```
#############################################################
# Energy Appliances Dataset
# The Dataset for this assignment is modified version of a subset of data used in Candanedo et
al, 2017.
# The experimental data have been used to create models of energy use of appliances in a
residential house.
# The modified Dataset provides the energy use of Appliances (denoted as Y).
# The Dataset comprises 5 features (variables), which are denoted as X1, X2, X3, X4 and X5.
# The details about these variables are given below:
# X1: Temperature in living room area (Celsius degrees)
# X2: Humidity in living room area (percentage)
# X3: Temperature in office room (Celsius degrees)
# X4: Humidity in office room (percentage)
# X5: Pressure (millimeter of mercury)
# Y: Appliances energy consumption (Wh)
# For more information about the variables see Candanedo et al, 2017.
#############################################################

# TASK T1.Understand the data 225440315

# ============================================================================================
# (i) Download the txt file (ENB.txt) from Olympus and save it to your R working directory.
# ============================================================================================

# Get current working directory - returns the absolute path of R's working directory
getwd()

# ============================================================================================
# (ii) Assign the data to a matrix, e.g. using
# ============================================================================================

# Load data from file into matrix format
the.data <- as.matrix(read.table("ENB.txt"))

# -----------------------------------
# 1. Structure of original dataset
# -----------------------------------
str(the.data)

# -----------------------------------
# 2. Missing values
# -----------------------------------
sum(is.na(the.data))          # Total missing values
colSums(is.na(the.data))      # Missing per column

# -----------------------------------
# 3. Duplicated rows
# -----------------------------------
sum(duplicated(the.data))     # Number of duplicates
the.data[duplicated(the.data), ]   # Show duplicates

# -----------------------------------
# 4. Non-numeric columns
# -----------------------------------
sapply(the.data, class)       # Check type of each column
non_numeric_cols <- names(the.data)[!sapply(the.data, is.numeric)]
non_numeric_cols

# -----------------------------------
# 5. Summary statistics
# -----------------------------------
summary(the.data)


# ============================================================================================
# (iii)  Use your student ID as the seed for reproducibility
# ============================================================================================

# Set random seed for reproducibility
```

```
set.seed(225440315)

# Number of rows (samples) in data
num_samples <- nrow(the.data)

# Number of columns (features) in data
num_col     <- ncol(the.data)

# Assign column names
colnames(the.data) <- c("sr_no", "X1", "X2", "X3", "X4","X5","Y")

# Display first 10 rows for data inspection
head(the.data,10)

# ============================================================================================
# (iv) The variable of interest is Y. To investigate Y, generate a subset of num_row=650 (use
the same setting for the following tasks as well) with numerical )
# ============================================================================================

num_row <- 650

# Randomly sample 650 rows with all columns
my.data <- the.data[sample(1:num_samples, num_row), c(1:num_col)]

# Determine number of rows (num_samples)
# Recalculate dimensions after sampling
num_samples <- nrow(my.data)
num_samples

# Determine number of columns (num_col)
num_col <- ncol(my.data)
num_col


