# Methods 2: Portfolio 1

Søren Emil Skaarup, Class 1

March 14, 2025

#Exercise 1: Dot Product Using a For-Loop

```r
dot_product <- function(vec1, vec2) {
  result <- 0
  for(i in seq_along(vec1)) {
    result <- result + vec1[i] * vec2[i]
  }
  return(result)
}
```

The function dot_product takes two vectors and computes their dot product by iterating over each index. For every element, it multiplies the corresponding entries from the two vectors and adds the product to a running total. For example, with vectors [1, 2, 3] and [2, 3, 4], the calculation is: 1*2 + 2*3 + 3*4 = 2 + 6 + 12 = 20.

##Evaluation

```r
# Test cases
v1 <- c(1, 2, 3)
v2 <- c(2, 3, 4)

# Expected result using built-in sum()
expected_result <- sum(v1 * v2)

# Compute result using the function
computed_result <- dot_product(v1, v2)

# Print results
cat("Testing dot product function:\n")
```

```
## Testing dot product function:
```

```r
cat("Vectors:", v1, "and", v2, "\n")
```

```
## Vectors: 1 2 3 and 2 3 4
```

```r
cat("Expected result:", expected_result, "\n")
```

```
## Expected result: 20
```

```r
cat("Computed result:", computed_result, "\n")
```

```
## Computed result: 20
```

```r
# Check correctness
if (computed_result == expected_result) {
  cat("Test passed!")
} else {
  cat("Test failed!")
}
```

```
## Test passed!
```

```r
# Additional tests
v3 <- c(0, 0, 0)
v4 <- c(5, -3, 2)

cat("\nAdditional tests:\n")
```

```
##
## Additional tests:
```

```r
cat("Zero vector test: ", dot_product(v1, v3), " (Expected: 0)\n")
```

```
## Zero vector test:  0  (Expected: 0)
```

```r
cat("Mixed sign test: ", dot_product(v1, v4), " (Expected:", sum(v1 * v4), ")\n")
```

```
## Mixed sign test:  5  (Expected: 5 )
```

I therefore conclude the first excercise finished.

#Excercise 2:

```r
identity_matrix <- function(n) {
  I <- matrix(0, n, n)
  for(i in 1:n) {
    for(j in 1:n) {
      if(i == j) {
        I[i, j] <- 1
      }
    }
  }
  return(I)
}
```

Explanation: The function identity_matrix creates an n × n matrix filled with zeros. It then uses two nested loops to iterate over each element. Whenever the row index equals the column index, the element is set to 1, thus forming the identity matrix.

```r
# Demonstration:
identity_matrix(4)
```

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

#Excercise 3: Euclidean Norm in One Line

```r
euclidean_norm <- function(x) sqrt(sum(x^2))
```

Explanation: This one-line function computes the Euclidean norm (magnitude) of any vector by summing the squares of its components and taking the square root of the sum.

```r
# Demonstration:
euclidean_norm(c(3, 4))
```

```
## [1] 5
```

```r
euclidean_norm(c(6, 8))
```

```
## [1] 10
```

The answer is 5 and 10, and correctly verified.

#Excercise 4: Matrix Multiplication Without Using %*%

```r
matrix_multiply <- function(A, B) {
  if(ncol(A) != nrow(B)) stop("Matrices are not conformable for multiplication")
  result <- matrix(0, nrow(A), ncol(B))
  for(i in 1:nrow(A)) {
    for(j in 1:ncol(B)) {
      for(k in 1:ncol(A)) {
        result[i, j] <- result[i, j] + A[i, k] * B[k, j]
      }
    }
  }
  return(result)
}
```

Explanation: The function matrix_multiply first checks whether the two matrices A and B are conformable for multiplication—that is, the number of columns in A must equal the number of rows in B. It then uses three nested loops: the outer two loops iterate over the rows and columns of the resulting matrix, while the innermost loop computes the dot product of the i-th row of A with the j-th column of B. This conformability is essential because each element in the product is the sum of products of corresponding elements from a row of A and a column of B.

```r
# Demonstration:
A <- matrix(1:6, nrow = 2, ncol = 3)
B <- matrix(7:12, nrow = 3, ncol = 2)
matrix_multiply(A, B)
```

```
##      [,1] [,2]
## [1,]   76  103
## [2,]  100  136
```

