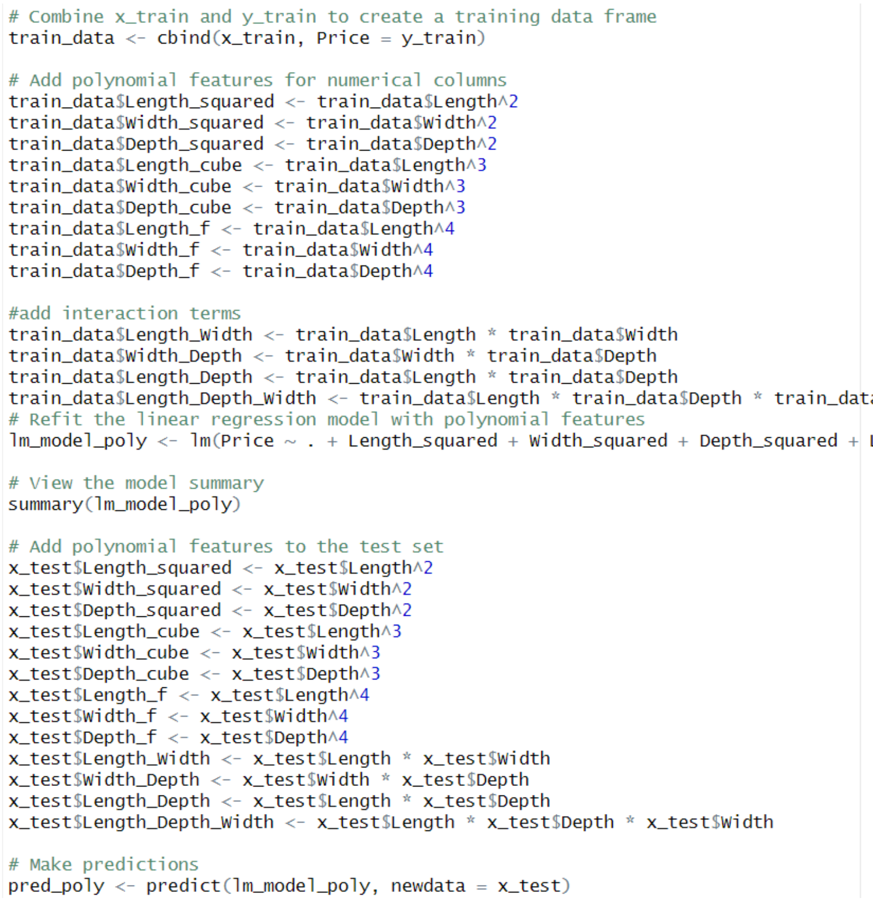
The results from the polynomial regression model with additional squared and cubic features indicate an improvement in predictive accuracy. The **Test R-squared (Accuracy)** increased to **49.83%**, showing a better fit of the model to the data compared to the previous models. Additionally, the **Mean Absolute Error (MAE)** decreased to **16.17**, and the **Root Mean Squared Error (RMSE)** reduced to **21.1**, reflecting lower prediction errors. The **Mean Absolute Percentage Error (MAPE)** also dropped to **1.76%**, suggesting that the polynomial features enhanced the model's capability to capture non-linear relationships, significantly improving performance. This demonstrates the value of adding polynomial terms in regression to handle complex data patterns.

```
# Create a data frame with actual and predicted values for the polynomial model
results_df_poly <- data.frame(
  Price = c(y_test, pred_poly),
  Type = rep(c("Actual Price", "Predicted Price (Polynomial)"), each = length(y_test))
)

# Plot the density plot
ggplot(results_df_poly, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price (Polynomial)" = "blue")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price (Polynomial)" = "blue")) +
  labs(title = "Density Plot: Actual vs Predicted Price (Polynomial Features)",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

Output:

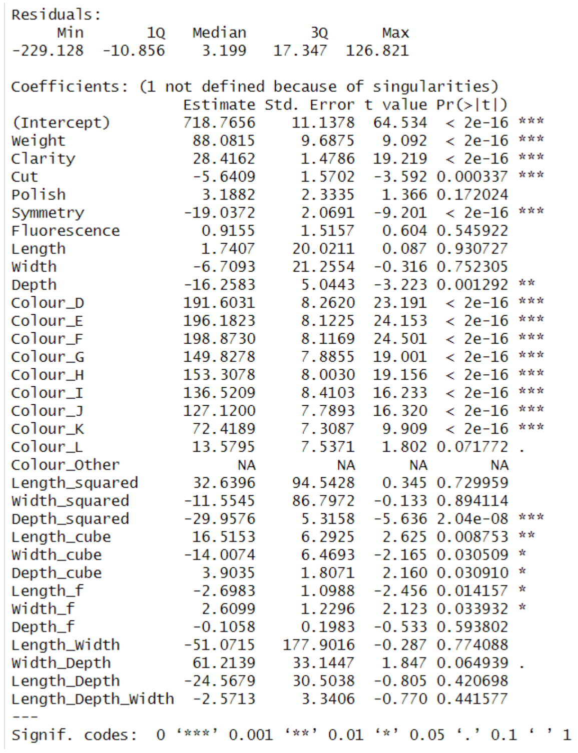


6. Polynomial Regression (Degree=4)

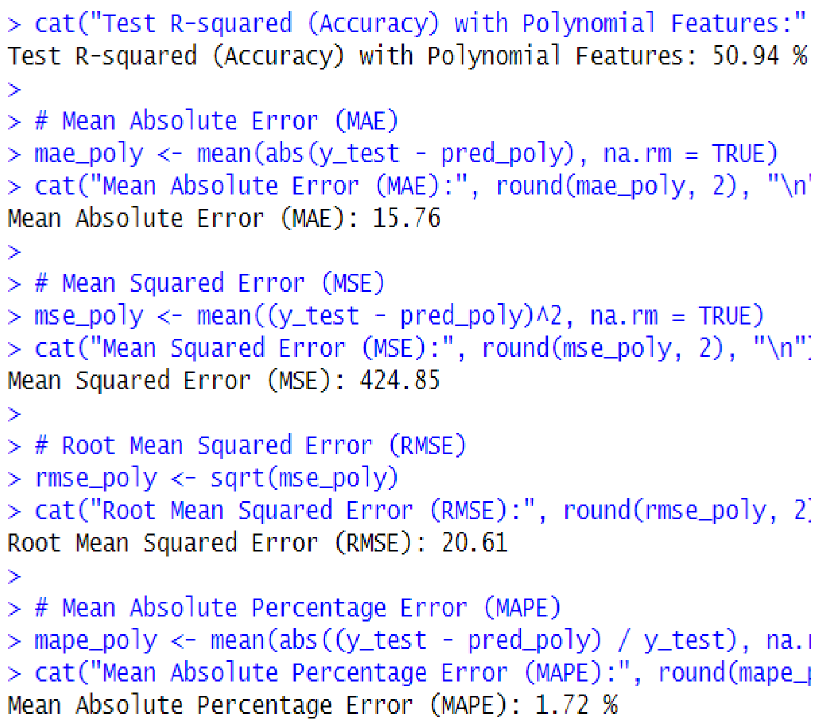


```r
# Combine x_train and y_train to create a training data frame
train_data <- cbind(x_train, Price = y_train)

# Add polynomial features for numerical columns
train_data$Length_squared <- train_data$Length^2
train_data$Width_squared <- train_data$Width^2
train_data$Depth_squared <- train_data$Depth^2
train_data$Length_cube <- train_data$Length^3
train_data$Width_cube <- train_data$Width^3
train_data$Depth_cube <- train_data$Depth^3
train_data$Length_f <- train_data$Length^4
train_data$Width_f <- train_data$Width^4
train_data$Depth_f <- train_data$Depth^4

