For this post then, we’re going to explore comparing medians in a bayesian  
framework and along the way spend a minute or two on credible intervals.  
Read on! I’ll say it again below but comments and critique are always welcomed  
via disqus or email. You’ll have no trouble finding the icon links in a couple  
of places.

**To the data cave!**

A colleague of mine loves to call her office her *“data cave”*. I love the term  
and it reminds me of the old school Batman series on television, “To the Bat  
Cave, Robin”. So let’s go to our virtual data cave and get things setup.

library(tidyverse)

library(BayesFactor)

library(ggsignif)

library(scales)

theme\_set(theme\_bw()) # set theme

location <- "https://github.com/WinVector/PDSwR2/raw/master/PUMS/incomedata.rds"

incomedata <- readRDS(url(location))

incomedata$gp <- NULL

incomedata$employment <- fct\_recode(incomedata$employment,

"Self Not Inc" = "Self employed not incorporated",

"Self Incorporated" = "Self employed incorporated",

"Private for Profit" = "Employee of a private for profit",

"Private Non Profit" = "Private not-for-profit employee",

"Federal Government" = "Federal government employee",

"State Government" = "State government employee",

"Local Government" = "Local government employee"

)

incomedata$employment <- forcats::fct\_reorder(

.f = incomedata$employment,

.x = incomedata$income,

.fun = mean

)

# In case I want to reduce down to 3 more fundamental

# categories Private, Government, or self employed

incomedata$empcategory <- fct\_collapse(incomedata$employment,

"Self" = c("Self Not Inc", "Self Incorporated"),

"Private" = c("Private for Profit", "Private Non Profit"),

"Government" = c("Federal Government", "State Government", "Local Government")

)

str(incomedata)

## 'data.frame': 22241 obs. of 6 variables:

## $ income : num 22000 21000 25800 25000 20200 36000 20000 30000 23000 5000 ...

## $ age : num 24 31 26 27 27 47 24 41 43 21 ...

## $ sex : Factor w/ 2 levels "Male","Female": 1 2 2 2 2 1 1 1 2 2 ...

## $ employment : Factor w/ 7 levels "Self Not Inc",..: 2 4 2 2 4 2 2 2 2 2 ...

## $ education : Factor w/ 9 levels "no high school diploma",..: 1 4 4 6 4 2 1 1 6 2 ...

## $ empcategory: Factor w/ 3 levels "Self","Private",..: 2 2 2 2 2 2 2 2 2 2 ...

Okay we have the data we need. For those who cynically believe that visual  
inspection may be an aphorism for “draw any conclusion you like from the  
picture” let’s just quickly confirm we have a positive skew with psych::skew  
(rounded). In our case the answer is round(psych::skew(incomedata$income), 2)  
= 1.78. While we’re at it let’s confirm  
my allegation that skewness varies by group using psych::describeBy. I wasn’t  
making it up!

psych::describeBy(incomedata$income,

group = incomedata$employment,

mat = TRUE,

digits = 2) %>%

mutate(.data = .,

Employment\_Type = group1,

trimmed\_mean = trimmed) %>%

select(Employment\_Type,

mean,

sd,

median,

skew,

n,

trimmed\_mean)

## Employment\_Type mean sd median skew n trimmed\_mean

## 1 Self Not Inc 41382.50 34209.02 30000 1.73 829 35792.72

## 2 Private for Profit 51163.99 38744.57 40000 1.82 16170 44932.92

## 3 State Government 52612.39 29331.94 47150 1.64 1028 49269.63

## 4 Private Non Profit 52996.20 34389.69 45000 1.85 1573 48085.83

## 5 Local Government 53720.47 29438.20 48000 1.29 1492 50763.72

## 6 Federal Government 65311.26 34790.20 60000 0.73 580 62641.60

## 7 Self Incorporated 66906.44 51983.28 50000 1.52 569 58579.12

**We’re skewed! How do we fix it?**

There are any number of ways we can address the skew issue (I won’t try methods such  
as: hoping your reader/reviewer is inattentive or praying to your favorite  
deity). When I say any number I mean **a lot** of different ways we can address  
the issue. While non-exhaustive they generally lump into one of three  
categories. We can trim, we can transform, or we can use non-parametric methods  
(which is a bit disingenuous since some of the non-parametric methods themselves  
rely on transforming our data – we’ll get to that in a minute). Let’s briefly  
discuss.

1. Trim. We can “trim” the high and lows hoping this will eliminate any  
   troubling values at both ends. You may be surprised to know that capability  
   is built into mean() in base R, and mean(incomedata$income, , trim = .1)  
   will work. For those of you paying attention I grabbed that info when I ran  
   psych::describeBy. As you would expect for each of our groups it has  
   produced a trimmed mean which is closer to the median than our original mean. My  
   problem with it is threefold. I hate throwing away data. In this case we have  
   plenty but still. Second, there is no one right answer as to how much we  
   should trim? Why .1? Why not .05 or .2? It can make you suspicious that we  
   are simply picking an amount of trimming that best serves our agenda. Finally,  
   it implicitly makes the extreme values wrong or invalid or “not right”. In  
   this case there is little evidence that this is true. The income values don’t  
   look impossible or wrong, they just seem to reflect that there are  
   people who make much more and that zero is as low as it goes. Personally I  
   tend to only use trimming when I think the measurement is producing bad  
   values typically on both ends of the spectrum.
2. Transform. One of the common solutions to the income data problem is to  
   transform the variable. The transformation can take the form of a square  
   root, a log, or a more complex method such as Box-Cox or Yeo-Johnson. The  
   later is especially useful if your variable can validly take on negative  
   values (negative income anyone?). All of these methods are likely to “fix” our  
   positive skew problem since they will all serve to “pull” our data to the  
   left in a consistent way. All of them make it more difficult to describe our  
   results since for many readers the log of income is not a natural concept.  
   The other limitation here is that there is no one approved transform all will  
   serve to “improve” our skew but we’re subject to being questioned as to why  
   we selected one unless it is a disciplinary custom or for reasons of  
   comparison.
3. Non-parametric. For our particular case there are two non parametric methods  
   we would select. For the overall comparison of income across the 7 levels of  
   Employment it would be the Kruskal-Wallis. For pairwise comparisons as we did  
   last post it would be the Mann-Whitney U (a.k.a. Wilcoxon rank sum) test. If  
   you look back you’ll even notice that the Wilcoxon rank sum test  
   wilcox.test() is the default for ggsignif. As I mentioned earlier these  
   tests are actually a sort of transformation as well since they rely on the  
   ranks of our data rather than the data itself. I admit to having a bias  
   towards (maybe even a fondness for) non-parametric tests. While they aren’t  
   perfect and have limitations I take a certain amount of solace in knowing  
   that any supported findings were by using tests that at least eliminate  
   some number of assumptions.

So after last week’s posting I went sleuthing for the Bayesian equivalent to the  
Mann Whitney and Kruskal-Wallis tests. Yes I recognize that with powerful  
software like Stan you can pretty much construct any data analysis you want but  
I was hoping for something more “off the shelf” something very much like  
BayesFactor::ttestBF. No luck, I won’t claim I searched everywhere but I did  
search a lot for anything already done for R. Lot’s of theoretical papers and  
some questions but no solutions. If I missed it, and someone knows about it,  
please share.

Since I have been tracking the progress of JASP (unsolicited praise – it  
looks like a great piece of software especially if you come from the world of  
SPSS which I do). A lot of the calculation that is done under the very nice  
GUI is powered by R code and libraries. I noticed that they had recently  
implemented a Bayesian version of the Mann Whitney so I decided to give it a  
try. No joy! Using the very same data I have been using for these blog posts  
it ran for well over an hour and hadn’t completed. When I came back the next day  
it had completed and produced what appeared to be reasonable results on the  
surface but I had to do some sleuthing because the very long run time had me  
confused.

The underlying code is below:

|  |
| --- |
| TTestBayesianIndependentSamples <- function(dataset=NULL, options, perform="run", callback=function(...) 0, ...) { |
|  | # |
|  | dependents <- unlist(options$variables) |
|  | grouping <- options$groupingVariable |
|  | options[["wilcoxTest"]] <- options$testStatistic == "Wilcoxon" |
|  |  |
|  | if (grouping == "") |
|  | grouping <- NULL |
|  |  |
|  | if (is.null(dataset)) |
|  | { |
|  | if (perform == "run") { |
|  |  |
|  | if (options$missingValues == "excludeListwise") { |
|  |  |
|  | dataset <- .readDataSetToEnd(columns.as.numeric=dependents, columns.as.factor=grouping, exclude.na.listwise=c(dependents, grouping)) |
|  |  |
|  | } else { |
|  |  |
|  | dataset <- .readDataSetToEnd(columns.as.numeric=dependents, columns.as.factor=grouping, exclude.na.listwise=grouping) |
|  | } |
|  |  |
|  | } else { |
|  |  |
|  | dataset <- .readDataSetHeader(columns.as.numeric=dependents, columns.as.factor=grouping) |
|  | } |
|  | } |
|  |  |
|  | # error handling ##### |
|  | if (length(options$variables) != 0 && options$groupingVariable != '') { |
|  | errors <- .hasErrors(dataset, perform, type = c('factorLevels', 'variance'), |
|  | factorLevels.target = options$groupingVariable, factorLevels.amount = '!= 2', |
|  | variance.target = dependents, |
|  | variance.grouping = options$groupingVariable, |
|  | exitAnalysisIfErrors = TRUE) |
|  | } |
|  |  |
|  | results <- list() |
|  |  |
|  | meta <- list() |
|  |  |
|  | meta[[1]] <- list(name="ttest", type="table") |
|  | meta[[2]] <- list(name="descriptives", type="object", |
|  | meta=list(list(name="descriptivesTable", type="table"), |
|  | list(name = "descriptivesPlots", type = "collection", meta="image") |
|  | ) |
|  | ) |
|  |  |
|  | meta[[3]] <- list(name="inferentialPlots", type="collection", |
|  | meta=list(name="plotGroups", type="object", |
|  | meta=list(list(name="PriorPosteriorPlot", type="image"), |
|  | list(name="BFrobustnessPlot", type="image"), |
|  | list(name="BFsequentialPlot", type="image") |
|  | ) |
|  | ) |
|  | ) |
|  |  |
|  | results[[".meta"]] <- meta |
|  | results[["title"]] <- ifelse(options$wilcoxTest, "Bayesian Mann-Whitney U Test", "Bayesian Independent Samples T-Test") |
|  |  |
|  | state <- .retrieveState() |
|  | diff <- NULL |
|  |  |
|  | if (!is.null(state)) { |
|  | diff <- .diff(options, state$options) |
|  | } |
|  |  |
|  | # Needs to include following function for reactive progress bar for MCMC sampling |
|  | env <- environment() |
|  | callBackWilcoxonMCMC <- function(results = NULL, progress = NULL) { |
|  |  |
|  | response <- callback(results, progress) |
|  | if (response[["status"]] == "changed") { |
|  |  |
|  | optsForSampling <- c( |
|  | "variables", "groupingVariable", "wilcoxTest", "wilcoxonSamplesNumber", |
|  | "missingValues", "priorWidth" |
|  | ) |
|  |  |
|  | change <- .diff(env[["options"]], response[["options"]]) |
|  | env[["options"]] <- response[["options"]] |
|  |  |
|  | # if not any of the relevant options changed, status is ok |
|  | if (!any(unlist(change[optsForSampling]))) |
|  | response[["status"]] <- "ok" |
|  | } |
|  | return(response) |
|  |  |
|  | } |
|  |  |
|  | ttest.results <- .ttestBayesianIndependentSamplesTTest(dataset, options, perform, state=state, diff=diff, callBackWilcoxonMCMC) |
|  | # If relevant settings are changed during sampling, ttest.results will be NULL |
|  | # This triggers an automatic redo of the analysis |
|  | if(is.null(ttest.results)) return(list(results = NULL, status = "inited")) |
|  |  |
|  | results[["ttest"]] <- ttest.results[["ttest"]] |
|  | status <- ttest.results[["status"]] |
|  | g1 <- ttest.results[["g1"]] |
|  | g2 <- ttest.results[["g2"]] |
|  | BFH1H0 <- ttest.results[["BFH1H0"]] |
|  | plottingError <- ttest.results[["plottingError"]] |
|  | BF10post <- ttest.results[["BF10post"]] |
|  | errorFootnotes <- ttest.results[["errorFootnotes"]] |
|  | tValue <- ttest.results[["tValue"]] |
|  | n\_group2 <- ttest.results[["n\_group2"]] |
|  | n\_group1 <- ttest.results[["n\_group1"]] |
|  | deltaSamplesWilcox <- ttest.results[["delta"]] |
|  | rHat <- ttest.results[["rHat"]] |
|  |  |
|  |  |
|  | if(is.null(options()$BFMaxModels)) options(BFMaxModels = 50000) |
|  | if(is.null(options()$BFpretestIterations)) options(BFpretestIterations = 100) |
|  | if(is.null(options()$BFapproxOptimizer)) options(BFapproxOptimizer = "optim") |
|  | if(is.null(options()$BFapproxLimits)) options(BFapproxLimits = c(-15,15)) |
|  | if(is.null(options()$BFprogress)) options(BFprogress = interactive()) |
|  | if(is.null(options()$BFfactorsMax)) options(BFfactorsMax = 5) |
|  |  |
|  | descriptivesTable <- .ttestBayesianIndependentSamplesDescriptives(dataset, options, perform) |
|  | results[["descriptives"]] <- list(descriptivesTable = descriptivesTable, title = "Descriptives") |
|  |  |
|  | if (options$hypothesis == "groupOneGreater") { |
|  |  |
|  | oneSided <- "right" |
|  |  |
|  | } else if (options$hypothesis == "groupTwoGreater") { |
|  |  |
|  | oneSided <- "left" |
|  |  |
|  | } else { |
|  |  |
|  | oneSided <- FALSE |
|  | } |
|  |  |
|  | plotGroups <- list() |
|  | plots.ttest <- list() |
|  | descriptPlotVariables <- list() |
|  | descriptivesPlots <- list() |
|  | plotTypes <- list() |
|  | plotVariables <- list() |
|  |  |
|  |  |
|  | if (options$plotPriorAndPosterior || options$plotSequentialAnalysis || options$plotBayesFactorRobustness || options$descriptivesPlots) { |
|  |  |
|  | iint <- 1 |
|  | q <- 1 |
|  | descriptInd <- 1 |
|  |  |
|  | BFtype <- options$bayesFactorType |
|  | BFtypeRequiresNewPlot <- TRUE |
|  |  |
|  | if ( ! is.null(state)) { |
|  |  |
|  | BFtypeRequiresNewPlot <- FALSE |
|  | BFtypeState <- state$options$bayesFactorType |
|  |  |
|  | if ((BFtypeState == "LogBF10" || BFtypeState == "BF10") && BFtype == "BF01") { |
|  | BFtypeRequiresNewPlot <- TRUE |
|  | } else if (BFtypeState == "BF01" && (BFtype == "LogBF10" || BFtype == "BF10")) { |
|  | BFtypeRequiresNewPlot <- TRUE |
|  | } |
|  | } |
|  |  |
|  | for (variable in options[["variables"]]){ |
|  |  |
|  | plotGroups[[iint]] <- list() |
|  | plotGroups[[iint]][["title"]] <- variable |
|  | plotGroups[[iint]][["name"]] <- variable |
|  |  |
|  | if (options$descriptivesPlots) { |
|  |  |
|  | if (!is.null(state) && variable %in% state$descriptPlotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$groupingVariable == FALSE && |
|  | diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && diff$descriptivesPlotsCredibleInterval == FALSE))) && options$descriptivesPlots) { |
|  |  |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$descriptPlotVariables == variable) |
|  |  |
|  | descriptivesPlots[[descriptInd]] <- state$descriptivesPlots[[index]] |
|  |  |
|  |  |
|  | } else { |
|  | descriptivesPlot <- list() |
|  |  |
|  | descriptivesPlot[["title"]] <- variable |
|  | descriptivesPlot[["width"]] <- options$plotWidth |
|  | descriptivesPlot[["height"]] <- options$plotHeight |
|  | #descriptivesPlot[["custom"]] <- list(width="plotWidth", height="plotHeight") |
|  | descriptivesPlot[["status"]] <- "waiting" |
|  | descriptivesPlot[["data"]] <- "" |
|  |  |
|  | descriptivesPlots[[descriptInd]] <- descriptivesPlot |
|  | } |
|  |  |
|  |  |
|  | descriptPlotVariables[[length(descriptPlotVariables)+1]] <- variable |
|  |  |
|  | descriptInd <- descriptInd + 1 |
|  | } |
|  |  |
|  | if (options$plotPriorAndPosterior){ |
|  |  |
|  | if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && diff$effectSizeStandardized == FALSE && |
|  | diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && |
|  | diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && diff$wilcoxTest == FALSE && |
|  | diff$wilcoxonSamplesNumber == FALSE && options$plotPriorAndPosteriorAdditionalInfo && "posteriorPlotAddInfo" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "posteriorPlotAddInfo") |
|  |  |
|  | plots.ttest[[q]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && diff$effectSizeStandardized == FALSE && |
|  | diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && |
|  | diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && diff$wilcoxTest == FALSE && |
|  | diff$wilcoxonSamplesNumber == FALSE && !options$plotPriorAndPosteriorAdditionalInfo && "posteriorPlot" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # if the requested plot already exists use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "posteriorPlot") |
|  |  |
|  | plots.ttest[[q]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else { |
|  |  |
|  | plot <- list() |
|  |  |
|  | plot[["title"]] <- "Prior and Posterior" |
|  | plot[["width"]] <- 530 |
|  | plot[["height"]] <- 400 |
|  | plot[["status"]] <- "waiting" |
|  |  |
|  | .plotFunc <- function() { |
|  | .plotPosterior.summarystats.ttest(addInformation = options$plotPriorAndPosteriorAdditionalInf, dontPlotData = TRUE) |
|  | } |
|  | content <- .writeImage(width = 530, height = 400, plot = .plotFunc, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | plots.ttest[[q]] <- plot |
|  | } |
|  |  |
|  |  |
|  | if (options$plotPriorAndPosteriorAdditionalInfo) { |
|  |  |
|  | plotTypes[[length(plotTypes)+1]] <- "posteriorPlotAddInfo" |
|  |  |
|  | } else { |
|  |  |
|  | plotTypes[[length(plotTypes)+1]] <- "posteriorPlot" |
|  | } |
|  |  |
|  | plotGroups[[iint]][["PriorPosteriorPlot"]] <- plots.ttest[[q]] |
|  | plotVariables[[length(plotVariables)+1]] <- variable |
|  |  |
|  | q <- q + 1 |
|  |  |
|  | } |
|  |  |
|  | if (options$plotBayesFactorRobustness) { |
|  |  |
|  | if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | "robustnessPlotAddInfo" %in% state$plotTypes && options$plotBayesFactorRobustnessAdditionalInfo) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "robustnessPlotAddInfo") |
|  |  |
|  | plots.ttest[[length(plots.ttest)+1]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | "robustnessPlot" %in% state$plotTypes && !options$plotBayesFactorRobustnessAdditionalInfo) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "robustnessPlot") |
|  |  |
|  | plots.ttest[[q]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else { |
|  |  |
|  | plot <- list() |
|  |  |
|  | plot[["title"]] <- "Bayes Factor Robustness Check" |
|  | plot[["width"]] <- 530 |
|  | plot[["height"]] <- 400 |
|  | plot[["status"]] <- "waiting" |
|  |  |
|  | .plotFunc <- function() { |
|  | .plotBF.robustnessCheck.ttest (oneSided= oneSided, BFH1H0= BFH1H0, dontPlotData= TRUE, additionalInformation=options$plotBayesFactorRobustnessAdditionalInfo) |
|  | } |
|  | content <- .writeImage(width = 530, height = 400, plot = .plotFunc, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | plots.ttest[[q]] <- plot |
|  | } |
|  |  |
|  | if (options$plotBayesFactorRobustnessAdditionalInfo) { |
|  | plotTypes[[length(plotTypes)+1]] <- "robustnessPlotAddInfo" |
|  | } else { |
|  | plotTypes[[length(plotTypes)+1]] <- "robustnessPlot" |
|  | } |
|  |  |
|  | plotVariables[[length(plotVariables)+1]] <- variable |
|  | plotGroups[[iint]][["BFrobustnessPlot"]] <- plots.ttest[[q]] |
|  |  |
|  | q <- q + 1 |
|  | } |
|  |  |
|  | if (options$plotSequentialAnalysis) { |
|  |  |
|  | if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | options$plotSequentialAnalysisRobustness && "sequentialRobustnessPlot" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "sequentialRobustnessPlot") |
|  |  |
|  | plots.ttest[[q]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | !options$plotSequentialAnalysisRobustness && "sequentialPlot" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "sequentialPlot") |
|  |  |
|  | plots.ttest[[q]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else { |
|  |  |
|  | plot <- list() |
|  |  |
|  | plot[["title"]] <- "Sequential Analysis" |
|  | plot[["width"]] <- 530 |
|  | plot[["height"]] <- 400 |
|  | plot[["status"]] <- "waiting" |
|  |  |
|  | .plotFunc <- function() { |
|  | .plotSequentialBF.ttest(oneSided= oneSided, BFH1H0= BFH1H0, dontPlotData= TRUE, options = options) |
|  | } |
|  | content <- .writeImage(width = 530, height = 400, plot = .plotFunc, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | plots.ttest[[q]] <- plot |
|  | } |
|  |  |
|  |  |
|  | if (options$plotSequentialAnalysisRobustness) { |
|  |  |
|  | plotTypes[[length(plotTypes)+1]] <- "sequentialRobustnessPlot" |
|  |  |
|  | } else { |
|  |  |
|  | plotTypes[[length(plotTypes)+1]] <- "sequentialPlot" |
|  | } |
|  |  |
|  | plotVariables[[length(plotVariables)+1]] <- variable |
|  | plotGroups[[iint]][["BFsequentialPlot"]] <- plots.ttest[[q]] |
|  |  |
|  | q <- q + 1 |
|  |  |
|  | } |
|  |  |
|  | iint <- iint + 1 |
|  |  |
|  | } |
|  |  |
|  |  |
|  | if (options$plotPriorAndPosterior || options$plotBayesFactorRobustness || options$plotSequentialAnalysis) |
|  | results[["inferentialPlots"]] <- list(title=ifelse(length(options[["variables"]]) > 1 || sum(c(options$plotPriorAndPosterior, options$plotBayesFactorRobustness, options$plotSequentialAnalysis)) > 1, |
|  | "Inferential Plots", "Inferential Plot"), collection=plotGroups) |
|  |  |
|  | if (options$descriptivesPlots) |
|  | results[["descriptives"]][["descriptivesPlots"]] <- list(title=ifelse(length(options[["variables"]]) > 1, "Descriptives Plots", "Descriptives Plot"), collection=descriptivesPlots) |
|  |  |
|  |  |
|  | if (perform == "run" && length(options$variables) > 0 && !is.null(grouping)) { |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  |  |
|  | statusInd <- 1 |
|  | i <- 1 |
|  | z <- 1 |
|  | descriptInd <- 1 |
|  |  |
|  | for (variable in options[["variables"]]) { |
|  |  |
|  | errors <- .hasErrors(dataset, perform, message = 'short', type = c('infinity','observations','variance'), |
|  | all.target = variable, observations.amount = "< 2", all.grouping = options$groupingVariable) |
|  |  |
|  | if (!identical(errors, FALSE)) { |
|  | errorMessage <- errors$message |
|  | } else { |
|  | errorMessage <- NULL |
|  | } |
|  |  |
