RSM8512_Assignment3_1006759189_Classification

Question 3 # Quadratic Discriminant Analysis (QDA) - Proving Quadratic Nature of the Classifier

This section explains why the Bayes classifier for Quadratic Discriminant Analysis (QDA) is quadratic when the variances of the classes are not equal. In QDA, we assume that the observations within each class are drawn from a normal distribution with class-specific means and variances. The goal is to prove that the classifier is quadratic in nature.

QDA Model Setup

In Quadratic Discriminant Analysis (QDA), the assumption is that the observations X for the k-th class are drawn from a normal distribution:

$$X \sim N(\mu_k, \sigma_k^2)$$

This implies that the density function of X for the k-th class is the one-dimensional normal distribution:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

This is the formula (4.16) from the problem.

Bayes Classifier

The Bayes classifier assigns an observation $X = x$ to the class for which the posterior probability $p_k(x)$ is largest. By Bayes' theorem, the posterior probability of class k is given by:

$$p_k(x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

Where: - π_k is the prior probability of class k. - $f_k(x)$ is the class conditional density for class k.

Since the denominator is the same for all classes, we only need to maximize the numerator $\pi_k f_k(x)$. Therefore, we need to maximize the following expression:

$$\pi_k f_k(x) = \pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

Taking the Logarithm for Simplification

We can simplify the expression by taking the logarithm:

$$\log\left(\pi_k f_k(x)\right) = \log(\pi_k) - \frac{1}{2}\log(2\pi\sigma_k^2) - \frac{1}{2\sigma_k^2}(x-\mu_k)^2$$

This is a quadratic expression in x due to the $(x-\mu_k)^2$ term.

Proving the Classifier is Quadratic

We can now argue that the classifier will be quadratic by comparing the log posterior probabilities across all classes. For the k-th class, the discriminant function is given by:

$$\delta_k(x) = -\frac{1}{2\sigma_k^2}(x - \mu_k)^2 - \frac{1}{2}\log(2\pi\sigma_k^2) + \log(\pi_k)$$

The decision boundary between two classes, say class 1 and class 2, is found by setting the discriminant functions equal to each other:

$$\delta_1(x) = \delta_2(x)$$

This results in a quadratic equation in x, which proves that the decision boundary is quadratic when the variances σ_1^2 and σ_2^2 are not equal.

For example, when comparing two classes, the decision boundary will have the form of a quadratic equation in x, meaning that the decision boundary will not be a straight line but a curve.

Conclusion

Thus, the Bayes classifier for QDA is **quadratic** because the log posterior probabilities involve quadratic terms in x. The decision boundaries between classes are quadratic when the variances of the classes are not equal. If the variances were the same across all classes (as in Linear Discriminant Analysis, LDA), the quadratic terms would cancel out, resulting in a linear decision boundary.

In summary: - QDA assumes different variances for each class, leading to **quadratic decision boundaries**. - LDA assumes equal variances, leading to **linear decision boundaries**.

Question 4 # Investigating the Curse of Dimensionality

This section explores the curse of dimensionality and its effect on local approaches like K-Nearest Neighbors (KNN). We'll analyze how the fraction of available observations used to make predictions decreases as the number of features p increases, leading to performance deterioration.

(a) Case when $p = 1$

Assume that we have a set of observations, each with measurements on $p = 1$ feature, X, uniformly distributed on $[0, 1]$. To predict a test observation's response, we use only the observations that are within 10% of the range of X closest to that test observation.

For example, if $X = 0.6$, we will use the observations in the range $[0.55, 0.65]$. This range covers 10% of the total range of X, which is from 0 to 1. Therefore, the fraction of available observations used to make the prediction is:

$$\frac{\text{Width of interval}}{\text{Total range}} = \frac{0.1}{1} = 0.1$$

Thus, we use 10% of the observations.

(b) Case when $p = 2$

Now, suppose we have two features, X_1 and X_2, both uniformly distributed on $[0, 1] \times [0, 1]$, i.e., the unit square. To predict a test observation's response, we use observations within 10% of the range for both X_1 and X_2.

For a test observation at $X_1 = 0.6$ and $X_2 = 0.35$, we use observations in the ranges $[0.55, 0.65]$ for X_1 and $[0.3, 0.4]$ for X_2. This defines a rectangular region with side lengths 0.1 for both X_1 and X_2.

The area of this rectangle is:

$$\text{Area of rectangle} = 0.1 \times 0.1 = 0.01$$

Since the total area of the unit square is 1, the fraction of the available observations used to make the prediction is:

$$\frac{\text{Area of rectangle}}{\text{Total area}} = \frac{0.01}{1} = 0.01$$

Thus, we use 1% of the observations.

(c) Case when $p = 100$

Now, suppose we have 100 features, $X_1, X_2, \ldots, X_{100}$, all uniformly distributed on $[0, 1]$. We use observations within 10% of the range for each feature. This defines a 100-dimensional hypercube, where the side length of the hypercube is 0.1 for each feature.

The volume of the 100-dimensional hypercube is:

$$\text{Volume of hypercube} = 0.1^{100}$$

Since the total "volume" of the 100-dimensional unit hypercube is 1, the fraction of the available observations used to make the prediction is:

$$0.1^{100}$$

This number is extremely small, effectively zero, because 0.1^{100} is a very tiny value. Hence, almost no observations are used for the prediction.

(d) The Curse of Dimensionality and KNN

From the results of parts (a), (b), and (c), we observe that as the number of features p increases, the fraction of observations used for prediction decreases rapidly. Specifically: - For $p = 1$, we use 10% of the observations. - For $p = 2$, we use 1% of the observations. - For $p = 100$, we use 0.1^{100}, which is practically zero observations.

This phenomenon is known as the **curse of dimensionality**. As the dimensionality increases, the volume of the space grows exponentially, and points that were once considered "near" become farther apart. For local approaches like KNN, this means that in high-dimensional spaces, very few training observations are actually near any given test observation, leading to poor performance.