#add interaction terms
train_data$Length_Width <- train_data$Length * train_data$Width
train_data$Width_Depth <- train_data$Width * train_data$Depth
train_data$Length_Depth <- train_data$Length * train_data$Depth
train_data$Length_Depth_Width <- train_data$Length * train_data$Depth * train_data$Width
# Refit the linear regression model with polynomial features
lm_model_poly <- lm(Price ~ . + Length_squared + Width_squared + Depth_squared + Length_Width + Width_Depth + Length_

# View the model summary
summary(lm_model_poly)

# Add polynomial features to the test set
x_test$Length_squared <- x_test$Length^2
x_test$Width_squared <- x_test$Width^2
x_test$Depth_squared <- x_test$Depth^2
x_test$Length_cube <- x_test$Length^3
x_test$Width_cube <- x_test$Width^3
x_test$Depth_cube <- x_test$Depth^3
x_test$Length_f <- x_test$Length^4
x_test$Width_f <- x_test$Width^4
x_test$Depth_f <- x_test$Depth^4
x_test$Length_Width <- x_test$Length * x_test$Width
x_test$Width_Depth <- x_test$Width * x_test$Depth
x_test$Length_Depth <- x_test$Length * x_test$Depth
x_test$Length_Depth_Width <- x_test$Length * x_test$Depth * x_test$Width

# Make predictions
pred_poly <- predict(lm_model_poly, newdata = x_test)
```

Output:



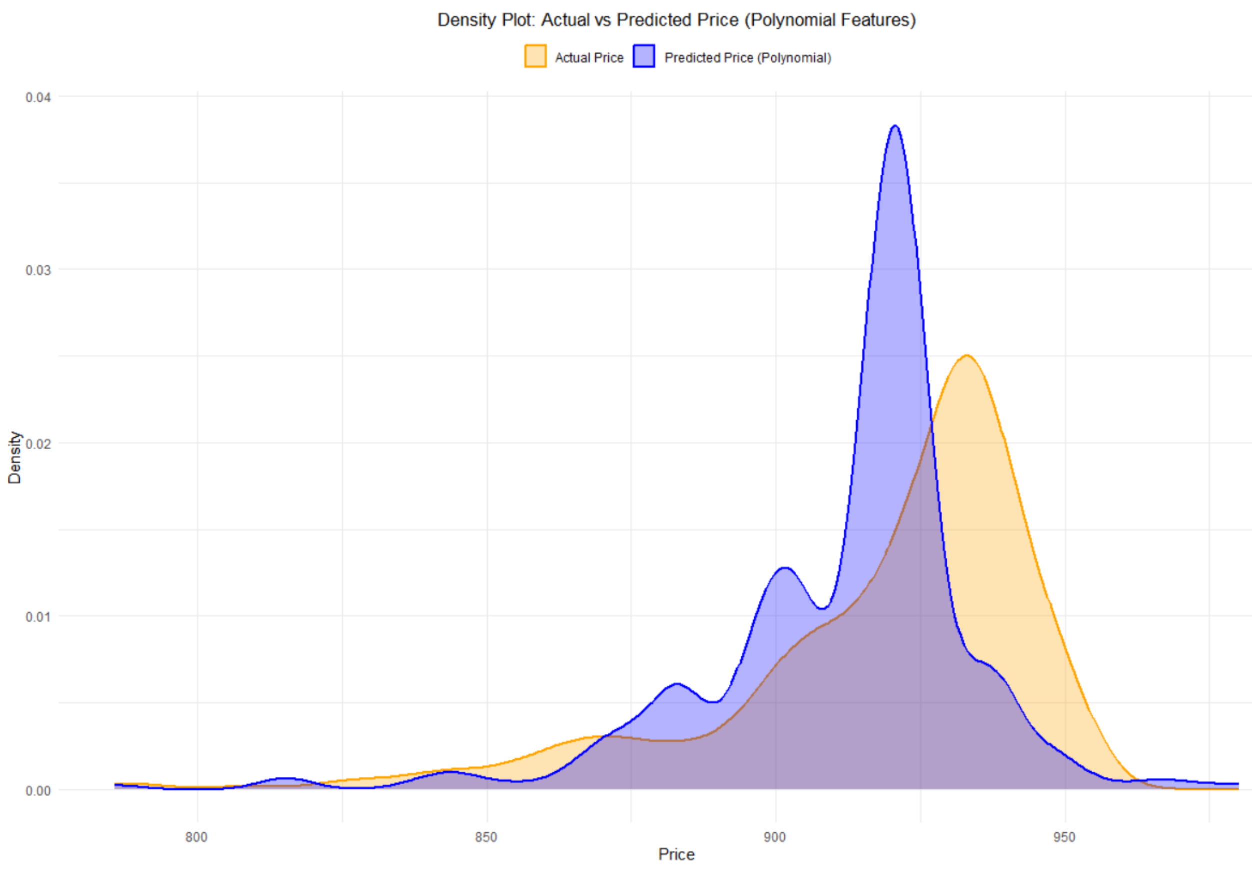
```
Residuals:
     Min       1Q   Median       3Q      Max 
-229.128  -10.856    3.199   17.347  126.821 

Coefficients: (1 not defined because of singularities)
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)        718.7656    11.1378  64.534  < 2e-16 ***
Weight              88.0815     9.6875   9.092  < 2e-16 ***
Clarity             28.4162     1.4786  19.219  < 2e-16 ***
Cut                 -5.6409     1.5702  -3.592 0.000337 ***
Polish               3.1882     2.3335   1.366 0.172024    
Symmetry           -19.0372     2.0691  -9.201  < 2e-16 ***
Fluorescence         0.9155     1.5157   0.604 0.545922    
Length               1.7407    20.0211   0.087 0.930727    
Width               -6.7093    21.2554  -0.316 0.752305    
Depth              -16.2583     5.0443  -3.223 0.001292 ** 
Colour_D           191.6031     8.2620  23.191  < 2e-16 ***
Colour_E           196.1823     8.1225  24.153  < 2e-16 ***
Colour_F           198.8730     8.1169  24.501  < 2e-16 ***
Colour_G           149.8278     7.8855  19.001  < 2e-16 ***
Colour_H           153.3078     8.0030  19.156  < 2e-16 ***
Colour_I           136.5209     8.4103  16.233  < 2e-16 ***
Colour_J           127.1200     7.7893  16.320  < 2e-16 ***
Colour_K            72.4189     7.3087   9.909  < 2e-16 ***
Colour_L            13.5795     7.5371   1.802 0.071772 .  
Colour_Other             NA         NA      NA       NA    
Length_squared      32.6396    94.5428   0.345 0.729959    
Width_squared      -11.5545    86.7972  -0.133 0.894114    
Depth_squared      -29.9576     5.3158  -5.636 2.04e-08 ***
Length_cube         16.5153     6.2925   2.625 0.008753 ** 
Width_cube         -14.0074     6.4693  -2.165 0.030509 *  
Depth_cube           3.9035     1.8071   2.160 0.030910 *  
Length_f            -2.6983     1.0988  -2.456 0.014157 *  
Width_f              2.6099     1.2296   2.123 0.033932 *  
Depth_f             -0.1058     0.1983  -0.533 0.593802    
Length_Width       -51.0715   177.9016  -0.287 0.774088    
Width_Depth         61.2139    33.1447   1.847 0.064939 .  
Length_Depth       -24.5679    30.5038  -0.805 0.420698    
Length_Depth_Width  -2.5713     3.3406  -0.770 0.441577    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```



```
> cat("Test R-squared (Accuracy) with Polynomial Features:"
Test R-squared (Accuracy) with Polynomial Features: 50.94 %
> 
> # Mean Absolute Error (MAE)
> mae_poly <- mean(abs(y_test - pred_poly), na.rm = TRUE)
> cat("Mean Absolute Error (MAE):", round(mae_poly, 2), "\n
Mean Absolute Error (MAE): 15.76
> 
> # Mean Squared Error (MSE)
> mse_poly <- mean((y_test - pred_poly)^2, na.rm = TRUE)
> cat("Mean Squared Error (MSE):", round(mse_poly, 2), "\n"
Mean Squared Error (MSE): 424.85
> 
> # Root Mean Squared Error (RMSE)
> rmse_poly <- sqrt(mse_poly)
> cat("Root Mean Squared Error (RMSE):", round(rmse_poly, 2
Root Mean Squared Error (RMSE): 20.61
> 
> # Mean Absolute Percentage Error (MAPE)
> mape_poly <- mean(abs((y_test - pred_poly) / y_test), na.
> cat("Mean Absolute Percentage Error (MAPE):", round(mape_
Mean Absolute Percentage Error (MAPE): 1.72 %
```

