Vietnamese Car Prices

Buu Dinh Ha

2025-03-31

1. Problem Understanding

Vietnamese automotive market is one of the South East Asia's fastest growing automotive markets. We want to analyze this market using the dataset `car_detail_en.csv`. It contains information about various vehicles, which can be mined to get insights about the market.

The primary objectives of this data mining project are: First, find which car features most influence the market price. Second, develop predictive models to estimate car prices based on these features. Third, segment the market into clusters to get more insights.

2. Data Extraction and Exploration

2.1. Data Extraction

The dataset is taken from Kaggle. The author is Thanh Luan Nguyen and he made the dataset by web crawling on the car sale website https://bonbanh.com. The dataset has CSV format.

The dataset has 30652 observations and 21 variables. The statistical unit of interest for our analysis is the single car sale. Each observation is a car listed for sale, presenting these attributes:

- `ad_id`, `url`: Unique ad ID and the link to the listing (not useful for modeling)
- `price`: Car price in VND (text format, e.g., "4 Billion 200 Million")
- `brand`, `grade`, `car_name`: Brand (e.g., Toyota), model/grade, and full car name
- `car_model`, `engine`, `transmission`, `drive_type`: Basic vehicle specs
- `year_of_manufacture`: year the car was manufactured
- `mileage`, `num_of_doors`, `seating_capacity`: Used for assessing wear and practicality
- `exterior_color`, `interior_color`: Color info
- `condition`: Either "New car" or "Used car"
- `fuel_system`, `fuel_consumption`: Fuel info
- `describe`: Description of the car

Three features `mileage`, `num_of_doors` and `seating_capacity` have values following this structure: a number + unit. We simplify this by cutting the unit part.

```
df$mileage <- as.numeric(gsub("[^0-9]", "", df$mileage))
df$num_of_doors <- as.numeric(gsub("[^0-9]", "", df$num_of_doors))
df$seating_capacity <- as.numeric(gsub("[^0-9]", "", df$seating_capacity))
print(head(df$num_of_doors))
```

```
## [1] 2 5 5 5 5 5
```

Let look at our label `price`. The values follow this structure: a number + Billion/ billion + a number + Million/ million. The data values are somewhat inconsistent with white space, uppercase, lowercase, etc. We handle all cases to simplify this by cutting the Billion/ Million part. We save it into a new column `price_million_vnd`.

```
print(head(df$price_million_vnd))
```

```
## [1]  249 4286  885  754  850  299
```

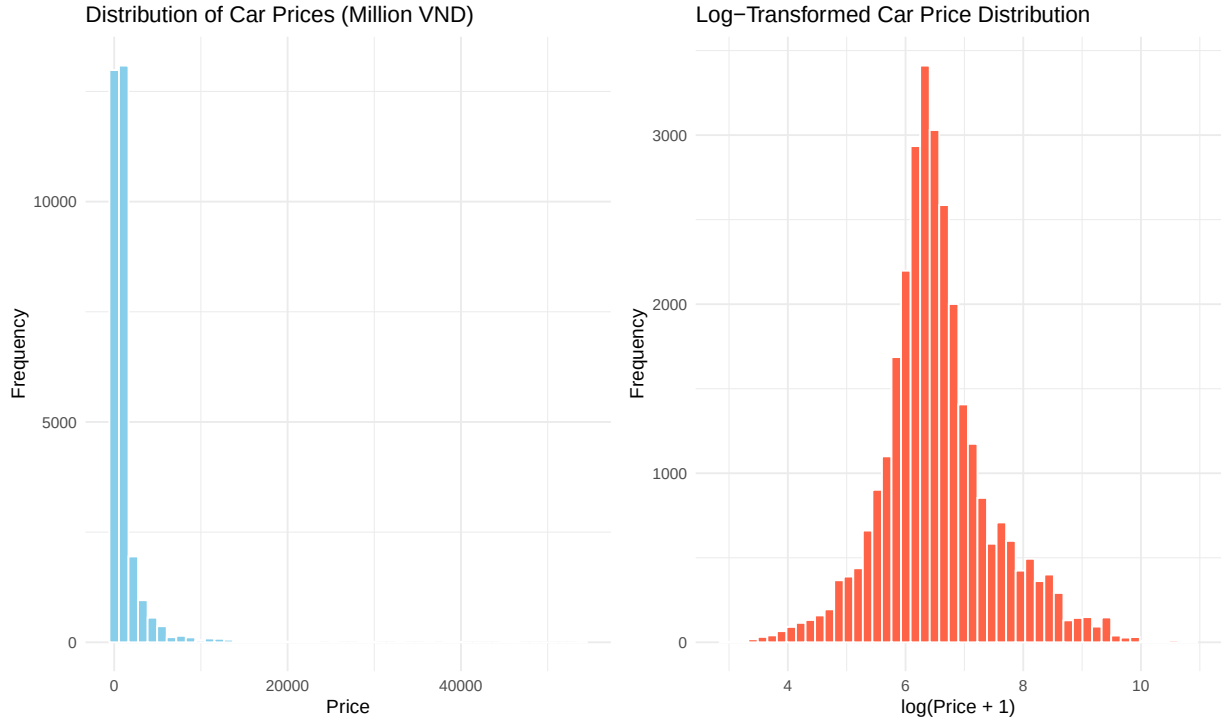
Features like `ad_id`, `describe` and `url` do not have an impact on our model. `fuel_system` has too many missing value. Values in `fuel_consumption` are incorrect. That's why we drop all of them.

```
df <- df[, !names(df) %in% c("ad_id", "fuel_system", "fuel_consumption", "url", "price", "describe")]
```

2.2. Data Exploration

2.2.1. Numerical features

First, we plot the distribution of car price in out dataset. We can see the problem is that our plot is extremely right skewed, since the mojority of car prices close to 0, while there is a long tail stretches far to the right. Most cars have price under 5 billion VND, whileas very few cars have prices above 10 billion VND. This makes it hard to interpret the shape of the data for most common cars. The solution is that we apply a log transform to the price. Now our plot is much more symmetric distribution, almost bell-shaped. This helps normalize the distribution, which will benefit us for applying machine learning methods later.



