7-8-2021

On Some Computer Packages For Combining P-Values

Breya Symone McGlown

ON SOME COMPUTER PACKAGES FOR COMBINING P-VALUES

by

Breya Symone McGlown

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Mathematical Sciences

The University of Memphis

August 2021

Acknowledgements

This Master's Thesis would not have been possible without the support and dedication to

Dr. E. Olusegun George.

Abstract

In scientific investigations, it is common to have several studies that deal with a common theme or hypothesis. In many such studies, independent test statistics or their corresponding p-values are usually available. Since the conclusions of such studies are not always conclusive, combining these independent results can reinforce the individual results and lead to a conclusive statement about the common hypothesis in question. In recent times, such meta-analytic procedure is usually required in such as genomics studies where detection of statistical significance based on one study is hard to obtain due to the inadequate sample size required because of the large number of genes that are usually interrogated. Simple computer tools for implementing these combination of P-values are not commonly available to non-statistically sophisticated user. In this thesis, we develop an easily deployed and feasible package that allows for the independent combination of p-values in two highly utilized programming languages. Users will be able to easily apply one of the affirmed methodologies. Packages were created in both R and Python, deployed to both main repositories (i.e., CRAN and PyPi), and tested against existing p-value data sets from other packages available for accuracy. With the need for this functionality to be readily available in two of the major programming languages used in both statistics and data science, the feasibility and utility of this package is apparent.

Table of Contents

List of Tables

Application of Combined P-Values

The need for combining p-values is well documented in the statistical literature and in multiple areas of statistical application (Fisher, R. A. 1934; Pearson, 1933; Tippett, L.H.C. 1931; Edgington, E. S. 1972; Mudholkar, G, & George, E. 1979; Stouffer et al., 1949; Birnbaum, 1954). Of these, meta-analyses prove to be one use case for the combination of independent p-values, as stated by several authors (Heard, N. A. & Rubin-Delanchy, P., 2018; Cheng,L. & Sheng, .X. S., 2017), among others such as large genomic experiments (Zaykin, D. et al. 2007).

Suppose *K* research centers conduct independent to resolve whether a hypothesis is true or not. With center *i* tests $H_{0i}\ vs. H_{1i}, i = 1, \dots, K$ and we let T_i be the test statistic for center $i, i = 1, \dots, K$. These test statistics may differ across research centers based on functional and distributional forms. For example, T_1 may be t-test, T_2 may be χ^2test, while T_3 may be a non-parametric Wilcoxon test. There is no simple algorithm for combining these tests for overall assessment of the hypothesis in question. However, if $P_1, \dots, P_K$ are the P-values of the $T_1, \dots T_K$, the p-values are conformable, since each is a random variable such that (a) $0 \leq P_i \leq 1, i = 1, \dots, K$ and (b)$P_i \sim U(0,1)$, $i = 1, \dots, K$ if under H_{01}if $T_i s$ are continuous. Given this fact, we can combine P-values since they are of the same distribution structure.

Let, $\Phi\ (P_1, \dots, P_K)$ be a real-valued random variable that is used to combine the results of the individual test for the purpose of testing H_0: All H_{01},…,H_{0k} are true versus H_1: At least one H_{1i} is true. We represent $H_0$and H_1 as $H_0 = \bigcap_{j=1}^{K} H_{0j}\ H_1 = \bigcup_{j=1}^{K} H_{1j}$

Given that datasets may not be fully available, as is the case with many published articles, or methodologies may differ across test statistics reported the test statistics cannot be combined directly (Loughin, 2004), tests based on combined P-values may be used to decide on the assertion made by H_1 in comparison to H_0. In addition, practical applications, combining p-

values gives the statistician the flexibility to weigh the individual statistics according to how informative they are and allows the designs of complex experiments to be determined independent of each other (Won, S. et al. 2009).

Previous work has shown the importance of combining p-values (Fisher, R. A. 1934; Pearson, 1933; Tippett, L.H.C. 1931; Edgington, E. S. 1972; George (1977); George & Mudholkar, 1978; Mudholkar, G, & George, E. 1979; Stouffer et al., 1949; Birnbaum, 1954; Heard, N. A. & Rubin-Delanchy, P., 2018), with six fundamental statistics used for combining p-values: $S_F = -2\sum_{i=1}^{K}\log p_i$ (Fisher, 1934), $S_P = -2\sum_{i=1}^{K}\log(1-p_i)$ (Pearson, 1933), $S_G = S_F - S_P = -\sum_{i=1}^{K}\log\{(p_{i)}/(1-p_i)\}$ (George, E. 1977), $S_s = \sum_{i=1}^{K}\phi^{-1}(p_i)$, where ϕ is the standard normal cumulative distribution function (Stouffer, et. al. 1949), $S_E = \sum_{i=1}^{K} p_i$ (Edgington, 1972), and $S_T = \min(p_1, \dots, p_K)$ (Tippett, L. H. C. 1931). Given *K* individual hypotheses $H_{0i}\ i = 1,2, \dots, K$ consider a test for the joint null hypothesis. $H_0$is true if all the individual null hypotheses are true, but false if at least one of its components is false.