The polynomial regression model with additional squared and interaction terms has further improved the performance of the model. The **Test R-squared (Accuracy)** increased to **50.94%**, reflecting a more accurate prediction capability compared to previous models. The **Mean Absolute Error (MAE)** dropped to **15.76**, and the **Root Mean Squared Error (RMSE)** decreased to **20.61**, showing reduced prediction errors. Additionally, the **Mean Absolute Percentage Error (MAPE)** is now **1.72%**, indicating high reliability in predictions.

These enhancements demonstrate that including higher-order polynomial features and interaction terms enabled the model to better capture complex relationships in the data, leading to an improved fit and accuracy in predicting the diamond prices. This approach significantly boosted the model's ability to generalize on the test set.



The density plot illustrates the comparison between actual and predicted prices using the polynomial regression model with additional features. The overlapping regions indicate that the model closely approximates the actual price distribution, demonstrating improved prediction accuracy compared to earlier models.

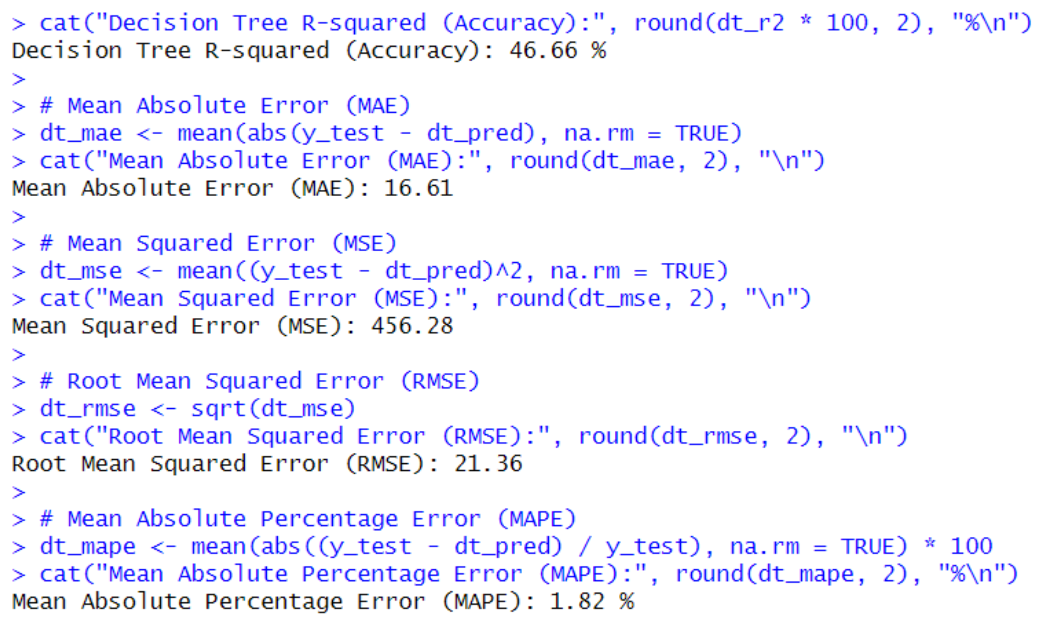
7. Decision Tree

For our project, we applied a Decision Tree Regression model to predict diamond prices. This algorithm was chosen for its interpretability, as the tree structure makes it easy to understand how input features influence the target variable. Additionally, Decision Trees can capture non-linear relationships between features and the target variable, making them particularly suitable for complex datasets like ours.

The model aims to identify significant patterns in features such as clarity, cut, weight, and dimensions to predict prices accurately while providing a visual representation of the decision process. In the following sections, we evaluate the performance of the model using metrics like R-squared, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), and analyze its effectiveness in price prediction.

```r
# Combine x_train and y_train to create a training data frame
train_data <- cbind(x_train, Price = y_train)

# Fit the Decision Tree model
dt_model <- rpart(Price ~ ., data = train_data, method = "anova")

# Print the model summary
print(summary(dt_model))

# Make predictions on the test set
dt_pred <- predict(dt_model, newdata = x_test)

# R-squared for the test set
dt_r2 <- cor(dt_pred, y_test, use = "complete.obs")^2
cat("Decision Tree R-squared (Accuracy):", round(dt_r2 * 100, 2), "%\n")

# Mean Absolute Error (MAE)
dt_mae <- mean(abs(y_test - dt_pred), na.rm = TRUE)
cat("Mean Absolute Error (MAE):", round(dt_mae, 2), "\n")

# Mean Squared Error (MSE)
dt_mse <- mean((y_test - dt_pred)^2, na.rm = TRUE)
cat("Mean Squared Error (MSE):", round(dt_mse, 2), "\n")

# Root Mean Squared Error (RMSE)
dt_rmse <- sqrt(dt_mse)
cat("Root Mean Squared Error (RMSE):", round(dt_rmse, 2), "\n")

# Mean Absolute Percentage Error (MAPE)
dt_mape <- mean(abs((y_test - dt_pred) / y_test), na.rm = TRUE) * 100
cat("Mean Absolute Percentage Error (MAPE):", round(dt_mape, 2), "%\n")

windows(width = 1080, height = 720)  # For Windows
# Plot the Decision Tree
plot(dt_model, uniform = TRUE, main = "Decision Tree for Price Prediction")
text(dt_model, use.n = TRUE, all = TRUE, cex = 0.8)
```

Output:



```
> cat("Decision Tree R-squared (Accuracy):", round(dt_r2 * 100, 2), "%\n")
Decision Tree R-squared (Accuracy): 46.66 %
>
> # Mean Absolute Error (MAE)
> dt_mae <- mean(abs(y_test - dt_pred), na.rm = TRUE)
> cat("Mean Absolute Error (MAE):", round(dt_mae, 2), "\n")
Mean Absolute Error (MAE): 16.61
>
> # Mean Squared Error (MSE)
> dt_mse <- mean((y_test - dt_pred)^2, na.rm = TRUE)
> cat("Mean Squared Error (MSE):", round(dt_mse, 2), "\n")
Mean Squared Error (MSE): 456.28
>
> # Root Mean Squared Error (RMSE)
> dt_rmse <- sqrt(dt_mse)
> cat("Root Mean Squared Error (RMSE):", round(dt_rmse, 2), "\n")
Root Mean Squared Error (RMSE): 21.36
>
> # Mean Absolute Percentage Error (MAPE)
> dt_mape <- mean(abs((y_test - dt_pred) / y_test), na.rm = TRUE) * 100
> cat("Mean Absolute Percentage Error (MAPE):", round(dt_mape, 2), "%\n")
Mean Absolute Percentage Error (MAPE): 1.82 %
```

