Azreen Haque

2/28/2025

Solutions

Problem 1

Solutions below.

Part A

```r
data <- read.csv("hw04pr01.csv", header = TRUE, sep = ",")
names(data)
```

```
## [1] "Copiers" "Minutes"
```

```r
# Fit simple linear regression model
model <- lm(Minutes ~ Copiers, data = data)

# summary of the model
summary(model)
```

```
##
## Call:
## lm(formula = Minutes ~ Copiers, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22.7723  -3.7371   0.3334   6.3334  15.4039
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.5802     2.8039  -0.207    0.837
## Copiers      15.0352     0.4831  31.123   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.914 on 43 degrees of freedom
## Multiple R-squared:  0.9575, Adjusted R-squared:  0.9565
## F-statistic: 968.7 on 1 and 43 DF,  p-value: < 2.2e-16
```

```r
# Extract coefficients
b0 <- coef(model)[1]  # Intercept
b1 <- coef(model)[2]  # Slope

# Print fitted equation
cat("Fitted Equation: Y (Minutes) =", round(b0, 2), "+", round(b1, 2), "* X (Copiers)\n")
```

```
## Fitted Equation: Y (Minutes) = -0.58 + 15.04 * X (Copiers)
```

Part B

```r
# Define confidence level
alpha <- 0.10

# Sample size
n <- nrow(data)

# Compute MSE
MSE <- sum(residuals(model)^2) / (n - 2)

# Compute the mean of X (Copiers)
Xbar <- mean(data$Copiers)

# Compute SXX
SXX <- sum((data$Copiers - Xbar)^2)

# Compute t critical value for (1 - alpha/2) confidence level
t_critical <- qt(1 - alpha/2, df = n-2)

# Function to calculate confidence interval at given X_h
confidence_interval <- function(X_h) {
  # Compute predicted Y value
  Y_hat <- coef(model)[1] + coef(model)[2] * X_h

  # Compute standard error
  SE_Y_hat <- sqrt(MSE * (1/n + ((X_h - Xbar)^2 / SXX)))

  # Compute confidence interval bounds
  margin_of_error <- t_critical * SE_Y_hat
  lower_bound <- Y_hat - margin_of_error
  upper_bound <- Y_hat + margin_of_error

  return(c(lower_bound, upper_bound))
}

# Calculate confidence intervals for X_h = 2, 4, 6, 8
X_h_values <- c(2, 4, 6, 8)
CI_results <- t(sapply(X_h_values, confidence_interval))

# Results
colnames(CI_results) <- c("Lower Bound", "Upper Bound")
rownames(CI_results) <- paste("X_h =", X_h_values)
print(CI_results)
```

```
##           Lower Bound Upper Bound
## X_h = 2    26.11796    32.86272
## X_h = 4    57.15175    61.96992
## X_h = 6    87.28387    91.97880
## X_h = 8   116.46245   122.94121
```

```r
# Interpretation at X_h = 4
X_selected <- 4
CI_selected <- confidence_interval(X_selected)

cat("We are 90% confident that the true mean service time\n",
    "for servicing", X_selected, "copiers is between\n",
    round(CI_selected[1], 2), "and", round(CI_selected[2], 2),
    "minutes.\n")
```

```
## We are 90% confident that the true mean service time
##  for servicing 4 copiers is between
##  57.15 and 61.97 minutes.
```