# ============================================================================================
# (iv)  Use scatter plots and histograms to understand the relationship between each of the
variables X1 X2, X3, X4, X5,
#       and your variable of interest Y, i.e., scatter plots of (X1, Y), (X2, Y), …, (X5, Y),
and histograms of X1 X2, X3,
#       X4, X5, Y.
# ============================================================================================
# SCATTER PLOT
# ============================================================================================
# Set up 2x3 grid for multiple plots
par(mfrow = c(2, 3))   # Display 6 plots in grid


# Create scatter plots of each predictor variable vs response variable Y
plot(my.data[,2], my.data[,7],
     xlab="X1", ylab="Y", main="Scatter Plot: X1 vs Y")

plot(my.data[,3], my.data[,7],
     xlab="X2", ylab="Y", main="Scatter Plot: X2 vs Y")

plot(my.data[,4], my.data[,7],
     xlab="X3", ylab="Y", main="Scatter Plot: X3 vs Y")

plot(my.data[,5], my.data[,7],
     xlab="X4", ylab="Y", main="Scatter Plot: X4 vs Y")

plot(my.data[,6], my.data[,7],
     xlab="X5", ylab="Y", main="Scatter Plot: X5 vs Y")

# ============================================================================================
# HISTOGRAMS
# ============================================================================================

# Set up 2x3 grid for multiple plots
par(mfrow = c(2, 3))   # Layout again
```

```r
# Plot each feature against target variable Y
hist(my.data[,2], main="Histogram: X1", xlab="X1")
hist(my.data[,3], main="Histogram: X2", xlab="X2")
hist(my.data[,4], main="Histogram: X3", xlab="X3")
hist(my.data[,5], main="Histogram: X4", xlab="X4")
hist(my.data[,6], main="Histogram: X5", xlab="X5")
hist(my.data[,7], main="Histogram: Y",  xlab="Y")

# ============================================================================================
# BoX PLOT for Outliers
# ============================================================================================

# Assuming my.data is a data.frame or matrix with columns X1–X5 and Y

par(mfrow = c(2, 3))   # arrange plots in 2 rows, 3 columns

boxplot(my.data[,1], main = "X1", col = "lightblue")
boxplot(my.data[,2], main = "X2", col = "lightgreen")
boxplot(my.data[,3], main = "X3", col = "lightpink")
boxplot(my.data[,4], main = "X4", col = "lightyellow")
boxplot(my.data[,5], main = "X5", col = "lightgray")
boxplot(my.data[,6], main = "Y",  col = "lightcyan")

par(mfrow = c(1, 1))   # reset layout


summary(my.data)

# ############################################################################################
# T2. TRANSFORM THE DATA
# ############################################################################################


# ============================================================================================
# (i) Choose any FOUR variables from X1, X2, X3, X4, X5.
# ============================================================================================
# Calculate the Pearson correlation coefficients between the five input variables (X1–X5)
# and the output variable Y. The purpose is to quantify the strength and direction of
# the linear relationship between each predictor and Y, which guides variable selection
# for the predictive model.

# Compute correlations between X1–X5 and Y
cor_with_Y <- cor(my.data[, 2:6], my.data[, 7])

# Convert to absolute correlation values
abs_cor <- abs(cor_with_Y)

# Get indices of the top 4 features
top4_index <- order(abs_cor, decreasing = TRUE)[1:4]

# Extract the feature names
top4_features <- colnames(my.data)[top4_index + 1]


# Print results
cor_with_Y
top4_features

# ============================================================================================
# Verifying feature selection with lasso (L1 regularization)
# Prepare data for LASSO regression by creating predictor matrix and response vector

# Install glmnet (run this once)
install.packages("glmnet")


# Load required library for LASSO regression
```

```
library(glmnet)

# Prepare predictor matrix (columns 2-6) and target variable Y
x <- my.data[, 2:6]
y <- my.data[,"Y"]

# Fit cross-validated LASSO regression model
# alpha = 1 specifies LASSO regularization (L1 penalty)
# standardize = TRUE standardizes variables before fitting
lasso_model <- cv.glmnet(x, y, alpha = 1, standardize = TRUE)

# Extract optimal lambda value that minimizes cross-validation error
best_lambda <- lasso_model$lambda.min

# Get coefficients at optimal lambda value
lasso_coefficients <- coef(lasso_model, s = best_lambda)
print(lasso_coefficients)

# Convert sparse matrix to regular matrix for proper indexing
coef_matrix <- as.matrix(lasso_coefficients)

# Create a data frame of coefficients excluding intercept
coef_df <- data.frame(
  feature = rownames(coef_matrix),
  coefficient = as.numeric(coef_matrix),
  abs_coef = abs(as.numeric(coef_matrix))
)
coef_df <- coef_df[coef_df$feature != "(Intercept)", ]  # Remove intercept

# Sort features by absolute coefficient value (descending order) and select top 4
coef_df <- coef_df[order(-coef_df$abs_coef), ]
selected_features <- coef_df$feature[1:4]

# Display the final selected features with their coefficients
print("Top 4 features selected by LASSO (sorted by importance):")
print(coef_df[1:4, c("feature", "coefficient")])



