Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Statistics Final Project

STAT & APPL - Dr. J Ebrahimi
Final Project

---

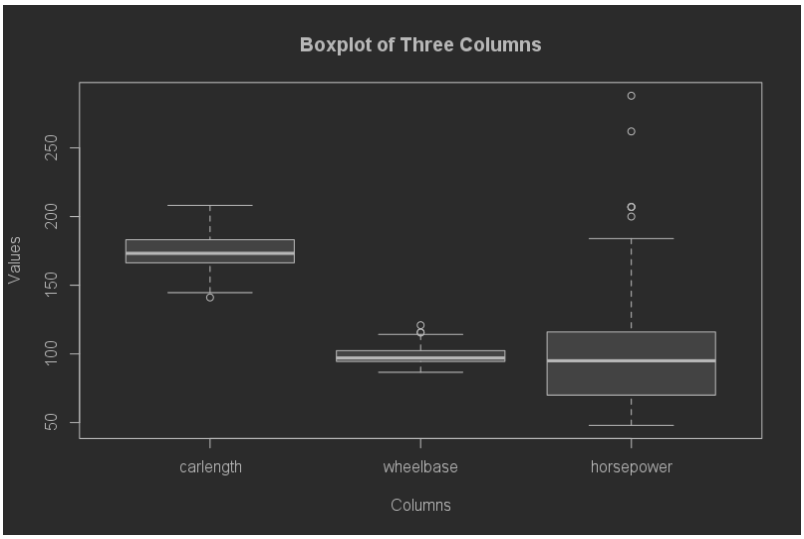
1 Loading and Preprocessing

1.1 Box-Plot 3 columns

Firstly, we load a CSV file named "CarPrice_Assignment.csv" using the "read.csv()" function. The loaded data is stored in the variable "my_data".

Next, we create a boxplot using three specific columns from the loaded data frame. The columns included in the boxplot are "carlength", "wheelbase", and "horsepower".

The resulting boxplot displays the distribution of values in the selected columns. A box plot typically consists of a rectangular box and two lines (whiskers) extending from the box. The box represents the interquartile range (IQR), which contains the middle 50% of the data. The line inside the box represents the median, which is the middle value of the dataset when it is sorted. The whiskers represent the range of the data, excluding any outliers.



The explanation of the box-plot is as below:

Median (Q2): It is the value that separates the dataset into two equal halves. Half of the values lie above the median, and half lie below it.

Quartiles (Q1 and Q3): The dataset is divided into four quartiles. Q1 represents the 25th percentile, meaning that 25% of the values are below this point. Q3 represents the 75th percentile, meaning that 75% of the values are below this point. The interquartile range (IQR) is the range between Q1 and Q3 and contains the middle 50

Whiskers: The whiskers extend from the box and represent the range of the data. By default, they usually extend 1.5 times the IQR from the upper and lower quartiles. Values beyond the whiskers are considered outliers and are plotted individually.

Outliers: Individual data points that fall outside the whiskers are plotted separately as points. Outliers may indicate extreme values or potential anomalies in the data.

1.2 Missing Values

When dealing with missing values in a table or dataset, there are several approaches to handle them without deleting any data. Here are a few common methods:

1. Imputation: Imputation involves estimating or filling in the missing values with plausible values. This can be done using various techniques such as mean imputation, median imputation, mode imputation, or more advanced methods like regression imputation or k-nearest neighbors imputation. The choice of imputation method depends on the nature of the data and the underlying assumptions.

2. Forward-fill or backward-fill: If the missing values occur in a time series or ordered data, you can use forward-fill or backward-fill methods. Forward-fill (or carry-forward) replaces missing values with the most recent preceding non-missing value, while backward-fill (or carry-backward) replaces missing values with the next available non-missing value.

3. Hot-deck imputation: Hot-deck imputation involves replacing missing values with values from similar cases or individuals within the dataset. The replacement values are chosen based on certain matching criteria, such as nearest neighbors or similar characteristics.

4. Multiple imputation: Multiple imputation is a technique that involves creating multiple imputed datasets by modeling the missing data. The missing values are imputed multiple times, generating several complete datasets. The analyses are then performed on each imputed dataset, and the results are combined to provide more robust estimates and account for uncertainty.

5. Model-based imputation: Model-based imputation utilizes statistical models to predict missing values based on other variables in the dataset. This can include techniques like linear regression, logistic regression, or machine learning algorithms.

6. Indicator variable: Another option is to create an additional binary indicator variable that denotes the presence or absence of missing values for each variable. This can help preserve the information about missingness in the data while still including the original variables.

Here we use the first proposed method and for numeric data we use median and for non-numeric data we use mode.

We use the following operation for numeric data:

```r
my_data$boreratio[is.na(my_data$boreratio)] <- median(my_data$boreratio, na.rm = TRUE)
```

And use the following function and operation for non numeric data:

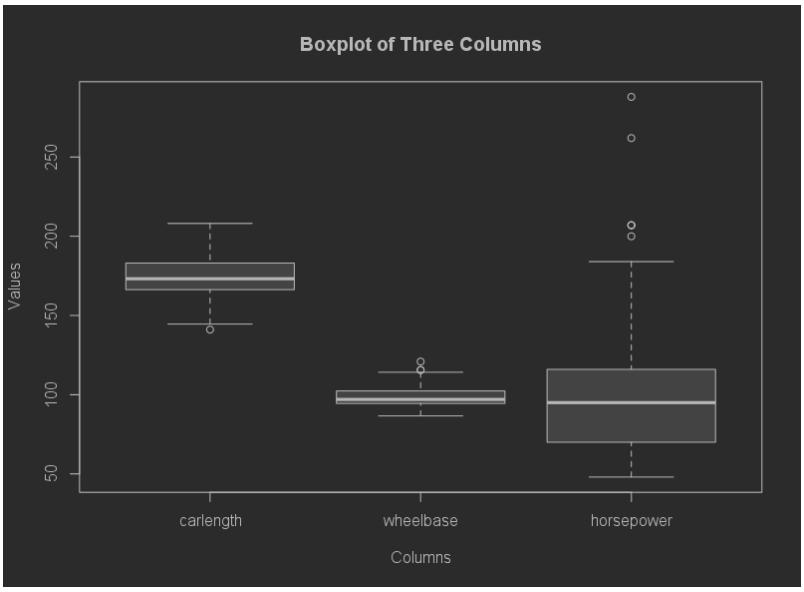
```r
# Calculate the mode function
get_mode <- function(x) {
  uniq_x <- unique(x)
  uniq_x[which.max(tabulate(match(x, uniq_x)))] ^get_mode
}

# Replace empty cells with mode
my_data$carbody[my_data$carbody == ""] <- get_mode(my_data$carbody)
my_data$cylindernumber[my_data$cylindernumber == ""] <- get_mode(my_data$cylindernumber)
```