Part C

```r
# Define confidence level
alpha <- 0.10
g <- 4

# Sample size
n <- nrow(data)

# Compute MSE
MSE <- sum(residuals(model)^2) / (n - 2)

# Compute the mean of X (Copiers)
Xbar <- mean(data$Copiers)

# Compute SXX
SXX <- sum((data$Copiers - Xbar)^2)

# Compute t critical value for Bonferroni correction
t_critical_bonf <- qt(1 - alpha / (2 * g), df = n - 2)

# Function to calculate Bonferroni confidence interval at given X_h
bonf_confidence_interval <- function(X_h) {
  # Compute predicted Y value
  Y_hat <- coef(model)[1] + coef(model)[2] * X_h

  # Compute standard error
  SE_Y_hat <- sqrt(MSE * (1/n + ((X_h - Xbar)^2 / SXX)))

  # Compute confidence interval bounds
  margin_of_error <- t_critical_bonf * SE_Y_hat
  lower_bound <- Y_hat - margin_of_error
  upper_bound <- Y_hat + margin_of_error

  return(c(lower_bound, upper_bound))
}

# Calculate joint confidence intervals for X_h = 2, 4, 6, 8
```

```r
X_h_values <- c(2, 4, 6, 8)
Bonf_CI_results <- t(sapply(X_h_values, bonf_confidence_interval))

# Results
colnames(Bonf_CI_results) <- c("Lower Bound", "Upper Bound")
rownames(Bonf_CI_results) <- paste("X_h =", X_h_values)
print(Bonf_CI_results)
```

```
##          Lower Bound Upper Bound
## X_h = 2    24.83096    34.14972
## X_h = 4    56.23236    62.88931
## X_h = 6    86.38800    92.87466
## X_h = 8   115.22620   124.17745
```

Part D

```r
# Define confidence level
alpha <- 0.10  # 90% confidence level

# Sample size
n <- nrow(data)

# Compute MSE
MSE <- sum(residuals(model)^2) / (n - 2)

# Compute the mean of X (Copiers)
Xbar <- mean(data$Copiers)

# Compute SXX
SXX <- sum((data$Copiers - Xbar)^2)

# Compute F critical value for Working-Hotelling procedure
F_critical_wh <- qf(1 - alpha, df1 = 2, df2 = n-2)

# Function to calculate Working-Hotelling confidence interval at given X_h
wh_confidence_interval <- function(X_h) {
  # Compute predicted Y value
  Y_hat <- coef(model)[1] + coef(model)[2] * X_h

  # Compute standard error
  SE_Y_hat <- sqrt(MSE * (1/n + ((X_h - Xbar)^2 / SXX)))

  # Compute confidence interval bounds
  margin_of_error <- sqrt(2 * F_critical_wh) * SE_Y_hat
  lower_bound <- Y_hat - margin_of_error
  upper_bound <- Y_hat + margin_of_error

  return(c(lower_bound, upper_bound))
}

# Calculate joint confidence intervals for X_h = 2, 4, 6, 8
```

```r
WH_CI_results <- t(sapply(X_h_values, wh_confidence_interval))

# Display results
colnames(WH_CI_results) <- c("Lower Bound", "Upper Bound")
rownames(WH_CI_results) <- paste("X_h =", X_h_values)
print(WH_CI_results)
```

```
##         Lower Bound Upper Bound
## X_h = 2    25.06746    33.91321
## X_h = 4    56.40131    62.72036
## X_h = 6    86.55263    92.71003
## X_h = 8   115.45338   123.95028
```