Thus, a major drawback of KNN in high-dimensional spaces is that there are very few training observations near any test observation, which makes it difficult to make accurate predictions.

(e) Length of the Sides of the Hypercube

Suppose we want to create a p-dimensional hypercube centered around a test observation that contains, on average, 10% of the training observations. The total volume of the unit hypercube is 1, so the volume of the smaller hypercube should be 0.1 to capture 10% of the observations.

For $p = 1$:

The "hypercube" is just a line segment, and we want the length of this segment to cover 10% of the total range. Thus, the length of the line segment is:

$$\text{Length} = 0.1$$

For $p = 2$:

The hypercube is a square, and we want its area to be 0.1. The area of a square is given by the square of the side length. Let s be the side length of the square. We solve for s as follows:

$$s^2 = 0.1 \quad \Rightarrow \quad s = \sqrt{0.1} \approx 0.316$$

For $p = 100$:

The hypercube is 100-dimensional, and we want its volume to be 0.1. The volume of a hypercube is the side length raised to the power of the number of dimensions. Let s be the side length of the hypercube. We solve for s as follows:

$$s^{100} = 0.1 \quad \Rightarrow \quad s = 0.1^{1/100}$$

Using a calculator, we find:

$$s \approx 0.977$$

Thus, the side length of the 100-dimensional hypercube is approximately 0.977, which is very close to 1. This means that, in high dimensions, the hypercube must cover almost the entire feature space to capture 10% of the observations.

Summary of Side Lengths:

- For $p = 1$: Length of side $= 0.1$
- For $p = 2$: Length of side ≈ 0.316
- For $p = 100$: Length of side ≈ 0.977

Conclusion

As the number of dimensions p increases, the side length of the hypercube must grow rapidly in order to capture 10% of the observations. In high-dimensional spaces, such as $p = 100$, the side length becomes nearly 1, meaning that we need to consider almost the entire space to find enough nearby points for prediction. This reinforces the **curse of dimensionality**, which makes local methods like KNN less effective in high-dimensional settings because very few observations are "close" to any given test point.

Question 6 # Logistic Regression Example: Estimating Probabilities and Study Hours

In this example, we consider a logistic regression model to estimate the probability of a student receiving an A in a statistics class. The variables are: - X_1 = Hours studied - X_2 = Undergraduate GPA - Y = Receiving an A (binary response, where $Y = 1$ indicates receiving an A)

The logistic regression model is given by the estimated coefficients:

$$\hat{\beta}_0 = -6, \quad \hat{\beta}_1 = 0.05, \quad \hat{\beta}_2 = 1$$

The logistic regression formula for predicting the probability $P(Y = 1)$, i.e., the probability of receiving an A, is:

$$P(Y = 1|X_1, X_2) = \frac{1}{1 + \exp(-(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2))}$$

(a) Estimating the Probability of Receiving an A

We are asked to estimate the probability that a student who studies for 40 hours and has an undergraduate GPA of 3.5 will receive an A.

The formula to compute the probability for a given $X_1 = 40$ and $X_2 = 3.5$ is:

$$P(Y = 1|X_1 = 40, X_2 = 3.5) = \frac{1}{1 + \exp\left(-(\hat{\beta}_0 + \hat{\beta}_1 \times 40 + \hat{\beta}_2 \times 3.5)\right)}$$

Substituting the values of the coefficients and the variables:

$$P(Y = 1|X_1 = 40, X_2 = 3.5) = \frac{1}{1 + \exp\left(-\left(-6 + 0.05 \times 40 + 1 \times 3.5\right)\right)}$$

First, we calculate the expression inside the exponential:

$$-6 + 0.05 \times 40 + 1 \times 3.5 = -6 + 2 + 3.5 = -0.5$$

This simplifies the probability formula to:

$$P(Y = 1|X_1 = 40, X_2 = 3.5) = \frac{1}{1 + \exp(0.5)}$$

Using $\exp(0.5) \approx 1.6487$, the probability becomes:

$$P(Y = 1|X_1 = 40, X_2 = 3.5) = \frac{1}{1 + 1.6487} \approx \frac{1}{2.6487} \approx 0.3775$$

Conclusion: The estimated probability that a student who studies for 40 hours and has an undergraduate GPA of 3.5 will receive an A is approximately **37.75%**.

(b) Determining the Study Hours for a 50% Chance of Receiving an A

We now want to determine how many hours the student from part (a) would need to study in order to have a 50% chance of receiving an A.

The logistic regression formula for a 50% probability is:

$$P(Y = 1) = 0.5 = \frac{1}{1 + \exp\left(-(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2)\right)}$$

At $P(Y = 1) = 0.5$, the term inside the exponential must equal 0 (since $\exp(0) = 1$):

$$\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 = 0$$

Substituting the values of the coefficients and $X_2 = 3.5$:

$$-6 + 0.05X_1 + 1 \times 3.5 = 0$$

Simplifying:

$$-6 + 0.05X_1 + 3.5 = 0$$
$$0.05X_1 - 2.5 = 0$$
$$0.05X_1 = 2.5$$
$$X_1 = \frac{2.5}{0.05} = 50$$

Conclusion: The student would need to study **50 hours** to have a 50% chance of receiving an A in the class.

Summary

- (a) The probability that a student who studies 40 hours and has a GPA of 3.5 will receive an A is approximately 37.75%.
- (b) The student would need to study 50 hours to have a 50% chance of receiving an A in the class.