|  | subDataSet <- subset(dataset, select=c(.v(variable), .v(options$groupingVariable))) |
|  | subDataSet <- na.omit(subDataSet) |
|  |  |
|  | r.size <- options$priorWidth |
|  |  |
|  | group2 <- subDataSet[subDataSet[[.v(options$groupingVariable)]]== g1,.v(variable)] |
|  | group1 <- subDataSet[subDataSet[[.v(options$groupingVariable)]]== g2,.v(variable)] |
|  |  |
|  |  |
|  | if (options$descriptivesPlots) { |
|  |  |
|  |  |
|  | if (!is.null(state) && variable %in% state$descriptPlotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$groupingVariable == FALSE && |
|  | diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && diff$descriptivesPlotsCredibleInterval == FALSE))) && options$descriptivesPlots) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$descriptPlotVariables == variable) |
|  |  |
|  | descriptivesPlots[[descriptInd]] <- state$descriptivesPlots[[index]] |
|  |  |
|  |  |
|  | } else { |
|  |  |
|  | results[["descriptives"]][["descriptivesPlots"]][["collection"]][[i]][["status"]] <- "running" |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  |  |
|  | plot <- descriptivesPlots[[descriptInd]] |
|  |  |
|  | if (!is.null(errorMessage)) { |
|  |  |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  |  |
|  | } else { |
|  |  |
|  | p <- try(silent= FALSE, expr= { |
|  |  |
|  | p <- .plot2GroupMeansBayesIndTtest(v1 = group2, v2 = group1, nameV1 = g1, nameV2 = g2, |
|  | groupingName = options$groupingVariable, dependentName = variable, descriptivesPlotsCredibleInterval=options$descriptivesPlotsCredibleInterval) |
|  | content <- .writeImage(width = options$plotWidth, height = options$plotHeight, plot = p, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["editable"]] <- FALSE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | }) |
|  |  |
|  | } |
|  |  |
|  | plot[["status"]] <- "complete" |
|  |  |
|  | descriptivesPlots[[descriptInd]] <- plot |
|  | } |
|  |  |
|  | results[["descriptives"]][["descriptivesPlots"]][["collection"]] <- descriptivesPlots |
|  |  |
|  | descriptInd <- descriptInd + 1 |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  |  |
|  | } |
|  |  |
|  | if (options$plotPriorAndPosterior) { |
|  |  |
|  | if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && diff$effectSizeStandardized == FALSE && |
|  | diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && |
|  | diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && diff$wilcoxTest == FALSE && |
|  | diff$wilcoxonSamplesNumber == FALSE && options$plotPriorAndPosteriorAdditionalInfo && "posteriorPlotAddInfo" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "posteriorPlotAddInfo") |
|  |  |
|  | plots.ttest[[z]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && |
|  | diff$hypothesis == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && |
|  | diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && |
|  | diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && diff$wilcoxTest == FALSE && diff$wilcoxonSamplesNumber == FALSE && |
|  | !options$plotPriorAndPosteriorAdditionalInfo && "posteriorPlot" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # if the requested plot already exists use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "posteriorPlot") |
|  |  |
|  | plots.ttest[[z]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else { |
|  |  |
|  | results[["inferentialPlots"]][["collection"]][[i]][["PriorPosteriorPlot"]][["status"]] <- "running" |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  |  |
|  | plot <- plots.ttest[[z]] |
|  |  |
|  | if (!is.null(errorMessage)) { |
|  |  |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  |  |
|  | } else { |
|  |  |
|  | p <- try(silent= FALSE, expr= { |
|  |  |
|  | .plotFunc <- function() { |
|  | .plotPosterior.summarystats.ttest(t = tValue[i], n1 = n\_group2[i], n2 = n\_group1[i], |
|  | paired = FALSE, oneSided = oneSided, BF = BF10post[i], |
|  | BFH1H0 = BFH1H0, rscale = options$priorWidth, |
|  | addInformation = options$plotPriorAndPosteriorAdditionalInfo, |
|  | options = options, delta = deltaSamplesWilcox[[i]]) |
|  | } |
|  | content <- .writeImage(width = 530, height = 400, plot = .plotFunc, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | }) |
|  |  |
|  | if (isTryError(p)) { |
|  | errorMessage <- .extractErrorMessage(p) |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  | } |
|  |  |
|  | } |
|  |  |
|  | plot[["status"]] <- "complete" |
|  |  |
|  | plots.ttest[[z]] <- plot |
|  | } |
|  |  |
|  | plotGroups[[i]][["PriorPosteriorPlot"]] <- plots.ttest[[z]] |
|  |  |
|  |  |
|  | results[["inferentialPlots"]][["collection"]] <- plotGroups |
|  |  |
|  | z <- z + 1 |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  | } |
|  |  |
|  | if (options$plotBayesFactorRobustness) { |
|  | if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | "robustnessPlotAddInfo" %in% state$plotTypes && options$plotBayesFactorRobustnessAdditionalInfo) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "robustnessPlotAddInfo") |
|  |  |
|  | plots.ttest[[z]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | "robustnessPlot" %in% state$plotTypes && !options$plotBayesFactorRobustnessAdditionalInfo) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "robustnessPlot") |
|  |  |
|  | plots.ttest[[z]] <- state$plotsTtest[[index]] |
|  |  |
|  | } else { |
|  |  |
|  | results[["inferentialPlots"]][["collection"]][[i]][["BFrobustnessPlot"]][["status"]] <- "running" |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  |  |
|  | plot <- plots.ttest[[z]] |
|  |  |
|  | if (options$effectSizeStandardized == "informative") { |
|  | errorMessage="Bayes factor robustness check plot currently not supported for informed prior." |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  | } |
|  |  |
|  | if (!is.null(errorMessage)) { |
|  |  |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  |  |
|  | } else { |
|  |  |
|  | p <- try(silent= FALSE, expr= { |
|  |  |
|  | .plotFunc <- function() { |
|  | .plotBF.robustnessCheck.ttest(x= group2, y= group1, BF10post=ifelse((options$bayesFactorType=="LogBF10"), exp(BF10post[i]), BF10post[i]), paired= FALSE, oneSided= oneSided, |
|  | rscale = options$priorWidth, BFH1H0= BFH1H0, additionalInformation=options$plotBayesFactorRobustnessAdditionalInfo) |
|  | } |
|  | content <- .writeImage(width = 530, height = 400, plot = .plotFunc, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | }) |
|  |  |
|  | if (isTryError(p)) { |
|  | errorMessage <- .extractErrorMessage(p) |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  | } |
|  |  |
|  | } |
|  |  |
|  | plot[["status"]] <- "complete" |
|  |  |
|  | plots.ttest[[z]] <- plot |
|  | } |
|  |  |
|  | plotGroups[[i]][["BFrobustnessPlot"]] <- plots.ttest[[z]] |
|  | results[["inferentialPlots"]][["collection"]] <- plotGroups # add plots without image object to results |
|  |  |
|  | z <- z + 1 |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  | } |
|  |  |
|  | if (options$plotSequentialAnalysis) { |
|  |  |
|  | if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && diff$plotWidth == FALSE && |
|  | diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && |
|  | diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) && |
|  | options$plotSequentialAnalysisRobustness && "sequentialRobustnessPlot" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists, use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "sequentialRobustnessPlot") |
|  |  |
|  | plots.ttest[[z]] <- state$plotsTtest[[index]] |
|  | results[["inferentialPlots"]][["collection"]][[i]][["BFsequentialPlot"]][["status"]] <- "complete" |
|  |  |
|  | } else if (!is.null(state) && variable %in% state$plotVariables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && |
|  | diff$hypothesis == FALSE && BFtypeRequiresNewPlot == FALSE && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$plotHeight == FALSE && |
|  | diff$plotWidth == FALSE && diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && diff$informativeCauchyLocation == FALSE && |
|  | diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && |
|  | diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE))) |
|  | && !options$plotSequentialAnalysisRobustness && "sequentialPlot" %in% state$plotTypes) { |
|  |  |
|  | # if there is state and the variable has been plotted before and there is either no difference or only the variables or requested plot types have changed |
|  | # then, if the requested plot already exists use it |
|  |  |
|  | index <- which(state$plotVariables == variable & state$plotTypes == "sequentialPlot") |
|  |  |
|  | plots.ttest[[z]] <- state$plotsTtest[[index]] |
|  | results[["inferentialPlots"]][["collection"]][[i]][["BFsequentialPlot"]][["status"]] <- "complete" |
|  | } else { |
|  |  |
|  | results[["inferentialPlots"]][["collection"]][[i]][["BFsequentialPlot"]][["status"]] <- "running" |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  |  |
|  | plot <- plots.ttest[[z]] |
|  |  |
|  | if (options$plotSequentialAnalysisRobustness && options$effectSizeStandardized == "informative") { |
|  | plot[["error"]] <- list(error="badData", errorMessage="Sequential analysis robustness check plot currently not supported for informed prior.") |
|  | } |
|  |  |
|  | if (!is.null(errorMessage)) { |
|  |  |
|  | plot[["error"]] <- list(error="badData", errorMessage=errorMessage) |
|  |  |
|  | } else { |
|  |  |
|  | p <- try(silent= FALSE, expr= { |
|  |  |
|  | .plotFunc <- function() { |
|  | .plotSequentialBF.ttest(x= group2, y= group1, paired= FALSE, oneSided= oneSided, rscale = options$priorWidth, BFH1H0= BFH1H0, BF10post=BF10post[i], |
|  | plotDifferentPriors= options$plotSequentialAnalysisRobustness, subDataSet=subDataSet, level1=g1, level2=g2, options = options) |
|  | } |
|  | content <- .writeImage(width = 530, height = 400, plot = .plotFunc, obj = TRUE) |
|  |  |
|  | plot[["convertible"]] <- TRUE |
|  | plot[["obj"]] <- content[["obj"]] |
|  | plot[["data"]] <- content[["png"]] |
|  |  |
|  | }) |
|  |  |
|  | } |
|  |  |
|  | plot[["status"]] <- "complete" |
|  |  |
|  | plots.ttest[[z]] <- plot |
|  | } |
|  |  |
|  | plotGroups[[i]][["BFsequentialPlot"]] <- plots.ttest[[z]] |
|  | results[["inferentialPlots"]][["collection"]] <- plotGroups |
|  |  |
|  |  |
|  | z <- z + 1 |
|  |  |
|  | if ( ! .shouldContinue(callback(results))) |
|  | return() |
|  | } |
|  |  |
|  | statusInd <- statusInd + 1 |
|  | i <- i + 1 |
|  | } |
|  | } |
|  | } |
|  |  |
|  |  |
|  | keep <- NULL |
|  |  |
|  | for (plot in plots.ttest) |
|  | keep <- c(keep, plot$data) |
|  |  |
|  | for (plot in descriptivesPlots) |
|  | keep <- c(keep, plot$data) |
|  |  |
|  | descriptivesPlots <- descriptivesPlots |
|  | plots.ttest <- plots.ttest |
|  |  |
|  | if (perform == "init") { |
|  |  |
|  | return(list(results=results, status="inited", state=state, keep=keep)) |
|  |  |
|  | } else { |
|  |  |
|  | return(list(results=results, status="complete", state=list(options=options, results=results, plotsTtest=plots.ttest, plotTypes=plotTypes, plotVariables=plotVariables, |
|  | descriptPlotVariables=descriptPlotVariables, descriptivesPlots=descriptivesPlots, status=status, plottingError=plottingError, BF10post=BF10post, errorFootnotes=errorFootnotes, |
|  | tValue = tValue, n\_group2 = n\_group2, n\_group1 = n\_group1, delta = deltaSamplesWilcox, rHat = rHat), |
|  | keep=keep)) |
|  | } |
|  |  |
|  | } |
|  |  |
|  |  |
|  | .ttestBayesianIndependentSamplesTTest <- function(dataset, options, perform, state, diff, callBackWilcoxonMCMC) { |
|  |  |
|  |  |
|  | g1 <- NULL |
|  | g2 <- NULL |
|  |  |
|  | ttest <- list() |
|  |  |
|  | ttest[["title"]] <- ifelse(options$wilcoxTest, "Bayesian Mann-Whitney U Test", "Bayesian Independent Samples T-Test") |
|  |  |
|  | if (options$effectSizeStandardized == "default" & !options$wilcoxTest) { |
|  | ttest[["citation"]] <- list( |
|  | "Morey, R. D., & Rouder, J. N. (2015). BayesFactor (Version 0.9.11-3)[Computer software].", |
|  | "Rouder, J. N., Speckman, P. L., Sun, D., Morey, R. D., & Iverson, G. (2009). Bayesian t tests for accepting and rejecting the null hypothesis. Psychonomic Bulletin & Review, 16, 225–237.") |
|  | } else if (options$wilcoxTest) { |
|  | ttest[["citation"]] <- list( |
|  | "van Doorn, J., Ly, A., Marsman, M., & Wagenmakers, E. J. (2018). Bayesian Latent-Normal Inference for the Rank Sum Test, the Signed Rank Test, and Spearman's rho. Manuscript submitted for publication and uploaded to arXiv: https://arxiv.org/abs/1703.01805" |
|  | ) |
|  | } else if (options$effectSizeStandardized == "informative") { |
|  | ttest[["citation"]] <- list( |
|  | "Gronau, Q. F., Ly, A., & Wagenmakers, E.-J. (2017). Informed Bayesian T-Tests. Manuscript submitted for publication and uploaded to arXiv: https://arxiv.org/abs/1704.02479") |
|  | } |
|  |  |
|  | fields <- list( |
|  | list(name=".variable", title="", type="string", combine=TRUE)) |
|  |  |
|  | if (options$bayesFactorType == "BF01") { |
|  |  |
|  | BFH1H0 <- FALSE |
|  |  |
|  | if (options$hypothesis == "groupsNotEqual"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="BF\u2080\u2081") |
|  | } |
|  | if (options$hypothesis == "groupOneGreater"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="BF\u2080\u208A") |
|  | } |
|  | if (options$hypothesis == "groupTwoGreater"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="BF\u2080\u208B") |
|  | } |
|  |  |
|  | } else if (options$bayesFactorType == "BF10") { |
|  |  |
|  | BFH1H0 <- TRUE |
|  |  |
|  | if (options$hypothesis == "groupsNotEqual"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="BF\u2081\u2080") |
|  | } |
|  | if (options$hypothesis == "groupOneGreater"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="BF\u208A\u2080") |
|  | } |
|  | if (options$hypothesis == "groupTwoGreater"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="BF\u208B\u2080") |
|  | } |
|  |  |
