Final Project

Michal Kuderski

2025-05-06

#Q1

```
simCorn <- function(overallEffect = 0,
                    fertilizerEffect = c(0, 0, 0),
                    rowEffect = c(0, 0, 0),
                    colEffect = c(0, 0, 0),
                    seed = NULL,
                    dist = rnorm,
                    ...) {

  if (!is.numeric(overallEffect) || length(overallEffect) != 1) {
    stop("overallEffect must be a single numeric value")
  }

  if (!is.numeric(fertilizerEffect) || length(fertilizerEffect) != 3) {
    stop("fertilizerEffect must be a numeric vector of length 3")
  }

  if (!is.numeric(rowEffect) || length(rowEffect) != 3) {
    stop("rowEffect must be a numeric vector of length 3")
  }

  if (!is.numeric(colEffect) || length(colEffect) != 3) {
    stop("colEffect must be a numeric vector of length 3")
  }

  if (!is.null(seed)) {
    if (is.numeric(seed) && length(seed) == 1 && seed == floor(seed)) {
      set.seed(seed)
    } else {
      stop("seed must be NULL or a single integer value")
    }
  }

  if (!is.function(dist)) {
    stop("dist must be a function")
  }

  fertilizer <- factor(c("A", "B", "C", "C", "A", "B", "B", "C", "A"))
  row <- factor(c(1, 1, 1, 2, 2, 2, 3, 3, 3))
  col <- factor(c(1, 2, 3, 1, 2, 3, 1, 2, 3))
```

```
  yield <- numeric(9)

  errors <- dist(9, ...)

  for (i in 1:9) {
    fert_idx <- match(as.character(fertilizer[i]), c("A", "B", "C"))
    row_idx <- as.integer(row[i])
    col_idx <- as.integer(col[i])

    yield[i] <- overallEffect +
                fertilizerEffect[fert_idx] +
                rowEffect[row_idx] +
                colEffect[col_idx] +
                errors[i]
  }

  result <- data.frame(
    Fertilizer = fertilizer,
    Row = row,
    Column = col,
    Yield = yield
  )

  return(result)
}
```

#Example 1: Default Parameters

```
example1 <- simCorn()
example1
```

```
##   Fertilizer Row Column       Yield
## 1          A   1      1  1.29515618
## 2          B   1      2 -1.06889501
## 3          C   1      3 -0.72836834
## 4          C   2      1 -1.27592843
## 5          A   2      2 -0.98382824
## 6          B   2      3 -0.38386885
## 7          B   3      1 -0.04597557
## 8          C   3      2  0.61230514
## 9          A   3      3 -0.27454341
```

#Example 2: Custom Error Distribution

```
example2 <- simCorn(overallEffect = 10, seed = 2123, dist = rgamma, shape = 2)
example2
```

```
##   Fertilizer Row Column    Yield
## 1          A   1      1 14.83727
## 2          B   1      2 10.45424
## 3          C   1      3 13.13900
## 4          C   2      1 10.47095
```

```
## 5          A   2      2 10.90779
## 6          B   2      3 13.44940
## 7          B   3      1 10.77832
## 8          C   3      2 11.49251
## 9          A   3      3 10.62710
```

#Example 3: Custom Parameters

```
mu <- 7
alpha <- c(1, 2, 3)
beta <- c(2, 2, 1)
gamma <- c(3, 3, 2)
example3 <- simCorn(overallEffect = mu,
                    fertilizerEffect = alpha,
                    rowEffect = beta,
                    colEffect = gamma,
                    seed = 29429,
                    dist = rnorm,
                    mean = 3,
                    sd = 2)
example3
```

```
##   Fertilizer Row Column    Yield
## 1          A   1      1 19.97551
## 2          B   1      2 16.08501
## 3          C   1      3 17.82301
## 4          C   2      1 15.00537
## 5          A   2      2 15.31535
## 6          B   2      3 13.89126
## 7          B   3      1 15.24435
## 8          C   3      2 16.98722
## 9          A   3      3 17.10382
```