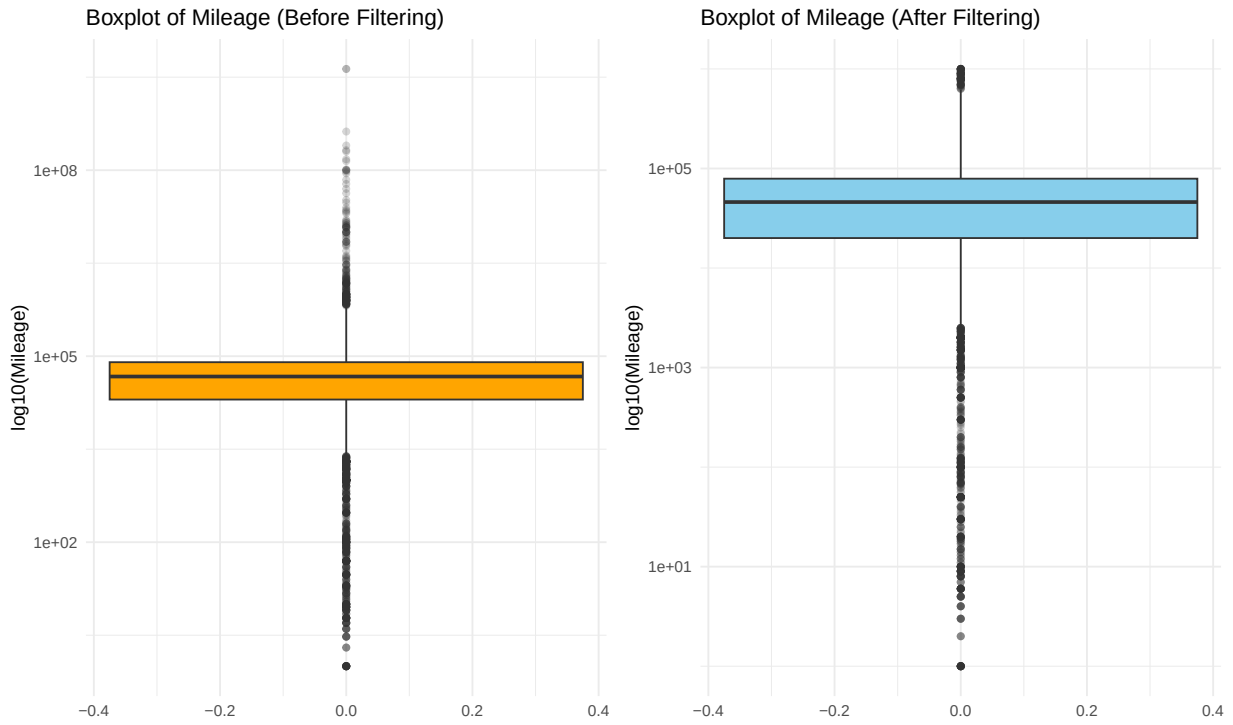
Let look at the summary of the numeric values: `mileage`, `number_of_doors`, `seating_capacity` and `year_of_manufacture`. Median mileage is 20,000 km, which are reasonable for used cars. The mean is

421,300 km, which is extremely high. Max value is 4.3 billion km, which is clearly unrealistic, thus, is an error. Let plot the boxplot of log scaled mileage values.

```
summary(df[, c("mileage", "num_of_doors", "seating_capacity", "year_of_manufacture")])
```

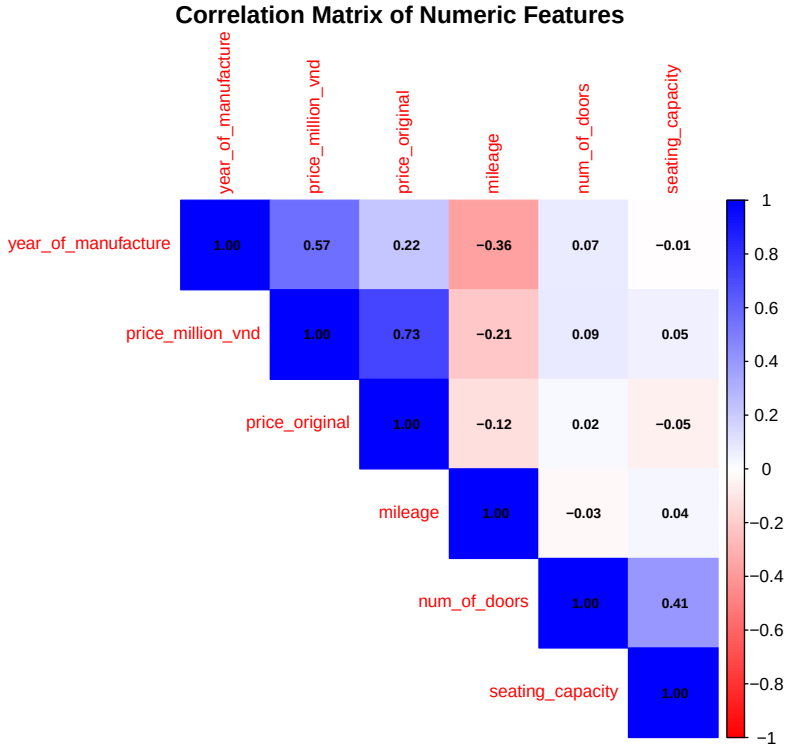
```
##     mileage           num_of_doors    seating_capacity year_of_manufacture
##  Min.   :0.000e+00   Min.   : 0.000   Min.   : 0.00    Min.   :1990
##  1st Qu.:0.000e+00   1st Qu.: 4.000   1st Qu.: 5.00    1st Qu.:2015
##  Median :2.000e+04   Median : 5.000   Median : 5.00    Median :2019
##  Mean   :4.123e+05   Mean   : 4.503   Mean   : 5.54    Mean   :2017
##  3rd Qu.:6.000e+04   3rd Qu.: 5.000   3rd Qu.: 7.00    3rd Qu.:2022
##  Max.   :4.295e+09   Max.   :54.000   Max.   :47.00    Max.   :2023
##                                                        NA's   :32
```

The bulk of mileages seem to fall between ~10,000 km to ~100,000 km (1e+4 to 1e+5), which is expected for used cars. There are lots of extreme outliers, especially above 100,000,000 km (1e+8). We will trim mileage at a realistic max: 1,000,000 km.



