DA_HW2

Howard

2025-03-05

Problem 1

Consider the simple linear regression model

$$y = c + \beta_1 x + \epsilon,$$

where the intercept c is known.

(a) Derivation of the Least-Squares Estimator of β_1

Minimize the sum of squared errors:

$$RSS(\beta_1) = \sum_{i=1}^{n} \Big(y_i - c - \beta_1 x_i\Big)^2.$$

Differentiate $RSS(\beta_1)$ with respect to β_1 and set the derivative to zero:

$$\frac{\partial RSS}{\partial \beta_1} = -2 \sum_{i=1}^{n} x_i \Big(y_i - c - \beta_1 x_i\Big) = 0.$$

This implies:

$$\sum_{i=1}^{n} x_i (y_i - c) - \beta_1 \sum_{i=1}^{n} x_i^2 = 0.$$

Solving for β_1, we obtain:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} x_i (y_i - c)}{\sum_{i=1}^{n} x_i^2}.$$

This estimator is reasonable because it essentially regresses the centered response $y_i - c$ on x_i, thereby isolating the effect of x on y.

(b) Variance of $\hat{\beta}_1$

Given the model $y_i = c + \beta_1 x_i + \epsilon_i$ and noting that

$$\hat{\beta}_1 = \beta_1 + \frac{\sum_{i=1}^{n} x_i \epsilon_i}{\sum_{i=1}^{n} x_i^2},$$

and assuming that ϵ_i are independent with $\mathrm{Var}(\epsilon_i) = \sigma^2$, the variance of $\hat{\beta}_1$ is:

$$\mathrm{Var}(\hat{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^{n} x_i^2}.$$

(c) Confidence Interval for β_1

Assuming the errors $\epsilon_i \sim N(0, \sigma^2)$, the estimator $\hat{\beta}_1$ is normally distributed. However, since σ^2 is usually unknown, we estimate it by:

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} \Big(y_i - c - \hat{\beta}_1 x_i\Big)^2,$$

where the degrees of freedom is $n - 1$ (because only β_1 is estimated).

Then the statistic

$$t = \frac{\hat{\beta}_1 - \beta_1}{\sqrt{\hat{\sigma}^2 / \sum_{i=1}^n x_i^2}}$$

follows a t-distribution with $n-1$ degrees of freedom. Thus, a 100$(1-\alpha)\%$ confidence interval for β_1 is given by:

$$\hat{\beta}_1 \pm t_{n-1,\,\alpha/2} \sqrt{\frac{\hat{\sigma}^2}{\sum_{i=1}^n x_i^2}},$$

where $t_{n-1,\alpha/2}$ is the critical value from the t-distribution with $n-1$ degrees of freedom.

Comparison with the Case When the Intercept Is Unknown

When the intercept is unknown, the least-squares estimator of β_1 is

$$\tilde{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2},$$

with variance

$$\mathrm{Var}(\tilde{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}.$$

Because

$$\sum_{i=1}^n (x_i - \bar{x})^2 = \sum_{i=1}^n x_i^2 - n\bar{x}^2,$$

this sum is smaller than $\sum_{i=1}^n x_i^2$ (unless $\bar{x} = 0$). Consequently, the variance—and hence the confidence interval—is narrower when the intercept c is known.

Problem 2

(a)Convert the 400 images into a 400 × 2576 data matrix. Add an additional column indicating the physical gender label.

List and Read PNG Images

```
library(png)
image_dir <- "D:/DA_HW/DAHW/ORL Faces"
image_files <- list.files(image_dir, pattern = "\\.png$", full.names = TRUE)
n <- length(image_files)
if(n != 400){
  stop("Expected 400 images, but found ", n)
}
```

Convert Images to a Data Matrix

```
data_matrix <- matrix(NA, nrow = n, ncol = 46 * 56)
for (i in 1:n) {
  img <- readPNG(image_files[i])
  data_matrix[i, ] <- as.vector(t(img))
}
dim(data_matrix)
```

```
## [1]  400 2576
```

Append the Gender Label

- **Classify the g by manual.**

```r
# There are 40 subjects, each with 10 images.
subject_ids <- rep(1:40, each = 10)

# male (1) and  female (0).
subject_gender <- c(0,rep(1,6),0,1,0,1,0,rep(1,19),0,rep(1,8))

# Create a gender vector for all 400 images:
gender_labels <- subject_gender[subject_ids]

# Append the gender labels as an additional column to the data matrix.
final_data <- cbind(data_matrix, gender = gender_labels)

# Check the dimensions of the final data matrix.
dim(final_data)
```

```
## [1]  400 2577
```

(b)Regress the gender label on all the 2576 pixels? What do you observe?