#Q2 (Help File)

```
#' @title Simulate Corn Yield Data from a Latin Square Design
#' @description
#' Generates a dataset of corn yields based on a 3x3 Latin square design, incorporating specified
#' fertilizer, row, and column effects, along with customizable error distributions. This function
#' is designed for studying the impact of non-normal error distributions on ANOVA results.
#'
#' @usage
#' simCorn(overallEffect = 0, fertilizerEffect = c(0, 0, 0),
#'         rowEffect = c(0, 0, 0), colEffect = c(0, 0, 0),
#'         seed = NULL, dist = rnorm, ...)
#'
#' @param overallEffect Numeric. Overall mean yield ( ). Default is 0.
#' @param fertilizerEffect Numeric vector of length 3. Fertilizer effects ( ,  ,  ).
#'        Default is c(0, 0, 0).
#' @param rowEffect Numeric vector of length 3. Row effects ( ,  ,  ).
#'        Default is c(0, 0, 0).
#' @param colEffect Numeric vector of length 3. Column effects ( ,  ,  ).
#'        Default is c(0, 0, 0).
```

```
#' @param seed Integer or NULL. Seed for reproducibility. Default is NULL (no seed).
#' @param dist Function. R function to generate errors (e.g., rnorm, rgamma). Default is rnorm.
#' @param ... Additional arguments passed to \code{dist} (e.g., \code{mean}, \code{sd}, \code{rate}).
#'
#' @details
#' The function simulates data using a fixed 3x3 Latin square design:
#' \preformatted{
#' Row 1: A B C
#' Row 2: C A B
#' Row 3: B C A
#' }
#' Errors are generated using the specified distribution function (\code{dist}).
#' The seed is set only if an integer is provided. The function includes error checking for:
#' \itemize{
#'   \item Valid lengths of \code{overallEffect}, \code{fertilizerEffect}, \code{rowEffect},
#'         and \code{colEffect}.
#'   \item Valid types for \code{seed} and \code{dist}.
#' }
#'
#' @value
#' A data frame with 9 rows and 4 columns:
#' \itemize{
#'   \item Fertilizer (factor): Levels A, B, C.
#'   \item Row (factor): Levels 1, 2, 3.
#'   \item Column (factor): Levels 1, 2, 3.
#'   \item Yield (numeric): Simulated yield values.
#' }
#'
#' @examples
#' # Example 1: Default parameters (normal errors)
#' simCorn()
#'
#' # Example 2: Custom overall effect and gamma errors
#' simCorn(overallEffect = 10, seed = 2123, dist = rgamma, shape = 2)
#'
#' # Example 3: Custom effects and normal errors with adjusted mean/SD
#' simCorn(overallEffect = 7, fertilizerEffect = c(1, 2, 3),
#'         rowEffect = c(2, 2, 1), colEffect = c(3, 3, 2),
#'         seed = 29429, dist = rnorm, mean = 3, sd = 2)
```