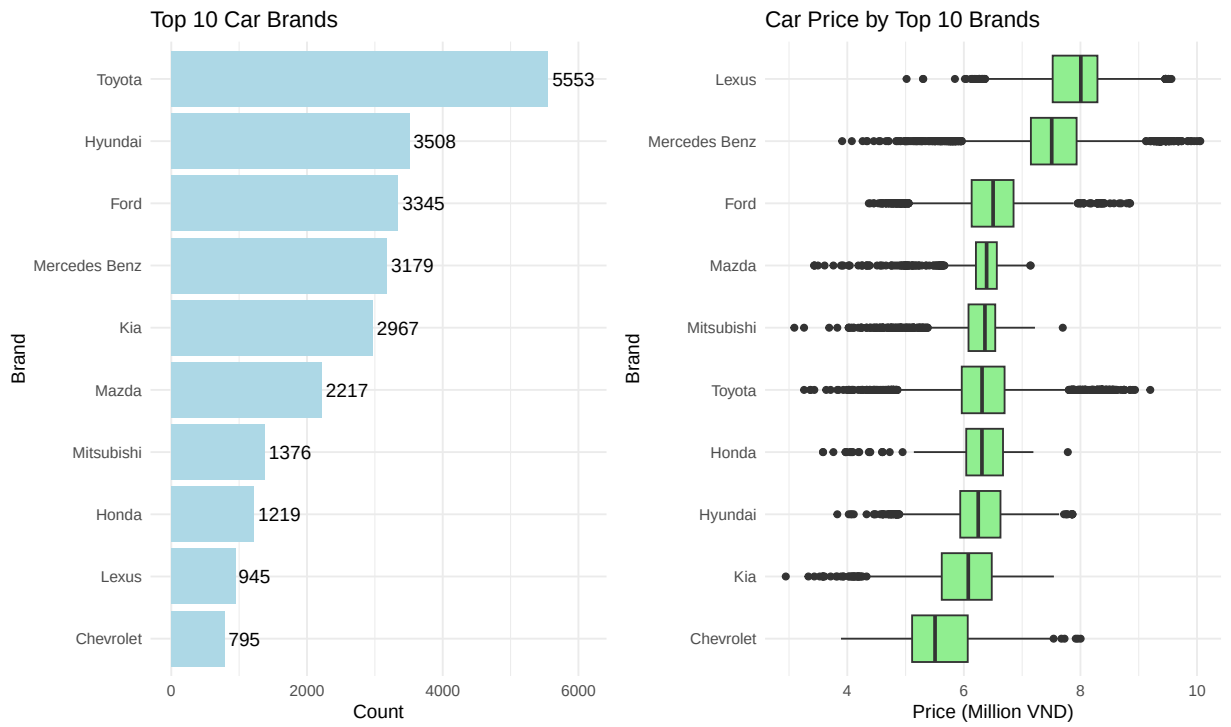
We do the same for other numeric features: trim at a realistic range number of doors (from 2 to 10) and seating capacity (2 to 20).

Let look at the correlation matrix of all numerical features. We can see that there are no strong correlations with price (all of them are weak < 0.3). Mileage has a small negative correlation with price (-0.12), suggesting older cars sell for slightly less. Mileage also have the most negative correlation with year of manufacture (-0.36), indicating order cars are used more than newer cars. Year of manufacture has a stronger positive correlation with price (+0.22), suggesting newer cars tend to be more expensive, but it's a moderate correlation. Lastly, number of doors and seating capacity have a highest positive correlation (+0.41), indicating more doors often mean more seats.



2.2.2. Categorical features

The figure below show the most popular car brands in the dataset and the distribution of their prices (log-transformed). We see that luxury brands like Lexus and Mercedes Benz have much higher median prices. Mercedes Benz and Toyota have a wide range of prices, from affordable models to higher-end ones. Mazda, Honda, Hyundai and Kia tend to be in the lower to mid-price segment. We can see that brand clearly has a strong relationship with the price. This means that brand should be treated as a key categorical feature when modeling car prices later on.



We do the same for other categorical features: `car_model`, `engine`, `drive_type`.

3. Data Preparation

First we check if there is any NA values and omit it.

```
df_filtered <- na.omit(df_filtered)
colSums(is.na(df_filtered))
```

```
##              origin           condition           car_model             mileage
##                   0                   0                   0                   0
##      exterior_color      interior_color       num_of_doors    seating_capacity
##                   0                   0                   0                   0
##              engine        transmission          drive_type               brand
##                   0                   0                   0                   0
##               grade year_of_manufacture            car_name     price_million_vnd
##                   0                   0                   0                   0
##      price_original
##                   0
```

3.1. For Association Rule Mining

Now we can begin to prepare data for doing Association Rule Mining later. First, we do binning continuous variables. We convert numerical features into categorical bins for use in ARM. All numerical features are broken into 3 levels: Low, Medium and High.

```r
df_filtered_cat <- df_filtered
df_filtered_cat$price_cat <- cut(df_filtered$price_million_vnd,
                          breaks = quantile(df_filtered$price_million_vnd, probs = c(0, 0.33, 0.66, 1),
                          labels = c("Low", "Medium", "High"),
                          include.lowest = TRUE)

df_filtered_cat$mileage_cat <- cut(df_filtered$mileage,
                            breaks = 3, labels = c("Low", "Medium", "High"))

df_filtered_cat$year_of_manufacture_cat <- cut(df_filtered$year_of_manufacture,
                         breaks = 3, labels = c("Old", "Mid", "New"))

df_filtered_cat$num_of_doors_cat <- cut(df_filtered$num_of_doors,
                         breaks = 3, labels = c("Low", "Medium", "High"))

df_filtered_cat$seating_capacity_cat <- cut(df_filtered$seating_capacity,
                         breaks = 3, labels = c("Low", "Medium", "High"))
```

We create a new dataframe `df_filtered_cat` which keeps only categorical columns and convert them all to factors (required by `arules` for `transactions`)

```r
df_filtered_cat <- df_filtered_cat %>%
  select(origin, condition, car_model, exterior_color, interior_color, engine, transmission, drive_type

df_filtered_cat[] <- lapply(df_filtered_cat, as.factor)
```

Next step is to convert the cleaned dataframe into a transaction object, where each row is treated like a shopping basket of features (e.g., `{brand=Toyota, transmission=Automatic}`). With the transaction, we are ready to implement Apriori.

