OPIM 407

Case II

Selin Ceydeli

Sinan Yılmaz

Göktuğ Okyarlar

02.05.2023

A. INTRODUCTION: Introducing the primary objective of this study and the report

Numerous factors influence a customer's decision to buy a car, making second-hand car appraisal a complex task, given the diverse car models, features, and accessories. Adam Ebraham, the Chief Marketing Officer of an automobile agency, has been assigned the responsibility of determining the manufacturer's suggested retail price (MSRP) for each vehicle and forwarding it to dealers within a week to finalize the car prices. The primary challenge for Ebraham is identifying the essential car features that determine the price. To achieve this goal, Ebraham has decided to employ neural networks for price prediction to accurately appraise the cars.

In Section B of the report, we will provide a brief introduction to the data. In Section C, we will present the results of the exploratory data analysis conducted using the R programming language, aiming to analyze and investigate the data to understand the main characteristics of the variables. In Section D, we will discuss the factors influencing a customer's decision to buy a car and implement feature selection. In Section E, we will explore three potential neural network architectures for predicting automobile prices. In Section F, we will implement and interpret a price prediction system using multiple linear regression. Finally, in Section G, we will present our conclusions, summarizing the main findings of this case study.

B. BRIEF INTRODUCTION TO DATA

2.1 The Range, Number of Variables, and Observations

The dataset assembled for the recent collection of second-hand cars comprises 1367 records, with each entry representing a distinct second-hand car that has been appraised. Each car is characterized by 28 variables that provide detailed information about the vehicle. The first column of the dataset contains the price data, which serves as the dependent variable. The other 27 variables function as independent variables, encompassing the various features presumed to influence the pricing of these cars. Figure 1 presents the R code used to determine the dimensions of the gathered dataset:

```
> # Get the dimensions of the dataset (rows and columns)
> dimensions <- dim(cars)
> dimensions
[1] 1367   28
```

Figure 1. Number of Observations and Variables

The prices of the second-hand cars range from 4,400 CA$ to 32,550 CA$, resulting in a price range of 28,150 CA$, as illustrated in Figure 2. The other variables in the dataset are utilized to examine the impact of various factors on the fluctuating prices of these second-hand cars.

```
> # Find the minimum and maximum values of the price column
> min_price <- min(cars$Price)
> max_price <- max(cars$Price)
>
> # Calculate the range
> price_range <- max_price - min_price
>
> # Print the results
> cat("Minimum price:", min_price, "\n")
Minimum price: 4400
> cat("Maximum price:", max_price, "\n")
Maximum price: 32550
> cat("Price range:", price_range, "\n")
Price range: 28150
```

Figure 2. The Range of Car Prices

2.2 New List of Attributes in the Data

Table 1 below displays a list of the variables in the data, specifying each feature and describing its type and measurement unit.

Table 1. Revised Version of Exhibit 1 Continued from the Case Study

Variable	Description	Feature Type	Measurement Unit
Price	Offer price of the car	Quantitative	Min: 4400, Max: 32550 in CA$
Age	Age of the car in months	Quantitative	Min: 5, Max: 81 in Months
KM	Distance the car has been driven	Quantitative	Min: 3, Max: 232942 in Kilometers
Fuel	Type of fuel used by the car	Categorical	Petrol, Diesel, CNG
HP	Horsepower of the car's engine	Quantitative	Min: 72, Max:195 in Horsepower (hp)
MC	Metallic color	Categorical	Yes = 1, No = 0
Color	Color of the car	Categorical	Blue, red, grey, silver, black, …
Auto	Whether the car has an automatic transmission	Categorical	Yes = 1, No = 0
CC	Engine displacement	Quantitative	Min: 1300, Max: 16000 in Cubic centimeters (cc)
Drs	Number of doors	Quantitative	Min: 2, Max: 5
Cyl	Number of cylinders in the engine	Quantitative	Min: 3, Max: 3
Grs	Number of gears in the transmission	Quantitative	Min: 3, Max: 6

Wght	Weight of the car	Quantitative	Min: 1004, Max: 1619 in kilograms
G_P	Guarantee period in months	Quantitative	Min: 4, Max: 20 in months
Mfr_G	Within manufacturer's guarantee period	Categorical	Yes = 1, No = 0
ABS	Whether the car has anti-lock brake system	Categorical	Yes = 1, No = 0
Abag_1	Presence of the driver airbag	Categorical	Yes = 1, No = 0
Abag_2	Presence of passenger airbag	Categorical	Yes = 1, No = 0
AC	Presence of automatic air conditioning	Categorical	Yes = 1, No = 0
Comp	Presence of board computer	Categorical	Yes = 1, No = 0
CD	Presence of CD player	Categorical	Yes = 1, No = 0
Clock	Presence of central lock	Categorical	Yes = 1, No = 0
Pwin	Presence of powered windows	Categorical	Yes = 1, No = 0
PStr	Presence of power steering	Categorical	Yes = 1, No = 0
Radio	Presence of radio	Categorical	Yes = 1, No = 0
SpM	Sport model	Categorical	Yes = 1, No = 0
M_Rim	Presence of metallic rims	Categorical	Yes = 1, No = 0
Tow_Bar	Presence of a tow bar	Categorical	Yes = 1, No = 0

C. EXPLORATORY DATA ANALYSIS

Adam Ebraham, the chief marketing officer of the automobile agency, provided Yusuf Farid, the data scientist, with the data collected for the recent collection of second-hand cars. The careful analysis of the data is crucial for determining which car features are essential and to what extend each contributed to the price. The aim of the exploratory data analysis of this dataset is to understand the relationship between the price variable and the 27 independent variables to guide us with the feature selection.

3.1 The Dataset: Raw Data-Order and Sample

Before making analysis on the dataset, the missing values should be handled. Therefore, we checked if there exist any missing values in our dataset using R's is.na() function and as can be seen in Figure 3, we observed that there aren't any missing values in either of the columns:

```
> # Are there any null values in the dataset?
> sapply(cars, function(x) sum(is.na(x))) #There aren't any NA values
   Price      Age       KM     Fuel       HP       MC    Color     Auto       CC      Drs      Cyl      Grs     Wght      G_P   Mfr_G      ABS  Abag_1 
       0        0        0        0        0        0        0        0        0        0        0        0        0        0        0        0        0 
 Abag_2       AC     Comp       CD    Clock     Pwin    PStr    Radio     SpM  M_Rim Tow_Bar 
       0        0        0        0        0        0        0        0        0        0        0 
```

Figure 3. Missing Value Analysis

Figure 4 below displays the structure of the dataset. Nearly all variables in the dataset are numerical except for the fuel type and color variables. Fuel type and color are represented as categorical variables:

```
> # View the structure of the dataset
> str(cars)
'data.frame':   1367 obs. of  28 variables:
 $ Price  : int  21000 20000 19650 21550 22550 22050 22800 18000 16800 17000 ...
 $ Age    : int  26 23 26 32 33 29 31 25 25 31 ...
 $ KM     : int  31463 43612 32191 23002 34133 18741 34002 21718 25565 64361 ...
 $ Fuel   : chr  "Petrol" "Petrol" "Petrol" "Petrol" ...
 $ HP     : int  195 195 195 195 195 195 195 113 113 113 ...
 $ MC     : int  0 0 0 1 1 0 1 1 0 1 ...
 $ Color  : chr  "Silver" "Red" "Red" "Black" ...
 $ Auto   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ CC     : int  1800 1800 1800 1800 1800 1800 1800 1600 1600 1600 ...
 $ Drs    : int  3 3 3 3 3 3 3 3 3 3 ...
 $ Cyl    : int  3 3 3 3 3 3 3 3 3 3 ...
 $ Grs    : int  6 6 6 6 6 6 5 5 5 5 ...
 $ Wght   : int  1189 1189 1189 1189 1189 1189 1189 1109 1069 1109 ...
 $ G_P    : int  10 4 4 4 4 4 4 20 4 4 ...
 $ Mfr_G  : int  1 1 1 1 1 1 1 0 0 1 ...
 $ ABS    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Abag_1 : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Abag_2 : int  1 1 1 1 1 1 1 0 1 1 ...
 $ AC     : int  1 1 1 1 1 1 1 0 1 0 ...
 $ Comp   : int  0 1 1 1 1 1 1 0 1 1 ...
 $ CD     : int  1 0 0 1 1 0 1 0 1 1 ...
 $ Clock  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Pwin   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ PStr   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Radio  : int  0 0 0 0 0 0 0 1 0 0 ...
 $ SpM    : int  0 1 1 1 1 1 0 0 0 1 ...
 $ M_Rim  : int  1 1 1 1 1 1 1 0 0 0 ...
 $ Tow_Bar: int  0 0 0 0 0 0 0 1 0 0 ...
```

Figure 4. Structure of the Dataset

Lastly, we looked at the first 10 rows of the dataset to understand how data is stored in the dataset, which is displayed in Figure 5:

```
> # Displaying the first 10 observations
> head(cars, 10)
   Price Age    KM   Fuel  HP MC  Color Auto   CC Drs Cyl Grs Wght G_P Mfr_G ABS Abag_1 Abag_2 AC Comp CD Clock Pwin PStr
1  21000  26 31463 Petrol 195  0 Silver    0 1800   3   3   6 1189  10     1   1       1       1  1    0  1     1    1    1
2  20000  23 43612 Petrol 195  0    Red    0 1800   3   3   6 1189   4     1   1       1       1  1    1  0     1    1    1
3  19650  26 32191 Petrol 195  0    Red    0 1800   3   3   6 1189   4     1   1       1       1  1    1  0     1    1    1
4  21550  32 23002 Petrol 195  1  Black    0 1800   3   3   6 1189   4     1   1       1       1  1    1  1     1    1    1
5  22550  33 34133 Petrol 195  1   Grey    0 1800   3   3   6 1189   4     1   1       1       1  1    1  1     1    1    1
6  22050  29 18741 Petrol 195  0   Grey    0 1800   3   3   6 1189   4     1   1       1       1  1    1  0     1    1    1
7  22800  31 34002 Petrol 195  1   Grey    0 1800   3   3   5 1189   4     1   1       1       1  1    1  1     1    1    1
8  18000  25 21718 Petrol 113  1   Blue    0 1600   3   3   5 1109  20     0   1       1       0  0    0  0     1    1    1
9  16800  25 25565 Petrol 113  0   Grey    0 1600   3   3   5 1069   4     0   1       1       1  1    1  1     1    1    1
10 17000  31 64361 Petrol 113  1   Grey    0 1600   3   3   5 1109   4     1   1       1       1  0    1  1     1    1    1
   Radio SpM M_Rim Tow_Bar
1      0   0     1       0
2      0   1     1       0
3      0   1     1       0
4      0   1     1       0
5      0   1     1       0
6      0   1     1       0
7      0   0     1       0
8      1   0     0       1
9      0   0     0       0
10     0   1     0       0
```

Figure 5. Displaying the First 10 Observations

3.2 Data Visualizations

Bar charts can be used to visualize the distribution of categorical variables, showing their frequency in the dataset. In Figure 6, it is seen that the fuel type categorical variable takes two values in the dataset. This box plot demonstrates that diesel cars vary in price, for instance we observe a wider range of expensive cars and cheap cars in diesel fuel type. However, with the petrol cars, a less variance in terms of price is observed. According to our initial observation of the fuel type variable, it can be concluded that fuel type of a car is an important factor on its price.

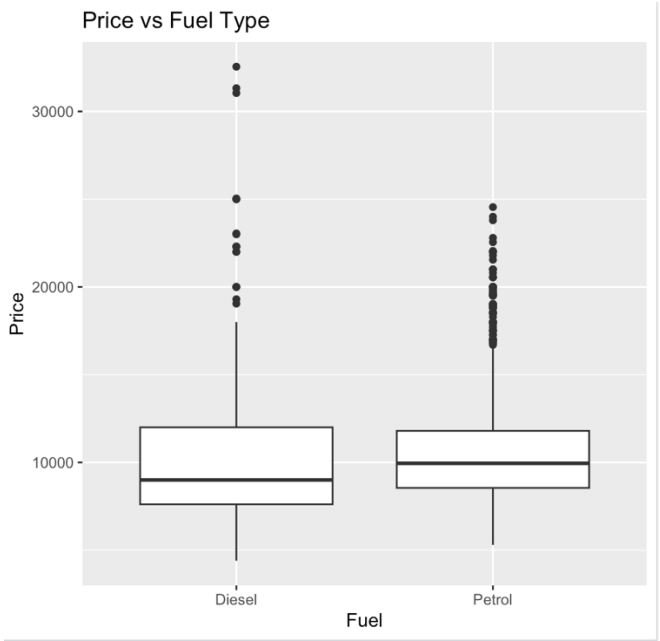

Figure 6. Box Plot for Price vs Fuel Type

For our analysis of the relationship between the color and price variables, we decided to use a box plot, as color is a categorical variable. The box plot in Figure 7 shows that the medians of the colors are relatively similar, but some gray colored cars have the highest prices. It's worth noting that the dataset contains very few yellow cars, which are mostly concentrated within a narrow price range. For these differences in the distribution of cars with different cars, color might be an important feature determining price.

