


```
install.packages("gridExtra")
```

 Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Setup

This function will download the dataset into your browser

This dataset was hosted on IBM Cloud object. Click [HERE](#) for free storage.

```
# URL tải file dữ liệu
```

```
path <- 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20file:'
```

you will need to download the dataset; if you are running locally, please comment out the following

```
# Đặt tên file mới khi lưu về local
```

```
local_file <- "auto.csv"
```

```
# Hàm download file
```

```
download_data <- function(url, filename) {
  tryCatch({
    download.file(url = url,
                  destfile = filename,
                  mode = "wb")

    if (file.exists(filename)) {
      message("File downloaded successfully!")
      return(TRUE)
    } else {
      warning("Download failed - File does not exist")
      return(FALSE)
    }
  }, error = function(e) {
    warning(paste("Error downloading file:", e$message))
    return(FALSE)
  })
}
```

```
# Tải file về máy
```

```
download_data(path, local_file)
```

```
# Cập nhật đường dẫn để trỏ đến file local
```

```
path <- local_file
```

```
# Đọc dữ liệu nếu file tồn tại
```

```
if (file.exists(path)) {
  df <- read.csv(path)
  print(head(df)) # Hiển thị vài dòng đầu tiên của dữ liệu
} else {
  warning("File không tồn tại sau khi tải về. Vui lòng kiểm tra URL và thử lại.")
}
```

File downloaded successfully!

```
TRUE
X Unnamed..0 symboling normalized.losses make aspiration num.of.doors
1 0 0 3 122 alfa-romero std two
2 1 1 3 122 alfa-romero std two
3 2 2 1 122 alfa-romero std two
4 3 3 2 164 audi std four
5 4 4 2 164 audi std four
6 5 5 2 122 audi std two
body.style drive.wheels engine.location wheel.base length width
1 convertible rwd front 88.6 0.8111485 0.8902778
2 convertible rwd front 88.6 0.8111485 0.8902778
3 hatchback rwd front 94.5 0.8226814 0.9097222
4 sedan fwd front 99.8 0.8486305 0.9194444
5 sedan 4wd front 99.4 0.8486305 0.9222222
6 sedan fwd front 99.8 0.8519942 0.9208333
height curb.weight engine.type num.of.cylinders engine.size fuel.system bore
1 48.8 2548 dohc four 130 mpfi 3.47
2 48.8 2548 dohc four 130 mpfi 3.47
3 52.4 2823 ohcv six 152 mpfi 2.68
4 54.3 2337 ohc four 109 mpfi 3.19
5 54.3 2824 ohc five 136 mpfi 3.19
6 53.1 2507 ohc five 136 mpfi 3.19
stroke compression.ratio horsepower peak.rpm city.mpg highway.mpg price
1 2.68 9.0 111 5000 21 27 13495
2 2.68 9.0 111 5000 21 27 16500
3 3.47 9.0 154 5000 19 26 16500
4 3.40 10.0 102 5500 24 30 13950
5 3.40 8.0 115 5500 18 22 17450
6 3.40 8.5 110 5500 19 25 15250
city.L.100km horsepower.binned diesel gas
1 11.190476 Medium 0 1
2 11.190476 Medium 0 1
3 12.368421 Medium 0 1
4 9.791667 Medium 0 1
5 13.055556 Medium 0 1
6 10.300000 Medium 0 1
```

First, let's only use numeric data:

```
str(df)
```

```
'data.frame': 201 obs. of 31 variables:
 $ X : int 0 1 2 3 4 5 6 7 8 9 ...
 $ Unnamed..0 : int 0 1 2 3 4 5 6 7 8 9 ...
 $ symboling : int 3 3 1 2 2 2 1 1 1 2 ...
 $ normalized.losses: int 122 122 122 164 164 122 158 122 158 192 ...
 $ make : chr "alfa-romero" "alfa-romero" "alfa-romero" "audi" ...
 $ aspiration : chr "std" "std" "std" "std" ...
 $ num.of.doors : chr "two" "two" "two" "four" ...
 $ body.style : chr "convertible" "convertible" "hatchback" "sedan" ...
 $ drive.wheels : chr "rwd" "rwd" "rwd" "fwd" ...
 $ engine.location : chr "front" "front" "front" "front" ...
 $ wheel.base : num 88.6 88.6 94.5 99.8 99.4 ...
 $ length : num 0.811 0.811 0.823 0.849 0.849 ...
 $ width : num 0.89 0.89 0.91 0.919 0.922 ...
 $ height : num 48.8 48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9 54.3 ...
 $ curb.weight : int 2548 2548 2823 2337 2824 2507 2844 2954 3086 2395 ...
 $ engine.type : chr "dohc" "dohc" "ohcv" "ohc" ...
 $ num.of.cylinders : chr "four" "four" "six" "four" ...
 $ engine.size : int 130 130 152 109 136 136 136 136 108 ...
 $ fuel.system : chr "mpfi" "mpfi" "mpfi" "mpfi" ...
 $ bore : num 3.47 3.47 2.68 3.19 3.19 3.19 3.19 3.19 3.13 3.5 ...
 $ stroke : num 2.68 2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 2.8 ...
 $ compression.ratio: num 9 9 9 10 8 8.5 8.5 8.5 8.3 8.8 ...
 $ horsepower : num 111 111 154 102 115 110 110 110 140 101 ...
 $ peak.rpm : num 5000 5000 5000 5500 5500 5500 5500 5500 5500 5800 ...
 $ city.mpg : int 21 21 19 24 18 19 19 19 17 23 ...
 $ highway.mpg : int 27 27 26 30 22 25 25 25 20 29 ...
 $ price : num 13495 16500 16500 13950 17450 ...
 $ city.L.100km : num 11.19 11.19 12.37 9.79 13.06 ...
 $ horsepower.binned: chr "Medium" "Medium" "Medium" "Medium" ...
 $ diesel : int 0 0 0 0 0 0 0 0 0 ...
 $ gas : int 1 1 1 1 1 1 1 1 1 ...
```

```
# Kiểm tra dữ liệu
head(df)
```



A data.frame: 6 × 31

	X	Unnamed..0	symboling	normalized.losses	make	aspiration	num.of.doors	body.style	drive.wheels	engine.location	...	comp
	<int>	<int>	<int>	<int>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	...	
1	0	0	3	122	alfa-romero	std	two	convertible	rwd	front	...	
2	1	1	3	122	alfa-romero	std	two	convertible	rwd	front	...	
3	2	2	1	122	alfa-romero	std	two	hatchback	rwd	front	...	
4	3	3	2	164	audi	std	four	sedan	fwd	front	...	
5	4	4	2	164	audi	std	four	sedan	4wd	front	...	
6	5	5	2	122	audi	std	two	sedan	fwd	front	...	

Part 1: Training and Testing

An important step in testing your model is to split your data into training and testing data. We will place the target data **price** in a separate dataframe **y_data**:

```
# Tạo dataframe y_data chứa biến price
y_data <- data.frame(price = df$price)
```

```
# Kiểm tra kết quả
head(y_data)
```



```
A
data.frame:
  6 × 1
  price
  <dbl>
```

```
1 13495
2 16500
3 16500
4 13950
5 17450
6 15250
```

Drop price data in dataframe **x_data**:

```
#x_data <- subset(df, select = -price)
```

```
# Sử dụng toán tử [-]
x_data <- df[, !names(df) %in% c("price")]
```

```
# Kiểm tra kết quả
head(x_data)
dim(x_data)
```



A data.frame: 6 × 30

	X	Unnamed..0	symboling	normalized.losses	make	aspiration	num.of.doors	body.style	drive.wheels	engine.location	...	strol
	<int>	<int>	<int>	<int>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	...	<dbl>
1	0	0	3	122	alfa-romero	std	two	convertible	rwd	front	...	2.6
2	1	1	3	122	alfa-romero	std	two	convertible	rwd	front	...	2.6
3	2	2	1	122	alfa-romero	std	two	hatchback	rwd	front	...	3.4
4	3	3	2	164	audi	std	four	sedan	fwd	front	...	3.4
5	4	4	2	164	audi	std	four	sedan	4wd	front	...	3.4
6	5	5	2	122	audi	std	two	sedan	fwd	front	...	3.4

201 · 30

Section 1: Splitting Data for Training and Testing (90-10 Split)

In this section, you will split the data into Training and Testing sets. The goal is to create a training set containing 90% of the data and a testing set containing 10% of the data. This random but controlled split ensures that the model's training and evaluation are reliable.

Using the indices created, split the data into two sets: **x_train** and **x_test** from **x_data**, and **y_train** and **y_test** from **y_data**.

```
install.packages("caret")
library(caret)
```



Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependencies 'listenv', 'parallelly', 'future', 'globals', 'shape', 'future.apply', 'numDeriv', 'progressr', 'SQUARE'

Loading required package: ggplot2

Loading required package: lattice

```
set.seed(123)
# TO DO: Use createDataPartition to generate indices for splitting the data
training_index <- createDataPartition(y_data$price, p = 0.9, list = FALSE)

# TO DO: Split the data into training and testing sets
x_train <- x_data[training_index, ]
x_test <- x_data[-training_index, ]
y_train <- y_data[training_index, ]
y_test <- y_data[-training_index, ]

# TO DO: Print the number of samples in each set and check the data dimensions
cat("number of test samples:", nrow(x_test), "\n")
cat("number of training samples:", nrow(x_train))
#check
dim(x_train)
dim(x_test)
```



number of test samples: 20
number of training samples: 181
181 · 30

Questions:

1. What is the purpose of setting a seed with `set.seed(123)`?

Answer: The `set.seed()` function is used to ensure that the random number generator (RNG) produces the same sequence of random numbers each time the code is run. This is important for reproducibility, as it allows you to get the same results when you rerun your code.

2. Why is the `createDataPartition` function used, and what does setting `p = 0.9` mean?

Answer: The `createDataPartition()` function is used to create balanced partitions of the data. Setting $p = 0.9$ means that 90% of the data will be randomly assigned to the training set and 10% to the testing set.