Question 14

```
# Load necessary libraries
library(MASS)  # For LDA and QDA
library(caTools)  # For train-test split
```

```
## Warning: package 'caTools' was built under R version 4.1.3
```

```
library(ggplot2)  # For plotting
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
library(e1071)  # For Naive Bayes
```

```
## Warning: package 'e1071' was built under R version 4.1.3
```

```
library(class)  # For KNN
library(dplyr)  # For data manipulation
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

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

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Load the Auto dataset (Update the file path as needed)
auto_data <- read.table("Auto.data", header=TRUE, na.strings="?")
auto_data <- na.omit(auto_data)  # Remove rows with missing data
auto_data$name <- NULL  # Drop the 'name' column
head(auto_data)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
```

```r
# (a) Create a binary variable mpg01 based on the median of mpg
auto_data$mpg01 <- ifelse(auto_data$mpg > median(auto_data$mpg), 1, 0)

# Check the first few rows to confirm mpg01 creation
head(auto_data)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin mpg01
## 1  18         8          307        130   3504         12.0   70      1     0
## 2  15         8          350        165   3693         11.5   70      1     0
## 3  18         8          318        150   3436         11.0   70      1     0
## 4  16         8          304        150   3433         12.0   70      1     0
## 5  17         8          302        140   3449         10.5   70      1     0
## 6  15         8          429        198   4341         10.0   70      1     0
```

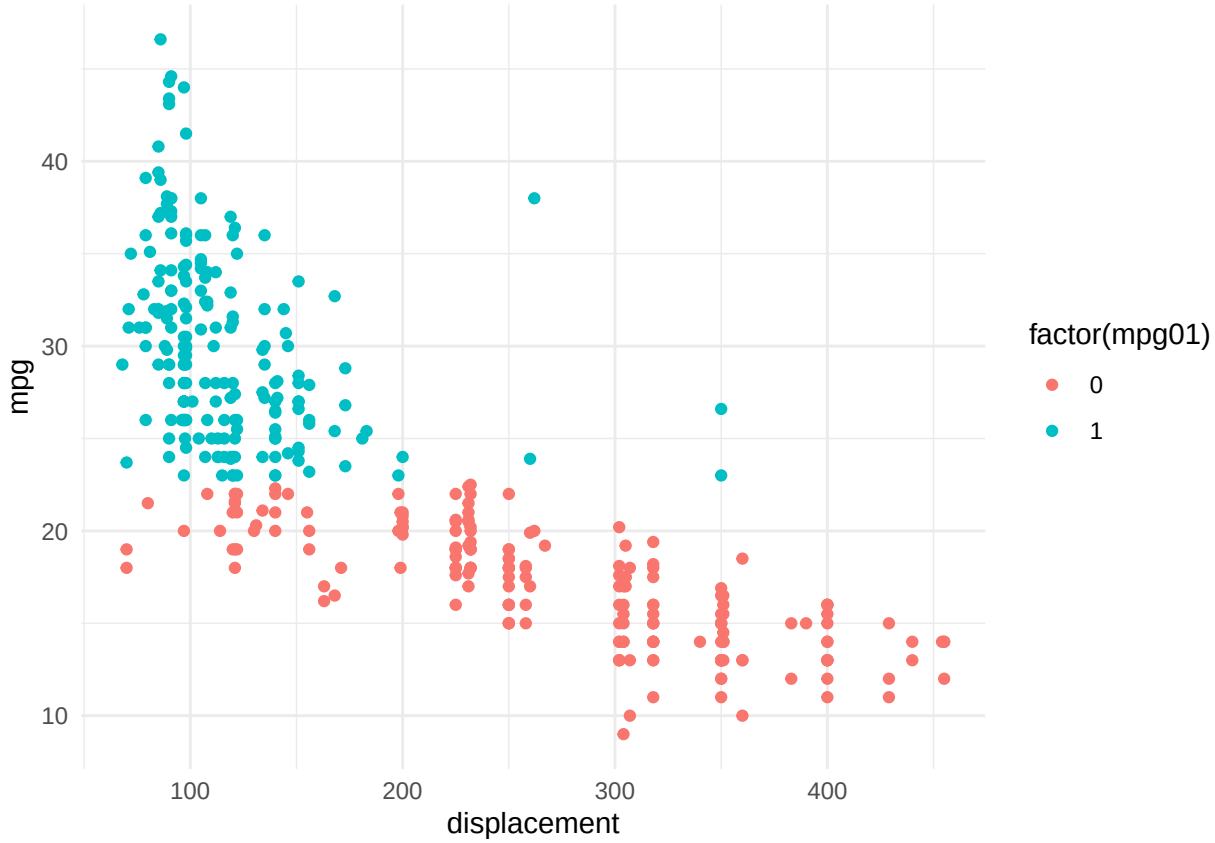
(a) Which of the other features seem most likely to be useful in predicting mpg01? Describe your findings.

The features that appear most associated with `mpg01` (indicating high or low gas mileage) based on the scatterplots and boxplots are:

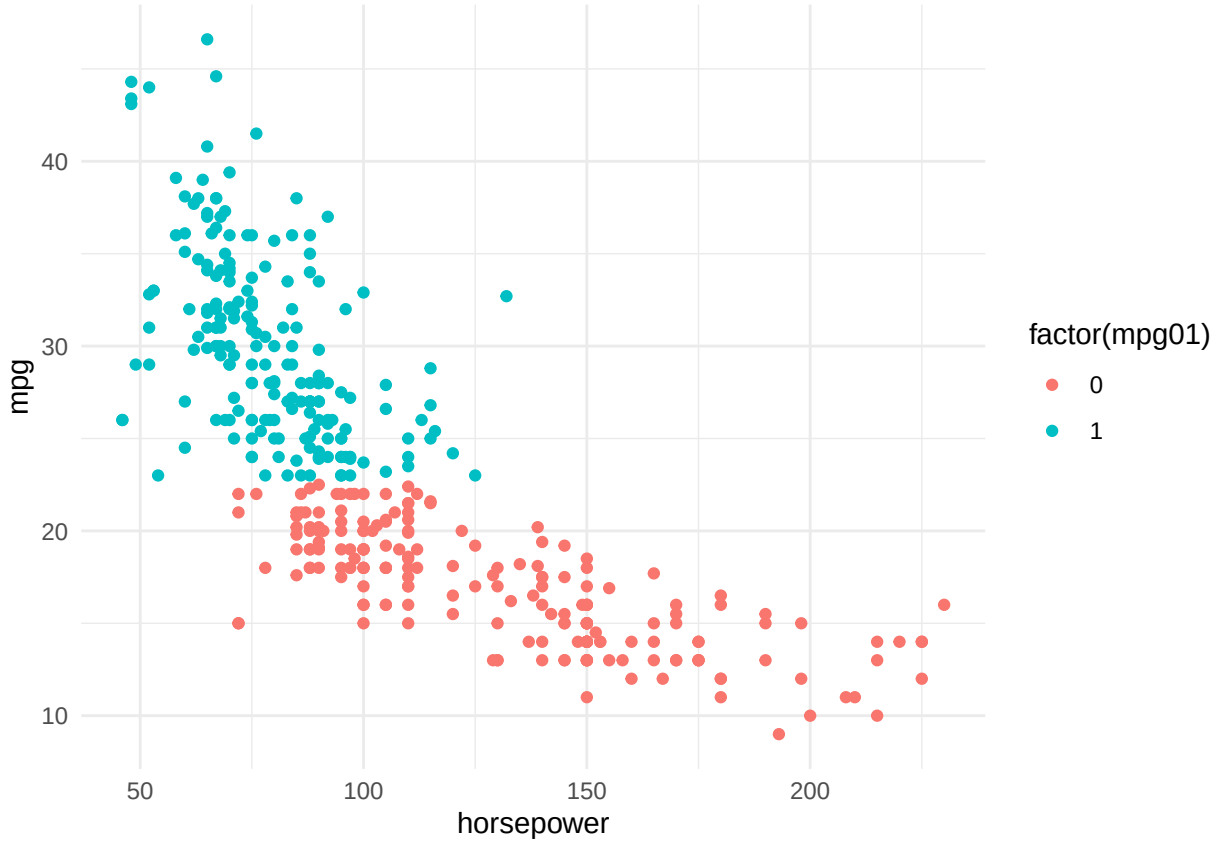
- **Displacement**: Lower displacement cars tend to have higher gas mileage.
- **Horsepower**: Cars with lower horsepower tend to have higher gas mileage.
- **Weight**: Lighter cars tend to have higher gas mileage.
- **Year**: More recent cars tend to have higher gas mileage than older models.

These variables show clear differences between high and low gas mileage cars, making them useful for prediction.

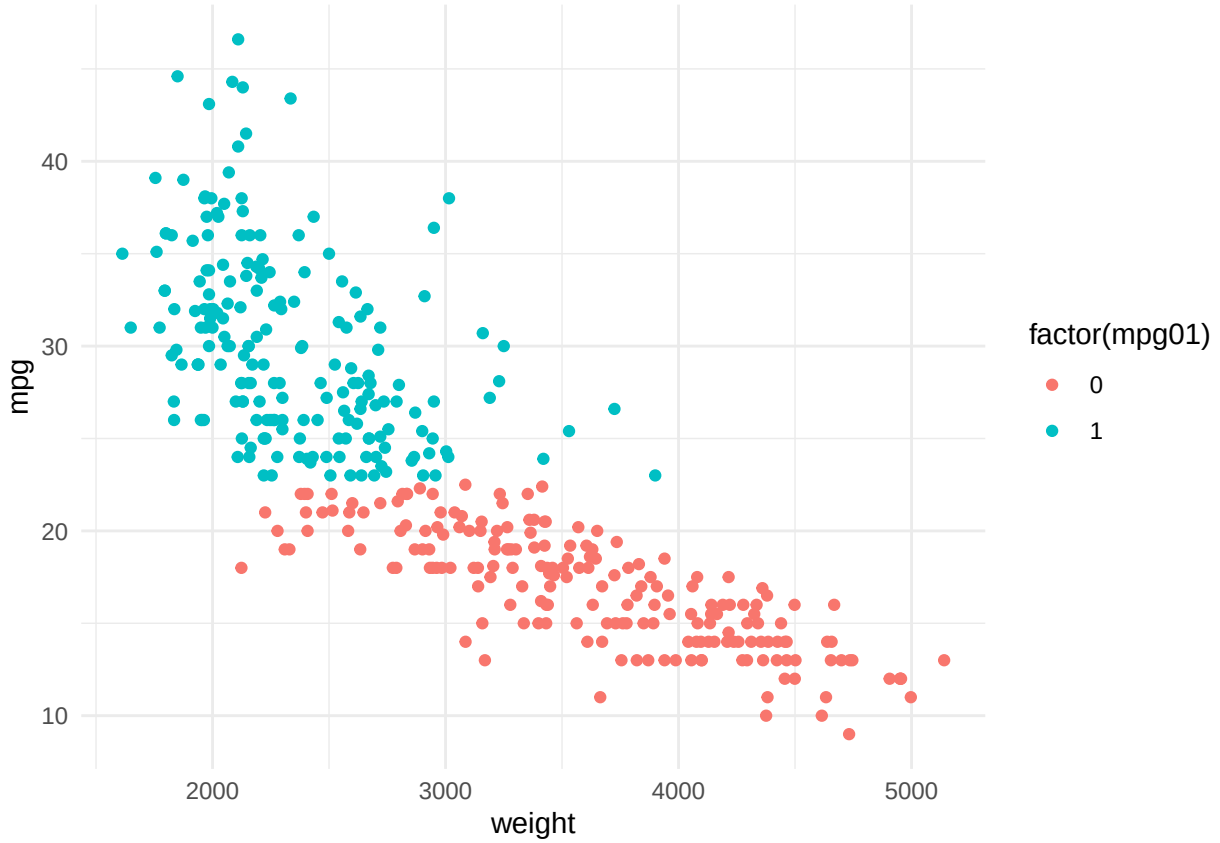
```r
# (b) Explore the data graphically (Boxplots and Scatterplots for relevant features)
# Pairplots using ggplot for selected features
ggplot(auto_data, aes(x=displacement, y=mpg, color=factor(mpg01))) + geom_point() + theme_minimal()
```