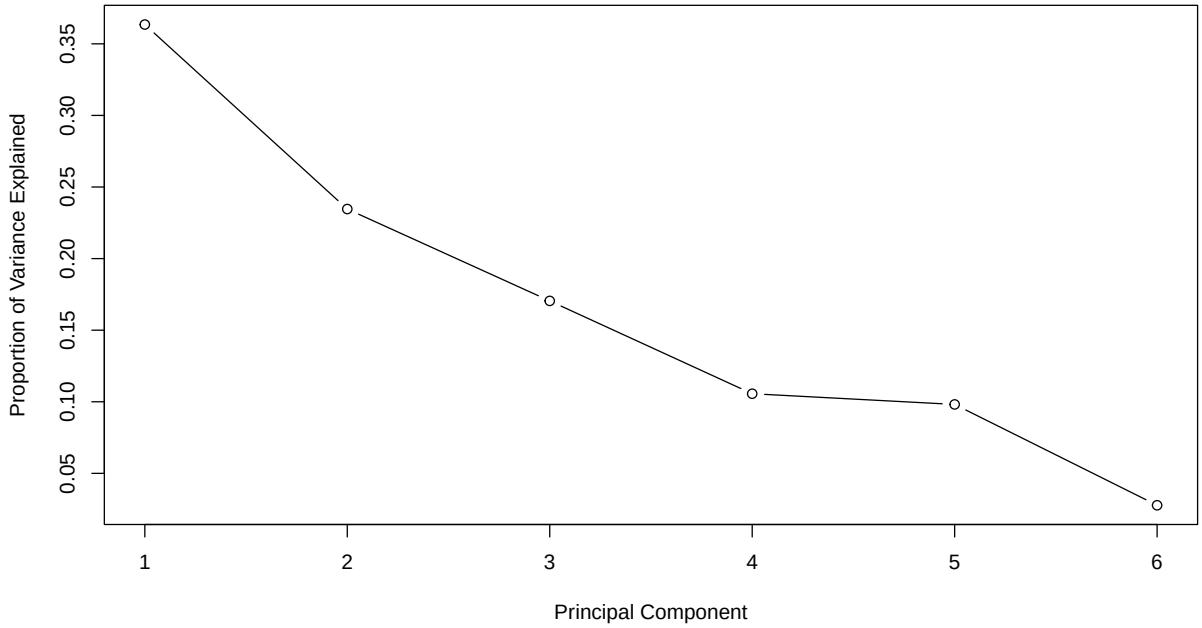
```r
car_trans <- as(df_filtered_cat, "transactions")
```

3.2. For Clustering (PCA)

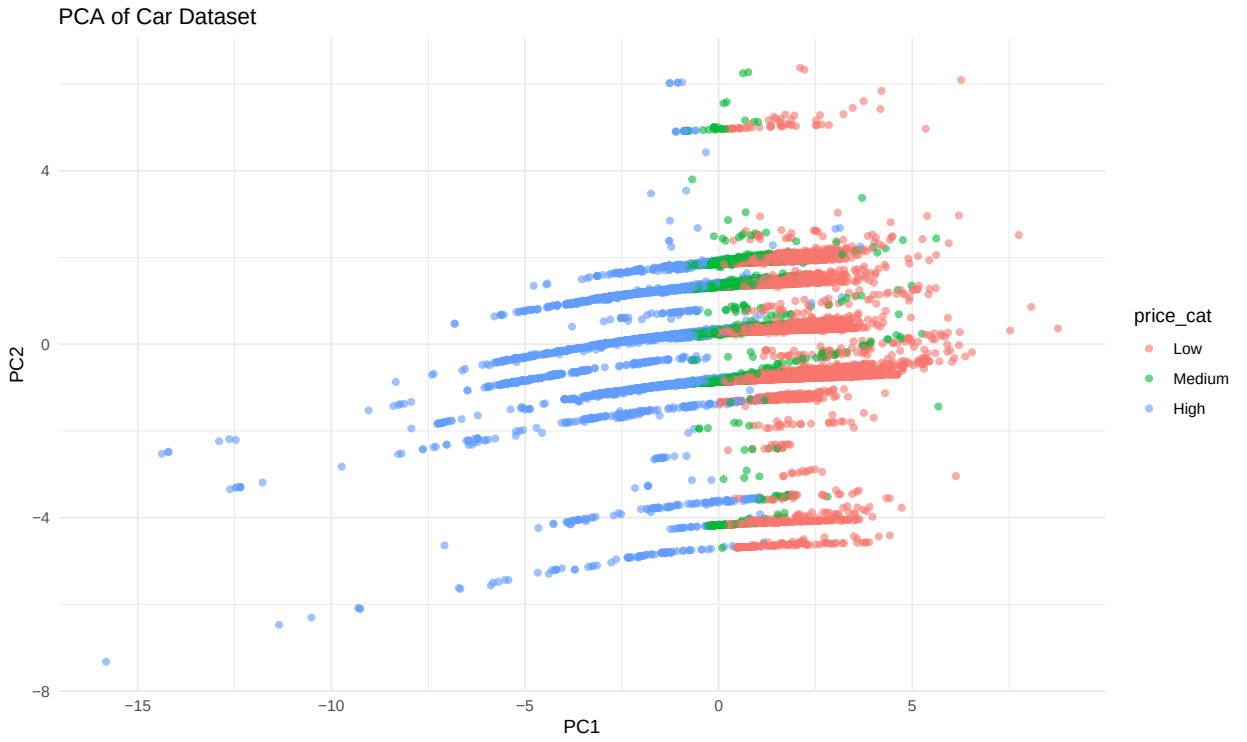
We do PCA to discover patterns and prepare for clustering. We perform PCA on a subset of our dataset containing numeric features (including price). The scree plot tells us that PC1 explains ~36%, PC2 ~27%, and PC3 ~16% of the variance. The first two PCs together explain ~63% of the total variance. After PC3, the contribution drops off, thus most of our data's stucture can be seen in PC1-PC3.

```r
df_numeric <- df_filtered %>%
  select_if(is.numeric)
df_numeric_scaled <- scale(df_numeric)
pca_result <- prcomp(df_numeric_scaled, center = TRUE, scale. = TRUE)

df_pca <- as.data.frame(pca_result$x[, 1:3])
df_pca$price_cat <- df_filtered_cat$price_cat

colnames(df_pca)[1:3] <- c("PC1", "PC2", "PC3")

variance <- pca_result$sdev^2
prop_variance <- variance / sum(variance)
```

```r
plot(prop_variance, type = "b", xlab = "Principal Component",
     ylab = "Proportion of Variance Explained")
```



