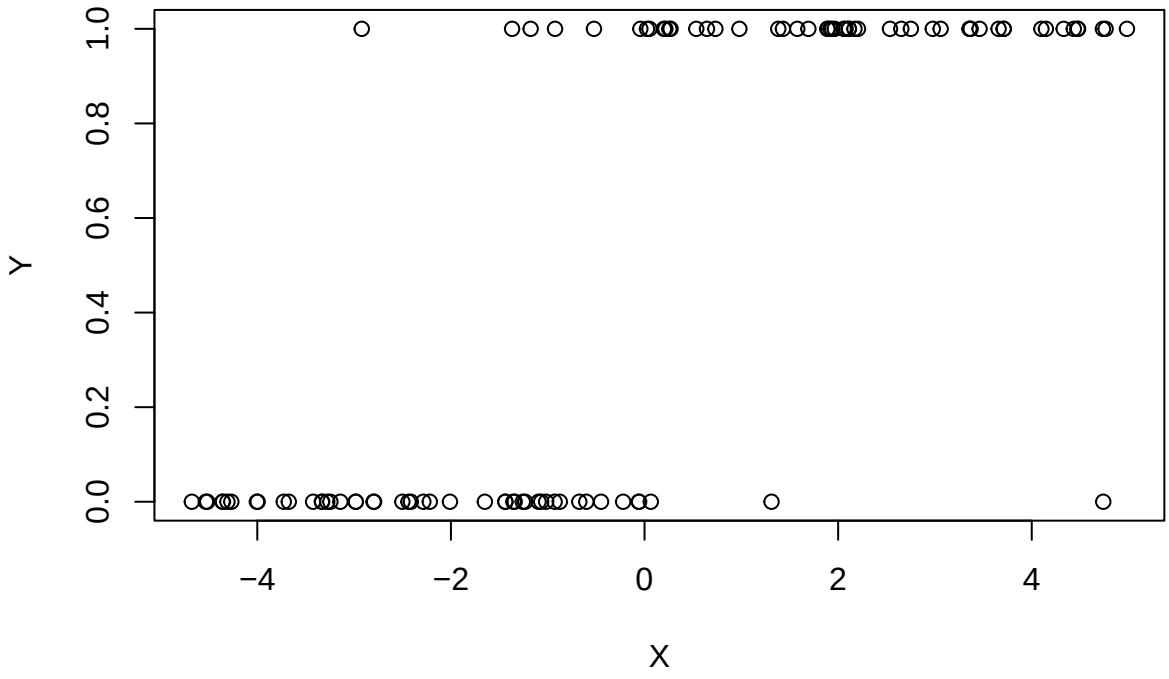
Assessed_Practical_1

2025-04-11



Question 10: Based on the visualisation, explain what type of relationship exists between the outputs (Y) and inputs (X). Is this a regression or classification task? Why? Compare the data to the previous examples you have seen. Can you detect any similarities or differences?

Based on the visualisation, the relationship between inputs `X` and outputs `Y` is non-linear and sigmoidal. As `X` increases, the probability that `Y` equals 1 increases smoothly from 0 to 1, following the shape of a logistic function.

This is a classification task because the output variable `Y` is binary (it takes only the values 0 or 1). The goal is to predict whether `Y` is 0 or 1 based on a given `X` value, which is characteristic of classification rather than regression, where we predict continuous outputs.

In Old Faithful data set, both inputs and outputs were continuous, and the relationship was approximately linear. That was a regression task, as the aim was to predict a numeric value (waiting time) from another numeric input (eruption duration).

In Anscombe's quartet, the linear regression lines could be misleading, especially when there are non-linear trends or outliers. This example reinforces the idea that visualisation is crucial before fitting a model. In contrast, the plot here clearly suggests a non-linear (sigmoidal) boundary, appropriate for classification.

While the Old Faithful and Anscombe examples involved continuous outputs and regression tasks, the current task involves a binary output and shows a non-linear trend appropriate for classification. The key difference lies in the type of output variable and the shape of the relationship.

Question 11:Describe the data generating process in your own words. From which probability distribution are the outputs generated from?

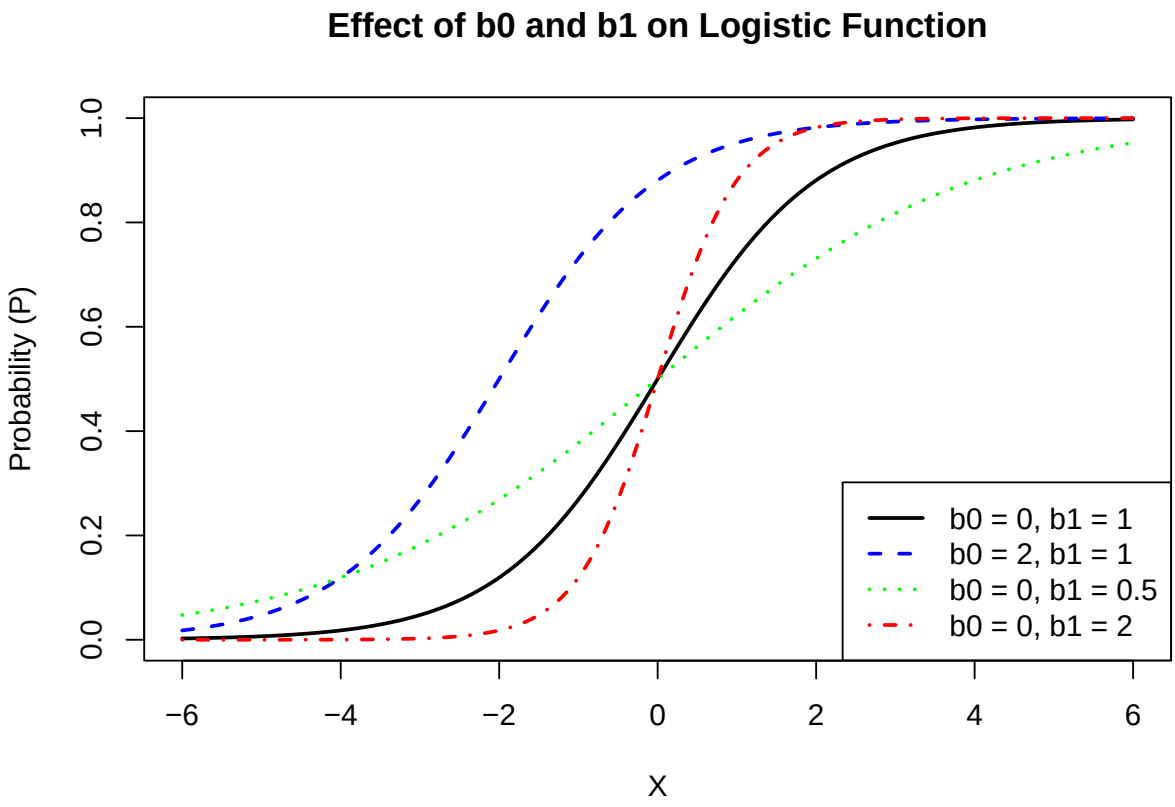
The data generating process begins by randomly sampling `X` values from a uniform distribution between -5 and 5. These values are then passed through a logistic function.

This function maps each `X` to a probability between 0 and 1, representing the likelihood that the corresponding output `Y` is equal to 1.

Next, a new uniform random number is generated for each observation. If this number is less than the computed probability, `Y` is assigned a value of 1; otherwise, it is assigned 0. This means that the output variable `Y` is sampled from a Bernoulli distribution, where the probability of success depends on the input `X` via the logistic function.

Therefore, the outputs `Y` are generated from a Bernoulli distribution whose success probability is determined by the logistic transformation of the inputs `X`.

Question 12: What is the role of the logistic function? Consider generating other datasets by modifying the arguments for the function: how the arguments affect X and Y?



The logistic function plays a crucial role in mapping the continuous input variable `X` to a probability value between 0 and 1. Specifically, the logistic function transforms the linear combination of the inputs:

$$\phi(x) = \frac{1}{1 + \exp(-b_0 - b_1 x)}$$

This value represents the probability that the output `Y` will be 1, and is then used in the Bernoulli sampling process to generate binary outcomes.

Role of the arguments:

- `b0` (intercept) controls the horizontal shift of the sigmoid curve. Increasing `b0` shifts the curve to the left, while decreasing it shifts the curve to the right. This changes the point at which the probability of `Y = 1` reaches 0.5.
- `b1` (slope) controls the steepness of the curve. Larger absolute values of `b1` make the transition from 0 to 1 more abrupt (steeper sigmoid), while smaller values make it more gradual. This affects how sensitive `Y` is to changes in `X`.