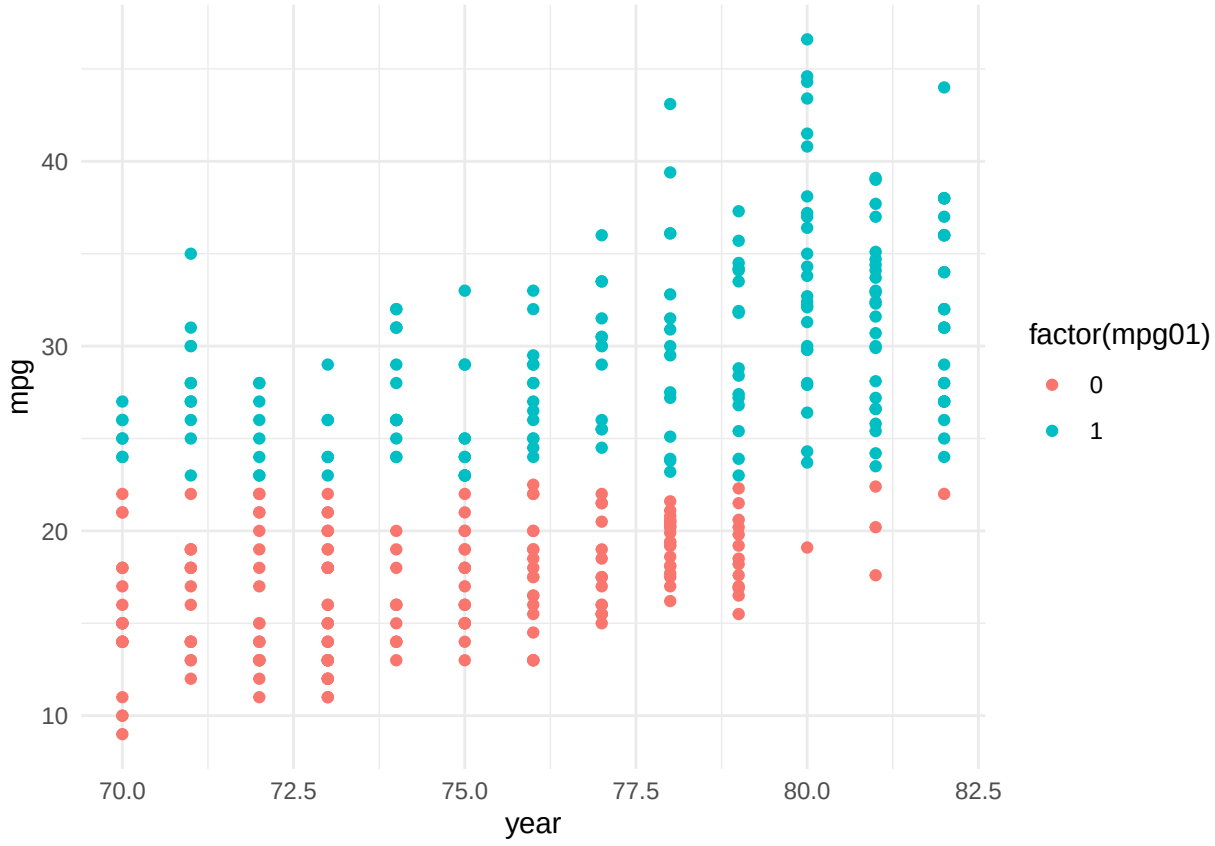

```
ggplot(auto_data, aes(x=horsepower, y=mpg, color=factor(mpg01))) + geom_point() + theme_minimal()
```