In order to evaluate the efficiency of a combination method, Bahadur (1976) recommended the use of the "exact slope." The exact slope of a test procedure is described as the rate at which the p-value of the combined test converges to zero as the sample size increases to infinity, often the null hypothesis is false.

Heard et. al, (2018) provided a comprehensive derivation of alternative hypotheses under which the six procedures are most powerful, and therefore proved that that are all "admissible." Although, Fisher and George's method have optimal Bahadur efficiency (George, 1977; Litell Folks, 1973). With the six procedures being documented—proven as admissible in the testing of independent tests-- there have been no simple methods for researchers, academics, and industry-based statisticians alike to access these methodologies for ease of use., when all sample sizes are

small. One R open-source solution of combination of p-values does currently exist, created by William Poole et al. (2016) include :https://github.com/IlyaLab/CombiningDependentPvaluesUsingEBM , and a Python open-source solution located in the scipy.stats module imported as combine_pvalues: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.combine_pvalues.html. Some of the short-comings of these two approaches are (1) the R method implemented is Brown's Method only, which is not included in the six statistical tests mentioned above, and (2) while there are three programming languages available for Combining Dependent P-values Using EBM (i.e., Python, R, and Matlab) the utility of the programs available in the code repository is not user-friendly and they are not deployed as packages that can be easily imported, (3) The Python method does not include Edgington's affirmed method and (4) several of the combining p-value give inconsistent results, as will be described in further detail in the Methods and Statistical Analysis section. All test statistics and their respective distributions in the developed packages in this thesis follow Heard & Delanchy, 2018. A next best package is the Metap deployed in R. This package developed by Michael Dewey (2020), contains several statistics such as Edgington, Fisher, Lancaster, Stouffer, Tippett, and Wilkinson. Deployed only in R and excluding George's method, this particular package although more comprehensive than William Poole's is not available in Python.

Thus, the purpose of this project is to create an open-source package does not currently exist that allows for the combination of p-values or significance testing based on the six fundamental combination statistics. The current paper will focus on the development of the open-source packages that were created for these six statistics, significance testing based on the distribution these statistics follow, and applications of these packages.

Methods

Requirement Specifications

Purpose

The purpose of these software packages is to provide end users with the ability to input a set number of independent p-values, select one of the aforementioned fundamental methods as a means of combining those p-values, return a test statistic as output based on the combined p-values, and return a combined p-value based on the test statistic received as output.

Intended Audience

The intended audience for these software packages is any statistician or data scientist in industry or academia that requires the need for combining independent p-values.

Intended Use

The software was developed for the purposes of combining independent p-values using a method mentioned in the Introduction section. It solves several problems such as (1) filling the existing gap in open-source software that does not cover all six methods and (2) having the software compiled and ready to install on users' systems regardless of hardware.

Overall Description

Two software packages were developed and deployed in R and Python programming languages for the purpose of allowing end users the functionality of combining independent p-values. Users have the option of selecting the package in their preferred programming language. In each package the user will find six functions developed based on the six fundamental statistics (Fisher, R. A. 1934; Pearson, 1933; Tippett, L.H.C. 1931; Edgington, E. S. 1972; George, E.

1977; Stouffer et al., 1949) and one method that returns the combined p-value based on the function selected.

User Needs

User needs shall be defined on the basis of the following rules. In particular, it will be listed in a row of table as presented in **Error! Reference source not found.**.

Table 1.

User Need
Users have the need to use a software package solution based on their preferences
Users have the need to select one out of six fundamental test statistics functions developed
Users have the need to input K number of p-values into one of the six functions, where $K \geq 2$
Users have the need to obtain the combined p-value based on the test statistic used as input

Note. User Needs were gathered and defined by reviewing existing code base of open-source solutions.

Assumptions and Dependencies

A major assumption that should be taken into account is that independent p-values are used as input to all test statistic functions and methods developed. The issue of correlated p-values was not taken into account when developing these functions or packages. Please see the results section for recommendations of mitigating this issue.