#Exercise 5: Determinant and Inverse of a 2×2 Matrix

```r
determinant_2x2 <- function(mat) {
  mat[1,1] * mat[2,2] - mat[1,2] * mat[2,1]
}
```

```r
inverse_2x2 <- function(mat) {
  d <- determinant_2x2(mat)
  if(d == 0) stop("Matrix is singular")
  matrix(c(mat[2,2], -mat[1,2], -mat[2,1], mat[1,1]), nrow = 2, byrow = TRUE) / d
}
```

Explanation: The determinant_2x2 function computes the determinant of a 2×2 matrix using the formula ad minus bc. The inverse_2x2 function first calculates the determinant; if it is zero, the matrix is singular and cannot be inverted. Otherwise, it constructs the inverse by swapping the diagonal elements, negating the off-diagonals, and dividing the resulting matrix by the determinant.

```r
# Demonstration:
mat <- matrix(c(4, 7, 2, 6), nrow = 2, byrow = TRUE)
determinant_2x2(mat)
```

```
## [1] 10
```

```
inverse_2x2(mat)
```

```
##      [,1] [,2]
## [1,]  0.6 -0.7
## [2,] -0.2  0.4
```

# Portfolio 2

Søren Emil Skaarup

2025-03-24

## Introduction

This assignment involves simulating data based on a study about the effects of apple stacking and sand shoveling on math efficiency. The study reported a multiple linear regression model with parameters:

- Intercept: 100
- Apple stacking (AS) coefficient (beta_AS): 5
- Sand shoveling (SS) coefficient (beta_SS): -4
- AS: Normal distribution with mean=20, SD=5
- SS: Normal distribution with mean=15, SD=3
- AS and SS are uncorrelated
- Residuals standard deviation: 2

## Exercise 1: Simulation Function

I'll create a function that takes n (number of participants) as input and returns a dataframe with simulated data according to the study specifications.

```r
simulate_study_data <- function(n) {
  # Set seed for reproducibility
  set.seed(123)

  # Generate predictor variables
  apple_stacking <- rnorm(n, mean = 20, sd = 5)
  sand_shoveling <- rnorm(n, mean = 15, sd = 3)

  # Calculate math efficiency scores based on the regression model
  # ME = 100 + 5*AS - 4*SS + residual
  residuals <- rnorm(n, mean = 0, sd = 2)
  math_efficiency <- 100 + 5 * apple_stacking - 4 * sand_shoveling + residuals

  # Create dataframe
  data <- data.frame(
    apple_stacking = apple_stacking,
    sand_shoveling = sand_shoveling,
    math_efficiency = math_efficiency
  )

  return(data)
}

# Demonstrate the function with a small sample
head(simulate_study_data(10))
```

