Constructing a medium complexity shiny app for power analysis

Ronald (Ryy) Glenn Thomas

2024-12-30

Table of contents

1 Introduction

One of the most common tasks for the practicing biostatistician is the calculation of a required sample size for a two group comparison based on a two sample Student's t-test. While common, this exercise is not trivial. There are numerous parameters that need to be set. Careful consideration is required of the biostatistician and often the clinical co-investigator prior to setting the values of these parameters.

To facilitate this process we've created a Shiny app to allow realtime intereaction between parameter settings and power table and curve generation. We've included tooltips to assist the user in making choices for the paramenters.

by the team of statisticians and the study

investigators.

In this post we'll describe the process we followed to create a shiny app of medium complexity for this task.

2 Methods

2.1 User interface

2.2 Parameters

2.3 Visualization

2.4 Reporting formats

2.5 Specifying effect size

2.6 Itt and completers

3 Code

```r
# Power Analysis Shiny Application
# Combines modular UI design with efficient server-side calculations

# Required Libraries
library(shiny)
library(bslib)
library(bsicons)
library(pwr)
library(ggplot2)
library(DT)
library(rmarkdown)

# Tooltip Helper Function
create_tooltip <- function(icon = bs_icon("info-circle"), text) {
  tooltip(icon, text, placement = "right")
}

# Modular Input Sections (UI)
create_sample_size_inputs <- function() {
  tagList(
    sliderInput(
      "N",
      label = div(
        "Total Sample Size",
        create_tooltip(text = "Total number of participants in both groups combined")
      ),
      min = 20,
      max = 500,
      value = 100,
      step = 10
    ),

    sliderInput(
      "dropout",
      label = div(
        "Dropout Rate",
        create_tooltip(text = "Percentage of participants expected to withdraw before study co
      ),
      min = 0,
```

```
      max = 0.5,
      value = 0.1,
      step = 0.05
    )
  )
}

# Effect Size Method Inputs
create_effect_size_inputs <- function() {
  tagList(
    radioButtons(
      "dmeth",
      "Effect Size Calculation Method",
      choices = c(
        "Standard Deviation Units" = "std",
        "Percent Reduction" = "pct",
        "Difference in Change Scores" = "diff",
        "Change in Active Group" = "active"
      ),
      selected = "std"
    ),

    # Conditional Panels for Different Effect Size Methods
    conditionalPanel(
      condition = "input.dmeth == 'std'",
      sliderInput("del", "Effect Size in Standard Deviations",
                  min = 0, max = 2, value = c(0.2, 1.0), step = 0.1)
    ),
    conditionalPanel(
      condition = "input.dmeth == 'diff'",
      sliderInput("dff", "Difference in Scores",
                  min = 0, max = 10, value = c(1, 5), step = 0.5),
      numericInput("sd0", "Placebo Standard Deviation", value = 10)
    ),
    conditionalPanel(
      condition = "input.dmeth == 'pct'",
      sliderInput("pct", "Percent Reduction",
                  min = 0, max = 1, value = c(0.1, 0.5), step = 0.05),
      numericInput("d0", "Placebo Change", value = 10)
    ),
```

```
    conditionalPanel(
      condition = "input.dmeth == 'active'",
      sliderInput("active", "Treatment Group Change",
                  min = 0, max = 10, value = c(0, 6), step = 0.5),
      numericInput("d0", "Placebo Change", value = 10),
      numericInput("sd0", "Placebo Standard Deviation", value = 10)
    )
  )
}

# Advanced Settings Section
create_advanced_settings <- function() {
  tagList(
    checkboxInput("show_advanced", "Show Advanced Settings"),
    conditionalPanel(
      condition = "input.show_advanced == true",
      fluidRow(
        column(6,
          numericInput("ratio", "Active to Control Group Ratio",
                       value = 1, min = 0.5, max = 5, step = 0.5),
          sliderInput("dropin", "Drop-in Rate",
                      min = 0, max = 0.4, value = 0, step = 0.05)
        ),
        column(6,
          numericInput("type1", "Type I Error Rate",
                       value = 0.05, min = 0.01, max = 0.2, step = 0.005),
          checkboxInput("onesided", "One-sided Test")
        )
      )
    )
  )
}

# Main UI Definition
ui <- page_sidebar(
  title = "Power Analysis Calculator for Two-Group Designs",
  theme = bs_theme(
    version = 5,
    bootswatch = "flatly",
    primary = "#2c3e50"
```

```
  ),

  sidebar = sidebar(
    create_sample_size_inputs(),
    hr(),
    create_effect_size_inputs(),
    hr(),
    create_advanced_settings()
  ),

  layout_column_wrap(
    width = 1/2,
    card(
      full_screen = TRUE,
      card_header("Power Curve"),
      plotOutput("power_plot")
    ),

    card(
      full_screen = TRUE,
      card_header("Detailed Results"),
      DT::dataTableOutput("results_table")
    ),

    card(