System Features and Requirements

Functional Requirements

The following are functional requirements of each open-source solution developed:

1. Six test statistic functions are developed that follow the methodology detailed out in Heard and Rubin-Delanchy, 2018.

2. One additional function developed to return the combined p-values based on the test statistics' distribution, also described in Heard and Rubin-Delanchy, 2018.
3. End-users have an easy way to install and import packages into their workspace regardless of hardware.
4. Results must return in the end-users' IDE or cmd.

System Features

The following system features are required for the successful use of the software packages:

1. Access to internet
2. IDE such as Visual Studio Code or R Studio
3. Latest Python and R interpreters such as Python 3.7.4 and R 3.5.2

Software Package Development Details

Python 3.7.4 and R 3.5.2 versions were used for the creation of both packages. The following repository contains the necessary code base for both programming languages, along with a general README and program specific README.md files for both languages: https://github.com/StatsGirl/Master2021. Each package was written following coding standards specified by PEP8 and R Style Guide. Each package was developed using Visual Studio Code Python and R designated environments. The following packages are listed as dependencies for the creation of both packages:

1. Python: numpy, copy, scipy.stats, and math, along with standard built-in Python packages
2. R: chi, utils, dplyr, qnorm, stats, dbeta, dchisq, and dnorm, along with standard built in R libraries

Development and Deployment Process

The following details describe the code development, rebase/commits, and deployment of each package to their respective package repositories (i.e., CRAN and Pypi).

Python code development

The initial code developed involved identifying both the current state and desired outcome of the solution. The purpose of outlining the current state and desired outcome was to specify the input, processes (i.e., functions), and output required. The program was written in Python 3.7.4 due to this being a stable release of Python with several bug fixes such as improved DeprecationWarning handling, Context Variables, avoiding ASCII as default encoding, customization of access to module attributes, core support for typing module and generic types, and data classes. Object oriented programming was used to create the class, variables, and methods contained in this package. There is one class-- CountPs, along with the self-instantiated variable called method, which is used as an argument for each method in the class. Finally, there are eight methods: SumOfPs, FisherMethod, PearsonMethod, GeorgeMethod, EdMethod, StoufferMethod, TippettMethod, and CombinedPvalue. SumOfPs method takes two parameters, self and *args, where self is defined as self.method, which is one of the allowable method names (i.e., Fisher, Pearson, Ed, Stouffer, George, Tippett) that are passed from CountPs and *args which allows for a varying number of arguments to a function—this functionality allows for *n* number of p-values to be used as input. Inside of SumOfPs, self.N is created as a list containing the *args, this will be used in CombinedPvalue. Each method (i.e., Fisher, Pearson, Ed, Stouffer, George, Tippett) takes two parameters, self and output, where self is defined as self.method, which is one of the allowable method names we created and output, which is the output of SumOfPs—a list of arguments. The specifics of each method will be reviewed next. Finally, CombinedPvalue method take two parameters, self and output, where self.method is defined as

one of the allowable method names and self.N is the length of p-values used as input for calculating the combined p-value within each method and output which is the test statistic returned from one of the six methods created (i.e., FisherMethod, PearsonMethod, GeorgeMethod, EdMethod, StoufferMethod, TippettMethod). Inside each if statement in CombinedPvalue the method's distribution, as defined by Heard and Rubin-Delanchy, 2018, is used to calculate the combined p-value.

FisherMethod. The FisherMethod takes the output of SumOfPs converts that to a variable called List, creates an empty temporary list called temp, and iterates over a for loop of each element in List. Within the for loop we take the log of each element in List and append it to temp. Finally, we take the sum of temp and multiple it by -2, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

PearsonMethod. The PearsonMethod takes the output of SumOfPs converts that to a variable called List, creates an empty temporary list called temp, and iterates over a for loop of each element in List. Within the for loop we take the negative log of 1 minus the element in List and append it to temp. Finally, we take the sum of temp and multiple it by -2, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

GeorgeMethod. The GeorgeMethod takes the output of SumOfPs converts that to a variable called List, creates an empty temporary list called temp, and iterates over a for loop of each element in List. Within the for loop we take the log of each element divided by 1 minus the element in List and append it to temp. Finally, we take the negative sum of temp, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

EdMethod. The EdMethod takes the output of SumOfPs converts that to a variable called List, creates an empty temporary list called temp, and iterates over a for loop of each

element in List. Within the for loop we take the element in List and append it to temp. Finally, we take the sum of temp, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

StoufferMethod. The StoufferMethod takes the output of SumOfPs converts that to a variable called List, creates an empty temporary list called temp, and iterates over a for loop of each element in List. Within the for loop we take the inverse CDF of the standard normal distribution the element in List and append it to temp. Finally, we take the sum of temp, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

TippettMethod. The TippettMethod takes the output of SumOfPs converts that to a variable called List. The minimum of List is taken, as defined by Heard and Rubin-Delanchy, 2018, and stored in the variable output.