For example a part of the data that has empty values and NA values and after applying the above operations the empty and NA cells will be filled:

carheight	curbweight	enginetype	cylindernumber	enginesize	fuelsystem	borera
48.8000	2548.00	dohc	four	130	mpfi	3.4700
48.8000	2548.00	dohc	four	130	mpfi	3.4700
52.4000	2823.00	ohcv		152	mpfi	2.6800
54.3000	2337.00	ohc	four	109	mpfi	3.1900
54.3000	*NA*	ohc		136	mpfi	3.1900
53.1000	2507.00	ohc	five	136	mpfi	*NA*
55.7000	2844.00	ohc	five	136	mpfi	*NA*
55.7000	2954.00	ohc	five	136	mpfi	3.1900
55.9000	3086.00	ohc		131	mpfi	3.1300
52.0000	*NA*	ohc	five	131	mpfi	3.1300
54.3000	2395.00	ohc		108	mpfi	3.5000
54.3000	2395.00	ohc	four	108	mpfi	3.5000
54.3000	2710.00	ohc	six	164	mpfi	3.3100
54.3000	2765.00	ohc	six	164	mpfi	3.3100
55.7000	3055.00	ohc	six	164	mpfi	3.3100
55.7000	3230.00	ohc		209	mpfi	3.6200
53.7000	3380.00	ohc		209	mpfi	3.6200
56.3000	3505.00	ohc	six	209	mpfi	3.6200
53.2000	1488.00	l	three	61	2bbl	2.9100

carheight	curbweight	enginetype	cylindernu...	enginesize	fuelsystem	boreratio
48.8000	2548.00	dohc	four	130	mpfi	3.47000
48.8000	2548.00	dohc	four	130	mpfi	3.47000
52.4000	2823.00	ohcv	four	152	mpfi	2.68000
54.3000	2337.00	ohc	four	109	mpfi	3.19000
54.3000	2417.00	ohc	four	136	mpfi	3.19000
53.1000	2507.00	ohc	five	136	mpfi	3.31000
55.7000	2844.00	ohc	five	136	mpfi	3.31000
55.7000	2954.00	ohc	five	136	mpfi	3.19000
55.9000	3086.00	ohc	four	131	mpfi	3.13000
52.0000	2417.00	ohc	five	131	mpfi	3.13000
54.3000	2395.00	ohc	four	108	mpfi	3.50000
54.3000	2395.00	ohc	four	108	mpfi	3.50000
54.3000	2710.00	ohc	six	164	mpfi	3.31000
54.3000	2765.00	ohc	six	164	mpfi	3.31000
55.7000	3055.00	ohc	six	164	mpfi	3.31000
55.7000	3230.00	ohc	four	209	mpfi	3.62000
53.7000	3380.00	ohc	four	209	mpfi	3.62000
56.3000	3505.00	ohc	six	209	mpfi	3.62000
53.2000	1488.00	l	three	61	2bbl	2.91000

Now again we plot box-plot of the three columns:

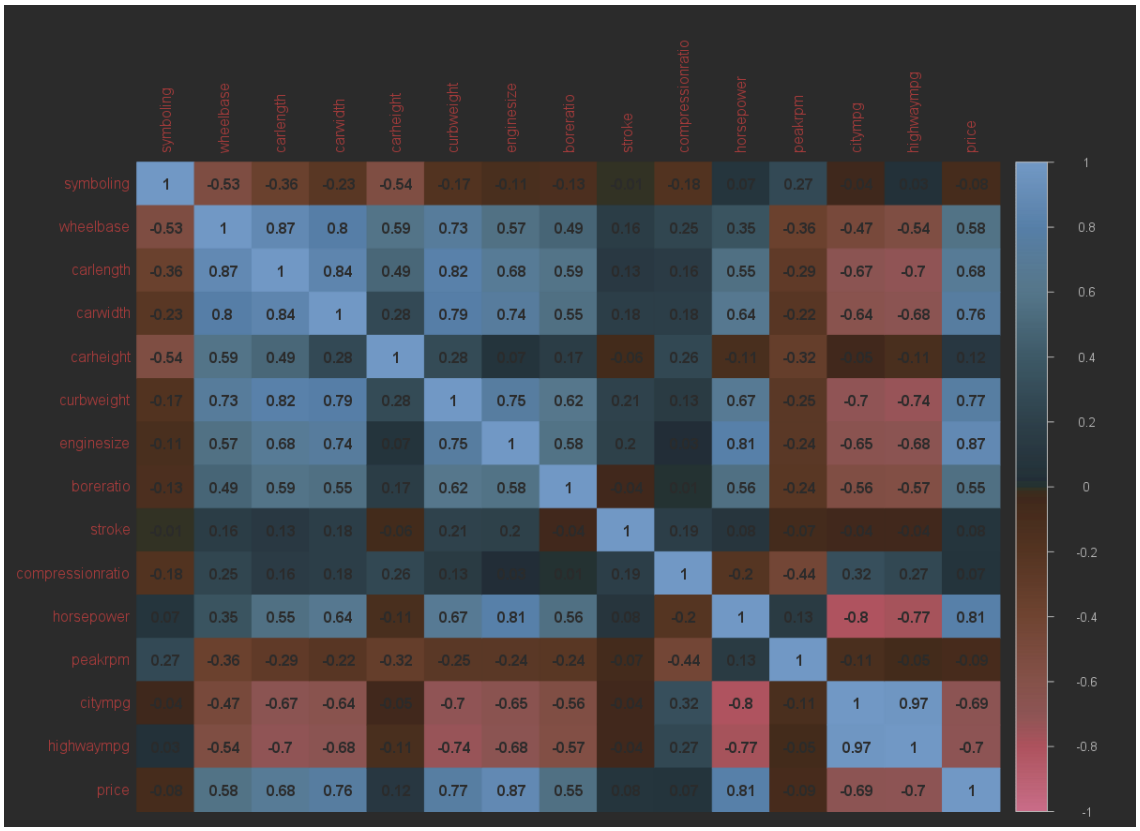


1.3 Correlation Map For Numeric Columns

First we plot the correlation map for numeric data by the following code:

```
# Select only numeric columns
numeric_data <- my_data[, sapply(my_data, is.numeric)]
# Compute the correlation matrix
cor_matrix <- cor(numeric_data)
# Print the correlation matrix
cor_matrix
# Load the required package
library(corrplot)
# Visualize the correlation matrix
corrplot(cor_matrix, method = "color", addCoef.col = 'white')
```

The corr-map is as below:



1.4 t-Test For Four Hypothesis

By considering the correlation map, we define 4 hypothesis and use the following piece of code:

```
variable1 <- my_data$carlength
variable2 <- my_data$wheelbase
# Calculate the correlation coefficient and perform a test
cor_test <- cor.test(variable1, variable2)
# Extract and examine the relevant information from the correlation test result
cor_coef <- cor_test$estimate   # Pearson correlation coefficient
p_value <- cor_test$p.value    # p-value
```

1.4.1 First Hypothesis

Car length and Wheel base are statistically correlated:

```
> p_value <- cor_test$p.value
> print(cor_test)

    Pearson's product-moment correlation

data:  variable1 and variable2
t = 25.7, df = 203, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8379841 0.9033561
sample estimates:
      cor
0.8745875

> cat("p_value:          ", p_value, "\n")
p_value:           9.699227e-66
> if (p_value < 0.05) {
+   print("Carlength and Wheelbase are significantly correlated.")
+ } else {
+   print("There is no significant correlation between Carlength and Wheelbase.")
+ }
[1] "Carlength and Wheelbase are significantly correlated."
```

So by significance level of 0.05 we can conclude that Car length and Wheel base are correlated.