Part E

```r
# Define the confidence intervals from Parts (b), (c), and (d)
CI_b <- matrix(c(26.11796, 32.86272,
                 57.15175, 61.96992,
                 87.28387, 91.97880,
                 116.46245, 122.94121),
               ncol = 2, byrow = TRUE)

CI_c <- matrix(c(24.83096, 34.14972,
                 56.23236, 62.88931,
                 86.38800, 92.87466,
                 115.22620, 124.17745),
               ncol = 2, byrow = TRUE)

CI_d <- matrix(c(25.06746, 33.91321,
                 56.40131, 62.72036,
                 86.55263, 92.71003,
                 115.45338, 123.95028),
               ncol = 2, byrow = TRUE)

# Compute Confidence Interval Widths
width_b <- CI_b[,2] - CI_b[,1]  # Part (b)
width_c <- CI_c[,2] - CI_c[,1]  # Part (c)
width_d <- CI_d[,2] - CI_d[,1]  # Part (d)

# Create a dataframe to display the widths
X_h_values <- c(2, 4, 6, 8)
CI_widths <- data.frame(
  X_h = X_h_values,
  Width_b = width_b,
  Width_c = width_c,
  Width_d = width_d
)

# Print results
print(CI_widths)
```

```
##   X_h Width_b Width_c Width_d
## 1   2 6.74476 9.31876 8.84575
## 2   4 4.81817 6.65695 6.31905
## 3   6 4.69493 6.48666 6.15740
## 4   8 6.47876 8.95125 8.49690
```

```r
# Comparison
cat("Bonferroni (Part c) produces the widest confidence intervals at every X_h.
    Individual CIs (Part b) are the narrowest and Working-Hotelling (Part d) produces
    slightly narrower intervals than Bonferroni.")
```

```
## Bonferroni (Part c) produces the widest confidence intervals at every X_h.
##     Individual CIs (Part b) are the narrowest and Working-Hotelling (Part d) produces
##     slightly narrower intervals than Bonferroni.
```

Part F

```r
# Fit a linear regression through the origin (no intercept)
model_origin <- lm(Minutes ~ 0 + Copiers, data = data)

# Display model summary
summary(model_origin)
```

```
##
## Call:
## lm(formula = Minutes ~ 0 + Copiers, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22.4723  -3.6306   0.2111   6.3694  15.2639
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## Copiers  14.9472     0.2264   66.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.816 on 44 degrees of freedom
## Multiple R-squared:   0.99,  Adjusted R-squared:  0.9898
## F-statistic:  4358 on 1 and 44 DF,  p-value: < 2.2e-16
```

```r
# Extract the coefficient (slope)
b1_origin <- coef(model_origin)[1]

# Print fitted equation
cat("Fitted Equation: Y (Minutes) =", round(b1_origin, 2), "* X (Copiers)\n")
```

```
## Fitted Equation: Y (Minutes) = 14.95 * X (Copiers)
```

Part G

```r
# Get predicted values
Y_pred_origin <- predict(model_origin)

# Compute residuals
residuals_origin <- data$Minutes - Y_pred_origin

# Check if residuals sum to zero
residuals_sum <- sum(residuals_origin)

# Print sum of residuals
cat("Sum of residuals:", residuals_sum, "does not equal 0")
```

```
## Sum of residuals: -5.862797 does not equal 0
```

Problem 2

Part A

```r
# Load library
library(ggplot2)

# Load dataset
data <- read.csv("hw04pr02.csv")

head(data)
```

```
##     GPA ACT
## 1 3.429  25
## 2 2.966  23
## 3 2.183  27
## 4 3.039  23
## 5 3.013  31
## 6 3.013  24
```

```r
# Fit a regular linear regression model (with intercept)
model <- lm(GPA ~ ACT, data = data)

# Display model summary
summary(model)
```

```
##
## Call:
## lm(formula = GPA ~ ACT, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.07700 -0.34269  0.03184  0.38990  1.07287
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.74662    0.37144   4.702 1.19e-05 ***
## ACT          0.05481    0.01475   3.716 0.000393 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5616 on 73 degrees of freedom
## Multiple R-squared:  0.1591, Adjusted R-squared:  0.1476
## F-statistic: 13.81 on 1 and 73 DF,  p-value: 0.0003933
```

```r
# Extract coefficients
b0 <- coef(model)[1]  # Intercept
b1 <- coef(model)[2]  # Slope

# Print estimated regression function
cat("Estimated Regression Function: Y =", round(b0, 2), "+", round(b1, 2), "* X (ACT Score)\n")
```

```
## Estimated Regression Function: Y = 1.75 + 0.05 * X (ACT Score)
```