✓ Section 2: Building and Evaluating a Linear Regression Model (60-40 Split)

In this section, you will:

1. Split the data into a new training and testing set using a 60% training and 40% testing ratio.
2. Fit a linear regression model using horsepower as the predictor for price based on the training data.
3. Evaluate the model using the R-squared metric for both the testing and training sets to assess the model's performance.

```
# Thiết lập random seed
set.seed(123)
```

```
# TO DO: Use createDataPartition to generate indices for splitting the data (60% for training)
training_index1 <- createDataPartition(y_data$price, p = 0.6, list = FALSE)
```

```
#Use the indices to split the data into x_train1 and x_test1 from x_data, and y_train1 and y_test1 from y_data.
# TO DO: Split the data into training and testing sets
x_train1 <- x_data[training_index1, ]
x_test1 <- x_data[-training_index1, ]
y_train1 <- y_data[training_index1, ]
y_test1 <- y_data[-training_index1, ]
```

```
# TO DO: Print the number of samples in each set and verify the data dimensions
cat("number of test samples:", nrow(x_test1), "\n")
cat("number of training samples:", nrow(x_train1))
```

```
↗ number of test samples: 80
number of training samples: 121
```

```
# Kiểm tra kích thước các tập dữ liệu
dim(x_train1)
dim(x_test1)
```

```
↗ 121 30
80 30
```

Use `lm()` to fit a linear regression model predicting price from **horsepower** using the training set **x_train1**.

```
# TO DO: Fit a linear regression model using lm() in R
lre <- lm(y_train ~ horsepower, data = x_train)
```

```
# Hiển thị tóm tắt mô hình
summary(lre)
```

```
↗ Call:
lm(formula = y_train ~ horsepower, data = x_train)

Residuals:
    Min       1Q   Median       3Q      Max
-9606.3 -2149.7 -399.6  1808.0 17691.3

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -4989.523   1007.615  -4.952 1.69e-06 ***
horsepower    177.708     9.197  19.323 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4646 on 179 degrees of freedom
Multiple R-squared:  0.6759,    Adjusted R-squared:  0.6741
```

Make Predictions for Testing Data:

- Predict the values of **y_test1** based on the trained model.
- Compute the R-squared value by comparing **y_test1** and **y_pred**.

```
# TO DO: Predict values for the testing set and calculate R^2
# Predict the values of y_train based on the trained model
y_pred <- predict(lre, newdata = x_test)
# Calculate R^2 for the training data
SSE <- sum((y_test - y_pred)^2) # Calculate the sum of squared errors (SSE) for the training data
SST <- sum((y_test - mean(y_test))^2) # Calculate the total sum of squares (SST) for the training data
R_squared <- 1 - (SSE/SST)

# Display the R^2 value for the training data
print(paste("R^2:", R_squared))
```

```
[1] "R^2: 0.27149977469876"
```

Make Predictions for Training Data:

- Predict the values of **y_train1** based on the trained model.
- Compute the **R-squared** value for the training set.

```
# TO DO: Predict values for the training set and calculate R^2 for the training data
# Predict y_train values using the trained model
y_train_pred <- predict(lre, newdata = x_train)
# Calculate R-squared for the training set
SSE_train <- sum((y_train - y_train_pred)^2) # Calculate the sum of squared errors (SSE) for the training set
SST_train <- sum((y_train - mean(y_train))^2) # Calculate the total sum of squares (SST) for the training set
R_squared_train <- 1 - (SSE_train / SST_train)
# Hiển thị giá trị R^2 cho dữ liệu huấn luyện
print(paste("R^2 (Training Data):", R_squared_train))
```

```
[1] "R^2 (Training Data): 0.675941422146126"
```

Questions:

1. What is the significance of using a different split ratio (60% for training)?

Answer: Splitting the data into a 60% training set and a 40% testing set allows for a more accurate evaluation of the model's performance, as it uses more data to train the model and thus gives it a better chance of learning the underlying patterns in the data.

2. How does horsepower serve as a predictor for price in the linear regression model?

Answer: In the linear regression model, horsepower is a predictor for price. This means that the model attempts to find a linear relationship between horsepower and price, such that higher horsepower values are associated with higher price values.

3. What does the R-squared value indicate about the model's performance?

Answer: The R-squared value, also known as the coefficient of determination, indicates the proportion of the variance in the dependent variable (price) that can be explained by the independent variable (horsepower). A higher R-squared value indicates a better fit of the model to the data.

4. Why is it necessary to calculate R-squared for both the training and testing sets?

Answer: Calculating R-squared for both the training and testing sets is necessary to assess the model's ability to generalize to new data. A model that fits the training data well but performs poorly on the testing data is considered overfit.

✓ Section 3: Cross-Validation and Model Evaluation for Linear Regression (60-40 Split)

In this section, you will:

1. Split the data into training and testing sets using a 60% training and 40% testing ratio.
2. Create a linear regression model to predict price using horsepower as the predictor.
3. Evaluate the model using R-squared on the testing set.
4. Implement cross-validation with 4 folds to assess model performance using R-squared and Mean Squared Error (MSE).

```
library(dplyr)
```

```
Attaching package: 'dplyr'
```

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
# Thiết lập random seed
set.seed(123)

# TO DO: Use createDataPartition to generate indices for splitting the data (60% for training)
training_index1 <- createDataPartition(y_data$price, p = 0.6, list = FALSE)

# TO DO: Split the data into training and testing sets
x_train1 <- x_data[training_index1, ]
x_test1 <- x_data[-training_index1, ]
y_train1 <- y_data[training_index1, ]
y_test1 <- y_data[-training_index1, ]

# TO DO: Fit a linear regression model using lm() in R
lre <- lm(price ~ horsepower, data = data.frame(x_train1, price = y_train1))

# TO DO: Predict values for the testing set and calculate R^2
y_pred1 <- predict(lre, newdata = x_test1)

# Calculate R-squared for the test set
SSE_test1 <- sum((y_test1 - y_pred1)^2) # Calculate the sum of squared errors (SSE) for the test set
SST_test1 <- sum((y_test1 - mean(y_test1))^2) # Calculate the total sum of squares (SST) for the test set
R_squared_test1 <- 1 - (SSE_test1 / SST_test1)

#print R^2 for testing set
cat("R^2 (Test Data):", R_squared_test1, "\n")
```

 R^2 (Test Data): 0.5537754

Sometimes you do not have sufficient testing data; as a result, you may want to perform cross-validation. Let's go over several methods that you can use for cross-validation.

Cross-Validation Score

Let's import **model_selection** from the module **cross_val_score**.

Create a Data Frame with Only the 'horsepower' Variable:

Extract the horsepower variable to create x_data_horsepower with 4 folds

```
# TO DO: Create a data frame containing only the 'horsepower' variable
x_data_horsepower <- data.frame(horsepower = x_data$horsepower)

# TO DO: Set up cross-validation control with 4 folds
ctrl <- trainControl(method = "cv", number = 4)

# TO DO: Perform cross-validation and retrieve R-squared values
set.seed(123) # To make results reproducible
lm_model <- train(x = x_data_horsepower, # Use the generated x_data_horsepower
                  y = y_data$price,
                  method = "lm",
                  trControl = ctrl)
#retrieve R-squared values
Rcross <- lm_model$resample$Rsquared
print(Rcross)
```

```
[1] 0.6773254 0.7456737 0.5443686 0.6269395
```

The default scoring is R^2 . Each element in the array has the average R^2 value for the fold:

We can calculate the average and standard deviation of our estimate:

Calculate Mean and Standard Deviation of R-squared:

- Compute and print the mean and standard deviation of the R-squared values from cross-validation.

```
# TO DO: Calculate and display mean and standard deviation of R-squared values
cat("The mean of the folds is", mean(Rcross), "and the standard deviation is", sd(Rcross), "\n")
```

```
The mean of the folds is 0.6485768 and the standard deviation is 0.08481708
```

Retrieve and Calculate MSE from Cross-Validation:

- Obtain the RMSE values from the cross-validation results, square them to get MSE.
- Multiply the MSE by -1 to get the negative MSE and calculate the mean and standard deviation.

```
# TO DO: Retrieve and calculate MSE from cross-validation, then calculate mean and standard deviation
# Get the MSE result (square of RMSE)
mse_results <- lm_model$resample$RMSE^2
# Multiply by -1 to get negative MSE
neg_mse_results <- -1 * mse_results
# Print the result
print(neg_mse_results)
# Calculate and print the mean and standard deviation of the negative MSE
cat("The mean of the negative MSE is", mean(neg_mse_results),
    "and the standard deviation is", sd(neg_mse_results), "\n")
```

```
[1] -21675218 -19348398 -24144482 -21900673
The mean of the negative MSE is -21767193 and the standard deviation is 1960302
```

Questions:

1. What is the purpose of cross-validation in evaluating a linear regression model?

Answer: Cross-validation is a technique used to evaluate the performance of a model by splitting the data into multiple folds and using each fold as a test set while the remaining folds are used as training sets. This helps to reduce the risk of overfitting and provide a more robust evaluation of the model's predictive capabilities.

2. How does the horsepower variable impact the prediction of price in the linear regression model?

Answer: The horsepower variable influences the prediction of price in the linear regression model by establishing a linear relationship between the two variables. The model aims to find a pattern where higher horsepower values are linked to higher price values.

3. Why is it important to calculate both the R-squared value and Mean Squared Error (MSE)?

Answer: R-squared and Mean Squared Error (MSE) are important metrics to assess the model's performance. R-squared measures the proportion of variance in the target variable that is explained by the model, whereas MSE indicates the average squared difference between the actual and predicted values. Both metrics are crucial for understanding different aspects of model performance.

4. What does the negative MSE indicate in this context, and why is it calculated?

Answer: A negative MSE is not a standard result and indicates an issue with the model. It usually suggests that the model is not capturing the relationship between the predictor and the target variable correctly. This might be because the model is overfitting and making predictions that are far off from the actual values.

✓ Section Title: Calculating and Analyzing Cross-Validation Results

In this section, you will:

1. Extract and analyze the R-squared values obtained from cross-validation.
2. Calculate the mean R-squared value to evaluate the overall performance of the model.
3. Implement a linear regression model using 4-fold cross-validation.
4. Predict values for the dataset using the trained model and examine a subset of the predictions.

Extract R-squared Values from Cross-Validation:

- Retrieve the R-squared values from the cross-validation results using `lm_model$resample$Rsquared`.