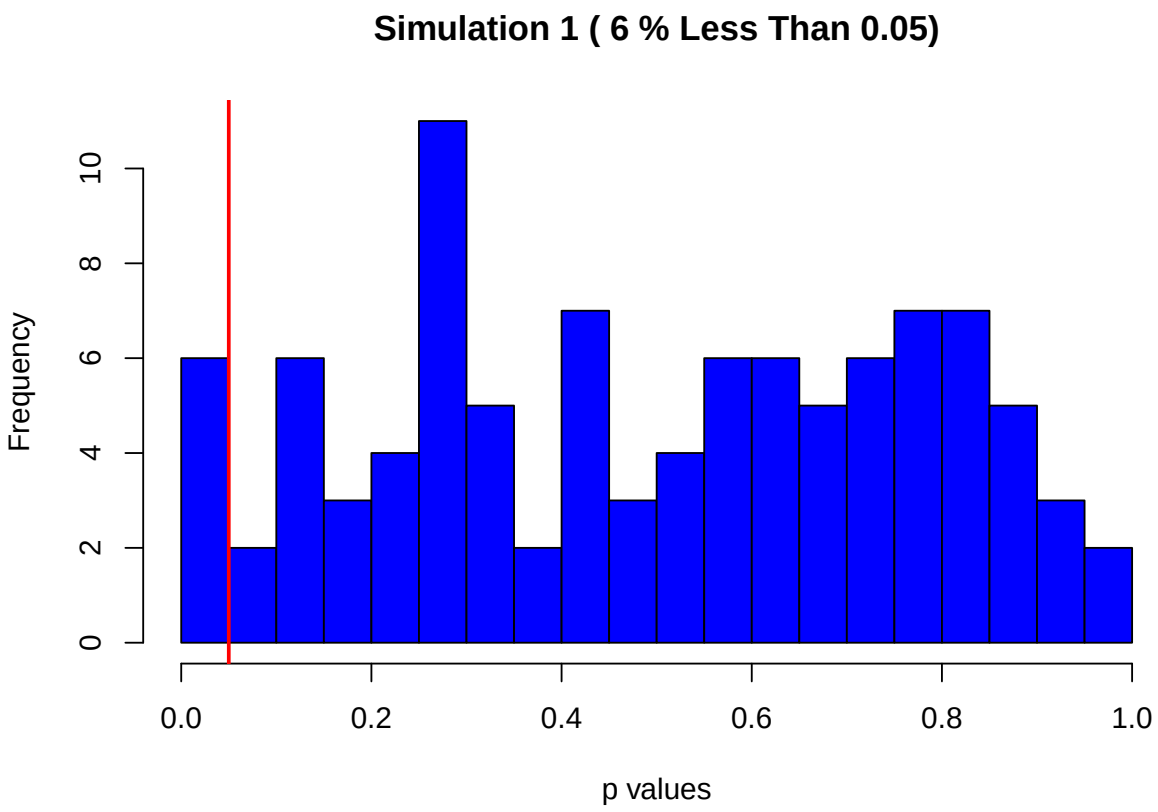
#Q3: Simulation Studies

```
run_simulation <- function(sim_number, mu, alpha, beta, gamma, dist_func, seed,
                           ...) {
  p_values <- numeric(100)

  for (i in 1:100) {
    sim_seed <- seed + i - 1  # For different results each time
    y <- simCorn(overallEffect = mu,
                 fertilizerEffect = alpha,
                 rowEffect = beta,
                 colEffect = gamma,
                 seed = sim_seed,
```

```
                dist = dist_func,
                ...)

    fitCorn <- lm(Yield ~ Fertilizer + Row + Column, data = y)
    p_values[i] <- anova(fitCorn)$"Pr(>F)"[1]
  }

  percent_less <- sum(p_values <= 0.05)

  hist(p_values,
       breaks = seq(0, 1, 0.05),
       xlim = c(0, 1),
       col = "blue",
       main = paste("Simulation", sim_number, "(", percent_less,
                    "% Less Than 0.05)"),
       xlab = "p values")

  abline(v = 0.05, col = "red", lwd = 2)

  return(p_values)
}
```