# ============================================================================================
# (ii) Make appropriate transformations so that the values can be aggregated in order to
#      predict the variable of interest Y.
# ============================================================================================

library(bestNormalize)
library(e1071)

#=========================================================
# FUNCTION to test transformations for X variables
#=========================================================
test_transforms_X <- function(x) {

  asinh_obj   <- asinh(x)
  yeo_obj     <- yeojohnson(x)
  boxcox_obj  <- boxcox(x)
  order_obj   <- orderNorm(x)

  results <- data.frame(
    method = c("arcsinh", "yeo_johnson", "boxcox", "orderNorm"),
    shapiro_p = c(
      shapiro.test(asinh_obj)$p.value,
      shapiro.test(yeo_obj$x.t)$p.value,
      shapiro.test(boxcox_obj$x.t)$p.value,
      shapiro.test(order_obj$x.t)$p.value
    ),
    skewness = c(
      skewness(asinh_obj),
      skewness(yeo_obj$x.t),
      skewness(boxcox_obj$x.t),
```

```
      skewness(order_obj$x.t)
    )
  )

  list(
    table   = results,
    objects = list(
      arcsinh     = asinh_obj,
      yeo_johnson = yeo_obj,
      boxcox      = boxcox_obj,
      orderNorm   = order_obj
    )
  )
}

#=========================================================
# FUNCTION to test transformations for Y (no orderNorm)
#=========================================================
test_transforms_Y <- function(y) {

  asinh_obj   <- asinh(y)
  yeo_obj     <- yeojohnson(y)
  boxcox_obj  <- boxcox(y)

  results <- data.frame(
    method = c("arcsinh", "yeo_johnson", "boxcox"),
    shapiro_p = c(
      shapiro.test(asinh_obj)$p.value,
      shapiro.test(yeo_obj$x.t)$p.value,
      shapiro.test(boxcox_obj$x.t)$p.value
    ),
    skewness = c(
      skewness(asinh_obj),
      skewness(yeo_obj$x.t),
      skewness(boxcox_obj$x.t)
    )
  )

  list(
    table   = results,
    objects = list(
      arcsinh     = asinh_obj,
      yeo_johnson = yeo_obj,
      boxcox      = boxcox_obj
    )
  )
}

#=========================================================
# Process X1–X4 (MATRIX INDEXING)
#=========================================================

vars <- c("X1","X2","X3","X4")
transform_results_X <- list()

for (v in vars) {
  cat("\nTesting:", v, "\n")

  x <- my.data[, v]       # MATRIX COLUMN ACCESS
  res <- test_transforms_X(x)
  print(res$table)

  transform_results_X[[v]] <- res
}

# Determine best method per X variable
best_methods_trans_X <- list()

for (v in vars) {
```

```
  df <- transform_results_X[[v]]$table
  df$score <- df$shapiro_p - abs(df$skewness)
  best_methods_trans_X[[v]] <- df$method[which.max(df$score)]
}

cat("\nBEST METHODS FOR X:\n")
print(best_methods_trans_X)

#=========================================================
# Build transformed X matrix
#=========================================================
transformed.X.matrix <- matrix(NA, nrow = nrow(my.data), ncol = length(vars))
colnames(transformed.X.matrix) <- vars

for (v in vars) {
  best <- best_methods_trans_X[[v]]
  obj  <- transform_results_X[[v]]$objects[[best]]

  if(best == "arcsinh") {
    transformed.X.matrix[, v] <- obj
  } else {
    transformed.X.matrix[, v] <- obj$x.t
  }
}

transformed.X.matrix <- apply(transformed.X.matrix, 2, as.numeric)

#=========================================================