The scatter plot shows our data projected into PC1 and PC2, colored by `price_cat` (Low, Medium, High). We can see that the low price cars (red) form distinct bands, mostly clustered toward the middle and right. The high price cars (blue) seem more concentrated on the left-hand side and the medium price cars (green) are scattered in between, overlapping both groups. Now we are ready for doing clustering.



3.3. For Regression

Now we prepare data for linear regression. Start with the filtered dataframe, we drop `car_name` since it is too specific for a general linear model. We do consistent cleaning by lowercasing, trimming and space normalization. Next, we apply the log for the price.

```
df_reg_base <- df_filtered
df_reg <- df_reg_base %>% select(-car_name, -price_million_vnd)
df_reg <- df_reg %>%
  mutate(across(where(is.character), ~ tolower(trimws(gsub("[\t ]+", " ", .)))))
df_reg <- df_reg %>%
  mutate(log_price = log1p(price_original))
```

Next step is very important, since we want to train our data on the train data and predict it on the test data, this raise an issue is that our test data contains specific values (called "levels") within a categorical column (a "factor") that the model did not encounter when it was being trained on train data. The key idea is to ensure that both the training and testing datasets share the exact same set of possible levels for each factor before the model is even trained. We achieve this by identifying levels that occur infrequently in the entire dataset (`df_reg` before splitting) and grouping them into a single, new category, often called "Other". This way, even if a rare level originally existed only in what would have become the test set, it gets recategorized as "Other" before the split. The model then learns a coefficient for this "Other" category based on the instances it sees in the training data. When it encounters an "Other" instance in the test data, it knows how to handle it.

```
factor_cols <- names(df_reg)[sapply(df_reg, is.character)]

n_engine = 20
n_grade = 50
```

```r
n_brand = 25
n_car_model = 10
n_color = 10

df_reg <- df_reg %>%
  mutate(
    engine = fct_lump_n(factor(engine), n = n_engine, other_level = "Other_Engine"),
    grade = fct_lump_n(factor(grade), n = n_grade, other_level = "Other_Grade"),
    brand = fct_lump_n(factor(brand), n = n_brand, other_level = "Other_Brand"),
    car_model = fct_lump_n(factor(car_model), n = n_car_model, other_level = "Other_CarModel"),
    exterior_color = fct_lump_n(factor(exterior_color), n = n_color, other_level = "Other_ExtColor"),
    interior_color = fct_lump_n(factor(interior_color), n = n_color, other_level = "Other_IntColor"),

    origin = factor(origin),
    condition = factor(condition),
    transmission = factor(transmission),
    drive_type = factor(drive_type)
  )
```

Then we split data into training and testing set.

```r
set.seed(123)
sample_index <- sample(1:nrow(df_reg), 0.8 * nrow(df_reg))

df_train <- df_reg[sample_index, ]
df_test <- df_reg[-sample_index, ]
```

4. Modeling

4.1. Association Rule Mining (Apriori)

Let run Apriori Algorithm to find association rules with `support >= 1%`, `confidence >= 60%` and minimum rule length = 2. We also filter out all the redundant rules. Let look at the top 5 rules with the highest `lift`. The output is following a structure: When conditions are met, the car is almost or always of a specific grade/model. For example, rule 2 indicates that every domestically assembled Honda sedan with a 1.5L petrol engine is a City. It is x81 more likely the car is a City given these conditions versus random chance. There are 782 cars following this rule. The probability that the car is a City given these conditions is 100%.

```r
rules <- apriori(car_trans, parameter = list(supp = 0.01, conf = 0.6, minlen = 2))
```

```r
rules_unique <- rules[!is.redundant(rules)]
inspect(sort(rules_unique, by = "lift")[1:5])
```

```
##     lhs                                    rhs             support confidence   coverage     lift c
## [1] {car_model=SUV,
##      brand=Honda}                       => {grade=CRV}  0.01227337  1.0000000 0.01227337 81.04267
## [2] {origin=Domestic assembly,
##      car_model=Sedan,
##      engine=Petrol   1.5 L,
##      brand=Honda}                       => {grade=City} 0.01168109  1.0000000 0.01168109 80.82713
## [3] {origin=Domestic assembly,
```

```
##      car_model=Sedan,
##      transmission=Automatic,
##      drive_type=FWD - Front-wheel drive,
##      brand=Honda,
##      year_of_manufacture_cat=New}         => {grade=City} 0.01207594  0.9683377 0.01247080 78.26796
## [4] {origin=Domestic assembly,
##      car_model=Sedan,
##      drive_type=FWD - Front-wheel drive,
##      brand=Honda,
##      year_of_manufacture_cat=New}         => {grade=City} 0.01224047  0.9662338 0.01266822 78.09790
## [5] {origin=Domestic assembly,
##      car_model=Sedan,
##      transmission=Automatic,
##      brand=Honda,
##      year_of_manufacture_cat=New}         => {grade=City} 0.01217466  0.9660574 0.01260242 78.08365
```

We subset the rules to find those are related to the car prices. We got these following interesting insights: All GLCs, new Lexus RXs, new Mercedes S-Class, AWD SantaFe are always high-priced. On the other hand, mid-age Daewoos, FWD Daewoos, used Daewoos, small hatchbacks with 1.0L engines are always low-priced.