      card_header("Study Design Summary"),
      verbatimTextOutput("summary_text")
    ),

    card(
      card_header("Generate Report"),
      radioButtons(
        "report_format",
        "Report Format",
        choices = c("PDF", "HTML", "Word"),
        inline = TRUE
      ),
      downloadButton("download_report", "Download Report")
    )
  )
```

```
)

# Server-side Logic
server <- function(input, output, session) {
  # Centralized effect size calculation function
  calculate_effect_size <- function(input) {
    switch(input$dmeth,
           "std" = seq(input$del[1], input$del[2], (input$del[2] - input$del[1]) / 15),
           "diff" = seq(input$del[1], input$del[2], (input$del[2] - input$del[1]) / 15) * input
           "pct" = seq(input$del[1], input$del[2], (input$del[2] - input$del[1]) / 15) * input$
           "active" = -(seq(input$del[1], input$del[2], (input$del[2] - input$del[1]) / 15) * i
    )
  }

  # Calculate sample sizes accounting for dropout and group ratio
  study_parameters <- reactive({
    req(input$N, input$sd0, input$d0)

    n1_comp <- input$ratio * input$N / (input$ratio + 1) *
               (1 - (input$dropin + input$dropout))
    n2_comp <- input$N / (input$ratio + 1) *
               (1 - (input$dropin + input$dropout))

    list(
      n1_comp = n1_comp,
      n2_comp = n2_comp,
      sided = input$onesided + 1
    )
  })

  # Robust power calculation function
  calculate_power <- function(n1, n2, effect_size, sig_level, sided) {
    tryCatch({
      pwr.t2n.test(
        n1 = n1,
        n2 = n2,
        sig.level = sig_level * sided,
        d = effect_size
      )$power
    }, error = function(e) {
```

```
    warning("Power calculation failed: ", e$message)
    return(NA)
  })
}

# Comprehensive power results calculation
power_results <- reactive({
  params <- study_parameters()
  effect_sizes <- calculate_effect_size(input)

  results <- sapply(effect_sizes, function(es) {
    calculate_power(
      params$n1_comp,
      params$n2_comp,
      es,
      input$type1,
      params$sided
    )
  })

  data.frame(
    effect_size = effect_sizes,
    power = results
  )
})

# Power Plot
output$power_plot <- renderPlot({
  req(power_results())

  results <- power_results()

  ggplot(results, aes(x = effect_size, y = power)) +
    geom_line(color = "blue") +
    geom_hline(yintercept = 0.8, color = "red", linetype = "dashed") +
    labs(
      title = "Power Curve",
      x = switch(input$dmeth,
                 "std" = "Effect Size (SD Units)",
                 "diff" = "Difference in Scores",
```

```
                  "pct" = "Percent Reduction",
                  "active" = "Treatment Group Change"
      ),
      y = "Statistical Power"
    ) +
    theme_minimal()
})

# Results Table
output$results_table <- DT::renderDataTable({
  req(power_results())
  datatable(power_results(),
            options = list(pageLength = 10),
            rownames = FALSE)
})

# Summary Text
output$summary_text <- renderText({
  req(power_results())

  paste(
    "Study Design Summary:",
    "\nTotal Sample Size:", input$N,
    "\nDropout Rate:", input$dropout * 100, "%",
    "\nEffect Size Method:", input$dmeth,
    "\nType I Error Rate:", input$type1,
    "\nOne-sided Test:", ifelse(input$onesided, "Yes", "No"),
    "\n\nPower Analysis Results:",
    "\nMaximum Power:", round(max(power_results()$power, na.rm = TRUE), 3),
    "\nEffect Size at 80% Power:",
    round(power_results()$effect_size[which.min(abs(power_results()$power - 0.8))], 3)
  )
})

# Report Download Handler
output$download_report <- downloadHandler(
  filename = function() {
    paste("power_analysis_report",
          format(Sys.Date(), "%Y%m%d"),
          ".",
```

```
            tolower(input$report_format),
            sep = "")
    },
    content = function(file) {
      # Placeholder for report generation
      # In a real-world scenario, you'd use rmarkdown to generate a comprehensive report
      report_data <- list(
        power_results = power_results(),
        plot = power_plot()
      )

      # Example: Simple report generation (customize as needed)
      writeLines(
        c("Power Analysis Report",
          "--------------------",
          capture.output(report_data)),
        file
      )
    }
  )
}

# Launch the Shiny App
shinyApp(ui, server)
```

4 setting up modules

5 setting up with golem. step by step

Start with the video

Building a basic Shiny app with Golem - Part I - YouTube

Notes: rstudio provides a template under new file

6 References

How to build a professional R Shiny app — part 1 | by Adrian Joseph, PhD | Towards Dev

How to build a professional R Shiny app — part 2 | by Adrian Joseph, PhD | Towards Dev

How to build a professional R Shiny app — part 3 | by Adrian Joseph, PhD | Towards Dev

Welcome | Outstanding User Interfaces with Shiny

Want to make it extensible: try Golem.

Introduction | Engineering Production-Grade Shiny Apps