Fit a Logistic Regression on All Pixels

- Because the predicted valued is binary,So I thinking using Logistic Regression is the proper regress method.

```r
# Convert final_data (matrix) to a data frame
df <- as.data.frame(final_data)
#names
pixel_names <- paste0("pixel", 1:2576)
colnames(df) <- c(pixel_names, "gender")
#Fit the logistic Regression for binary response(gender = 0/1)
model_glm <- glm(gender ~ ., data = df,family = binomial)
s <- summary(model_glm)
print(s$coefficients[1:200])
```

```
##   [1]   159.36374   781.67373  -847.97814  4132.16870 -1505.34419    78.55472
##   [7] -2335.17627   467.06981  -944.42390 -1327.05245   828.05710   220.75294
##  [13]    55.92137  -603.85901   369.55916  -509.05263   981.80508  -434.72787
##  [19]   735.32352    23.75581   762.45535  -513.81574   227.33318 -1100.63479
##  [25]  1328.37120 -1063.43422  1836.92143 -1077.17562   368.28560  -762.24699
##  [31]   772.16267  -360.81218 -1323.68696  1408.91931  -585.32422  1328.85971
##  [37] -1263.23467  2663.00645   340.72642  -743.59963  1885.03681  3493.06216
##  [43]  -502.27881   -27.99947 -1763.74385 -4162.54316 -2895.92094   143.93773
##  [49] -2175.22095  3485.58077 -2095.45336   500.75854    53.79110  1270.80878
##  [55]   411.84152  1196.93241 -1838.86515   451.48558  -348.56685  1739.46126
##  [61] -1161.34255   -28.16144  -838.39268   314.46349  1296.34321 -1659.48584
##  [67] -1417.32145  1040.81201  -533.16037   776.09260   800.26049 -2347.81683
##  [73]  2108.39340  -438.25105  -533.38410  -595.66634  -550.78434  -963.81107
##  [79]   597.94529  1765.45249    49.45760 -1522.55324   836.91697 -3491.14019
##  [85]   586.50565 -3117.07163   602.04702 -2637.38396 -1487.82409 -3737.64655
##  [91]  4424.48404  2955.70501  3022.72058 -3462.76600 -1687.81616  -762.62803
##  [97]    93.09846   257.62831  -225.02492   836.00995 -2434.72318  2049.22764
## [103]  -625.89656   176.58230 -1168.79181    83.71876   705.07207 -1016.71674
## [109]  1886.62843 -2245.08138  1391.25305  1947.52269 -1445.76777  -353.74296
## [115]  1239.45675   547.53479 -1889.73058  1533.50759  -720.98089    47.36489
## [121]   139.75813 -2951.32977  3526.81847 -1133.71428  1444.74560   -27.30775
## [127]   151.27809  -743.32376  -965.43718  2283.04942  1196.75206    63.50886
## [133]  2086.54104 -1524.22715 -1628.34090   903.73755 -1032.29360   549.30811
## [139]  3119.47475 -1867.81772  4032.06622  -245.71497   716.20895 -1865.63878
## [145]   805.68048  -897.32426  2198.53437 -1089.22153   645.74200    71.36172
## [151]  2612.50959 -2154.30703  2075.38177 -1697.00650  1039.19271  -618.84286
## [157]   659.66269 -2251.38484  1968.81497  -603.99859   730.14379 -2451.62835
## [163]  1078.50836  -319.38788   244.53508   818.12681  1860.23544 -2370.00334
## [169]  1879.95634  -710.69152  -496.29313   419.32721   150.27125 -1405.36643
## [175]  -651.86554   754.03169 -2238.81007   284.15360  -760.59163 -1681.53257
## [181]  4157.63040  -377.30888  1433.21036   165.01893 -5530.84316  5824.24475
## [187] -1577.16195   876.19625   559.72526  1478.47693 -1922.88391   822.30552
## [193]  -281.86754  -374.10463 -1145.15256 -1472.60154  -242.80459  1748.03985
## [199] -2973.69879  2442.65283
```