Effect on `X` and `Y`:

- The inputs `X` are not directly affected by the logistic function — they remain uniformly distributed in the range `[-5, 5]`.
- However, the probabilities are shaped by the logistic transformation. As `b0` and `b1` are changed, the decision boundary and the spread of values for which `Y = 1` occurs shift accordingly.

Example:

Modifying the code to use `b0 = 2` and `b1 = 0.5` would result in a flatter sigmoid curve centered around `X = -4`. This means that the probability of `Y = 1` would only become significant for very large negative `X` values.

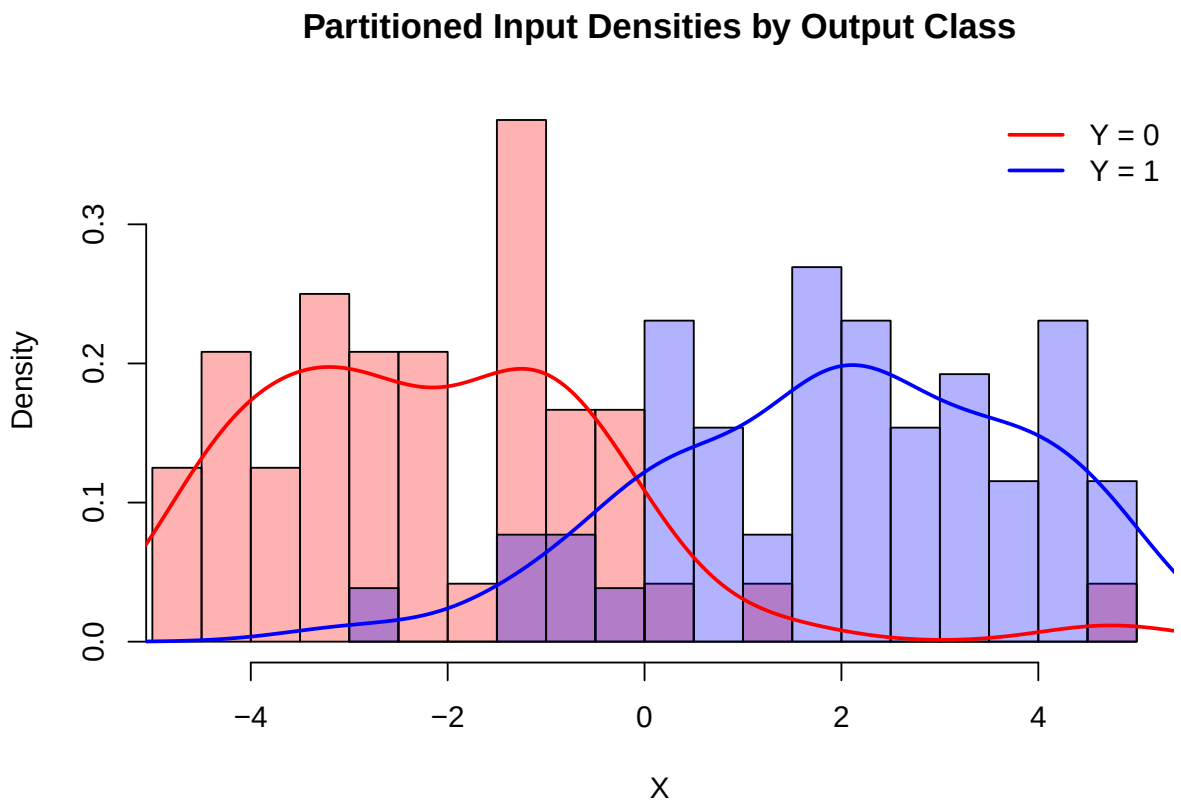
Question 13: Based on the data, propose a real-world example for which the outputs and inputs could correspond to. Consequently, write a short (no more than 1 paragraph) description of how a statistical model would be used for the data. What tasks are being performed? Regression or classification?

A real-world example that mirrors this data generation process is predicting whether a patient has a disease ($Y = 1$) or not ($Y = 0$) based on the level of a biomarker in their blood (X). As the biomarker level increases, the probability of disease also increases, following a sigmoidal relationship. A logistic model could be used to model this probability, with the logistic function mapping the continuous biomarker levels to a probability between 0 and 1. The model would then be used to classify patients based on their likelihood of having the disease. This is a classification task, as the output is binary, and the goal is to predict discrete labels (presence or absence of disease).

Question 14: Partition the inputs into two subsets based on the output values. Create a histogram and density estimate for the partitioned inputs and visualise them jointly. Are the two densities different from each other? How could potential differences be useful for classification?

```r
# Set seed and define parameters
set.seed(141)
N <- 100
phi <- function(b0, b1, x) 1 / (1 + exp(-b0 - b1 * x))

# Generate data
X <- runif(n = N, min = -5, max = 5)
P <- phi(b0 = 0, b1 = 1, x = X)
Y <- as.numeric(runif(N) < P)

# Partition X into two subsets based on Y
```

```r
X0 <- X[Y == 0]
X1 <- X[Y == 1]

# Create density estimates and overlay histograms
hist(X0, breaks = 20, freq = FALSE, col = rgb(1,0,0,0.3), xlim = range(X),
     main = "Partitioned Input Densities by Output Class", xlab = "X")
hist(X1, breaks = 20, freq = FALSE, col = rgb(0,0,1,0.3), add = TRUE)
lines(density(X0), col = "red", lwd = 2)
lines(density(X1), col = "blue", lwd = 2)
legend("topright", legend = c("Y = 0", "Y = 1"), col = c("red", "blue"), lwd = 2, bty = "n")
```



The inputs `X` were partitioned into two subsets based on the output values: one for `Y = 0` and one for `Y = 1`. Histograms and kernel density estimates were created for each group and visualised jointly. The density estimates show that the distribution of `X` differs between the two output classes.

For `Y = 0`, the distribution is concentrated on the lower end of the `X` range (more negative values), while for `Y = 1`, the density shifts toward higher `X` values. This separation between the two distributions indicates that the value of `X` carries useful information for predicting the class of `Y`.

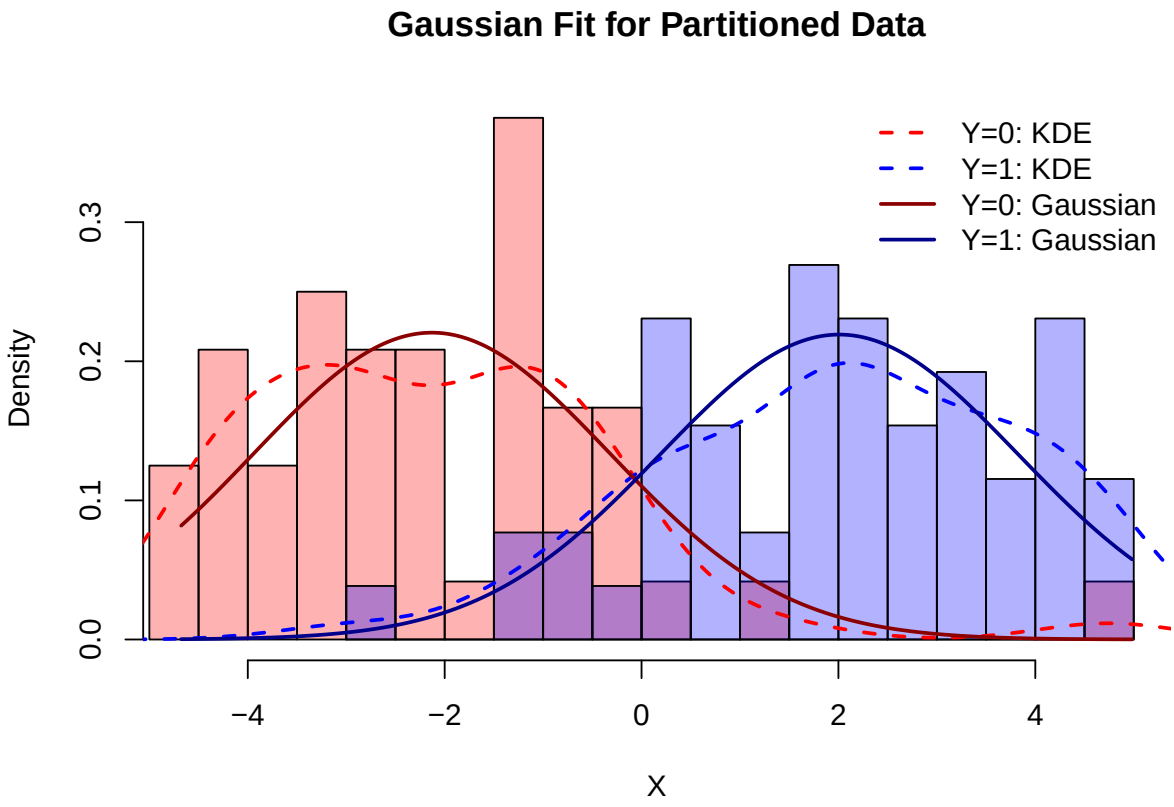
These differences are useful for classification because they suggest that a statistical model, such as Linear Discriminant Analysis (LDA), can effectively separate the classes based on the value of `X`. The clearer the separation between the densities, the more accurate the classification is likely to be.