# Process Y (column 7)
#=========================================================
cat("\nTesting Y...\n")

Y_vec <- my.data[, 7]              # CORRECT MATRIX INDEX FOR Y

resY <- test_transforms_Y(Y_vec)
print(resY$table)

dfY <- resY$table
dfY$score <- dfY$shapiro_p - abs(dfY$skewness)
best_method_trans_Y <- dfY$method[which.max(dfY$score)]

cat("\nBEST METHOD FOR Y:", best_method_trans_Y, "\n")

objY <- resY$objects[[best_method_trans_Y]]

if(best_method_trans_Y == "arcsinh") {
  transformed.Y <- objY
} else {
  transformed.Y <- objY$x.t
}

#=========================================================
# Add sr_no + create final matrix
#=========================================================

final_matrix <- cbind(
  sr_no = my.data[, 1],      # MATRIX INDEXING, NOT $
  transformed.X.matrix,
  Y_trans = transformed.Y
)

final_matrix <- apply(final_matrix, 2, as.numeric)

#=========================================================
# PLOT DISTRIBUTIONS OF BEST TRANSFORMATIONS
#=========================================================

par(mfrow=c(2,3))
for (v in vars) {
```

```
  best <- best_methods_trans_X[[v]]
  obj  <- transform_results_X[[v]]$objects[[best]]

  if(best == "arcsinh") x_t <- obj else x_t <- obj$x.t

  hist(x_t, main=paste("Best:", v, "→", best), col="skyblue", breaks=20)
}

hist(transformed.Y, main=paste("Best: Y →", best_method_trans_Y), col="orange", breaks=20)

head(final_matrix)

# ============================================================================================
# scaling method on final_matix to bring every thing between 0 and 1
# ============================================================================================

# Suppose your original matrix is 'final_matrix'
mat <- as.matrix(final_matrix)

# Columns you want to scale (for example: X1, X2, X3, x4, Y_trans)
# Adjust the column indices if needed — here I assume:
#   col-names(mat) include "X1","X2","X3","x4","Y_trans"
cols_to_scale <- c("X1","X2","X3","X4","Y_trans")

# Create a min-max scaling function
min_max <- function(x) {
  (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}

# Make a copy, scale selected columns, keep as matrix
scaled_data <- mat
scaled_data[, cols_to_scale] <- apply(mat[, cols_to_scale, drop = FALSE], 2, min_max)

# Optionally check:
head(scaled_data)
summary(scaled_data[, cols_to_scale])


# Now plot distributions (histograms) of the scaled columns
par(mfrow = c(2, 3))  # adjust layout: here 2 rows × 3 columns of plots

for (col in cols_to_scale) {
  v <- scaled_data[, col]
  hist(v, probability = TRUE,
       main = paste("Density of", col),
       xlab = col, col = "lightgray", border = "white")
  lines(density(v, na.rm = TRUE), col = "blue", lwd = 2)
}


# Choose the columns you want to keep
keep_cols <- c("X1","X2","X3","X4","Y_trans")

# Create a new matrix containing only those columns
scaled_subset <- scaled_data[, keep_cols, drop = FALSE]

# Check structure to confirm
print(class(scaled_subset))    # should be "matrix"
print(dim(scaled_subset))      # should be n rows × 5
head(scaled_subset)

write.table(
  scaled_subset,
  file = "name-new-transformed.txt",
  sep = "\t",          # tab-separated
  row.names = FALSE,  # do not save row names
  col.names = TRUE,   # save column names
  quote = FALSE       # no quotes around values
)
```

```r
# Verify scaled & subset data
summary(scaled_subset)
apply(scaled_subset[, 1:4], 2, range)


head(scaled_subset)