The following packages were used for the development of this Python solution and details related to each package such as reliability, repositories, maintenance etc. are to follow. Numpy 1.17.2, scipy.stats 1.3.1, and math 0.0.1. Numpy 1.17.2 was released September 6, 2019 and is maintained by Travis E. Oliphant et al. located here:https://pypi.org/project/numpy/1.17.2/ and with the repository located here: https://github.com/numpy/numpy. There is a large community of developers who utilize and maintain this package. Any dependencies or deprecations would be handled within this group. Scipy.stats was released August 8, 2019 and is maintained Scipy Developers: scipy-dev@python.org located here: https://pypi.org/project/scipy/#description and with the repository located here: https://github.com/scipy/scipy. There is a large community of developers who utilize and maintain this package. All issues and releases are handled within this group. Math 0.0.1 is a built-in Python module that is constantly maintained and does not require an install. The

documentation for this package is located here: https://docs.python.org/3/library/math.html#module-math. These three packages were selected and utilized in this solution for the following reasons: (1) numpy was used for the development of __main__ test cases described below, (2) scipy.stats contains norm, t, chi2, gamma, and beta classes which contain PDF and CDF functions, and (3) math contains mathematical functions such as log().

R code development

The initial code developed involved identifying both the current state and desired outcome of the solution. The purpose of outlining the current state and desired outcome was to specify the input, processes (i.e., functions), and output required. The program was written in R 3.5.2. due to it being a stable release with several bug fixes such as parse data now has deterministic parent nodes and buffering is disabled for file() connections to non-regular files among other fixes. User defined functions were created for this solution. There are eight user defined functions: SumOfPs, FishersMethod, PearsonsMethod, GeorgeMethod, EdgingtonMethod, StoufferMethod, TippettMethod, and CombinedPValues. SumOfPs function takes two parameters, x and … , where x is the initial p-value and "…" is indicative of n p-values used as input. Inside of SumOfPs, … is converted to a list and stored in the variable kwargs, we create a positive integer variable called pos which is used to instantiate a new environment called envir. X and kwargs are converted into a single list and stored in the variable output. Output is assigned to the new environment, envir, and returned at the end of the function. Output is assigned to the new environment for the purposes of CombinedPValues. Each function (i.e., FishersMethod, PearsonsMethod, GeorgeMethod, EdgingtonMethod, StoufferMethod, TippettMethod) takes one parameter, x, which is the output of SumOfPs—a list of arguments.

Each function will be described in detail below. Finally, CombinedPValues method take two parameters, x and name, where x is the test statistic returned from one of the six functions and name is set equal to the test statistic used to generate x. Within CombinedPValues, we utilize output, which was assigned to the new environment we created in SumOfPs, this is done in order to calculate the total number of p-values used in SumOfPs. The length, *n,* is used to determine the combined p-value based on one of the six functions created (i.e., FishersMethod, PearsonsMethod, GeorgeMethod, EdgingtonMethod, StoufferMethod, TippettMethod), as described by Heard and Rubin-Delanchy, 2018.

FishersMethod. The FishersMethod takes the output, x, of SumOfPs, create a temporary List—temp, equal to the length of x, where each element of x is iterated over a for loop. Within the for loop we take the log of each element in x and assign it to each empty element in temp. Finally, we take the sum of temp and multiple it by -2, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

PearsonsMethod. The PearsonsMethod takes the output, x, of SumOfPs, create a temporary List—temp, equal to the length of x, where each element of x is iterated over a for loop. Within the for loop we take the log of negative log of 1 minus the element in x and assign it to an empty element in temp. Finally, we take the sum of temp and multiple it by -2, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

GeorgeMethod. The GeorgeMethod takes the output, x, of SumOfPs, create a temporary List—temp, equal to the length of x, where each element of x is iterated over a for loop. Within the for loop we take the log of each element divided by 1 minus the element in x and assign it to an empty element in temp. Finally, we take the negative sum of temp, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

EdgingtonMethod. The EdgingtonMethod takes the output, x, of SumOfPs, create a temporary List—temp, equal to the length of x, where each element of x is iterated over a for loop. Within the for loop we take each element and assign it to an empty element in temp. Finally, we take the sum of temp, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

StoufferMethod. The StoufferMethod takes the output, x, of SumOfPs, create a temporary List—temp, equal to the length of x, where each element of x is iterated over a for loop. Within the for loop we take the inverse CDF of the standard normal distribution the element in x and assign it to an empty element in temp. Finally, we take the sum of temp, as defined by Heard and Rubin-Delanchy, 2018, which is stored in the variable output.