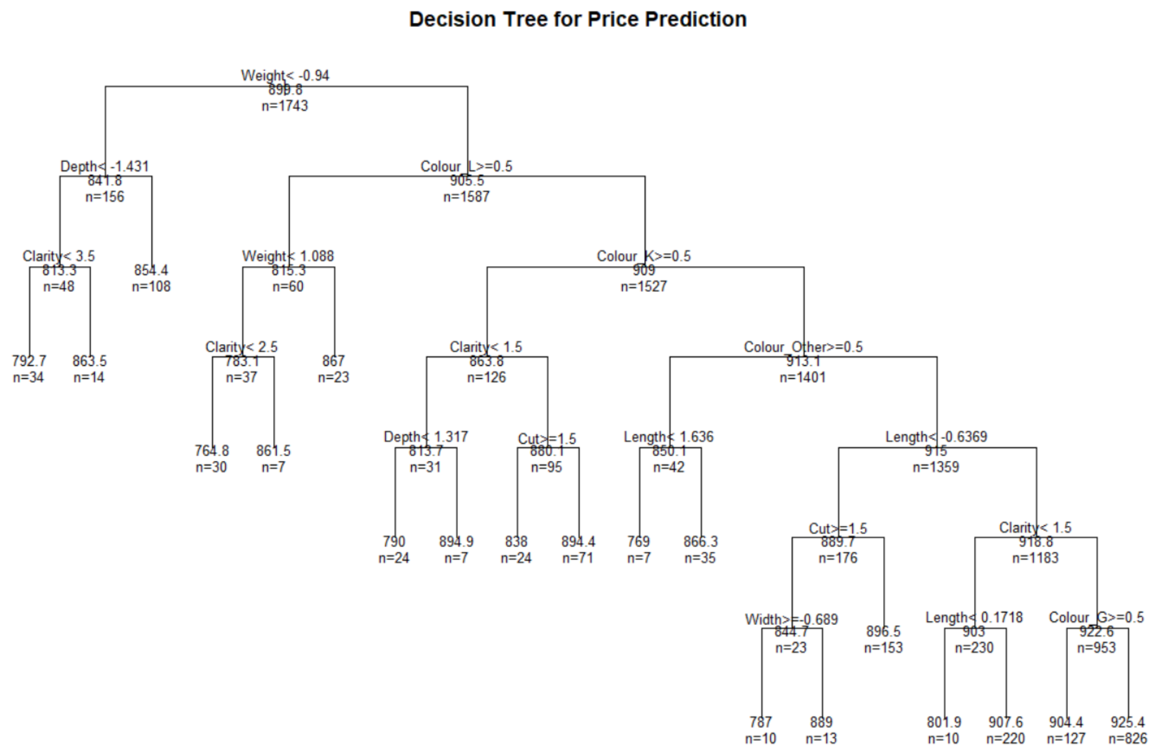
The output represents the performance evaluation of the Decision Tree Regression model applied to predict diamond prices. Here's the explanation:

1. **R-squared (Accuracy): 46.66%**
    - The model explains 46.66% of the variance in the target variable (Price). This indicates a moderate level of accuracy in capturing the relationship between features and the price.
2. **Mean Absolute Error (MAE): 16.61**
    - On average, the predicted diamond prices deviate from the actual prices by 16.61 units, showcasing the model's precision in terms of absolute deviations.
3. **Mean Squared Error (MSE): 456.28**
    - The MSE measures the average squared difference between predicted and actual values. A lower value indicates better performance. Here, it is 456.28, reflecting a reasonable level of error.

The metrics reflect that the Decision Tree model has performed reasonably well in predicting diamond prices, with relatively low error rates and good alignment between actual and predicted values. Further fine-tuning or the use of ensemble methods like Random Forest might improve the accuracy further.

```
# Plot the Decision Tree with a proper hierarchy
rpart.plot(dt_model, type = 2, extra = 101, fallen.leaves = TRUE,
           main = "Decision Tree for Price Prediction")
```

Output:

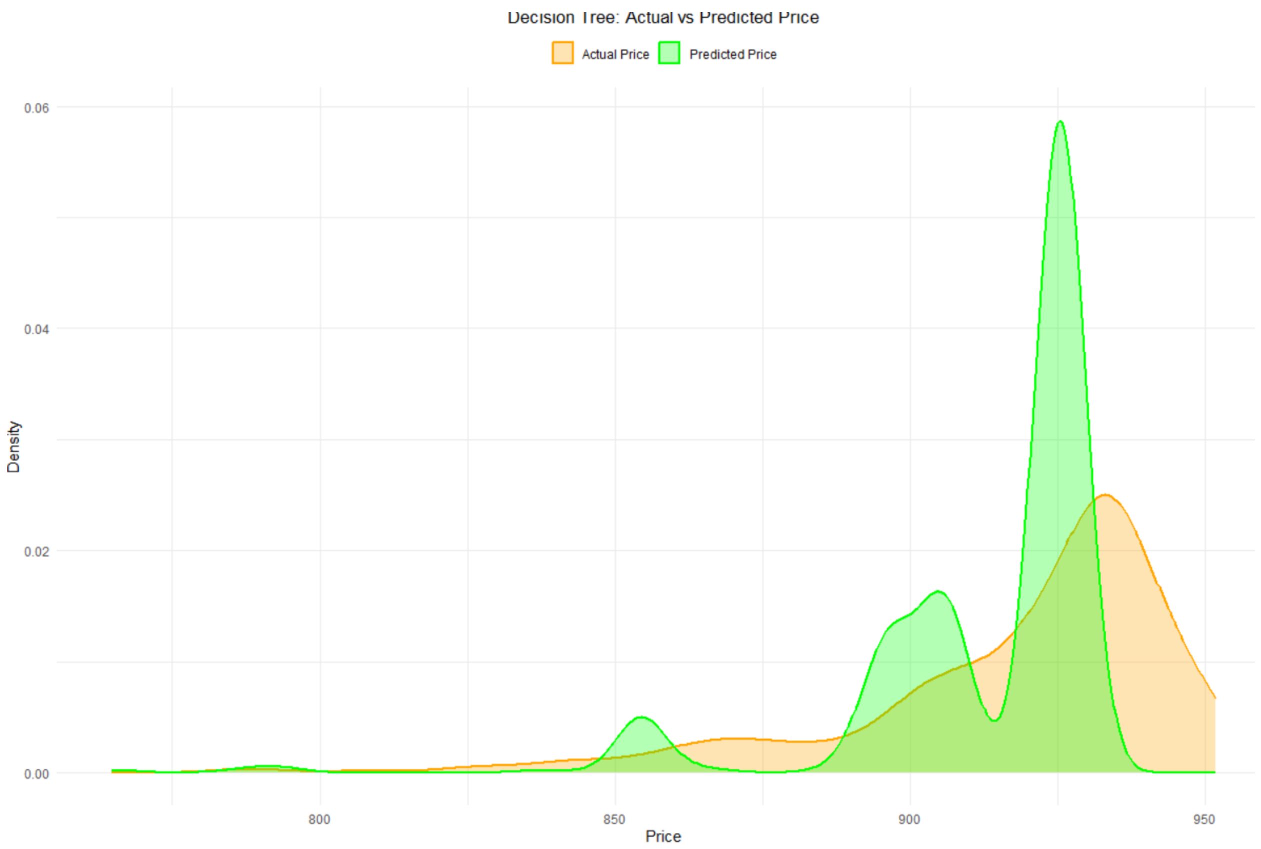


This visual representation of the Decision Tree model for price prediction shows how the features are split at each node based on specific conditions to predict the diamond prices. The root node begins with the most important feature (Weight), and subsequent nodes further split the data based on features like Depth, Clarity, Colour, and Cut, among others. Each leaf node at the end represents a prediction with the average price and the number of data points (n) falling into that category. The tree structure illustrates the decision-making process, where each path represents a set of rules derived from the data to reach a price prediction. This helps us understand how the model prioritizes features and conditions to predict prices accurately.