1.4.2 Second Hypothesis

Engine Size and Compression ratio are statistically correlated:

```
    Pearson's product-moment correlation

data:  variable1 and variable2
t = 0.41295, df = 203, p-value = 0.6801
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1084944  0.1653499
sample estimates:
       cor
0.02897136

> cat("p_value:          ", p_value, "\n")
p_value:           0.6800779
> if (p_value < 0.05) {
+   print("Enginesize and Compressionratio are significantly correlated.")
+ } else {
+   print("There is no significant correlation between Enginesize and Compressionratio.")
+ }
[1] "There is no significant correlation between Enginesize and Compressionratio."
```

So by significance level of 0.05 we can conclude that Engine Size and Compression ratio are not correlated.

1.4.3 Third Hypothesis

Mean of the column car height is significantly different from 50.

```
    One Sample t-test

data:  variable
t = 21.826, df = 204, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 50
95 percent confidence interval:
 53.38839 54.06137
sample estimates:
mean of x
 53.72488

> if (p_value < 0.05) {
+   print("The mean of Car Height is significantly different from 50.")
+ } else {
+   print("The mean of Car Height is not significantly different from 50.")
+ }
[1] "The mean of Car Height is significantly different from 50."
```

So by significance level of 0.05 we can conclude that mean of the column car height is statistically different from 50.

1.4.4 Forth Hypothesis

Mean of the column car width is significantly different from 66.

```
    One Sample t-test

data:  variable
t = -0.61534, df = 204, p-value = 0.539
alternative hypothesis: true mean is not equal to 66
95 percent confidence interval:
 65.61240 66.20321
sample estimates:
mean of x
  65.9078

> if (p_value < 0.05) {
+   print("The mean of Car Width is significantly different from 66.")
+ } else {
+   print("The mean of Car Width is not significantly different from 66.")
+ }
[1] "The mean of Car Width is not significantly different from 66."
```

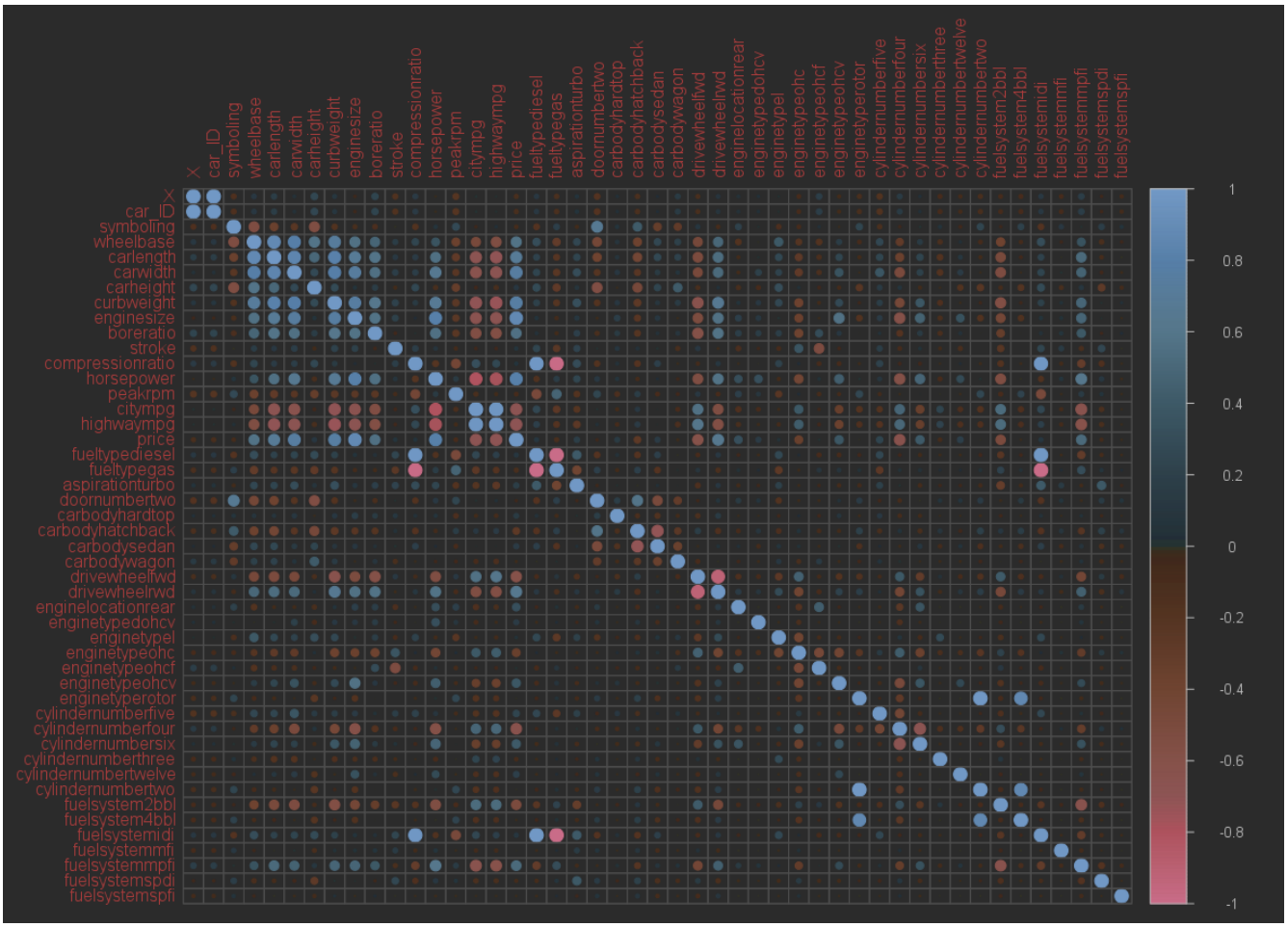
So by significance level of 0.05 we can conclude that mean of the column car height is not statistically different from 66.

1.5 Define Dummy Variables And Plot Correlation Map

To analyse the data better, we define dummy variables for non-numeric data by the following piece of code:

```
# Define dummy variables for categorical columns
my_data.dummies <- model.matrix( fueltype + aspiration + doornumber + carbody + drivewheel
+ enginelocation + enginetype + cylindernumber + fuelsystem - 1, data=my_data)
# Combine the original and dummy variables
my_data.final <- cbind(my_data, my_data.dummies)
```

The correlation map of new set of data will be:



1.6 Divide Data Set to Two Sets of Train Set And Test Set

By the following piece of code we divide the data set to train set which is 70% percent of data and test set which is 30% of data:

```r
# Load the caret package
library(caret)
# Set the random seed for reproducibility
set.seed( seed: 123)
# Split the data set into training and testing sets
training.index <- createDataPartition(numeric_data1$price, p = 0.7, list = FALSE)
training <- numeric_data1[training.index, ]
testing <- numeric_data1[-training.index, ]
```

The purpose of this division is to assess the performance and generalization ability of a model trained on the train set by evaluating it on unseen data from the test set. Here are a few justifications for using an 80-20 train-test split:

1. Sufficient Training Data: By allocating 70% of the dataset to the train set, we have a substantial amount of data for training our model. This larger train set can help the model learn patterns and relationships more effectively.

2. Adequate Test Data: With 30% of the dataset allocated to the test set, we have a reasonable amount of data to evaluate the performance of your trained model. Having a sizable test set helps provide a more reliable estimate of the model's performance on unseen data.

3. Reducing Overfitting: By evaluating the model on a separate test set, we can assess whether it has overfit the training data. Overfitting occurs when a model performs well on the training data but fails to generalize to new, unseen data. The test set helps detect overfitting and ensures that the model's performance is not inflated by memorizing the training examples.