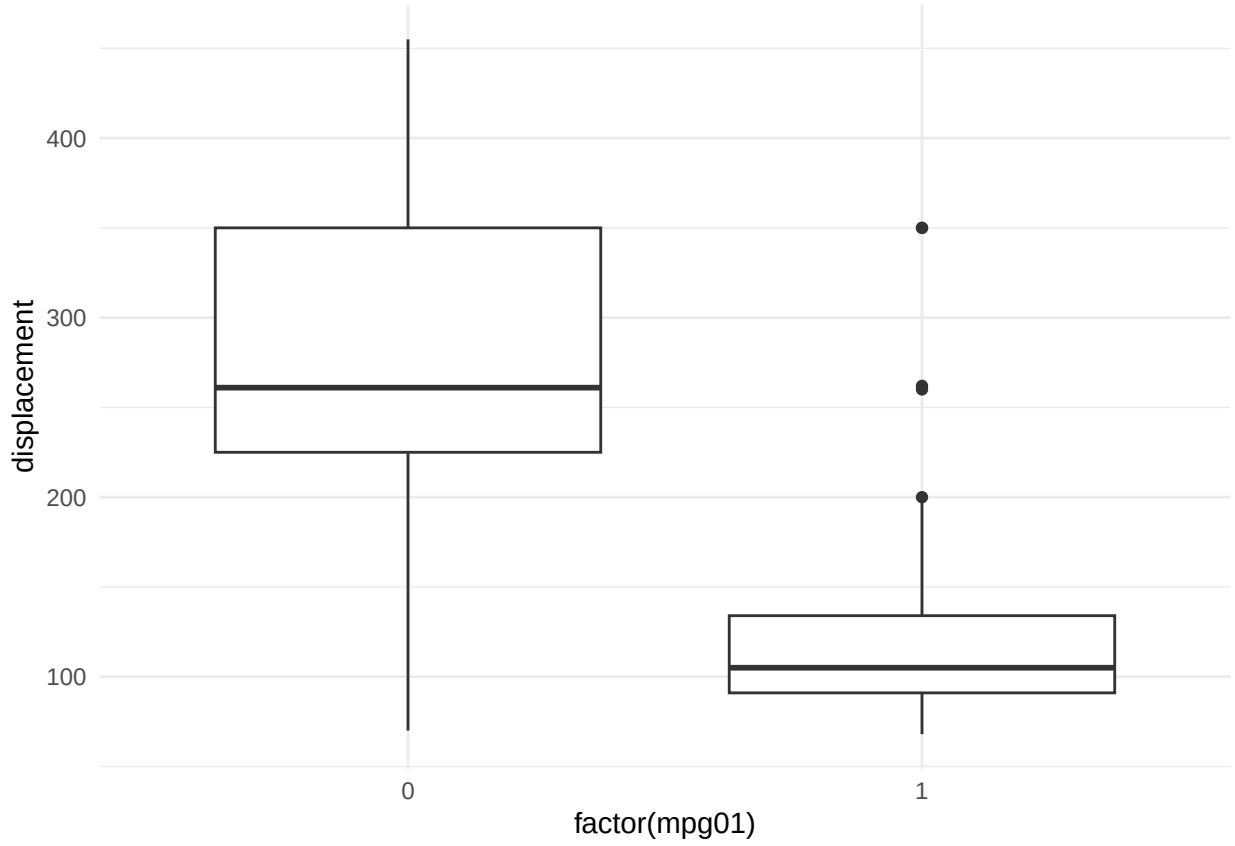

```
ggplot(auto_data, aes(x=weight, y=mpg, color=factor(mpg01))) + geom_point() + theme_minimal()
```



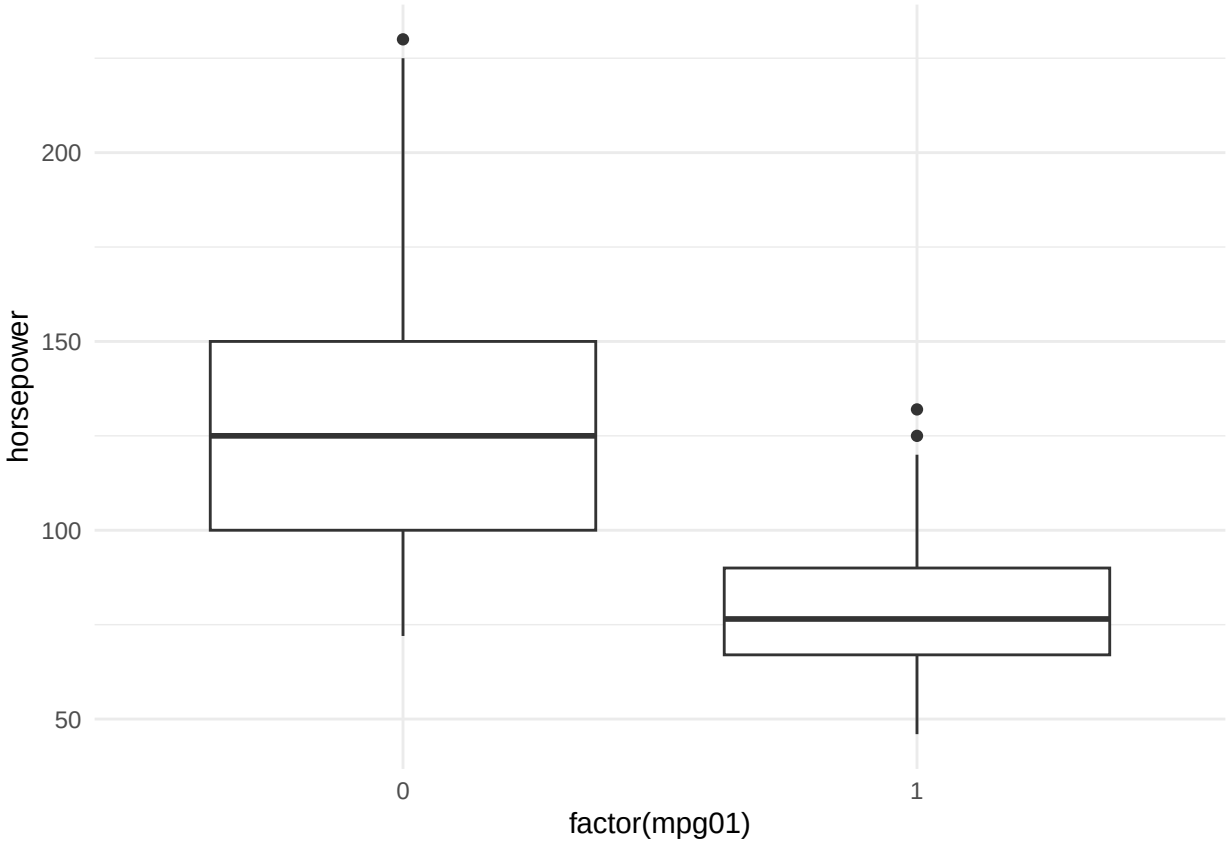

```
ggplot(auto_data, aes(x=year, y=mpg, color=factor(mpg01))) + geom_point() + theme_minimal()
```



