Power Analysis App

Astra S. Bryant

5/3/2020

Introduction/Abstract

The goal of this project is to generate a Shiny web app for conducing simple power analyses on user-provided data. Most online resources currently require users to provide pre-processed data in the form of individual and group averages. Excel spreadsheets with embedded lookup tables for calculating power analyses require users to be comfortable with several advanced excel features, and assumes familiarity with lookup tables such as those provided by Bausell and Li (2002).

These requirements may act as a barrier for some researchers. The overall goal of this project is to encourage researchers to conduct power analyses on pilot data by providing a user-friendly web-based application that takes a csv file with raw data as input, and calculates sample sizes for specified power/alpha levels over a set of statistical tests. As much as possible, the results provided by the script have been cross-validating using G*Power.

Dependencies

The shiny script is located in the file `Power_Analysis_App.R`, which in turn calls `ComputeSampleSize.R` for the actual calculations. Taken together, the app uses several libraries to calculate effect sizes and power analyses for the separate statistical tests.
Key amongst these is the WebPower libary, which handles the power analyses. For background/additional references relating to the WebPower library, the manual is avaliable here.

```
library(shiny)
library(WebPower)
library(reshape2)
library(effsize)
library(tidyverse)
library(readxl)
library(DescTools)
library(powerAnalysis)
library(shinythemes)
library(rmarkdown)
library(openxlsx)
library(knitr)
library(xtable)
source("Server/ComputeSampleSize.R")
```

Define UI

The user interface was designed to take either an excel file or a .cvs file input containing raw data that will act as pilot data for calculating an effect size. The effect size will them be used to calculate the required sample size, given selected alpha and power levels. Users can select the alpha/power levels they desire; defaults are set to an alpha level of 0.01 and a power level of 0.9.

```
ui <- fluidPage(
        theme = shinytheme("darkly"),

        # App title
        titlePanel("Sample Size Calculator"),

        # Sidebar layout with input and output definitions
        fluidRow(

                ## Sidebar panel for inputs
                #sidebarPanel(
                source("UI/sidebar-ui.R", local = TRUE)$value,

                # Main panel for displaying outputs
                source("UI/mainPanel-ui.R", local = TRUE)$value
        ),

        # Application Instructions
        fluidRow(
                source("UI/appInstructions-ui.R", local = TRUE)$value
        ),

        # About this App (Static)
        fluidRow(
                source("UI/appNotes-ui.R", local = TRUE)$value
        )
)
```

UI Sidebar

Source = `UI/sidebar-ui.R`

```
column(4,
       wellPanel(
         # Input: Select a file
         div(h4("Data Input"),
             class = "text-primary"),
         fileInput("loadfile",
                   "Load an input file (Excel or.csv)",
                   multiple = FALSE),
         p("For file formatting tips
                                and examples, see Application Instructions
                                section, below.",
           class = "text-muted"),
         style = "padding-top: 0px;
                                padding-bottom: 0px;"

       ),

       wellPanel(
         ## Input: Select a statisical test
         div(h4("Analysis Options"),
             class = "text-primary"
         ),
```

```
        selectInput("test_type",
                    "Type of Statistical Test:",
                    choices = c("Unpaired T-test",
                                "Paired T-test",
                                "Chi-squared",
                                "One-way ANOVA",
                                "Two-way ANOVA"),
                    selected = "Unpaired T-test"),

        ## Input: Select an alpha level
        radioButtons("alpha",
                     "Alpha Level",
                     choices = c(0.05, 0.01),
                     selected = 0.01),

        ## Input: Select a power level
        radioButtons("power",
                     "Power Level",
                     choices = c(0.80, 0.90, 0.95),
                     selected = 0.90),
        style = "padding-top: 0px;
                              padding-bottom: 0px;"
      ))
```