It's important to note that the 70-30 train-test split is a common convention, but the specific split ratio can vary depending on the size of the dataset, the complexity of the problem, and other considerations. The key idea is to strike a balance between having enough data for training and having a reasonable amount of data for unbiased evaluation.

We note that number of test data should be more than number of variables.

1.7 Find Effective Features

By considering the column of price in correlation map plotted in the previous section, we can conclude that the blue and red cells show a significant relation between price and the feature corresponding to the row of the cells.

So we can see that wheelbase, car length, car width, curb weight, engine size, bore ratio, horse power, city mpg, highway mpg, drive wheel fwd, drive wheel rwd, cylinder number four, cylinder number six, fuel system mpfi and fuel system 2bbl are the features which are caused to have a higher or lower price for the car. And other variables don't have a significant affect on the price independently.

2 Processing the data with Multilinear Regression

2.1 Regress on Train data

We regress the model by the code below:

```
# Fit a multiple linear regression model
model <- lm(price ~ ., data = training[, -1])
# View the summary of the model
summary(model)
```

Now we calculate the predicted values for the training data and then calculate residuals:

```
# Calculate the predicted values for the training data
predicted <- predict(model, newdata = training[, -1])
# Calculate the residuals
residuals <- training$price - predicted
```

RSS, TSS, MSE, R-squared and adjusted R-squared of the model are as below:

```
> cat("RSS:           ", RSS, "\n")
RSS:           479341262
> cat("TSS:           ", TSS, "\n")
TSS:           8803177219
> cat("MSE:           ", MSE, "\n")
MSE:           3305802
> cat("R-squared:     ", Rsquared, "\n")
R-squared:     0.9455491
> cat("adj. R-squared:", adjRsquared, "\n")
adj. R-squared: 0.8443375
```

2.2 RSS, TSS, MSE, R-squared and adjusted R-squared

These metrics are commonly used in regression analysis to assess the performance, fit, and explanatory power of regression models. They help in understanding how well the model captures the relationships between the independent and dependent variables and in comparing different models to choose the best one for a given dataset:

RSS (Residual Sum of Squares): RSS is a measure of the overall error or discrepancy between the predicted values and the actual observed values in a regression model. It is calculated as the sum of the squared differences between the predicted values and the actual values. Lower RSS values indicate a better fit of the model to the data.

TSS (Total Sum of Squares): TSS represents the total variation in the dependent variable. It quantifies the total deviation of the observed values from their mean. TSS is calculated as the sum of the squared differences between the actual values and the mean of the dependent variable. TSS is useful for comparing the performance of a model to a baseline (e.g., a model that only predicts the mean).

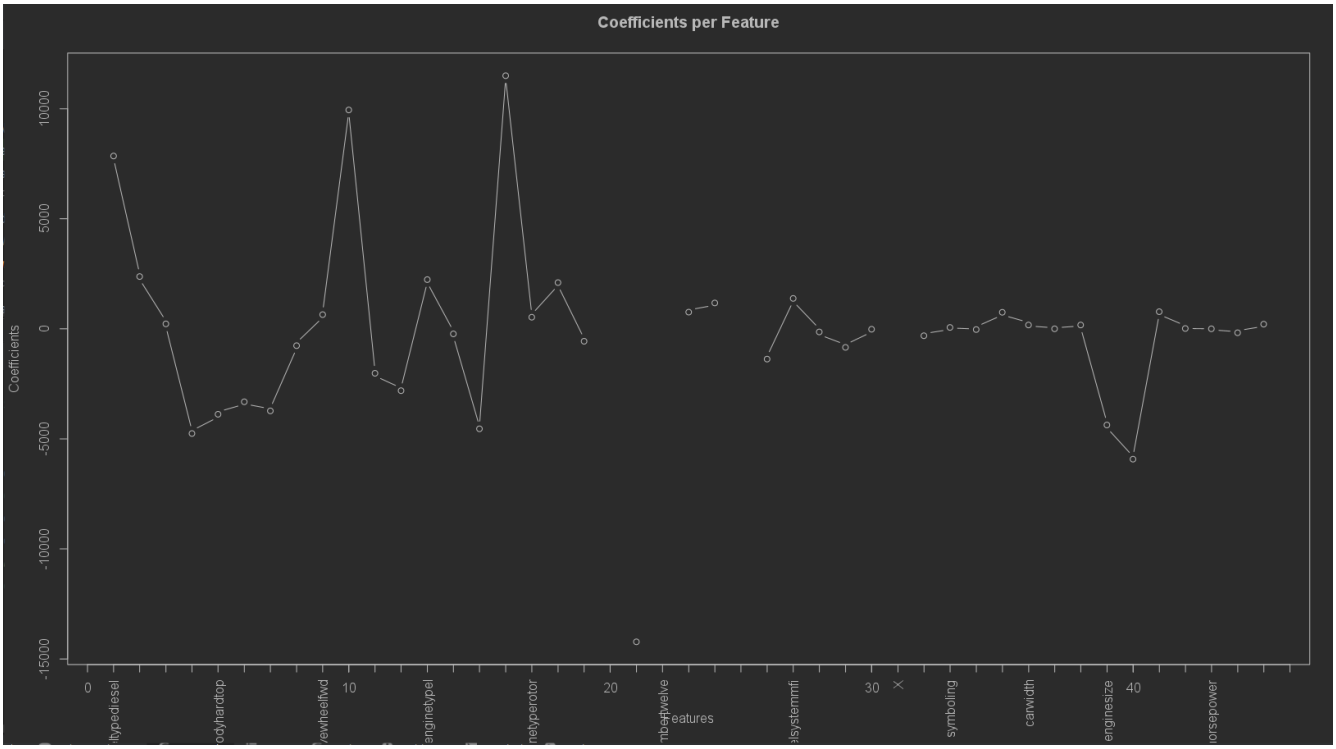
MSE (Mean Squared Error): MSE is the average of the squared differences between the predicted values and the actual values. It is calculated as the RSS divided by the number of data points or degrees of freedom. MSE is a common metric to assess the average prediction error of a regression model.

R-squared: R-squared, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that can be explained by the independent variables in a regression model. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data. R-squared is calculated as the ratio of the explained variation (sum of squares of the predicted values) to the total variation (TSS).

Adjusted R-squared: Adjusted R-squared is a modification of the R-squared metric that penalizes the inclusion of additional independent variables in a regression model. It accounts for the number of predictors and adjusts the R-squared value accordingly. Adjusted R-squared provides a more accurate measure of model fit when comparing models with different numbers of predictors.