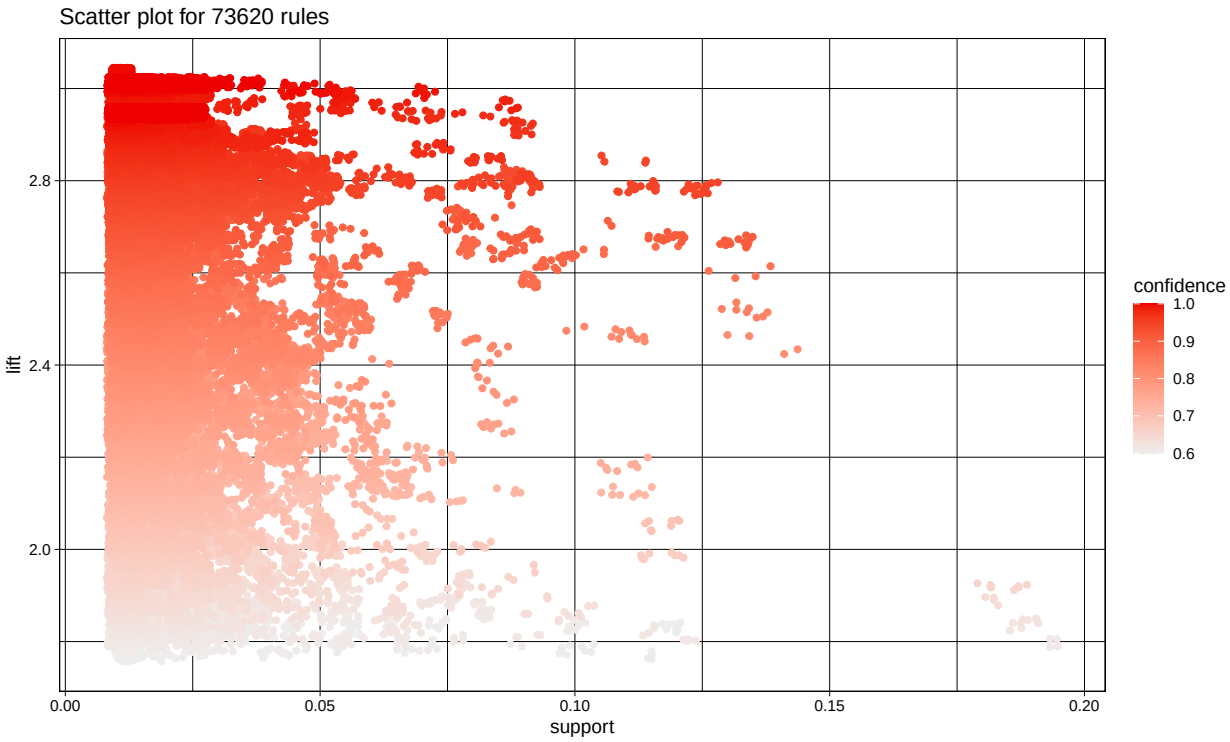
```
rules_price <- subset(rules, rhs %pin% "price_cat")
inspect(sort(rules_price, by = "confidence")[1:5])
```

```
##     lhs                                     rhs                 support confidence   coverage     li
## [1] {grade=GLC}                          => {price_cat=High} 0.02573130          1 0.02573130 2.947
## [2] {grade=RX,
##      year_of_manufacture_cat=New}        => {price_cat=High} 0.01059524          1 0.01059524 2.947
## [3] {brand=Daewoo,
##      year_of_manufacture_cat=Mid}        => {price_cat=Low}  0.01237208          1 0.01237208 3.012
## [4] {drive_type=FWD - Front-wheel drive,
##      brand=Daewoo}                       => {price_cat=Low}  0.01335922          1 0.01335922 3.012
## [5] {condition=Used car,
##      brand=Daewoo}                       => {price_cat=Low}  0.01342503          1 0.01342503 3.012
```

Let visualize 73,620 rules related to car price using a scatter plot. We can see that there are ton of rules with confidence = 1 (top of the color scale). Most rules have lift between 2 and 3, which is solid (2-3 times better than random chance). Very few rules conver more than 10% of the dataset. This is normal since association rule mining often reveals niche but strong patterns.

```
plot(rules_price, method = "scatterplot", measure = c("support", "lift"), shading = "confidence")
```