|  | } else if (options$bayesFactorType == "LogBF10") { |
|  |  |
|  | BFH1H0 <- TRUE |
|  |  |
|  | if (options$hypothesis == "groupsNotEqual"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="Log(\u0042\u0046\u2081\u2080)") |
|  | } |
|  | if (options$hypothesis == "groupOneGreater"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="Log(\u0042\u0046\u208A\u2080)") |
|  | } |
|  | if (options$hypothesis == "groupTwoGreater"){ |
|  | fields[[length(fields)+1]] <- list(name="BF", type="number", format="sf:4;dp:3", title="Log(\u0042\u0046\u208B\u2080)") |
|  | } |
|  | } |
|  |  |
|  | if (options$hypothesis == "groupsNotEqual" & !options$wilcoxTest) { |
|  |  |
|  | fields[[length(fields)+1]] <- list(name="error", type="number", format="sf:4;dp:3", title="error %") |
|  |  |
|  | } else if(!options$wilcoxTest) { |
|  |  |
|  | fields[[length(fields)+1]] <- list(name="error", type="number", format="sf:4;dp:3;~", title="error %") |
|  |  |
|  | } else if (options$wilcoxTest) { |
|  | # display Wilcoxon statistic instead of error % |
|  | fields[[length(fields)+1]] <- list(name="error", type="number", format="sf:4;dp:3;", title="W") |
|  | fields[[length(fields)+1]] <- list(name="rHat", type="number", format="sf:4;dp:3", title="R^") |
|  | } |
|  |  |
|  | ttest[["schema"]] <- list(fields=fields) |
|  |  |
|  | footnotes <- .newFootnotes() |
|  |  |
|  | levels <- base::levels(dataset[[ .v(options$groupingVariable) ]]) |
|  |  |
|  | if (length(levels) != 2) { |
|  |  |
|  | g1 <- "1" |
|  | g2 <- "2" |
|  |  |
|  | } else { |
|  |  |
|  | g1 <- levels[1] |
|  | g2 <- levels[2] |
|  | } |
|  |  |
|  | if (options$hypothesis == "groupOneGreater") { |
|  |  |
|  | message <- paste("For all tests, the alternative hypothesis specifies that group <em>", g1, "</em> is greater than group <em>", g2, "</em>.", sep="") |
|  | .addFootnote(footnotes, symbol="<em>Note.</em>", text=message) |
|  |  |
|  | } else if (options$hypothesis == "groupTwoGreater") { |
|  |  |
|  | message <- paste("For all tests, the alternative hypothesis specifies that group <em>", g1, "</em> is less than group <em>", g2, "</em>.", sep="") |
|  | .addFootnote(footnotes, symbol="<em>Note.</em>", text=message) |
|  | } |
|  |  |
|  | if (options$wilcoxTest) { |
|  |  |
|  | message <- paste("Result based on data augmentation algorithm with 5 chains of ", options$wilcoxonSamplesNumber, " iterations.", sep="") |
|  | .addFootnote(footnotes, symbol="<em>Note.</em>", text=message) |
|  | } |
|  |  |
|  | ttest.rows <- list() |
|  |  |
|  | status <- rep("ok", length(options$variables)) |
|  | BF10post <- numeric(length(options$variables)) |
|  | tValue <- rep(NA, length(options[["variables"]])) |
|  | n\_group2 <- rep(NA, length(options[["variables"]])) |
|  | n\_group1 <- rep(NA, length(options[["variables"]])) |
|  | plottingError <- rep("error", length(options$variables)) |
|  | errorFootnotes <- rep("no", length(options$variables)) |
|  | delta <- list() |
|  | rHat <- list() |
|  |  |
|  |  |
|  | for (variable in options[["variables"]]) { |
|  |  |
|  | if (!is.null(state) && variable %in% state$options$variables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && |
|  | diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && |
|  | diff$informativeTDf == FALSE && diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE && diff$wilcoxTest == FALSE && diff$wilcoxonSamplesNumber == FALSE)))) { |
|  |  |
|  | index <- which(state$options$variables == variable) |
|  |  |
|  | if (state$errorFootnotes[index] == "no") { |
|  |  |
|  | ttest.rows[[length(ttest.rows)+1]] <- state$results$ttest$data[[index]] |
|  |  |
|  | if (! (is.logical(diff) && diff == FALSE) && diff$bayesFactorType) { |
|  |  |
|  | if (state$options$bayesFactorType == "BF10") { |
|  |  |
|  | if (options$bayesFactorType == "BF01") { |
|  | ttest.rows[[length(ttest.rows)]]$BF <- 1 / state$results$ttest$data[[index]]$BF |
|  | } else if (options$bayesFactorType == "LogBF10") { |
|  | ttest.rows[[length(ttest.rows)]]$BF <- log(state$results$ttest$data[[index]]$BF) |
|  | } |
|  |  |
|  | } else if (state$options$bayesFactorType == "BF01") { |
|  |  |
|  | if (options$bayesFactorType == "BF10") { |
|  | ttest.rows[[length(ttest.rows)]]$BF <- 1 / state$results$ttest$data[[index]]$BF |
|  | } else if (options$bayesFactorType == "LogBF10") { |
|  | ttest.rows[[length(ttest.rows)]]$BF <- log(1 / state$results$ttest$data[[index]]$BF) |
|  | } |
|  |  |
|  | } else if (state$options$bayesFactorType == "LogBF10") { |
|  |  |
|  | if (options$bayesFactorType == "BF10") { |
|  | ttest.rows[[length(ttest.rows)]]$BF <- exp(state$results$ttest$data[[index]]$BF) |
|  | } else if (options$bayesFactorType == "BF01") { |
|  | ttest.rows[[length(ttest.rows)]]$BF <- 1 / exp(state$results$ttest$data[[index]]$BF) |
|  | } |
|  | } |
|  | } |
|  |  |
|  | } else { |
|  |  |
|  | index2 <- .addFootnote(footnotes, state$errorFootnotes[index]) |
|  |  |
|  | ttest.rows[[length(ttest.rows)+1]] <- list(.variable=variable, BF=.clean(NaN), error="", .footnotes=list(BF=list(index2))) |
|  |  |
|  | } |
|  |  |
|  | } else { |
|  |  |
|  | ttest.rows[[length(ttest.rows)+1]] <- list(.variable=variable) |
|  | } |
|  | } |
|  |  |
|  | rowCompleted <- logical(length(ttest.rows)) |
|  |  |
|  | for (i in seq\_along(ttest.rows)) |
|  | rowCompleted[i] <- ifelse(length(ttest.rows[[i]]) > 1, TRUE, FALSE) |
|  |  |
|  | if (!is.null(state) && all(options[["variables"]] %in% state$options$variables) && options$groupingVariable == state$options$groupingVariable && all(rowCompleted)) |
|  | ttest[["status"]] <- "complete" |
|  |  |
|  | if (perform == "run" && length(options$variables) != 0 && options$groupingVariable != "") { |
|  |  |
|  | if (length(levels) != 2) { |
|  |  |
|  | ttest[["error"]] <- list(errorType="badData", errorMessage="The Grouping Variable must have 2 levels") |
|  |  |
|  | status <- rep("error", length(options$variables)) |
|  | plottingError <- rep("Plotting is not possible: The Grouping Variable must have 2 levels", length(options$variables)) |
|  |  |
|  | } else { |
|  |  |
|  | rowNo <- 1 |
|  |  |
|  | i <- 1 |
|  |  |
|  | if (options$wilcoxTest) { |
|  | progressbar <- .newProgressbar(ticks=length(options[["variables"]]) \* round(5 \* options$wilcoxonSamplesNumber / 1e2, 0), |
|  | callback = callBackWilcoxonMCMC, response=TRUE) |
|  | } |
|  |  |
|  | for (variable in options[["variables"]]) { |
|  |  |
|  | # BayesFactor package doesn't handle NAs, so it is necessary to exclude them |
|  |  |
|  | subDataSet <- subset(dataset, select=c(.v(variable), .v(options$groupingVariable))) |
|  | subDataSet <- na.omit(subDataSet) |
|  |  |
|  | gs <- base::levels(levels) |
|  |  |
|  | group2 <- subDataSet[subDataSet[[.v(options$groupingVariable)]]== g1,.v(variable)] |
|  | group1 <- subDataSet[subDataSet[[.v(options$groupingVariable)]]== g2,.v(variable)] |
|  |  |
|  | f <- as.formula(paste( .v(variable), "~", .v(options$groupingVariable))) |
|  | r.size <- options$priorWidth |
|  |  |
|  | if (options$hypothesis == "groupOneGreater") { |
|  |  |
|  | null.interval <- c(0, Inf) |
|  | oneSided <- "right" |
|  |  |
|  | } else if (options$hypothesis == "groupTwoGreater") { |
|  |  |
|  | null.interval <- c(-Inf, 0) |
|  | oneSided <- "left" |
|  |  |
|  | } else { |
|  |  |
|  | null.interval <- c(-Inf, Inf) |
|  | oneSided <- FALSE |
|  | } |
|  |  |
|  |  |
|  | if (!is.null(state) && variable %in% state$options$variables && !is.null(diff) && ((is.logical(diff) && diff == FALSE) || (is.list(diff) && (diff$priorWidth == FALSE && diff$hypothesis == FALSE |
|  | && diff$groupingVariable == FALSE && diff$missingValues == FALSE && diff$effectSizeStandardized == FALSE && diff$informativeStandardizedEffectSize == FALSE && |
|  | diff$informativeCauchyLocation == FALSE && diff$informativeCauchyScale == FALSE && diff$informativeTLocation == FALSE && diff$informativeTScale == FALSE && diff$informativeTDf == FALSE && |
|  | diff$informativeNormalMean == FALSE && diff$informativeNormalStd == FALSE && diff$wilcoxTest == FALSE && diff$wilcoxonSamplesNumber == FALSE)))) { |
|  |  |
|  | index <- which(state$options$variables == variable) |
|  |  |
|  | if (state$errorFootnotes[index] == "no") { |
|  |  |
|  | ttest.rows[[rowNo]] <- state$results$ttest$data[[index]] |
|  |  |
|  | if (! (is.logical(diff) && diff == FALSE) && diff$bayesFactorType) { |
|  |  |
|  | if (state$options$bayesFactorType == "BF10") { |
|  |  |
|  | if (options$bayesFactorType == "BF01") { |
|  | ttest.rows[[rowNo]]$BF <- 1 / state$results$ttest$data[[index]]$BF |
|  | } else if (options$bayesFactorType == "LogBF10") { |
|  | ttest.rows[[rowNo]]$BF <- log(state$results$ttest$data[[index]]$BF) |
|  | } |
|  |  |
|  | } else if (state$options$bayesFactorType == "BF01") { |
|  |  |
|  | if (options$bayesFactorType == "BF10") { |
|  | ttest.rows[[rowNo]]$BF <- 1 / state$results$ttest$data[[index]]$BF |
|  | } else if (options$bayesFactorType == "LogBF10") { |
|  | ttest.rows[[rowNo]]$BF <- log(1 / state$results$ttest$data[[index]]$BF) |
|  | } |
|  |  |
|  | } else if (state$options$bayesFactorType == "LogBF10") { |
|  |  |
|  | if (options$bayesFactorType == "BF10") { |
|  | ttest.rows[[rowNo]]$BF <- exp(state$results$ttest$data[[index]]$BF) |
|  | } else if (options$bayesFactorType == "BF01") { |
|  | ttest.rows[[rowNo]]$BF <- 1 / exp(state$results$ttest$data[[index]]$BF) |
|  | } |
|  | } |
|  | } |
|  |  |
|  | } else { |
|  |  |
|  | index2 <- .addFootnote(footnotes, state$errorFootnotes[index]) |
|  |  |
|  | errorFootnotes[rowNo] <- state$errorFootnotes[index] |
|  |  |
|  | ttest.rows[[rowNo]] <- list(.variable=variable, BF=.clean(NaN), error="", .footnotes=list(BF=list(index2))) |
|  |  |
|  | } |
|  |  |
|  | BF10post[rowNo] <- ttest.rows[[rowNo]]$BF # state$BF10post[index] |
|  | tValue[rowNo] <- state$tValue[index] |
|  | n\_group2[rowNo] <- state$n\_group2[index] |
|  | n\_group1[rowNo] <- state$n\_group1[index] |
|  | status[rowNo] <- state$status[index] |
|  | plottingError[rowNo] <- state$plottingError[index] |
|  | delta[[rowNo]] <- state$delta[[index]] |
|  | rHat[[rowNo]] <- state$rHat[[index]] |
|  |  |
|  | } else { |
|  |  |
|  | errors <- .hasErrors(dataset, perform, message = 'short', type = c('observations', 'variance', 'infinity'), |
|  | all.target = variable, |
|  | observations.amount = '< 2', |
|  | all.grouping = options$groupingVariable) |
|  |  |
|  | if (!identical(errors, FALSE)) { |
|  | errorMessage <- errors$message |
|  | } else { |
|  | errorMessage <- NULL |
|  | } |
|  |  |
|  | result <- try (silent=FALSE, expr= { |
|  |  |
|  | n\_group2[i] <- length(group2) |
|  | n\_group1[i] <- length(group1) |
|  |  |
|  | if (!options$wilcoxTest) { |
|  |  |
|  | r <- .generalTtestBF(x = group2, y = group1, paired = FALSE, oneSided = oneSided, options = options) |
|  | bf.raw <- r[["bf"]] |
|  | error <- .clean(r[["error"]]) |
|  | tValue[i] <- r[["tValue"]] |
|  | delta[[i]] <- NA |
|  | rHat[[i]] <- NA |
|  |  |
|  |  |
|  | } else if (options$wilcoxTest) { |
|  | # If the samples can be reused, don't call the Gibbs sampler again, but recalculate the |
|  | # Bayes factor with new settings and take the samples from state. |
|  | if (!is.null(diff) && length(state$delta) >= i && !is.null(state$delta[[i]]) && diff$hypothesis == TRUE && diff$priorWidth == FALSE && diff$groupingVariable == FALSE && |
|  | diff$missingValues == FALSE && diff$wilcoxonSamplesNumber == FALSE) { |
|  |  |
|  | delta[[i]] <- state$delta[[i]] |
|  | rHat[[i]] <- state$rHat[[i]] |
|  | bf.raw <- .computeBayesFactorWilcoxon(deltaSamples = delta[[i]], cauchyPriorParameter = options$priorWidth, oneSided = oneSided) |
|  |  |
|  | } else { |
|  |  |
|  | r <- .rankSumGibbsSampler(x = group2, y = group1, nSamples = options$wilcoxonSamplesNumber, nBurnin = 1, |
|  | cauchyPriorParameter = options$priorWidth, progressbar = progressbar) |
|  | if(is.null(r)) return() # Return null if settings are changed |
|  | delta[[i]] <- r[["deltaSamples"]] |
|  | rHat[[i]] <- r[["rHat"]] |
|  | bf.raw <- .computeBayesFactorWilcoxon(deltaSamples = r[["deltaSamples"]], cauchyPriorParameter = options$priorWidth, oneSided = oneSided) |
|  |  |
|  | } |
|  |  |
|  | wValue <- unname(wilcox.test(group2, group1, paired = FALSE)$statistic) |
|  | error <- wValue |
|  | tValue[i] <- median(delta[[i]]) |
|  |  |
|  | } |
|  |  |
|  | if (options$bayesFactorType == "BF01") |
|  | bf.raw <- 1 / bf.raw |
|  |  |
|  | BF10post[i] <- bf.raw |
|  | BF <- .clean(bf.raw) |
|  |  |
|  | if (options$bayesFactorType == "LogBF10") { |
|  |  |
|  | BF <- log(BF10post[i]) |
|  | BF <- .clean(BF) |
|  | } |
|  |  |
|  | if (is.na(bf.raw)) { |
|  |  |
|  | status[rowNo] <- "error" |
|  | plottingError[rowNo] <- "Plotting is not possible: Bayes factor could not be calculated" |
|  | } |
|  |  |
|  | if(is.infinite(bf.raw)){ |
|  |  |
|  | if(options$plotPriorAndPosterior | options$plotSequentialAnalysis | options$plotSequentialAnalysisRobustness | options$plotBayesFactorRobustness){ |
|  |  |
|  |  |
|  | status[rowNo] <- "error" |
|  | plottingError[rowNo] <- "Plotting is not possible: Bayes factor is infinite" |
|  | } |
|  | } |
|  |  |
|  | if(is.infinite(1/bf.raw)){ |
|  |  |
|  | if(options$plotPriorAndPosterior | options$plotSequentialAnalysis | options$plotSequentialAnalysisRobustness | options$plotBayesFactorRobustness){ |
|  |  |
|  | status[rowNo] <- "error" |
|  | plottingError[rowNo] <- "Plotting is not possible: The Bayes factor is too small" |
|  | } |
|  | } |
|  |  |
|  | ind <- which(group1 == group1[1]) |
|  | idData <- sum((ind+1)-(1:(length(ind))) == 1) |
|  |  |
|  | ind2 <- which(group2 == group2[1]) |
|  | idData2 <- sum((ind2+1)-(1:(length(ind2))) == 1) |
|  |  |
|  | if(idData > 1 && idData2 > 1 && (options$plotSequentialAnalysis | options$plotSequentialAnalysisRobustness)){ |
|  |  |
|  | if(options$plotPriorAndPosterior | options$plotSequentialAnalysis | options$plotSequentialAnalysisRobustness | options$plotBayesFactorRobustness){ |