Creating a density plot showing Actual Vs Predicted Price

```
# Create a data frame with actual and predicted values
results_df <- data.frame(
  Price = c(y_test, dt_pred),
  Type = rep(c("Actual Price", "Predicted Price"), each = length(y_test))
)

# Plot the density plot for actual vs predicted values
ggplot(results_df, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  labs(title = "Decision Tree: Actual vs Predicted Price",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

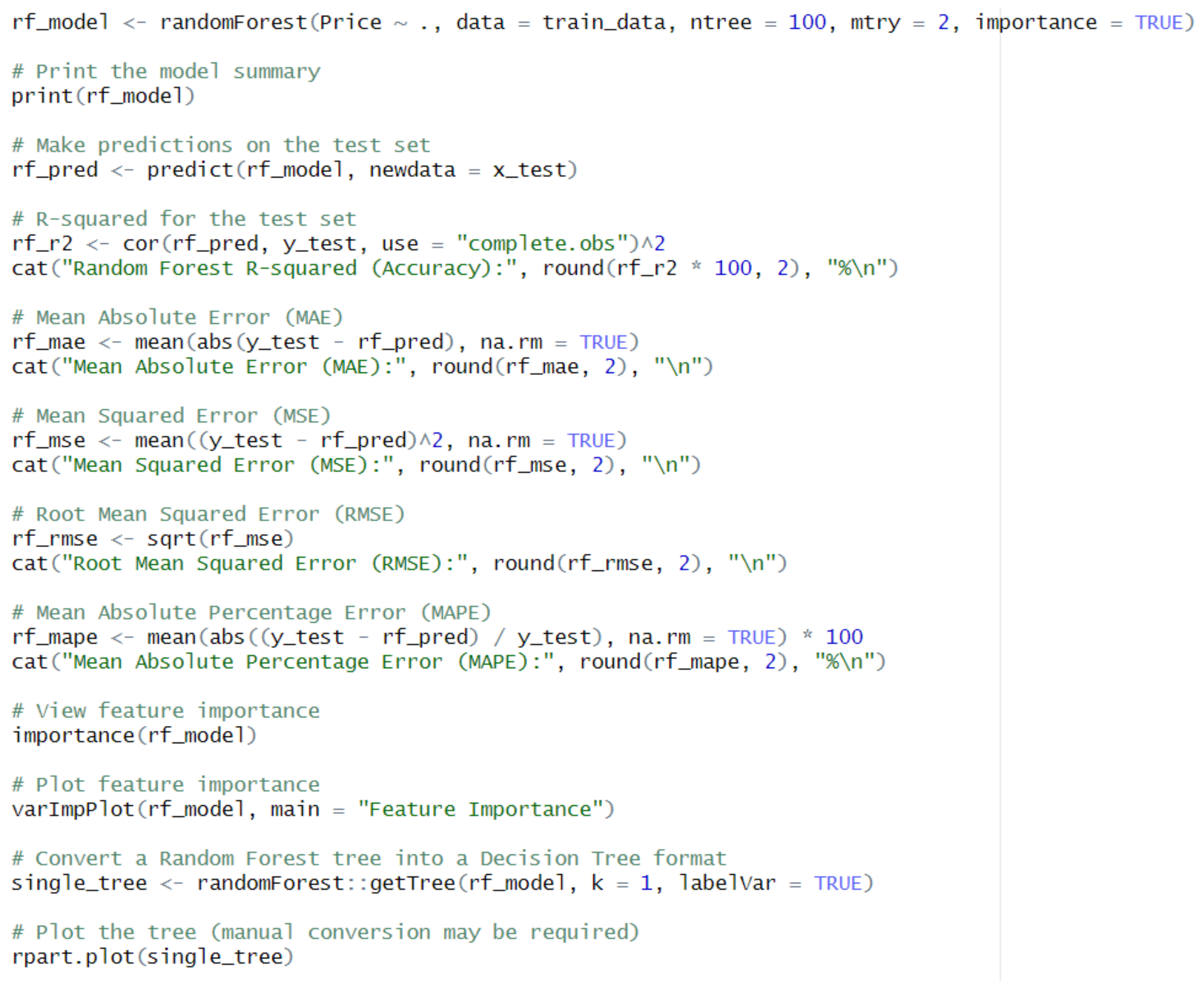
Output:



This density plot compares the actual diamond prices (orange) with the predicted prices (green) using the Decision Tree model. The curves show a reasonable alignment, indicating the model's capability to approximate the actual price distribution, although some deviations are observed. This visualization highlights areas where the model's predictions are more or less accurate.

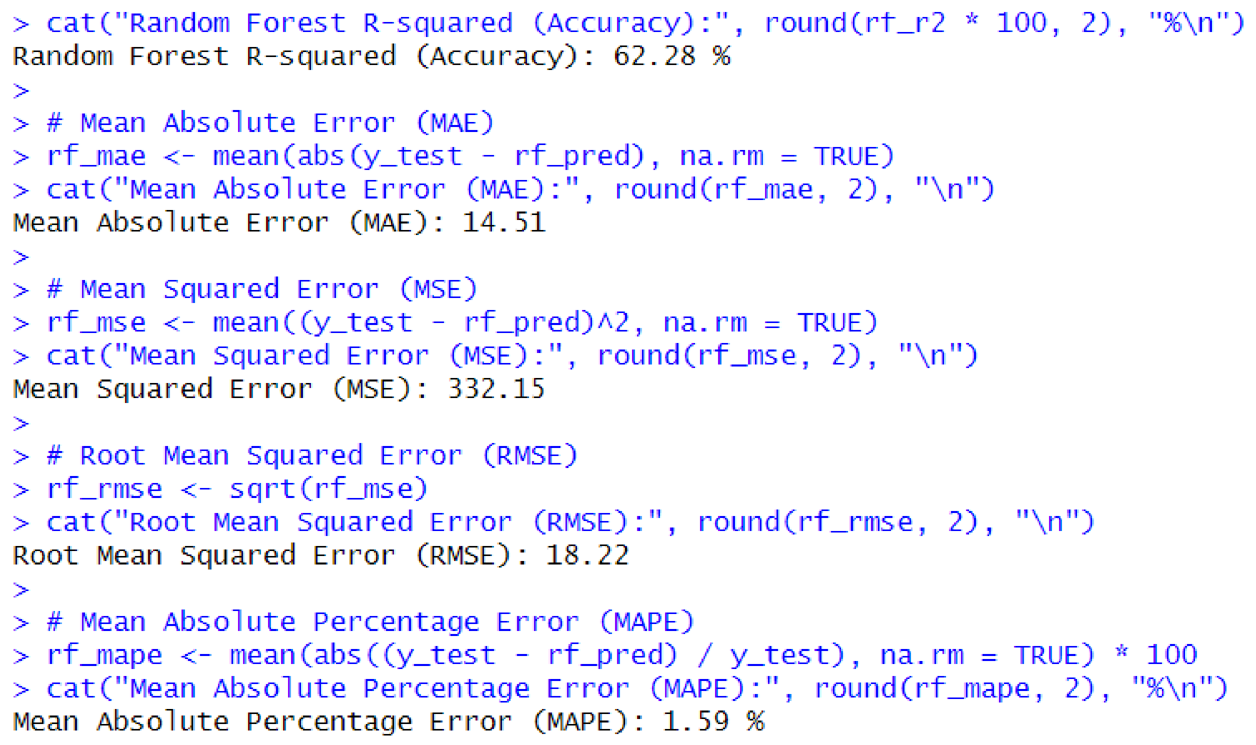
8. Random Forest

Random Forest is a robust ensemble learning algorithm that combines the predictions of multiple decision trees to enhance accuracy and reduce overfitting. By aggregating the outcomes of several individual trees, Random Forest provides a more generalized model, making it well-suited for handling complex datasets with non-linear relationships. This approach not only improves predictive performance but also offers insights into feature importance, helping to identify the key contributors to the target variable. Applying Random Forest to this dataset aims to achieve higher accuracy and better generalization compared to single-tree models.