UI Main Panel

Source = `UI/mainPanel-ui.R`

```
# Main panel for displaying outputs
column(8,
       conditionalPanel(condition = "output.sample_size",
                        wellPanel(
                            ## Output: Test type
                            tags$h3(textOutput("test_type"),
                                    class = "text-warning"),

                            ## Outout: Results table with
                            # sample size calculation
                            tableOutput("sample_size"),

                            ## Output: Notes re: power analysis
                            tags$h4("Analysis Notes",
                                    class = "text-primary"),
                            htmlOutput("test_notes"),

                            ## Output: Explaining the variables
                            tags$h4("Reported Variables",
                                    class = "text-primary"),
                            uiOutput("variables"),

                            ## Alternative Download
                            downloadButton(
                                    "generate_excel_report",
                                    "Create Excel Report",
```

```
                                    class = "btn btn-primary"
                            )
                        )


        ),
        ## Error Display
        conditionalPanel(condition = "output.error",
                         wellPanel(
                             # Output: display error codes
                             htmlOutput("error")))

)
```

UI Application Instructions

Source = `UI/appInstructions-ui.R`

```
column(12,
       wellPanel(
               h3("Application Instructions",
                  class = "text-primary"),
               p("To begin, load an Excel or a .csv file containing pilot data.
                           Then select the desired statistical
                           test using the pull down menu.
                           You may also select the desired
                           alpha and power levels.
                           The suggested default is an alpha
                           of 0.01 and a power of 0.9."),
               h4("Formatting your input files",
                  class = "text-primary"),
               p("Warning, the .csv format is often finicky. Make sure
                           that your file does not have extraneous
                           'empty' data columns."),
               h5("T-tests",
                  class = "text-warning"),
               p("Data should be separated into two
                 adjacent columns (e.g. control vs experimental)."),
               tags$ul(
                       tags$li("For unpaired tests,
                                       the n per condition does
                                       not have to be equal."),
                       tags$li("For paired tests, the number of rows
                                       must either match;
                                       uneven rows will be ignored.")
               ),
               h5("Chi-squared tests",
                  class="text-warning"),
               p("Data (proportions) should be
                               separated into two adjacent columns."),
               h5("One-way ANOVA",
                  class = "text-warning"),
               p("Data should be separated into at least 3
                           adjacent columns. The number of
```

```
                                rows does not have to be even
                                across all conditions."),
                    h5("Two-way ANOVA",
                       class = "text-warning"),
                    p("The power analysis for both main
                                effects and the interaction
                                effect will be calculated.
                                Data should be arranged in columns as follows:"),
                    tags$ul(
                            tags$li("Column 1: Factor A Condition 1"),
                            tags$li("Column 2: Factor A Condition 2"),
                            tags$li("Column 3: Factor B Condition 1"),
                            tags$li("Column 4: Factor B Condition 2")
                    ),
                    fluidRow(
                            column(6,
                                   div(selectInput("example_type",
                                                   "Download example .csv file:",
                                                   choices = c("Unpaired T-test",
                                                               "Paired T-test",
                                                               "Chi-squared",
                                                               "One-way ANOVA",
                                                               "Two-way ANOVA"),
                                                   selected = "Unpaired T-test"),
                                       class = "text-warning",
                                       style = "padding-bottom: 0px"),
                                   div(downloadLink('downloadData',
                                                    'Download'),
                                       style = "text-align: right;
                                                padding-top: 0px"))),
                    style = "padding-top: 0px;
                             padding-bottom: 5px;")
)
```