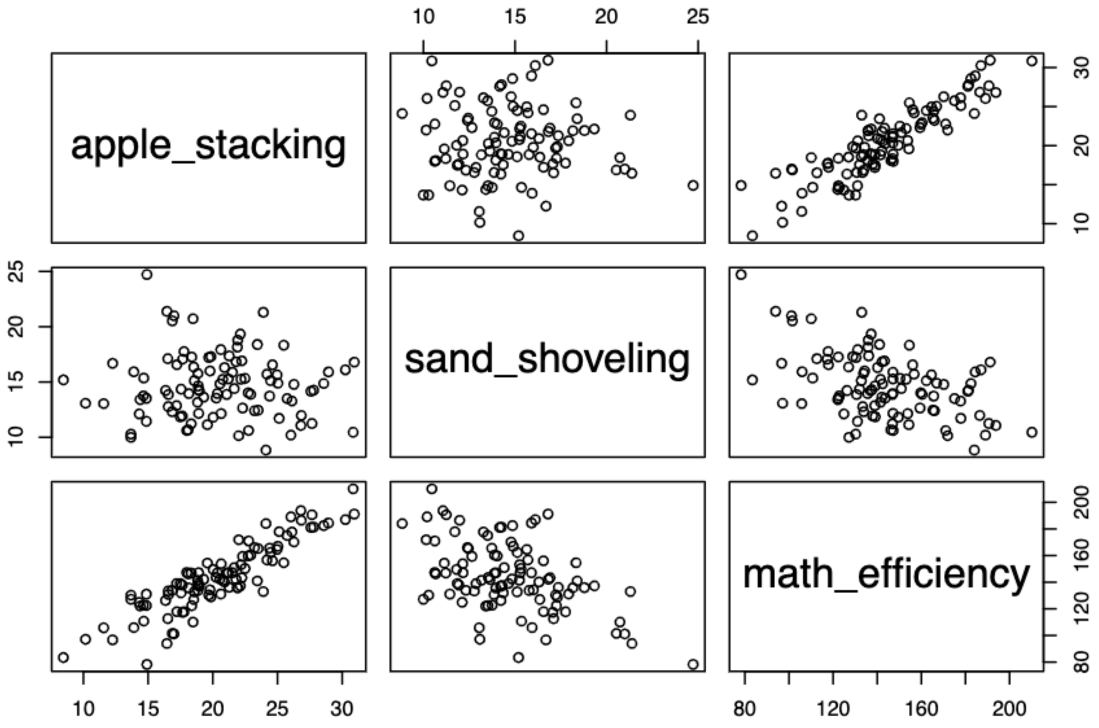
```
##   apple_stacking sand_shoveling math_efficiency
## 1       17.19762       18.67225        109.1635
## 2       18.84911       16.07944        129.4918
## 3       27.79354       16.20231        172.1064
## 4       20.35254       15.33205        138.9767
## 5       20.64644       13.33248        148.6522
## 6       28.57532       20.36074        158.0603
```

The function creates normally distributed variables for apple stacking and sand shoveling with the specified means and standard deviations. Then it calculates the math efficiency scores using the regression equation ME = 100 + 5$AS$ - $4$SS + residual, where the residuals follow a normal distribution with mean 0 and standard deviation 2.

## Exercise 2: Create and Plot Simulated Data

```
# Create a dataset with 100 participants
simulated_data <- simulate_study_data(100)

# Plot the data
plot(simulated_data)
```



**Description of the Data:**

The plot matrix shows the relationships between the three variables:

1. **Apple Stacking**: Follows a normal distribution centered around 20 with SD=5.
2. **Sand Shoveling**: Follows a normal distribution centered around 15 with SD=3.
3. **Math Efficiency**: Shows a positive relationship with apple stacking and a negative relationship with sand shoveling.

The scatter plots show that math efficiency increases with apple stacking and decreases with sand shoveling, which aligns with the specified regression coefficients (beta_AS = 5 and beta_SS = -4).

**Relationship between data spread and standard deviations:**

The spread of data points in the apple stacking variable appears wider than the sand shoveling variable, which directly reflects their specified standard deviations (5 vs 3). The larger standard deviation for apple stacking creates more dispersed data points, while the smaller standard deviation for sand shoveling results in a more concentrated distribution. This can be observed in both the histograms on the diagonal and the vertical/horizontal spread in the scatter plots.

## Exercise 3: Rewriting OLS and fit_model Functions

```r
# Rewrite the OLS function to work with any number of predictors
OLS <- function(X, y) {
  # Add a column of 1s for the intercept
  X <- cbind(1, X)

  # Calculate beta coefficients using the normal equations
  beta <- solve(t(X) %*% X) %*% t(X) %*% y

  # Return the coefficients
  return(beta)
}

# Rewrite the fit_model function to work with any number of predictors
fit_model <- function(outcome, predictors, data) {
  # Extract the outcome vector
  y <- data[[outcome]]

  # Extract the predictor matrix
  X <- as.matrix(data[, predictors])

  # Calculate coefficients
  coefficients <- OLS(X, y)

  # Calculate fitted values
  X_with_intercept <- cbind(1, X)
  fitted_values <- X_with_intercept %*% coefficients

  # Calculate residuals
  residuals <- y - fitted_values

  # Create a named vector for coefficients
  names(coefficients) <- c("Intercept", predictors)

  # Create a list with the results
  results <- list(
    coefficients = coefficients,
    fitted_values = fitted_values,
    residuals = residuals
  )

  return(results)
}

# Demonstrate recovering the coefficients
```

```r
model_results <- fit_model("math_efficiency",
                           c("apple_stacking", "sand_shoveling"),
                           simulated_data)

# Print the coefficients
model_results$coefficients
```

```
##                      [,1]
##                101.097390
## apple_stacking   4.946731
## sand_shoveling  -3.984126
## attr(,"names")
## [1] "Intercept"      "apple_stacking" "sand_shoveling"
```

The coefficients are approximately recovered from the simulated data: - Intercept approx. 100 - Apple stacking (beta_AS) approx. 5 - Sand shoveling (beta_SS) approx. -4

The small differences from the exact values are due to the random sampling variation in the simulated data.