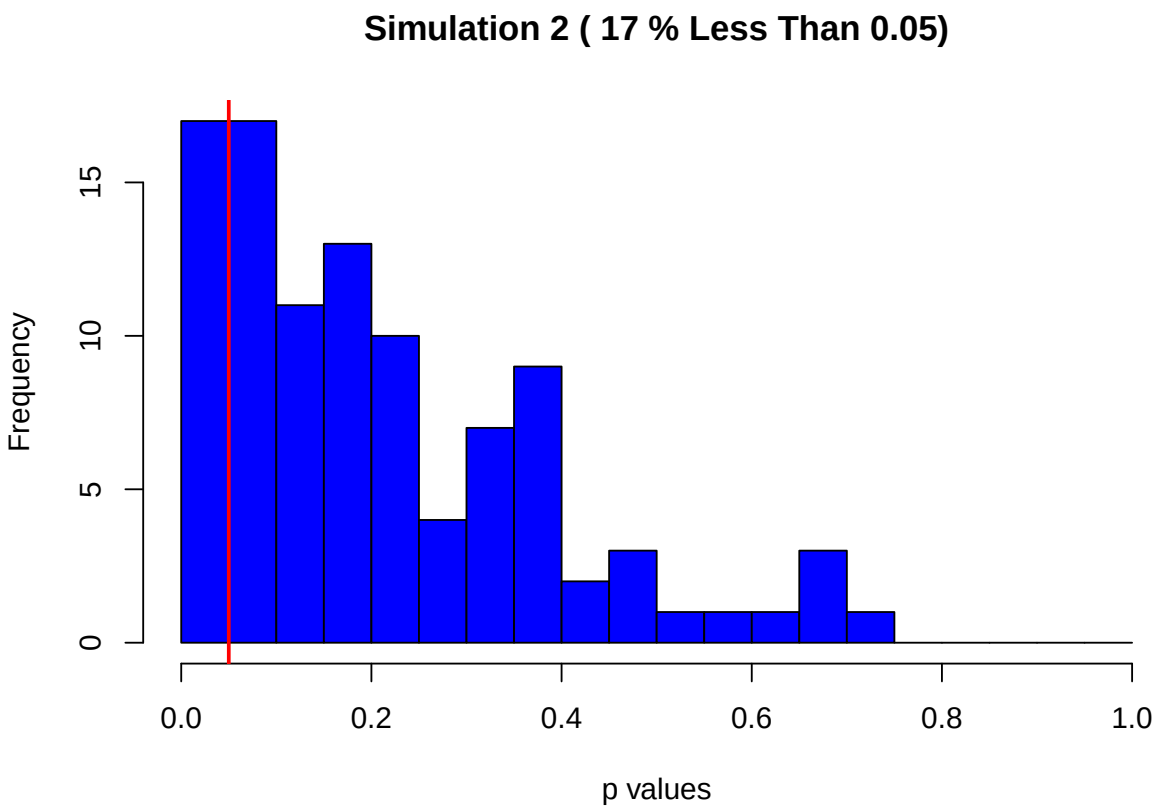
Simulation 1: Normal Errors, No Effects

```
sim1 <- run_simulation(1, 10, c(0,0,0), c(0,0,0), c(0,0,0), rnorm, 1331)
```