Part B

```r
# Fit linear regression through the origin (no intercept)
model_origin <- lm(GPA ~ 0 + ACT, data = data)

# Display model summary
summary(model_origin)
```

```
##
## Call:
## lm(formula = GPA ~ 0 + ACT, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.31055 -0.25802  0.05889  0.40696  1.59042
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## ACT 0.123088   0.002919   42.17   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6367 on 74 degrees of freedom
## Multiple R-squared:  0.9601, Adjusted R-squared:  0.9595
## F-statistic:  1778 on 1 and 74 DF,  p-value: < 2.2e-16
```

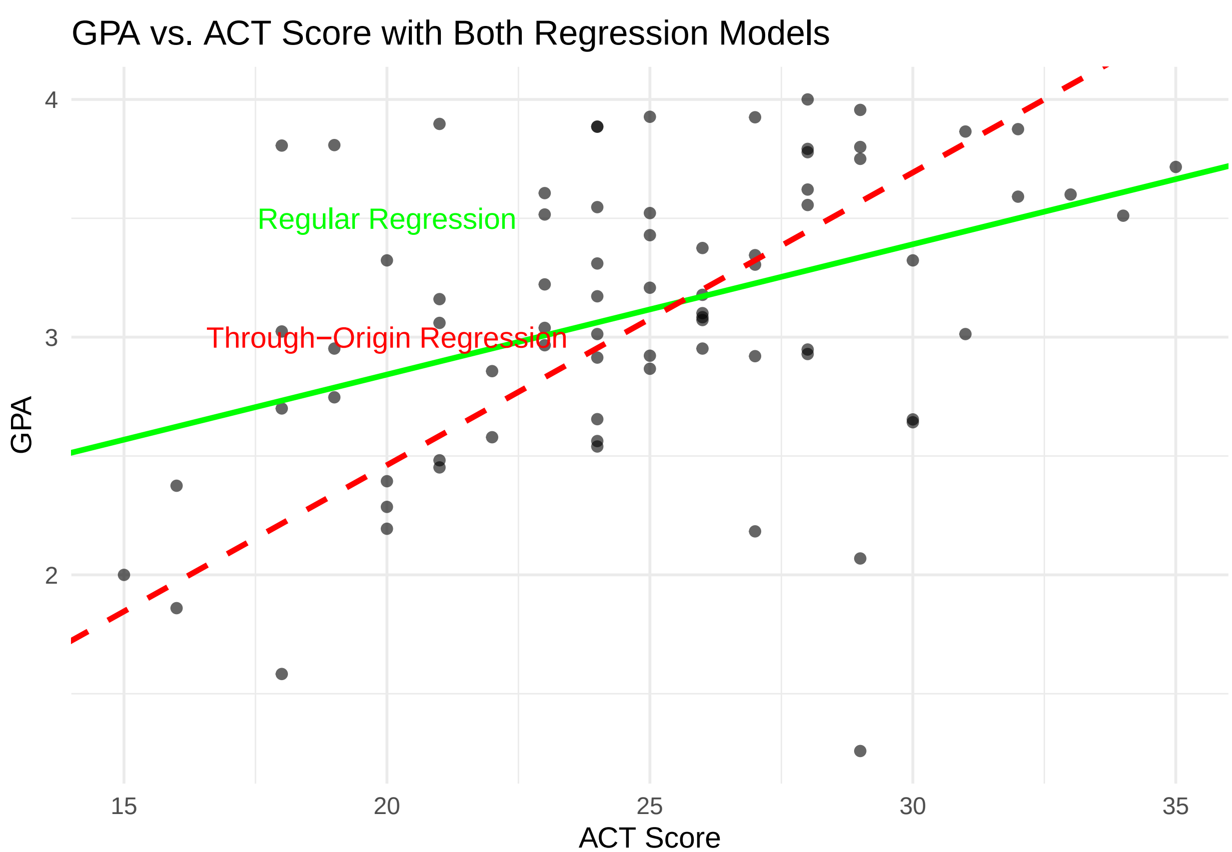
```r
# Extract slope coefficient (no intercept)
b1_origin <- coef(model_origin)[1]

# Print estimated regression function
cat("Estimated Regression Function (Through Origin): Y =", round(b1_origin, 2), "* X (ACT Score)\n")
```

```
## Estimated Regression Function (Through Origin): Y = 0.12 * X (ACT Score)
```