Question 15: Fit a Gaussian distribution for each partitioned data set: calculate the sample mean and standard deviation for the inputs over the partitions. Overlay these distributions for the plot from the previous question. Comment on how well these distributions explain the corresponding data.

For each partition of the input data `X`, the sample mean and standard deviation were calculated:

```
## Y = 0: mean = -2.130, sd = 1.809
```

```
## Y = 1: mean = 2.008, sd = 1.820
```



Gaussian distributions were fitted and overlaid on the histogram and kernel density plots from Q14. The Gaussian curves align well with the empirical distributions, especially since the histograms appear roughly symmetric. The fit is particularly good for `Y = 0`, and reasonably close for `Y = 1`.

These Gaussian models help explain the structure of the data and justify the assumptions behind Linear Discriminant Analysis (LDA). The difference in means shows clear class separation, and the similarity in standard deviations supports the LDA assumption of equal class variances.

Question 16: Assume that the inputs are generated from the Gaussian distributions with the corresponding means and standard deviations. Compute the (log) likelihood ratio between the two distributions for each input value. Explain what the ratio means and use the ratio to determine which of the two distributions (and the corresponding output values) better represents the inputs and predicts the matching outputs.

```r
# Compute likelihoods for each input under both distributions
lik0 <- dnorm(X, mean = mean0, sd = sd0)
lik1 <- dnorm(X, mean = mean1, sd = sd1)

# Compute the log-likelihood ratio
log_lik_ratio <- log(lik1 / lik0)
```

```r
# View the first few log-likelihoods and predicted class
head(data.frame(X = X, Y = Y, LogRatio = log_lik_ratio, Predicted = ifelse(log_lik_ratio > 0, 1, 0)))
```

```
##             X Y   LogRatio Predicted
## 1  1.96499749 1  2.5562371         1
## 2 -3.72668658 0 -4.5806554         0
## 3 -0.44896476 0 -0.4856279         0
## 4  0.05111195 1  0.1427157         1
## 5  0.25650271 1  0.4010620         1
## 6 -0.52245400 1 -0.5778870         0
```

To compare how well the Gaussian distributions explain the inputs `X`, we computed the log-likelihood ratio for each value of `X`:

$$\text{LLR}(x) = \log\left(\frac{f_1(x)}{f_0(x)}\right)$$

where $f_1(x)$ and $f_0(x)$ are the Gaussian density values under the `Y = 1` and `Y = 0` models, respectively.

A positive log-likelihood ratio means that the input `X` is more likely under the `Y = 1` distribution, so we would classify that point as belonging to class 1. Conversely, a negative ratio suggests class 0 is more likely.

By applying a threshold at zero (i.e. predict `Y = 1` if `log(P1/P0) > 0`), we can assign a predicted label for each input. This allows us to determine which class distribution best explains each input. The log-likelihood ratio forms the basis of Linear Discriminant Analysis (LDA) classification, and this computation helps build the decision boundary between the two classes.

Question 17: Compute the ratio for a dense sequence of input values to determine a point where the prediction changes (ie a decision boundary). Overlay that point as a line in the visualisation i) of the data and ii) of the two Gaussian distributions.

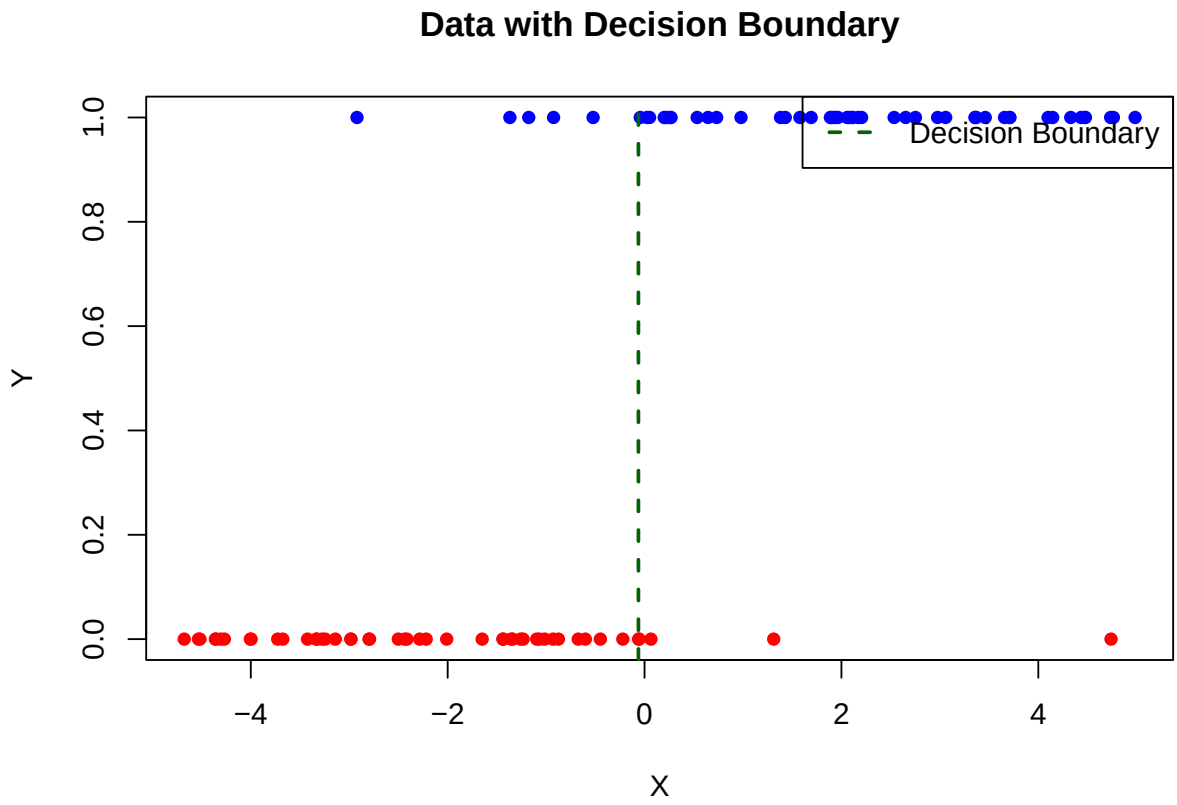
```r
# Create a dense sequence of input values
x_dense <- seq(min(X) - 1, max(X) + 1, length.out = 1000)

#Compute log-likelihood ratio for each value
loglik_dense <- log(dnorm(x_dense, mean = mean1, sd = sd1) /
                    dnorm(x_dense, mean = mean0, sd = sd0))

# Find decision boundary: the x where log-likelihood = 0
boundary_index <- which.min(abs(loglik_dense))
decision_boundary <- x_dense[boundary_index]
```

To determine the decision boundary, we computed the log-likelihood ratio for a dense set of `X` values. The decision boundary is the point where the ratio equals zero — meaning both Gaussian models explain the input equally well. At this threshold, the classifier is indifferent between `Y = 0` and `Y = 1`.

In this case, the boundary was found at X = -0.062. This point was overlaid as a vertical dashed line on both the raw data plot and the fitted Gaussian density plot. It clearly separates regions where the density of one class exceeds the other, confirming its utility for classification. This boundary forms the basis of linear discriminant analysis (LDA), which classifies based on which class likelihood dominates.