What do we observe

- **Convergence Problems**:With 2576 features for only 400 samples, the logistic regression might fail to converge or produce perfect separations.
- **Many Coefficient with Large Standard Errors**: Because the model is extremely high dimensional,so many parameters are unstable.
- **n << p**. With 2576 predictors and only 400 samples, the model is severely overparameterized. Many predictors are highly collinear,so many coefficients are set to NA.
- **in my opinion** This regression highlights the need for dimensionality reduction (like PCA), or regularization methods(like, Ridge, Lasso) when working with high dimensional image data.

(c) Perform the stepwise regression from a null model to find the important pixels. Plot the chosen pixels on a 46 × 56 canvas.

Stepwise Feature Selection and Pixel Plot

```
library(MASS)   # for stepAIC
library(stringr)
```

```
## Warning: 套件 'stringr' 是用 R 版本 4.2.3 來建造的
```

```
df$gender <- as.factor(df$gender)  # logistic regression expects a factor
```

Define a Null and Full Model for Logistic Regression

```r
# Null model: includes only the intercept
null_model <- glm(gender ~ 1, data = df, family = binomial)

# Full model: includes all pixels
full_model <- glm(gender ~ ., data = df, family = binomial)
```

```
## Warning: glm.fit:演算法沒有聚合
```

Forward Stepwise Selection (Up to 50 Steps)

```r
stepwise_model <- stepAIC(
  object = null_model,
  scope = list(lower = null_model, upper = full_model),
  direction = "forward",
  trace = FALSE,  # set to TRUE to see iteration details
  steps = 50      # limit the maximum number of forward steps
)

summary(stepwise_model)
```

```
##
## Call:
## glm(formula = gender ~ pixel2320 + pixel685 + pixel1558 + pixel607 +
##     pixel1701 + pixel2574 + pixel1475 + pixel2534 + pixel208 +
##     pixel452 + pixel1401 + pixel1878 + pixel2332 + pixel1049,
##     family = binomial, data = df)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -5.678e-04   2.000e-08   2.000e-08   2.000e-08   5.080e-04
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   3958.4   166857.5   0.024    0.981
## pixel2320    -3242.5   136932.9  -0.024    0.981
## pixel685     -3236.0   136104.6  -0.024    0.981
## pixel1558     2371.0   102844.8   0.023    0.982
## pixel607       947.7    44840.1   0.021    0.983
## pixel1701     1835.2    83277.9   0.022    0.982
## pixel2574    -2100.5    90737.2  -0.023    0.982
## pixel1475    -1578.0    69222.6  -0.023    0.982
## pixel2534     1392.0    60491.8   0.023    0.982
## pixel208     -1745.1    80635.0  -0.022    0.983
## pixel452      1766.3    81781.3   0.022    0.983
## pixel1401    -2049.9    91745.4  -0.022    0.982
## pixel1878     1287.7    56104.4   0.023    0.982
## pixel2332    -1484.6    69783.6  -0.021    0.983
## pixel1049     -708.0    54728.7  -0.013    0.990
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3.0142e+02  on 399  degrees of freedom
## Residual deviance: 1.5914e-06  on 385  degrees of freedom
## AIC: 30
##
## Number of Fisher Scoring iterations: 25
```

Identify the Selected Pixels

```r
selected_features <- names(coef(stepwise_model))[-1]
selected_features
```

```
##  [1] "pixel2320" "pixel685"  "pixel1558" "pixel607"  "pixel1701" "pixel2574"
##  [7] "pixel1475" "pixel2534" "pixel208"  "pixel452"  "pixel1401" "pixel1878"
## [13] "pixel2332" "pixel1049"
```

```
pixel_idx <- as.numeric(sub("pixel", "", selected_features))
pixel_idx
```

```
##  [1] 2320  685 1558  607 1701 2574 1475 2534  208  452 1401 1878 2332 1049
```