```
# TO DO: Extract R-squared values from cross-validation results
Rcross <- lm_model$resample$Rsquared
# TO DO: Calculate the mean of the R-squared values
mean_Rcross <- mean(Rcross)
print(mean_Rcross)
```

➦ [1] 0.6485768

We input the object, the feature "**horsepower**", and the target data **y_data**. The parameter 'cv' determines the number of folds. In this case, it is 4.

Set Up Cross-Validation Control:

- Use `trainControl()` to configure a 4-fold cross-validation process
- Use `train()` to conduct cross-validation and train a linear regression model using the horsepower variable to predict price.

```
# TO DO: Set up cross-validation control with 4 folds
ctrl <- trainControl(method = "cv", number = 4)

# TO DO: Train a linear regression model using cross-validation
set.seed(123) # To make results reproducible
lm_model <- train(x = x_data_horsepower, # Use the generated x_data_horsepower
                 y = y_data$price,
                 method = "lm",
                 trControl = ctrl)

# TO DO: Use the trained model to predict values for the dataset
yhat <- predict(lm_model, newdata = x_data_horsepower)

# TO DO: Display the first 5 predicted values
yhat[1:5]
```

➦ 1 14544 7688344373 14544 7688344373 84848 8484768844 14544 7688344373 14544 7688344373 14544 7688344373

Questions:

1. What does the mean R-squared value indicate in this context?

Answer: The mean R-squared value across multiple folds in cross-validation represents the average predictive power of the model. It gives a more robust estimate of how well the model generalizes to unseen data, as it accounts for variability in the model's performance across different data subsets.

2. Why is cross-validation used, and how does it help improve model reliability?

Answer: Cross-validation is used to evaluate the model's performance more reliably, particularly when dealing with limited data. By splitting the data into multiple folds and using each fold as a test set while training on the remaining folds, cross-validation reduces the risk of overfitting. This helps to ensure that the model's performance on unseen data is not biased by the specific random selection of the training and testing sets.

3. How does setting a random seed (set.seed) affect the results?

Answer: Setting a random seed ensures that the random number generator (RNG) produces the same sequence of random numbers every time the code is run. This is critical for reproducibility, allowing you to get the same results if you rerun the code. In cross-validation, setting a seed ensures that the data is split consistently into the same folds each time.

4. Why is it important to examine the predicted values after training the model?

Answer: Examining the predicted values after training the model is crucial for several reasons:

- **Understanding Model Behavior:** It helps visualize how the model is responding to the data, identifying patterns and potential issues like overfitting or underfitting.
- **Assessing Model Accuracy:** By comparing the predicted values to the actual values, you can get a visual sense of how well the model is performing and make informed decisions about whether to adjust the model or explore different approaches.
- **Identifying Outliers or Unusual Data:** Examining the predictions can help spot outliers or data points that the model is struggling to predict accurately. This might lead you to investigate these data points further and potentially correct errors or improve the model's handling of such cases.
- **Confidence Building:** Visualizing the predictions gives you a better understanding of the model's strengths and weaknesses, building confidence in its accuracy and performance.

✓ Part 2: Overfitting, Underfitting, and Model Selection

This section aims to explore the concepts of overfitting, underfitting, and model selection by implementing and evaluating linear and polynomial regression models. The goal is to analyze how different model complexities affect the prediction performance, determine the best polynomial degree for fitting the data, and visualize the results.

Data Preparation: Use a dataset containing automobile data, where the target variable (price) is predicted based on features like horsepower, curb.weight, engine.size, and highway.mpg. The dataset should be split into training and testing sets. Utilize a 55% subset of the training data for training a more complex model and the remaining 45% for evaluation.

Let's create Multiple Linear Regression objects and train the model using **'horsepower'**, **'curb-weight'**, **'engine-size'** and **'highway-mpg'** as features.

Linear Regression with Multiple Predictors:

1. Create a multiple linear regression model using the predictors horsepower, curb.weight, engine.size, and highway.mpg.
2. Display the summary of the model.
3. Make predictions using the training and test datasets.
4. Calculate metrics such as Root Mean Square Error (RMSE) and R-squared (R^2) for training and test data.
5. Plot a comparison between predicted and actual values for the test set using a scatter plot with a reference line.

Model Evaluation - Data Subset (55% Training Data):

6. Split the initial training set into two parts: 55% for training and 45% for evaluation.
7. Train a linear regression model on the 55% training data.
8. Evaluate the model performance on the remaining 45% and the initial test set.
9. Calculate and display RMSE and R^2 values for both datasets.

Polynomial Regression - Degree 3:

10. Create polynomial features of degree 3 for the horsepower variable.
11. Train a polynomial regression model with degree 3 using the 55% training data.

12. Make predictions for the remaining 45% and the test set.

13. Evaluate the model with RMSE and R^2 metrics.

Polynomial Regression - Degree 5:

14. Create polynomial features of degree 5 for the horsepower variable.

15. Train a polynomial regression model with degree 5 using the 55% training data.

16. Make predictions for the remaining 45% and the test set.

17. Evaluate the model with RMSE and R^2 metrics.

Polynomial Regression - Degree 11:

18. Create polynomial features of degree 11 for the horsepower variable.

19. Train a polynomial regression model with degree 11 using the 55% training data.

20. Make predictions for the remaining 45% and the test set.

21. Evaluate the model with RMSE and R^2 metrics.

Model Complexity Analysis - Varying Polynomial Degrees:

22. Train polynomial regression models for degrees ranging from 1 to 11.

23. Calculate the R^2 values for each model.

24. Plot a line graph showing R^2 values for each polynomial degree to visualize how complexity affects performance.

Visualization Requirements:

- Scatter Plot for the comparison of predicted vs. actual values on the test set.
- Distribution Plots for actual vs. predicted prices for each model on the training, remaining training (45%), and test datasets.
- Line Plot showing the R^2 values for polynomial degrees from 1 to 11

✓ Section 1: Linear Regression Model - Training, Evaluation, and Visualization

In this section, you will:

- Construct a linear regression model using multiple predictors.
- Evaluate the model's performance using both training and testing datasets.
- Visualize the model's predictions compared to actual values and analyze their distribution.

Tasks:

- Construct a Linear Regression Model:
 - Use multiple predictors (horsepower, curb.weight, engine.size, and highway.mpg) to predict price.
 - Implement the model using the `lm()` function.

```
# TO DO: Create a linear regression model with multiple predictors
lre_multi <- lm(price ~ horsepower + curb.weight + engine.size + highway.mpg, data = data.frame(x_train, price = y_train))

# TO DO: Display the summary of the regression model
summary(lre_multi)
```



```
Call:
lm(formula = price ~ horsepower + curb.weight + engine.size +
    highway.mpg, data = data.frame(x_train, price = y_train))
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-9296.6 -1702.5    9.8   1266.9 13158.7
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -16827.622   4702.107  -3.579 0.000446 ***
horsepower    62.443     15.778    3.958 0.000110 ***
curb.weight    4.753      1.198    3.967 0.000106 ***
engine.size   78.146     14.624    5.344 2.79e-07 ***
highway.mpg   55.466     78.565    0.706 0.481125
---
```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
Residual standard error: 3533 on 176 degrees of freedom
Multiple R-squared:  0.8158,    Adjusted R-squared:  0.8116
```

- Predict price values for both the training and test datasets using the trained model.
- Print the first 5 predicted values for both sets to get a sense of the prediction.

```
# TO DO: Predict values for training data and display the first 5 predictions
y_train_pred <- predict(lre_multi, newdata = x_train)
#display the first 5 predictions
y_train_pred[1:5]
```



```
# TO DO: Predict values for test data and display the first 5 predictions
y_test_pred <- predict(lre_multi, newdata = x_test)
y_test_pred[1:5]
```



Calculate the Root Mean Squared Error (RMSE) and R-squared values for both training and test datasets to evaluate the model's performance.

```
library(ggplot2)
```

Let's examine the distribution of the predicted values of the training data.

```
# TO DO: Evaluate the model's performance using RMSE and R-squared for training data
train_rmse <- sqrt(mean((y_train - y_train_pred)^2))
train_r2 <- summary(lre_multi)$r.squared
```

```
# TO DO: Evaluate the model's performance using RMSE and R-squared for test data
test_rmse <- sqrt(mean((y_test - y_test_pred)^2))
test_r2 <- 1 - sum((y_test - y_test_pred)^2) / sum((y_test - mean(y_test))^2)
```

```
# TO DO: Print evaluation metrics for training and test datasets
cat("Training Data:\n")
cat("RMSE:", train_rmse, "\n")
cat("R-squared:", train_r2, "\n")
```

```
cat("\nTest Data:\n")
cat("RMSE:", test_rmse, "\n")
cat("R-squared:", test_r2, "\n")
```

```
# TO DO: Visualize comparison between predicted and actual values using ggplot2
library(ggplot2)
```

```
ggplot() +
  geom_point(aes(x = y_test, y = y_test_pred), color = 'blue') +
  geom_abline(slope = 1, intercept = 0, linetype = 'dashed', color = 'red') +
  labs(x = 'Actual Price', y = 'Predicted Price',
       title = 'Comparison between Predicted and Actual Values') +
```

```
theme_minimal()

# TO DO: Define a custom function to plot distribution of actual vs predicted values
# Distribution plot function
distribution_plot <- function(actual, predicted, dataset_type) {
  ggplot() +
    geom_density(aes(x = actual, color = "Actual"), size = 1.2) +
    geom_density(aes(x = predicted, color = "Predicted"), size = 1.2, linetype = "dashed") +
    labs(title = paste("Distribution of Actual vs Predicted -", dataset_type),
         x = "Price", y = "Density") +
    scale_color_manual(name = "Legend", values = c("Actual" = "blue", "Predicted" = "red")) +
    theme_minimal()
}

# TO DO: Plot the distribution for training data
distribution_plot(y_train, y_train_pred, "Training Data")

# TO DO: Plot the distribution for test data
distribution_plot(y_test, y_test_pred, "Test Data")
```