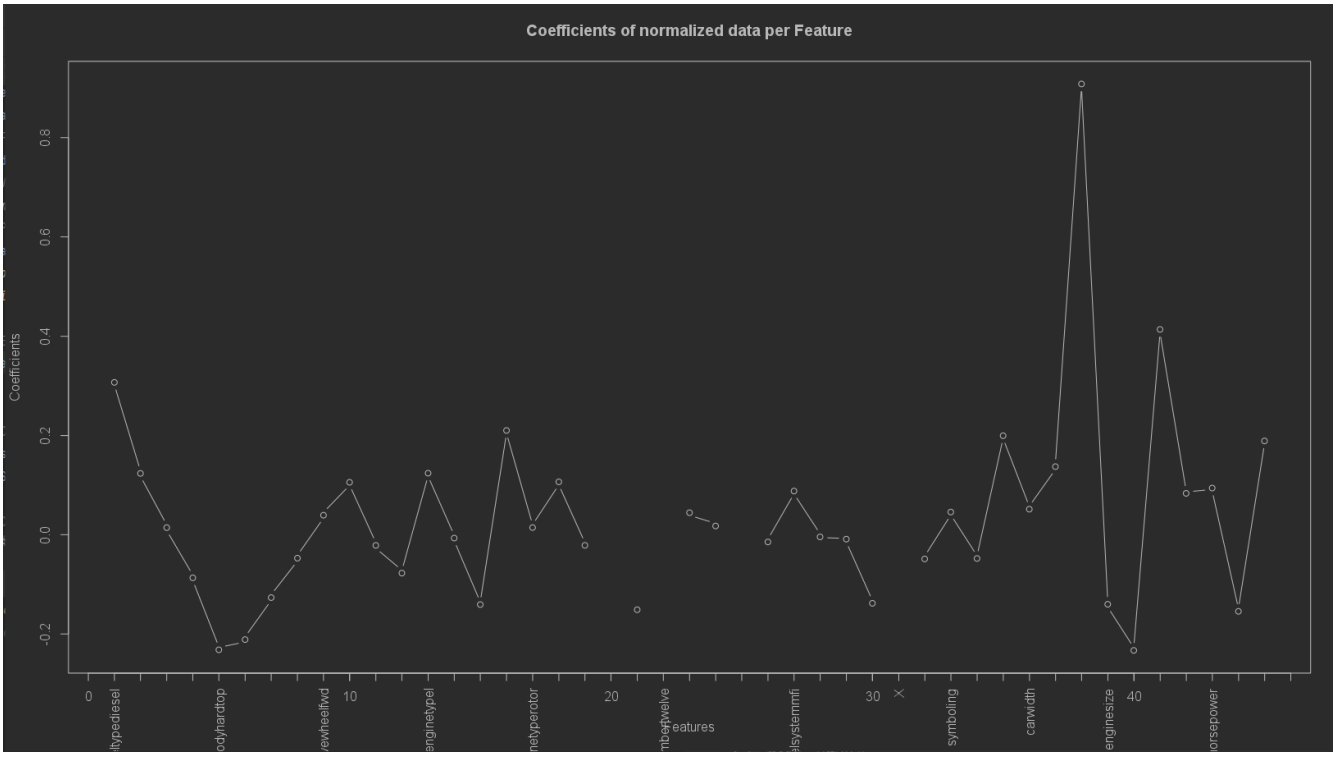
2.3 Coefficients Per Features

The following plot displays the coefficients of each feature in the multiple linear regression model.
The x-axis represents the features or independent variables used in the regression model.
The y-axis represents the coefficients, which indicate the change in the dependent variable (in this case, the ”price” variable) associated with a one-unit change in the corresponding independent variable.
Each point on the plot represents the coefficient value for a specific feature.



If there is no point for some features, the reason is singularities. Also if you want to check that the result is true, you can check the coefficient of X in the plot which is near zero. It makes sense because obviously something such as car ID does not affect the price of car.

But if a coefficient is higher, it does not indicate that its corresponding feature is more important. Because the scale of this data is not the same. Now we normalize each column of data to mean = 0 and var = 1 and then plot the coefficients:



2.4 Check the Model on Test data

Now we test our model by predicting test sample prices with the following piece of code:

```
# Evaluate the model on the testing data
predictions <- predict(model, newdata = testing[, -1])
```

Evaluating the prediction:

```
> cat("RSS:          ", RSS, "\n")
RSS:          508277857
> cat("TSS:          ", TSS, "\n")
TSS:          4215454728
> cat("MSE:          ", MSE, "\n")
MSE:          8471298
> cat("R-squared     ", Rsquared, "\n")
R-squared     0.8794251
> cat("adjRsquared   ", adjRsquared, "\n")
adjRsquared   0.4527756
```

Increasing interpretability and predictability of a regression model applied to a dataset with nearly 50 columns can be challenging. However, here are some strategies that can help:

1. Feature Selection: Start by selecting a subset of the most relevant features. Consider using techniques such as correlation analysis, feature importance ranking, or regularization methods (e.g., LASSO) to identify the most informative variables. Removing irrelevant or redundant features can simplify the model and improve interpretability.

2. Feature Engineering: Transform or combine features to create more meaningful representations. For example, you can create interaction terms, polynomial features, or statistical aggregations (e.g., mean, median, standard deviation) to capture complex relationships. These engineered features can improve model performance and make the results more interpretable.

3. Data Preprocessing: Ensure that the data is properly preprocessed before fitting the regression model. This includes handling missing values, scaling numerical features, encoding categorical variables, and addressing outliers. By standardizing the data, you can make the coefficients of the regression model directly comparable and easier to interpret.

4. Regularization Techniques: Consider using regularization techniques such as L1 (Lasso) or L2 (Ridge) regularization. These techniques introduce penalties on the coefficients of the regression model, encouraging it to select only the most important features. Regularization can help reduce overfitting and enhance interpretability.

5. Visualize the Data: Use visualizations to explore the relationship between the target variable and the features. Scatter plots, box plots, histograms, or line plots can provide insights into the data and reveal patterns. Visualizations can aid in understanding the impact of individual features on the target variable.

6. Model Interpretation Techniques: Employ model interpretation techniques specific to regression models. Some common approaches include calculating feature importances, examining coefficient values and their statistical significance, partial dependence plots, and permutation importance. These techniques help understand the relative importance and effects of each feature on the regression model's predictions.

7. Model Evaluation: Assess the model's performance using appropriate evaluation metrics such as mean squared error (MSE), mean absolute error (MAE), or R-squared. These metrics can help you quantify the model's predictive power and compare different models or feature sets.

8. Documentation: Clearly document the entire process, including feature selection, preprocessing steps, and modeling techniques. This documentation will assist in replicating and understanding the model's results in the future.

Remember, the choice of techniques will depend on the specific characteristics of your dataset and the goals of your analysis. Experiment with different approaches and iterate to find the best balance between interpretability and predictive performance.

3 Feature Selection And Analysis

3.1 Reducing the features based on t-Test and P-values

By the following piece of code we select the features which their P-values are less than 0.05 and again train the model on training data:

```r
# Perform t-test and select features based on p-values
p_values <- summary(model)$coefficients[, "Pr(>|t|)"]
significant_features <- names(p_values[p_values < 0.05 & names(p_values) != "(Intercept)"])
# Subset the training and testing data with significant features
training <- training[, c("price", significant_features)]
testing <- testing[, c("price", significant_features)]
# Fit a new model with the reduced feature set
reduced_model <- lm(price ~ ., data = training)
# View the summary of the reduced model
summary(reduced_model)
```

Significant features with significance level of 0.05:

```
 [1] "aspirationturbo"      "carbodyhardtop"       "carbodyhatchback"
 [4] "carbodysedan"         "carbodywagon"         "enginelocationrear"
 [7] "enginetypeohcv"       "enginetyperotor"      "cylindernumbertwelve"
[10] "carwidth"             "curbweight"           "enginesize"
[13] "boreratio"            "stroke"               "peakrpm"
```