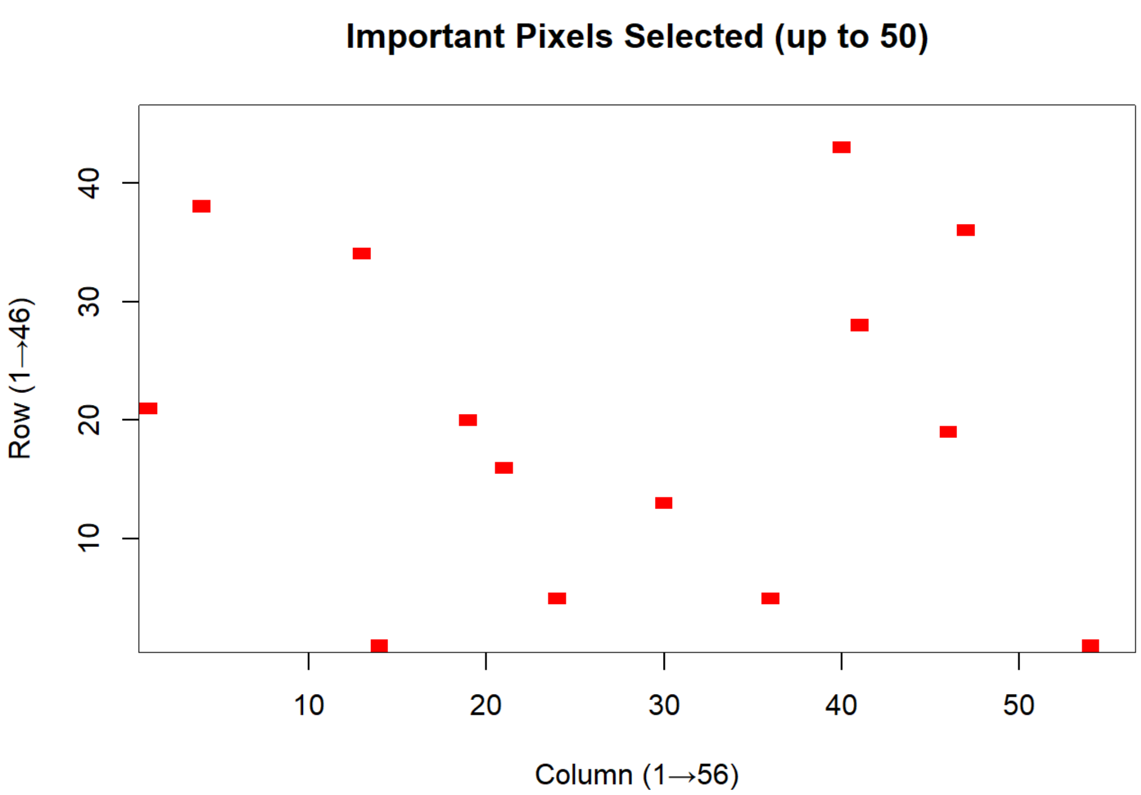
Plot the Important Pixels on a 46×56 Canvas

```
# Initialize all zeros
important_pixel_map <- rep(0, 46 * 56)

# Mark selected pixel indices as 1
important_pixel_map[pixel_idx] <- 1

# Convert to a 46x56 matrix. We assume row-major flattening.
pixel_matrix <- matrix(important_pixel_map, nrow = 46, ncol = 56, byrow = TRUE)

# We'll use base R's image() function with two colors:
image(1:56, 1:46,
      t(apply(pixel_matrix, 2, rev)),  # flip to put row=1 at top
      col = c("white", "red"),
      xlab = "Column (1→56)",
      ylab = "Row (1→46)",
      main = "Important Pixels Selected (up to 50)")
```




```
# 'red' = selected pixel
# 'white' = not selected
```

Problem 3

Step1 Load the Volcano Data

- Using R bulit-in `Volcano` dataset.
- `Volcano_data[i,j]` = height at (x1 = i, x2 = j).
- We want to find the (x_1,x_2) that gives the max height.

```
volcano_data <- volcano
dim(volcano_data)
```

```
## [1] 87 61
```