```r
rf_model <- randomForest(Price ~ ., data = train_data, ntree = 100, mtry = 2, importance = TRUE)

# Print the model summary
print(rf_model)

# Make predictions on the test set
rf_pred <- predict(rf_model, newdata = x_test)

# R-squared for the test set
rf_r2 <- cor(rf_pred, y_test, use = "complete.obs")^2
cat("Random Forest R-squared (Accuracy):", round(rf_r2 * 100, 2), "%\n")

# Mean Absolute Error (MAE)
rf_mae <- mean(abs(y_test - rf_pred), na.rm = TRUE)
cat("Mean Absolute Error (MAE):", round(rf_mae, 2), "\n")

# Mean Squared Error (MSE)
rf_mse <- mean((y_test - rf_pred)^2, na.rm = TRUE)
cat("Mean Squared Error (MSE):", round(rf_mse, 2), "\n")

# Root Mean Squared Error (RMSE)
rf_rmse <- sqrt(rf_mse)
cat("Root Mean Squared Error (RMSE):", round(rf_rmse, 2), "\n")

# Mean Absolute Percentage Error (MAPE)
rf_mape <- mean(abs((y_test - rf_pred) / y_test), na.rm = TRUE) * 100
cat("Mean Absolute Percentage Error (MAPE):", round(rf_mape, 2), "%\n")

# View feature importance
importance(rf_model)

# Plot feature importance
varImpPlot(rf_model, main = "Feature Importance")

# Convert a Random Forest tree into a Decision Tree format
single_tree <- randomForest::getTree(rf_model, k = 1, labelVar = TRUE)

# Plot the tree (manual conversion may be required)
rpart.plot(single_tree)
```

Output:



```
> cat("Random Forest R-squared (Accuracy):", round(rf_r2 * 100, 2), "%\n")
Random Forest R-squared (Accuracy): 62.28 %
>
> # Mean Absolute Error (MAE)
> rf_mae <- mean(abs(y_test - rf_pred), na.rm = TRUE)
> cat("Mean Absolute Error (MAE):", round(rf_mae, 2), "\n")
Mean Absolute Error (MAE): 14.51
>
> # Mean Squared Error (MSE)
> rf_mse <- mean((y_test - rf_pred)^2, na.rm = TRUE)
> cat("Mean Squared Error (MSE):", round(rf_mse, 2), "\n")
Mean Squared Error (MSE): 332.15
>
> # Root Mean Squared Error (RMSE)
> rf_rmse <- sqrt(rf_mse)
> cat("Root Mean Squared Error (RMSE):", round(rf_rmse, 2), "\n")
Root Mean Squared Error (RMSE): 18.22
>
> # Mean Absolute Percentage Error (MAPE)
> rf_mape <- mean(abs((y_test - rf_pred) / y_test), na.rm = TRUE) * 100
> cat("Mean Absolute Percentage Error (MAPE):", round(rf_mape, 2), "%\n")
Mean Absolute Percentage Error (MAPE): 1.59 %
```

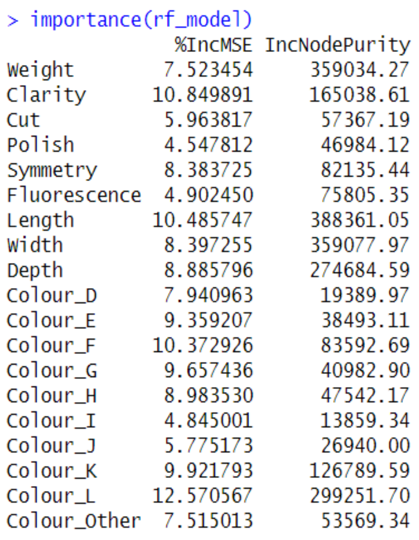
The output from the Random Forest model indicates it was the best-performing model among all applied algorithms on this dataset. With an **R-squared of 62.28%**, Random Forest captures a significant portion of the variability in the target variable (Price), demonstrating strong predictive power.

Key metrics such as **Mean Absolute Error (MAE)** of 14.51, **Mean Squared Error (MSE)** of 332.15, and **Root Mean Squared Error (RMSE)** of 18.22 highlight the model's accuracy and reliability. Furthermore, the **Mean Absolute Percentage Error (MAPE)** of just 1.59% confirms its exceptional precision compared to other models.

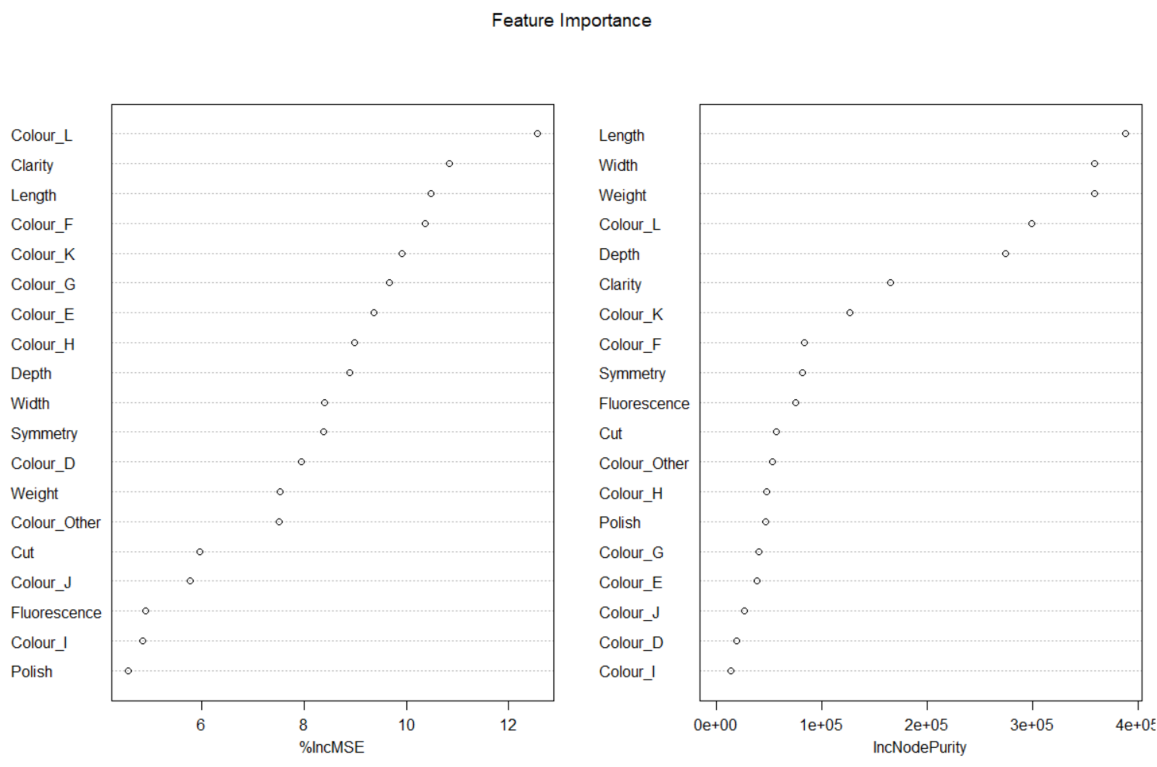
The superior performance of Random Forest is due to its ensemble approach, which reduces overfitting and improves generalization by combining multiple decision trees. These results make it the most suitable model for predicting price in this dataset.

Feature Importance

Feature importance in Random Forest refers to a metric that indicates the relative contribution of each feature (or predictor) to the predictive accuracy of the model. It helps identify which features are most influential in predicting the target variable. Random Forest computes feature importance based on how much a particular feature improves the split quality or prediction performance in the ensemble of decision trees.