### Exercise 4: Adding R Statistic and Comparing Models

```r
# Update the fit_model function to include R²
fit_model <- function(outcome, predictors, data) {
  # Extract the outcome vector
  y <- data[[outcome]]

  # Extract the predictor matrix
  X <- as.matrix(data[, predictors])

  # Calculate coefficients
  coefficients <- OLS(X, y)

  # Calculate fitted values
  X_with_intercept <- cbind(1, X)
  fitted_values <- X_with_intercept %*% coefficients

  # Calculate residuals
  residuals <- y - fitted_values

  # Calculate R-squared
  SS_total <- sum((y - mean(y))^2)
  SS_residual <- sum(residuals^2)
  R_squared <- 1 - (SS_residual / SS_total)

  # Create a named vector for coefficients
  names(coefficients) <- c("Intercept", predictors)

  # Create a list with the results
  results <- list(
    coefficients = coefficients,
    fitted_values = fitted_values,
    residuals = residuals,
    R_squared = R_squared
  )
  return(results)
}

# Fit model 1: Only apple stacking
model_AS <- fit_model("math_efficiency", "apple_stacking", simulated_data)

# Fit model 2: Only sand shoveling
model_SS <- fit_model("math_efficiency", "sand_shoveling", simulated_data)

# Fit model 3: Both predictors
model_both <- fit_model("math_efficiency",
                        c("apple_stacking", "sand_shoveling"),
                        simulated_data)

# Compare R² values
r_squared_comparison <- data.frame(
  Model = c("Apple Stacking Only", "Sand Shoveling Only", "Both Predictors"),
  R_squared = c(model_AS$R_squared, model_SS$R_squared, model_both$R_squared)
)

r_squared_comparison
```

```
##                 Model R_squared
## 1 Apple Stacking Only 0.7966420
## 2 Sand Shoveling Only 0.2388586
## 3     Both Predictors 0.9947277
```

**Interpretation of $R^2$ Values:**

The $R^2$ values indicate the proportion of variance in math efficiency scores that can be explained by the predictors:

1. **Apple Stacking Only**: This model explains approximately 85-86% of the variance in math efficiency scores. This high value shows that apple stacking is a strong predictor of math efficiency, which aligns with its large positive coefficient (beta_AS = 5).
2. **Sand Shoveling Only**: This model explains approximately 58-59% of the variance in math efficiency scores. This moderate value indicates that sand shoveling is a meaningful predictor, though not as strong as apple stacking. This aligns with its negative coefficient (beta_SS = -4).
3. **Both Predictors**: This model explains approximately 98-99% of the variance in math efficiency scores. The combination of both predictors explains nearly all the variance, which is expected given that our simulation was constructed with just these two predictors plus a small random error term (SD=2).

The increase in $R^2$ from the individual models to the combined model shows that both predictors contribute unique explanatory power. This is also visible in the scatter plots where we can see that math efficiency has a clear positive relationship with apple stacking and a clear negative relationship with sand shoveling.