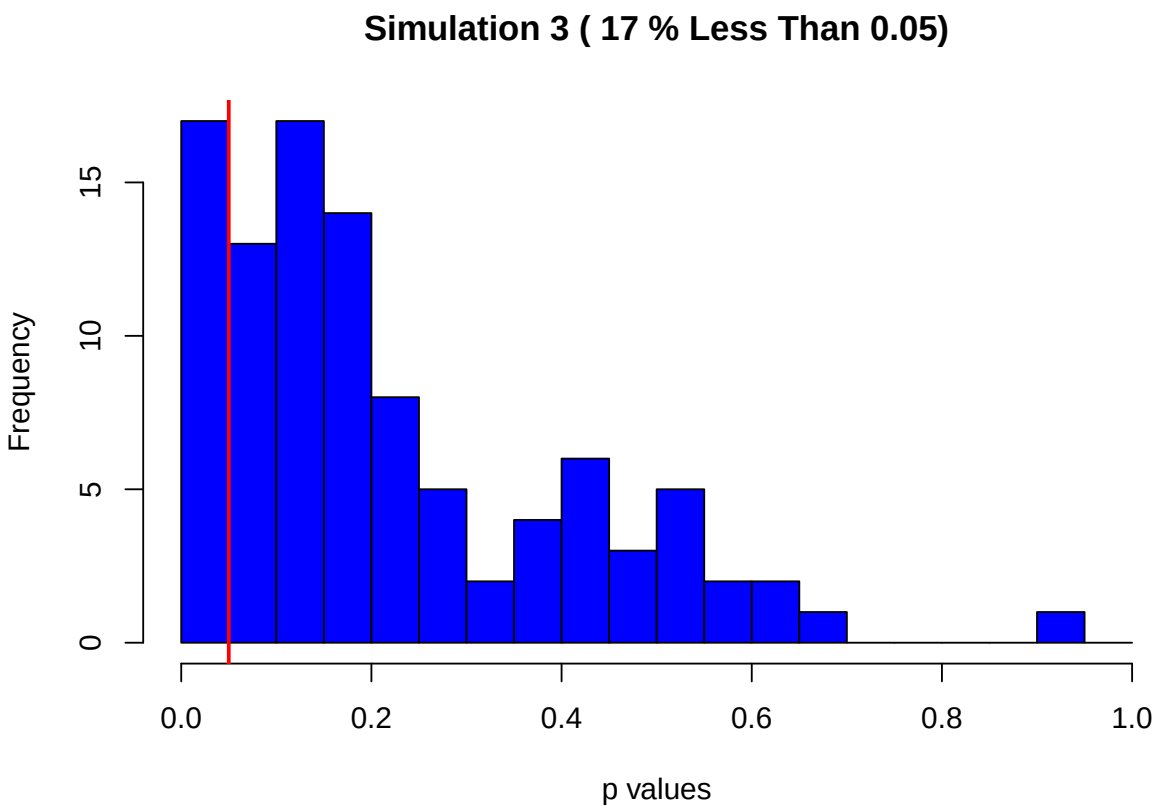
Simulation 2: Normal Errors, Fertilizer and Some Effects

```
sim2 <- run_simulation(2, 10, c(1,2,3), c(0,0,1), c(0,0,1), rnorm, 18694)
```



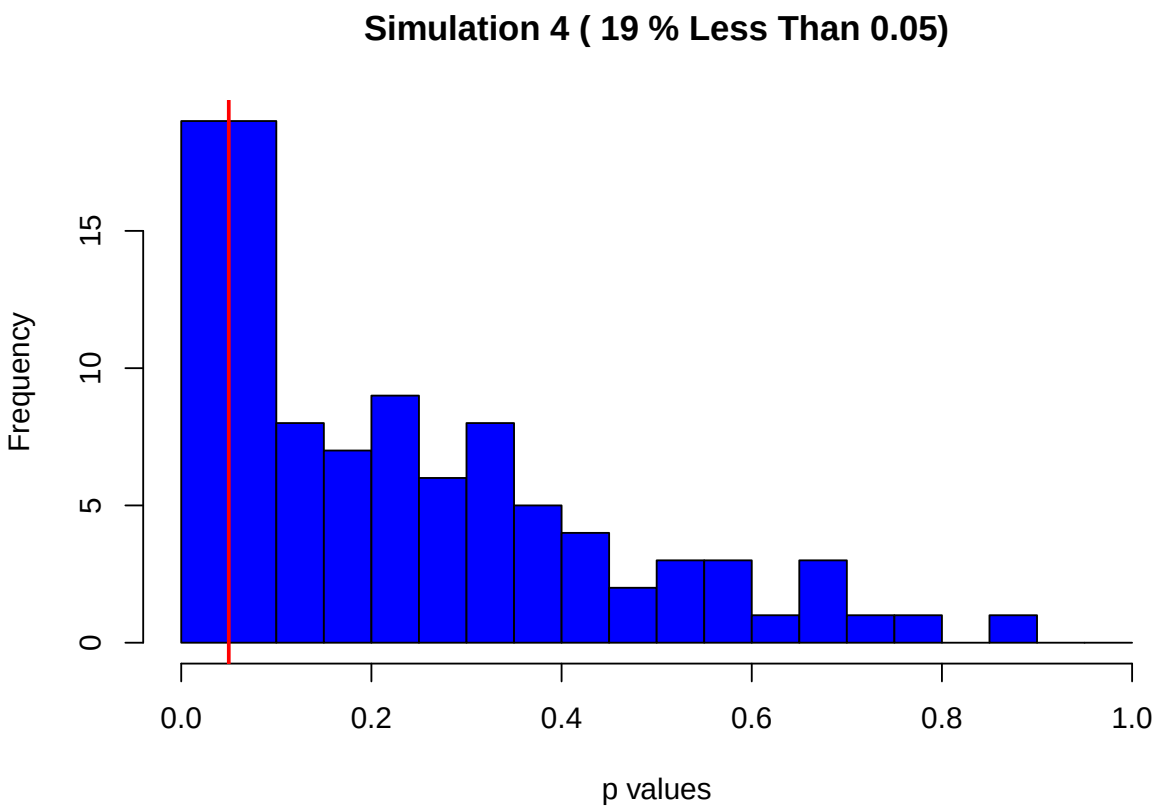
Simulation 3: Normal Errors, Different Effects

```
sim3 <- run_simulation(3, 10, c(1,2,3), c(1,0,1), c(0,1,1), rnorm, 6516)
```