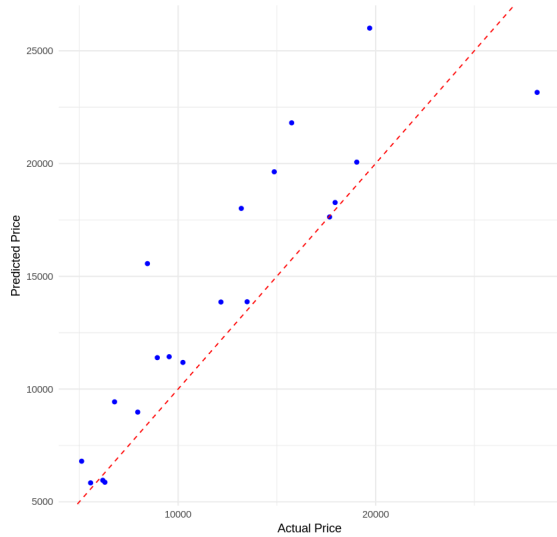


Training Data:
RMSE: 3483.854
R-squared: 0.815763

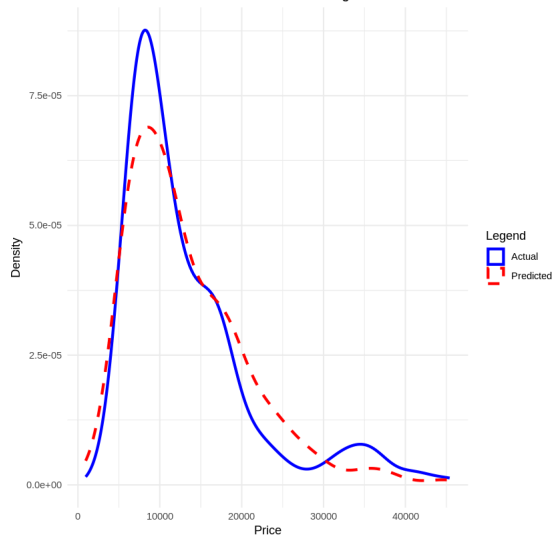
Test Data:
RMSE: 3345.171
R-squared: 0.6761707

Warning message:
"Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
Please use `linewidth` instead."

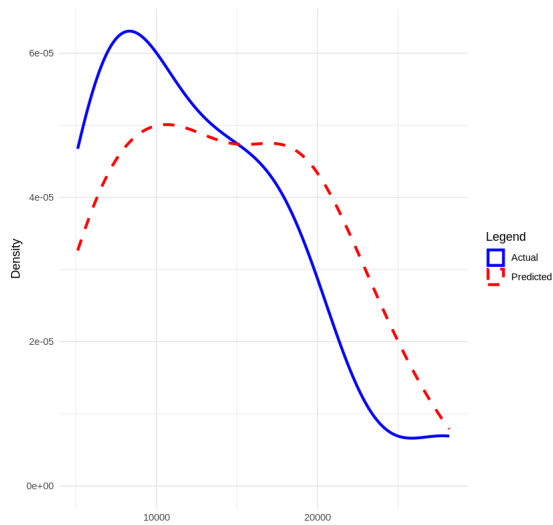
Comparison between Predicted and Actual Values



Distribution of Actual vs Predicted - Training Data



Distribution of Actual vs Predicted - Test Data



✓ Questions:

1. How does the R-squared value help in understanding the model's performance?

Answer: R-squared measures the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (predictors). A higher R-squared indicates a model that explains more variance, meaning it fits the data better.

2. What does the RMSE value signify in this context?

Answer: Root Mean Square Error (RMSE) measures the average magnitude of the prediction errors. It is the square root of the average of squared differences between predicted and observed values. Lower RMSE values indicate a model that fits the data more closely, as it signifies smaller prediction errors.

3. Why is it important to visualize the distribution of actual vs predicted values?

Answer: Visualizing actual vs. predicted values allows us to identify patterns, trends, and potential biases in the model's predictions. It helps to see if predictions are consistently overestimating or underestimating across different ranges, thus informing about model accuracy and possible improvements.

Overfitting


Overfitting occurs when the model fits the noise, but not the underlying process. Therefore, when testing your model using the test set, your model does not perform as well since it is modelling noise, not the underlying process that generated the relationship. Let's create a degree 5 polynomial model.

✓ Section 2: Linear Regression Model - Data Subset Training and Evaluation

In this section, you will:

- Use a subset of the original training data (55%) to create a linear regression model.
- Evaluate the model's performance on the remaining 45% of the training data and the original test dataset.
- Visualize the predicted results compared to actual values for analysis.

```
install.packages("polycor")
```

 Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependencies 'mvtnorm', 'admisc'

Use 55% of the initial training set to train the model, leaving the remaining 45% to validate the model.

```
# TO DO: Split 55% of the original training data for model training
set.seed(123)
train_index_55 <- sample(1:nrow(x_train), size = 0.55 * nrow(x_train))

x_train_55 <- x_train[train_index_55, ]
y_train_55 <- y_train[train_index_55]

# TO DO: Store the remaining 45% for validation
x_train_remain <- x_train[-train_index_55, ]
y_train_remain <- y_train[-train_index_55]

# TO DO: Create a linear regression model using 55% of the training data
lre_multi_55 <- lm(price ~ horsepower + curb.weight + engine.size + highway.mpg, data = data.frame(x_train_55, price = y_train_55))

# TO DO: Display the summary of the regression model trained on 55% data
summary(lre_multi_55)

# TO DO: Predict values for the remaining 45% of the training data
y_train_remain_pred <- predict(lre_multi_55, newdata = x_train_remain)

# TO DO: Predict values for the original test data
y_test_pred_55 <- predict(lre_multi_55, newdata = x_test)
```

```
# TO DO: Evaluate the model's performance using RMSE and R-squared for remaining training data (45%)
train_remain_rmse <- sqrt(mean((y_train_remain - y_train_remain_pred)^2))
train_remain_r2 <- 1 - sum((y_train_remain - y_train_remain_pred)^2) / sum((y_train_remain - mean(y_train_remain))^2)

# TO DO: Evaluate the model's performance using RMSE and R-squared for the test data
test_rmse_55 <- sqrt(mean((y_test - y_test_pred_55)^2))
test_r2_55 <- 1 - sum((y_test - y_test_pred_55)^2) / sum((y_test - mean(y_test))^2)

# TO DO: Print evaluation metrics for remaining training and test datasets
cat("Remaining Training Data (45%):\n")
cat("RMSE:", train_remain_rmse, "\n")
cat("R-squared:", train_remain_r2, "\n")

cat("\nTest Data:\n")
cat("RMSE:", test_rmse_55, "\n")
cat("R-squared:", test_r2_55, "\n")

# TO DO: Plot the distribution for the remaining 45% of the training data
distribution_plot(y_train_remain, y_train_remain_pred, "Remaining Training Data (45%)")

# TO DO: Plot the distribution for the test data
distribution_plot(y_test, y_test_pred_55, "Test Data")
```




Call:

```
lm(formula = price ~ horsepower + curb.weight + engine.size +
    highway.mpg, data = data.frame(x_train_55, price = y_train_55))
```

Residuals:

Min	1Q	Median	3Q	Max
-8370.2	-1855.6	142.8	1462.6	11679.1

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-12550.540	7146.396	-1.756	0.08231 .
horsepower	73.915	23.387	3.161	0.00212 **
curb.weight	4.367	1.722	2.536	0.01287 *
engine.size	52.732	21.945	2.403	0.01823 *
highway.mpg	5.681	124.617	0.046	0.96374

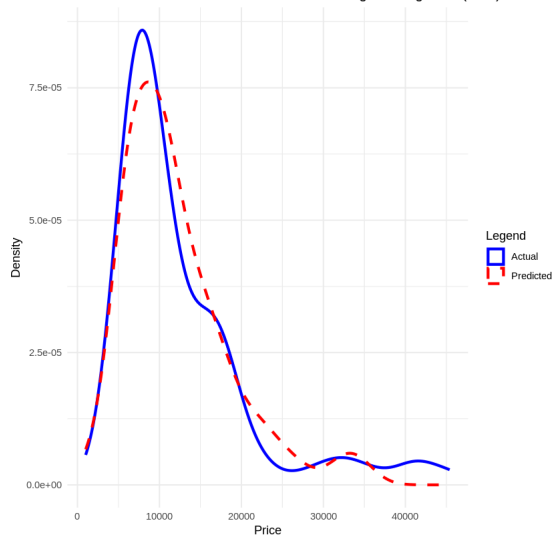
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3690 on 94 degrees of freedom
 Multiple R-squared: 0.782, Adjusted R-squared: 0.7727
 F-statistic: 84.3 on 4 and 94 DF, p-value: < 2.2e-16
 Remaining Training Data (45%):
 RMSE: 3559.781
 R-squared: 0.8265403

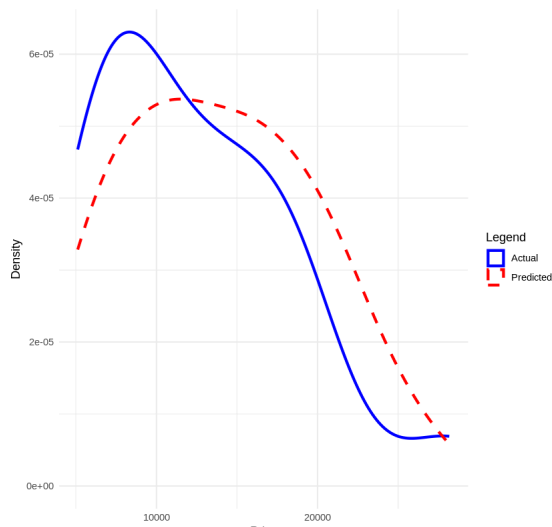
Test Data:

RMSE: 3242.91
 R-squared: 0.6956667

Distribution of Actual vs Predicted - Remaining Training Data (45%)



Distribution of Actual vs Predicted - Test Data



Questions:

1. What does training the model on only 55% of the data demonstrate?

Answer: Training on a subset like 55% demonstrates the model's ability to learn from a smaller dataset. It can indicate how well the model generalizes and avoids overfitting, as it is later evaluated on the remaining 45% and other data sets.