UI Static Application Notes

Source = `UI/appNotes-ui.R`

```
column(12,
       wellPanel(
                h4("About this application",
                   class = "text-primary"),
                p("This Shiny app calculates statistical power analyses
                                on user-provided data pilot data. The app takes a .csv
                                file with raw data as input, and calculates sample
                                sizes for specified power and alpha levels for several
                                common statistical tests.", class = "text-muted"),
                p("For all these tests, the assumption is made that the
                               data are pulled from a normal distribution,
                               i.e. that the statistical test used will be parametric.
                               Keep in mind that sample sizes provided may be an
                               underestimation in the case where the intention is
                               to use non-parametric statistical tests.
                               The Prism User Guide suggests that in the absence of
```

```
                              easy-to-apply mathematical tools for conducting power
                              analyses of non-parametric data, ",
                a(href = "https://www.graphpad.com/guides/prism/7/
                                  statistics/stat_sample_size_for_nonparametric_.htm",
                  "values can be estimated calculating the sample size
                                  for a parametric test and adding 15%",
                  .noWS = "outside"),".",
                .noWS = c("after-begin", "before-end"),
                class = "text-muted"),
              p("These calculations are primarily powered by the
                               WebPower Library. For background and/or additional
                               references relating to the WebPower library, ",
                a(href = "https://webpower.psychstat.org/wiki/",
                  "a manual and a wiki site are avaliable online",
                  .noWS = "outside"), '.',
                .noWS = c("after-begin", "before-end"),
                class = "text-muted"
              ),
              p("Created by Astra S, Bryant, PhD",
                class = "text-muted"),
              style = "padding-top: 0px;
                              padding-bottom: 0px;")
)
```

Define Server Logic / Run the App

The server logic section calls a separate R script containing the sample size calculator (see below), and provides text/table output to the UI. The final line runs the app.

```
server <- function (input, output){
        # Vals will contain all output variables
        vals <- reactiveValues(o = NULL, v = NULL, t = NULL,
                               n = NULL, dat = NULL, inputs = NULL)

        # Sample Size Calculations
        dataOutput <- reactive({
                req(input$loadfile) ## Don't run the code unless a file has been selected
                filename <-(input$loadfile$datapath)
                split_filename<-SplitPath(input$loadfile$name)

                ## Import Data, either an xls/xlsx or a csv file
                validate(need(split_filename$extension == "xls"||
                                      split_filename$extension == "xlsx" ||
                                      split_filename$extension == "csv",
                              message = "Please select an Excel or .csv file"))

                if (split_filename$extension == "xls" ||
                    split_filename$extension == "xlsx"){
                        dat <- read_excel(filename)
                } else if (split_filename$extension == "csv") {
                        dat <- read.csv(filename)}

                vals$inputs <- split_filename$fullfilename
                vals$dat <- dat
```

```
                result<-ComputeSampleSize(dat, input, vals)
        })

        # Parse outputs to Shiny UI
        source("Server/shinyOutputs-srv.R", local = TRUE)

        # Explanations of Reported Variables
        source("Server/variableCb-srv.R", local = TRUE)

        # Generate and Download report
        source("Server/excel-srv.R", local = TRUE)

        # Error Messages
        source("Server/error-srv.R", local = TRUE)

        # Generate example .csv files for download
        source("Server/exampleCSV-srv.R", local = TRUE)
}
shinyApp(ui = ui, server = server)
```

Parse Outputs to Shiny UI

Source = Server/shinyOutputs-srv.R

```
# Parsing results of sample size
# calculation for Shiny display
output$sample_size<- renderTable({
    result<-dataOutput()
    if (!is.null(result)){
        vals$o <- dplyr::select(result,
                                -c(note,method,url))
        vals$o}
})

outputOptions(output, 'sample_size', suspendWhenHidden = FALSE)

# Parsing statistical test type for Shiny display
output$test_type <- renderText({
    result<-dataOutput()
    if (!is.null(result)){
        vals$t <- dplyr::pull(result,method)%>%
            dplyr::first()
        vals$t
    }
})

# Parsing notes relevent to statistical test type
# for Shiny display
output$test_notes <- renderUI({
    result<-dataOutput()
    if (!is.null(result)){
        #str0 <- c('<h4 class = "text-primary">Notes</h4>')
        str0 <- dplyr::pull(result,note) %>%
            dplyr::first() %>%
```

```
            paste(". ", sep = "")

        str1 <- paste('This calculation assumes that data are ',
                      'pulled from a normal distribution.',
                       sep = "")
        str2 <- paste('If you plan to use a non-parametric test, ',
                    'add 15% to the calculated n.', sep = "")
        vals$n <-data.frame(notes = c(str0,str1,str2))
        apply(vals$n, 1, function(x) tags$p(x['notes']))


    }
})
```