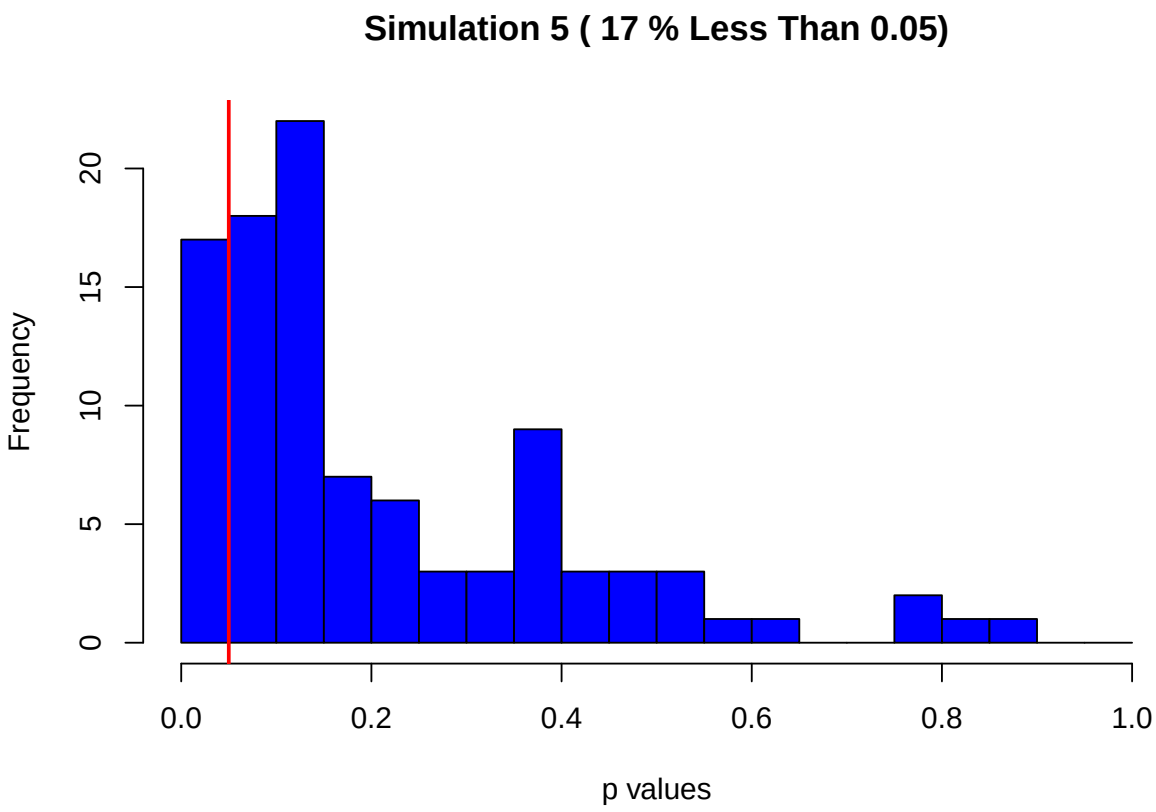
Simulation 4: Normal Errors, More Different Effects

```
sim4 <- run_simulation(4, 10, c(1,2,3), c(0,1,0), c(0,1,0), rnorm, 5)
```