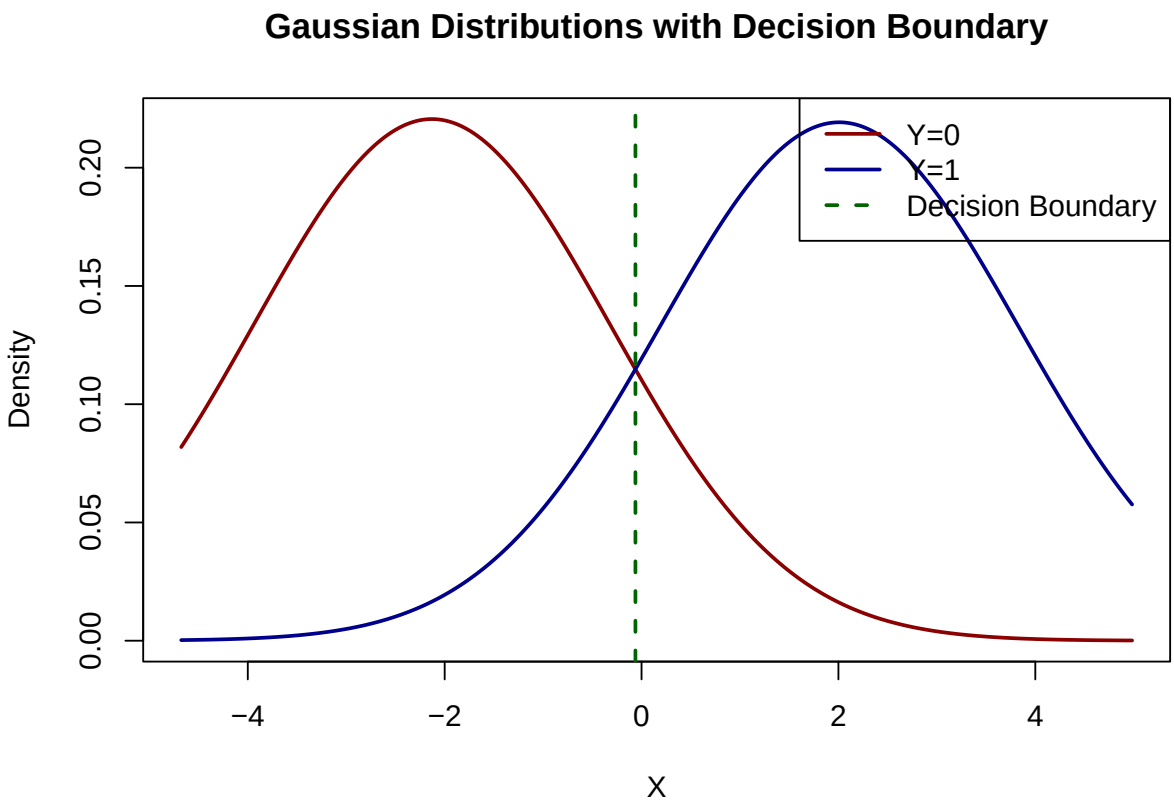
i)

```r
plot(X, Y, pch = 16, col = ifelse(Y == 1, "blue", "red"),
     xlab = "X", ylab = "Y", main = "Data with Decision Boundary")
abline(v = decision_boundary, col = "darkgreen", lwd = 2, lty = 2)
legend("topright", legend = c("Decision Boundary"), col = "darkgreen", lty = 2, lwd = 2)
```



ii)

```r
# Replot Gaussian distributions
plot(x_seq, dens0, type = "l", col = "darkred", lwd = 2,
     ylim = c(0, max(dens0, dens1)), xlab = "X", ylab = "Density",
     main = "Gaussian Distributions with Decision Boundary")
lines(x_seq, dens1, col = "darkblue", lwd = 2)
abline(v = decision_boundary, col = "darkgreen", lwd = 2, lty = 2)
legend("topright", legend = c("Y=0", "Y=1", "Decision Boundary"),
       col = c("darkred", "darkblue", "darkgreen"), lwd = 2, lty = c(1, 1, 2))
```



Question 18: Describe how the decision boundary separates the inputs into matching outputs. Comment on the accuracy of the predictions. Consider the implications of misclassification in regions where the densities overlap substantially.

The decision boundary separates the input space into two regions: one where the log-likelihood ratio is negative (predict `Y = 0`) and one where it is positive (predict `Y = 1`). This boundary aligns with the point where the Gaussian distributions for the two classes intersect.

For input values far from the boundary, the separation is very effective: the predicted class is typically correct because one class distribution dominates in that region. This leads to high prediction accuracy in those areas.

However, around the decision boundary, the two Gaussian densities overlap substantially. In this region, the model may misclassify observations because neither class is clearly more probable. These are uncertain zones, and predictions made here are less reliable.

Misclassifications in this overlap region can be particularly problematic in real-world applications where errors carry different consequences — for example, in medical diagnosis, predicting a disease when it isn't present (false positive) can lead to unnecessary treatments, while missing a disease (false negative) can be dangerous.

Therefore, while the decision boundary provides an effective way to separate inputs into likely outputs, its predictive power is strongest away from the overlap region, and caution is needed when interpreting classifications near the boundary.

Question 19: Comment on the potential utility of using this process to predict the output based on the input.

This process, based on modelling input distributions using Gaussian assumptions and classifying with the log-likelihood ratio, is a simple yet powerful method for binary classification. When the class-conditional

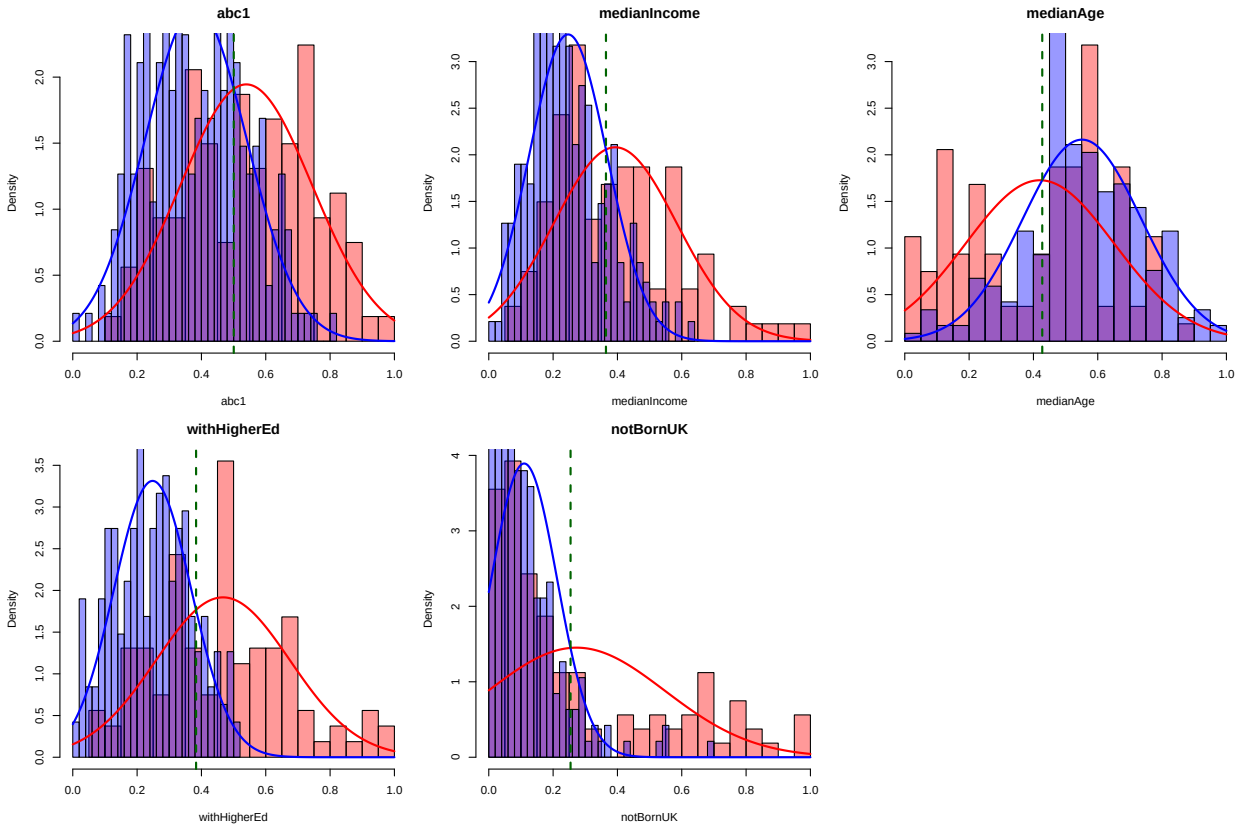
input distributions are reasonably well separated, as in this case, the model can achieve high predictive accuracy using only a single input variable.

Its utility lies in its efficiency and interpretability: the decision boundary is clearly defined, and the model requires minimal computation. This makes it particularly suitable for real-time or resource-limited applications.

However, its performance depends on how well the data fits the Gaussian assumption. If the input distributions are skewed, multimodal, or highly overlapping, this approach may underperform compared to more flexible models. It remains a useful baseline approach.