# ##############################################################################################
# T3. Build models and investigate the importance of each variable.
# ##############################################################################################

# ==============================================================================================
# (i) Download the AggWaFit.R file to your working directory and load into the R work space
using,
#     source("AggWaFit718.R")
# ==============================================================================================

# Set the working directory to the folder where your assignment files are stored
setwd("E:/2. MDS DEAKIN UNIVERSITY/5. SIG 718 - Real world analytics/21. WEEK6 - Mid Term
Assignment/SIG718 submission")

# Display the current working directory to confirm it is set correctly
getwd()

# List all files available in the working directory
list.files()

# Install the 'lpSolve' package (only needs to be done once)
install.packages("lpSolve")

# Load the 'lpSolve' library for linear programming functions
library(lpSolve)

# Source the custom function file 'AggWaFit718.R'
# echo = TRUE prints each line as it is executed
# verbose = TRUE provides detailed information when sourcing
source("AggWaFit718.R", echo = TRUE, verbose = TRUE)

# List all objects in the workspace whose names include 'fit'
ls(pattern = "fit")
print(fit.QAM)

# List all objects currently in the R environment
ls()

# Display the structure of all functions available in the current environment
lsf.str()




# ==============================================================================================
# T3 (ii) Use the fitting functions to learn the parameters for
# ==============================================================================================

# ----------------------------------------------------------
# 1. Load required data and inspect structure
# ----------------------------------------------------------

# View first few rows of the scaled dataset
head(scaled_data)

# Convert selected columns to a matrix for fitter functions
data_matrix <- as.matrix(scaled_data[, c("X1","X2","X3","X4","Y_trans")])
```

```r
# Check the converted matrix
head(data_matrix)


# ------------------------------------------------------------
# 2. Fit models: WAM, WPM (p=0.5, p=2), and OWA
# ------------------------------------------------------------

# --- 2a. Weighted Arithmetic Mean (WAM) ------------------------
# Using AM (Arithmetic Mean) as generator function g
wam_model <- fit.QAM(
  data_matrix,
  output.1 = "wam_output.txt",
  stats.1  = "wam_stats.txt",
  g        = AM,
  g.inv    = invAM
)


# --- 2b. Weighted Power Mean, p = 0.5 --------------------------
# g = PM(x, p=0.5) and inverse = invPM05
wpm_p05_model <- fit.QAM(
  data_matrix,
  output.1 = "wpm_p05_output.txt",
  stats.1  = "wpm_p05_stats.txt",
  g        = function(x) PM(x, p = 0.5),
  g.inv    = invPM05
)


# --- 2c. Weighted Power Mean, p = 2 ----------------------------
# g = PM(x, p=2); inverse = invQM (Quadratic Mean inverse)
wpm_p2_model <- fit.QAM(
  data_matrix,
  output.1 = "wpm_p2_output.txt",
  stats.1  = "wpm_p2_stats.txt",
  g        = function(x) PM(x, p = 2),
  g.inv    = invQM   # Provided by your AggWaFit script
)


# --- 2d. Ordered Weighted Averaging (OWA) ----------------------
owa_model <- fit.OWA(
  data_matrix,
  output.1 = "owa_output.txt",
  stats.1  = "owa_stats.txt"
)

# Optional: Open WAM stats file to visually inspect
file.show("wam_stats.txt")



# ------------------------------------------------------------
# 3. Helper functions to extract learned weights from stats files
# ------------------------------------------------------------

# ---- Generic extractor for WAM & WPM (same structure) --------
extract_weights <- function(file) {
  stats <- read.table(file, header = FALSE, fill = TRUE)

  # Weight table always starts at row 6 (after metrics)
  weights <- stats[6:nrow(stats), ]

  colnames(weights) <- c("i", "w")

  return(weights$w)
}
```

```
# ---- Special extractor for OWA (different file format) --------
extract_weights_owa <- function(file) {

  # Read full file as text lines
  lines <- readLines(file)

  # Find line containing header "i w_i"
  header_line <- grep("^i\\s+w_i", lines)

  # Extract all lines after header
  weight_lines <- lines[(header_line + 1):length(lines)]

  # Parse these lines as a table
  df <- read.table(text = weight_lines, header = FALSE)

  # Convert weights to numeric
  weights <- as.numeric(df$V2)

  return(weights)
}



# ------------------------------------------------------------
# 4. Extract weights for all models from stats files
# ------------------------------------------------------------

# WAM weights
weights_wam <- extract_weights("wam_stats.txt")

# WPM p = 0.5 weights
weights_wpm05 <- extract_weights("wpm_p05_stats.txt")

# WPM p = 2 weights
weights_wpm2 <- extract_weights("wpm_p2_stats.txt")

# OWA weights
weights_owa <- extract_weights_owa("owa_stats.txt")


# Inspect extracted weights
weights_wam
weights_wpm05
weights_wpm2
weights_owa