|  |  |
|  | # errorMessage <- paste("Sequential Analysis not possible: The first observations are identical") |
|  | # index <- .addFootnote(footnotes, errorMessage) |
|  |  |
|  | # status[rowNo] <- "sequentialNotPossible" |
|  | # plottingError[rowNo] <- "Sequential Analysis not possible: The first observations are identical" |
|  | } |
|  | } |
|  |  |
|  | if(!is.null(errorMessage)){ |
|  |  |
|  | index <- .addFootnote(footnotes, errorMessage) |
|  | list(.variable=variable, BF=BF, error=error, .footnotes=list(BF=list(index))) |
|  | } else if (!is.null(rHat)) { |
|  |  |
|  | list(.variable=variable, BF=BF, error=error, rHat=rHat[[i]]) |
|  | } else { |
|  |  |
|  | list(.variable=variable, BF=BF, error=error) |
|  | } |
|  | }) |
|  |  |
|  | if(!is.null(errorMessage)){ |
|  |  |
|  | index <- .addFootnote(footnotes, errorMessage) |
|  |  |
|  | errorFootnotes[rowNo] <- errorMessage |
|  |  |
|  | result <- list(.variable=variable, BF=.clean(NaN), error="", .footnotes=list(BF=list(index))) |
|  |  |
|  | status[rowNo] <- "error" |
|  | } |
|  |  |
|  | ttest.rows[[rowNo]] <- result |
|  |  |
|  | } |
|  |  |
|  | rowNo <- rowNo + 1 |
|  | i <- i + 1 |
|  | } |
|  | } |
|  |  |
|  | ttest[["status"]] <- "complete" |
|  | } |
|  |  |
|  | ttest[["footnotes"]] <- as.list(footnotes) |
|  | ttest[["data"]] <- ttest.rows |
|  |  |
|  |  |
|  | list(ttest = ttest, status = status, g1 = g1, g2 = g2, BFH1H0 = BFH1H0, plottingError = plottingError, |
|  | BF10post = BF10post, errorFootnotes = errorFootnotes, tValue = tValue, n\_group2 = n\_group2, |
|  | n\_group1 = n\_group1, delta = delta, rHat = rHat) |
|  | } |
|  |  |
|  | .base\_breaks\_x <- function(x) { |
|  |  |
|  | b <- unique(as.numeric(x)) |
|  | d <- data.frame(y=-Inf, yend=-Inf, x=min(b), xend=max(b)) |
|  | list(ggplot2::geom\_segment(data=d, ggplot2::aes(x=x, y=y, xend=xend, yend=yend), inherit.aes=FALSE, size = 1)) |
|  |  |
|  | } |
|  |  |
|  | .base\_breaks\_y3 <- function(x) { |
|  |  |
|  | ci.pos <- c(x$ciLower, x$ciUpper) |
|  | b <- pretty(ci.pos) |
|  | d <- data.frame(x=-Inf, xend=-Inf, y=min(b), yend=max(b)) |
|  | list( ggplot2::geom\_segment(data=d, ggplot2::aes(x=x, y=y, xend=xend, yend=yend), inherit.aes=FALSE, size = 1), |
|  | ggplot2::scale\_y\_continuous(breaks=c(min(b),max(b))) ) |
|  | } |
|  |  |
|  | .plot2GroupMeansBayesIndTtest <- function(v1=NULL, v2=NULL, nameV1=NULL, nameV2=NULL, groupingName=NULL, dependentName=NULL, descriptivesPlotsCredibleInterval=.95) { |
|  |  |
|  | v1 <- na.omit(v1) |
|  | v2 <- na.omit(v2) |
|  |  |
|  | posteriorSummary1 <- .posteriorSummaryGroupMean(variable=v1, descriptivesPlotsCredibleInterval=descriptivesPlotsCredibleInterval) |
|  | posteriorSummary2 <- .posteriorSummaryGroupMean(variable=v2, descriptivesPlotsCredibleInterval=descriptivesPlotsCredibleInterval) |
|  | summaryStat <- data.frame( groupingVariable=c(nameV1, nameV2), dependent=c(posteriorSummary1$median, posteriorSummary2$median), |
|  | ciLower=c(posteriorSummary1$ciLower, posteriorSummary2$ciLower), ciUpper=c(posteriorSummary1$ciUpper, |
|  | posteriorSummary2$ciUpper), |
|  | group = 1) |
|  |  |
|  | pd <- ggplot2::position\_dodge(.2) |
|  |  |
|  | p <- ggplot2::ggplot(summaryStat, ggplot2::aes(x=groupingVariable, y=dependent, group=group)) + |
|  | ggplot2::geom\_errorbar(ggplot2::aes(ymin=ciLower, ymax=ciUpper), colour="black", width=.2, position=pd) + |
|  | ggplot2::geom\_line(position=pd, size = .7) + |
|  | ggplot2::geom\_point(position=pd, size=4) + |
|  | ggplot2::ylab(dependentName) + |
|  | ggplot2::xlab(groupingName) + |
|  | .base\_breaks\_y3(summaryStat) + |
|  | .base\_breaks\_x(summaryStat$groupingVariable) |
|  |  |
|  | p <- JASPgraphs::themeJasp(p) |
|  |  |
|  | return(p) |
|  | } |
|  |  |
|  | .ttestBayesianIndependentSamplesDescriptives <- function(dataset, options, perform, |
|  | state = NULL, diff = NULL) { |
|  | if (options$descriptives == FALSE) return(NULL) |
|  |  |
|  | descriptives = list("title" = "Group Descriptives") |
|  |  |
|  | ## sets up the table for the descriptives |
|  | fields <- list( |
|  | list(name = "variable", title = "", type = "string", combine = TRUE), |
|  | list(name = "group", title = "Group", type = "string"), |
|  | list(name = "N", title = "N", type = "number"), |
|  | list(name = "mean", title = "Mean", type = "number", format = "sf:4;dp:3"), |
|  | list(name = "sd", title = "SD", type = "number", format = "sf:4;dp:3"), |
|  | list(name = "se", title = "SE", type = "number", format = "sf:4;dp:3") |
|  | ) |
|  |  |
|  | ## add credible interval values if asked for in plot |
|  | if (options$descriptivesPlots) { |
|  | interval <- 100 \* options$descriptivesPlotsCredibleInterval |
|  | title <- paste0(interval, "% Credible Interval") |
|  | fields[[length(fields) + 1]] <- list(name = "lowerCI", type = "number", |
|  | format = "sf:4;dp:3", title = "Lower", |
|  | overTitle = title) |
|  | fields[[length(fields) + 1]] <- list(name = "upperCI", type = "number", |
|  | format = "sf:4;dp:3", title = "Upper", |
|  | overTitle = title) |
|  | } |
|  |  |
|  | descriptives[["schema"]] <- list(fields = fields) |
|  | data <- list() |
|  |  |
|  |  |
|  | ## function to check if everything is alright with the options |
|  | isAllright <- function(variable, options, state = NULL, diff = NULL) { |
|  |  |
|  | # check if the variable is in the state variables |
|  | cond1 <- !is.null(state) && variable %in% state$options$variables |
|  |  |
|  | # check if either diff is true, or it's a list and descriptives, |
|  | # and groupingVariable, missingValues are FALSE |
|  | cond2 <- (!is.null(diff) && (is.logical(diff) && diff == FALSE) || (is.list(diff) |
|  | && !any(diff$descriptives,diff$groupingVariable, diff$missingValues))) |
|  |  |
|  | cond1 && cond2 |
|  | } |
|  |  |
|  | variables <- options$variables |
|  | if (length(variables) == 0) variables <- "." |
|  |  |
|  | for (variable in variables) { |
|  |  |
|  | if (isAllright(variable, options, state, diff)) { |
|  |  |
|  | stateDat <- state$results$descriptives$data |
|  | descriptivesVariables <- as.character(length(stateDat)) |
|  |  |
|  | for (i in seq\_along(stateDat)) |
|  | descriptivesVariables[i] <- stateDat[[i]]$variable |
|  |  |
|  | indices <- which(descriptivesVariables == variable) |
|  | data[[length(data) + 1]] <- stateDat[[indices[1]]] |
|  | data[[length(data) + 1]] <- stateDat[[indices[2]]] |
|  |  |
|  | } else { |
|  | data[[length(data) + 1]] <- list(variable = variable, .isNewGroup = TRUE) |
|  | data[[length(data) + 1]] <- list(variable = variable) |
|  | } |
|  | } |
|  |  |
|  | ## check if we are done with all this crap |
|  | done <- (!is.null(state) && |
|  | state$options$descriptives && |
|  | all(variables %in% state$options$variables)) |
|  |  |
|  | if (done) descriptives[["status"]] <- "complete" |
|  |  |
|  | groups <- options$groupingVariable |
|  |  |
|  | ## if we actually have to do the test, and we have a grouping variable |
|  | if (perform == "run" && groups != "") { |
|  | levels <- base::levels(dataset[[ .v(groups) ]]) |
|  |  |
|  | ## if people don't know what a t-test is... |
|  | if (length(levels) != 2) { |
|  | descriptives[["error"]] <- list(errorType = "badData") |
|  |  |
|  | } else { |
|  |  |
|  | rowNo <- 1 |
|  | groupingData <- dataset[[.v(groups)]] |
|  |  |
|  | ## do the whole loop as above again |
|  | for (variable in variables) { |
|  |  |
|  | # if everything is alright, add stuff to data |
|  | if (isAllright(variable, options, state, diff)) { |
|  |  |
|  | stateDat <- state$results$descriptives$data |
|  | descriptivesVariables <- as.character(length(stateDat)) |
|  |  |
|  | for (i in seq\_along(stateDat)) |
|  | descriptivesVariables[i] <- stateDat[[i]]$variable |
|  |  |
|  | indices <- which(descriptivesVariables == variable) |
|  |  |
|  | data[[rowNo]] <- stateDat[[indices[1]]] |
|  | data[[rowNo]] <- stateDat[[indices[2]]] |
|  |  |
|  | rowNo <- rowNo + 2 |
|  |  |
|  | } else { |
|  |  |
|  | for (i in 1:2) { |
|  |  |
|  | level <- levels[i] |
|  | variableData <- dataset[[.v(variable)]] |
|  |  |
|  | groupData <- variableData[groupingData == level] |
|  | groupDataOm <- na.omit(groupData) |
|  |  |
|  | if (class(groupDataOm) != "factor") { |
|  |  |
|  | posteriorSummary <- .posteriorSummaryGroupMean(variable=groupDataOm, descriptivesPlotsCredibleInterval=options$descriptivesPlotsCredibleInterval) |
|  | ciLower <- .clean(posteriorSummary$ciLower) |
|  | ciUpper <- .clean(posteriorSummary$ciUpper) |
|  |  |
|  | n <- .clean(length(groupDataOm)) |
|  | mean <- .clean(mean(groupDataOm)) |
|  | std <- .clean(sd(groupDataOm)) |
|  | sem <- .clean(sd(groupDataOm) / sqrt(length(groupDataOm))) |
|  |  |
|  | result <- list(variable = variable, group = level, |
|  | N = n, mean = mean, sd = std, se = sem, lowerCI = ciLower, |
|  | upperCI = ciUpper) |
|  |  |
|  | } else { |
|  |  |
|  | n <- .clean(length(groupDataOm)) |
|  | result <- list(variable = variable, group = "", |
|  | N = n, mean = "", sd = "", se = "", lowerCI = "", |
|  | upperCI = "") |
|  | } |
|  |  |
|  | if (i == 1) { |
|  | result[[".isNewGroup"]] <- TRUE |
|  | } |
|  |  |
|  | data[[rowNo]] <- result |
|  | rowNo <- rowNo + 1 |
|  | } |
|  | } |
|  | } |
|  | } |
|  | descriptives[["status"]] <- "complete" |
|  | } |
|  |  |
|  | descriptives[["data"]] <- data |
|  | descriptives |
|  | } |
|  |  |
|  |  |
|  | .rankSumGibbsSampler <- function(xVals, yVals, nSamples = 1e3, cauchyPriorParameter = 1/sqrt(2), |
|  | nBurnin = 0, nGibbsIterations = 10, nChains = 5, progressbar){ |
|  | n1 <- length(xVals) |
|  | n2 <- length(yVals) |
|  |  |
|  | allRanks <- rank(c(xVals,yVals)) |
|  | xRanks <- allRanks[1:n1] |
|  | yRanks <- allRanks[(n1+1):(n1+n2)] |
|  |  |
|  | deltaSamples <- numeric(nSamples) |
|  | deltaSamplesMatrix <- matrix(ncol = nChains, nrow = nSamples-nBurnin) |
|  | totalIterCount <- 0 |
|  |  |
|  | for(thisChain in 1:nChains) { |
|  |  |
|  |  |
|  | currentVals <- sort(rnorm((n1+n2)))[allRanks] # initial values |
|  |  |
|  |  |
|  | oldDeltaProp <- 0 |
|  |  |
|  | for (j in 1:nSamples) { |
|  |  |
|  | totalIterCount <- totalIterCount + 1 |
|  |  |
|  | if (totalIterCount %% 1e2 == 0 ) { |
|  | response <- progressbar() |
|  | if (response[["status"]] != "ok") |
|  | return() |
|  | } |
|  |  |
|  | for (i in sample(1:(n1+n2))) { |
|  |  |
|  | currentRank <- allRanks[i] |
|  |  |
|  | currentBounds <- .upperLowerTruncation(ranks=allRanks, values=currentVals, currentRank=currentRank) |
|  | if (i <= n1) { |
|  | oldDeltaProp <- -0.5\*oldDeltaProp |
|  | } else if (i > n1) { |
|  | oldDeltaProp <- 0.5\*oldDeltaProp |
|  | } |
|  |  |
|  | currentVals[i] <- .truncNormSample(currentBounds[["under"]], currentBounds[["upper"]], mu=oldDeltaProp, sd=1) |
|  |  |
|  | } |
|  |  |
|  | xVals <- currentVals[1:n1] |
|  | yVals <- currentVals[(n1+1):(n1+n2)] |
|  |  |
|  | gibbsResult <- .sampleGibbsTwoSampleWilcoxon(x = xVals, y = yVals, n1 = n1, n2 = n2, nIter = nGibbsIterations, |
|  | rscale = cauchyPriorParameter) |
|  |  |
|  | deltaSamples[j] <- oldDeltaProp <- gibbsResult |
|  |  |
|  | } |
|  | deltaSamples <- -1 \* deltaSamples[-(1:nBurnin)] |
|  | deltaSamplesMatrix[, thisChain] <- deltaSamples |
|  | } |
|  |  |
|  | betweenChainVar <- (nSamples / (nChains - 1)) \* sum((apply(deltaSamplesMatrix, 2, mean) - mean(deltaSamplesMatrix))^2) |
|  | withinChainVar <- (1/ nChains) \* sum(apply(deltaSamplesMatrix, 2, var)) |
|  |  |
|  | fullVar <- ((nSamples - 1) / nSamples) \* withinChainVar + (betweenChainVar / nSamples) |
|  | rHat <- sqrt(fullVar/withinChainVar) |
|  |  |
|  | return(list(deltaSamples = as.vector(deltaSamplesMatrix), rHat = rHat)) |
|  | } |
|  |  |
|  | .sampleGibbsTwoSampleWilcoxon <- function(x, y, n1, n2, nIter = 10, rscale = 1/sqrt(2)) { |
|  | meanx <- mean(x) |
|  | meany <- mean(y) |
|  | n1 <- length(x) |
|  | n2 <- length(y) |
|  | sigmaSq <- 1 # Arbitrary number for sigma |
|  | g <- 1 |
|  | for(i in 1:nIter){ |
|  | #sample mu |
|  | varMu <- (4 \* g \* sigmaSq) / ( 4 + g \* (n1 + n2) ) |
|  | meanMu <- (2 \* g \* (n2 \* meany - n1 \* meanx)) / ((g \* (n1 + n2) + 4)) |
|  | mu <- rnorm(1, meanMu, sqrt(varMu)) |
|  | # sample g |
|  | betaG <- (mu^2 + sigmaSq \* rscale^2) / (2\*sigmaSq) |
|  | g <- 1/rgamma(1, 1, betaG) |
|  | # convert to delta |
|  | delta <- mu / sqrt(sigmaSq) |
|  | } |
|  | return(delta) |
|  | } |
|  |  |
|  | .truncNormSample <- function(lBound = -Inf, uBound = Inf, mu = 0, sd = 1) { |
|  |  |
|  | lBoundUni <- pnorm(lBound, mean = mu, sd = sd) |
|  | uBoundUni <- pnorm(uBound, mean = mu, sd = sd) |
|  | mySample <- qnorm(runif(1, lBoundUni, uBoundUni), mean = mu, sd = sd) |
|  |  |
|  | return(mySample) |
|  | } |
|  |  |
|  | .upperLowerTruncation <- function(ranks, values, currentRank, n, ranksAreIndices = FALSE) { |
|  |  |
|  | if (currentRank == min(ranks)) { |
|  | under <- -Inf |
|  | } else { |
|  | under <- max(values[ranks < currentRank]) |
|  | } |
|  |  |
|  | if (currentRank == max(ranks)) { |
|  | upper <- Inf |
|  | } else { |
|  | upper <- min(values[ranks > currentRank]) |
|  | } |
|  |  |
|  | return(list(under=under, upper=upper)) |
|  | } |
|  |  |
|  | .computeBayesFactorWilcoxon <- function(deltaSamples, cauchyPriorParameter, oneSided) { |
|  | postDens <- logspline::logspline(deltaSamples) |
|  | densZeroPoint <- logspline::dlogspline(0, postDens) |
|  | priorDensZeroPoint <- dcauchy(0, scale = cauchyPriorParameter) |
|  |  |
|  | corFactorPosterior <- ifelse(oneSided == "right" , 1 - logspline::plogspline(0, postDens), logspline::plogspline(0, postDens) ) |
|  | corFactorPrior <- pcauchy(0, scale = cauchyPriorParameter, lower.tail = (oneSided != "right" )) |
|  |  |
|  | bf <- ifelse(oneSided == FALSE, priorDensZeroPoint / densZeroPoint, |
|  | (priorDensZeroPoint / corFactorPrior) / (densZeroPoint / corFactorPosterior)) |
|  | return(bf) |
|  | } |