2. How can RMSE and R-squared metrics change when the training data is limited?

Answer: When training data is limited, the model may struggle to capture complex patterns, leading to higher RMSE (larger errors) and lower R-squared values. Limited data often reduces model reliability, as it may not represent the overall variability in the dataset.

3. Why is it important to evaluate the model on different subsets of data?

Answer: Evaluating on different data subsets helps assess model robustness and generalizability. It shows how well the model can predict unseen data, indicating its performance in real-world scenarios where data differs from the training set.

Section 3: Polynomial Regression - Degree 3 Model Training and Evaluation

In this session, you will:

1. Create polynomial features (degree 3) for the horsepower variable.
2. Use these polynomial features to build a polynomial regression model.
3. Evaluate the model's performance on the remaining 45% of the training data and the original test dataset.
4. Visualize the predicted results compared to actual values for analysis.

```
# TO DO: Load the libraries
library(dplyr)
library(ggplot2)

# TO DO: Generate cubic polynomial features for horsepower
x_train_55_poly <- x_train_55 %>%
  mutate(horsepower_poly = horsepower^3)

x_train_remain_poly <- x_train_remain %>%
  mutate(horsepower_poly = horsepower^3)

x_test_poly <- x_test %>%
  mutate(horsepower_poly = horsepower^3)

# TO DO: Train a polynomial regression model using cubic features
poly_model <- lm(price ~ horsepower_poly + curb.weight + engine.size + highway.mpg,
  data = data.frame(x_train_55_poly, price = y_train_55))

# TO DO: Display the summary of the polynomial regression model
summary(poly_model)

# TO DO: Predict values for the remaining 45% of the training data using the polynomial model
y_train_remain_poly_pred <- predict(poly_model, newdata = x_train_remain_poly)

# TO DO: Predict values for the original test data using the polynomial model
y_test_poly_pred <- predict(poly_model, newdata = x_test_poly)

# TO DO: Evaluate the polynomial model using RMSE and R-squared for remaining training data (45%)
train_remain_poly_rmse <- sqrt(mean((y_train_remain - y_train_remain_poly_pred)^2))
train_remain_poly_r2 <- 1 - sum((y_train_remain - y_train_remain_poly_pred)^2) / sum((y_train_remain - mean(y_train_remain))^2)

# TO DO: Evaluate the polynomial model using RMSE and R-squared for the test data
test_poly_rmse <- sqrt(mean((y_test - y_test_poly_pred)^2))
test_poly_r2 <- 1 - sum((y_test - y_test_poly_pred)^2) / sum((y_test - mean(y_test))^2)

# TO DO: Print evaluation metrics for remaining training and test datasets
cat("Remaining Training Data (45%) with Polynomial Model:\n")
cat("RMSE:", train_remain_poly_rmse, "\n")
cat("R-squared:", train_remain_poly_r2, "\n")

cat("\nTest Data with Polynomial Model:\n")
cat("RMSE:", test_poly_rmse, "\n")
cat("R-squared:", test_poly_r2, "\n")

# TO DO: Plot the distribution for the remaining 45% of the training data using the polynomial model
distribution_plot(y_train_remain, y_train_remain_poly_pred, "Remaining Training Data (45%) - Polynomial Model")

# TO DO: Plot the distribution for the test data using the polynomial model
distribution_plot(y_test, y_test_poly_pred, "Test Data - Polynomial Model")
```



Call:

```
lm(formula = price ~ horsepower_poly + curb.weight + engine.size +
    highway.mpg, data = data.frame(x_train_55_poly, price = y_train_55))
```

Residuals:

Min	1Q	Median	3Q	Max
-10150.1	-1947.7	59.1	1354.9	12156.7

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-2.060e+03	5.930e+03	-0.347	0.72903
horsepower_poly	8.882e-04	2.892e-04	3.071	0.00279 **
curb.weight	4.742e+00	1.766e+00	2.686	0.00856 **
engine.size	4.935e+01	2.302e+01	2.144	0.03464 *
highway.mpg	-1.519e+02	1.016e+02	-1.495	0.13819

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

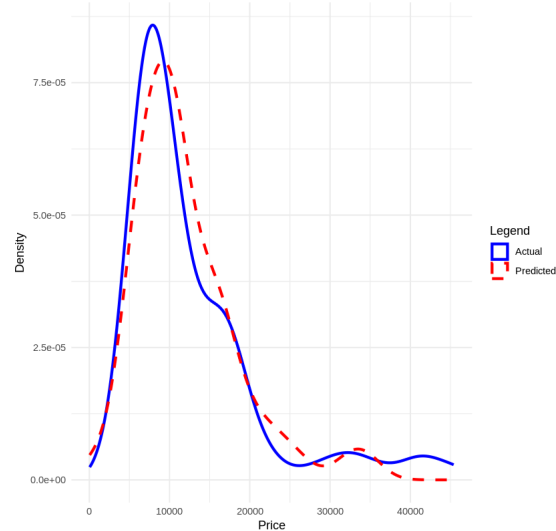
Residual standard error: 3700 on 94 degrees of freedom
 Multiple R-squared: 0.7808, Adjusted R-squared: 0.7715
 F-statistic: 83.73 on 4 and 94 DF, p-value: < 2.2e-16
 Remaining Training Data (45%) with Polynomial Model:
 RMSE: 3573.54
 R-squared: 0.8251969

Test Data with Polynomial Model:

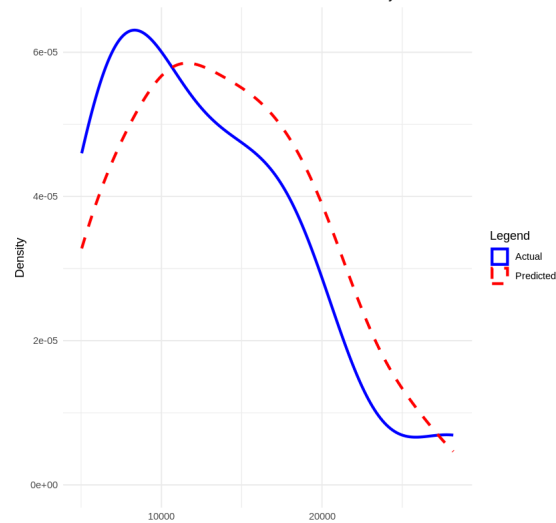
RMSE: 3064.006

R-squared: 0.7283193

Distribution of Actual vs Predicted - Remaining Training Data (45%) - Polynon



Distribution of Actual vs Predicted - Test Data - Polynomial Model



Questions:

1. What is the purpose of using polynomial features for regression?

Answer: Polynomial features enable the model to capture non-linear relationships between the features and the target variable. By introducing polynomial terms, the model can better fit complex data patterns that a linear model might miss.

2. How does the cubic transformation of the horsepower variable impact model predictions?

Answer: A cubic transformation allows the model to fit a curve that accounts for potential non-linear effects of horsepower on the price, enhancing prediction accuracy for data where the relationship is not strictly linear.

3. Why is it important to compare polynomial models with simpler linear models?

Answer: Comparing polynomial and linear models helps identify the simplest model that provides a good fit. While polynomial models can capture complexity, they risk overfitting. Testing both helps find a balance between accuracy and generalizability.

```
install.packages("polycor")
```

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Section 4: Comparing Polynomial Regression Models - Degree 5 vs. Degree 11

In this session, you will:

1. Create polynomial features for the horsepower variable using degrees 5 and 11.
2. Train two polynomial regression models with these features.
3. Evaluate the performance of the models on the remaining 45% of the training data and the test dataset.
4. Compare the models to understand the impact of polynomial degree on prediction accuracy.
5. Visualize the results to analyze and compare the distribution of actual vs. predicted values.

```
# Create 5th degree polynomial variables for horsepower
library(dplyr)

# Add polynomial variables for horsepower
x_train_55_poly <- x_train_55 %>%
  mutate(horsepower_poly = poly(horsepower, degree = 5, raw = TRUE))

x_train_remain_poly <- x_train_remain %>%
  mutate(horsepower_poly = poly(horsepower, degree = 5, raw = TRUE))

x_test_poly <- x_test %>%
  mutate(horsepower_poly = poly(horsepower, degree = 5, raw = TRUE))

# Create a 5th degree polynomial linear regression model
lre_poly_5 <- lm(price ~ horsepower_poly, data = data.frame(x_train_55_poly, price = y_train_55))

# Show model summary
summary(lre_poly_5)

# Predict with the remaining 45% of training data
y_train_remain_poly_pred <- predict(lre_poly_5, newdata = x_train_remain_poly)

# Prediction with test data
y_test_poly_pred <- predict(lre_poly_5, newdata = x_test_poly)

# Evaluate the model on the remaining 45% of the training data
train_remain_poly_rmse <- sqrt(mean((y_train_remain - y_train_remain_poly_pred)^2))
train_remain_poly_r2 <- 1 - sum((y_train_remain - y_train_remain_poly_pred)^2) / sum((y_train_remain - mean(y_train_remain))^2)

# Evaluate the model on test data
test_poly_rmse <- sqrt(mean((y_test - y_test_poly_pred)^2))
test_poly_r2 <- 1 - sum((y_test - y_test_poly_pred)^2) / sum((y_test - mean(y_test))^2)

# In các chỉ số đánh giá
cat("Remaining Training Data (45% - Polynomial Degree 5):\n")
cat("RMSE:", train_remain_poly_rmse, "\n")
```

```
cat("R-squared:", train_remain_poly_r2, "\n")

cat("\nTest Data (Polynomial Degree 5):\n")
cat("RMSE:", test_poly_rmse, "\n")
cat("R-squared:", test_poly_r2, "\n")

# Draw DistributionPlot for remaining training data (45%) with polynomial model
distribution_plot(y_train_remain, y_train_remain_poly_pred, "Remaining Training Data (45% - Polynomial Degree 5)")

# Draw DistributionPlot for test data with polynomial model
distribution_plot(y_test, y_test_poly_pred, "Test Data (Polynomial Degree 5)")
```