TippettMethod. The TippetMethod takes the output, x, of SumOfPs. The minimum of x is taken, as defined by Heard and Rubin-Delanchy, 2018, and stored in the variable output.

The following packages were used for the development of the R solution and details related to each package such as reliability, repositories, maintenance etc. are to follow. Chi 0.1, utils 2.10.1, dplyr 1.0.6, stats 3.6.2, along with standard built in R libraries. Chi 0.1 was released May 7, 2017, is maintained by David Kahle. The repository and CRAN location of the package are located here: https://github.com/dkahle/chi and here: https://cran.r-project.org/web/packages/chi/ There is a large community of developers who utilize, while the package has not been updated the current version is stable. Any dependencies or deprecations would be handled by David Kahle. Dplyr 1.0.6 was released May 05, 2021 and is maintained Hadley Wickham: hadley@rstudio.com located here: https://cran.r-project.org/web/packages/dplyr/index.html and with the repository located here: https://github.com/tidyverse/dplyr. There is a large community of developers who utilize and

maintain this package, as seen in the details of the repository. Stats 3.6.2 is the current release of the package and is maintained by R-core@r-project.org located here: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html. This package is handled and maintained by the core group of R developers. There is a large community of users and this package is consistently maintained.

Python code deployment

The P-CombiningPValuesFinal package 0.1.3 was deployed to both the Python Package Index and Test Python Package Index as of July, 5th 2021. The following location contains the latest release of P-CombiningPValuesFinal: https://pypi.org/project/P-CombiningPValuesFinal/. P-CombiningPValuesFinal works on both windows and IOS operating systems. The process outlined was followed in order to deploy to both Python Package Index and Test Python Package Index: https://packaging.python.org/tutorials/packaging-projects/ . All standard processes such as unit tests, READMe.md, Licenses, and Descriptions were created/tested/followed during the deployment of this package. Please see the master repository for details: https://github.com/StatsGirl/Master2021/tree/main/Python/packaging_tutorial. Release notes and documentation for the current deployment can be located here: https://libraries.io/pypi/P-CombiningPValuesFinal

R code deployment

The combinationpvalues package 0.1.3 was deployed to CRAN as of July 5th, 2021. The following location contains the latest release of combinationpvalues: https://cran.r-project.org/web/packages/combinationpvalues/index.html. Combinationpvalues works on Windows, Linux, and IOS operating systems. Testing notes related to compilation can be located here: https://cran.r-project.org/web/checks/check_results_combinationpvalues.html. The process

outlined was followed in order to deploy to CRAN: https://www.r-bloggers.com/2020/07/how-to-write-your-own-r-package-and-publish-it-on-cran/. All standard processes such as unit tests for functionality, README.md, NEWS, Description, and Licenses, along with making modifications requested by CRAN prior to deployment was followed during the deployment of this package. Please see the master repository for details: https://github.com/StatsGirl/Master2021/tree/main/R/combinationpvalues.

Testing of Packages

of packages, each function was tested within each package during the direct execution of the program . This was completed for feasibility and debugging purposes prior to deployment. Additionally, testing of significance of combined p-values for each statistic was also completed. The following tests were run in __main__ in Python and R prior to obtaining external data, all tests can be reviewed in the repositories.

1. Generate random sample sets of length 10,12,15,18, and 20 for $N(\mu, \sigma^2)$ where μ and σ^2 are random values
2. Generate random sample sets of length 10,12,15,18, and 20 for $N(0, \sigma^2)$ where σ^2 is random values

Tests in __main__ for both Python and R can be reviewed in the *Appendix A*.

Prior to deploying to CRAN, unit tests were required to be developed and tested. The unit tests and the results of these tests are also available in *Appendix B*.

Finally, datasets containing independent p-values were used for testing of each test statistic and significance of the combined p-value prior to deployment. Data was obtained from Michael Dewey's metap CRAN package and used for testing of each packages' functionality: https://cran.r-project.org/web/packages/metap/index.html along with an existing Python package

scipy.stats combined-pvalues, mentioned in the Introduction: https://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.stats.combine_pvalues.html.

Statistical Analyses

Application of Packages

In order to test the application of the two developed packages, data from the metap package was used and can be located in the Appendix. All test statistic functions in both Python and R were tested using this data. Once accordance across both the paper and deployed versions of the packages occurred, the data were applied to each of remaining four statistics. In addition to the Cheng and Sheng paper, data from the metap R package were also used. This package contains several p-value data sets from meta-analyses for testing purposes. In addition to utilized metap test data, functions developed were also used to test the precision of the functions created in both Python and R packages for the six statistics: Fishers', Pearson, Tippet, Stouffer, George, and Edgington.