After a little bit of fiddling I was able to separate the JASP interface pieces  
from the R code that produces the results. I was thwarted again. Even on a much  
smaller dataset like ggplot2::mpg the results are slow and unstable. That’s not  
meant as a complaint I know they are working on it and I know working through  
the ties is challenging.

**A Eureka and a “YouRankA”**

Temporarily thwarted – something in the back of my mind made me remember dimly  
from the past considering the case of t tests on rankings. It had been a long  
time (nowadays it so trivial to run for example wilcox.test) and I wasn’t quite  
sure how comparable the results would be so I went looking

So if I can’t have a mannwhitneyBF maybe I can ttestBF(rank(income))? Seemed  
promising and the initial results seemed to be reasonable as well. The only  
challenge seemed to be that the distribution of rank(anything) tends to be  
anything but normal. The result tends to be very very flat with almost no tails  
at all. Almost a uniform distribution. Hmmmmm, which is a worse violation of  
t-test assumptions, skewed or flat?

“Take-home message: I am not saying that the rank-then-inverse-normal-transform  
strategy is always or even often a good idea. What I’m saying is that, if you  
were planning to do a rank transformation before analyzing your data, I  
recommend this z-score approach rather than the classical Wilcoxon method.” Gelman (July 2015)

That sounded promising. Was there an easy way for someone like me to implement  
his better than Wilcoxon suggestion (point taken that he repeatedly says “I’m  
not saying you should transform the data. I’m saying that if you were going to  
do Wilcoxon, then you’re already doing a rank transformation, in which case I  
think it makes sense to do it in an expandable way, rather than getting hung up  
on some pre-tabulated test statistics.”)?

Turns out there is. And it’s easy. And it’s already written and packaged in the  
RNOmni package on CRAN. The  
function is rankNorm and straight from the description what it does is:  
*“Applies the rank based inverse normal transform (INT) to a numeric vector. The  
INT can be broken down into a two-step procedure. In the first, the  
observations are transformed onto the probability scale using the empirical cumulative  
distribution function (ECDF). In the second, the observations are transformed  
onto the real line, as Z-scores, using the probit function.”*. The key component  
of the function is qnorm((r - k)/(n - 2 \* k + 1)) where r is the rank, n the  
number of observations, and k = 3/8 = the Blom transform.

I just realized I’ve been rambling for quite some time without a single graphic  
or plot. Perhaps this is a good time to just display the original income data  
and what the various transformations would do to them. I’ll plot them as density  
plots and superimpose the theoretical normal curve in a different color.

Let’s use cowplot and make 7 plots in order:

1. The original income data
2. Transform income to a z score (mean = 0, sd = 1) scale(income)
3. Transform sqrt(income)
4. Transform log(income)
5. Transform VGAM::yeo.johnson(income, lambda = .1)
6. Transform rank(income)
7. Transform RNOmni::rankNorm(income)

So as a first step we’ll use the mutate function to create these new variables. We’ll deliberately name them all so they end with “\_income”. In a minute we’ll use that little bit of consistency to make our lives easier.

incomedata <-

incomedata %>%

mutate(z\_income = as.numeric(scale(income)),

sqrt\_income = sqrt(income),

log\_income = log(income),

yj\_income = VGAM::yeo.johnson(income, lambda = .1),

rank\_income = rank(income),

rint\_income = RNOmni::rankNorm(income)

)

glimpse(incomedata)

## Observations: 22,241

## Variables: 12

## $ income 22000, 21000, 25800, 25000, 20200, 36000, 20000, 300…

## $ age 24, 31, 26, 27, 27, 47, 24, 41, 43, 21, 29, 30, 28, …

## $ sex Male, Female, Female, Female, Female, Male, Male, Ma…

## $ employment Private for Profit, Private Non Profit, Private for …

## $ education "no high school diploma", "some college credit, no d…

## $ empcategory Private, Private, Private, Private, Private, Private…

## $ z\_income -0.791115497, -0.817539654, -0.690703699, -0.7118430…

## $ sqrt\_income 148.32397, 144.91377, 160.62378, 158.11388, 142.1267…

## $ log\_income 9.998798, 9.952278, 10.158130, 10.126631, 9.913438, …

## $ yj\_income 17.17967, 17.05353, 17.61618, 17.52934, 16.94867, 18…

## $ rank\_income 3941.5, 3694.5, 5444.5, 5082.0, 3531.0, 9310.0, 3191…

## $ rint\_income -0.9260927, -0.9697185, -0.6910214, -0.7438702, -0.9…

A quick glimpse indicates we have what we need. So we have seven variables and  
we’re going to create 7 identical plots using ggplot. For each plot we’ll  
plot the density function for our observations (a more granular histogram so  
to speak). We’ll also plot the theoretical normal curve for that variable using  
dnorm() with the mean and standard deviation for the variable.