Call:

```
lm(formula = price ~ horsepower_poly, data = data.frame(x_train_55_poly,
  price = y_train_55))
```

Residuals:

	Min	1Q	Median	3Q	Max
	-8586.2	-2046.7	-383.6	1401.0	14575.8

Coefficients:

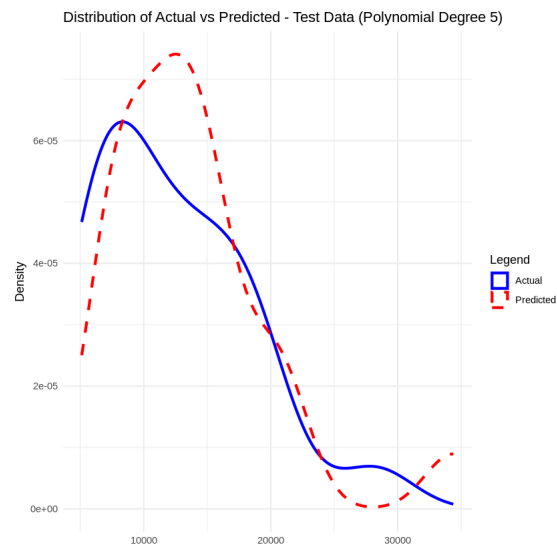
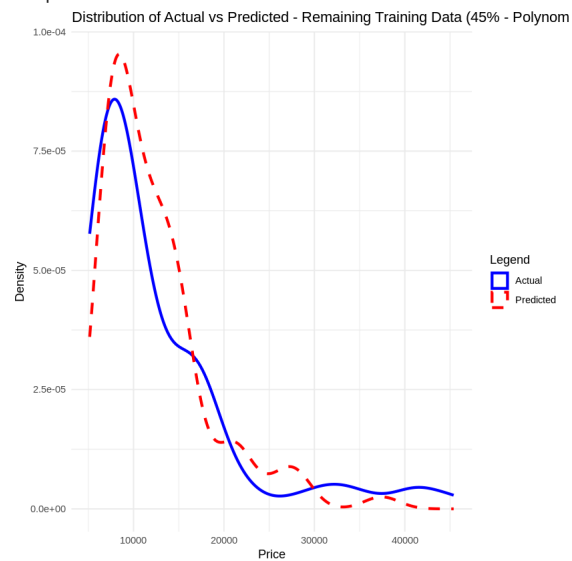
	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.010e+05	5.716e+04	1.767	0.0806 .
horsepower_poly1	-4.348e+03	2.375e+03	-1.831	0.0703 .
horsepower_poly2	7.412e+01	3.733e+01	1.986	0.0500 .
horsepower_poly3	-5.793e-01	2.778e-01	-2.086	0.0397 *
horsepower_poly4	2.146e-03	9.799e-04	2.190	0.0310 *
horsepower_poly5	-3.003e-06	1.313e-06	-2.287	0.0245 *

 Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4180 on 93 degrees of freedom
 Multiple R-squared: 0.7233, Adjusted R-squared: 0.7084
 F-statistic: 48.62 on 5 and 93 DF, p-value: < 2.2e-16
 Remaining Training Data (45% - Polynomial Degree 5):
 RMSE: 4872.838
 R-squared: 0.6749757

Test Data (Polynomial Degree 5):

RMSE: 5275.235
 R-squared: 0.1946898



```
# Create 11th degree polynomial variables for horsepower
x_train_55_poly_11 <- x_train_55 %>%
  mutate(horsepower_poly_11 = poly(horsepower, degree = 11, raw = TRUE))
```

```
x_train_remain_poly_11 <- x_train_remain %>%
  mutate(horsepower_poly_11 = poly(horsepower, degree = 11, raw = TRUE))

x_test_poly_11 <- x_test %>%
  mutate(horsepower_poly_11 = poly(horsepower, degree = 11, raw = TRUE))

# Create a polynomial linear regression model of degree 11
lre_poly_11 <- lm(price ~ horsepower_poly_11, data = data.frame(x_train_55_poly_11, price = y_train_55))

# Show model summary
summary(lre_poly_11)

# Predict with the remaining 45% of training data
y_train_remain_poly_11_pred <- predict(lre_poly_11, newdata = x_train_remain_poly_11)

# Prediction with test data
y_test_poly_11_pred <- predict(lre_poly_11, newdata = x_test_poly_11)

# Evaluate the model on the remaining 45% of the training data
train_remain_poly_11_rmse <- sqrt(mean((y_train_remain - y_train_remain_poly_11_pred)^2))
train_remain_poly_11_r2 <- 1 - sum((y_train_remain - y_train_remain_poly_11_pred)^2) / sum((y_train_remain - mean(y_train_remain))^2)

# Evaluate the model on test data
test_poly_11_rmse <- sqrt(mean((y_test - y_test_poly_11_pred)^2))
test_poly_11_r2 <- 1 - sum((y_test - y_test_poly_11_pred)^2) / sum((y_test - mean(y_test))^2)

# Print evaluation indexes
cat("Remaining Training Data (45% - Polynomial Degree 11):\n")
cat("RMSE:", train_remain_poly_11_rmse, "\n")
cat("R-squared:", train_remain_poly_11_r2, "\n")

cat("\nTest Data (Polynomial Degree 11):\n")
cat("RMSE:", test_poly_11_rmse, "\n")
cat("R-squared:", test_poly_11_r2, "\n")

# Draw DistributionPlot for remaining training data (45%) with 11th degree polynomial model
distribution_plot(y_train_remain, y_train_remain_poly_11_pred, "Remaining Training Data (45% - Polynomial Degree 11)")

# Draw DistributionPlot for test data with 11th degree polynomial model
distribution_plot(y_test, y_test_poly_11_pred, "Test Data (Polynomial Degree 11)")
```



Call:

```
lm(formula = price ~ horsepower_poly_11, data = data.frame(x_train_55_poly_11,
  price = y_train_55))
```

Residuals:

	Min	1Q	Median	3Q	Max
	-8947.1	-1637.1	-332.7	1230.0	16067.8

Coefficients: (1 not defined because of singularities)

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	7.087e+06	2.012e+07	0.352	0.725
horsepower_poly_111	-6.701e+05	1.854e+06	-0.361	0.719
horsepower_poly_112	2.775e+04	7.462e+04	0.372	0.711
horsepower_poly_113	-6.612e+02	1.727e+03	-0.383	0.703
horsepower_poly_114	1.003e+01	2.541e+01	0.395	0.694
horsepower_poly_115	-1.009e-01	2.479e-01	-0.407	0.685
horsepower_poly_116	6.788e-04	1.616e-03	0.420	0.676
horsepower_poly_117	-2.984e-06	6.881e-06	-0.434	0.666
horsepower_poly_118	7.964e-09	1.777e-08	0.448	0.655
horsepower_poly_119	-1.037e-11	2.240e-11	-0.463	0.644
horsepower_poly_1110	NA	NA	NA	NA
horsepower_poly_1111	1.068e-17	2.164e-17	0.493	0.623

Residual standard error: 4129 on 88 degrees of freedom

Multiple R-squared: 0.7445, Adjusted R-squared: 0.7155

F-statistic: 25.64 on 10 and 88 DF, p-value: < 2.2e-16

Remaining Training Data (45% - Polynomial Degree 11):

RMSE: 4059.367

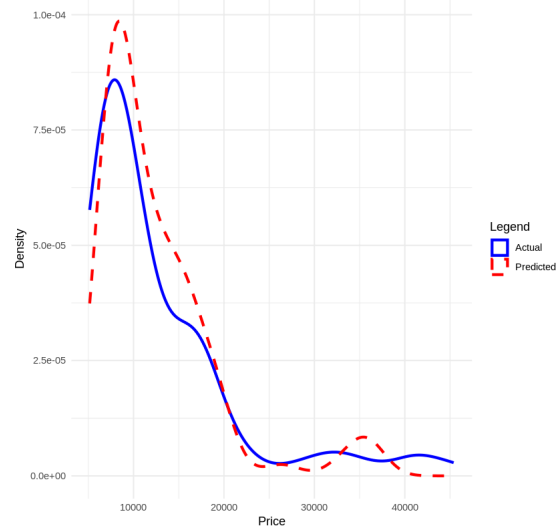
R-squared: 0.7744366

Test Data (Polynomial Degree 11):

RMSE: 8271.134

R-squared: -0.9797472

Distribution of Actual vs Predicted - Remaining Training Data (45% - Polynom



Distribution of Actual vs Predicted - Test Data (Polynomial Degree 11)

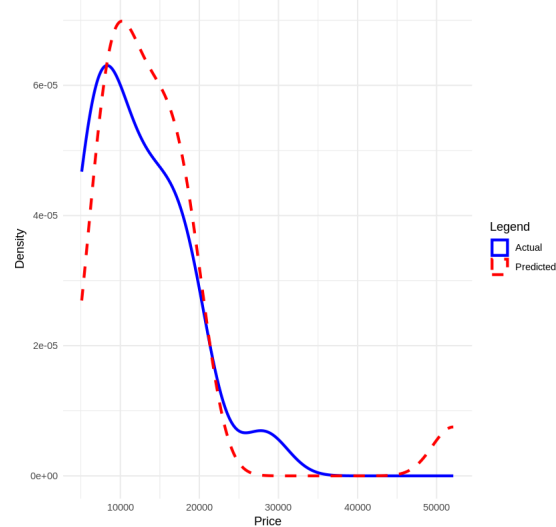


Figure 3: A polynomial regression model where red dots represent training data, green dots represent test data, and the blue line represents the model prediction.