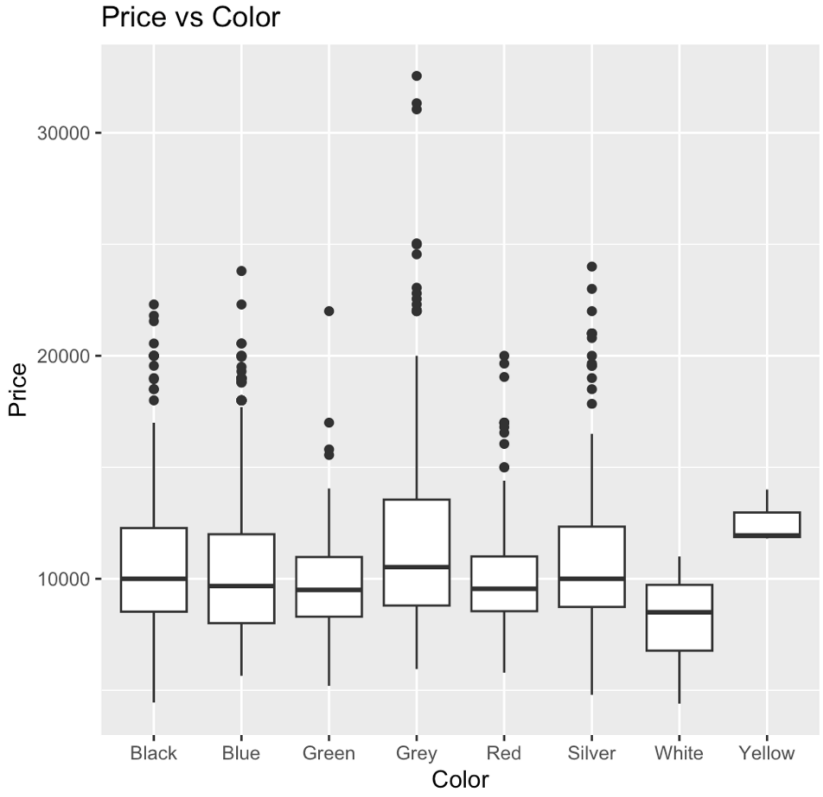

Figure 7. Box Plot of price of different colored cars

To visualize the linear relationship between the price variable and age and accumulated kilometers variables, two scatter plots are drawn side by side as shown in Figure 8. The aim is to see the overall trend in the data and strength of the relationship between these continuous variables so that whether the age and accumulated kilometers features are essential and contribute to the price.

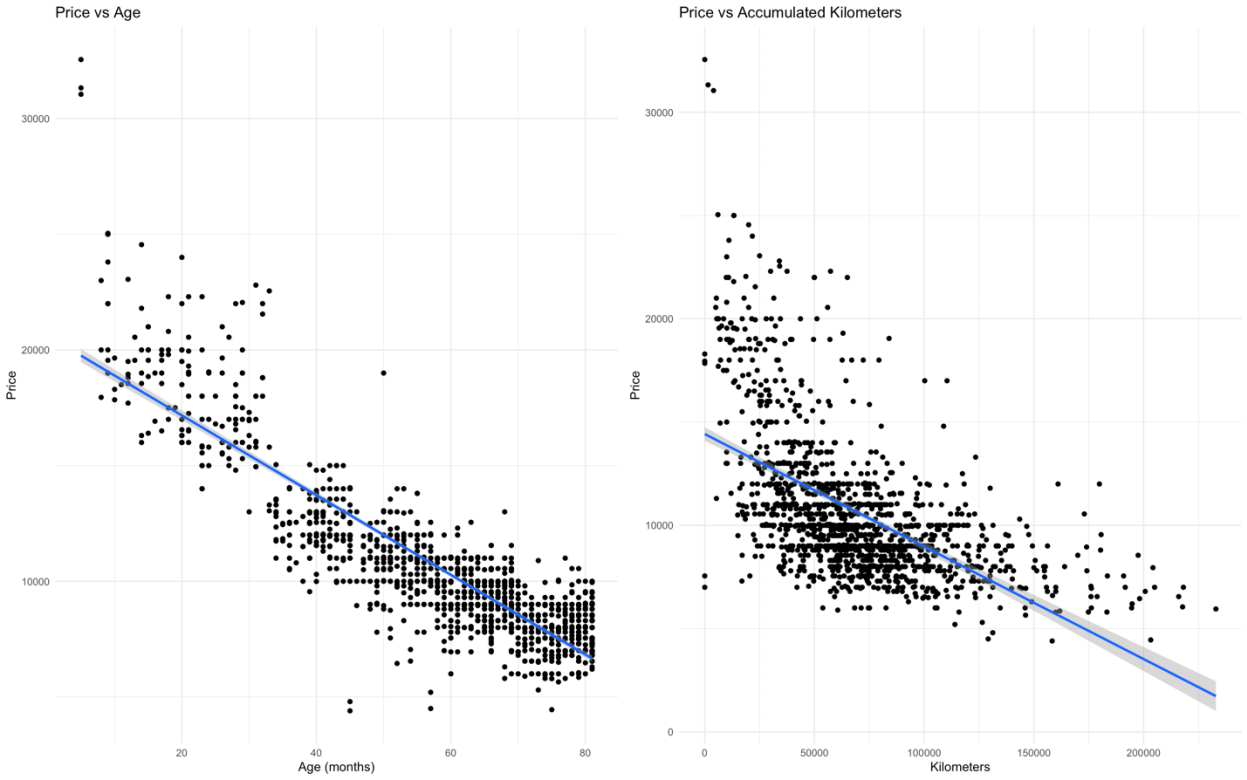

Figure 8. Scatter Plots for Price vs. Continuous Variables Age and Accumulated Kilometers

The scatter plots demonstrate that there is a strong negative relationship between price of a car and its age as well as between its price and its accumulated kilometers. Therefore, both age and accumulated kilometers variables can be chosen in the feature selection.

Furthermore, a parallel coordinates plot is drawn to display the relationships between the technical car features (Age, HP, CC, and Cyl) and the car price. Parallel coordinates plot help to identify relationships between multiple variables, especially when analyzing high-dimensional data, and can provide insights into correlations, clusters, and outliers. As shown in Figure 9 below, the colors of the parallel coordinates plot represent the price range, with blue representing lower prices and red representing higher prices:

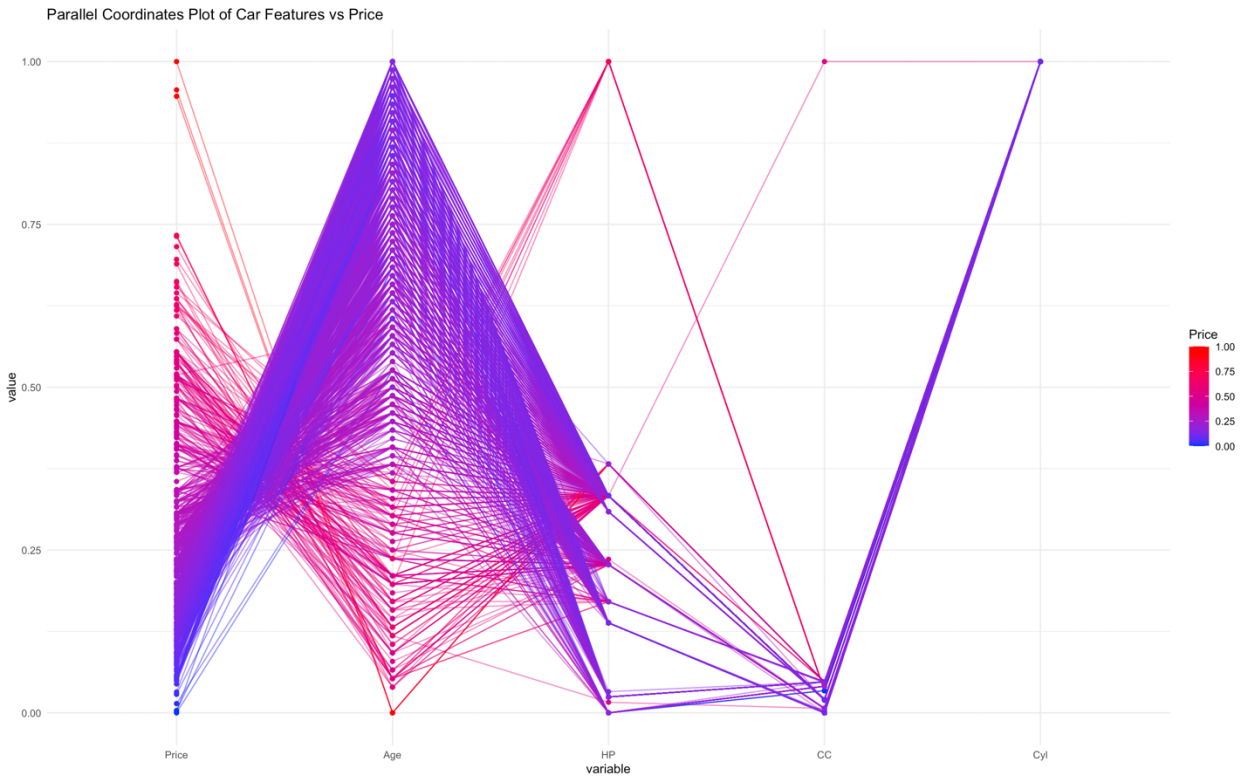

Figure 9. Parallel Coordinates Plot of Car Features vs Price

By examining the plot in Figure 9, several patterns and trends in the data are observed and conclusions about the importance of different car features in determining car prices is made. Firstly, it is observed that higher-prices cars tend to have lower age values since the red colored lines (corresponding to high price) are accumulated for the lower values of age variable. Conversely, it is observed that lower-price cars tend to have higher age values since the blue colored lines (corresponding to low price) are accumulated for the higher values of age variable.

Secondly, it is observed that the cyl variable has the same value for all prices, which means that this feature does not provide any predictive information for the car price and thus, should not be selected as one of the features.

Lastly, in the HP and CC variables, no obvious relationship is observed because the red and blue lines are placed in a mixed fashion. For example, we cannot make any conclusions such as higher-priced cars tend to have higher horsepower and larger CC values or conversely, lower-priced cars tend to have lower horsepower and smaller CC values. Therefore, we are inconclusive about the importance of HP and CC features. Further visualizations are needed to explore these variables.

Next, to analyze the effect of variables AC, radio, M-rim, and Towbar on price in another plot, we changed the values of the variables from numerical to categorical. While transforming them to categorical, we took 1 as a yes and 0 as a no. Figure 10 describes the box plots of these four variables with respect to price:

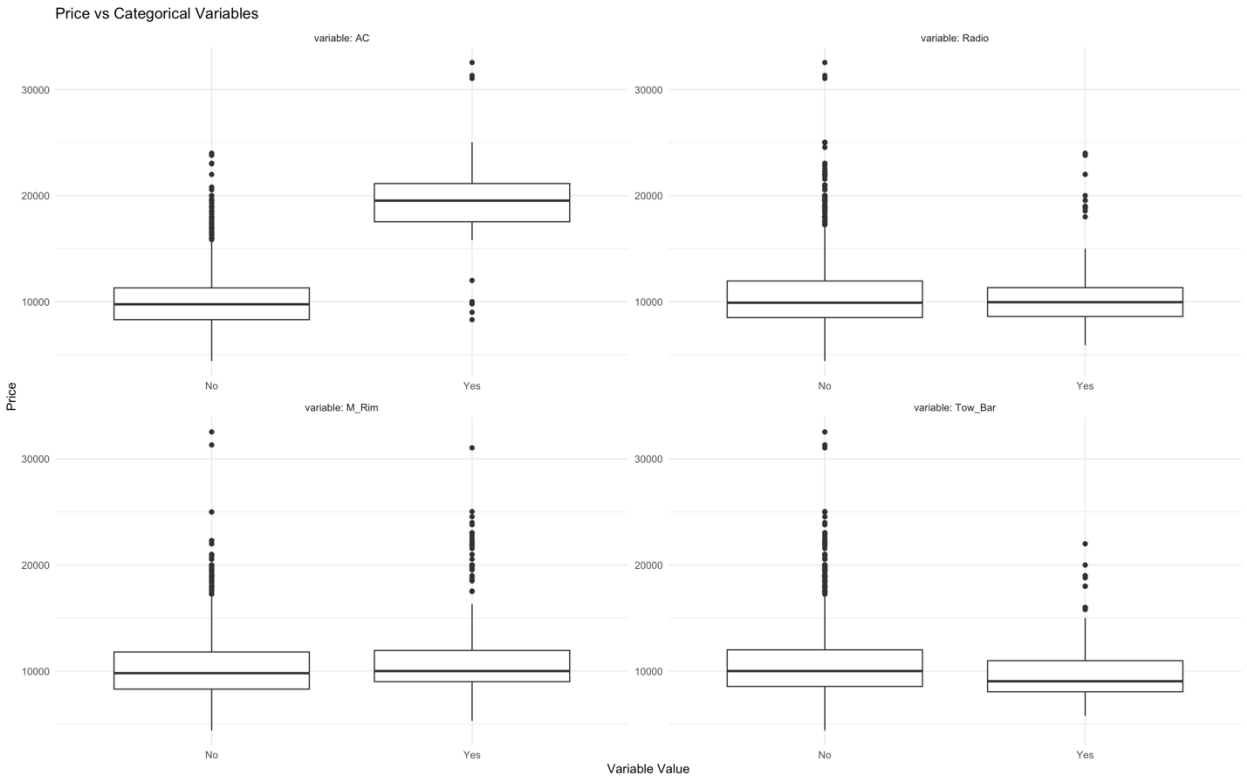

Figure 10. Price vs AC/m-rim/Radio/towbar

From Figure 10, it can be inferred that the features Radio (upper-right plot), M-rim (lower-left plot) and Tow_Bar (lower-right plot) have almost no effect on determining the price value because the box plots for “No” and “Yes” cases were nearly the same except for the outliers for all three of these variables. For instance, a car with a radio, or a car without a radio have the same mean with respect to price. However, AC feature on a car influences the price. The mean price of the cars with an AC is significantly higher than the mean price of the cars without an AC.

The box plots in Figure 11 below display the effect of the presence of a metallic color on the price of a car. We can compare the distributions of the cars with metallic color and cars without to get a sense of the importance of the metallic color in determining the price.