Step2 Set Initial Parameters and Initialize the Search

- When I set the half-Window size as 5,the Highest Point is (27,37),And the Elevation at Highest Point is 180.
- So I change the `window_size` to ten will find the highest point.
- The half-window size for local regression which refers to how far out from the current point you look in each direction when defining a local neighborhood for regression.

```
# Starting position: (x₁, x₂) = (87, 1)
current_x1 <- nrow(volcano_data)  # 87
current_x2 <- 1
# Record the search path as a list of (x1, x2) coordinates
path <- list(c(current_x1, current_x2))

# Parameters for the search
window_size <- 8   # half-window size for local regression
step_size <- 1    # step size
```

Step3 Iterative Search for the Peak

- *Local Window Definition*: Determines which part of the data to use for local approximation.
- *Grid Creation and Elevation Extraction*: Constructs a coordinate grid with corresponding elevation values.
- *Local Linear Regression*: Fits a plane to the local data and extracts the gradient.
- *Movement*: Uses the gradient and step size to update the search position.
- *Convergence Check*: Stops the algorithm when no further improvement is detected.

```r
for (i in 1:100) {
  row_min <- max(1, current_x1 - window_size)
  row_max <- min(nrow(volcano_data), current_x1 + window_size)
  col_min <- max(1, current_x2 - window_size)
  col_max <- min(ncol(volcano_data), current_x2 + window_size)

  # Create a grid of coordinates for the local window
  x1_seq <- row_min:row_max
  x2_seq <- col_min:col_max
  grid <- expand.grid(x1 = x1_seq, x2 = x2_seq)
  # Extract the corresponding elevations.
  # Note: volcano_data[row, col] where row corresponds to x1 and col to x2.
  grid$z <- as.vector(volcano_data[x1_seq, col_min:col_max])

  # Fit a local linear model: z ~ x1 + x2
  model <- lm(z ~ x1 + x2, data = grid)
  coefs <- coef(model)
  # Extract the slopes (gradient components)
  grad <- coefs[c("x1", "x2")]

  # Determine move direction: sign(gradient) multiplied by step_size
  move_x1 <- sign(grad["x1"]) * step_size
  move_x2 <- sign(grad["x2"]) * step_size

  # Update position
  new_x1 <- current_x1 + move_x1
  new_x2 <- current_x2 + move_x2

  # Ensure the new position is within bounds
  new_x1 <- max(1, min(new_x1, nrow(volcano_data)))
  new_x2 <- max(1, min(new_x2, ncol(volcano_data)))

  # If the new position is unchanged, assume a local maximum has been reached
  if ((new_x1 == current_x1) && (new_x2 == current_x2)) {
    break
  }

  # Update the current position and add it to the path
  current_x1 <- new_x1
  current_x2 <- new_x2
  path <- c(path, list(c(current_x1, current_x2)))
}
```

Step4 Report the Highest Point

```r
highest_point <- c(current_x1, current_x2)  # (x1, x2) in 1-based indexing
highest_elevation <- volcano_data[current_x1, current_x2]

cat("Highest Point (1-based index):", highest_point, "\n")
```

```
## Highest Point (1-based index): 21 29
```

```r
cat("Elevation at Highest Point:", highest_elevation, "\n")
```