We see that the estimated function appears to track the data but around 200 horsepower, the function begins to diverge from the data points.

R^2 of the training data: R-squared: 0.7744366

R^2 of the test data: R-squared: -0.9797472

We see the R^2 for the training data is 0.7744366 while the R^2 on the test data was -0.9797472. The lower the R^2 , the worse the model. A negative R^2 is a sign of overfitting.

Let's see how the R^2 changes on the test data for different order polynomials and then plot the results:

✓ Task Description and Requirements

The objective of this task is to analyze and visualize the performance of polynomial regression models for predicting car prices based on horsepower. Specifically, we will compare polynomial regression models of degrees ranging from 1 to 11 and evaluate the quality of the fit using the R-squared metric.

```
# Initialize a vector to store R^2 values
r_squared_values <- numeric(11)

# TO DO: Loop to create polynomial regression models for degrees 1 to 11
for (degree in 1:11) {
  # Create polynomial variables for horsepower
  x_train_55_poly <- x_train_55 %>%
    mutate(horsepower_poly = poly(horsepower, degree = degree, raw = TRUE))
# Create linear regression model
  model <- lm(price ~ horsepower_poly, data = data.frame(x_train_55_poly, price = y_train_55))

# Calculate R^2
  r_squared_values[degree] <- summary(model)$r.squared
}
# Print R^2 values for each polynomial degree
for (degree in 1:11) {
  cat("Degree:", degree, "R^2:", r_squared_values[degree], "\n")
}

# Plot R^2 values against polynomial degrees - Visualization
library(ggplot2)

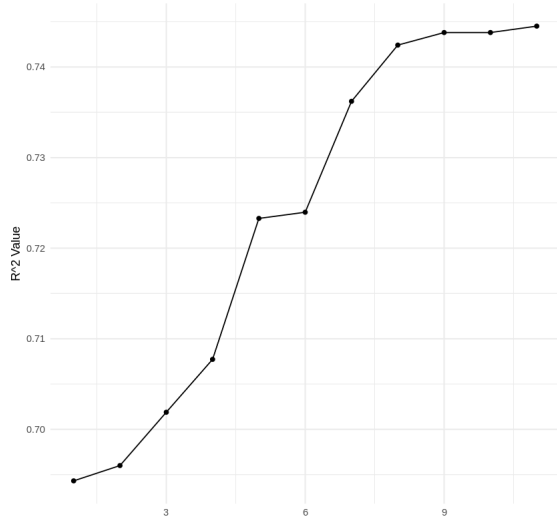
# Create dataframe for visualization
r_squared_df <- data.frame(Degree = 1:11, R_squared = r_squared_values)

# Generate plot - TO DO
ggplot(r_squared_df, aes(x = Degree, y = R_squared)) +
  geom_line() +
  geom_point() +
  labs(title = "R^2 Values for Polynomial Degrees 1 to 11",
       x = "Polynomial Degree",
       y = "R^2 Value") +
  theme_minimal()
```

```

Degree: 1 R^2: 0.6943162
Degree: 2 R^2: 0.6960052
Degree: 3 R^2: 0.701891
Degree: 4 R^2: 0.707729
Degree: 5 R^2: 0.7232873
Degree: 6 R^2: 0.7239691
Degree: 7 R^2: 0.7362115
Degree: 8 R^2: 0.7424176
Degree: 9 R^2: 0.7438045
Degree: 10 R^2: 0.7438045
Degree: 11 R^2: 0.7445109
R^2 Values for Polynomial Degrees 1 to 11

```



✓ Đánh Giá R²

1. Bậc 1 (R² = 0.6943):

- Mô hình hồi quy tuyến tính đơn giản với biến horsepower có thể giải thích khoảng 69.43% phương sai của biến mục tiêu price. Đây là một R² tốt cho mô hình đơn giản, nhưng có thể cải thiện.

2. Bậc 2 (R² = 0.6960):

- Việc thêm một bậc đa thức đã cải thiện nhẹ độ chính xác, với (R²) tăng lên 69.60%. Sự cải thiện là nhỏ nhưng cho thấy mô hình đang bắt đầu nắm bắt thêm một số biến thể trong dữ liệu.

3. Bậc 3 (R² = 0.7019):

- Tăng thêm một bậc nữa dẫn đến việc giải thích khoảng 70.19% phương sai, cho thấy mô hình có thể bắt đầu nắm bắt được các mối quan hệ phi tuyến tính giữa horsepower và price.

4. Bậc 4 (R² = 0.7077):

- Với (R²) tăng lên 70.77%, mô hình tiếp tục cải thiện độ chính xác. Điều này cho thấy rằng mối quan hệ giữa horsepower và price có thể phức tạp hơn và một mô hình phi tuyến tính hơn là cần thiết.

5. Bậc 5 (R² = 0.7233):

- Mô hình với bậc 5 đã giải thích khoảng 72.33% phương sai, cho thấy một sự cải thiện đáng kể. Điều này cho thấy mô hình đang bắt đầu nắm bắt được các xu hướng quan trọng trong dữ liệu.

6. Bậc 6 (R² = 0.7240):

- Chỉ số này rất gần với bậc 5, cho thấy rằng việc thêm bậc thứ 6 không tạo ra sự cải thiện đáng kể trong độ chính xác của mô hình.

7. Bậc 7 (R² = 0.7362):

- Với (R²) là 73.62%, mô hình bắt đầu cho thấy sự cải thiện lại. Điều này có thể cho thấy rằng độ phức tạp của mô hình đang tạo ra lợi ích.

8. Bậc 8 (R² = 0.7424):

- Tăng nhẹ (R²) lên 74.24%, mô hình bắt đầu giải thích một phần lớn hơn của phương sai trong price.

9. Bậc 9 (R² = 0.7438):

- R² chỉ tăng một chút (0.7438), cho thấy mô hình đang đạt đến giới hạn khả năng giải thích của nó.

10. Bậc 10 ($R^2 = 0.7438$):

- Không có sự cải thiện so với bậc 9. Điều này có thể cho thấy mô hình đã đạt đến một mức độ phức tạp tối ưu.

11. Bậc 11 ($R^2 = 0.7445$):

- Mặc dù có sự tăng nhẹ, nhưng mức tăng rất nhỏ. do đó khi thêm một bậc đa thức nữa không cải thiện đáng kể độ chính xác và có thể dẫn đến hiện tượng overfitting.

Kết Luận:

- **Tăng Trưởng Đầu Tiên:** Các bậc thấp (từ 1 đến 5) cho thấy sự cải thiện đáng kể về (R^2). Điều này cho thấy mô hình đang học hỏi từ dữ liệu và nắm bắt được các mối quan hệ quan trọng.
- **Khả Năng Tăng Trưởng Giảm Dần:** Sau bậc 5, mặc dù có sự tăng trưởng trong (R^2), nhưng mức độ cải thiện bắt đầu giảm dần và trở nên không đáng kể.
- **Overfitting:** Các mô hình với bậc cao (trên 8) có thể dẫn đến hiện tượng overfitting, vì mô hình học hỏi quá mức từ dữ liệu huấn luyện và không tổng quát tốt cho dữ liệu mới.

Part 3: Ridge Regression

The objective of this task is to perform Ridge Regression with a polynomial degree of 11 on the horsepower variable, evaluate the model's predictive performance, and visualize the results. Ridge Regression, a regularized form of linear regression, helps prevent overfitting by adding a penalty to the regression coefficients based on a regularization parameter, lambda.

```
install.packages("MASS")
```

```
Installing package into ‘/usr/local/lib/R/site-library’  
(as ‘lib’ is unspecified)
```

```
# Libraries
```

```
library(MASS) # Library for Ridge Regression  
library(ggplot2)
```

```
# Generate polynomial features for horsepower up to degree 11 in the training data - TO DO  
x_train_poly <- data.frame(horsepower = x_train$horsepower)  
x_train_poly <- cbind(x_train_poly, poly(x_train$horsepower, degree = 11, raw = TRUE))
```

```
# Set column names for the polynomial features - TO DO  
colnames(x_train_poly) <- c("horsepower", paste0("X", 1:11))
```

```
# Train Ridge Regression model with lambda = 0.5 - TO DO  
lambda_value <- 0.5  
ridge_model <- lm.ridge(price ~ ., data = data.frame(x_train_poly, price = y_train), lambda = lambda_value)
```

```
# Generate polynomial features for the test data - TO DO  
x_test_poly <- data.frame(horsepower = x_test$horsepower)  
x_test_poly <- cbind(x_test_poly, poly(x_test$horsepower, degree = 11, raw = TRUE))
```

```
# Set column names for the polynomial features in the test data - TO DO  
colnames(x_test_poly) <- c("horsepower", paste0("X", 1:11))
```

```
# Convert test data to matrix for prediction  
x_test_matrix <- as.matrix(x_test_poly)
```

```
# Predict prices using the trained Ridge Regression model - TO DO  
y_pred_ridge <- as.vector(cbind(1, x_test_matrix) %*% coef(ridge_model))
```

```
# Calculate  $R^2$  for the test data - Evaluation  
rss <- sum((y_test - y_pred_ridge) ^ 2)  
tss <- sum((y_test - mean(y_test)) ^ 2)  
r_squared_ridge <- 1 - (rss / tss)
```

```

# Print R^2 value for Ridge Regression (degree 11) with lambda
cat("R^2 for Ridge Regression (degree 11) with lambda =", lambda_value, ":", r_squared_ridge, "\n")

# Visualization: Scatter plot for Actual vs. Predicted values
comparison_df <- data.frame(Actual = y_test, Predicted = y_pred_ridge)

ggplot(comparison_df, aes(x = Actual, y = Predicted)) +
  geom_point(color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "So sanh gia tri thuc va gia tri du doan (Hoi quy Ridge)",
        x = "Gia tri thuc",
        y = "Gia tri du doan") +
  theme_minimal()

# Function to create a Distribution Plot - TO DO
distribution_plot <- function(actual, predicted, title) {
  data <- data.frame(Actual = actual, Predicted = predicted)

  ggplot(data) +
    geom_density(aes(x = Actual, fill = "Actual"), alpha = 0.5) +
    geom_density(aes(x = Predicted, fill = "Predicted"), alpha = 0.5) +
    labs(title = title, x = "Value", y = "Mat do") +
    scale_fill_manual(name = "Tham so", values = c("Actual" = "blue", "Predicted" = "orange")) +
    theme_minimal()
}

# Create Distribution Plot for test data
distribution_plot(y_test, y_pred_ridge, "Test Data (Polynomial Degree 11)")