The statistics of the model are as below:

```
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)           -4.333e+04  1.367e+04  -3.169 0.001909 **
aspirationturbo        1.646e+03  6.418e+02   2.564 0.011483 *
carbodyhardtop        -3.503e+03  2.088e+03  -1.678 0.095843 .
carbodyhatchback      -4.139e+03  1.461e+03  -2.833 0.005348 **
carbodysedan          -2.807e+03  1.414e+03  -1.984 0.049362 *
carbodywagon          -3.619e+03  1.595e+03  -2.268 0.024973 *
enginelocationrear     7.455e+03  2.986e+03   2.496 0.013804 *
enginetypeohcv        -7.087e+03  1.303e+03  -5.441 2.57e-07 ***
enginetyperotor        9.646e+03  1.834e+03   5.258 5.87e-07 ***
cylindernumbertwelve -1.176e+04  3.949e+03  -2.978 0.003463 **
carwidth               8.537e+02  2.079e+02   4.106 7.10e-05 ***
curbweight             1.239e+00  1.097e+00   1.130 0.260641
enginesize             1.809e+02  1.663e+01  10.878  < 2e-16 ***
boreratio             -4.818e+03  1.387e+03  -3.474 0.000698 ***
stroke                -4.727e+03  9.388e+02  -5.035 1.57e-06 ***
peakrpm                1.768e+00  5.282e-01   3.348 0.001068 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2671 on 129 degrees of freedom
Multiple R-squared:  0.8955,    Adjusted R-squared:  0.8833
F-statistic: 73.66 on 15 and 129 DF,  p-value: < 2.2e-16
```

As you see, by comparing with the previous model, our $adj - R^2$ has improved.
Now test our new model on test data:

```
> cat("RSS:          ", RSS, "\n")
RSS:          316962501
> cat("TSS:          ", TSS, "\n")
TSS:          4215454728
> cat("MSE:          ", MSE, "\n")
MSE:          5282708
> cat("R-squared     ", Rsquared, "\n")
R-squared     0.9248094
> cat("adjRsquared   ", adjRsquared, "\n")
adjRsquared   0.8991763
```

As you see in the figure above, both R^2 and $adj - R^2$ have improved.

3.2 Reducing the features based on ANOVA and f-statistics

Here by the following code which has used the anova function, we calculate the f-statistics and then sort them from higher to lower and select 10 features which have higher f-statistics:

```
# Fit the linear regression model
model <- lm(price ~ ., data = training)
anova_table = anova(model)
sorted_anova_table <- anova_table[order(anova_table$`F value`, decreasing = TRUE), ]
selected_features = rownames(sorted_anova_table)
selected_features <- selected_features[-which(selected_features == "X")]
first_10_elements <- selected_features[1:10]
# Select the 10 features from the training dataset
summary(model)
selected_features <- first_10_elements
```

Now we fit the model with these 10 features and the results are as below:

```
Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       -32722.33    9389.62  -3.485 0.000666 ***
drivewheelfwd        -284.73    2268.25  -0.126 0.900293
carbodyhatchback    -1374.10     851.42  -1.614 0.108903
cylindernumbersix   -4752.93    2052.05  -2.316 0.022066 *
wheelbase             520.48      91.18   5.708 7.02e-08 ***
cylindernumberfive  -4049.88    2673.53  -1.515 0.132177
drivewheelrwd        6297.92    2398.49   2.626 0.009651 **
enginetypel         -7727.96    2010.94  -3.843 0.000187 ***
aspirationturbo      2217.27     961.20   2.307 0.022602 *
cylindernumberfour  -7854.11    2084.81  -3.767 0.000246 ***
enginetypeohcv       2901.98    1795.26   1.616 0.108345
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4343 on 134 degrees of freedom
Multiple R-squared:  0.7129,    Adjusted R-squared:  0.6915
F-statistic: 33.28 on 10 and 134 DF,  p-value: < 2.2e-16
```

Both R^2 and $adj - R^2$ have decreased. The reason is that we used just 10 features. Now to improve the fitting accuracy we combine P-value and f-statistics method. I mean that we will choose these 10 features from the features that have a P-value less than 0.05.

```
Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)          -59490.509  14652.733  -4.060 8.30e-05 ***
carwidth                916.155    240.105   3.816 0.000207 ***
enginetypeohcv        -3798.629   1400.755  -2.712 0.007568 **
enginesize              119.292     16.312   7.313 2.14e-11 ***
curbweight                2.773      1.244   2.230 0.027445 *
enginelocationrear    11283.332   3437.340   3.283 0.001312 **
carbodyhatchback       -707.156    597.281  -1.184 0.238526
aspirationturbo         360.507    734.081   0.491 0.624159
carbodywagon          -1319.528   1032.131  -1.278 0.203301
stroke                -2792.801    988.601  -2.825 0.005451 **
cylindernumbertwelve  -3274.084   4423.592  -0.740 0.460509
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3167 on 134 degrees of freedom
Multiple R-squared:  0.8473,    Adjusted R-squared:  0.8359
F-statistic: 74.36 on 10 and 134 DF,  p-value: < 2.2e-16
```

It did not reach to accuracy of the model which we used all features that had P-value less than 0.05. The reason is that we used less features.

3.3 Synergy Features

By adding synergy variable ”carwidth * enginetypeohcv” we will have:

```
Residual standard error: 3172 on 133 degrees of freedom
Multiple R-squared:  0.848, Adjusted R-squared:  0.8354
F-statistic: 67.44 on 11 and 133 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.

By adding synergy variable ”enginesize * curbweight” we will have:

```
Residual standard error: 3049 on 133 degrees of freedom
Multiple R-squared:  0.8595,    Adjusted R-squared:  0.8479
F-statistic: 73.97 on 11 and 133 DF,  p-value: < 2.2e-16
```

We can conclude that this synergy variable has a significant good effect on R^2 and $adj - R^2$.

By adding synergy variable ”enginelocationrear * carbodyhatchback” we will have:

```
Residual standard error: 3167 on 134 degrees of freedom
Multiple R-squared:  0.8473,    Adjusted R-squared:  0.8359
F-statistic: 74.36 on 10 and 134 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.

By adding synergy variable ”aspirationturbo * carbodywagon” we will have:

```
Residual standard error: 3120 on 133 degrees of freedom
Multiple R-squared:  0.853, Adjusted R-squared:  0.8408
F-statistic: 70.14 on 11 and 133 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important. By adding synergy variable "stroke * cylindernumbertwelve" we will have:

```
Residual standard error: 3167 on 134 degrees of freedom
Multiple R-squared:  0.8473,    Adjusted R-squared:  0.8359
F-statistic: 74.36 on 10 and 134 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.
By adding synergy variable "carwidth * carbodyhatchback" we will have:

```
Residual standard error: 3158 on 133 degrees of freedom
Multiple R-squared:  0.8493,    Adjusted R-squared:  0.8369
F-statistic: 68.15 on 11 and 133 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.
By adding synergy variable "carwidth * enginelocationrear" we will have:

```
Residual standard error: 3167 on 134 degrees of freedom
Multiple R-squared:  0.8473,    Adjusted R-squared:  0.8359
F-statistic: 74.36 on 10 and 134 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.
By adding synergy variable "carbodywagon * enginetypeohcv" we will have:

```
Residual standard error: 3160 on 133 degrees of freedom
Multiple R-squared:  0.8491,    Adjusted R-squared:  0.8366
F-statistic: 68.04 on 11 and 133 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.
By adding synergy variable "enginetypeohc * enginetypeohcv" we will have:

```
Residual standard error: 3167 on 134 degrees of freedom
Multiple R-squared:  0.8473,    Adjusted R-squared:  0.8359
F-statistic: 74.36 on 10 and 134 DF,  p-value: < 2.2e-16
```

We do not see any significant effect on R^2 and $adj - R^2$. So this synergy variable is not important.
By adding synergy variable "enginetypel * horsepower" we will have:

```
Residual standard error: 3071 on 133 degrees of freedom
Multiple R-squared:  0.8575,    Adjusted R-squared:  0.8457
F-statistic: 72.76 on 11 and 133 DF,  p-value: < 2.2e-16
```

We can conclude that this synergy variable has a significant good effect on R^2 and $adj - R^2$.

By adding synergy variable "enginesize * horsepower" we will have:

```
Residual standard error: 3036 on 133 degrees of freedom
Multiple R-squared:  0.8608,    Adjusted R-squared:  0.8492
F-statistic: 74.74 on 11 and 133 DF,  p-value: < 2.2e-16
```

We can conclude that this synergy variable has a significant good effect on R^2 and $adj - R^2$.

3.4 Another way to check Synergy Features

Now we add all ten products to our columns and based on P-values we decide that a synergy variable is important or not:

```
training_selected$synergy <- training$horsepower * training$enginesize
training_selected$synergy2 <- training$enginetypeohc * training$horsepower
training_selected$synergy3 <- training_selected$carbodywagon * training_selected$enginetypeohcv
training_selected$synergy4 <- training_selected$carwidth * training_selected$enginetypeohc
training_selected$synergy5 <- training_selected$carwidth * training_selected$carbodyhatchback
training_selected$synergy6 <- training_selected$stroke * training$carbodyhardtop
training_selected$synergy7 <- training_selected$aspirationturbo * training_selected$carbodywagon
training_selected$synergy8 <- training_selected$enginetypeohc * training_selected$carbodyhatchback
training_selected$synergy9<- training_selected$enginesize * training_selected$curbweight
training_selected$synergy10 <- training_selected$carwidth * training$carbodyhardtop
model <- lm(price ~ ., data = training_selected)
summary(model)
```

If a variable has P-value less than 0.025 it is effective on accuracy of fitting the model. As you see in the figure below,

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)           -4.933e+04  1.437e+04  -3.433 0.000812 ***
carwidth               1.051e+03  2.180e+02   4.819 4.14e-06 ***
enginetypeohcv         2.541e+04  5.626e+04   0.452 0.652264
enginesize            -1.013e+02  4.707e+01  -2.152 0.033322 *
curbweight            -1.203e+00  2.056e+00  -0.585 0.559505
enginelocationrear     1.193e+04  3.329e+03   3.584 0.000484 ***
carbodyhatchback       4.224e+04  2.377e+04   1.777 0.077960 .
aspirationturbo       -6.936e+02  6.767e+02  -1.025 0.307350
carbodywagon          -1.529e+03  1.026e+03  -1.490 0.138882
stroke                -3.044e+03  9.612e+02  -3.166 0.001943 **
cylindernumbertwelve -1.517e+03  6.697e+03  -0.226 0.821192
synergy                3.320e-01  9.427e-02   3.522 0.000600 ***
synergy2               2.351e+01  6.073e+00   3.871 0.000174 ***
synergy3              -4.065e+03  3.658e+03  -1.111 0.268648
synergy4              -4.050e+02  8.166e+02  -0.496 0.620767
synergy5              -6.666e+02  3.643e+02  -1.830 0.069689 .
synergy6              -6.685e+04  3.015e+04  -2.217 0.028433 *
synergy7               4.657e+03  2.351e+03   1.981 0.049802 *
synergy8               2.014e+03  2.936e+03   0.686 0.494004
synergy9               4.041e-02  1.574e-02   2.567 0.011461 *
synergy10              3.477e+03  1.575e+03   2.208 0.029093 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

the variables below can help the prediction:
horsepower × enginesize, enginetypeohc × horsepower, enginesize × curbweight.

4 Learn Other Models on Data

4.1 SVM

SVM stands for Support Vector Machines. It is a powerful supervised learning algorithm used for classification and regression tasks. SVMs are particularly effective when dealing with complex datasets and have been widely used in various domains, including image classification, text categorization, bioinformatics, and finance.

The main idea behind SVM is to find an optimal hyperplane that separates the data points of different classes in a high-dimensional feature space. The hyperplane is chosen in such a way that it maximizes the margin or the distance between the classes, allowing for better generalization and improved classification performance on unseen data.

Here are some key concepts associated with SVM:

1. Support Vectors: Support vectors are the data points closest to the decision boundary (hyperplane) and have the most influence in determining the hyperplane's position. They are crucial in defining the decision boundary and are used to calculate the margin.

2. Kernel Trick: The kernel trick is a technique used in SVM to transform the input data into a higher-dimensional feature space, where it becomes easier to find a hyperplane that separates the classes. This allows SVMs to handle complex data distributions and nonlinear decision boundaries.

3. Soft Margin: In cases where the data is not linearly separable, SVMs can be used with a soft margin. A soft margin allows for some misclassifications by introducing a penalty term for data points that fall within the margin or are misclassified. This flexibility allows SVMs to handle overlapping classes and noisy data.

4. C Parameter: The C parameter in SVM controls the trade-off between maximizing the margin and minimizing the misclassifications. A smaller value of C creates a larger margin but may allow more misclassifications, while a larger C puts more emphasis on correctly classifying each data point but may result in a smaller margin.

SVMs are known for their ability to handle high-dimensional data, robustness to outliers, and good generalization performance. However, they can be computationally expensive for large datasets.

```r
svm_model <- svm(price ~ ., data = training, kernel = "radial")
summary(svm_model)
predictions <- predict(svm_model, newdata = testing)
```

Results:

```
Call:
svm(formula = price ~ ., data = training, kernel = "radial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.02222222
    epsilon:  0.1
```

```
> cat("RSS:          ", RSS, "\n")
RSS:          989095611
> cat("TSS:          ", TSS, "\n")
TSS:          4215454728
> cat("MSE:          ", MSE, "\n")
MSE:          16484927
> cat("R-squared     ", Rsquared, "\n")
R-squared     0.7653644
> cat("adjRsquared   ", adjRsquared, "\n")
adjRsquared   0.01117868
```

For different values of TOL:

```
> tolerance <- 1000
> accurate_predictions <- abs(predictions - testing$price) <= tolerance
> accuracy <- sum(accurate_predictions) / length(testing$price)
> accuracy
[1] 0.4833333
```

```
> tolerance <- 2000
> accurate_predictions <- abs(predictions - testing$price) <= tolerance
> accuracy <- sum(accurate_predictions) / length(testing$price)
> accuracy
[1] 0.75
```