Generate a "Codebook" of Returned Variables

Source = `Server/variableCb-srv.R`

```
# Variable Codebook
# Explanations of Reported Variables
output$variables <- renderUI({
        result<-dataOutput()
        if (!is.null(result)){
                method <- dplyr::pull(result,method)
                str0 <- c('n = sample size')
                str2 <- c('alpha = significance level (aka false positive rate)')
                str3 <- c('power = statistical power (aka 1 - false negative rate)')

                if (grepl('t-test', method[1])){
                        str1<- c('d = effect size (Cohens D)')
                        str4<- c('alternative = direction of the alternative hypothesis')
                        str5<- c(' ')
                        str6<- c(' ')

                } else if (grepl('proportion', method[1])){
                        str1<- c('h = effect size')
                        str4<- c(' ')
                        str5<- c(' ')
                        str6<- c(' ')

                } else if (grepl('One-way', method[1])){
                        str1<- c('f = effect size (f-ratio)')
                        str4<- c('k = number of groups')
                        str5<- c(' ')
                        str6<- c(' ')

                } else if (grepl('Two-way', method[1])){
                        str1<- c('f = effect size (f-ratio)')
                        str4<- c('ndf = numerator degrees of freedom')
                        str5<- c('ddf = denominator degrees of freedom')
                        str6<- c('ng = number of groups')
                }
                vals$v <-data.frame(variables = c(str0,str1,str2,str3,str4,str5,str6))
                apply(vals$v, 1, function(x) tags$p(x['variables']))
```

```
        }})
```

Generate Error Messages

Source = `Server/error-srv.R`

```
# Error Messages
output$error <- renderUI({
    result<-dataOutput()
    if (is.null(result)){
        str0 <-c('<h3 class = "text-danger">Warning</h3>')
        str1 <-c('<p>Number of data columns or rows does
                                not match the selected statistical test. </p>
                                <p>Please pick another file.</p>')
        str2 <-c('<p class = "text-muted">
                                Instructions for correct formating of .csv files
                                can be found under the Application Instructions
                                section below. </p>')

        HTML(paste(str0,str1, str2,sep = '<br/>'))}
})
outputOptions(output, 'error', suspendWhenHidden = FALSE)
```

Generate Example .csv Files

These can be downloaded from a handle in the static AppNotes section Source = `Server/exampleCSV-srv.R`

```
# Generate example .csv files for download
exampleOutput <- reactive({
    req(input$example_type) ## Don't run the code unless an output type
    if (grepl('T-test',input$example_type)){
        example_data <- cbind(Control = c(0.3, 0.2, 0.5),
                              Experimental = c(0.8, 0.7, 1.1))
    } else if (grepl('Chi',input$example_type)){
        example_data <- cbind(Control = 0.5, Experimental = 0.8)
    }else if (grepl('One-way',input$example_type)){
        example_data <- cbind(Control = c(0.3, 0.2, 0.5),
                              ExperimentalA = c(0.8, 0.7, 1.1),
                              ExperimentalB = c(0.5, 0.9, 1.3))
    }else if (grepl('Two-way',input$example_type)){
        example_data <- cbind(Control_ConditionA = c(0.3, 0.2, 0.5),
                              Control_ConditionB = c(0.8, 0.7, 1.1),
                              Experimental_ConditionA = c(0.1, 0.2, 1.1),
                              Experimental_ConditionB = c(0.5, 0.9, 1.3))
    }
})

output$downloadData<- downloadHandler(
    filename = function(){
        paste('example_',input$example_type, '.csv', sep='')
    },
    content = function(con) {
        example_data <- exampleOutput()
        write.csv(example_data,con)
```

```
    }
)
```

Sample Size Calculator

The sample size is calculated in a separate script `ComputeSampleSize.R`.