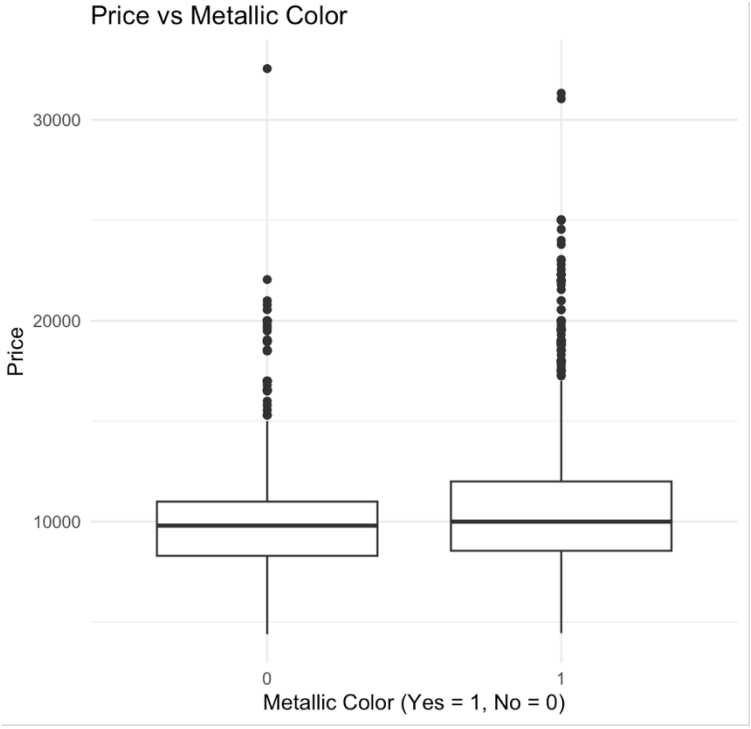

Figure 11. Box Plots of Price vs Metallic Color

Based on the box plot in Figure 11, we can see that the medians of car prices for both metallic and non-metallic colors are quite similar. However, there appears to be more density on the upper whisker of the box plot of the metallic-colored cars, which suggests that metallic-colored cars may be more expensive. While the box plot provides some insight into the relationship between metallic color and price, it is not a definitive indicator and should be interpreted with caution during feature selection.

In the dataset, we observed two features related with transmission mechanism of a car: AUTO feature which keeps the information whether the car has a direct or automatic gear and GRS feature which is the number of gear positions. In Figure 12 below, the box plot visualizes the transmission features separated by number of gear positions. At each gear position, the data is divided according to gear type; direct being number 0, automatic being number 1. In Figure 10, it is observable that there is almost no car having 3 and 4 gears. Data has been accumulated over 5 and 6 gears. In the dataset, almost all cars having 6-gear are also direct. On the other hand, the 5-gear cars can be both automatic and direct. However, it is observed that among the 5-gear cars, the direct ones are more expensive. Therefore, both the gear type and gear number of a car are important features for determining price.

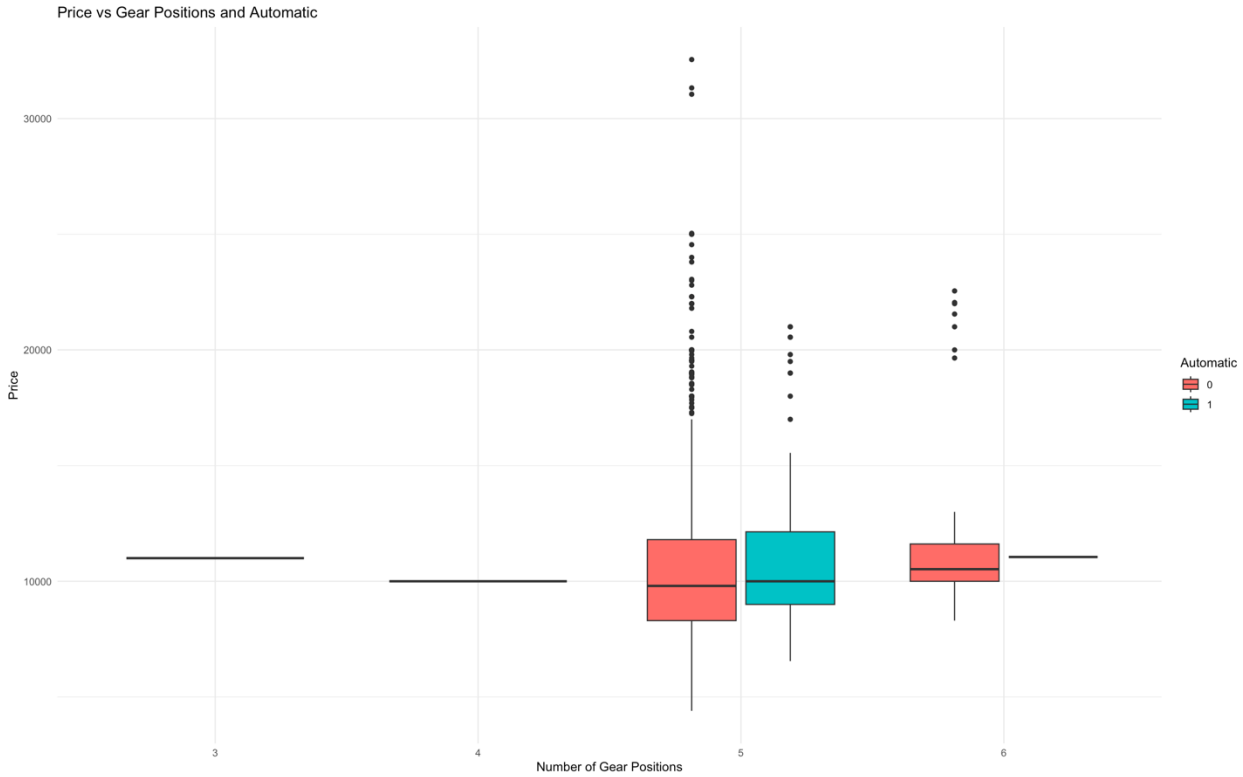

Figure 12. Price vs Transmission Features

To analyze the impact of the door number on the price of a car, we created a jitter plot showing the distribution of data points for different number of doors with respect to price, as shown in Figure 13:

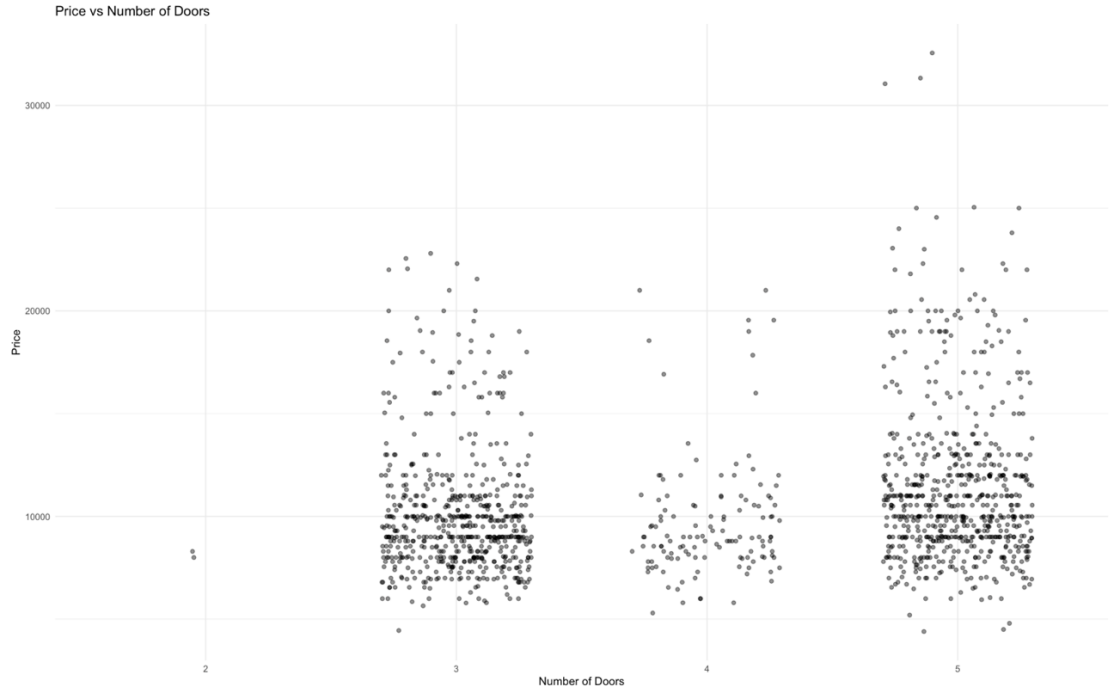

Figure 13. Jitter Plot of Price vs Number of Doors

After plotting Figure 13, we realized a problem with the number of doors column. The number of doors concept is differently understood by different second-hand car retailers. By analyzing the data, we assume that some retailers described the cars as 2-doored cars while some described the same type of cars as 3-doored counting the trunk as a door as well. The same anomaly is present for describing 4-doored cars, sometimes as 4-doored and sometimes as 5-doored cars. This anomaly in the dataset makes the analysis harder so we decided to handle by combining the values 2 and 3 because we assume they indicate to the same type of cars, and likewise 4 and 5.

After updating the door numbers in the dataset (i.e. updating value 2 as 3 and value 4 as 5), another jitter plot is drawn showing the distribution of data points for different number of doors with respect to price. The revised jitter plot is displayed in Figure 14:

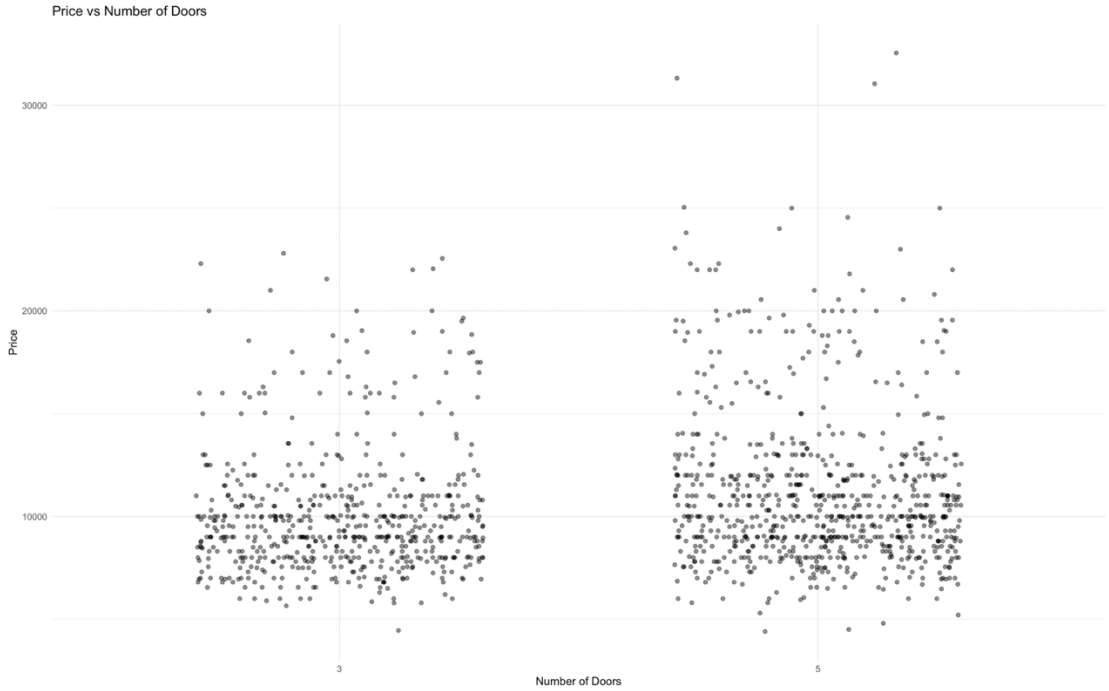

Figure 14. Revised Jitter Plot of Price vs Number of Doors

Figure 14 demonstrates that there exist more expensive cars with 5 doors than the cars with 3 doors. That's why number of doors may be an important feature for determining car price.

To analyze effect other technological features, which are ABS, Abag_1, Abag_2, Comp, CD, Clock, Pwin and Pstr, again box plot has been used since all the features mentioned are Boolean values. In the figure 15 below, it could be observed that all features have differences in the box area of their box plots, medians, and outliers with respect to the two categories: having or not having the respected feature in the car. So, the technological features are determining features of price.

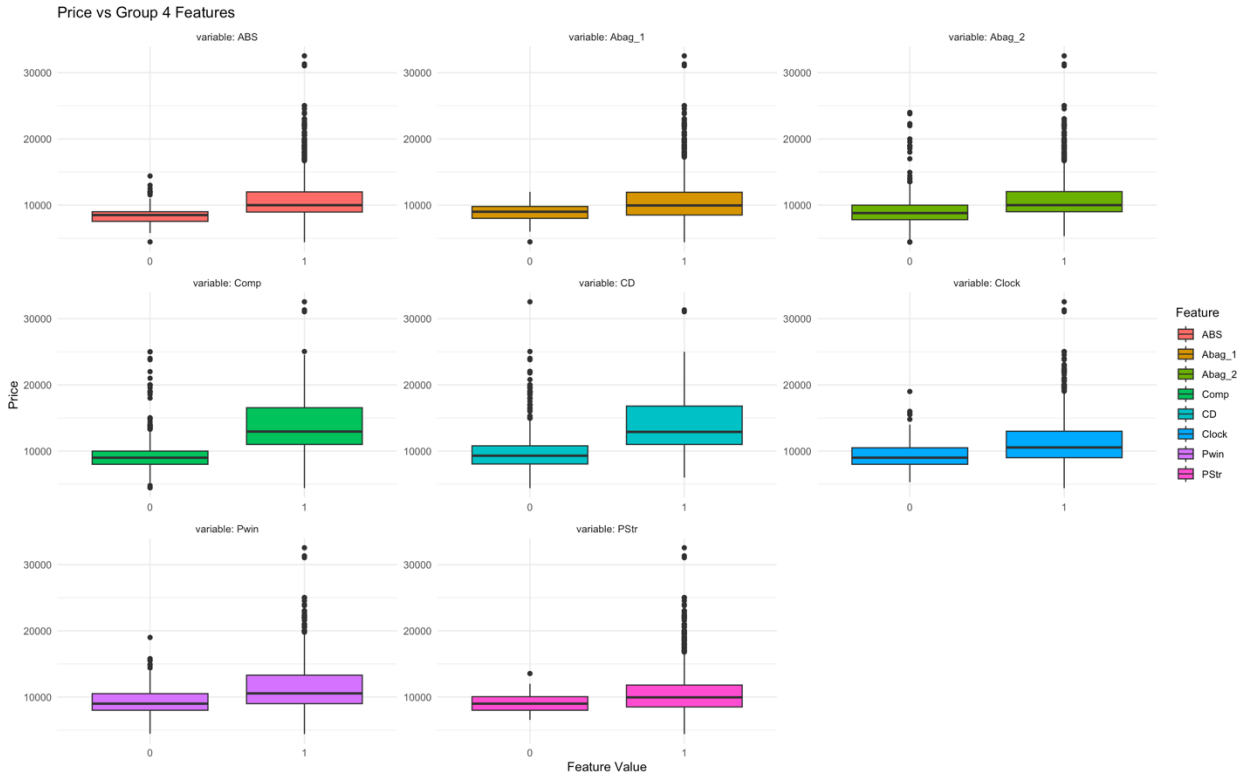

Figure 15. Cumulative analysis of Technological features

Furthermore, the effect of being a sport model in determining the car price is visualized by drawing another box plot, as shown in Figure 16, and it is observed that the box plot corresponding to sport model (left box plot) has a significantly larger box area containing higher prices and thus, it can be concluded that the sport model feature is important in determining car price.