Part C

```r
# Create scatterplot with both regression lines
ggplot(data, aes(x = ACT, y = GPA)) +
  geom_point(color = "black", alpha = 0.6) +  # Scatterplot points
  geom_abline(
    intercept = b0, slope = b1,
    color = "green", linewidth = 1, linetype = "solid"
  ) +  # Regular regression
  geom_abline(
    intercept = 0, slope = b1_origin,
    color = "red", linewidth = 1, linetype = "dashed"
  ) +  # Regression through origin
  labs(
    title = "GPA vs. ACT Score with Both Regression Models",
    x = "ACT Score",
    y = "GPA"
  ) +
  theme_minimal() +
  annotate("text", x = 20, y = 3.5, label = "Regular Regression", color = "green") +
  annotate("text", x = 20, y = 3.0, label = "Through-Origin Regression", color = "red")
```



Part D

```r
# Compute residuals for regular regression
residuals_reg <- residuals(model)
sum_residuals_reg <- sum(residuals_reg)

# Compute residuals for regression through origin
residuals_origin <- residuals(model_origin)
sum_residuals_origin <- sum(residuals_origin)

# Print sum of residuals
cat("Sum of residuals (Regular Regression):", sum_residuals_reg, "\n")
```

```
## Sum of residuals (Regular Regression): 0
```

```r
cat("Sum of residuals (Through-Origin Regression):", sum_residuals_origin, "\n")
```

```
## Sum of residuals (Through-Origin Regression): 3.992276
```

```r
# Explanation of the difference
if (sum_residuals_reg == 0) {
  cat("- The sum of residuals for the regular regression is exactly zero\n")
  cat("  because the model includes an intercept.\n")
  cat("- According to the least squares property, this ensures residuals\n")
  cat("  are symmetrically distributed around zero.\n\n")
}
```

```
## - The sum of residuals for the regular regression is exactly zero
##   because the model includes an intercept.
## - According to the least squares property, this ensures residuals
##   are symmetrically distributed around zero.
```

```r
if (sum_residuals_origin != 0) {
  cat("- The sum of residuals for the through-origin regression is not zero\n")
  cat("  because this model does not include an intercept.\n")
  cat("- Based on Chapter 2, when an intercept is missing, the model\n")
  cat("  cannot correct for shifts in the data, potentially causing\n")
  cat("  systematic bias.\n\n")
}
```

```
## - The sum of residuals for the through-origin regression is not zero
##   because this model does not include an intercept.
## - Based on Chapter 2, when an intercept is missing, the model
##   cannot correct for shifts in the data, potentially causing
##   systematic bias.
```

Problem 3

Part A

```r
# (a) Create the matrices
A <- matrix(c(1, -2, 0, 4, 3, -1), nrow=2, byrow=TRUE)
B <- matrix(c(0, -1, 6, 3, 0, 2, -3, 2, 1), nrow=3, byrow=TRUE)
C <- matrix(c(2, -2, 3, 1), nrow=2, byrow=TRUE)
D <- matrix(c(5, 0, -2), nrow=3, byrow=TRUE)

# Display matrices
print("Matrix A:")
```

```
## [1] "Matrix A:"
```

```r
print(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1   -2    0
## [2,]    4    3   -1
```

```r
print("Matrix B:")
```

```
## [1] "Matrix B:"
```

```r
print(B)
```

```
##      [,1] [,2] [,3]
## [1,]    0   -1    6
## [2,]    3    0    2
## [3,]   -3    2    1
```

```r
print("Matrix C:")
```

```
## [1] "Matrix C:"
```

```r
print(C)
```

```
##      [,1] [,2]
## [1,]    2   -2
## [2,]    3    1
```

```r
print("Matrix D:")
```

```
## [1] "Matrix D:"
```

```r
print(D)
```

```
##      [,1]
## [1,]    5
## [2,]    0
## [3,]   -2
```