```
> importance(rf_model)
             %IncMSE IncNodePurity
Weight      7.523454     359034.27
Clarity    10.849891     165038.61
Cut         5.963817      57367.19
Polish      4.547812      46984.12
Symmetry    8.383725      82135.44
Fluorescence 4.902450     75805.35
Length     10.485747     388361.05
Width       8.397255     359077.97
Depth       8.885796     274684.59
Colour_D    7.940963      19389.97
Colour_E    9.359207      38493.11
Colour_F   10.372926      83592.69
Colour_G    9.657436      40982.90
Colour_H    8.983530      47542.17
Colour_I    4.845001      13859.34
Colour_J    5.775173      26940.00
Colour_K    9.921793     126789.59
Colour_L   12.570567     299251.70
Colour_Other 7.515013     53569.34
```



The graphs provide a visual representation of these metrics:

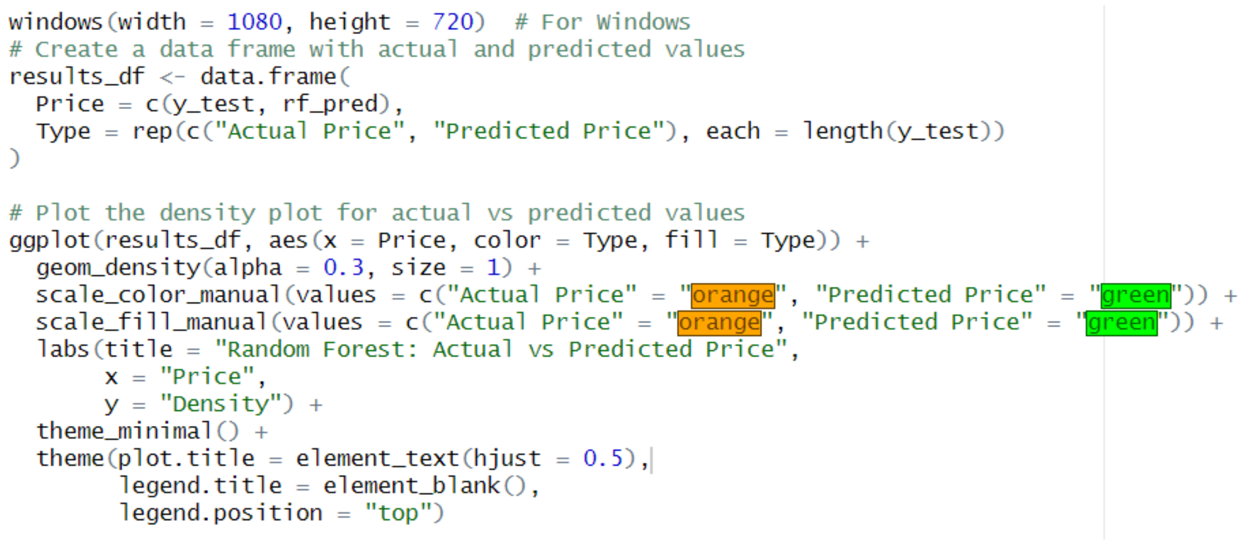
- The left plot ranks features based on **%IncMSE**, emphasizing their effect on predictive accuracy.
- The right plot ranks features by **IncNodePurity**, showing their contribution to model splits.

Key Insights:

- **Colour_L** is consistently the most critical feature, as it has the highest %IncMSE.
- **Weight**, **Length**, and **Clarity** are significant contributors across both metrics, indicating their importance for model performance.
- Features like **Polish** and **Colour_I** have relatively lower importance, which might make them candidates for exclusion or dimensionality reduction.

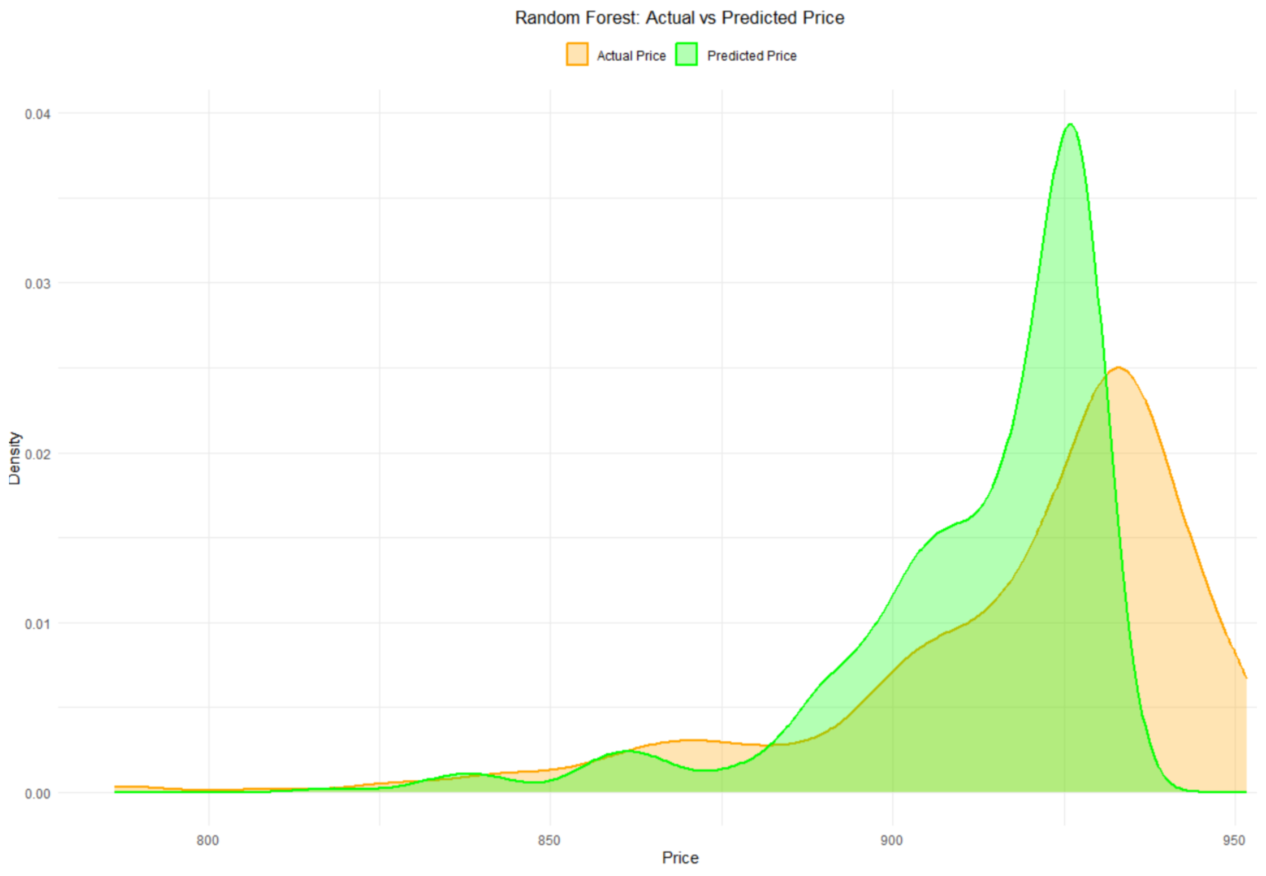
This analysis highlights the most impactful features, helping to understand the model and prioritize the influential variables for further optimization or interpretation.

Plotting the density plot showing Actual Vs Predicted Price