```



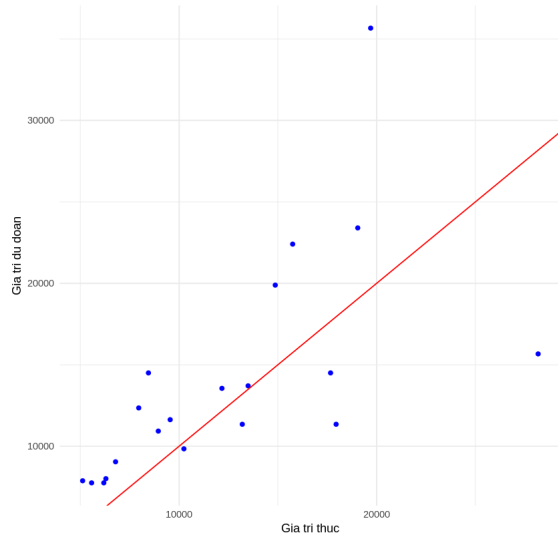
Attaching package: 'MASS'

The following object is masked from 'package:dplyr':

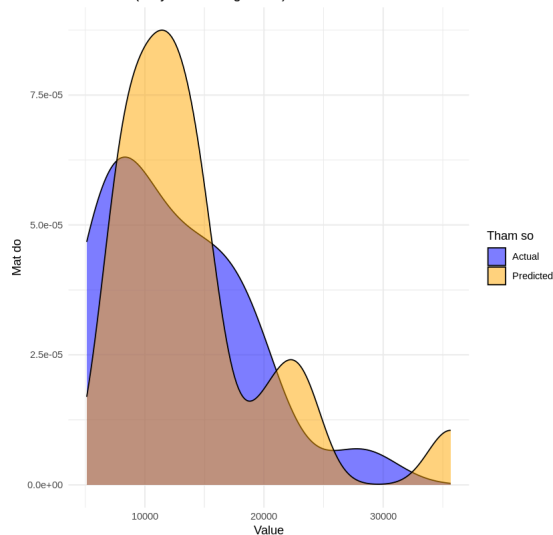
select

R² for Ridge Regression (degree 11) with lambda = 0.5 : 0.06512191

So sánh giá trị thực và giá trị dự đoán (Hội quy Ridge)



Test Data (Polynomial Degree 11)



✓ Description and Requirements

- This task focuses on applying Ridge Regression using polynomial features of degree 11 for the horsepower variable. The goal is to explore different values of the regularization parameter (lambda), identify the optimal value using Generalized Cross-Validation (GCV), and evaluate the predictive performance of the model using test data.

```
# Libraries
library(MASS) # Library for Ridge Regression
library(ggplot2)

# Generate polynomial features for horsepower up to degree 11 in the training data - TO DO
x_train_poly <- data.frame(horsepower = x_train$horsepower)
x_train_poly <- cbind(x_train_poly, poly(x_train$horsepower, degree = 11, raw = TRUE))

# Set column names for the polynomial features - TO DO
colnames(x_train_poly) <- c("horsepower", paste0("X", 1:11))

# Perform Ridge Regression for a range of lambda values - TO DO
```

```

lambdas <- seq(0, 10, by = 0.1)
ridge_models <- lm.ridge(price ~ ., data = data.frame(x_train_poly, price = y_train), lambda = lambdas)

# Select optimal lambda based on GCV - TO DO
optimal_lambda <- ridge_models$lambda[which.min(ridge_models$GCV)]
cat("Gia tri lambda toi uu:", optimal_lambda, "\n")

# Train Ridge Regression model with the optimal lambda - TO DO
ridge_model <- lm.ridge(price ~ ., data = data.frame(x_train_poly, price = y_train), lambda = optimal_lambda)

# Generate polynomial features for the test data - TO DO
x_test_poly <- data.frame(horsepower = x_test$horsepower)
x_test_poly <- cbind(x_test_poly, poly(x_test$horsepower, degree = 11, raw = TRUE))

# Set column names for the polynomial features in the test data - TO DO
colnames(x_test_poly) <- c("horsepower", paste0("X", 1:11))

# Convert test data to matrix for prediction
x_test_matrix <- as.matrix(x_test_poly)

# Predict prices using the trained Ridge Regression model with optimal lambda - TO DO
y_pred_ridge <- as.vector(cbind(1, x_test_matrix) %*% coef(ridge_model))

# Calculate R^2 for the test data - Evaluation
rss <- sum((y_test - y_pred_ridge) ^ 2)
tss <- sum((y_test - mean(y_test)) ^ 2)
r_squared_ridge <- 1 - (rss / tss)

# Print R^2 value for Ridge Regression (degree 11) with optimal lambda
cat("R^2 for Ridge Regression (degree 11) with optimal lambda =", optimal_lambda, ":", r_squared_ridge, "\n")
#print
cat("R^2 for Ridge Regression Model (base 11) with lambda =", optimal_lambda, ":", r_squared_ridge, "\n")

# Visualization: Scatter plot for Actual vs. Predicted values
comparison_df <- data.frame(Actual = y_test, Predicted = y_pred_ridge)

ggplot(comparison_df, aes(x = Actual, y = Predicted)) +
  geom_point(color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "Compare the actual value and the predicted value (Ridge Regression)",
       x = "Actual values",
       y = "Predicted values") +
  theme_minimal()

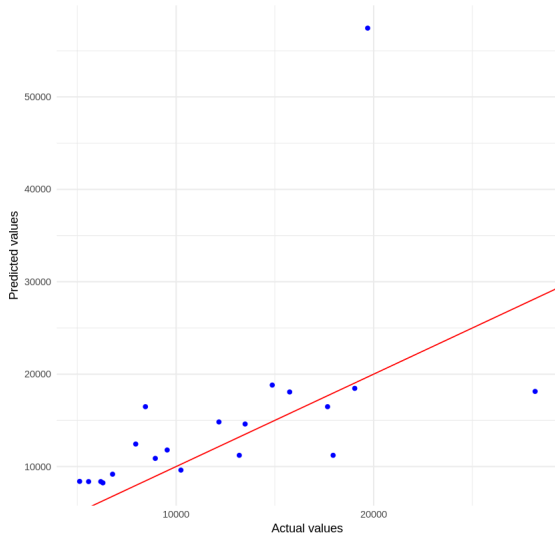
# Function to create a Distribution Plot - TO DO
distribution_plot <- function(actual, predicted, title) {
  data <- data.frame(Actual = actual, Predicted = predicted)

  ggplot(data) +
    geom_density(aes(x = Actual, fill = "Actual"), alpha = 0.5) +
    geom_density(aes(x = Predicted, fill = "Predicted"), alpha = 0.5) +
    labs(title = title, x = "Value", y = "Mat do") +
    scale_fill_manual(name = "Tham so", values = c("Actual" = "blue", "Predicted" = "orange")) +
    theme_minimal()
}

# Create Distribution Plot for test data
distribution_plot(y_test, y_pred_ridge, "Test Data (Polynomial Degree 11)")

```

↻ Gia tri lambda toi uu: 0
 R² for Ridge Regression (degree 11) with optimal lambda = 0 : -1.507141
 R² for Ridge Regression Model (base 11) with lambda = 0 : -1.507141
 Compare the actual value and the predicted value (Ridge Regression)



Test Data (Polynomial Degree 11)

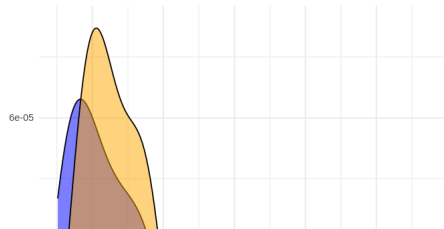


Figure 4: The blue line represents the R² of the validation data, and the red line represents the R² of the training data. The x-axis represents the different values of Alpha.



Here the model is built and tested on the same data, so the training and test data are the same.

The red line in Figure 4 represents the R² of the training data. As alpha increases the R² decreases. Therefore, as alpha increases, the model performs worse on the training data

The blue line represents the R² on the validation data. As the value for alpha increases, the R² decreases and converges at a point.

Question #5):

Perform Ridge regression. Calculate the R² using the polynomial features, use the training data to train the model and use the test data to test the model. The parameter alpha should be set to 10.

```
install.packages("glmnet")
```

↻ Installing package into ‘/usr/local/lib/R/site-library’
 (as ‘lib’ is unspecified)

also installing the dependency ‘RcppEigen’