# ##############################################################################################
# T4. Use your model for prediction.
# ##############################################################################################

# ============================================================================================
# (i) Use your best fitting model from T3, i.e., WAM, WPM(0.5), WPM(2), or OWA, to predict Y
(Appliances)
#     for the following inputs:
#
#                     X1= 23.5, X2=35.87125, X3=24.89, X4=32.93, X5=758.55
#
#     You must use the same pre-processing as in T2.
# ============================================================================================

# ============================================================================================
# Convert all weight vectors to numeric
# ============================================================================================
weights_wam    <- as.numeric(weights_wam)
weights_wpm05  <- as.numeric(weights_wpm05)
```

```
weights_wpm2   <- as.numeric(weights_wpm2)
weights_owa    <- as.numeric(weights_owa)

cat("Numeric Weights Loaded:\n")
print(list(
  WAM = weights_wam,
  WPM05 = weights_wpm05,
  WPM2 = weights_wpm2,
  OWA = weights_owa
))

# ============================================================================================
# 1. New Input
# ============================================================================================
new_raw <- c(X1 = 23.5, X2 = 35.87125, X3 = 24.89, X4 = 32.93)
vars <- c("X1", "X2", "X3", "X4")

# ============================================================================================
# 2. Transform new input using stored best transformation
# ============================================================================================
new_transformed <- numeric(length(vars))
names(new_transformed) <- vars

for (v in vars) {
  best <- best_methods_trans_X[[v]]
  obj  <- transform_results_X[[v]]$objects[[best]]

  if (best == "arcsinh") {
    new_transformed[v] <- asinh(new_raw[v])
  } else {
    new_transformed[v] <- predict(obj, new_raw[v])
  }
}

# ============================================================================================
# 3. Scale using training min-max
# ============================================================================================
training_min <- apply(transformed.X.matrix[, vars], 2, min)
training_max <- apply(transformed.X.matrix[, vars], 2, max)

new_scaled <- (new_transformed - training_min) / (training_max - training_min)

cat("\nScaled Inputs:\n")
print(new_scaled)


# ============================================================================================
# 4. Inverse scaling & inverse transform for Y
# ============================================================================================
y_trans <- scaled_subset[, "Y_trans"]
min_Y_trans <- min(y_trans)
max_Y_trans <- max(y_trans)

inverse_scale_Ytrans <- function(y_scaled) {
  y_scaled * (max_Y_trans - min_Y_trans) + min_Y_trans
}

inverse_transform_Y <- function(y_trans_val) {
  predict(objY, newdata = y_trans_val, inverse = TRUE)
}


# ============================================================================================
# 5. MODEL 1 — OWA
# ============================================================================================
predict_OWA <- function(x, weights) {
  sorted_x <- sort(x, decreasing = TRUE)
  sum(weights * sorted_x)
}
```

```
owa_pred_scaled <- predict_OWA(new_scaled, weights_owa)
owa_pred_Ytrans <- inverse_scale_Ytrans(owa_pred_scaled)
OWA_final <- inverse_transform_Y(owa_pred_Ytrans)

cat("\n----- OWA Prediction -----\n")
cat("Scaled Prediction =", owa_pred_scaled, "\n")
cat("Unscaled Y_trans  =", owa_pred_Ytrans, "\n")
cat("FINAL PREDICTED Y (OWA) =", OWA_final, "\n")


# ============================================================================================
# 6. MODEL 2 — WAM
# ============================================================================================
predict_WAM <- function(x, weights) sum(weights * x)

wam_pred_scaled <- predict_WAM(new_scaled, weights_wam)
wam_pred_Ytrans <- inverse_scale_Ytrans(wam_pred_scaled)