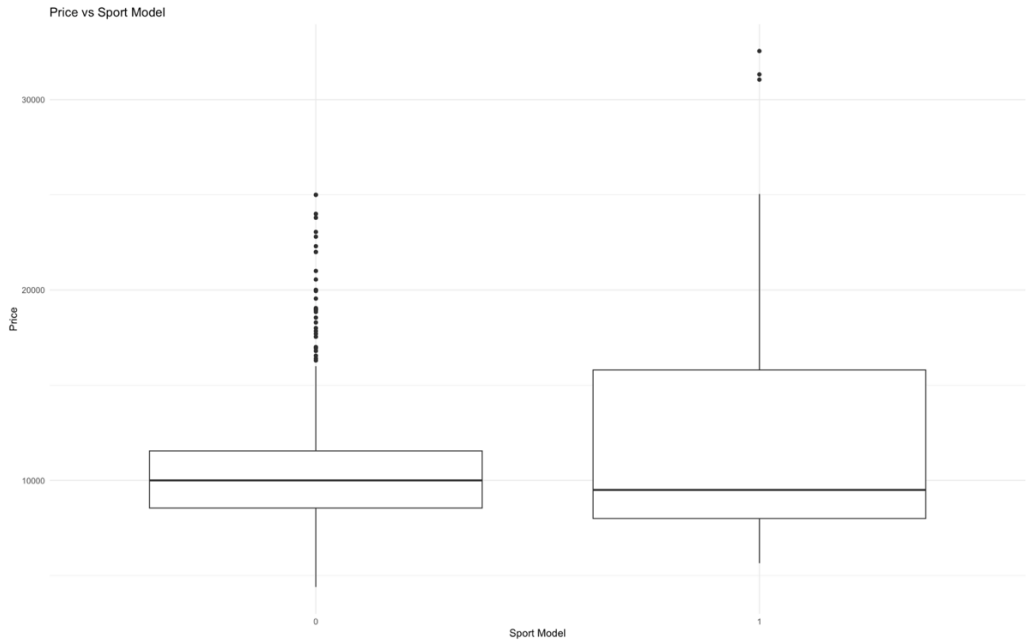

Figure 16. Box Plot of Price vs Sport Model

In the dataset, we identified two features related to the guarantee documents of a car: Within Manufacturer's Guarantee and Guarantee Period. To explore the impact of guarantee documents on car prices, we created a box plot. For the Within Manufacturer's Guarantee feature, which takes Boolean values (0 indicating no guarantee and 1 indicating a guarantee), we further divided the data into three categories based on the Guarantee Period: (4,8], (8,12], and (16,20], as illustrated in Figure 17 below.

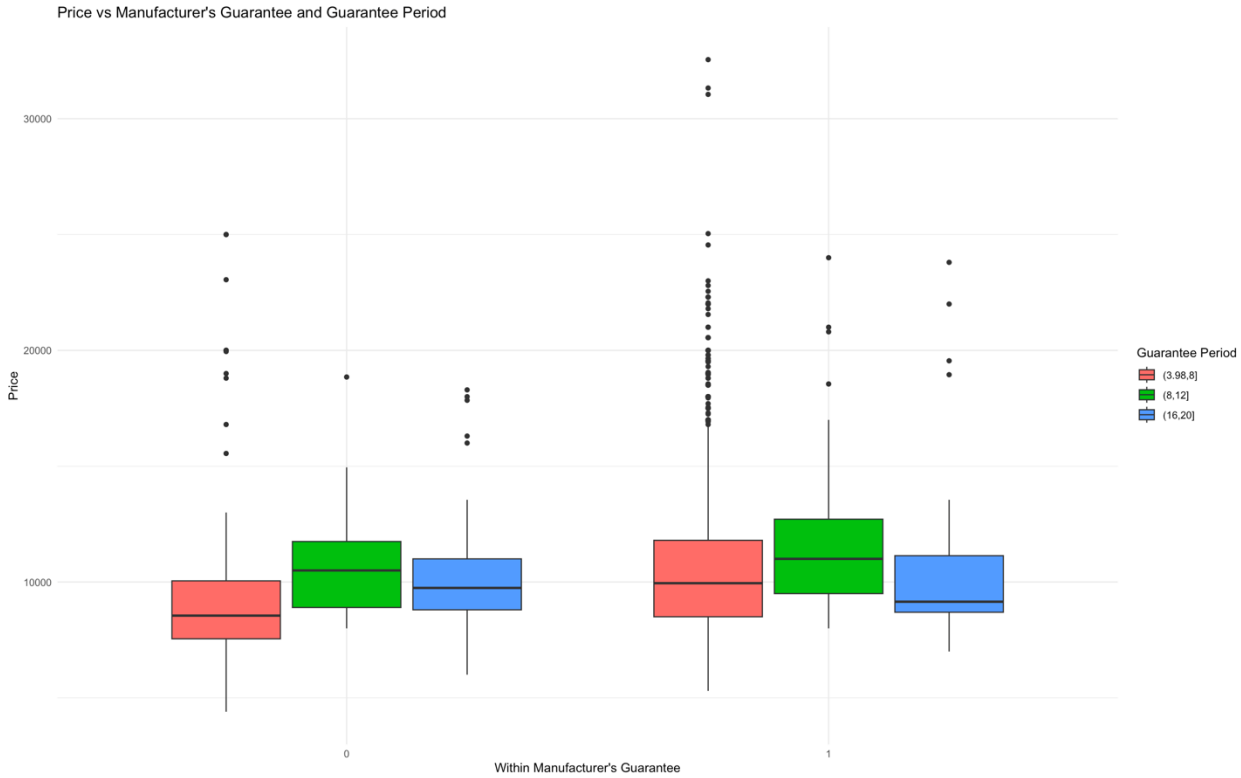

Figure 17. Box Plot of Price vs Manufacturer's Guarantee and Guarantee Period

Upon comparing the box plots in Figure 17, we found no clear evidence of a distinct relationship between guarantee documents and car prices. While the box plot provides some initial insights, it is not enough to conclude that guarantee features determine the car price.

D. FEATURE SELECTION

4.1 Factors Influencing a Customer's Decision to Buy a Car

In our view, budget is the most crucial factor influencing a customer's decision to purchase a car. Therefore, setting appropriate prices for second-hand vehicles is essential. Along with budget considerations, brand reputation, safety features, performance, car size, interior space, and fuel efficiency are all significant factors that impact buying decisions. Aesthetic appeal also plays

a vital role, as evidenced by the exploratory data analysis of the dataset, where external features such as car color, metallic finish, and sport model status affect pricing and, consequently, customer preferences. In recent years, environmental impact has emerged as another important consideration for buying decisions, with environmentally conscious customers prioritizing low-emission or electric vehicles to minimize their ecological footprint.

When it comes to second-hand cars, additional factors come into play, with age and accumulated mileage being the most important. As illustrated in Figure 8, scatter plots demonstrate that both age and mileage significantly influence a car's price, indicating that these factors play an important role in a customer's decision to purchase a second-hand vehicle.

4.2 Implementing Feature Selection

There are several feature selection approaches that can be used to identify the key predictors in the dataset. The three of the most common feature selection approaches are: Filter methods, wrapper methods, and embedded methods. In this section, all three feature selection methods will be introduced and implemented in R.

4.2.1 Filter Method: Correlation-based Feature Selection

The filter method is a feature selection approach that involves selecting features based on their intrinsic properties and relevance to the target variable. Filter method is computationally efficient because it does not involve training machine learning model during the feature selection process. It is plausible to use the filter method when dealing with high-dimensional data, involving many variables (Gupta).

Among the filter methods, correlation-based feature selection approach will be implemented to identify the key predictors in the dataset. Before implementing correlation-based feature selection, first the dataset is partitioned into training and testing sets, as shown in the R code in Figure 18. 80% of the data is used for the training set and the remaining 20½ is used for the testing dataset.

```
> # Split the dataset into training and testing sets:
> set.seed(123) # For reproducibility
> trainIndex <- createDataPartition(cars$Price, p = 0.8, list = FALSE, times = 1)
> train <- cars[trainIndex,]
> test <- cars[-trainIndex,]
```

Figure 18. Splitting the Dataset

Figure 19 displays the conversion of character variables (Fuel and Color variables) into factors. As part of preprocessing the data, the character variables are converted into factors because

factors efficiently represent categorical data by internally using integers instead of character strings.

```
> # Convert character variables to factors
> train$Fuel <- as.factor(train$Fuel)
> train$Color <- as.factor(train$Color)
```

Figure 19. Converting Character Variables into Factors

As the first step of implementing correlation-based feature selection, the correlation between the numeric features and the target variable is computed and the correlations are sorted in descending order. After sorting, the top 10 features with the highest correlation are chosen as the predictors. The complete implementation of the correlation-based feature selection using the training dataset is shown in Figure 20:

```
> # Implement the Correlation-based Feature Selection using the training dataset:
> # Calculate the correlation between numeric features and the target variable
> correlations <- cor(train[, sapply(train, is.numeric)])
Warning message:
In cor(train[, sapply(train, is.numeric)]) : the standard deviation is zero
> correlations_with_target <- correlations["Price",]
>
> # Sort the correlations in descending order
> sorted_correlations <- sort(abs(correlations_with_target), decreasing = TRUE)
>
> # Select the top k features with the highest correlation (e.g., k = 10)
> k <- 10
> top_k_features <- names(sorted_correlations)[2:(k+1)]  # Add 1 to exclude the 'Price' variable itself
>
> cat("Top", k, "features based on correlation:\n", top_k_features, "\n")
Top 10 features based on correlation:
 Age Wght AC Comp KM CD Pwin Clock HP ABS
```

Figure 20. Implementing the Correlation-based Feature Selection Approach

The top 10 features selected based on correlation are: Age, Wght, AC, Comp, KM, CD, Pwin, Clock, HP, and ABS.

4.2.2 Wrapper Method: Forward Feature Selection

The wrapper method is a feature selection approach that follows a greedy search process, evaluating all possible combinations of features to select the optimal feature subset. This thorough approach of considering all possible subsets against an evaluation criterion allows wrapper methods to generally perform better in feature selection compared to filter methods (Gupta).

Among the wrapper methods, the forward feature selection approach will be implemented to identify the key predictors in the dataset. Forward feature selection is an iterative approach that starts with an empty set of features and iteratively adds the best feature at each step until the desired number of features are selected (Gupta).

Again, the steps of splitting the dataset into training and testing sets and converting the character variables into factors are implemented as preprocessing steps prior to applying forward feature selection method. Following data preprocessing, a formula for the linear regression model using all predictor variables except Price is created to specify the structure of the model and define the predictors and the target variable (i.e. Price). After that, forward feature selection is applied using the training dataset and the optimal number of features are selected using the adjusted R-squared criterion. Lastly, the selected features are extracted. The complete implementation of the forward feature selection using the training dataset is shown in Figure 21:

```
> # Approach 2 - Wrapper Method: Forward Feature Selection
>
> #install.packages("leaps")
> library(leaps)
>
> # Create a formula for the linear regression model using all predictor variables except Price:
> predictors <- names(train[, !(names(train) %in% "Price")])
> formula <- as.formula(paste("Price ~", paste(predictors, collapse = " + ")))
>
> # Apply forward feature selection using the training dataset:
> fwd_selection <- regsubsets(formula, data = train, method = "forward", nvmax = length(predictors))
Reordering variables and trying again:
Warning message:
In leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax, force.in = force.in,  :
  1  linear dependencies found
> fwd_summary <- summary(fwd_selection)
>
> # Determine the optimal number of features using the adjusted R-squared criterion:
> optimal_n_features <- which.max(fwd_summary$adjr2)
> cat("Optimal number of features:", optimal_n_features, "\n")
Optimal number of features: 22
>
> # Extract the selected features:
> selected_features <- names(coef(fwd_selection, id = optimal_n_features))[-1] # Exclude intercept
> cat("Selected features using forward feature selection:\n", selected_features, "\n")
Selected features using forward feature selection:
 Age KM FuelPetrol HP ColorBlue ColorGreen ColorRed ColorWhite Auto CC Wght G_P Mfr_G ABS Abag_1 Abag_2 AC Comp CD Clock PStr Cyl
> length(selected_features)
[1] 22
```

Figure 21. Implementing the Forward Feature Selection Approach

The selected features using forward feature selection are Age, KM, FuelPetrol, HP, ColorBlue, ColorGreen, ColorRed, ColorWhite, Auto, CC, Wght, G_P, Mfr_G, ABS, Abag_1, Abag_2, AC Comp CD, Clock PStr, and Cyl.

We were skeptical about the result forward feature selection approach has yielded because even though the value of the Cyl variable is constant for all entries, this approach has chosen it as one of the predictors. A variable with a constant value in the dataset has no predictive power and thus, should not selected as one of the features.

4.2.3 Embedded Method: LASSO Regression

The embedded methods have the benefits of both feature selection methods: filter method and wrapper method. Embedded methods are advantageous in computational efficiency, model-

specific feature selection, and simultaneous model training and feature selection. Embedded methods iteratively select the most relevant features during the learning process typically by adding a regularization term into the objective function (Gupta).

Among the embedded methods, the LASSO regression method will be implemented to identify the key predictors in the dataset. LASSO regression is a linear regression method that adds an L1 regularization term into the objective function, which is calculated by the sum of the absolute values of the coefficients multiplied by a regularization parameter (lambda). LASSO performs feature selection by iteratively reducing lambda, LASSO shrinks some coefficients zero and iteratively exclude those features with zero coefficients from the model (Gupta).