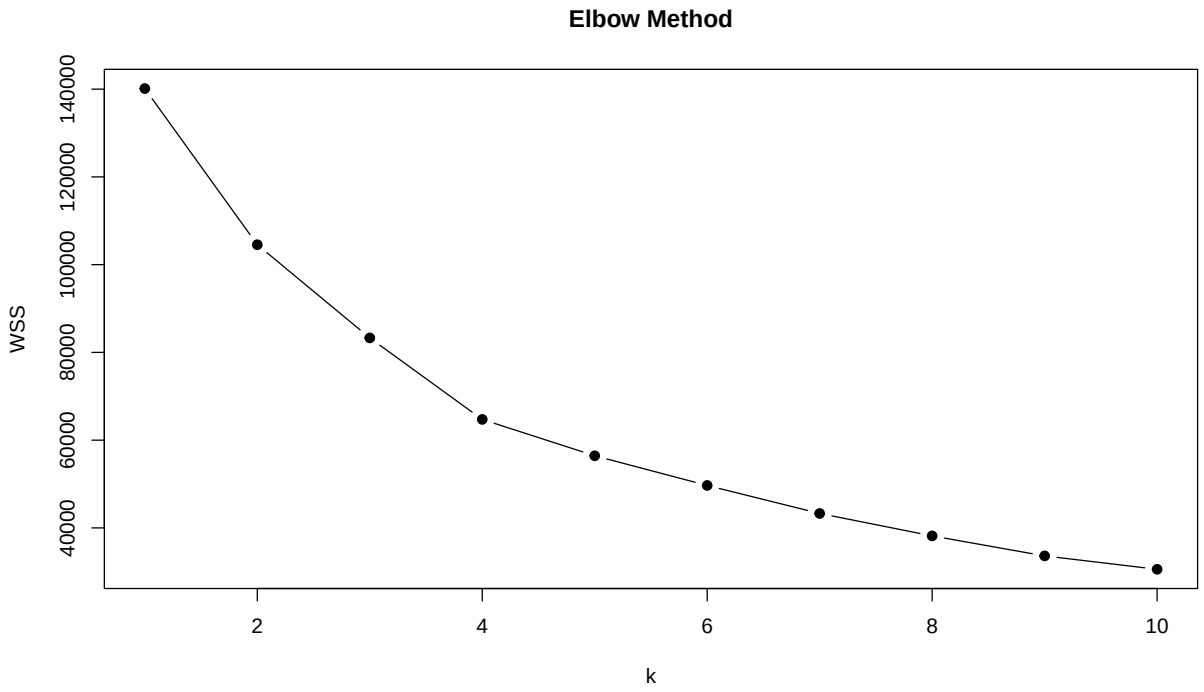
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



4.2. Clustering (K-means)

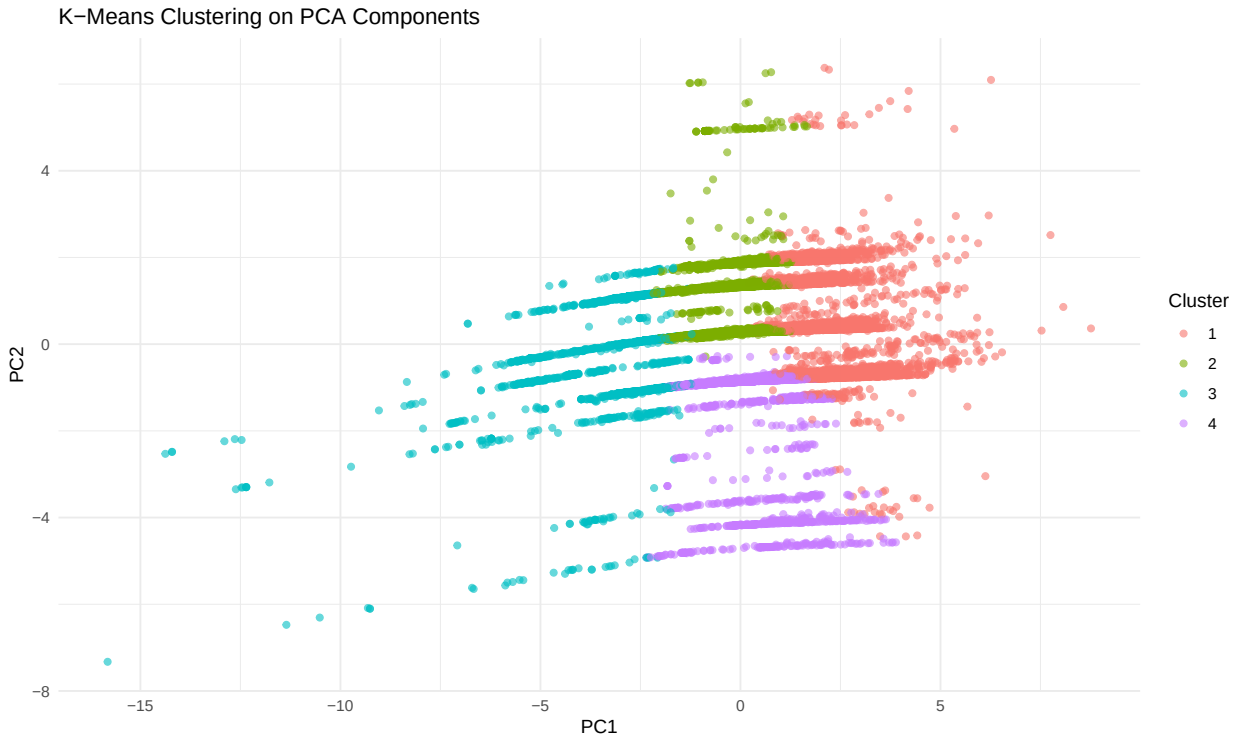
Before doing k-means clustering, we implement the Elbow Method for choosing the optimal number of cluster (k). The X-axis is the number of cluster k, Y-axis is the total within-cluster sum of squares (WSS) - we prefer the lowest as possible. We see that WSS drops sharply from k = 1 to k = 4. After k = 4, the decrease becomes more gradual and the elbow is most visible at around k = 4. Therefore, we choose k = 4 for clustering.

```
pca_features <- pca_result$x[, 1:3]

wss <- numeric(10)
for (k in 1:10) {
  wss[k] <- sum(kmeans(pca_features, centers = k, nstart = 10)$withinss)
}
plot(1:10, wss, type = "b", pch = 19,
     xlab = "k", ylab = "WSS",
     main = "Elbow Method")
```




```r
set.seed(123)
kmeans_result <- kmeans(pca_features, centers = 4, nstart = 25)
df_pca$cluster <- as.factor(kmeans_result$cluster)
```