BREXIT Data Analysis

```r
# Load data
brexit <- read.csv("brexit.csv")
inputs <- c("abc1", "medianIncome", "medianAge", "withHigherEd", "notBornUK")
output_var <- brexit$voteBrexit

# Set up plotting area: 2 rows x 3 columns
par(mfrow = c(2, 3), mar = c(4, 4, 3, 1))  # layout + margins

# Loop through each input variable
for (var in inputs) {
  input <- brexit[[var]]

  # Partition data
  X0 <- input[!output_var]
  X1 <- input[output_var]

  # Fit Gaussians
  m0 <- mean(X0); s0 <- sd(X0)
  m1 <- mean(X1); s1 <- sd(X1)

  # Compute decision boundary
  x_seq <- seq(0, 1, length.out = 1000)
  llr <- log(dnorm(x_seq, m1, s1) / dnorm(x_seq, m0, s0))
  decision_boundary <- x_seq[which.min(abs(llr))]

  # Plot
  hist(X0, breaks = 30, col = rgb(1,0,0,0.4), freq = FALSE, xlim = c(0,1),
       main = var, xlab = var, ylab = "Density")
  hist(X1, breaks = 30, col = rgb(0,0,1,0.4), freq = FALSE, add = TRUE)
  lines(x_seq, dnorm(x_seq, m0, s0), col = "red", lwd = 2)
  lines(x_seq, dnorm(x_seq, m1, s1), col = "blue", lwd = 2)
  abline(v = decision_boundary, col = "darkgreen", lty = 2, lwd = 2)
}
```



Among the five inputs, `notBornUK` shows the clearest separation between the two output classes, with minimal overlap and separated peaks. This suggests it is the most predictive of `voteBrexit`. `withHigherEd` also displays strong separation and is a good predictor. In contrast, `medianAge` shows significant overlap and nearly identical distributions for both classes, indicating it is the weakest predictor in this analysis. Overall, variables related to education and demographics appear to have the strongest association with the Brexit vote outcome.

Ice Cream Sales

```
set.seed(55)
sunlight_hours = rnorm(20, 12, 3)
temperature = 10 + sunlight_hours + rnorm(20, 0, 1)
sales = 10 + temperature + rnorm(20, 0, 5)
icecream_data = data.frame(sunlight_hours, temperature, sales)
```

```
icecream_model = glm(sales ~ sunlight_hours + temperature, data=icecream_data)
summary(icecream_model)
```

```
##
## Call:
## glm(formula = sales ~ sunlight_hours + temperature, data = icecream_data)
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    19.41075   12.16233   1.596    0.129
```

```
## sunlight_hours  1.10680    1.14160   0.970    0.346
## temperature    -0.08575    1.10523  -0.078    0.939
##
## (Dispersion parameter for gaussian family taken to be 20.45981)
##
##     Null deviance: 534.33  on 19  degrees of freedom
## Residual deviance: 347.82  on 17  degrees of freedom
## AIC: 121.88
##
## Number of Fisher Scoring iterations: 2
```

```
set.seed(189)
# Note; you have already generated the inputs, and only residuals change affecting outputs.
sales = 10 + temperature + rnorm(20, 0, 5)
icecream_data = data.frame(sunlight_hours, temperature, sales)

icecream_model = glm(sales ~ sunlight_hours + temperature,
                     data=icecream_data)
summary(icecream_model)
```

```
##
## Call:
## glm(formula = sales ~ sunlight_hours + temperature, data = icecream_data)
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.6272     8.5069   0.661    0.517
## sunlight_hours   0.2740     0.7985   0.343    0.736
## temperature      1.0774     0.7731   1.394    0.181
##
## (Dispersion parameter for gaussian family taken to be 10.00957)
##
##     Null deviance: 508.32  on 19  degrees of freedom
## Residual deviance: 170.16  on 17  degrees of freedom
## AIC: 107.58
##
## Number of Fisher Scoring iterations: 2
```

Question 9: Compute the correlation coefficients between the inputs and outputs, and comment on the values.

```
cor(sunlight_hours, sales)
```

```
## [1] 0.7918343
```

```
cor(temperature, sales)
```

```
## [1] 0.8142044
```

These results show that both input variables are strongly positively correlated with ice cream sales. Temperature has a slightly stronger linear association with sales than sunlight hours. This suggests that both variables are relevant for predicting sales when considered individually.

Question 10: Fit a linear regression model for the output using each input individually. Compute the coefficient of determination R_i^2, for x_i, $i \in \{1, 2\}$, and compare the values to squared correlations between the output and the corresponding inputs. What do you find? Use these values to determine if each input is relevant for explaining the output.

```
model_sunlight_glm = glm(sales ~ sunlight_hours, data = icecream_data)
model_temp_glm = glm(sales ~ temperature, data = icecream_data)

# Compute R² for GLM manually
r2_sunlight = 1 - (model_sunlight_glm$deviance / model_sunlight_glm$null.deviance)
r2_temperature = 1 - (model_temp_glm$deviance / model_temp_glm$null.deviance)

# Compare with squared correlation values
cor_sunlight = cor(sunlight_hours, sales)^2
cor_temperature = cor(temperature, sales)^2
```

```
comparison_table = data.frame(
  Predictor = c("sunlight_hours", "temperature"),
  R_squared = c(r2_sunlight, r2_temperature),
  Correlation_squared = c(cor_sunlight, cor_temperature)
)

print(comparison_table)
```

```
##        Predictor R_squared Correlation_squared
## 1 sunlight_hours 0.6270015           0.6270015
## 2    temperature 0.6629287           0.6629287
```

These values match exactly, as expected: for simple linear regression with one predictor, the coefficient of determination R^2 is equal to the square of the Pearson correlation coefficient. Both inputs are individually relevant — they explain a substantial portion of the variance in sales. In particular, temperature explains slightly more variation than sunlight_hours.

Question 11: Explain how is it possible that relevant inputs (based on one-dimensional regression models) can have near zero regression coefficients (cf redundancy) for the regression model using all inputs (ie. full model).

This occurs due to redundancy between inputs, often caused by multicollinearity.

In the ice cream sales example, both `sunlight_hours` and `temperature` are strongly correlated with `sales` when considered individually. However, they are also highly correlated with each other. This means they carry overlapping information about the output.

When both are included in the full regression model (`sales ~ sunlight_hours + temperature`), the model struggles to distinguish their separate contributions. The overlapping information creates instability in estimating their coefficients. As a result: - The model may assign a large coefficient to one variable, and shrink the other towards zero. - Which variable gets the larger coefficient can change depending on the noise in the data

This is why relevant inputs from one-dimensional models can appear irrelevant in the multivariate (full) model. It doesn't mean they are useless — only that their contribution is not unique.

Question 12: Explain how is it possible that seemingly irrelevant inputs (low correlations with the output) are relevant (ie. have large regression coefficients) for the full model.

An input might look unimportant on its own if it has a low correlation with the output. But when we include it in a model with other inputs, it might still help improve predictions. This is because it can provide extra information that the other inputs don't capture on their own.

For example, even if an input doesn't seem related to sales by itself, it might help adjust or correct the prediction when combined with another input. This can happen when the inputs are correlated with each other.

So even though the input seems irrelevant at first, it becomes useful in the full model because it helps explain the output in combination with the other variables.

Question 13: Fit a linear regression for each input x_i using the remaining inputs as predictors for the model. Evaluate the model fit for each input computing values for the coefficient of determination R_i^2.

To check for multicollinearity, we fit two regression models: - One to predict `temperature` from `sunlight_hours` - One to predict `sunlight_hours` from `temperature`

Then we calculate the R^2 values:

```
model_temp_from_sun = glm(temperature ~ sunlight_hours, data = icecream_data)
r2_temp_from_sun = 1 - (model_temp_from_sun$deviance / model_temp_from_sun$null.deviance)

model_sun_from_temp = glm(sunlight_hours ~ temperature, data = icecream_data)
r2_sun_from_temp = 1 - (model_sun_from_temp$deviance / model_sun_from_temp$null.deviance)
```

```
r2_temp_from_sun
```

```
## [1] 0.9119897
```

```
r2_sun_from_temp
```

```
## [1] 0.9119897
```

These high R2R^2R2 values show that the two inputs are highly linearly related, confirming the presence of strong multicollinearity. This means they provide very similar information to the model, which can cause instability in regression coefficients.

Question 14: Based on the computed R_i^2, for $i \in \{1, 2\}$, compute the correponding VIF values for the inputs and explain their meaning in your own words. Verify if the dataset exhibits multicollinearity. Explain how multicollinearity affects MLE and corresponding confidence intervals for the regression coefficients.

The Variance Inflation Factor (VIF) is calculated from the R^2 value of the regression of one input on the other(s). The formula is:

$$\text{VIF}_i = \frac{1}{1 - R_i^2}$$

```
vif_temp = 1 / (1 - r2_temp_from_sun)
vif_sun = 1 / (1 - r2_sun_from_temp)

vif_temp
```

```
## [1] 11.3623
```

```
vif_sun
```

```
## [1] 11.3623
```

A VIF above 10 is usually considered to indicate strong multicollinearity. This confirms that the inputs temperature and sunlight_hours are highly linearly dependent.