Inputs/Outputs

Input from the UI includes the raw data, and the user-selected statistical test type, alpha level, and power level.
Output to the UI includes the sample size for the desired statistical test, as well as the effect size, alpha level, power level, and notes about whether the given sample sizes refer to the number within a group.

Users may download an excel report containing the outputs and a copy of the inputs.

Types of Power Calculations

The app will calculate the required sample sizes for 5 different types of statistical tests:

- Unpaired T-test
- Paired T-test
- Chi-squared Test
- One-way ANOVA
- Two-way ANOA

For all these tests, the assumption is made that the data are pulled from a normal distribution, i.e. that the statistical test used will be parametric. User's should keep in mind that sample sizes provided may be an underestimation in the case where the intention is to use non-parametric statistical tests. The Prism User Guide suggests that in the absence of easy-to-apply mathematical tools for conducting power analyses of non-parametric data, values can be estimated calculating the sample size for a parametric test and adding 15%.

Unpaired T-test

```
        if (input$test_type == "Unpaired T-test"){
                if (ncol(dat) > 2){return(NULL)}
                if (nrow(dat) < 2){return(NULL)}
                dat_melt <- melt(dat,
                                 variable.name = 'condition',
                                 value.name = 'result',
                                 na.rm = TRUE)

                ES <- cohen.d(d = dat_melt$result, f = dat_melt$condition)

                Sample_size <- wp.t(d = ES$estimate,
                                    alpha = as.numeric(input$alpha),
                                    power = as.numeric(input$power),
                                    type = "two.sample")%>%
                        unclass() %>%
                        as_tibble
                #Sample_size$note <- paste("Note:", Sample_size$note)
                return(Sample_size)
```

Paired T-test

```
        } else if (input$test_type == "Paired T-test"){
                if (ncol(dat) > 2){return(NULL)}
                if (nrow(dat) < 2){return(NULL)}
                dat_complete <- dat[complete.cases(dat),]
                n <- nrow(dat_complete)
                dat_complete$id <- (1:n)

                dat_melt <- reshape2::melt(dat_complete,
                                measure.vars = 1:2,
                                variable.name = 'condition',
                                value.name = 'result',
                                na.rm = TRUE)


                ES <- cohen.d(d = dat_melt$result,
                              f = dat_melt$condition,
                              subject = dat_melt$id,
                              paired = TRUE)


                Sample_size <- wp.t(d = ES$estimate,
                                    alpha = as.numeric(input$alpha),
                                    power = as.numeric(input$power),
                                    type = "paired") %>%
                        unclass() %>%
                        as_tibble
                #Sample_size$note <- paste("Note:", Sample_size$note)
                return(Sample_size)
```

Chi-squared Test

```
        } else if (input$test_type == "Chi-squared"){
                if (ncol(dat) > 2){return(NULL)}
                if (nrow(dat) > 2){return(NULL)}
                dat_melt <- melt(dat,
                                variable.name = 'condition',
                                value.name = 'proportion')
                ES <- (2 * asin(sqrt(dat_melt$proportion[[1]]))) -
                        (2 * asin(sqrt(dat_melt$proportion[[2]])))
                Sample_size <- wp.prop(h = ES,
                                       alpha = 0.01, ##as.numeric(input$alpha),
                                       power = 0.9, ##as.numeric(input$power),
                                       type = "2p") %>%
                        unclass() %>%
                        as_tibble
               # Sample_size$note <- paste("Note:", Sample_size$note)
                Sample_size <- Sample_size %>%
                        select("n", "h", everything())
                return(Sample_size)
```