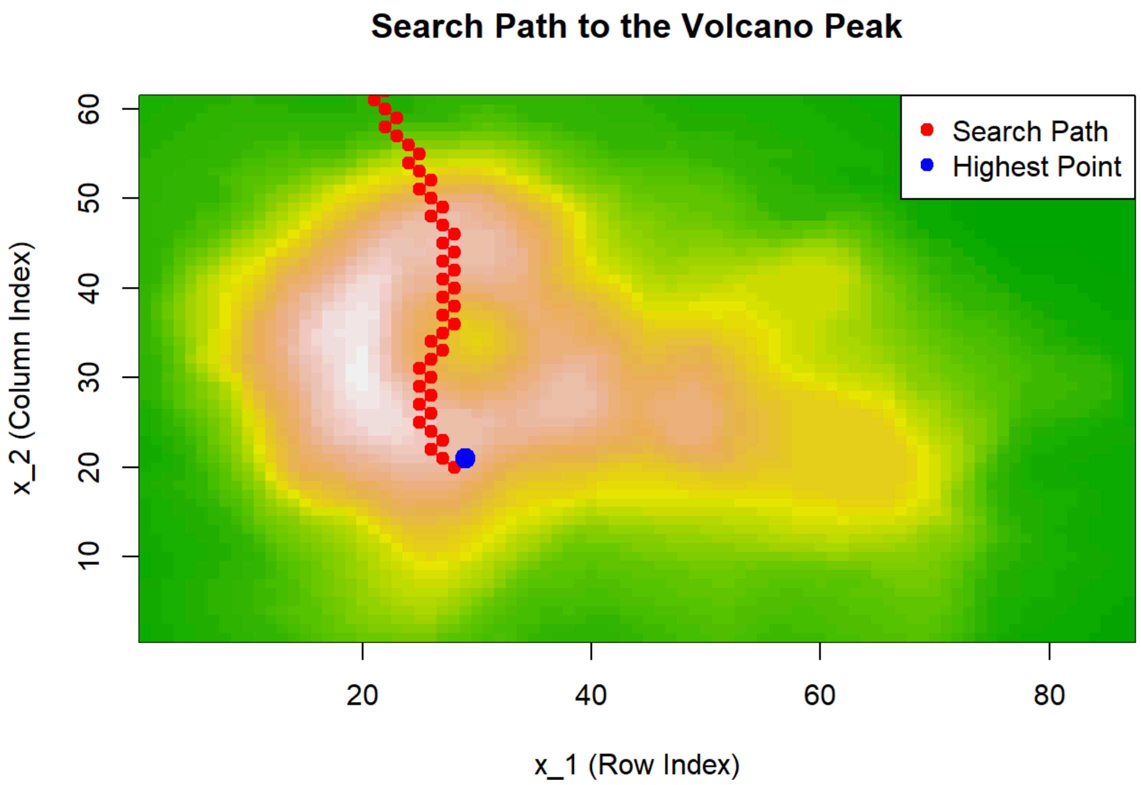
```
## Elevation at Highest Point: 191
```

Step5 Visualize the Path on a Contour Plot

```
# Create a grid for plotting: x1 = rows, x2 = columns
x1_range <- 1:nrow(volcano_data)
x2_range <- 1:ncol(volcano_data)

# Use image() to display the volcano data
image( x1_range, x2_range, volcano_data,
      col = terrain.colors(50),
      xlab = "x_1 (Row Index)",
      ylab = "x_2 (Column Index)",
      main = "Search Path to the Volcano Peak")

# Convert the path (list) to a matrix for plotting.
path_mat <- do.call(rbind, path)
# In the plot, x corresponds to columns (x₂) and y to rows (x₁).
lines(path_mat[,2], path_mat[,1], col = "red", type = "o", pch = 19)
points(highest_point[2], highest_point[1], col = "blue", pch = 19, cex = 1.5)
legend("topright", legend = c("Search Path", "Highest Point"),
       col = c("red", "blue"), pch = c(19, 19))
```



Problem 4

- We assume the model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon, \epsilon \sim N(0, \sigma^2)$.

(a) Use a regression package in R to analyze the problem and interpret the results.

- Simulate the data with $x_1 \sim N(0,1)$,$x_2 \sim N(0,1)$
- And $y = 2 + 3x_1 + (-1)x_2 + \epsilon)$
- where $\epsilon \sim N(0, \sigma^2)$

1. Simulate the Data

```
n <- 50000
beta0 <- 2
beta1 <- 3
beta2 <- -1
sigma <- 2  # standard deviation of the noise

# Generate predictors from N(0,1)
x1 <- rnorm(n, mean = 0, sd = 1)
x2 <- rnorm(n, mean = 0, sd = 1)

# Generate noise term: epsilon ~ N(0, sigma^2)
epsilon <- rnorm(n, mean = 0, sd = sigma)

# Generate response variable y
y <- beta0 + beta1 * x1 + beta2 * x2 + epsilon

# Combine into a data frame
df <- data.frame(y, x1, x2)

# Display the first few rows of the data
head(df)
```

```
##            y         x1         x2
## 1  0.3731012 -0.1126949 -0.2658292
## 2  3.4582951 -0.6940615 -0.9665603
## 3 -0.1095448 -1.4536800 -0.9555962
## 4 -2.0788741 -1.0576313  1.7959559
## 5 -0.1108864 -1.0796584 -0.1700542
## 6  0.7635127 -0.6163644 -0.2222210
```