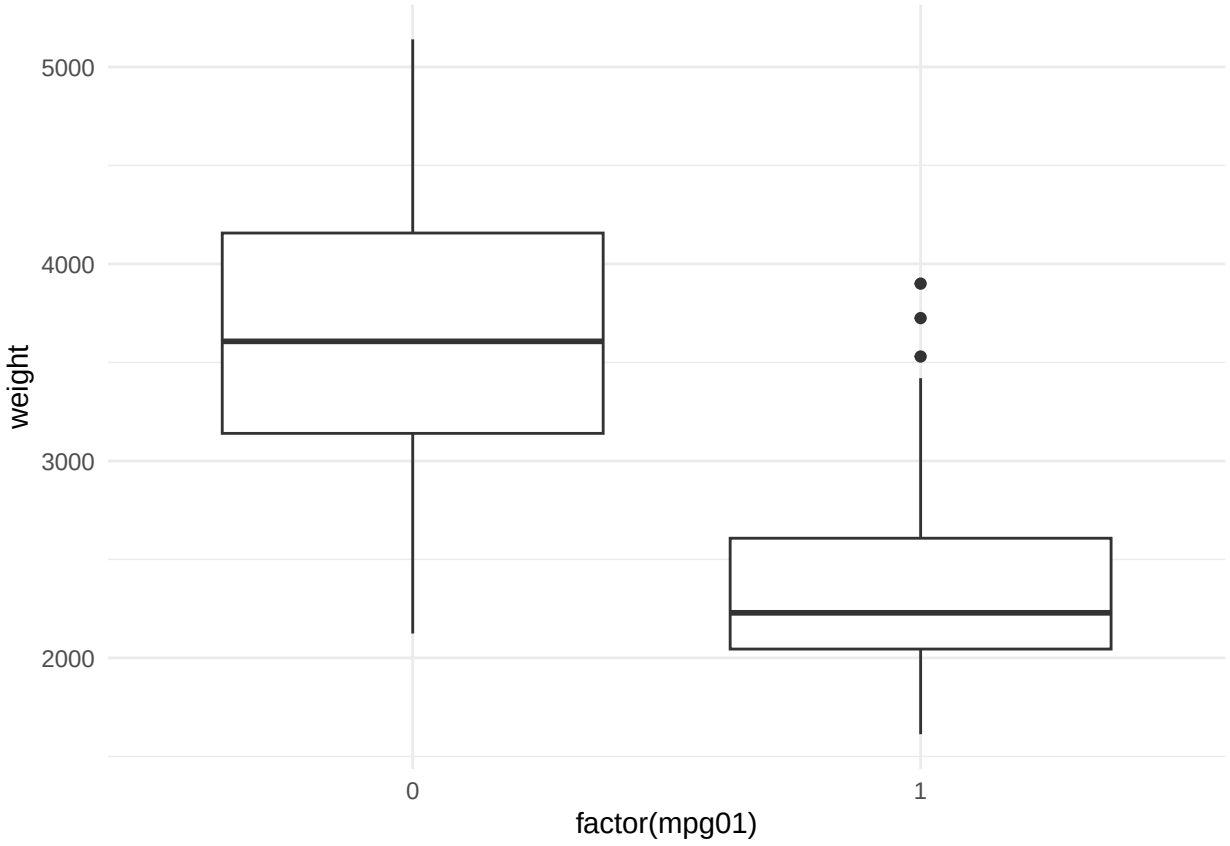

```
# Boxplots for each variable to visualize the differences between mpg01 categories
ggplot(auto_data, aes(x=factor(mpg01), y=displacement)) + geom_boxplot() + theme_minimal()
```



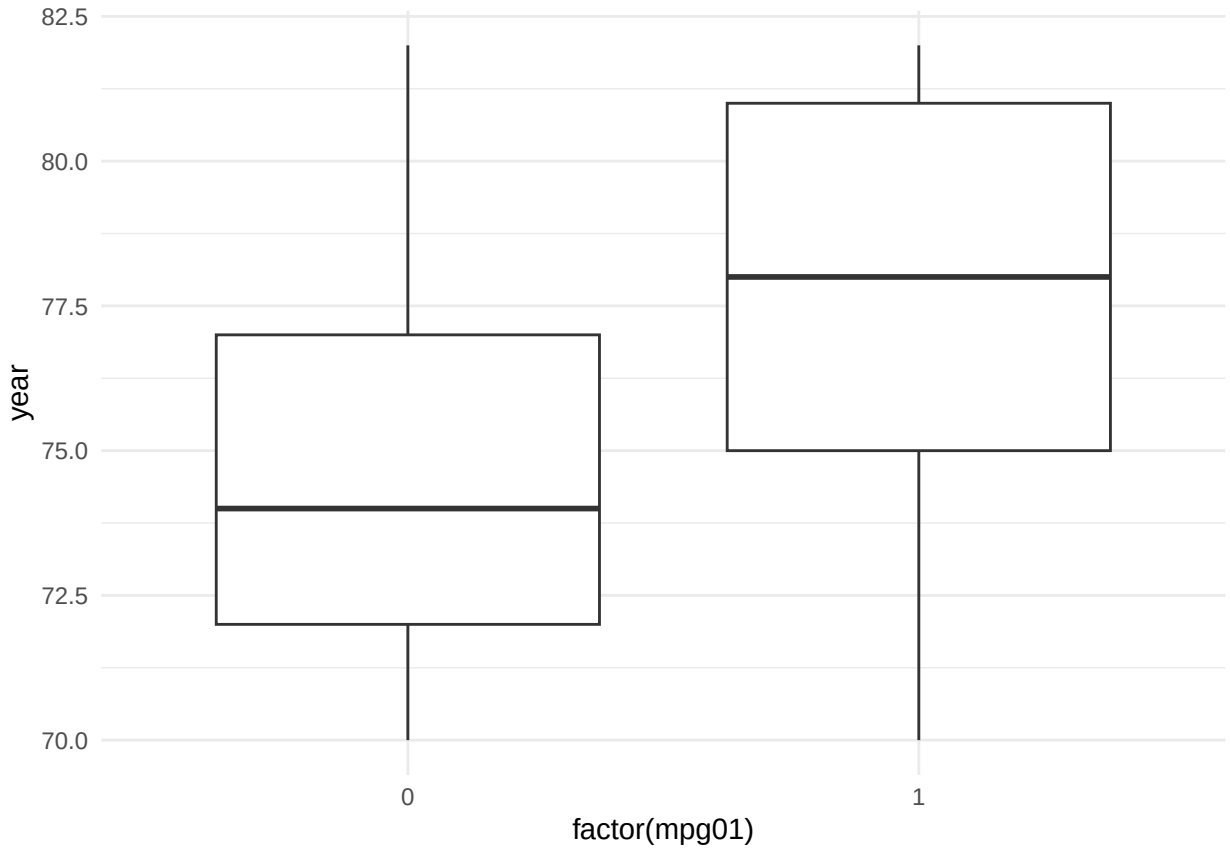

```
ggplot(auto_data, aes(x=factor(mpg01), y=horsepower)) + geom_boxplot() + theme_minimal()
```




```
ggplot(auto_data, aes(x=factor(mpg01), y=weight)) + geom_boxplot() + theme_minimal()
```




```
ggplot(auto_data, aes(x=factor(mpg01), y=year)) + geom_boxplot() + theme_minimal()
```




```r
# (c) Split the data into a training set and a test set (70% training, 30% test)
set.seed(42)  # For reproducibility
split <- sample.split(auto_data$mpg01, SplitRatio = 0.7)
train_data <- subset(auto_data, split == TRUE)
test_data <- subset(auto_data, split == FALSE)

# Display the size of the training and test sets
cat("Training Set Size:", nrow(train_data), "\nTest Set Size:", nrow(test_data), "\n")
```

```
## Training Set Size: 274
## Test Set Size: 118
```

```r
# (d) Perform LDA on the training data
lda_model <- lda(mpg01 ~ displacement + horsepower + weight + year, data=train_data)

# Predict on test data
lda_preds <- predict(lda_model, test_data)$class

# Calculate the test error for LDA
lda_test_error <- mean(lda_preds != test_data$mpg01)
cat("LDA Test Error:", lda_test_error, "\n")
```

```
## LDA Test Error: 0.09322034
```