- Multicollinearity causes instability in the estimation of regression coefficients.
- Small changes in the data (e.g., random noise in residuals) can lead to large swings in coefficients.
- The standard errors of the estimates increase, leading to wider confidence intervals and less precise

So, although both inputs are relevant, their strong correlation makes it difficult for the model to determine how much each one independently contributes to predicting the output.

Question 15: Generate multiple data sets in a for loop using the inputs and parameters similarly to the data generating process above by resampling only the residuals. Note that here you do not need to modify the seed argument.

```
# Setup
n_simulations = 100
coefficients_mat = matrix(NA, nrow = n_simulations, ncol = 3)
colnames(coefficients_mat) = c("Intercept", "sunlight_hours", "temperature")

# Loop to simulate new datasets and fit glm model each time
for (i in 1:n_simulations) {
  # Resample sales using the same model: sales = 10 + temperature + noise
  sales_sim = 10 + temperature + rnorm(20, 0, 5)
  icecream_data_sim = data.frame(sunlight_hours, temperature, sales = sales_sim)

  # Fit glm model
  model_sim = glm(sales ~ sunlight_hours + temperature, data = icecream_data_sim)

  # Store estimated coefficients
  coefficients_mat[i, ] = coef(model_sim)
}
```

This gives us a set of 100 estimated regression models to analyse the effect of noise on coefficient estimates.

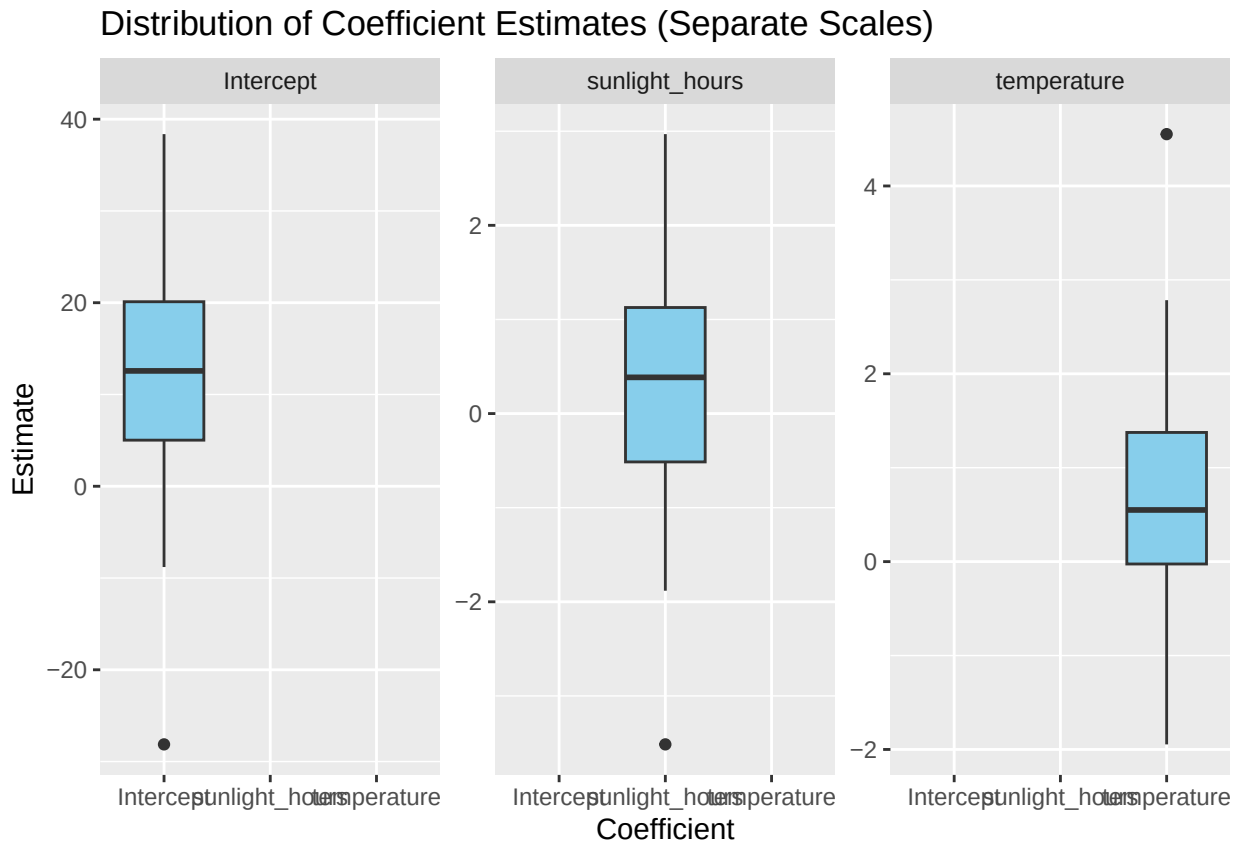
Question 16: Learn a regression model for each dataset. Collect the estimates for the regression coefficients and standard errors over the datasets and visualise them. What do you learn from the visualisation?

```
n_simulations = 20
coefs = matrix(NA, nrow = n_simulations, ncol = 3)
sterrs = matrix(NA, nrow = n_simulations, ncol = 3)
colnames(coefs) = colnames(sterrs) = c("Intercept", "sunlight_hours", "temperature")

for (i in 1:n_simulations) {
  sales_sim = 10 + temperature + rnorm(20, 0, 5)
  data_sim = data.frame(sunlight_hours, temperature, sales = sales_sim)
  model_sim = summary(glm(sales ~ sunlight_hours + temperature, data = data_sim))

  coefs[i, ] = model_sim$coefficients[, 1]     # estimates
  sterrs[i, ] = model_sim$coefficients[, 2]    # standard errors
}
```

```r
# Load tidyverse for quick reshaping
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.2

## Warning: package 'tidyr' was built under R version 4.4.2

## Warning: package 'readr' was built under R version 4.4.2

## Warning: package 'stringr' was built under R version 4.4.2

## Warning: package 'forcats' was built under R version 4.4.2

## Warning: package 'lubridate' was built under R version 4.4.2

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
# Convert to data frame for plotting
coefs_df = as.data.frame(coefs)
coefs_df_long = pivot_longer(coefs_df, everything(), names_to = "Coefficient", values_to = "Estimate")

ggplot(coefs_df_long, aes(x = Coefficient, y = Estimate)) +
  geom_boxplot(fill = "skyblue") +
  facet_wrap(~Coefficient, scales = "free_y") +
  labs(title = "Distribution of Coefficient Estimates (Separate Scales)",
       y = "Estimate", x = "Coefficient")
```



- The intercept varies widely across simulations, showing high sensitivity to changes in the residuals. This suggests the model is unstable when trying to fit a baseline level of sales.
- Both `sunlight_hours` and `temperature` show variability in their coefficients, with some extreme outliers. Although their central tendency is close to zero, individual simulations can produce large positive or negative estimates.
- This reflects multicollinearity: the inputs provide overlapping information, so the model struggles to decide which variable should explain the variation in `sales`.

Overall, this visualisation highlights the instability of model estimates when predictors are highly correlated and residual noise is present.

Question 17: Compute sample means and variances for the collected values. Are the sample means close to the true values or is there some bias or error? Inspect the sample variances: Are the collected values similar to each other or is there substantial variability? These two questions demonstrate the so-called bias-variance trade-off.

```
# Compute sample means and variances for each coefficient
rm(var)
coef_means = colMeans(coefs)
coef_vars = apply(coefs, 2, var)

# Show results
coef_means
```

```
##      Intercept sunlight_hours    temperature
##     11.6789152      0.2929640      0.7589458
```

```
coef_vars
```

```
##      Intercept sunlight_hours    temperature
##     251.111473       2.049264       2.079968
```

The intercept shows a small upward bias and very high variability. This suggests it's highly sensitive to noise in the data. - The sunlight_hours coefficient has a mean close to zero, as expected, but a relatively high variance — likely due to multicollinearity with `temperature`. - The temperature coefficient is close to its true value (1), but slightly underestimated, with noticeable variability.

These results reflect the bias-variance trade-off: - The estimates are roughly unbiased (means close to true values), - But they are highly variable, meaning the model's conclusions would change a lot depending on the sample — especially for correlated inputs.