Table 2

P-value data from metap R package used for testing purposes

	x
1	0.016
2	0.067
3	0.25
4	0.405
5	0.871

Note: metap_beckerp data was used for testing purposes. Variable x contains p-values. All other data sets can be found inside of the metap R package by calling data()

Table 3

Metap test results: Python vs. R

	Python	R
Stouffer	SS = -3.426708569291571, p = 0.06308810244433122	SS = -3.426709, p = 0.0630881
Tippett	ST = 0.016, p = 0.07748063336857602	ST = 0.016, p = 0.07748063
George	SG = 6.326206810885221, p = 0.0005199978719669197	SG = 6.326206810885221, p = 0.00051999
Fisher	SF = 18.533010182917298, p = 0.014522139680083129	SF = 18.53301, p = 0.01452214
Pearson	SP = 5.880596561146859, p = 0.0822941576158173	SP = 5.880597, p = 0.08229416
Edgington	SE = 1.609, p = 0.05587700414599501	SE = 1.609, p = 0.055877

Note: The test statistic is reported first followed by the p-value

Table 4

Scip.stats vs. Python vs. R deployed packages

	Scipy.stats	Python	R
Stouffer	SS = 1.5324706600034004, p = 0.06270316620567318	SS = -3.426708569291571, p = 0.06308810244433122	SS = -3.426709, p = 0.0630881
Ed	*NA*	SE = 1.609, p = 0.05587700414599501	SE = 1.609, p = 0.055877
George	SG = 6.326206810885219, p = 1.752434469305912e-07	SG = 6.326206810885221, p = 0.0005199978719669197	SG = 6.326206810885221, p = 0.00051999
Pearson	SP = 5.88059656114686, p = 0.8251932103488528	SP = 5.880596561146859, p = 0.0822941576158173	SP = 5.880597, p = 0.08229416
Fisher	SF = 18.533010182917298, p = 0.046611089917967294	SF = 18.533010182917298, p = 0.014522139680083129	SF = 18.53301, p = 0.01452214
Tippett	ST = 0.016, p = 0.922519366631424*	ST = 0.016, p = 0.07748063336857602	ST = 0.016, p = 0.07748063

Note: Sections of Scipy.stats that are empty do not contain functionality for those test statistics, and * is erroneous due to using the wrong tail of the distribution. This should be corrected as p = 1-0.92251936631424.

Results

The results of the testing between Python and R functions are in accordance with each other and the procedures described in Heard and Rubin-Delanchy (2018).

There are also several discrepancies between scipy.stats and the Python/R deployed packages that can be accounted for. There is no Edgington functionality available in the scipy.stats package and the discrepancy between the p-values reported in the Python package vs. the R package is addressed in the paragraph above. Regarding George's method, scipy.stats p-value is computed based on the test statistic $S_p - S_F$, an error is made in the calculation where the p-value is using the upper tail of the null distribution. Whereas the Python and R packages utilize the test statistic $S_F - S_P$ and use the lower tail of the null distribution. The Tippett method uses the lower tail of the null distribution in the scipy.stats functions, whereas the Python and R packages use the upper tail of the null distribution, which accounts for the *1-p-value* difference in Python and R reported p-values. Regarding the Stouffer method, scipy.stats uses the Stouffer Z-score method, which is described as $Z = \sum_{i=1}^{k} \frac{Z_i}{\sqrt{k}}$, where $Z_i = \phi^{-1}(1 - p_i)$, where Z is the Z-score for the overall independent tests. While the Python and R versions use the inverse normal cdf, $S_s = \sum_{i=1}^{n} \phi^{-1}(p_i)$, thus the slight differences in the reported combined p-values.

Results displayed in table 4 demonstrate that the current state of the Python and R packages are reliable based on the functionality in use. There are also several inconsistencies with the published scipy.stats package would be result in incorrect p-values if used. Overall, the discrepancies between the three available packages are apparent based on methodologies used. Through investigation into why these discrepancies exist and possible revisions to the scipy.stats

package would be recommended. Overview, Python and R results are in accordance and aligned with the affirmed methodology described in Heard and Rubin-Delanchy, 2018.