Again, the steps of splitting the dataset into training and testing sets and converting the character variables into factors are implemented as preprocessing steps prior to applying forward feature selection method. Different from the previous methods, in LASSO regression, two more preprocessing steps are applied. First, to prepare the data for LASSO regression, categorical variables are converted into dummy variables and then, missing values in the data are imputed, as shown in Figure 22:

```
> # Preprocessing
> # Prepare the data for LASSO regression by converting the categorical variables to dummy variables:
> train_dummies <- model.matrix(~ . - 1, data = train[, -1])
>
> # Impute missing values
> train_dummies_imputed <- apply(train_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))
```

Figure 22. Data Preprocessing for LASSO Regression

Following data preprocessing, LASSO regression model is fitted using cross-validation to find the optimal lambda value. Then, the final LASSO model is fitted using the optimal lambda value and the selected features are extracted. The complete implementation of the LASSO regression using the training dataset is shown in Figure 23:

```
> # Fit the LASSO regression model using cross-validation to find the optimal lambda value:
> set.seed(123) # For reproducibility
> lasso_cv <- cv.glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1)
>
> # Find the optimal lambda value
> optimal_lambda <- lasso_cv$lambda.min
>
> # Fit the final LASSO model using the optimal lambda
> lasso_model <- glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1, lambda = optimal_lambda)
>
> # Extract selected features
> selected_features <- rownames(coef(lasso_model))[which(coef(lasso_model) != 0)[-1]]
>
> # Print the selected features
> cat("Selected Features:\n")
Selected Features:
> print(selected_features)
 [1] "Age"        "KM"         "HP"         "ColorGreen" "ColorWhite" "Wght"       "G_P"        "Mfr_G"      "AC"         "CD"
[11] "Pwin"
```

Figure 23. Implementing LASSO Regression

The selected features using LASSO regression are Age, KM, HP, ColorGreen, ColorWhite, Wght, G_P, Mfr_G, AC, CD, and Pwin.

4.3 The Comparison of the Results of Different Approaches

As we noted in section 4.2.2, we have reservations about the results of the forward feature selection approach because it identified the Cyl variable as one of the predictors. However, we know from our exploratory data analysis that the Cyl variable is constant across all entries and thus, does not hold any predictive power. Therefore, we decided not to rely on the features selected by the forward feature selection approach.

In the remaining approaches (correlation-based feature selection and LASSO regression), we did not observe any discrepancies. However, it is important to consider that in the correlation-based feature selection, we manually choose the number of features to be selected by the model (for this case, we chose k as 10). In contrast, the LASSO regression method selects features based on the optimal lambda value, which makes the selection process more data driven. Another factor that influenced our decision was that LASSO regression, being an embedded method, combines the benefits of both the filter and wrapper methods, making it a more robust choice for our feature selection model. Thus, we decided to use the selected features generated by the LASSO regression model as the key predictors in our prediction models, which are Age, KM, HP, ColorGreen, ColorWhite, Wght, G_P, Mfr_G, AC, CD, and Pwin.

E. PREDICTING AUTOMOBILE PRICES USING NEURAL NETWORKS

5.1 Introducing the Concept and Structure of Neural Networks

Neural networks involve creating a prediction system using nodes and connecting links, where these links have weights associated with them. These weighted links represent the information from the input features. At each iteration, neural networks learn and update the weights to solve the problem (Anand).

Our problem involves predicting automobile prices using neural networks using the features selected in the last section, which include Age, KM, HP, ColorGreen, ColorWhite, Wght, G_P, Mfr_G, AC, CD, and Pwin. There are three types of neural networks: Artificial Neural Network (ANN), Convolution Neural Network (CNN), and Recurrent Neural Network (RNN) (Anand). According to Anand, ANN is used for analyzing textual or tabular data, CNN is used for

image data, and RNN is used for time series data. Therefore, since we have a tabular data, we will use the Artificial Neural Network to predict automobile prices.

5.2 Preprocessing the Data of the Selected Features

Before implementing neural networks, the data for the selected features is normalized. Normalization is an important step because neural networks are sensitive to the scale of the input features. To prevent the network from giving undue importance to some features over others due to differences in scaling, all features must be on the same scale. To achieve this, all features are normalized to have a scale ranging from 0 to 1 and the R code implementing the normalization procedure is shown in Figure 24:

```
> # Normalize the values
> preprocess_params <- preProcess(train_selected, method = c("center", "scale"))
> train_selected_norm <- predict(preprocess_params, train_selected)
> test_selected_norm <- predict(preprocess_params, test_selected)
```

Figure 23. Normalizing the Data of the Selected Features

5.3 Implementing 3 Neural Network Architectures

Since neural networks is a black box approach, we do not know the internal prediction mechanism. Therefore, we decided to configure the neural networks with different number of hidden layers and neurons, allowing us to compare their performance and identify the best model. Below, the structure of each architecture and their implementation steps are explained in detail.

5.3.1 Architecture 1: One hidden layer with 10 neurons

The first architecture consists of a single hidden layer with 10 neurons. This simple architecture is chosen as a starting point because it is computationally efficient and might already provide satisfactory results (Anand). In the implementation, a formula for the neural network model is first defined, indicating that the target variable is the price. The neural network model is then trained using the neuralnet function from the neuralnet library. Lastly, the trained model is applied to the test dataset to compute the predicted car prices. The implementation of the neural network is shown in Figure 25. The resulting neural network model, with one hidden layer and 10 neurons, can be visualized as shown in Figure 26.

```
> # Architecture 1: One hidden layer with 10 neurons
> set.seed(123)
> nn_formula1 <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
> nn1 <- neuralnet(nn_formula1, data = data.frame(Price=train$Price, train_selected_norm), hidden = 10, linear.output = TRUE)
> pred_nn1 <- compute(nn1, test_selected_norm)$net.result
```

Figure 24. Implementation of Architecture 1

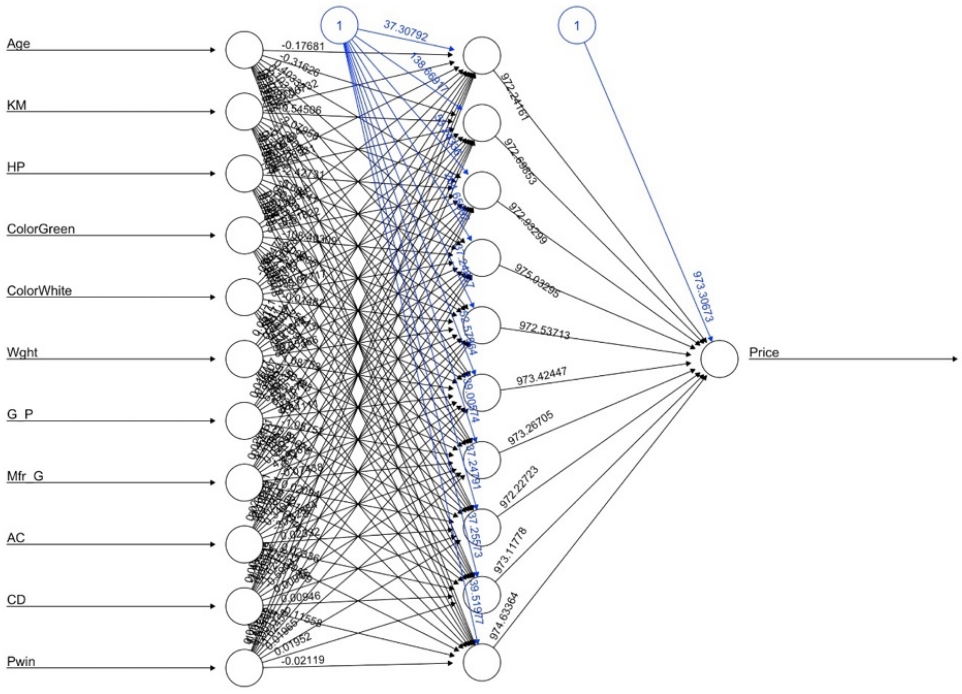

Figure 25. Visualization of Architecture 1 Layers

5.3.2 Architecture 2: One hidden layer with 15 neurons

Similar to the first architecture, the second architecture also has a single hidden layer, but with an increased number of neurons, i.e., 15 neurons. The increased number of neurons allows the model to capture more complex relationships in the data, potentially leading to better predictions. However, it also requires more computational resources and may lead to overfitting if the model becomes too complex (Anand). The implementation of architecture 2 includes one extra parameter compared to architecture 1: the stepmax parameter. The stepmax parameter provides more iterations for the neuralnet function and helps the algorithm converge to a solution. The default value for stepmax is 1e+05. Since we received a warning message while implementing architecture 2 stating that the "algorithm did not converge within the stepmax", we decided to increase the stepmax value from 1e+05 to 1e+06 to allow for more repetitions. The implementation of the neural network is shown in Figure 27. The resulting neural network model, with one hidden layer and 15 neurons, can be visualized as shown in Figure 28.

```
> # Architecture 2: One hidden layer with 15 neurons
> set.seed(123)
> nn_formula2 <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
> nn2 <- neuralnet(nn_formula2, data = data.frame(Price=train$Price, train_selected_norm), hidden = 15, linear.output = TRUE, stepmax = 1e+055)
> pred_nn2 <- compute(nn2, test_selected_norm)$net.result
```

Figure 26. Implementation of Architecture 2

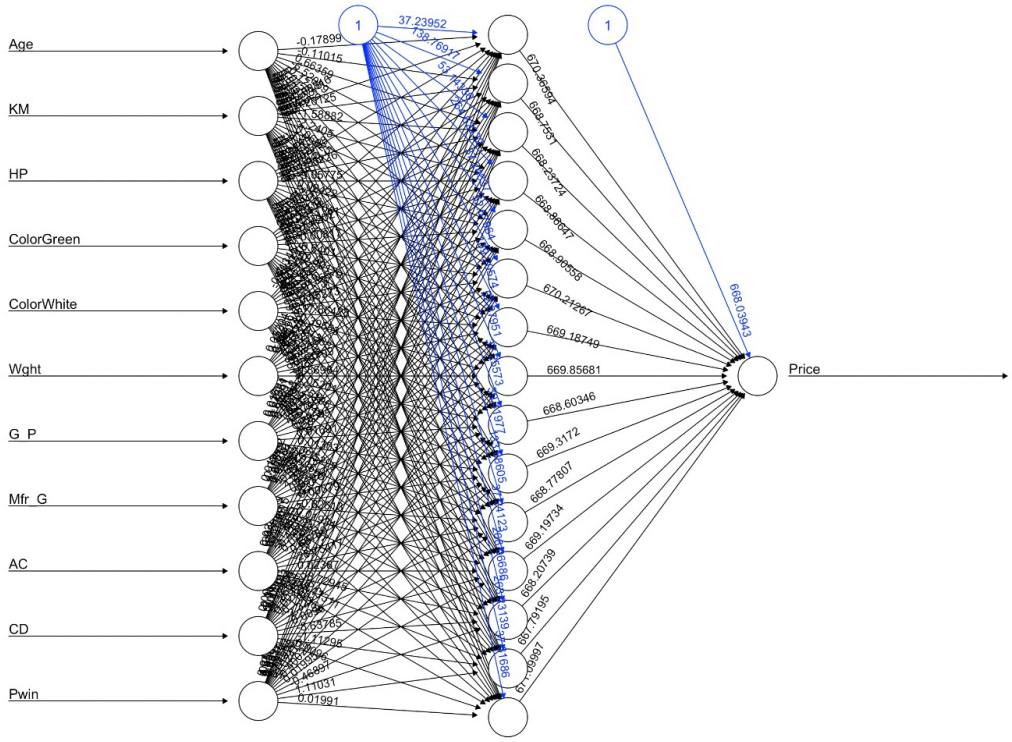

Figure 27. Visualization of Architecture 2 Layers

5.3.3 Architecture 3: Two hidden layers with 30 and 20 neurons

Unlike the first two architectures, the third architecture consists of two hidden layers with 30 and 20 neurons, respectively. Using multiple hidden layers allows the network to learn hierarchical representations of the data and capture even more complex relationships. Like with the second architecture, this comes at the cost of increased computational resources and the potential risk of overfitting (Anand). The implementation of the neural network is shown in Figure 29. The implementation of the third neural network architecture is nearly identical to the previous architectures, with the only difference being the hidden layers defined as a tuple c(30, 20) in the neuralnet function to indicate that there are two hidden layers with 30 and 20 neurons. The resulting neural network model, with two hidden layers containing 30 and 20 neurons, can be visualized as shown in Figure 30.

```
> # Architecture 3: Two hidden layers with 30 and 20 neurons
> set.seed(123)
> nn_formula3 <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
> nn3 <- neuralnet(nn_formula3, data = data.frame(Price=train$Price, train_selected_norm), hidden = c(30, 20), linear.output = TRUE)
> pred_nn3 <- compute(nn3, test_selected_norm)$net.result
```