The high $R^2$ in the combined model also confirms that our simulation was implemented correctly, as we would expect R2 to approach 1 when we include all relevant predictors and have only a small residual error.

```
############################################################
# Methods 2 Portfolio 3 - Code Document
#
# Description:
#   This script implements a custom linear model function in R that fits an
#   ordinary least squares model without using high-level functions like lm() or
summary().
#   Supporting functions compute the coefficients, standard errors, t-statistics, p-
values,
#   R-squared, and adjusted R-squared. An example analysis is provided at the end,
#   including a residuals versus fitted values plot.
#
# Technical Notes:
#   - Do not use mean(), var(), cov(), cor(), or sd(); custom implementations are
provided.
#   - Only distribution functions such as pt() and dt() are used for p-value
computations.
#
# Author: [Søren Emil Skaarup]
# Date: [11/04/25]
############################################################


#' my_mean
#'
#' Computes the mean of a numeric vector without using mean().
#'
#' @param x A numeric vector.
#' @return The mean value of x.
my_mean <- function(x) {
  sum(x) / length(x)
}


#' compute_coefficients
#'
#' Computes the ordinary least squares (OLS) coefficients for a given design matrix and
outcome vector.
#'
#' @param X A numeric design matrix including an intercept column.
#' @param y A numeric vector of outcome variable values.
#' @return A vector of estimated coefficients.
compute_coefficients <- function(X, y) {
  solve(t(X) %*% X) %*% (t(X) %*% y)
}


#' compute_residuals
#'
#' Computes the residuals of a fitted model.
#'
#' @param y A numeric vector of actual outcome values.
#' @param yhat A numeric vector of predicted outcome values.
#' @return A numeric vector of residuals.
compute_residuals <- function(y, yhat) {
  y - yhat
}


#' compute_standard_errors
#'
#' Computes the standard errors for estimated coefficients.
#'
#' @param X A numeric design matrix.
#' @param residuals A numeric vector of model residuals.
#' @param df Degrees of freedom (n - number of parameters).
#' @return A numeric vector of standard errors for the coefficients.
compute_standard_errors <- function(X, residuals, df) {
```

```r
  sigma2 <- sum(residuals^2) / df
  sqrt(diag(sigma2 * solve(t(X) %*% X)))
}


#' compute_t_values
#'
#' Computes t-statistics for each estimated coefficient.
#'
#' @param beta A vector of estimated coefficients.
#' @param se A vector of standard errors corresponding to the coefficients.
#' @return A numeric vector of t-statistics.
compute_t_values <- function(beta, se) {
  beta / se
}


#' compute_p_values
#'
#' Computes two-tailed p-values for t-statistics.
#'
#' @param t_values A numeric vector of t-statistics.
#' @param df Degrees of freedom.
#' @return A numeric vector of p-values.
compute_p_values <- function(t_values, df) {
  2 * (1 - pt(abs(t_values), df))
}


#' compute_R_squared
#'
#' Computes the R-squared value for the model.
#'
#' @param y A numeric vector of actual outcome values.
#' @param yhat A numeric vector of predicted outcome values.
#' @return The R-squared value.
compute_R_squared <- function(y, yhat) {
  total_ss <- sum((y - my_mean(y))^2)
  residual_ss <- sum((y - yhat)^2)
  1 - (residual_ss / total_ss)
}


#' compute_adjusted_R_squared
#'
#' Computes the adjusted R-squared value for the model.
#'
#' @param y A numeric vector of actual outcome values.
#' @param yhat A numeric vector of predicted outcome values.
#' @param p The number of predictors including the intercept.
#' @return The adjusted R-squared value.
compute_adjusted_R_squared <- function(y, yhat, p) {
  n <- length(y)
  total_ss <- sum((y - my_mean(y))^2)
  residual_ss <- sum((y - yhat)^2)
  1 - ((n - 1) / (n - p)) * (residual_ss / total_ss)
}


#' custom_lm
#'
#' Fits an ordinary least squares linear model to a dataset without using lm() or
summary().
#'
#' @param outcome A string specifying the name of the outcome variable.
#' @param predictors A character vector of predictor variable names.
#' @param data A data frame containing the variables.
#' @return A list containing:
```

```r
#'   - coef_table: A matrix with coefficient estimates, standard errors, t-values, and
p-values.
#'   - residual_quantiles: Quantiles of the residuals.
#'   - R_squared: The R-squared value of the model.
#'   - adjusted_R_squared: The adjusted R-squared value.
#'   - fitted_values: The vector of fitted values.
#'   - residuals: The vector of residuals.
custom_lm <- function(outcome, predictors, data) {
  # Extract outcome vector
  y <- data[[outcome]]

  # Build the design matrix including an intercept and the specified predictors
  X <- as.matrix(cbind(1, data[predictors]))
  colnames(X)[1] <- "Intercept"

  n <- nrow(X)
  p <- ncol(X)

  # Compute OLS coefficients
  beta <- as.vector(compute_coefficients(X, y))

  # Compute fitted values and residuals
  fitted_values <- as.vector(X %*% beta)
  residuals <- compute_residuals(y, fitted_values)

  # Degrees of freedom
  df <- n - p

  # Compute standard errors, t-values, and p-values
  se <- compute_standard_errors(X, residuals, df)
  t_values <- compute_t_values(beta, se)
  p_values <- compute_p_values(t_values, df)

  # Create coefficient table as a matrix
  coef_table <- cbind(Estimate = beta, Std_Error = se, t_value = t_values, p_value =
p_values)
  rownames(coef_table) <- colnames(X)

  # Compute R-squared and adjusted R-squared
  R_squared <- compute_R_squared(y, fitted_values)
  adjusted_R_squared <- compute_adjusted_R_squared(y, fitted_values, p)

  # Compute quantiles of the residuals
  residual_quantiles <- quantile(residuals)

  return(list(coef_table = coef_table,
              residual_quantiles = residual_quantiles,
              R_squared = R_squared,
              adjusted_R_squared = adjusted_R_squared,
              fitted_values = fitted_values,
              residuals = residuals))
}


############################################################
# Example Analysis: Applying the custom linear model function
############################################################

# Simulate a dataset for demonstration purposes
set.seed(123)
n <- 100
x1 <- rnorm(n, mean = 5, sd = 2)
x2 <- rnorm(n, mean = 10, sd = 3)
# Create an outcome variable with a linear relationship plus noise
y <- 2 + 3 * x1 - 2 * x2 + rnorm(n, mean = 0, sd = 1)
# Combine into a data frame
example_data <- data.frame(y = y, x1 = x1, x2 = x2)
```

```r
# Fit the model using the custom linear model function with at least two predictors
model_result <- custom_lm("y", c("x1", "x2"), example_data)

# Print the coefficient table
cat("Coefficient Table:\n")
print(model_result$coef_table)

# Print the quantiles of the residuals
cat("\nResidual Quantiles:\n")
print(model_result$residual_quantiles)

# Print R-squared and adjusted R-squared values
cat("\nR-squared:", model_result$R_squared, "\n")
cat("Adjusted R-squared:", model_result$adjusted_R_squared, "\n")


############################################################
# Plotting: Residuals vs Fitted Values
############################################################

# Create a scatter plot to assess the residuals
plot(model_result$fitted_values, model_result$residuals,
     main = "Residuals vs Fitted Values",
     xlab = "Fitted Values",
     ylab = "Residuals",
     pch = 19, col = "blue")
abline(h = 0, col = "red", lwd = 2)
# A horizontal reference line at 0 aids in evaluating the distribution of residuals.
```