One-way ANOVA

```
        } else if (input$test_type == "One-way ANOVA"){
                if (ncol(dat) < 3){return(NULL)}
                k = ncol(dat) # Number of groups

                dat_melt <- melt(dat,
                                 variable.name = 'condition',
                                 value.name = 'result',
                                 na.rm = TRUE
                )

                anova1 <- aov(formula = dat_melt$result ~ dat_melt$condition)
                summary(anova1)
                etasquared <- EtaSq(anova1)
                CohensF <- sqrt(etasquared[2]/(1-etasquared[2]))
                Sample_size <- wp.anova(k = k,
                                        f = CohensF,
                                        alpha = as.numeric(input$alpha),
                                        power = as.numeric(input$power)) %>%
                        unclass() %>%
                        as_tibble

                Sample_size$n <- Sample_size$n/Sample_size$k
                Sample_size$note <- c("n is the sample size *in each group*")
                Sample_size$method <- c("One-way ANOVA")
                Sample_size <- Sample_size %>%
                        dplyr::select("n", "f", everything())
                return(Sample_size)
```

Two-way ANOVA

```
        } else if (input$test_type == "Two-way ANOVA"){
                if (ncol(dat) <4){return(NULL)}
                k <- ncol(dat) # Total number of groups
                r_num <- 2 ## Number of factors located in rows,
                ## for a 2 way anova this is hard-wired
                c_num <- k/r_num ## Number of factors located in columns

                ## Next couple of lines thanks of K. Zalocusky, PhD
                id_list <- c(1:r_num)
                id_list2 <- c(1:c_num)

                ids <- expand.grid(id_list, id_list2)
                ids <- ids[rep(seq_len(nrow(ids)), each = nrow(dat)), ]

                dat_melt <- melt(dat)
                dat_melt <- cbind(dat_melt, ids)
                dat_melt <- dat_melt[!is.na(dat_melt$value),] ## Get rid of NA values

                dat_melt$Var1 <- as_factor(dat_melt$Var1)
                dat_melt$Var2 <- as_factor(dat_melt$Var2)

                anova2 <- aov(formula = dat_melt$value ~ dat_melt$Var1 * dat_melt$Var2)
                etasquared <- EtaSq(anova2)
```

```
        CohensF <- sqrt(etasquared[,2]/(1-etasquared[,2]))
        ndf <- summary.aov(anova2) %>%
                flatten_df() %>%
                dplyr::select(Df) %>%
                pull()

        Sample_size_Interaction <- wp.kanova(f = CohensF[
                "dat_melt$Var1:dat_melt$Var2"],
                                             ndf = ndf[3],
                                             alpha = as.numeric(input$alpha),
                                             power = as.numeric(input$power),
                                             ng = 4) %>%
                unclass() %>%
                as_tibble %>%
                add_column(name = 'Interaction Effect', .before = "n")

        Sample_size_Var1 <- wp.kanova(f = CohensF["dat_melt$Var1"],
                                      ndf = ndf[1],
                                      alpha = as.numeric(input$alpha),
                                      power = as.numeric(input$power),
                                      ng = 4) %>%
                unclass() %>%
                as_tibble %>%
                add_column(name = 'Main Effect of Rows', .before = "n")

        Sample_size_Var2 <- wp.kanova(f = CohensF["dat_melt$Var2"],
                                      ndf = ndf[2],
                                      alpha = as.numeric(input$alpha),
                                      power = as.numeric(input$power),
                                      ng = 4) %>%
                unclass() %>%
                as_tibble() %>%
                add_column(name = 'Main Effect of Cols', .before = "n")

        Sample_size <- full_join(x = Sample_size_Var1,y = Sample_size_Var2) %>%
                full_join(y = Sample_size_Interaction)


        Sample_size$n <- Sample_size$n/Sample_size$ng
        Sample_size$note <- c("n is the sample size *in each group*")
        Sample_size$method <- c("Two-way ANOVA")
        Sample_size <- Sample_size %>%
                dplyr::select("name", "n", "f", everything())
        return(Sample_size)
}
```