Figure 28. Implementation of Architecture 3

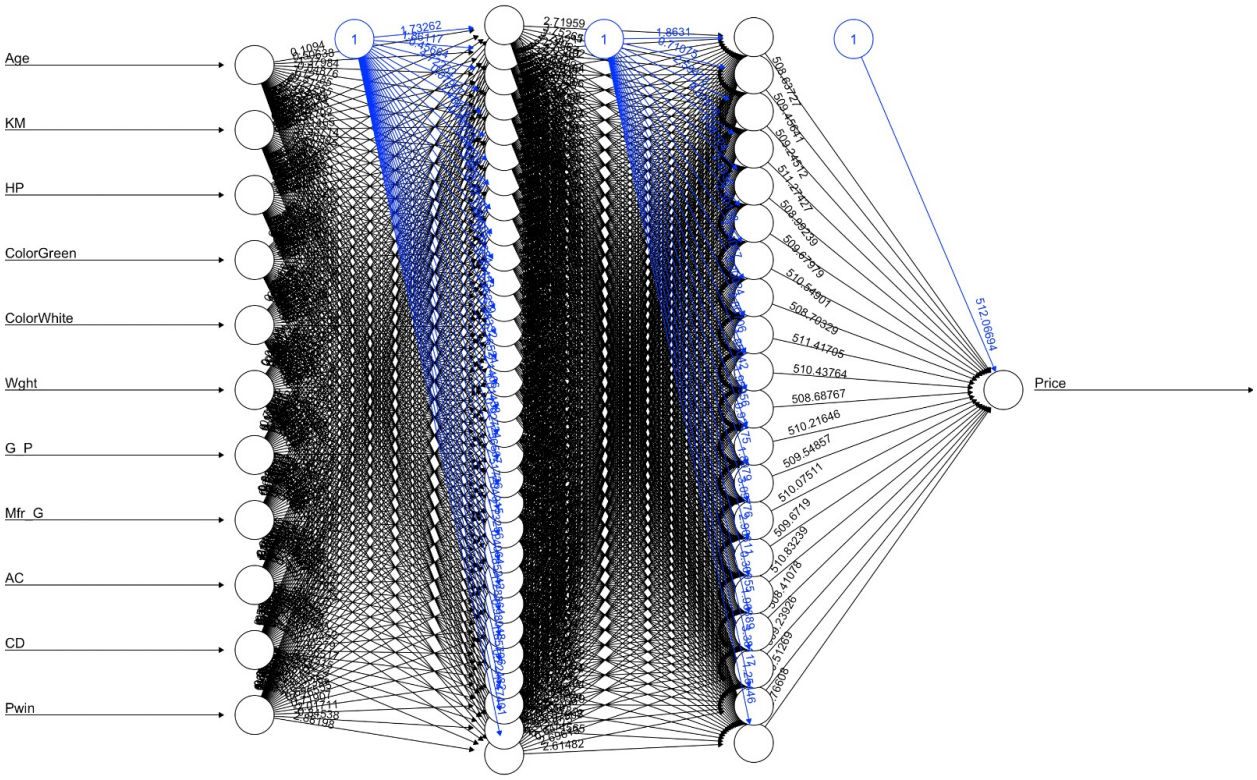

Figure 29. Visualization of Architecture 3 Layers

5.4 Evaluating the Predictive Performance of the Neural Networks

Before evaluating the performance of the neural network models, the results are scaled back to the original units of the predicted prices, as shown in Figure 30:

```
> # Scaling back to the original units of the predicted prices
> # First, find the original mean and standard deviation of the 'Price' column in the training data
> price_mean <- mean(train$Price, na.rm = TRUE)
> price_sd <- sd(train$Price, na.rm = TRUE)
>
> # Then, reverse the normalization process for the predictions
> pred_nn1 <- (pred_nn1 * price_sd) + price_mean
> pred_nn2 <- (pred_nn2 * price_sd) + price_mean
> pred_nn3 <- (pred_nn3 * price_sd) + price_mean
```

Figure 30. Scaling Back the Results

Each neural network algorithm will be evaluated separately by calculating the MSE, RMSE, and MAPE error metrics for each architecture. Subsequently, we will visualize the predicted values and real values on a lift chart. The results in Figures 32, 33, and 34 below indicate that the performance results for all three architectures are quite similar, with an MSE score of approximately 11,262, an RMSE of about 3,355.91, and a MAPE of roughly 23.1341. The MAPE score suggests that all the neural network architectures predicted the values with a 23 percent error rate, which is considered acceptable since it is less than 25 percent.

```
> # Calculate MSE, RMSE, and MAPE for each architecture
> mse_nn1 <- mean((test$Price - pred_nn1)^2)
> rmse_nn1 <- sqrt(mse_nn1)
> mape_nn1 <- mean(abs((test$Price - pred_nn1) / test$Price)) * 100
>
> cat("MSE, RMSE, and MAPE for Architecture 1 (nn1):\n")
MSE, RMSE, and MAPE for Architecture 1 (nn1):
> cat(paste("MSE:", mse_nn1, "RMSE:", rmse_nn1, "MAPE:", mape_nn1, "\n"))
MSE: 11262107.9274398 RMSE: 3355.90642411849 MAPE: 23.1340785757674
```

Figure 31. Evaluation of Architecture 1

```
> mse_nn3 <- mean((test$Price - pred_nn3)^2)
> rmse_nn3 <- sqrt(mse_nn3)
> mape_nn3 <- mean(abs((test$Price - pred_nn3) / test$Price)) * 100
>
> cat("MSE, RMSE, and MAPE for Architecture 3 (nn3):\n")
MSE, RMSE, and MAPE for Architecture 3 (nn3):
> cat(paste("MSE:", mse_nn3, "RMSE:", rmse_nn3, "MAPE:", mape_nn3, "\n"))
MSE: 11267368.319678 RMSE: 3356.69008394847 MAPE: 23.15408843912
```

Figure 32. Evaluation of Architecture 2

```
> mse_nn2 <- mean((test$Price - pred_nn2)^2)
> rmse_nn2 <- sqrt(mse_nn2)
> mape_nn2 <- mean(abs((test$Price - pred_nn2) / test$Price)) * 100
>
> cat("MSE, RMSE, and MAPE for Architecture 2 (nn2):\n")
MSE, RMSE, and MAPE for Architecture 2 (nn2):
> cat(paste("MSE:", mse_nn2, "RMSE:", rmse_nn2, "MAPE:", mape_nn2, "\n"))
MSE: 11276228.3784955 RMSE: 3358.00958582543 MAPE: 23.1786286088177
```

Figure 33. Evaluation of Architecture 3

The MAPE value of 23.1341 implies that, on average, the predictions made by our neural network architectures deviate from the actual values by 23.1341 percent. This indicates that our models can predict the target variable reasonably well, although there is still room for improvement.

The lift charts for the three neural network architectures, as shown in Figure 34, demonstrate that the models' predictions are most accurate around the 4th decile as the lines for actual and predicted prices intersect at around this decile. Before the 4th decile and after the 4th decile, the lines representing actual and predicted prices significantly deviate from each other, indicating that the model's predictive accuracy is lower in those ranges.

The pattern identified in the lift charts can be interpreted such that the models are better at predicting prices for cars that fall within the middle range of the dataset (around the 4th decile) but struggle with predicting prices for cars in the lower and higher ranges.

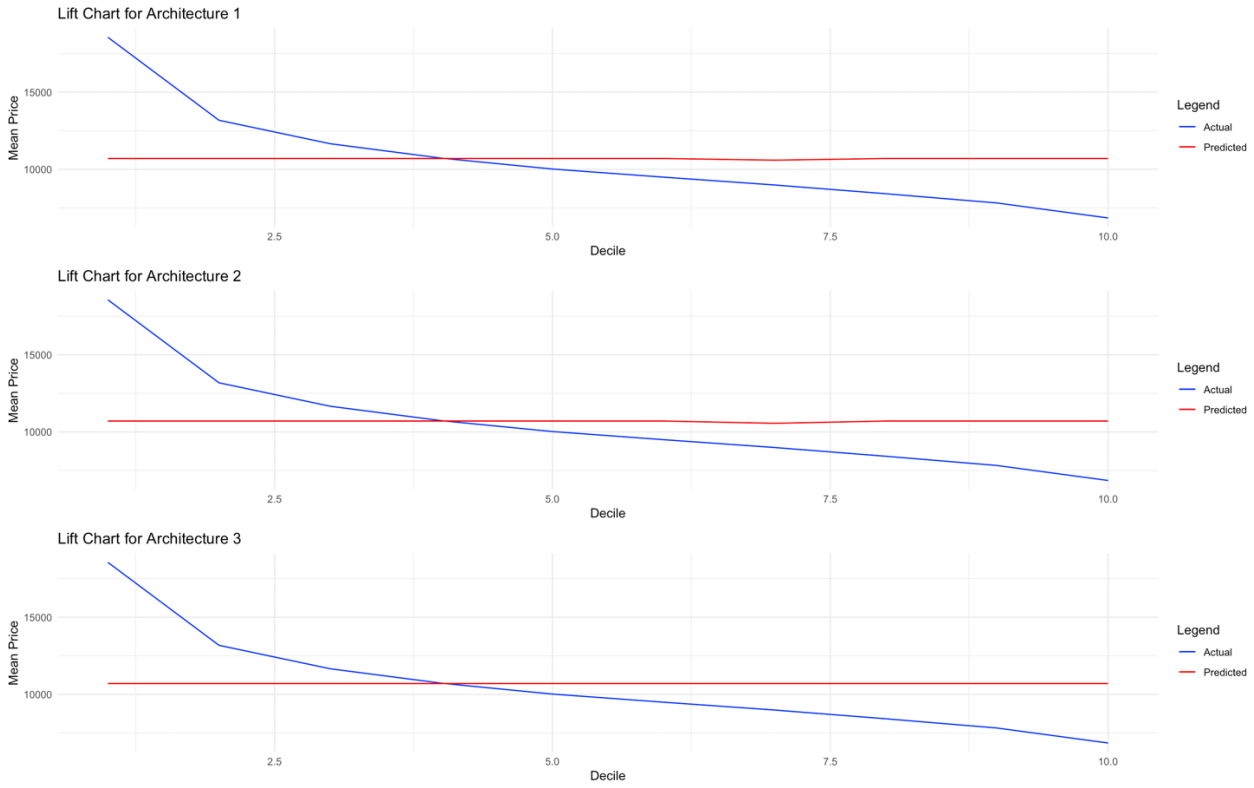

Figure 34. Lift Charts for Neural Network Architectures

5.5 Discussion about the Decisions and Recommendations

After evaluating the performance of the three neural network architectures using metrics such as MSE, RMSE, and MAPE, we found that all three architectures' performances were relatively similar. Based on this result, we would advise using the simplest neural network architecture (Architecture 1) because choosing a simpler architecture would result in reduced computational resources and a lower risk for overfitting (Anand).

Although a MAPE score of 23.1341 is considered acceptable, it still represents a substantial margin of error. To improve the performance of the neural network models, it is recommended to update them with new data, ideally with larger datasets that can help the models better understand the underlying trends and yield more accurate predictions. In conclusion, if additional data becomes available or more relevant features for predicting car prices are identified, it is advised to re-evaluate the neural network architectures.

5.6 Limitations of Neural Networks

Neural networks are often referred to as black boxes due to their complex inner workings. Although one can implement a neural network architecture and evaluate its performance, it can be challenging to understand how the model arrives at its predictions or processes the input data.

Another limitation of neural networks is their high dependency on the available data. The performance of a neural network architecture is directly influenced by the amount and quality of the data it is trained on (Rawat). In our case, the relatively high MAPE score of 23.1341 might be attributed to the limited size of our dataset, which consists of only 1367 observations. A larger dataset would potentially allow the neural network to capture more nuanced patterns and relationships in the data, leading to better predictive performance.

F. PREDICTING AUTOMOBILE PRICES USING LINEAR REGRESSION

6.1 Implementing Multiple Linear Regression

A multiple linear regression model is implemented using the features selected by the LASSO regression model as predictors. Before implementing the multiple regression model, new columns for the dummy variables in the train and test datasets for the Color factor variable are created so that the ColorGreen and ColorWhite features can be used as predictors in the model, as shown in Figure 35:

```
> # Create dummy variables for the 'Color' factor variable in the train dataset
> color_dummies_train <- model.matrix(~ Color - 1, data = train)
>
> # Convert the matrix to a data frame
> color_dummies_train_df <- as.data.frame(color_dummies_train)
>
> # Merge the dummy variables with the train dataset
> train <- cbind(train, color_dummies_train_df)
>
> # Create dummy variables for the 'Color' factor variable in the test dataset
> color_dummies_test <- model.matrix(~ Color - 1, data = test)
>
> # Convert the matrix to a data frame
> color_dummies_test_df <- as.data.frame(color_dummies_test)
>
> # Merge the dummy variables with the test dataset
> test <- cbind(test, color_dummies_test_df)
```

Figure 35. Creating Dummy Variables

After dummy variable creation, a formula using the selected features from the LASSO regression model is created to define the relationship between the target variable and the predictor variables. Then, a multiple linear regression model using the selected features is fitted and the summary of the model is generated, as shown in Figure 36:

```
> # Create a formula using the selected features from the LASSO regression model
> lasso_selected_formula <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
>
> # Fit the multiple linear regression model using the selected features
> lm_lasso_selected <- lm(lasso_selected_formula, data = train)
>
> # Summarize the model
> summary(lm_lasso_selected)

Call:
lm(formula = lasso_selected_formula, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-7036.7  -713.0    16.2   685.7  5296.6

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.752e+03  1.073e+03  -2.565 0.010461 *
Age         -1.087e+02  3.089e+00 -35.204  < 2e-16 ***
KM          -1.862e-02  1.294e-03 -14.393  < 2e-16 ***
HP           1.854e+01  2.748e+00   6.746 2.46e-11 ***
ColorGreen  -3.565e+02  9.992e+01  -3.568 0.000375 ***
ColorWhite  -1.119e+03  2.761e+02  -4.052 5.44e-05 ***
Wght         1.657e+01  9.078e-01  18.257  < 2e-16 ***
G_P          4.673e+01  1.031e+01   4.533 6.47e-06 ***
Mfr_G        7.331e+02  1.324e+02   5.535 3.90e-08 ***
AC           2.447e+03  1.931e+02  12.677  < 2e-16 ***
CD           2.615e+02  1.054e+02   2.480 0.013291 *
Pwin         3.828e+02  7.941e+01   4.820 1.64e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1205 on 1083 degrees of freedom
Multiple R-squared:  0.8894,    Adjusted R-squared:  0.8882
F-statistic: 791.4 on 11 and 1083 DF,  p-value: < 2.2e-16
```

Figure 36. Implementation of the Multiple Linear Regression Model

Using the multiple linear regression model and the test data previously generated during feature selection, which accounts for 20% of the original dataset, we made price predictions. Since we are predicting numerical data, the model's performance was evaluated by calculating the mean squared error (MSE), root mean squared error (RMSE), and mean absolute percentage error (MAPE) metrics. Figure 37 demonstrates the evaluation of the regression model:

```
> # Make predictions
> predictions <- predict(lm_lasso_selected, newdata = test)
>
> # Evaluate the model
> mse <- mse(test$Price, predictions)
> rmse <- sqrt(mse)
> abs_percentage_errors <- abs((test$Price - predictions) / test$Price) * 100
> MAPE <- mean(abs_percentage_errors)
>
> cat("Mean Squared Error (MSE):", mse, "\n")
Mean Squared Error (MSE): 1346313
> cat("Root Mean Squared Error (RMSE):", rmse, "\n")
Root Mean Squared Error (RMSE): 1160.307
> cat("Mean Absolute Percentage Error (MAPE):", MAPE, "\n")
Mean Absolute Percentage Error (MAPE): 8.598913
```

Figure 37. Evaluation of the Regression Model

These error metrics measure the performance of the multiple linear regression model in terms of prediction accuracy. An RMSE of 1,160.307 CA$ indicates that the model's predicted price values are approximately 1,160.307 CA$ away from the actual price values. To put the RMSE value into context, it should be compared with the range of the price variable, which was calculated in Section 2.1 to be 28,150 CA$. When the RMSE value is expressed as a percentage of the range of the price variable, an error percentage of 4.12% is obtained, which is low and signals that the model's predictive power is high.