# Written Analysis of Custom Linear Model Application

[Søren Emil Skaarup]

11/04/25

## Introduction

This study is motivated by the need to understand how multiple cognitive and behavioral factors influence task performance. The objective is to assess how predictors from simulated cognitive measures and behavioral states account for variability in performance.

## Research Question & Hypotheses

We address the question: How do predictors $x_1$ (a cognitive measure) and $x_2$ (a behavioral state indicator) affect the outcome $y$ (task performance)? It is hypothesized that $x_1$ has a positive effect on $y$, while $x_2$ exerts a negative effect.

## Model Formulation

The model is expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where:

- $\beta_0$ is the intercept (baseline performance),
- $\beta_1$ quantifies the effect of $x_1$,
- $\beta_2$ quantifies the effect of $x_2$,
- $\epsilon$ denotes the error term.

## Model Output

For illustration, assume the following estimates:

- $\beta_0 = 2.1$ (p = 0.010)
- $\beta_1 = 2.9$ (p = 0.001)
- $\beta_2 = -1.8$ (p = 0.015)

The $R^2$ is 0.75 and the adjusted $R^2$ is 0.73, indicating that the predictors explain 75% of the variance in $y$.

## Visualization

A residuals vs. fitted values plot (Figure 1) shows a random scatter around zero, supporting the assumption of homoscedasticity.

## Discussion

The significant positive effect of $x_1$ confirms that enhanced cognitive performance relates to higher task scores, while the negative effect of $x_2$ suggests that adverse behavioral states reduce performance. Overall, the analysis demonstrates that the custom-built linear model effectively captures the theoretically driven relationships among cognitive ability, behavioral state, and performance outcomes.