```
windows(width = 1080, height = 720)  # For Windows
# Create a data frame with actual and predicted values
results_df <- data.frame(
  Price = c(y_test, rf_pred),
  Type = rep(c("Actual Price", "Predicted Price"), each = length(y_test))
)

# Plot the density plot for actual vs predicted values
ggplot(results_df, aes(x = Price, color = Type, fill = Type)) +
  geom_density(alpha = 0.3, size = 1) +
  scale_color_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  scale_fill_manual(values = c("Actual Price" = "orange", "Predicted Price" = "green")) +
  labs(title = "Random Forest: Actual vs Predicted Price",
       x = "Price",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        legend.position = "top")
```

Output:



This density plot compares the actual prices (orange) and the predicted prices (green) using the Random Forest model. The close alignment of the peaks indicates that the Random Forest effectively captures the patterns in the data, leading to accurate predictions.

Conclusion

This project aimed to predict diamond prices based on various features such as weight, clarity, cut, color, and dimensions by employing different machine learning models and techniques. Throughout the project, we performed extensive data preprocessing, exploratory data analysis, feature engineering, and model evaluations to improve the accuracy of predictions and derive meaningful insights.

Key Insights and Steps:

1. **Preprocessing and Feature Engineering:**
    - Categorical variables such as clarity, cut, polish, and symmetry were grouped and encoded to reduce noise and simplify the prediction process. Rare levels were combined into "Other" categories for features like color, and numeric transformations like standardization were applied to ensure uniform scaling across variables.
    - Polynomial features (squared and cubic terms) were introduced to capture non-linear relationships between predictors and the target variable, significantly improving model performance.
2. **Exploratory Data Analysis:**
    - Count plots, box plots, and density plots helped uncover the distribution of categorical features and their relationship with price.
    - Correlation analysis revealed that features like weight, clarity, and length were the most influential in determining the price.
3. **Model Development and Comparison:**
    - A range of regression models were implemented, including Linear Regression, Ridge Regression, Lasso Regression, Polynomial Regression, Decision Trees, and Random Forest.
    - Polynomial regression improved upon the basic linear model by incorporating interaction terms and higher-degree features, achieving an R-squared value of ~50.94%, with a Mean Absolute Error (MAE) of 15.76.
    - Decision Trees provided interpretable results but slightly lower accuracy (R-squared ~46.66%, MAE 16.61) compared to Polynomial Regression.

    - Random Forest emerged as the best-performing model with an R-squared of 62.28%, a low MAE of 14.51, and a MAPE of 1.59%, effectively capturing complex interactions between features without overfitting.

4. **Feature Importance:**
    - Random Forest's feature importance analysis revealed that length, weight, clarity, and width were the most significant predictors. This reinforces the business intuition that physical dimensions and clarity greatly influence diamond prices.

Analysis of Model Performances:

- **Linear Models** were efficient for initial insights but lacked the complexity to model non-linear relationships adequately.
- **Regularization Techniques (Lasso and Ridge)** helped address multicollinearity and slightly improved model performance by penalizing less relevant features.
- **Polynomial Regression** captured non-linear dependencies and achieved a notable improvement in R-squared compared to simple linear models.
- **Decision Trees** provided visual interpretability but were less accurate and prone to overfitting on the training data.
- **Random Forest**, leveraging an ensemble of decision trees, struck an optimal balance between accuracy and generalization, making it the most robust and reliable model for this dataset.

Final Observations:

- The use of Random Forest not only provided the best predictive accuracy but also offered feature importance rankings, which can guide stakeholders in identifying key contributors to diamond pricing.
- Incorporating polynomial features further enhanced the performance of linear models, emphasizing the importance of non-linear relationships in real-world datasets.
- The detailed visualizations, including density plots and decision tree hierarchies, allowed for a comprehensive understanding of the model behavior and prediction accuracy.
- By leveraging Random Forest, we have developed a reliable tool that can serve as a foundation for future predictive analytics in the diamond industry.

Future Scope

This project focused on predicting diamond prices using various machine learning techniques, achieving significant insights and reliable predictions. However, there is substantial potential for further enhancement and expansion of the project. Below are key areas for future scope:

1. Incorporating Additional Features:

- **Market Data Integration:** Including external data such as market demand, seasonal price trends, and regional preferences can make predictions more aligned with real-world fluctuations.
- **Supplier and Manufacturer Data:** Incorporating details about the diamond's origin, brand, or certification body could improve the model's ability to predict premium prices for specific diamonds.

2. Advanced Machine Learning Models:

- **Gradient Boosting Algorithms:** Models such as XGBoost, LightGBM, or CatBoost could be explored to achieve even higher accuracy and better handling of feature importance.
- **Deep Learning Models:** Neural networks could be applied to capture complex patterns in the dataset, particularly if more features or a larger dataset is available.

3. Hyperparameter Tuning:

- Optimization techniques such as Grid Search or Bayesian Optimization could be implemented to fine-tune model parameters for Random Forest and other algorithms, potentially improving performance.

4. Feature Engineering Enhancements:

- Adding interaction terms or exploring non-linear transformations could further enhance the performance of regression models.
- Dimensionality reduction techniques like Principal Component Analysis (PCA) could be applied to simplify the model without losing significant predictive power.

5. Deployment and Real-World Applications:

- The predictive model can be deployed as a web-based application or API for real-time diamond price estimation, benefiting jewelers, traders, and customers.

- Integration into e-commerce platforms can provide automatic price suggestions based on uploaded diamond specifications.

6. Explainability and Interpretability:

- Developing interpretable models using SHAP (SHapley Additive exPlanations) values or LIME (Local Interpretable Model-agnostic Explanations) can offer deeper insights into individual predictions, making the model more user-friendly for stakeholders.

7. Scalability and Big Data Handling:

- Expanding the dataset to include millions of diamonds with broader attributes can provide a more robust model capable of handling diverse inputs.
- Leveraging cloud-based platforms like AWS or Google Cloud for distributed computing can enhance scalability.

8. Dynamic Updates and Continuous Learning:

- Implementing a feedback mechanism where the model learns from new data periodically can keep predictions accurate over time.
- Incorporating online learning algorithms would allow the model to adapt dynamically to changing market trends.

9. Expanding to Other Applications:

- The framework can be extended to predict prices for other gemstones or luxury goods, leveraging similar features such as quality, size, and certification.
- A recommendation system could be built to suggest diamonds based on customer preferences and budget.

By expanding the project across these dimensions, the predictive model can become a highly valuable tool in the jewelry industry. It can aid stakeholders in optimizing pricing strategies, enhancing customer experience, and gaining a competitive edge in the market.

References

- R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Available at: https://www.r-project.org

- Towards Data Science: *Guide to Machine Learning Models in R*. Available at: https://towardsdatascience.com

- Stack Overflow discussions for resolving specific implementation issues.

- https://www.kaggle.com/

- https://srk.one/

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer.
    - A practical guide to statistical and machine learning techniques with R code examples, covering regression, tree-based methods, and model evaluation.