```
> tolerance <- 3000
> accurate_predictions <- abs(predictions - testing$price) <= tolerance
> accuracy <- sum(accurate_predictions) / length(testing$price)
> accuracy
[1] 0.8166667
```

4.2 Random Forest

A random forest model is an ensemble learning method used in machine learning. It is based on the concept of decision trees, which are individual models that can make predictions by partitioning the input space into regions and assigning labels to those regions.

A random forest combines multiple decision trees to make more accurate predictions. The ”forest” in random forest refers to the collection of decision trees. Each tree in the forest is trained independently on a random subset of the training data, using a technique called bootstrap aggregating or ”bagging.”

In a random forest, when making a prediction, each tree in the forest independently produces a prediction, and the final prediction is determined by combining the individual predictions through voting or averaging. This ensemble approach helps to reduce overfitting and improve the model's generalization performance.

Random forests have several advantages. They can handle both classification and regression tasks, they are robust to outliers and noisy data, and they can automatically select important features. Additionally, random forests can provide estimates of feature importance, which can be helpful for understanding the underlying relationships in the data.

We implement it by the following code:

```r
install.packages( pkgs: "randomForest")  # install the randomForest package if not already installed
library(randomForest)
# Specify the predictor variables and the target variable
predictors <- names(training)[-which(names(training) == "price")]
target <- "price"
# Train the Random Forest model
rf_model <- randomForest(x = training[, predictors],
                         y = training[, target],
                         ntree = 100,  # number of trees in the forest
                         importance = TRUE)  # calculate variable importance
predictions <- predict(rf_model, newdata = testing[, predictors])
```

We evaluate it by the following parameters:

```r
# Mean Absolute Error (MAE)
mae <- mean(abs(predictions - testing$price))
# Root Mean Squared Error (RMSE)
rmse <- sqrt(mean((predictions - testing$price)^2))
# R-squared (coefficient of determination)
r_squared <- 1 - sum((predictions - testing$price)^2) / sum((mean(training$price) - testing$price)^2)
```

```
> cat("Mean Absolute Error (MAE):", mae, "\n")
Mean Absolute Error (MAE): 1658.327
> cat("Root Mean Squared Error (RMSE):", rmse, "\n")
Root Mean Squared Error (RMSE): 2256.763
> cat("R-squared:", r_squared, "\n")
R-squared: 0.9275344
```

4.3 SVR

SVR stands for Support Vector Regression. It is a machine learning algorithm used for regression tasks, which involve predicting continuous numeric values rather than discrete classes. SVR is based on the concept of Support Vector Machines (SVM) and extends it to regression problems.

The main idea behind SVR is to find a hyperplane in a high-dimensional feature space that best fits the training data. Unlike traditional regression algorithms that aim to minimize the error between predicted and actual values, SVR focuses on minimizing the deviation of predictions from a specified margin around the target values. The margin is defined by two hyperparameters: (epsilon) and C.

In SVR, the training data points that lie within the margin are considered support vectors, and they influence the placement of the hyperplane. SVR aims to find a hyperplane that has the maximum possible margin while still fitting within the -tube around the target values. This -tube acts as a tolerance for errors.

SVR can handle nonlinear relationships between features and target values by employing a technique called the kernel trick. The kernel trick allows SVR to implicitly map the input features to a higher-dimensional space, where it becomes easier to find a linear hyperplane that separates the data points.

To summarize, SVR is a regression algorithm based on Support Vector Machines. It seeks to find a hyperplane that best fits the training data within a specified margin of tolerance, using the kernel trick to handle nonlinear relationships.

We implement it by the following code:

```r
install.packages( pkgs: "e1071")
library(e1071)
# Specify the formula for the model
formula <- price ~ .
# Train the SVR model
svr_model <- svm(formula, data = training, kernel = "radial", epsilon = 0.5)
# Make predictions on testing data
predictions <- predict(svr_model, newdata = testing)
```

We evaluate it by the following parameters:

```r
mae <- mean(abs(predictions - testing$price))
rmse <- sqrt(mean((predictions - testing$price)^2))
ssr <- sum((predictions - mean(testing$price))^2)
sst <- sum((testing$price - mean(testing$price))^2)
r2 <- 1 - (ssr / sst)
n <- nrow(testing)
p <- ncol(testing) - 1  # Exclude the target variable
adj_r2 <- 1 - ((1 - r2) * ((n - 1) / (n - p - 1)))
```

```
> mae
[1] 2147.686
> rmse
[1] 4060.163
> r2
[1] 0.4294982
> adj_r2
[1] -1.404258
```

4.4 Decision Tree

A decision tree is a flowchart-like model that represents decisions and their possible consequences as a tree-like structure. It is a supervised machine learning algorithm used for both classification and regression tasks.

In a decision tree, the internal nodes represent tests on input features, while the branches represent the possible outcomes of those tests. The leaves of the tree represent the predicted or class labels. The decision tree algorithm constructs this tree by recursively partitioning the data based on the values of the input features, aiming to minimize the impurity or maximize the information gain at each step.

The process of constructing a decision tree involves the following steps:

1. Feature Selection: The algorithm evaluates different features to determine which ones are most informative for making decisions. It chooses the feature that best splits the data into distinct classes or reduces the uncertainty the most (e.g., using measures like Gini impurity or information gain).

2. Splitting: Once a feature is selected, the algorithm determines a threshold or set of values that divide the data into two or more subsets. Each subset corresponds to a branch stemming from the selected feature node.

3. Recursive Partitioning: The algorithm repeats the process for each subset, creating child nodes and splitting them further based on the selected features until a stopping criterion is met. This criterion

could be reaching a maximum depth, having a minimum number of samples in a leaf node, or any other condition defined by the user.

4. Leaf Node Prediction: Once the recursive partitioning is complete, each leaf node is assigned a predicted value for regression tasks or a class label for classification tasks. For regression, this value is often the average or median of the target variable values in that leaf. For classification, it could be the majority class or the class probabilities.

During the prediction phase, a new input instance traverses the decision tree by following the branches based on the values of the corresponding features. Eventually, it reaches a leaf node, and the predicted value or class label associated with that leaf is returned as the output.

Decision trees have several advantages, including interpretability, as the resulting model can be visualized and easily understood. They can handle both categorical and numerical features, and they are robust to outliers and missing values. However, decision trees can be prone to overfitting, particularly when the tree becomes too complex or the training data contains noise. To mitigate overfitting, techniques like pruning, ensemble methods (such as random forests or gradient boosting), and regularization can be employed.

Overall, decision trees are versatile and widely used in machine learning due to their simplicity, interpretability, and ability to handle various types of data.

We implement it by the following code:

```
# Specify the formula for the model
formula <- price ~ .
# Train the decision tree model
dt_model <- rpart(formula, data = training)
# Predict on the testing data
predictions <- predict(dt_model, newdata = testing)
# Calculate the mean squared error (MSE) for the predictions
mse <- caret::RMSE(predictions, testing$price)
print(paste("Root Mean Squared Error (RMSE):", mse))
```

We evaluate it by the following parameters:

```
# Calculate the mean squared error (MSE) for the predictions
mse <- caret::RMSE(predictions, testing$price)
print(paste("Root Mean Squared Error (RMSE):", mse))
tolerance <- 2000  # Set the tolerance level
# Assuming 'predictions' contain the SVM model predictions and 'actual_labels' contains the true class labels
accurate_predictions <- abs(predictions - testing$price) <= tolerance
accuracy <- sum(accurate_predictions) / length(testing$price)
accuracy
```

Result:

```
> print(paste("Root Mean Squared Error (RMSE):", mse))
[1] "Root Mean Squared Error (RMSE): 3379.15731456325"
> tolerance <- 2000
> accurate_predictions <- abs(predictions - testing$price) <= tolerance
> accuracy <- sum(accurate_predictions) / length(testing$price)
> accuracy
[1] 0.5166667
```