WAM_final <- inverse_transform_Y(wam_pred_Ytrans)

cat("\n----- WAM Prediction -----\n")
cat("Scaled Prediction =", wam_pred_scaled, "\n")
cat("Unscaled Y_trans  =", wam_pred_Ytrans, "\n")
cat("FINAL PREDICTED Y (WAM) =", WAM_final, "\n")


# ============================================================================================
# 7. MODEL 3 — WPM p = 0.5
# ============================================================================================
predict_WPM <- function(x, weights, p) {
  sum_power <- sum(weights * (x ^ p))
  sum_power^(1/p)
}

wpm05_scaled <- predict_WPM(new_scaled, weights_wpm05, p = 0.5)
wpm05_Ytrans <- inverse_scale_Ytrans(wpm05_scaled)
WPM_p05_final <- inverse_transform_Y(wpm05_Ytrans)

cat("\n----- WPM p = 0.5 Prediction -----\n")
cat("Scaled Prediction =", wpm05_scaled, "\n")
cat("Unscaled Y_trans  =", wpm05_Ytrans, "\n")
cat("FINAL PREDICTED Y (WPM p=0.5) =", WPM_p05_final, "\n")


# ============================================================================================
# 8. MODEL 4 — WPM p = 2
# ============================================================================================
wpm2_scaled <- predict_WPM(new_scaled, weights_wpm2, p = 2)
wpm2_Ytrans <- inverse_scale_Ytrans(wpm2_scaled)
WPM_p2_final <- inverse_transform_Y(wpm2_Ytrans)

cat("\n----- WPM p = 2 Prediction -----\n")
cat("Scaled Prediction =", wpm2_scaled, "\n")
cat("Unscaled Y_trans  =", wpm2_Ytrans, "\n")
cat("FINAL PREDICTED Y (WPM p=2) =", WPM_p2_final, "\n")


# ============================================================================================
cat("\n========================================\n")
cat(" All model predictions completed.\n")
cat("========================================\n")
# ============================================================================================


##############################################################################################
#(ii) Compare your prediction with the measured Y=120.
##############################################################################################

# ============================================================================================
```

```r
# MODEL COMPARISON AND ERROR ANALYSIS
# Using predictions from: WAM, WPM(p=0.5), WPM(p=2), OWA
# ============================================================================================

# ----------------------------------------------------------
# 1. Store Final Predictions and Actual Value
# ----------------------------------------------------------

# Replace these with the predictions computed earlier
pred_WAM      <- WAM_final
pred_WPM05    <- WPM_p05_final
pred_WPM2     <- WPM_p2_final
pred_OWA      <- OWA_final      # OWA prediction

# Actual measured value
actual_Y <- 120


# ----------------------------------------------------------
# 2. Create Summary Table of All Models
# ----------------------------------------------------------

model_results <- data.frame(
  Model     = c("WAM", "WPM p=0.5", "WPM p=2", "OWA"),
  Predicted = c(pred_WAM, pred_WPM05, pred_WPM2, pred_OWA),
  Actual    = actual_Y
)

# Compute absolute error for each model
model_results$Absolute_Error <- abs(model_results$Predicted - model_results$Actual)

# Display full model comparison table
print(model_results)


# ----------------------------------------------------------
# 3. Identify Best Model (Minimum Absolute Error)
# ----------------------------------------------------------

best_model <- model_results[which.min(model_results$Absolute_Error), ]

cat("\n==================== BEST MODEL ====================\n")
print(best_model)
cat("====================================================\n\n")


# ----------------------------------------------------------
# 4. Visualization Section
# ----------------------------------------------------------
library(ggplot2)
library(tidyr)


# ----------------------------------------------------------
# PLOT 1: Predicted Values vs Actual Value (Bar Plot)
# ----------------------------------------------------------


ggplot(model_results, aes(x = Model)) +
  geom_bar(aes(y = Predicted), stat = "identity", fill = "steelblue") +