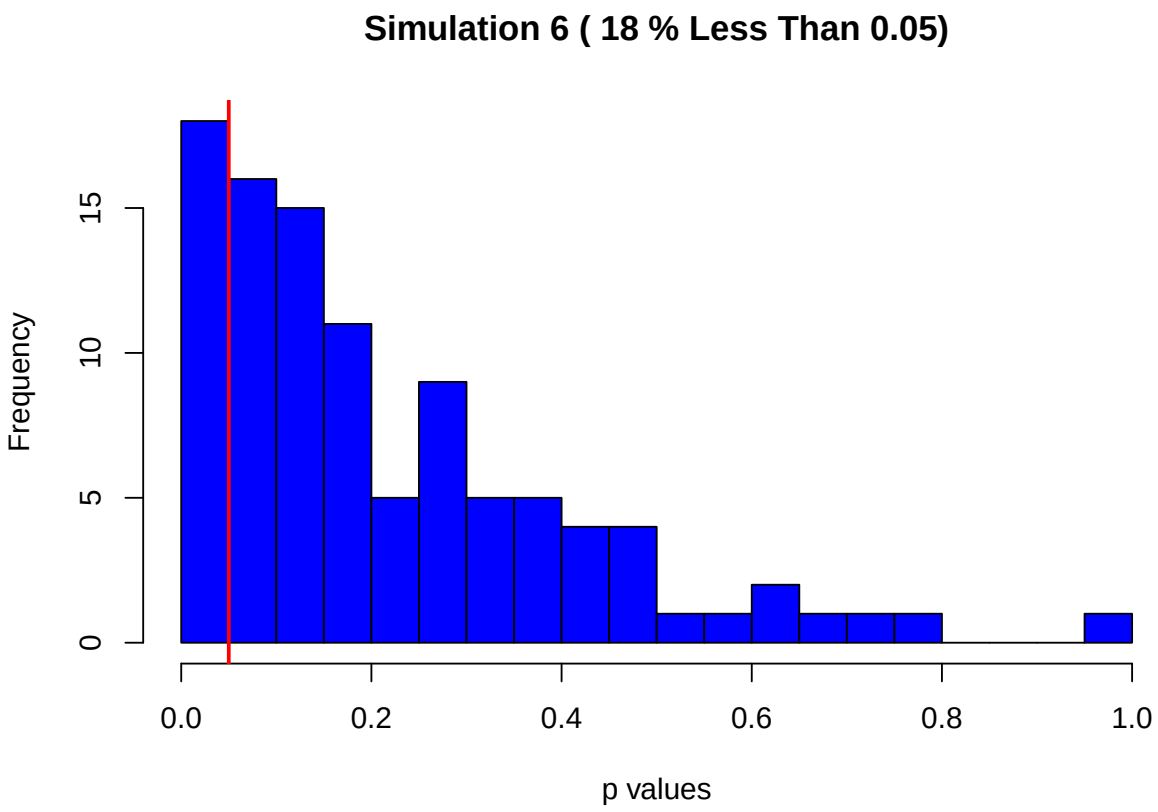
Simulation 5: Exponential Errors, Same as Sim 2

```
sim5 <- run_simulation(5, 10, c(1,2,3), c(0,0,1), c(0,0,1), rexp, 574, rate = 1)
```