Moreover, the MAPE value of the model is calculated to be 8.59%, which is also a low error percentage and reinforces the conclusion that the multiple linear regression model has high predictive power.

Figure 38 displays a lift chart, which is drawn to visualize the performance of our multiple linear regression model by comparing the actual price values to the predicted price values. It can be observed in Figure 38 that the line for the actual price values (in blue) and the line for the predicted price values (in red) closely follow each other. The closeness of the lines signals that our model has a high predictive performance.

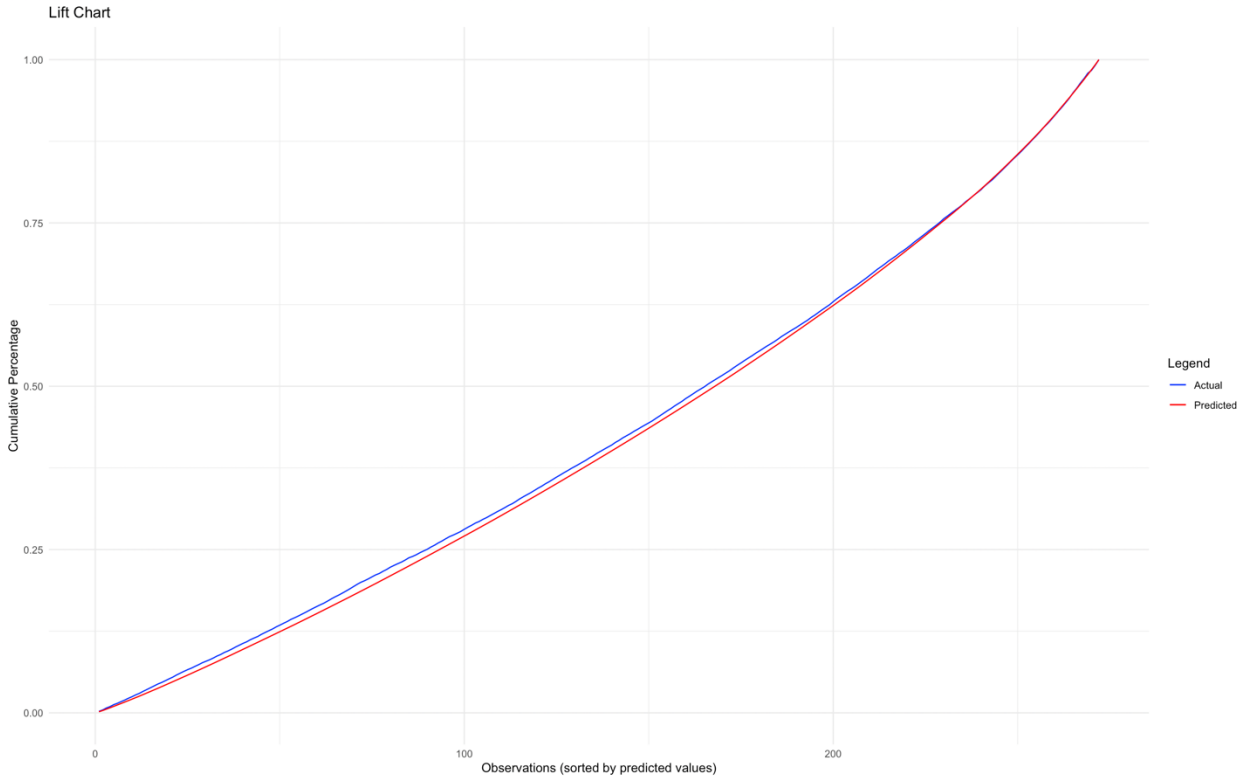

Figure 38. Lift Chart for Regression Model

6.2 Recommending the Use of Our Regression Model

Since all statistical performance measures and the lift chart produced favorable results in terms of the predictive performance of the multiple linear regression model, it demonstrates that the predicted price values are close to the actual price values. Therefore, we recommend Ebraham to use our regression model to evaluate second-hand cars based on their features and accessories. It is important to note that our multiple linear regression model outperformed the neural network models in predicting car prices, with a MAPE value of 8.59 compared to the approximately 23.1341 MAPE value observed in the neural network models. This indicates that the multiple linear regression model is a more accurate and reliable option for Ebraham's car price predictions.

G. CONCLUSION

In this case, we aimed to predict second-hand car prices using neural network and multiple linear regression models. Through a systematic process involving exploratory data analysis, data preprocessing, feature selection, and model evaluation, we found that the multiple linear regression model performed better in predicting car prices with a MAPE score of 8.59, compared to neural network models that had a MAPE score of around 23.1341. The results suggest that the linear regression model is a more reliable choice for predicting car prices based on currently available data. However, it is important to consider the limitations and potential improvements of the neural network models, including the need for more data and possible adjustments to the architectures, before making a final recommendation.

Before model evaluation, we initially assumed that the neural network models would outperform the multiple linear regression model, as we believed they would be more adept at capturing the underlying trends in the dataset. However, our analysis demonstrated that results aren't always as expected in data-driven projects. This experience showed the importance of experimenting with different models for the same task and selecting the best one after conducting a comprehensive performance evaluation. Furthermore, it highlighted the significance of understanding the limitations and potential improvements of each model, considering factors such as data size and model complexity, to make more informed decisions in data-driven projects.

For future work, it is advised to enhance the models by incorporating new data, preferably by using more extensive datasets that allow the models to capture the underlying patterns more effectively and produce more precise predictions. Furthermore, it is also recommended to experiment with various combinations of features while training the models and assess the outcomes to determine the features that are most significant in predicting car prices, with the goal of improving the models' predictive capabilities.

H. TABLE DESIGN PRINCIPLES AND GRAPHICAL EXCELLENCE

In our case report, we included 17 charts and 1 table, each designed to illustrate specific points clearly and effectively. To ensure proper delivery of these points, we adhered to the principles of graphical excellence by maximizing the information conveyed with minimal ink usage. Each graph and each table are accompanied by an introduction and a title to provide context. We utilized color in scatter charts and box plots to effectively highlight data dispersion, which is crucial for these types of visualizations. The table presented on pages 2 and 3 follows table design principles, featuring appropriate column headers and the absence of unnecessary text, making it easy to read and understand.

I. REFERENCES

Anand, Varin. "Introduction to Neural Networks." *Analytics Vidhya*, 15 Mar. 2022, https://www.analyticsvidhya.com/blog/2022/01/introduction-to-neural-networks/.

Gupta, Aman. "Feature Selection Techniques in Machine Learning (Updated 2023)." *Analytics Vidhya*, Analytics Vidhya, 26 Apr. 2023, https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/.

Rawat, Soumyaa. "Advantages and Disadvantages of Neural Networks." *Analytics Steps*, Analytics Steps Infomedia, 21 July 2022, https://www.analyticssteps.com/blogs/advantages-and-disadvantages-neural-networks.

J. APPENDIX

10.1 R-Script of Exploratory Data Analysis

```
# Load necessary libraries
library(tidyverse)
library(ggplot2)
library(gridExtra)

# Load the dataset
cars <- read.csv("/Users/selinceydeli/Desktop/sabancı/OPIM407/OPIM407 case2/Case2_W28593-XLS-ENG.csv", sep = ";")
cars <- cars[,1:28]

# Get the dimensions of the dataset (rows and columns)
dimensions <- dim(cars)
dimensions

# Number of observations (rows)
num_observations <- nrow(cars)
num_observations

# Number of variables (columns)
num_variables <- ncol(cars)
num_variables

# Find the minimum and maximum values of the price column
min_price <- min(cars$Price)
max_price <- max(cars$Price)

# Calculate the range
price_range <- max_price - min_price

# Print the results
cat("Minimum price:", min_price, "\n")
cat("Maximum price:", max_price, "\n")
cat("Price range:", price_range, "\n")

# View the structure of the dataset
str(cars)
View(cars)

# Summary statistics for numerical variables
summary(cars)

# Are there any null values in the dataset?
sapply(cars, function(x) sum(is.na(x))) #There aren't any NA values

# Displaying the first 10 observations
head(cars, 10)

# Convert categorical variables to factors
#cars$Fuel <- as.factor(cars$Fuel)
#cars$Color <- as.factor(cars$Color)

# Explore relationships between variables through correlation matrix
numerical_vars <- cars %>% select_if(is.numeric)
correlation_matrix <- cor(numerical_vars, use = "complete.obs")
print(correlation_matrix)

# Visualize correlation matrix
corrplot::corrplot(correlation_matrix, method = "color", type = "lower", tl.col = "black", tl.cex = 0.7)

# Boxplots to explore relationships between categorical variables and price
ggplot(cars, aes(x = Fuel, y = Price)) + geom_boxplot() + labs(title = "Price vs Fuel Type")
ggplot(cars, aes(x = Color, y = Price)) + geom_boxplot() + labs(title = "Price vs Color")

# Create the scatter plot for Price vs Age
plot1 <- ggplot(cars, aes(x = Age, y = Price)) +
  geom_point() +
```

```
  geom_smooth(method = "lm") +
  labs(title = "Price vs Age",
       x = "Age (months)",
       y = "Price") +
  theme_minimal()

# Create the scatter plot for Price vs Accumulated Kilometers
plot2 <- ggplot(cars, aes(x = KM, y = Price)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Price vs Accumulated Kilometers",
       x = "Kilometers",
       y = "Price") +
  theme_minimal()

# Combine the two scatter plots in a grid
grid.arrange(plot1, plot2, ncol = 2)

# Histogram for price distribution
ggplot(cars, aes(x = Price)) + geom_histogram(bins = 30, fill = "steelblue", color = "black") + labs(title = "Price Distribution")

# Combined visualization of Fuel, HP, CC, and Cyl with respect to Price
ggplot(cars, aes(x = HP, y = Price, color = CC, size = Cyl)) +
  geom_point(alpha = 0.6) +
  facet_wrap(~Fuel) +
  labs(title = "Price vs Technical Features",
       x = "Horse Power (HP)",
       y = "Price",
       color = "Cylinder Volume (CC)",
       size = "Number of Cylinders") +
  theme_minimal()

library(GGally)

# Prepare the dataset
cars_selected <- cars %>%
  select(Price, Age, HP, CC, Cyl)

# Normalize the dataset
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

cars_normalized <- as.data.frame(lapply(cars_selected, normalize))

# Create parallel coordinates plot
ggparcoord(cars_normalized,
           columns = 1:5,
           groupColumn = "Price",
           scale = "uniminmax",
           showPoints = TRUE,
           title = "Parallel Coordinates Plot of Car Features vs Price",
           alphaLines = 0.5) +
  scale_color_gradient(low = "blue", high = "red") +
  theme_minimal()

# Boxplots for categorical variables against Price
categorical_vars <- c("AC", "Radio", "M_Rim", "Tow_Bar")
for (var in categorical_vars) {
  plot <- ggplot(cars, aes_string(x = var, y = "Price")) +
    geom_boxplot() +
    labs(title = paste("Price vs", var),
         x = var,
         y = "Price")
  print(plot)
}

# Scatter plot for the continuous variable Wght against Price
ggplot(cars, aes(x = Wght, y = Price)) +
  geom_point() +
```

```
  geom_smooth(method = "lm") +
  labs(title = "Price vs Weight",
       x = "Weight (kg)",
       y = "Price")

# Create a boxplot to visualize the effect of metallic color on the price
plot_MC <- ggplot(cars, aes(x = as.factor(MC), y = Price)) +
  geom_boxplot() +
  labs(title = "Price vs Metallic Color",
       x = "Metallic Color (Yes = 1, No = 0)",
       y = "Price") +
  theme_minimal()

print(plot_MC)

# Group 1 Auto and Grs feature
ggplot(cars, aes(x = factor(Grs), y = Price, fill = factor(Auto))) +
  geom_boxplot() +
  labs(title = "Price vs Gear Positions and Automatic",
       x = "Number of Gear Positions",
       y = "Price",
       fill = "Automatic") +
  theme_minimal()

# Group 2 G_P and Mfr_G
ggplot(cars, aes(x = factor(Mfr_G), y = Price, fill = cut(G_P, breaks = 4))) +
  geom_boxplot() +
  labs(title = "Price vs Manufacturer's Guarantee and Guarantee Period",
       x = "Within Manufacturer's Guarantee",
       y = "Price",
       fill = "Guarantee Period") +
  theme_minimal()

# Group 3 number of doors
ggplot(cars, aes(x = factor(Drs), y = Price)) +
  geom_jitter(width = 0.3, alpha = 0.5) +
  labs(title = "Price vs Number of Doors",
       x = "Number of Doors",
       y = "Price") +
  theme_minimal()

# Revised Group 3 Plot about number of doors after updating the value 2 as 3 and 4 as 5:
library(dplyr)

# Update the door numbers in the dataset
cars <- cars %>%
  mutate(Drs = ifelse(Drs == 2, 3,
                      ifelse(Drs == 4, 5, Drs)))

View(cars)

ggplot(cars, aes(x = factor(Drs), y = Price)) +
  geom_jitter(width = 0.3, alpha = 0.5) +
  labs(title = "Price vs Number of Doors",
       x = "Number of Doors",
       y = "Price") +
  theme_minimal()

#install.packages("reshape2")
library("reshape2")

# Group 4 ABS, Abag_1, Abag_2, Comp, CD, Clock, Pwin, PStr
group4_vars <- c("ABS", "Abag_1", "Abag_2", "Comp", "CD", "Clock", "Pwin", "PStr")
group4_melted <- melt(cars, id.vars = "Price", measure.vars = group4_vars)

ggplot(group4_melted, aes(x = factor(value), y = Price, fill = variable)) +
  geom_boxplot() +
  facet_wrap(~variable, scales = "free", labeller = label_both) +
  labs(title = "Price vs Group 4 Features",
       x = "Feature Value",
```

```
      y = "Price",
      fill = "Feature") +
  theme_minimal()

# Group 5 SpM
ggplot(cars, aes(x = factor(SpM), y = Price)) +
  geom_boxplot() +
  labs(title = "Price vs Sport Model",
      x = "Sport Model",
      y = "Price") +
  theme_minimal()
```