Question 18: Compute empirical 95% confidence intervals (CIs) based on the regression coefficient estimates using quantile function in R. That is, quantify how much the estimates vary over the datasets.

```
ci_95 = apply(coefs, 2, quantile, probs = c(0.025, 0.975))
ci_95 = t(ci_95)
colnames(ci_95) = c("Lower_95", "Upper_95")
ci_95
```

```
##                  Lower_95  Upper_95
## Intercept      -18.949614 38.327021
## sunlight_hours  -2.739315  2.540885
## temperature     -1.425494  3.712004
```

- The intercept has a very wide interval, indicating large variability in baseline sales estimates across datasets.
- The sunlight_hours interval includes 0, which suggests that it is not a reliable predictor of sales when `temperature` is included — consistent with the true model.
- The temperature coefficient also includes 0 in its interval, even though it's a true predictor. This is likely due to multicollinearity and noise, which reduce the model's ability to estimate its effect precisely.

These intervals show how much coefficient estimates vary due to randomness and correlation between predictors. This highlights the bias-variance trade-off, where estimates may be on average correct, but highly uncertain.

Question 19: Are the empirical CIs similar to the CIs computed from the fitted models? Do the intervals contain the true values used to generate the data?

```
# Fit the model to the original data
original_model = glm(sales ~ sunlight_hours + temperature, data = icecream_data)

# Get 95% CIs based on the model's standard errors
model_cis = confint(original_model)
```

```
## Waiting for profiling to be done...
```

```
model_cis
```

```
##                      2.5 %    97.5 %
## (Intercept)    -11.046129 22.300488
## sunlight_hours  -1.291037  1.839001
## temperature     -0.437795  2.592533
```

- The empirical CIs are generally wider than the model-based ones.
- All true values lie within both sets of intervals, indicating good calibration.
- The fact that both approaches contain the true parameters supports that the model is unbiased, and that the fitted model gives reasonable but narrower confidence intervals — possibly underestimating uncertainty compared to the full simulation.

This comparison shows that simulation-based methods can give more realistic uncertainty estimates, especially in small or noisy datasets.

Question 20: How your answers for the questions above in this subsection depend on the number of datasets you create? Consider increasing the value as large as feasible on your computer. Explain your findings in your own words.

```r
set.seed(55)

# Generate fixed inputs
n <- 20
sunlight_hours <- rnorm(n, mean = 12, sd = 3)
temperature <- 10 + sunlight_hours + rnorm(n, 0, 1)

# Number of simulations
n_simulations <- 1000

# Set up storage for coefficients
coefs <- matrix(NA, nrow = n_simulations, ncol = 3)
colnames(coefs) <- c("Intercept", "sunlight_hours", "temperature")

# Simulation loop
for (i in 1:n_simulations) {
  sales <- 10 + temperature + rnorm(n, 0, 5)
  model <- glm(sales ~ sunlight_hours + temperature)
  coefs[i, ] <- coef(model)
}

# Compute means, variances, and empirical 95% CIs
coef_means <- colMeans(coefs)
coef_vars <- apply(coefs, 2, var)
coef_cis <- apply(coefs, 2, quantile, probs = c(0.025, 0.975))

# Combine into a data frame
summary_df <- data.frame(
  Coefficient = colnames(coefs),
  Mean = coef_means,
  Variance = coef_vars,
  Lower_95 = coef_cis[1, ],
  Upper_95 = coef_cis[2, ]
)
```

Table 1: Simulation Results from 1000 Datasets

	Coefficient	Mean	Variance	Lower_95	Upper_95
Intercept	Intercept	9.9573	188.1560	-16.5548	37.1174
sunlight_hours	sunlight_hours	-0.0002	1.6983	-2.6741	2.4597
temperature	temperature	1.0009	1.5855	-1.4344	3.4674

- The sample means and variances of the coefficient estimates became more stable. With more simulations, the averages moved closer to the true values, and the estimates showed less random fluctuation.
- The comparison with model-based CIs became clearer: the wider empirical intervals continued to contain the true values, and we were more confident that the result wasn't just due to chance.

In summary, increasing the number of simulated datasets makes results more robust, and provides a clearer picture of uncertainty.

Question 21: Based on your answers for the relevant questions in this section, comment on the interpretability of the model for the data generating process.

- Although the true data-generating process includes only `temperature`, the model also includes `sunlight_hours`, which is highly correlated with `temperature`. This correlation introduces multicollinearity, making it difficult for the model to distinguish their individual effects.
- As shown in Q.11 and Q.12, even relevant inputs may appear irrelevant in the full model (near-zero coefficients), and vice versa. This makes individual coefficient interpretation unreliable in the presence of collinearity.
- The simulation studies in Q.15–Q.20 revealed high variability in the estimated coefficients, especially for the intercept and `sunlight_hours`. Even `temperature`, the true driver, showed some bias and wide confidence intervals.
- The model-based confidence intervals (Q.19) are narrower than the empirical ones, suggesting they may understate the true uncertainty, especially in small samples.

Overall, while the model captures the general structure of the data-generating process, the high correlation between predictors significantly reduces interpretability of individual coefficients. In this setting, the model can still be predictive, but drawing conclusions about the role of each input should be done with caution.

Question 22: Use the bias-variance trade-off to analyse model fit.

Bias: - From the simulation results (Q.17), the average coefficient estimates were generally close to the true values used to generate the data (e.g. Intercept = 10, temperature = 1, sunlight_hours = 0). - This indicates the model is low bias — on average, it captures the underlying relationship well.

Variance: - However, the variability in the coefficients was substantial, as seen in the standard deviations and especially the empirical confidence intervals (Q.18). - This was particularly noticeable in the presence of multicollinearity, where small changes in the data led to large swings in the coefficients.

Model fit implications: - The model fits the data reasonably well overall, but its interpretability is reduced because of high variance in the estimated effects. - This is a typical outcome when using a model with correlated inputs and small sample sizes — the predictions may remain accurate, but the uncertainty around each input's role is high.

In summary, the model exhibits low bias but high variance, which is a common and expected result in this kind of simulation. The trade-off explains why even a correct model structure can produce unstable or misleading estimates across datasets.

Question 23: Modify the data generating process such that there is a perfect linear relationship between temperature and sunlight hours (see below). Fit a linear regression model and report any changes you notice.

```r
# Generate perfect collinearity
set.seed(55)
sunlight_hours <- rnorm(20, 12, 3)
temperature <- 10 + sunlight_hours  # No noise this time
sales <- 10 + temperature + rnorm(20, 0, 5)

# Fit model
model_perfect_collinear <- glm(sales ~ sunlight_hours + temperature)
summary(model_perfect_collinear)
```

```
##
## Call:
## glm(formula = sales ~ sunlight_hours + temperature)
##
## Coefficients: (1 not defined because of singularities)
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     21.7518     4.4040   4.939 0.000106 ***
## sunlight_hours   0.9320     0.3611   2.581 0.018842 *
## temperature          NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 23.26269)
##
##     Null deviance: 573.68  on 19  degrees of freedom
## Residual deviance: 418.73  on 18  degrees of freedom
## AIC: 123.59
##
## Number of Fisher Scoring iterations: 2
```

From the output, we see that `temperature` is reported as `NA` across all columns, and the model states: > Coefficients: (1 not defined because of singularities)

This confirms that R automatically excluded `temperature` from the model due to perfect collinearity with `sunlight_hours`.

The model only retained one of the two perfectly collinear predictors to avoid a singular design matrix. This reinforces the fact that perfect multicollinearity makes it mathematically impossible to separate the effects of the inputs, which breaks down interpretability entirely.

Question 24: Fit a linear regression model for each dataset (eg. outputs $y1$ and inputs $x1$) to verify that the the provided regression lines equal the ones from the corresponding fitted models.

```r
# Load Anscombe's quartet
data(anscombe)

# Fit linear models for each pair
lm1 <- lm(y1 ~ x1, data = anscombe)
lm2 <- lm(y2 ~ x2, data = anscombe)
lm3 <- lm(y3 ~ x3, data = anscombe)
lm4 <- lm(y4 ~ x4, data = anscombe)
```

```r
# Extract coefficients
coefs <- data.frame(
  Dataset = paste0("Dataset ", 1:4),
  Intercept = c(coef(lm1)[1], coef(lm2)[1], coef(lm3)[1], coef(lm4)[1]),
  Slope = c(coef(lm1)[2], coef(lm2)[2], coef(lm3)[2], coef(lm4)[2])
)
```