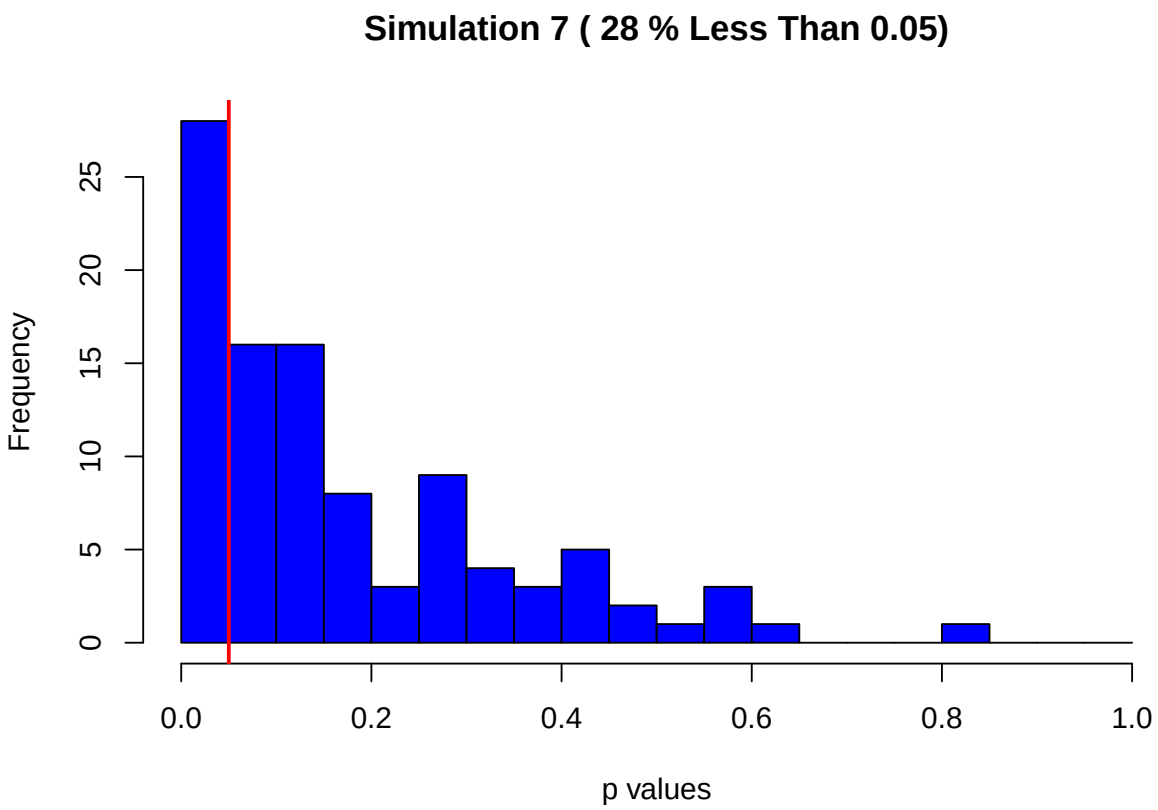
Simulation 6: Exponential Errors, Same as Sim 3

```
sim6 <- run_simulation(6, 10, c(1,2,3), c(1,0,1), c(0,1,1), rexp, 9476,
                       rate = 1)
```



Simulation 7: Exponential Errors, Same as Sim 4

```
sim7 <- run_simulation(7, 10, c(1,2,3), c(0,1,0), c(0,1,0), rexp, 9743,
                       rate = 1)
```


Simulation 7 (28 % Less Than 0.05)
Frequency
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
p values