Discussion

The need for combining p-values is well documented in the statistical literature and in multiple areas of statistical application (Fisher, R. A. 1934; Pearson, 1933; Tippett, L.H.C. 1931; Edgington, E. S. 1972; Mudholkar, G, & George, E. 1979; Stouffer et al., 1949; Birnbaum, 1954). Of these, meta-analyses prove to be one use case for the combination of independent p-values, as stated by several authors (Heard, N. A. & Rubin-Delanchy, P., 2018; Cheng,L. & Sheng, .X. S., 2017), among others such as large genomic experiments (Zaykin, D. et al. 2007).

With differing test statistics, we are unable to combine these raw tests for overall assessment of the hypothesis in question. However, if $P1, \dots, Pk$ are the P-values of the $T1, \dots Tk$ are the different tests, the p-values are conformable, since each is random variable such that (a) $0 \leq P \leq 1, i = 1, \dots, k$ and (b)$\varphi i \sim U(0,1)$, $i = 1, \dots, k$ if Tis are continuous. Given this fact, we can combine P-values since they are of the same structure. Thus, Φ $(P1, \dots, Pk)$ is a real-valued random variable that is used to combine the results of the individual test for the purpose of testing H_0: All H_{01},…,H_{0k} are true versus H_1: At least one H_{1i} is true.

The need for a feasible and easy open-source solution that would allow for the combination of multiple p-values—derived from separate raw tests—was apparent after an extensive review of the current landscape of solutions that exist. With two current solutions, there were several limitations that would not allow for the use of all six test statistics stated in Heard and Rubin-Delanchy (2018) nor are all solutions user friendly.

Hence, the purpose of creating two open-source solutions that can be used in multiple operating systems and environments. Python and R are two popular programming languages that are used by both statisticians and data scientists. The development, testing, deployment, and release of both packages followed strict software development lifecycle principles and mathematical methodology lead out by Heard and Rubin-Delanchy (2018). The results of the two packages, along with an already available, yet limited, solution (i.e., scipy.stats.combined_pvalues) are detailed above. Although, there are methodology differences between our open-source solutions and the scipy.stats.combined_pvalues solution, the alignment and accordance with what is affirmed in the literature provides confidence that the two deployed solutions are reliable.

No solution is the perfect solution as it cannot cover all use cases. The current packages do not take into account correlations between independent variables. A solution for this caveat is discussed in Cheng and Sheng, 2017, as the combination of combination of p-values (CCP) test. The CCP test is based on a simple union of rejections decision rule that exploits the similarity between any two p-value combination methods (Cheng & Sheng, 2017). If at least one of the two p-value combination methods yields a rejection at the significance level γ, then the joint null hypothesis is rejected at the significance level α. The CCP test is proven to be reliable, however there is no easy way to implement its use. Currently there is no open-source solution for the CCP test. This functionality is not available in the current releases of the packages, however, with development and permissions from the authors, it can be developed and included.

Conclusions

There have been six affirmed methodologies that can be used to examine independent p-values, which are a result of independent hypotheses. The six test statistics are $S_F = -2\sum_{i=1}^{n}\log p_i$ (Fisher, 1934), $S_P = -2\sum_{i=1}^{n}\log(1-p_i)$ (Pearson, 1933), $S_G = S_F - S_P = -\sum_{i=1}^{K}\log\{(p_i)/(1-p_i)\}$ (George, E., 1977), $S_E = \sum_{i=1}^{K} p_i$ (Edgington, 1972), $S_s = \sum_{i=1}^{n}\phi^{-1}(p_i)$, where ϕ is the standard normal cumulative distribution function (Stouffer, et. al. 1949), and $S_T = \min(p_1, \dots, p_n)$ (Tippett, L. H. C. 1931). The required need for test statistics is apparent with the use of meta-analyses and genomics which often time have disparate data or differing tests that cannot be combined. With the need for these methodologies there was an apparent lack of programmable solutions that can be used by statisticians and data scientists. Thus, the two packages in Python and R were created to address this gap. The methodology for the test statistics and distributions were created based on Heard and Rubin-Delanchy, 2018. Each package followed strict Python and R programming standards and software development lifecycle principles. Based on strict criteria from CRAN and PyPi, the packages were granted permission for deployment and release. The feasibility of these two packages is apparent, as the gap for an open-source solution that contains all six test statistics was missing. Although this solution was needed, it does not go without its limitations. The current packages do not have functionality that account for correlations between independent variables. One possible solution for this is combination of combination of p-values (CCP) test, created by Cheng and Sheng, 2017. This solution would account for correlations between independent p-values and could be programed into both Python and R. The current state of both packages would allow users to combined independent p-values with affirmed tests and confidently report the combined p-values of multiple tests.

Acknowledgments

This Master's Thesis would not have been possible without the support and dedication to Dr. E. Olusegun George. His shift guidance and robust knowledge of the subject matter is immensely appreciated and admired.