2. Fit the Linear Model

```
model <- lm(y ~ x1 + x2, data = df)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.067 -1.345  0.000  1.336  8.438
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.006376   0.008929   224.7   <2e-16 ***
## x1           2.999468   0.008948   335.2   <2e-16 ***
## x2          -1.013448   0.008892  -114.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.996 on 49997 degrees of freedom
## Multiple R-squared:  0.7157, Adjusted R-squared:  0.7157
## F-statistic: 6.292e+04 on 2 and 49997 DF,  p-value: < 2.2e-16
```

3. Explanation and Interpretation

- The fitting model is $\hat{y} = 2.011668 + \quad 3.004628x_1 - 1.00265x_2$
- Extremely low p-values (< 2.2e-16) for all coefficients indicate that each is statistically significant at typical thresholds (0.05, 0.01, etc.).
- Very large t-values (225.5, 358.1, -111.5) confirm the significance.

- R^2 around 71.5% of the variance in y is explain by x_1, x_2
- Overall Model Significance F-statistic is 6.281 × 10^4, p-value < 2.2e-16, strongly indicating the model is significant and reject the null hypothesis that all slope coefficients are zero.

(b)Estimating Regression Coefficients via Gradient Descent

1.Define the Loss Function and Its Gradient

```
# Loss function (mean squared error) multiplied by 1/2 for convenience.
loss_function <- function(beta, y, x1, x2) {
  n <- length(y)
  predictions <- beta[1] + beta[2]*x1 + beta[3]*x2
  loss <- sum((y - predictions)^2) / (2*n)
  return(loss)
}

# Gradient of the loss function with respect to beta parameters.
gradient_function <- function(beta, y, x1, x2) {
  n <- length(y)
  predictions <- beta[1] + beta[2]*x1 + beta[3]*x2
  error <- y - predictions
  grad0 <- -sum(error) / n
  grad1 <- -sum(error * x1) / n
  grad2 <- -sum(error * x2) / n
  return(c(grad0, grad1, grad2))
}
```

2.Implement Gradient Descent

```
# Initialization
beta <- c(0, 0, 0)  # initial guess for (beta0, beta1, beta2)
alpha <- 0.05       # learning rate (you may adjust this)
max_iter <- 1000
tol <- 1e-6

# To store the loss and parameters at each iteration
loss_history <- numeric(max_iter)
beta_history <- matrix(0, nrow = max_iter, ncol = 3)

for (iter in 1:max_iter) {
  current_loss <- loss_function(beta, y, x1, x2)
  loss_history[iter] <- current_loss
  beta_history[iter, ] <- beta

  grad <- gradient_function(beta, y, x1, x2)
  beta_new <- beta - alpha * grad

  # Check for convergence: change in beta is small
  if (sum(abs(beta_new - beta)) < tol) {
    beta <- beta_new
    beta_history[iter, ] <- beta
    loss_history[iter] <- loss_function(beta, y, x1, x2)
    cat("Converged at iteration", iter, "\n")
    loss_history <- loss_history[1:iter]
    beta_history <- beta_history[1:iter, ]
    break
  }

  beta <- beta_new
}
```

```
## Converged at iteration 247
```

```r
cat("Final beta from gradient descent:\n")
```

```
## Final beta from gradient descent:
```

```r
print(beta)
```

```
## [1]  2.006369  2.999458 -1.013446
```

```r
cat("Final loss:", tail(loss_history, 1), "\n")
```

```
## Final loss: 1.992845
```

3.Compare with OLS Solution

```r
ols_model <- lm(y ~ x1 + x2, data = df)
ols_beta <- coef(ols_model)
cat("OLS coefficients:\n")
```