We could just write (mainly cut and paste) the call to ggplot 7 times. But  
that seems wasteful. Instead we’ll use purrr:pmap to do what we want with a  
lot less code. pmap wants a list that has as it’s components the 3 things that  
are going to vary for each of the seven iterations. The 3 things are the name of  
the variable, the mean, and the sd for the variable. Since ggplot can be  
fickle about passing it bare variable names we’ll pass it character strings for  
the name. The mean and the sd of course are numeric types. Again we could  
manually build those vectors but that would be tedious so let’s use some  
dplyr::select\_at and summarise\_at statements to get what we need. Since we  
were careful to name our variables so that they all end with “income” we can use  
vars(ends\_with("income")) to grab just what we need and not all the variables  
in incomedata.

a <- incomedata %>%

select\_at(vars(ends\_with("income"))) %>% names

b <- incomedata %>%

summarise\_at(vars(ends\_with("income")), mean, na.rm = TRUE)

c <- incomedata %>%

summarise\_at(vars(ends\_with("income")), sd, na.rm = TRUE)

plot\_list <- list(

which = a,

means = b,

stddevs = c

)

plot\_list

## $which

## [1] "income" "z\_income" "sqrt\_income" "log\_income" "yj\_income"

## [6] "rank\_income" "rint\_income"

##

## $means

## income z\_income sqrt\_income log\_income yj\_income rank\_income

## 1 51939.1 2.884567e-17 214.8975 10.61248 18.97702 11121

## rint\_income

## 1 -4.205901e-05

##

## $stddevs

## income z\_income sqrt\_income log\_income yj\_income rank\_income

## 1 37844.16 1 75.88434 0.7348988 2.09446 6419.503

## rint\_income

## 1 0.9991945

That took more text to describe than it took code to enact!

Now we invoke ggplot as a function .f. Inside the function call you’ll see **..1, ..2 & ..3** those mark the places where pmap will substitute in the appropriate value of plot\_list. The output of pmap is itself a list. A list of ggplot objects. We’ll take that list and give it to cowplot::plot\_grid along with some labeling and formatting and voila!

list\_of\_plots <- pmap(.l = plot\_list,

.f = ~ ggplot(data = incomedata,

aes\_string(x = ..1)

) +

geom\_density(alpha = .2) +

stat\_function(fun = dnorm,

color = "red",

args = list(mean = ..2,

sd = ..3)

)

)

cowplot::plot\_grid(plotlist = list\_of\_plots,

labels = c("Income (Original)",

"Z score Income",

"SQRT Income",

"Log Income",

"Yeo Johnson Income",

"Ranked Income",

"Rank Inv Norm Income"),

label\_size = 12,

vjust = 1,

hjust = 0,

scale = .9)

Not surprisingly the rank inverse normal transform works as advertised. We have  
succeeded in taking the ranks and backing them into a near perfect normal curve.  
That’s good but we did that work to prepare for comparing across employment  
groups so we would be wise to take a look at our transformed income variable  
rint\_income by group. Once again we’ll make use of psych::describeBy. This  
time we’ll also include the standard deviation since we’d like to have some  
sense of whether the variances are at least roughly equal among the groups. While  
we’re at it we’ll plot the density curves on a by group basis.

psych::describeBy(incomedata$rint\_income,

group = incomedata$employment,

mat = TRUE,

digits = 2) %>%

mutate(.data = .,

Employment\_Type = group1,

trimmed\_mean = trimmed) %>%

select(Employment\_Type,

mean,

sd,

median,

skew,

n,

trimmed\_mean)

## Employment\_Type mean sd median skew n trimmed\_mean

## 1 Self Not Inc -0.40 1.10 -0.49 0.04 829 -0.41

## 2 Private for Profit -0.04 1.02 -0.05 0.09 16170 -0.05

## 3 State Government 0.14 0.80 0.18 -0.29 1028 0.17

## 4 Private Non Profit 0.10 0.89 0.12 -0.06 1573 0.10

## 5 Local Government 0.16 0.83 0.20 -0.51 1492 0.20

## 6 Federal Government 0.43 0.88 0.54 -0.71 580 0.49

## 7 Self Incorporated 0.31 1.13 0.27 0.02 569 0.31

incomedata %>%

ggplot(aes(x = rint\_income,

fill = employment,

color = employment)

) +

geom\_density(alpha = .2)

Well, we certainly seem to have improved the distributions and made them more  
suitable for parametric manipulation. Certainly looks like there are shifts in  
central tendency by group. For the astute observer yes I’ll acknowledge that the  
variances among groups aren’t “equal”. But an **Fmax** ratio of  
2 isn’t that awful.

From last post let’s bring back our comparisons\_list function which will  
create a list of all the possible paired comparisons for us. We’ll store them in a  
list called comp.list.

comparisons\_list <- function(data,

x) {

data <-

dplyr::select(

.data = data,

x = !!rlang::enquo(x)

) %>%

dplyr::mutate(.data = .,

x = droplevels(as.factor(x)))

grouplevels <- levels(data$x)

g1\_list <- combn(grouplevels, 2)[1, ]

g2\_list <- combn(grouplevels, 2)[2, ]

comparisons\_list <- lapply(

1:length(g1\_list),

function(i) c(

combn(grouplevels, 2)[2, i],

combn(grouplevels, 2)[1, i]

)

)

return(comparisons\_list)

}

comp.list <- comparisons\_list(incomedata, employment)

**Welcome back my friends to the show that never ends (ELP)**

Last week we produced this plot using the original income data and using  
ggsignif. We ran the wilcox.test against all the pairings to produce p  
values on the plot. Here it is unretouched to refresh your memory.

If Gelman and Zimmerman have steered me correctly (or lol if I paid enough  
attention) we should be able to substitute in rint\_income our ranked inverse  
normalized transformed version of income and get very similar results using  
the very same code and wilcox.test.

ggplot(data = incomedata,

aes(

x = employment,

y = rint\_income,

fill = employment,

group = employment

)) +

geom\_boxplot(show.legend = FALSE) +

geom\_signif(

comparisons = comp.list,

step\_increase = .1

) +

scale\_y\_continuous(breaks = seq(from = -4,

to = 4,

by = 1)

) +

ggtitle(label = "ACS 2016 Rank Inverse Transformed Income by Employer Type",

subtitle = "Mann Whitney multiple comparisons non directional hypothesis using wilcox.test")

ggplot(data = incomedata,

aes(

x = employment,

y = rank\_income,

fill = employment,

group = employment

)) +

geom\_boxplot(show.legend = FALSE) +

geom\_signif(

comparisons = comp.list,

step\_increase = .1

) +

scale\_y\_continuous(breaks = seq(from = 0,

to = 25000,

by = 5000)

) +

ggtitle(label = "ACS 2016 Ranked Income by Employer Type",

subtitle = "Mann Whitney multiple comparisons non directional hypothesis using wilcox.test")

**Reminder** – Note that in the two-sample case the estimator for the difference in location parameters does not estimate the difference in medians (a common misconception) but rather the median of the difference between a sample from x and a sample from y.

Very similar indeed! Notice that the reported p values are essentially identical  
even as we can see from the box plots that the transformations are having an  
effect.

But of course this has all been prep work to this point. What I’m really after  
is to shift over to a bayesian framework and generate bayes factors that are the  
equivalent of the frequentist’s Mann Whitney. So let’s get on with that!

Not having been able to find the R code to do the work directly I’ve transformed  
the income variable and will now apply the ttestBF function. Here it is again.

pairwise\_bf <- function(x = NULL,

y = NULL,

display\_type = "bf",

k = 2) {

results <- ttestBF(x = x, y = y) %>%

extractBF() %>%

mutate(support = case\_when(

bf < .01 ~ "extreme BF01",

bf < .03 & bf >= .01 ~ "very strong BF01",

bf < .1 & bf >= .03 ~ "strong BF01",

bf < 1 / 3 & bf >= .1 ~ "moderate BF01",

bf < 1 & bf >= 1 / 3 ~ "anecdotal BF01",

bf >= 1 & bf < 3 ~ "anecdotal BF10",

bf >= 3 & bf < 10 ~ "moderate BF10",

bf >= 10 & bf < 30 ~ "strong BF10",

bf >= 30 & bf < 100 ~ "very strong BF10",

bf >= 100 ~ "extreme BF10"

)) %>%

mutate(logged = case\_when(

bf < 1 ~ paste("log(BF01) = ", round(log(1 / bf), k)),

bf >= 1 ~ paste("log(BF10) = ", round(log(bf), k))

)) %>%

mutate(human = case\_when(

bf < .000001 ~ "BF01 >= 1,000,000 : 1",

bf < .001 & bf >= .000001 ~ "BF01 >= 1,000 : 1",

bf < .01 & bf >= .001 ~ "BF01 >= 100 : 1",

bf < 1 & bf >= .01 ~ paste("BF01 = ", round(1 / bf, k), ": 1"),

bf >= 1 & bf < 100 ~ paste("BF01 = ", round(bf, k), ": 1"),

bf >= 100 & bf < 1000 ~ "BF10 >= 100 : 1",

bf >= 1000 & bf < 1000000 ~ "BF10 >= 1,000 : 1",

bf >= 1000000 ~ "BF10 >= 1,000,000 : 1"

))

if (display\_type == "support") {

results <- mutate(results, p.value = support)

} else if (display\_type == "log") {

results <- mutate(results, p.value = logged)

} else if (display\_type == "human") {

results <- mutate(results, p.value = human)

} else {

results <- mutate(results, p.value = bf)

}

return(results)

}

# pairwise\_bf(incomedata$employment, incomedata$rint\_income)

comp.list2 <- comparisons\_list(incomedata, empcategory)

theme\_set(hrbrthemes::theme\_ipsum())

Using the employment factor in the original dataset has been useful so far.  
Having 7 levels has allowed us to make sure that the functions supported lots of  
comparisons. The downside is the resulting plots have an overwhelming amount of  
information. So let’s shift over to the variable empcategory which collapses  
employment to just 3 levels. We’ll be comparing the self-employed, to those  
employed in the private sector, to those employed in local, state or federal  
jobs.

Seems reasonable to believe that the income levels across those broad  
categories, might be different. I don’t have any strong prior beliefs or  
information before this data. It’s not an area I study or have a lot of prior  
information amount. Clearly, it’s out there, the ACS survey has been running for  
years, but for now I’m perfectly happy to say my priors are flat.

What we’re going to do:

1. Show all the possible pairwise comparisons for empcategory as bayes factors in ggplot (we’ve already built the list above in comp.list2)
2. Change the theme just to make the plots nicer theme\_set(hrbrthemes::theme\_ipsum())
3. Apply the ttestBF test from the BayesFactor package to the data piped  
   into ggplot. We’ll compare income (the original untransformed data),  
   rint\_income which is our rank inverse transformed variant, and  
   rank\_income which is the income data with a simple rank transformation.  
   *N.B. – I’ve expressed the ranks so they are directionally equivalent to the  
   raw data so the highest recorded income in our data is $250,000 which  
   corresponds to rank of 22226.5 and after inverse transformation = 3.20299*
4. Display the **BF10** and/or **BF01** values that are between 1 and 100 as  
   is (rounded to 2 digits) and then create ranges between 100 to 1,000, 1,000 to one million and  
   greater than one million. At some point talking about odds over 100:1 (in my  
   humble opinion) loses the need for precision. After all is there really much  
   difference between odds of 1,000:1 versus 1,001:1?
5. Since this post is very much about mean() versus median() both of them  
   are plotted on top of the violin geom. The mean is displayed as a black box  
   and the median is displayed as a dark red circle.

**Caveat Emptor**

Before I draw any conclusions about the results a reminder. While the data are  
**“real”** drawn from the U.S. Census Bureau they most assuredly are not totally  
representative of the entire U.S. population! The very fact that the max value  
for income is $250,000 tells us that this isn’t everyone and I’m not sure at all  
how “representative” it is. Please don’t stray from my desire to investigate the  
statistical properties of the methods to think I’m trying to convince you of  
some conclusion about the income levels of Americans.

Conclusions we can draw.