Part B

```r
print_matrix_dimensions <- function(matrix_name, matrix_obj) {
  dim_values <- dim(matrix_obj)
  cat("Dimensions of", matrix_name, ":", dim_values[1], "x", dim_values[2], "\n")
}

print_matrix_dimensions("A", A)
```

```
## Dimensions of A : 2 x 3
```

```r
print_matrix_dimensions("B", B)
```

```
## Dimensions of B : 3 x 3
```

```r
print_matrix_dimensions("C", C)
```

```
## Dimensions of C : 2 x 2
```

```r
print_matrix_dimensions("D", D)
```

```
## Dimensions of D : 3 x 1
```

Part C

```r
# Function to check if two matrices can be multiplied
can_multiply <- function(mat1, mat2, name1, name2) {
  if (ncol(mat1) == nrow(mat2)) {
    result <- mat1 %*% mat2
    cat("Matrix", name1, "*", name2, "is valid.\n")
    print(result)
    cat("Dimensions of", name1, "*", name2, ":", dim(result), "\n\n")
  } else {
    cat("Matrix", name1, "*", name2, "is NOT valid due to incompatible dimensions.\n\n")
  }
}

# Check all possible matrix multiplications
can_multiply(A, B, "A", "B")  # A (2x3) * B (3x3) = Valid (2x3)
```

```
## Matrix A * B is valid.
##      [,1] [,2] [,3]
## [1,]   -6   -1    2
## [2,]   12   -6   29
## Dimensions of A * B : 2 3
```

```r
can_multiply(B, A, "B", "A")  # B (3x3) * A (2x3) = Invalid
```

```
## Matrix B * A is NOT valid due to incompatible dimensions.
```

```r
can_multiply(A, C, "A", "C")  # A (2x3) * C (2x2) = Invalid
```

```
## Matrix A * C is NOT valid due to incompatible dimensions.
```

```r
can_multiply(C, A, "C", "A")  # C (2x2) * A (2x3) = Valid (2x3)
```

```
## Matrix C * A is valid.
##      [,1] [,2] [,3]
## [1,]   -6  -10    2
## [2,]    7   -3   -1
## Dimensions of C * A : 2 3
```

```r
can_multiply(B, D, "B", "D")  # B (3x3) * D (3x1) = Valid (3x1)
```

```
## Matrix B * D is valid.
##      [,1]
## [1,]  -12
## [2,]   11
## [3,]  -17
## Dimensions of B * D : 3 1
```

```r
can_multiply(D, B, "D", "B")  # D (3x1) * B (3x3) = Invalid
```

```
## Matrix D * B is NOT valid due to incompatible dimensions.
```

```r
can_multiply(C, D, "C", "D")  # C (2x2) * D (3x1) = Invalid
```

```
## Matrix C * D is NOT valid due to incompatible dimensions.
```

```r
can_multiply(D, C, "D", "C")  # D (3x1) * C (2x2) = Invalid
```

```
## Matrix D * C is NOT valid due to incompatible dimensions.
```