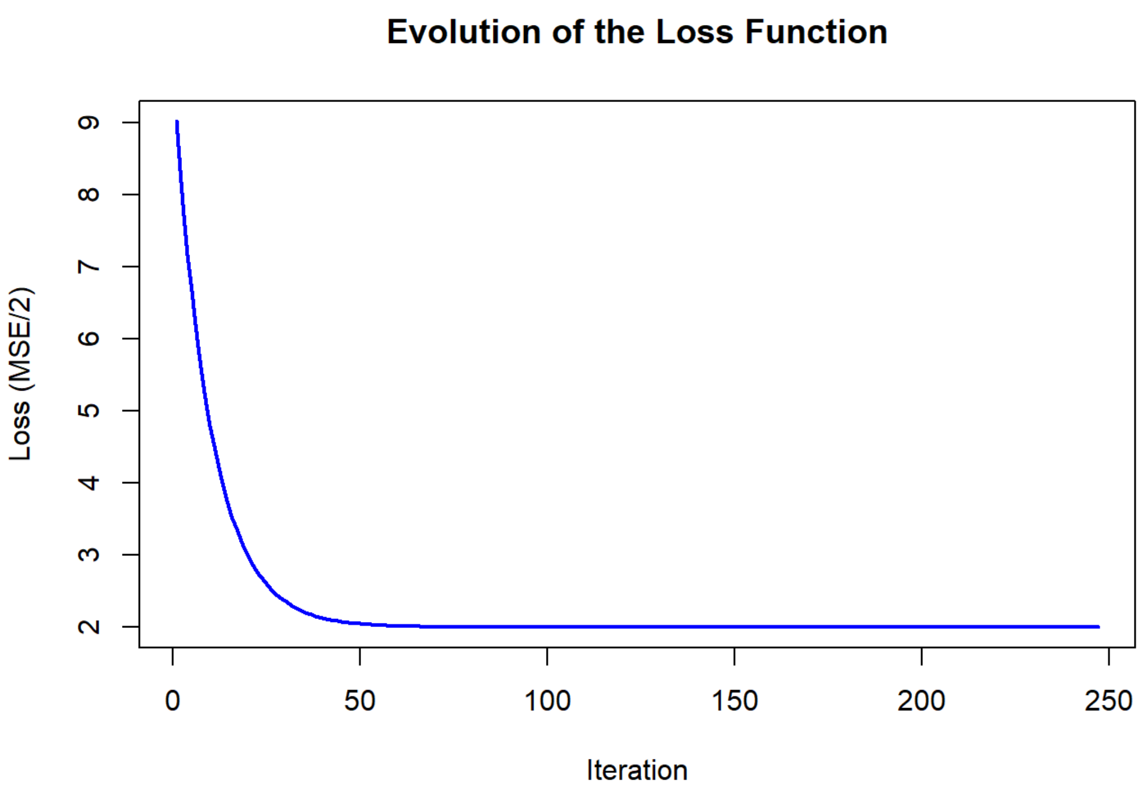
```
## OLS coefficients:
```

```r
print(ols_beta)
```

```
## (Intercept)          x1          x2
##    2.006376    2.999468   -1.013448
```

4.Plot the Evolution of the Loss Function and contour plot

```r
plot(loss_history, type = "l", col = "blue", lwd = 2,
     xlab = "Iteration", ylab = "Loss (MSE/2)",
     main = "Evolution of the Loss Function")
```



```
# Fix beta0 at its final value
beta0_final <- beta[1]

# Define a grid for beta1 and beta2
beta1_range <- seq(ols_beta[2] - 1, ols_beta[2] + 1, length.out = 100)
beta2_range <- seq(ols_beta[3] - 1, ols_beta[3] + 1, length.out = 100)
grid <- expand.grid(beta1 = beta1_range, beta2 = beta2_range)

# Compute the loss over the grid for each pair (fixing beta0)
grid$loss <- apply(grid, 1, function(b) {
  loss_function(c(beta0_final, b["beta1"], b["beta2"]), y, x1, x2)
})

# Create a contour plot
contour(beta1_range, beta2_range,
        matrix(grid$loss, nrow = 100, byrow = TRUE),
        xlab = expression(beta[1]), ylab = expression(beta[2]),
        main = "Search Path in (β_1, β_2) Space (β_0 fixed)",
        nlevels = 30, col = "gray")

# Overlay the search path for beta1 and beta2
lines(beta_history[,2], beta_history[,3], col = "red", lwd = 2)
points(beta_history[,2], beta_history[,3], col = "red", pch = 19)
```