1. Distributions matter! There is a difference between investigating the mean()  
   and the median() income and using bayesian methods doesn’t change that.
2. Bayes factors give us information in both directions! When you look at the  
   first plot using the mean for untransformed income values **BF01** = 29.36  
   tells you that the evidence is “strong” that the mean income for the  
   self-employed is the same as those in the private sector.
3. Whether we use simple rank transformation or reverse inverse transformed  
   ranks our overall conclusions are similar. We have weaker “anecdotal”  
   evidence that there is no difference between the self-employed and those in  
   the private sector. To say it differently we are less convinced from our data  
   that there is no difference. On the other hand we have beyond very strong  
   evidence that government employee income is different than either the self  
   employed or those in the private sector.
4. The difference in the bayes factors generated by using ranks versus rank  
   inverse transformation for our data were not **substantively** important in  
   magnitude. Yes they were different, for example 1.63 versus 2.48, but don’t  
   lead us to any practically different conclusions.
5. Not withstanding #4 above until there is a “true” equivalent to the Mann  
   Whitney for bayesian analysis I personally am far more comfortable applying a  
   “t-test” like analysis to the data distribution that results from RINT.

**Let’s get credible**

I know this post is already pretty long but I promised myself I’d capture what I  
learned about generating credible intervals as part of this workflow. I had been  
asked the question by a reader I did the work and posted it in a comment but now  
is my chance to capture it more formally.

In my usual fashion I’m not going to try and explain the concept of a credible  
interval in general when others have done so already and probably done a better  
job. So if you’re unfamiliar with the concept I recommend [a trip to  
Wikipedia](https://en.wikipedia.org/wiki/Credible_interval) to read up. A simple  
Google search will also yield lots of informative answers.

What I will focus on for the remainder of this post is simply helping you with  
the mechanics of constructing credible intervals if you have been following this  
series of posts.

So far we have been making heavy use of BayesFactor::ttestBF to accomplish our  
objectives. For example continuing with our last example our plot says the  
difference between the mean income for the privately employed and government  
workers is “BF10 >= 1,000:1”. We got that nice neat answer as a result of:

library(bayestestR)

tempdf <- incomedata %>%

filter(empcategory == "Private" | empcategory == "Government") %>%

droplevels %>%

as.data.frame

ttestBF(formula = income ~ empcategory, data = tempdf)

## Bayes factor analysis

## --------------

## [1] Alt., r=0.707 : 357043.1 ±0%

##

## Against denominator:

## Null, mu1-mu2 = 0

## ---

## Bayes factor type: BFindepSample, JZS

There it is – 357,043:1 to be more precise. And note the part that says  
**“Null, mu1-mu2 = 0”**. That looks familiar that’s a t test alright it is the  
difference between the means = 0. So let’s take the output from that ttestBF  
and but it in an object called bf.object. While that sounds wickedly complex  
the function we need is built right into BayesFactor as posterior so one  
line does the trick posterior.data <- posterior(bf.object, iterations = 10000). What’s in posterior.data? About 10,000 rows so let’s head it.

The first column mu is the overall mean, the second column beta (Private - Government) is exactly what we need to work on it’s the difference in means  
**“mu1-mu2”** between Private and Government. The sign tells us that at least  
for the first 7 rows Government is higher which agrees with our plot. I have no  
desire to scroll through 10,000 rows so let’s get some summary information and  
even a plot of our 10,000 samples.

bf.object <- ttestBF(formula = income ~ empcategory, data = tempdf)

posterior.data <- posterior(bf.object, iterations = 10000)

head(posterior.data)

## Markov Chain Monte Carlo (MCMC) output:

## Start = 1

## End = 7

## Thinning interval = 1

## mu beta (Private - Government) sig2 delta g

## [1,] 53712.39 -4138.397 1388783139 -0.11104907 0.3962511

## [2,] 53328.80 -3951.455 1372469541 -0.10666102 0.5661250

## [3,] 53798.28 -3568.424 1396930789 -0.09547487 0.1124928

## [4,] 53216.72 -3013.836 1401327895 -0.08051000 0.2242916

## [5,] 52885.07 -2276.827 1415166246 -0.06052381 0.4882304

## [6,] 52591.16 -3624.291 1381320679 -0.09751598 0.5180922

## [7,] 53115.20 -3875.964 1410243480 -0.10321260 2.2364977

summary(posterior.data)

##

## Iterations = 1:10000

## Thinning interval = 1

## Number of chains = 1

## Sample size per chain = 10000

##

## 1. Empirical mean and standard deviation for each variable,

## plus standard error of the mean:

##

## Mean SD Naive SE Time-series SE

## mu 5.343e+04 3.639e+02 3.639e+00 6.213e+00

## beta (Private - Government) -4.197e+03 7.303e+02 7.303e+00 1.269e+01

## sig2 1.396e+09 1.361e+07 1.361e+05 1.361e+05

## delta -1.124e-01 1.955e-02 1.955e-04 3.396e-04

## g 2.744e+00 4.663e+01 4.663e-01 4.663e-01

##

## 2. Quantiles for each variable:

##

## 2.5% 25% 50% 75%

## mu 5.271e+04 5.318e+04 5.343e+04 5.367e+04

## beta (Private - Government) -5.614e+03 -4.689e+03 -4.198e+03 -3.718e+03

## sig2 1.370e+09 1.386e+09 1.395e+09 1.405e+09

## delta -1.501e-01 -1.255e-01 -1.123e-01 -9.948e-02

## g 6.897e-02 1.877e-01 3.776e-01 9.061e-01

## 97.5%

## mu 5.413e+04

## beta (Private - Government) -2.745e+03

## sig2 1.422e+09

## delta -7.344e-02

## g 1.009e+01

plot(posterior.data[,"beta (Private - Government)"])

Whether we look at the right hand plot, or the tabular information Mean =  
-4.188e+03, 50% = -4.182e+03, sure looks like the difference estimate is about ~  
-$4,180. Also telling is 2.5% = -5.615e+03 and 97.5% = -2.775e+03 which means  
95% of our estimates were between -$5,615 and -$2,775! Anyone care to guess what  
the 95% credible interval is?

Well of course we don’t have to guess. We can take our posterior.data object and change it to a dataframe. After we’ve done that we can use bayestestR::ci (you may have noticed I loaded the library a few lines back). And there you have it the 89% credible interval is **[-5352.79, -3042.40]**. Wait. What? **89%**????

posterior.data <- as.data.frame(posterior.data)

str(posterior.data)

## 'data.frame': 10000 obs. of 5 variables:

## $ mu : num 53712 53329 53798 53217 52885 ...

## $ beta (Private - Government): num -4138 -3951 -3568 -3014 -2277 ...

## $ sig2 : num 1.39e+09 1.37e+09 1.40e+09 1.40e+09 1.42e+09 ...

## $ delta : num -0.111 -0.1067 -0.0955 -0.0805 -0.0605 ...

## $ g : num 0.396 0.566 0.112 0.224 0.488 ...

bayestestR::ci(posterior.data$`beta (Private - Government)`)

## # Credible Interval

##

## 89% CI

## [-5349.28, -3016.02]

.  
While you’re there good chance to review what a credible interval is, how it is  
interpreted, and our next topic which is that there are at least two equally  
valid ways of computing it. The first we used already it’s the **Equal-tailed  
Interval (ETI)** which is the default when you invoke ci. The other is Highest  
Density Interval (HDI) hdi().so I won’t repeat merely show you how.

ci() will accept a vector with multiple cut-offs so let’s do 89% and 95% once  
with equal-tailed and then with highest density. To be fair at least for the  
equal tailed you can use base R’s quantile function to get the same answers.  
BTW back a little bit when we were looking at summary(posterior.data) the  
answer for 95% was right there just hard to pick out.

ci(posterior.data$`beta (Private - Government)`,

ci = c(.89, .95))

## # Credible Intervals

##

## 89% CI

## [-5349.28, -3016.02]

##

##

## 95% CI

## [-5614.50, -2745.04]

hdi(posterior.data$`beta (Private - Government)`,

ci = c(.89, .95))

## # Highest Density Intervals

##

## 89% HDI

## [-5327.73, -2999.18]

##

##

## 95% HDI

## [-5625.08, -2759.35]

round(quantile(posterior.data$`beta (Private - Government)`,

probs = c(.025,

.055,

(1 - .055),

(1 - .025)

)

),

2)

## 2.5% 5.5% 94.5% 97.5%

## -5614.50 -5349.28 -3016.02 -2745.04

So given our observed data, the mean difference in income between the private sector respondents and the government employed respondents has a 95% probability of falling between -$5,614.99 and -$2,774.84.

We can preform the exact same operations on rint\_income if we want to convince  
ourselves that the mean difference (which for rint\_income is more akin to the  
median difference) is not equal to zero). The problem of course is that it  
doesn’t return a dollar value and our transformations are non-linear.

bf.object <- ttestBF(formula = rint\_income ~ empcategory, data = tempdf)

posterior.data <- posterior(bf.object, iterations = 10000)

summary(posterior.data)

##

## Iterations = 1:10000

## Thinning interval = 1

## Number of chains = 1

## Sample size per chain = 10000

##

## 1. Empirical mean and standard deviation for each variable,

## plus standard error of the mean:

##

## Mean SD Naive SE Time-series SE

## mu 0.08782 0.009480 9.480e-05 1.581e-04

## beta (Private - Government) -0.22912 0.019066 1.907e-04 3.165e-04

## sig2 0.96652 0.009474 9.474e-05 9.474e-05

## delta -0.23306 0.019439 1.944e-04 3.222e-04

## g 3.00664 61.759312 6.176e-01 6.176e-01

##

## 2. Quantiles for each variable:

##

## 2.5% 25% 50% 75% 97.5%

## mu 0.06913 0.08155 0.08785 0.09424 0.1062

## beta (Private - Government) -0.26596 -0.24200 -0.22917 -0.21659 -0.1917

## sig2 0.94811 0.96013 0.96647 0.97285 0.9851

## delta -0.27039 -0.24625 -0.23310 -0.22022 -0.1948

## g 0.07617 0.19808 0.40026 0.94421 9.5910

plot(posterior.data[,"beta (Private - Government)"])

posterior.data <- as.data.frame(posterior.data)

bayestestR::ci(posterior.data$`beta (Private - Government)`)

## # Credible Interval

##

## 89% CI

## [-0.26, -0.20]

ci(posterior.data$`beta (Private - Government)`,

ci = c(.89, .95))

## # Credible Intervals

##

## 89% CI

## [-0.26, -0.20]

##

##

## 95% CI

## [-0.27, -0.19]

hdi(posterior.data$`beta (Private - Government)`,

ci = c(.89, .95))

## # Highest Density Intervals

##

## 89% HDI

## [-0.26, -0.20]

##

##

## 95% HDI

## [-0.27, -0.19]

**A final look at why it matters**

So I wanted to end this post by reinforcing why mean() vs median() matters.  
Continuing to focus on the differences between income in the private sector and  
government employees remember we saw the BF associated with the difference  
between the two groups increase an order of magnitude when we shifted from  
assessing mean differences to median differences. Why?

Well take a look at the differences expressed in the original dollar units. The  
mean difference is about ~$4,200, the median difference is more than twice as  
much at ~$10,000, that’s a big difference.

psych::describeBy(incomedata$income,

group = incomedata$empcategory,

mat = TRUE,

digits = 2) %>%

mutate(.data = .,

Employment\_Type = group1,

trimmed\_mean = trimmed) %>%

select(Employment\_Type,

mean,

sd,

median,

skew,

n,

trimmed\_mean)

## Employment\_Type mean sd median skew n trimmed\_mean

## 1 Self 51771.00 44154.93 40000 1.82 1398 44342.71

## 2 Private 51326.42 38381.11 40000 1.82 17743 45221.13

## 3 Government 55521.62 30830.33 50000 1.28 3100 52263.65

a <- median(incomedata[incomedata$empcategory == "Private", "income"])

b <- median(incomedata[incomedata$empcategory == "Government", "income"])

c <- mean(incomedata[incomedata$empcategory == "Private", "income"])

d <- mean(incomedata[incomedata$empcategory == "Government", "income"])

median\_diff\_priv\_govt <- a - b

median\_diff\_priv\_govt

## [1] -10000

mean\_diff\_priv\_govt <- c - d

mean\_diff\_priv\_govt

## [1] -4195.196

For a final plot we’ll show it in the original units and then to make it easier  
to see give you a zoomed perspective with coord\_cartesian(). The dashed lines  
are the median values by group (which are color-coded) and the solid lines are  
the mean.

tempdf %>%

ggplot(aes(x = income,

fill = empcategory,

color = empcategory)

) +

geom\_density(alpha = .3) +

geom\_vline(xintercept = a, linetype = 5, color = "coral1") +

geom\_vline(xintercept = b, linetype = 5, color = "cyan3") +

geom\_vline(xintercept = c, color = "coral1") +

geom\_vline(xintercept = d, color = "cyan3") +

scale\_x\_continuous(label = dollar)

tempdf %>%

ggplot(aes(x = income,

fill = empcategory,

color = empcategory)

) +

geom\_density(alpha = .3) +

geom\_vline(xintercept = a, linetype = 5, color = "coral1") +

geom\_vline(xintercept = b, linetype = 5, color = "cyan3") +

geom\_vline(xintercept = c, color = "coral1") +

geom\_vline(xintercept = d, color = "cyan3") +

scale\_x\_continuous(label = dollar) +

coord\_cartesian(xlim = c(10000, 100000))

**Done**

Hope you enjoyed the post. Comments always welcomed.