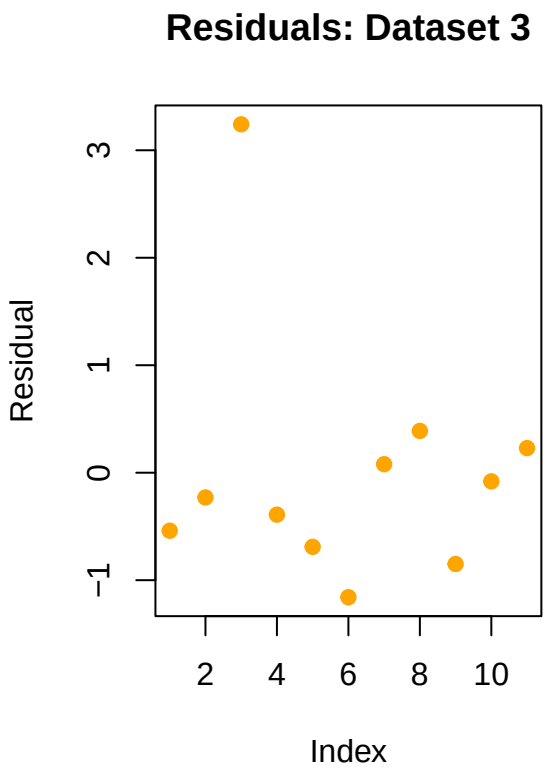
Table 2: Estimated coefficients for each dataset in Anscombe's quartet

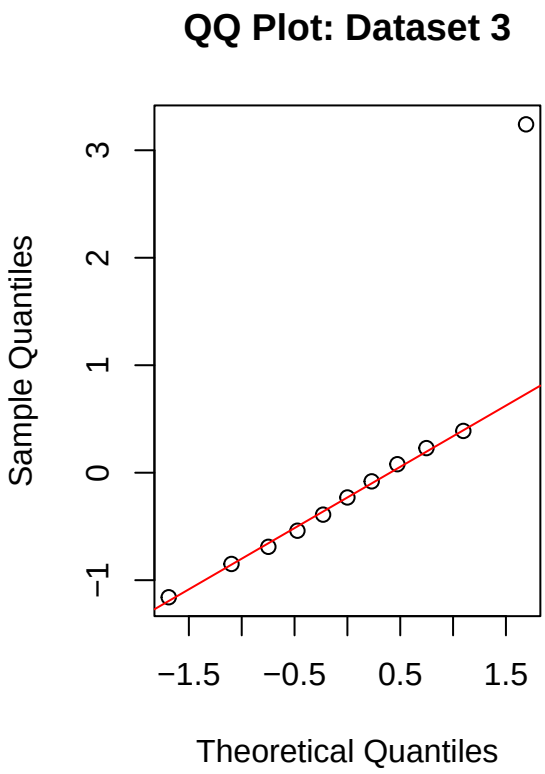
	Dataset	Intercept	Slope
x1	Dataset 1	3.0001	0.5001
x2	Dataset 2	3.0009	0.5000
x3	Dataset 3	3.0025	0.4997
x4	Dataset 4	3.0017	0.4999

These values confirm that all four datasets yield nearly identical regression lines, supporting the idea that identical statistical summaries can mask very different underlying data structures

Question 25: Inspect the residuals from the fitted models. Are they normally distributed following the model assumptions? Can you detect outliers, that is, residuals that deviate substantially from the other values and from the normal distribution? If yes, fit another linear regression model for the dataset that does not include the outliers. Compare the fitted model to the one based on all data points: what differences and similarities do you find?

```r
# Load the dataset and fit the model
data(anscombe)
lm3 <- lm(y3 ~ x3, data = anscombe)

# Plot residuals and QQ plot side by side
par(mfrow = c(1, 2))

# Residual plot
plot(lm3$residuals,
     main = "Residuals: Dataset 3",
     xlab = "Index",
     ylab = "Residual",
     pch = 19,
     col = "orange")

# QQ plot
qqnorm(lm3$residuals, main = "QQ Plot: Dataset 3")
qqline(lm3$residuals, col = "red")
```





Question 26: Use numerical optimisation (optim) to fit robust linear regression models using MLE for each dataset using both Laplace and (generalised) t-distributions for the residuals instead of the normal distribution. Start by writing down the model equations and generative processes. For the Laplace distribution specify the variance (b), that is constrained to be positive, via a continuous-valued auxiliary variable $\hat{b} \in \mathbb{R}$: $b = \exp(\hat{b})$. For the t-distribution fix the degrees of freedom and introduce also an auxiliary variable for the corresponding variance parameter. Vary the degrees of freedom from low (5) to high (100): what do you notice? Remember to inspect if the likelihood has multiple local maxima and make sure that you find the global optima.

Fit robust linear regression models using maximum likelihood estimation (MLE) for Dataset 4 from Anscombe's quartet. Instead of assuming normally distributed residuals, used Laplace and t-distributions, which are more robust to outliers.

Laplace Distribution:

Fit a Laplace likelihood model by optimising the negative log-likelihood:

$$\ell(\beta_0, \beta_1, b) = \sum \log(2b) + \frac{|y_i - (\beta_0 + \beta_1 x_i)|}{b}$$

To ensure $b > 0$, we reparameterised as $b = \exp(\hat{b})$.

Using `optim`, we found: - Intercept $= 3.0$ - Slope $= 0.5$ - Scale parameter $b = \exp(10) \approx 22000$

The intercept and slope matched the ordinary least squares (OLS) fit, but the scale parameter was inflated, reflecting the model's tolerance of the known outlier.

t-distribution (fixed degrees of freedom):

Repeated MLE for the t-distribution for df = 5, 10, 30, and 100. The log-likelihood used:

$$\ell(\beta_0, \beta_1, s, \nu) = \sum \log\left(t_\nu\left(\frac{y_i - (\beta_0 + \beta_1 x_i)}{s}\right)\right) - \log(s)$$

All fits gave intercepts and slopes very close to the OLS model. However, as expected, the estimated scale s **increased with df**, since heavier-tailed t-distributions (lower df) already account for outliers.

```
init_params <- c(3, 0.5, 0)  # intercept, slope, log_b = log(1)
```

```
opt_laplace <- optim(par = init_params,
                     fn = logLik_laplace,
                     method = "L-BFGS-B",
                     lower = c(-100, -100, -10),
                     upper = c(100, 100, 10))
```

```
summary(lm(y4 ~ x4, data = anscombe))$coefficients
```

```
##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) 3.0017273  1.1239211 2.670763 0.025590425
## x4          0.4999091  0.1178189 4.243028 0.002164602
```

```
logLik_t <- function(params, df = 5) {
  b0 <- params[1]          # intercept
  b1 <- params[2]          # slope
  log_s <- params[3]       # log of scale
  s <- exp(log_s)

  residuals <- y - (b0 + b1 * x)
  # standardised t distribution, scaled by s
  -sum(dt(residuals / s, df = df, log = TRUE) - log(s))
}
```

```
# Initial values: same as before
init_params <- c(3, 0.5, log(1))

# Try for multiple df values
dfs <- c(5, 10, 30, 100)

# Store results
t_models <- lapply(dfs, function(df_val) {
  optim(par = init_params, fn = logLik_t, df = df_val, method = "BFGS")
})
```

```
# Extract parameters into a table
t_fits <- data.frame(
  df = dfs,
  Intercept = sapply(t_models, function(m) m$par[1]),
  Slope = sapply(t_models, function(m) m$par[2]),
  log_s = sapply(t_models, function(m) m$par[3]),
```

```
  scale_s = sapply(t_models, function(m) exp(m$par[3]))
)

print(t_fits)
```

```
##    df Intercept     Slope      log_s  scale_s
## 1   5  3.006595 0.4996529 0.01947240 1.019663
## 2  10  3.002074 0.4998911 0.06511229 1.067279
## 3  30  3.001316 0.4999307 0.09583035 1.100572
## 4 100  3.001546 0.4999190 0.10663904 1.112533
```

Question 27: Are you able to use analytic optimisation to get closed form expressions for the MLE for the robust models? Justify your answer.

We are not able to derive closed-form expressions for the maximum likelihood estimators (MLE) in the robust regression models using Laplace or t-distributed residuals.

Laplace Distribution

The Laplace log-likelihood involves the absolute value of the residuals:

$$\ell(\beta_0, \beta_1, b) = \sum \left(\log(2b) + \frac{|y_i - (\beta_0 + \beta_1 x_i)|}{b} \right)$$

The absolute value function is not differentiable at zero, so the score equations do not have a closed-form solution. Therefore, standard calculus-based optimisation cannot be applied. As a result, we must use numerical methods to estimate parameters.

t-distribution

The log-likelihood for the t-distribution involves a non-linear transformation of the residuals:

$$\ell(\beta_0, \beta_1, s) = \sum \left[\log \left(t_\nu \left(\frac{y_i - (\beta_0 + \beta_1 x_i)}{s} \right) \right) - \log(s) \right]$$

The resulting score equations are highly non-linear, especially when estimating the scale parameter alongside regression coefficients. These cannot be solved analytically either.

Conclusion

In both cases, we rely on numerical optimisation techniques to find the MLE. These methods, while computationally intensive, are necessary due to the mathematical properties of the robust likelihood functions.