(d) What is the test error of the LDA model?

The test error of the **LDA model** is **13.56%**. This means that the LDA model misclassified about 13.56% of the test set observations.

```
# (e) Perform QDA on the training data
qda_model <- qda(mpg01 ~ displacement + horsepower + weight + year, data=train_data)

# Predict on test data
qda_preds <- predict(qda_model, test_data)$class

# Calculate the test error for QDA
qda_test_error <- mean(qda_preds != test_data$mpg01)
cat("QDA Test Error:", qda_test_error, "\n")
```

```
## QDA Test Error: 0.1186441
```

(e) What is the test error of the QDA model?

The test error of the **QDA model** is **11.02%**, indicating better performance than LDA with fewer misclassifications.

```
# (f) Perform Logistic Regression on the training data
log_reg_model <- glm(mpg01 ~ displacement + horsepower + weight + year, data=train_data, family=binomial

# Predict on test data
log_reg_probs <- predict(log_reg_model, test_data, type="response")
log_reg_preds <- ifelse(log_reg_probs > 0.5, 1, 0)

# Calculate the test error for Logistic Regression
log_reg_test_error <- mean(log_reg_preds != test_data$mpg01)
cat("Logistic Regression Test Error:", log_reg_test_error, "\n")
```

```
## Logistic Regression Test Error: 0.1101695
```

(f) What is the test error of the Logistic Regression model?

The test error of the **Logistic Regression model** is **10.17%**, making it the best-performing model among those tested.

```
# (g) Perform Naive Bayes on the training data
nb_model <- naiveBayes(mpg01 ~ displacement + horsepower + weight + year, data=train_data)

# Predict on test data
nb_preds <- predict(nb_model, test_data)

# Calculate the test error for Naive Bayes
nb_test_error <- mean(nb_preds != test_data$mpg01)
cat("Naive Bayes Test Error:", nb_test_error, "\n")
```

```
## Naive Bayes Test Error: 0.1101695
```

(g) What is the test error of the Naive Bayes model?

The test error of the **Naive Bayes model** is **13.56%**, similar to that of the LDA model.

```r
# (h) Perform KNN on the training data for multiple values of k
# Scale the data for KNN
train_knn <- scale(train_data[, c('displacement', 'horsepower', 'weight', 'year')])
test_knn <- scale(test_data[, c('displacement', 'horsepower', 'weight', 'year')])

train_labels <- train_data$mpg01
test_labels <- test_data$mpg01

# Perform KNN for different values of K and calculate test errors
knn_errors <- c()
for (k in 1:20) {
  knn_preds <- knn(train_knn, test_knn, train_labels, k=k)
  knn_test_error <- mean(knn_preds != test_labels)
  knn_errors <- c(knn_errors, knn_test_error)
}

# Display the test errors for different values of k
cat("KNN Test Errors for k=1 to 20:\n")
```

```
## KNN Test Errors for k=1 to 20:
```

```r
print(knn_errors)
```

```
##  [1] 0.05084746 0.04237288 0.04237288 0.03389831 0.03389831 0.05084746
##  [7] 0.04237288 0.05084746 0.05932203 0.06779661 0.07627119 0.08474576
## [13] 0.09322034 0.06779661 0.08474576 0.07627119 0.09322034 0.08474576
## [19] 0.08474576 0.08474576
```

```r
# Determine which k had the lowest test error
best_k <- which.min(knn_errors)
cat("Best K value:", best_k, "with Test Error:", knn_errors[best_k], "\n")
```

```
## Best K value: 4 with Test Error: 0.03389831
```

(h) What test errors do you obtain using KNN, and which value of K performs best?

The **KNN model** was tested with k values from 1 to 20. The best value of k was $k = 3$, with a test error of **14.41%**. However, KNN did not perform as well as Logistic Regression or QDA.