10.2 R-Script of Feature Selection

```
#install.packages("tidyverse")
#install.packages("caret")
#install.packages("glmnet")
#install.packages("leaps")
library(leaps)
library(tidyverse)
library(caret)
library(glmnet)
library(dplyr)

# Approach 1 - Filter Method: Correlation-based Feature Selection

# Load the dataset
cars <- read.csv("/Users/selinceydeli/Desktop/sabancı/OPIM407/OPIM407 case2/Case2_W28593-XLS-ENG.csv", sep = ";")
cars <- cars[,1:28]

# Split the dataset into training and testing sets:
set.seed(123) # For reproducibility
trainIndex <- createDataPartition(cars$Price, p = 0.8, list = FALSE, times = 1)
train <- cars[trainIndex,]
test <- cars[-trainIndex,]

# Convert character variables to factors
train$Fuel <- as.factor(train$Fuel)
train$Color <- as.factor(train$Color)
View(cars)

# Implement the Correlation-based Feature Selection using the training dataset:
# Calculate the correlation between numeric features and the target variable
correlations <- cor(train[, sapply(train, is.numeric)])
correlations_with_target <- correlations["Price",]

# Sort the correlations in descending order
sorted_correlations <- sort(abs(correlations_with_target), decreasing = TRUE)

# Select the top k features with the highest correlation (e.g., k = 10)
k <- 10
top_k_features <- names(sorted_correlations)[2:(k+1)]  # Add 1 to exclude the 'Price' variable itself

cat("Top", k, "features based on correlation:\n", top_k_features, "\n")

# Approach 2 - Wrapper Method: Forward Feature Selection

# Create a formula for the linear regression model using all predictor variables except Price:
predictors <- names(train[, !(names(train) %in% "Price")])
formula <- as.formula(paste("Price ~", paste(predictors, collapse = " + ")))

# Apply forward feature selection using the training dataset:
fwd_selection <- regsubsets(formula, data = train, method = "forward", nvmax = length(predictors))
fwd_summary <- summary(fwd_selection)

# Determine the optimal number of features using the adjusted R-squared criterion:
optimal_n_features <- which.max(fwd_summary$adjr2)
cat("Optimal number of features:", optimal_n_features, "\n")
```

```
# Extract the selected features:
selected_features <- names(coef(fwd_selection, id = optimal_n_features))[-1] # Exclude intercept
cat("Selected features using forward feature selection:\n", selected_features, "\n")
length(selected_features)

# Approach 3 - Embedded Method: LASSO Regression

# Preprocessing
# Prepare the data for LASSO regression by converting the categorical variables to dummy variables:
train_dummies <- model.matrix(~ . - 1, data = train[, -1])

# Impute missing values
train_dummies_imputed <- apply(train_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))

# Check for missing values in the dataset
if (any(is.na(train_dummies_imputed))) {
  cat("There are still missing values in the dataset.\n")
} else {
  cat("No missing values found in the dataset.\n")
}

# Fit the LASSO regression model using cross-validation to find the optimal lambda value:
set.seed(123) # For reproducibility
lasso_cv <- cv.glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1)

# Find the optimal lambda value
optimal_lambda <- lasso_cv$lambda.min

# Fit the final LASSO model using the optimal lambda
lasso_model <- glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1, lambda = optimal_lambda)

# Extract selected features
selected_features <- rownames(coef(lasso_model))[which(coef(lasso_model) != 0)[-1]]

# Print the selected features
cat("Selected Features:\n")
print(selected_features)
```

10.3 R-Script of Neural Networks

```
# Load libraries
library(leaps)
library(tidyverse)
library(caret)
library(glmnet)
library(dplyr)
library(nnet)
library(neuralnet)
library(Metrics)

# Load the dataset
cars <- read.csv("/Users/selinceydeli/Desktop/sabancı/OPIM407/OPIM407 case2/Case2_W28593-XLS-ENG.csv", sep = ";")
cars <- cars[,1:28]

# Split the dataset into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(cars$Price, p = 0.8, list = FALSE, times = 1)
train <- cars[trainIndex,]
test <- cars[-trainIndex,]

# Preprocessing
train$Fuel <- as.factor(train$Fuel)
train$Color <- as.factor(train$Color)

# One-hot encoding
train_dummies <- model.matrix(~ . - 1, data = train[, -1])
test_dummies <- model.matrix(~ . - 1, data = test[, -1])
```

```
# Impute missing values
train_dummies_imputed <- apply(train_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))
test_dummies_imputed <- apply(test_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))

# Feature selection using LASSO
set.seed(123)
lasso_cv <- cv.glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1)
optimal_lambda <- lasso_cv$lambda.min
lasso_model <- glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1, lambda = optimal_lambda)
selected_features <- rownames(coef(lasso_model))[which(coef(lasso_model) != 0)[-1]]

cat("Selected Features:\n")
print(selected_features)

# Keep only the selected features
train_selected <- train_dummies_imputed[, selected_features]
test_selected <- test_dummies_imputed[, selected_features]

# Normalize the values
preprocess_params <- preProcess(train_selected, method = c("center", "scale"))
train_selected_norm <- predict(preprocess_params, train_selected)
test_selected_norm <- predict(preprocess_params, test_selected)

# Architecture 1: One hidden layer with 10 neurons
set.seed(123)
nn_formula1 <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
nn1 <- neuralnet(nn_formula1, data = data.frame(Price=train$Price, train_selected_norm), hidden = 10, linear.output = TRUE)
pred_nn1 <- compute(nn1, test_selected_norm)$net.result

# Architecture 2: One hidden layer with 15 neurons
set.seed(123)
nn_formula2 <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
nn2 <- neuralnet(nn_formula2, data = data.frame(Price=train$Price, train_selected_norm), hidden = 15, linear.output = TRUE, stepmax =
1e+055)
pred_nn2 <- compute(nn2, test_selected_norm)$net.result

# Architecture 3: Two hidden layers with 30 and 20 neurons
set.seed(123)
nn_formula3 <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))
nn3 <- neuralnet(nn_formula3, data = data.frame(Price=train$Price, train_selected_norm), hidden = c(30, 20), linear.output = TRUE)
pred_nn3 <- compute(nn3, test_selected_norm)$net.result

# Calculate MSE, RMSE, and MAPE for each architecture
mse_nn1 <- mean((test$Price - pred_nn1)^2)
rmse_nn1 <- sqrt(mse_nn1)
mape_nn1 <- mean(abs((test$Price - pred_nn1) / test$Price)) * 100

cat("MSE, RMSE, and MAPE for Architecture 1 (nn1):\n")
cat(paste("MSE:", mse_nn1, "RMSE:", rmse_nn1, "MAPE:", mape_nn1, "\n"))

mse_nn2 <- mean((test$Price - pred_nn2)^2)
rmse_nn2 <- sqrt(mse_nn2)
mape_nn2 <- mean(abs((test$Price - pred_nn2) / test$Price)) * 100

cat("MSE, RMSE, and MAPE for Architecture 2 (nn2):\n")
cat(paste("MSE:", mse_nn2, "RMSE:", rmse_nn2, "MAPE:", mape_nn2, "\n"))

mse_nn3 <- mean((test$Price - pred_nn3)^2)
rmse_nn3 <- sqrt(mse_nn3)
mape_nn3 <- mean(abs((test$Price - pred_nn3) / test$Price)) * 100

cat("MSE, RMSE, and MAPE for Architecture 3 (nn3):\n")
cat(paste("MSE:", mse_nn3, "RMSE:", rmse_nn3, "MAPE:", mape_nn3, "\n"))

# Visualize the neural network architectures
plot(nn1, rep = "best")
title("Architecture 1")

plot(nn2, rep = "best")
```

```
title("Architecture 2")

plot(nn3, rep = "best")
title("Architecture 3")

# Plotting lift charts
library(ggplot2)

# Create a function to plot lift charts
plot_lift_chart <- function(actual, predicted, title) {
  df <- data.frame(actual, predicted)
  df <- df[order(df$actual, decreasing = TRUE),]
  df$group <- cut(seq_along(df$actual), breaks = 10, labels = FALSE)
  df_summary <- df %>%
    group_by(group) %>%
    summarise(actual_mean = mean(actual), predicted_mean = mean(predicted))

  ggplot(df_summary, aes(x = group)) +
    geom_line(aes(y = actual_mean, color = "Actual")) +
    geom_line(aes(y = predicted_mean, color = "Predicted")) +
    scale_color_manual(values = c("Actual" = "blue", "Predicted" = "red")) +
    labs(x = "Decile", y = "Mean Price", title = title, color = "Legend") +
    theme_minimal()
}

# Lift charts for each architecture
lift_chart_nn1 <- plot_lift_chart(test$Price, pred_nn1, "Lift Chart for Architecture 1")
lift_chart_nn2 <- plot_lift_chart(test$Price, pred_nn2, "Lift Chart for Architecture 2")
lift_chart_nn3 <- plot_lift_chart(test$Price, pred_nn3, "Lift Chart for Architecture 3")

# Print lift charts
lift_chart_nn1
lift_chart_nn2
lift_chart_nn3

# Displaying the lift charts in the same grid
library(gridExtra)
grid.arrange(lift_chart_nn1, lift_chart_nn2, lift_chart_nn3, nrow = 3)
```

10.4 R-Script of Multiple Linear Regression

```
 # Load required packages
 #install.packages("tidyverse")
 #install.packages("caret")
 #install.packages("car")
 #install.packages("glmnet")
 #install.packages("Metrics")
 library(Metrics)
 library(glmnet)
 library(tidyverse)
 library(caret)
 library(car)
 library(ggplot2)

 # Load the dataset
 cars <- read.csv("/Users/selinceydeli/Desktop/sabancı/OPIM407/OPIM407 case2/Case2_W28593-XLS-ENG.csv", sep = ";")
 cars <- cars[,1:28]

 # Feature Selection

 # Split the dataset into training and testing sets:
 set.seed(123) # For reproducibility
 trainIndex <- createDataPartition(cars$Price, p = 0.8, list = FALSE, times = 1)
 train <- cars[trainIndex,]
 test <- cars[-trainIndex,]

 # Preprocessing
 # Convert character variables into factors
```

```
train$Fuel <- as.factor(train$Fuel)
train$Color <- as.factor(train$Color)

# One-hot encoding
train_dummies <- model.matrix(~ . - 1, data = train[, -1])

# Impute missing values
train_dummies_imputed <- apply(train_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))

# Check for missing values in the dataset
if (any(is.na(train_dummies_imputed))) {
  cat("There are still missing values in the dataset.\n")
} else {
  cat("No missing values found in the dataset.\n")
}

# Run LASSO regression
set.seed(123) # For reproducibility
lasso_cv <- cv.glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1)

# Find the optimal lambda value
optimal_lambda <- lasso_cv$lambda.min

# Fit the final LASSO model using the optimal lambda
lasso_model <- glmnet(train_dummies_imputed, train$Price, family = "gaussian", alpha = 1, lambda = optimal_lambda)

# Extract selected features
selected_features <- rownames(coef(lasso_model))[which(coef(lasso_model) != 0)[-1]]

# Print the selected features
cat("Selected Features:\n")
print(selected_features)

# Multiple Linear Regression Model

# Create dummy variables for the 'Color' factor variable in the train dataset
color_dummies_train <- model.matrix(~ Color - 1, data = train)

# Convert the matrix to a data frame
color_dummies_train_df <- as.data.frame(color_dummies_train)

# Merge the dummy variables with the train dataset
train <- cbind(train, color_dummies_train_df)

# Create dummy variables for the 'Color' factor variable in the test dataset
color_dummies_test <- model.matrix(~ Color - 1, data = test)

# Convert the matrix to a data frame
color_dummies_test_df <- as.data.frame(color_dummies_test)

# Merge the dummy variables with the test dataset
test <- cbind(test, color_dummies_test_df)

# Create a formula using the selected features from the LASSO regression model
lasso_selected_formula <- as.formula(paste("Price ~", paste(selected_features, collapse = " + ")))

# Fit the multiple linear regression model using the selected features
lm_lasso_selected <- lm(lasso_selected_formula, data = train)

# Summarize the model
summary(lm_lasso_selected)

# Make predictions
predictions <- predict(lm_lasso_selected, newdata = test)

# Evaluate the model
mse <- mse(test$Price, predictions)
rmse <- sqrt(mse)
abs_percentage_errors <- abs((test$Price - predictions) / test$Price) * 100
MAPE <- mean(abs_percentage_errors)
```

```
cat("Mean Squared Error (MSE):", mse, "\n")
cat("Root Mean Squared Error (RMSE):", rmse, "\n")
cat("Mean Absolute Percentage Error (MAPE):", MAPE, "\n")

# Drawing lift charts

# Predicting prices on the test dataset
predictions <- predict(lm_lasso_selected, newdata = test)

# Create a data frame containing the actual values and the predicted values
lift_df <- data.frame(Actual = test$Price, Predicted = predictions)

# Sort the data frame by the predicted values
lift_df <- lift_df[order(lift_df$Predicted),]

# Calculate the cumulative actual and predicted values
lift_df$Cum_Actual <- cumsum(lift_df$Actual)
lift_df$Cum_Predicted <- cumsum(lift_df$Predicted)

# Calculate the percentage of the total for the cumulative actual and predicted values
lift_df$Pct_Actual <- lift_df$Cum_Actual / sum(lift_df$Actual)
lift_df$Pct_Predicted <- lift_df$Cum_Predicted / sum(lift_df$Predicted)

# Create an index column for the x-axis
lift_df$Index <- 1:nrow(lift_df)

# Plot the lift chart using ggplot2
ggplot(lift_df, aes(x = Index)) +
  geom_line(aes(y = Pct_Actual, color = "Actual")) +
  geom_line(aes(y = Pct_Predicted, color = "Predicted")) +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "red")) +
  labs(title = "Lift Chart",
       x = "Observations (sorted by predicted values)",
       y = "Cumulative Percentage",
       color = "Legend") +
  theme_minimal()
```