  geom_hline(yintercept = actual_Y, color = "red", size = 1.3) +
  annotate("text", x = 1, y = actual_Y + 5,
           label = paste("Actual =", actual_Y), color = "red") +
  theme_minimal() +
  ylab("Predicted Value") +
  ggtitle("Model Predictions vs Actual Value (Y = 120)")


# ----------------------------------------------------------
```

```r
# PLOT 2: Absolute Error Comparison (Bar Plot)
# ------------------------------------------------------------


ggplot(model_results, aes(x = Model, y = Absolute_Error, fill = Model)) +
  geom_bar(stat = "identity") +
  theme_minimal() +
  ylab("Absolute Error") +
  ggtitle("Absolute Error of Each Model")


# ------------------------------------------------------------
# PLOT 3: Predicted vs Actual (Point + Line Plot)
# ------------------------------------------------------------


# Convert to long format for ggplot
df_long <- gather(model_results, Type, Value, Predicted:Actual)

ggplot(df_long, aes(x = Model, y = Value, color = Type, group = Type)) +
  geom_point(size = 4) +
  geom_line(size = 1) +
  theme_minimal() +
  ggtitle("Predicted vs Actual Value for All Models") +
  ylab("Value")




# ================================================================================================
# T5. Summarise your data analysis in up to 18 slides for a 5-minute video presentation
#     The slides must include the following content to support your findings.
#      All bolded terms listed below must appear in the slide titles.
#      Explanations and reasoning may be provided either verbally or in written form
# ================================================================================================



#------------------------------------------------------------------------------------------------
# BEST MODEL — Compute Absolute Error and Percentage Error
#------------------------------------------------------------------------------------------------

best_model_name  <- best_model$Model
best_pred        <- best_model$Predicted
best_actual      <- best_model$Actual

# Absolute Error
best_abs_error <- abs(best_pred - best_actual)

# Percentage Error = |Predicted - Actual| / Actual * 100
best_pct_error <- (best_abs_error / best_actual) * 100

cat("\n==================== BEST MODEL ERROR SUMMARY ====================\n")
cat("Best Model          :", best_model_name, "\n")
cat("Predicted Value     :", best_pred, "\n")
cat("Actual Value        :", best_actual, "\n")
cat("Absolute Error      :", round(best_abs_error, 4), "\n")
cat("Percentage Error    :", round(best_pct_error, 2), "%\n")
cat("==================================================================\n\n")


install.packages("gridExtra")

library(tidyr)
library(ggplot2)
library(dplyr)

# Scaled input values for the first observation
```

```
new_scaled <- scaled_subset[1, 1:4]
names(new_scaled) <- c("X1", "X2", "X3", "X4")

# Weight vectors for your models
weights_list <- list(
  WAM = c(0.316394566, 0.297810600, 0.375850847, 0.009943988),
  WPM05 = c(0.316394566, 0.297810600, 0.375850847, 0.009943988),
  WPM2 = c(0.316394566, 0.297810600, 0.375850847, 0.009943988),
  OWA = c(0.0000000, 0.4097345, 0.1587713, 0.4314942)
)

plot_model <- function(weights, model_name) {
  df <- data.frame(
    Variable = names(new_scaled),
    Scaled_Value = as.numeric(new_scaled),
    Weight = weights
  )

  df_long <- pivot_longer(df, cols = c("Scaled_Value", "Weight"),
                          names_to = "Type", values_to = "Value")

  ggplot(df_long, aes(x = Variable, y = Value, fill = Type)) +
    geom_bar(stat = "identity", position = "dodge") +
    scale_fill_manual(values = c("steelblue", "orange")) +
    theme_minimal() +
    labs(title = paste("Comparison of Weights and Scaled Input Values for", model_name),
         y = "Value",
         fill = "Legend") +
    theme(text = element_text(size=14))
}

# Create plots for all models
plots <- lapply(names(weights_list), function(m) plot_model(weights_list[[m]], m))

# Display all plots (example with gridExtra)
library(gridExtra)
grid.arrange(grobs = plots, ncol = 2)


# ============================== END OF ANALYSIS ============================================
```