The output of k-means are shown below. To analyze better, let look at the table comparing cluster vs price categories. This shows how well the new clusters reflect price patterns. For cluster 1, most of the high-priced cars (6736) are here. There are also significant medium-priced cars (4818) and few low-priced (1305). This cluster likely represents higher-spec or luxury cars such as Mercedes, Lexus, S-Class, GLC, newer models. Cluster 2 evenly mix across price levels. It represents cars with average specs and price spread such as Toyota, Mazda, Hyundai. This cluster seems to be the "Middle Market" with popular models with wide pricing ranges depending on features/age. Cluster 3 seems to be a tiny, niche segment. It is the smallest cluster (only 191 cars). Most of them are low price, with some medium price. It may contain outliers, such as very unique or rage configurations. Cluster 3 cleary contains budget vehicles with overwhelmingly low-priced (5668 / 6368 = ~89%). It represents low-cost cars with basic features, older or used condition, such as Daewoo, old sedans, basic hatchbacks, high mileage.




```
##
##       Low Medium High
##   1 4505    547  114
##   2 2010   4915 5235
##   3    0      0 2266
##   4 3573   4531 2695
```

4.3. Regression (Linear Regression)

We train the linear model and predict `log_price` using all other features in the training set. The training performance indicates a very strong fit on the training data. It means that our model explain roughly 94.6% of the variance in the logarithm of the car price (`log_price`) within the training data. This is a high R-squared for real-world data, suggesting the selected features (brand, year, mileage, condition, lumped factors, etc.) are powerful predictors. The closeness of R-squared and Adjusted R-squared confirms that model complexity isn't artificially inflating the score; the predictors are genuinely useful.

```
model <- lm(log_price ~ ., data = df_train)
print(paste("Model R-squared (Training):", round(summary(model)$r.squared, 4)))
```

```
## [1] "Model R-squared (Training): 0.9456"
```

```
print(paste("Model Adj. R-squared (Training):", round(summary(model)$adj.r.squared, 4)))
```

```
## [1] "Model Adj. R-squared (Training): 0.9453"
```

We make predictions on the test set.

```r
predictions_log <- predict(model, newdata = df_test)
```

5. Evaluation

5.1. K-means

We evaluate K-Means with ratio of Between-Cluster SS to Total SS (Pseudo R-squared). This value indicates that approximately 53.8% of the total variance (sum of squares) in the first 3PCs is captured between the clusters. The remaining ~46.2% is variance within the clusters. A value of 0.5381 suggests a moderate degree of separation between the clusters. It's significantly better than random assignment (~0), but it's not extremely high. This aligns with our PCA plot where there was some overlap, particularly between clusters 1 and 2. This indicates that the clusters capture a meaningful amount of the data's structure, but they aren't perfectly isolated groups.

```r
total_ss <- sum(scale(pca_features, center = TRUE, scale = FALSE)^2)
between_ss <- kmeans_result$betweenss
within_ss <- kmeans_result$tot.withinss
bss_tss_ratio <- between_ss / total_ss
print(paste("Ratio of Between SS / Total SS:", round(bss_tss_ratio, 4)))
```

```
## [1] "Ratio of Between SS / Total SS: 0.5381"
```

Another metric can be use for evaluating K-Means is Average Silhouette Width. It measures how similar an object is to its own cluster compared to other clusters. We got 0.3325, which is a weak or somewhat ambiguous cluster structure. It's positive, which is better than negative (suggesting points are generally closer to their own cluster center than the next nearest one). However, values below 0.5 suggest there are substantial overlap between clusters.

```r
sil_scores <- silhouette(kmeans_result$cluster, dist(pca_features))
avg_sil_width <- mean(sil_scores[, 'sil_width'])
print(paste("Average Silhouette Width:", round(avg_sil_width, 4)))
```

```
## [1] "Average Silhouette Width: 0.3325"
```

In conclusion, our K-Means clustering on the first three PCA components has identified groups that capture a reasonable amount of the data's variance (BSS/TSS 54%), but these groups are not very distinct or well-separated (Avg. Silhouette 0.33).

5.2. Linear Regression

Now we evaluate the model performance by calculate Root Mean Squared Error (RMSE) and R-squared on the test set. The RMSE indicates the typical error magnitude on the log scale. We got a very low value (0.0088), indicating that the predictions are very close to the actual values on the log scale.

```r
y_test_log <- df_test$log_price
rmse_log <- sqrt(mean((predictions_log - y_test_log)^2, na.rm = TRUE))
print(paste("RMSE (log scale):", round(rmse_log, 4)))
```

```
## [1] "RMSE (log scale): 0.2224"
```

The R-squared on the test set is the crucial metric. The fact that the R-squared on the unseen test data is almost identical to the training R-squared (0.9466 vs 0.9456) is a very strong sign that the model is not overfit. It generalizes very well to new data.

```
sst_log <- sum((y_test_log - mean(y_test_log))^2)
ssr_log <- sum((predictions_log - y_test_log)^2)
r_squared_log <- 1 - (ssr_log / sst_log)
print(paste("R-squared (test set, log scale):", round(r_squared_log, 4)))
```

```
## [1] "R-squared (test set, log scale): 0.9466"
```

Now we want to calculate the RMSE between predicted original scale (million VND, not log-transformed) and actual original scale. First we convert predictions back to original scale, then we get the actual original prices for the test set. RMSE is 4.77 Billion VND indicates that the model has a large average absolute error when converted back to original VND scale. It does not mean that every prediction is off by ~4.77 Billion VND. Predictions for lower and mid-priced cars are likely much closer in absolute terms. The average is skewed upwards by the errors on the most expensive vehicles. This is primarily driven by the squaring effect of RMSE penalizing large misses on expensive cars and the nature of back-transforming from a logarithmic scale. It highlights the challenge of achieving low absolute error across the entire price range for highly skewed data like car prices.

```
predictions_orig <- expm1(predictions_log)
actual_original_prices_test <- df_reg_base[-sample_index, ]$price_original
rmse_orig <- sqrt(mean((predictions_orig - actual_original_prices_test)^2, na.rm = TRUE))
print(paste("RMSE (original scale, Million VND):", round(rmse_orig, 2)))
```

```
## [1] "RMSE (original scale, Million VND): 4772.25"
```