Part D

```r
# Compute transposes
A_t <- t(A)  # A'
B_t <- t(B)  # B'
C_t <- t(C)  # C'
D_t <- t(D)  # D'

# Function to display matrix and its dimensions
```

```r
print_matrix_info <- function(matrix_name, matrix_obj) {
  cat("\nTranspose of", matrix_name, ":\n")
  print(matrix_obj)
  cat("Dimensions of", matrix_name, "':", dim(matrix_obj)[1], "x", dim(matrix_obj)[2], "\n")
}

# Display transposes and their dimensions
print_matrix_info("A", A_t)
```

```
##
## Transpose of A :
##      [,1] [,2]
## [1,]    1    4
## [2,]   -2    3
## [3,]    0   -1
## Dimensions of A ': 3 x 2
```

```r
print_matrix_info("B", B_t)
```

```
##
## Transpose of B :
##      [,1] [,2] [,3]
## [1,]    0    3   -3
## [2,]   -1    0    2
## [3,]    6    2    1
## Dimensions of B ': 3 x 3
```

```r
print_matrix_info("C", C_t)
```

```
##
## Transpose of C :
##      [,1] [,2]
## [1,]    2    3
## [2,]   -2    1
## Dimensions of C ': 2 x 2
```

```r
print_matrix_info("D", D_t)
```

```
##
## Transpose of D :
##      [,1] [,2] [,3]
## [1,]    5    0   -2
## Dimensions of D ': 1 x 3
```

Part E

```r
# Function to check valid multiplication and display result
can_multiply <- function(mat1, mat2, name1, name2) {
  if (ncol(mat1) == nrow(mat2)) {
```

```r
    result <- mat1 %*% mat2
    cat("\nMatrix", name1, "*", name2, "is valid.\n")
    print(result)
    cat("Dimensions of", name1, "*", name2, ":", dim(result)[1], "x", dim(result)[2], "\n")
  } else {
    cat("\nMatrix", name1, "*", name2, "is NOT valid due to incompatible dimensions.\n")
  }
}

# Identify at least two valid multiplications involving transposed matrices
can_multiply(A_t, C, "A'", "C")  # A' (3x2) * C (2x2) = Valid (3x2)
```

```
##
## Matrix A' * C is valid.
##      [,1] [,2]
## [1,]   14    2
## [2,]    5    7
## [3,]   -3   -1
## Dimensions of A' * C : 3 x 2
```

```r
can_multiply(B_t, D, "B'", "D")  # B' (3x3) * D (3x1) = Valid (3x1)
```

```
##
## Matrix B' * D is valid.
##      [,1]
## [1,]    6
## [2,]   -9
## [3,]   28
## Dimensions of B' * D : 3 x 1
```

Part F

```r
# Store in a named list
matrices <- list(A=A, B=B, C=C, D=D, A_t=A_t, B_t=B_t, C_t=C_t, D_t=D_t)

# Generate all pairs of matrices (unique)
matrix_pairs <- combn(names(matrices), 2, simplify = FALSE)

# Check valid additions
valid_additions <- lapply(matrix_pairs, function(pair) {
  if (all(dim(matrices[[pair[1]]]) == dim(matrices[[pair[2]]]))) {
    return(pair)
  } else {
    return(NULL)
  }
})

# Filter out invalid additions
valid_additions <- Filter(Negate(is.null), valid_additions)
```

```r
# Results
if (length(valid_additions) > 0) {
  cat("The following matrix pairs can be added together:\n")
  for (pair in valid_additions) {
    cat(pair[1], "and", pair[2], "(Dimensions:",
  dim(matrices[[pair[1]]])[1], "x", dim(matrices[[pair[1]]])[2], ")\n")
  }
} else {
  cat("No matrices can be added together as none share the same dimensions.\n")
}
```

```
## The following matrix pairs can be added together:
## B and B_t (Dimensions: 3 x 3 )
## C and C_t (Dimensions: 2 x 2 )
```

```r
# Explanation
cat("Only two pairs of the matrices in this example can be added together\n")
```

```
## Only two pairs of the matrices in this example can be added together
```

```r
cat("because matrix addition requires that both matrices\n")
```

```
## because matrix addition requires that both matrices
```

```r
cat("have the same dimensions.\n")
```

```
## have the same dimensions.
```