References

Bahadur, R. R. (1976). Rates of Convergence of Estimates and Test Statistics. *The Annals of Mathematical Statistics*, *38*(2), 303–324. https://doi.org/10.1214/aoms/1177698949

Birnbaum, A. (1954). Combining Independent Tests of Significance. *Journal of the American Statistical Association*, *49*(267), 559–574.

Casella, G., & Berger, R. L. (2002). Statisticial Inference. Belmont, CA: Duxbury.

Cheng, L., & Sheng, X. S. (2017). Combination of "combinations of p values." *Empirical Economics*, *53*(1), 329–350. https://doi.org/10.1007/s00181-017-1230-9

Edgington, E. S. (1972). An Additive Method For Combining Probability Values From Independent Experiments. *The Journal of Psychology*, *80*, 351–363. https://doi.org/https://doi.org/10.1080/00223980.1972.9924813

Fisher, R. A. (1934). Statistical Methods for Research Workers. In *Annals of Applied Biology* (4th ed., Vol. 13, Issue 1). Edinburgh:Oliver & Boyd. https://doi.org/10.1111/j.1744-7348.1926.tb04258.x

George, E. O., & G. S. Mudholkar (1983). "On the convolution of logistic random variables." Metrika 30.1: 1-13.

George, E. O. (1977). Combining Independent one-sided and Statisticial Tests- Some Theory. Unpublished, PhD Dissertation, University of Rochester, 1977.

Heard, N. A., & Rubin-Delanchy, P. (2018). Choosing between methods of combining p -values. *Biometrika*, *105*(1), 239–246. https://doi.org/10.1093/biomet/asx076

Litell, R. & Folks, L. (1973). Asymptotic Optimality of Fisher's Method of Combining Independent Tests II. J. Amer. Stat. Assoc. 68:341, 193-194.

Loughin, T. M. (2004). A systematic comparison of methods for combining p-values from independent tests. *Computational Statistics and Data Analysis*, *47*(3), 467–485.

https://doi.org/10.1016/j.csda.2003.11.020

Mudholkar, G. S., & George, E. O. (1979). The logit statistic for combining probabilities. *Symposium on Optimizing Methods in Statistics*, *January 1979*, 345–366.

Pearson, K. (1933). On a Method of Determining Whether a Sample of Size n Supposed to Have Been Drawn from a Parent Population Having a Known Probability Integral has Probably Been Drawn at Random. *Biometrika*, *25*(3/4), 379. https://doi.org/10.2307/2332290

Poole, W., Gibbs, D.L., Shmulevich, I., Bernard, B., & Knijnenburg, T.A. (2016). Combining dependent P-Values with an Empirical Brown's Method. Bioinformatics, 32(17) 1430-1436.

Robert H. Berk & Arthur Cohen (1979) Asymptotically Optimal Methods of Combining Tests, Journal of the American Statistical Association, 74:368, 812-814

Stouffer, S. A., Suchman, E. A., Devinney, L. C., Star, S. A., & Williams, R. M. (1949). The American Soldier: Adjustment During Army Life. In *Princeton University Press* (Vol. 1). https://doi.org/10.2307/2087216

Tippett, L. H. C. (1931). *The Methods of Statistics*. London: Williams and Norgate, Ltd.

Zaykin, D. V., Zhivotovsky, L. A., Westfall, P. H., & Weir, B. S. (2002). Truncated product method for combining P-values. *Genetic Epidemiology*, *22*(2), 170–185. https://doi.org/10.1002/gepi.0042

Appendix A

Link to repository for original code can be found here: https://github.com/StatsGirl/Master2021
If you would like to contribute to the refinement of this code, please submit a request and your GitHub user will be approved to access and make commits to the repository.

Python code:

```
"""
    Functions within this module allow for multiple p values
    to be defined within each method below
    Method options include: Fisher, Pearson, Ed, Stouffer, George, Tippett
"""

#Purpose: Combining P-values methodology
#Author: Breya McGlown
#Math Master's Thesis

__version__ = "0.1.1"

import numpy as np
import copy
from scipy.stats import norm
import math as mt
from scipy.stats import t
from scipy.stats import chi2
from scipy.stats import gamma
from scipy.stats import beta

class CountPs:
    """
    Functions within this class allow for multiple p values
    to be defined within each method below
    Method options include: Fisher, Pearson, Ed, Stouffer, George, Tippett
    """

    def __init__(self, method):
        self.method = method

    def InfinitePs(self, *args):
        """
        Select n number of p-values to use with desired method
        enter p values into args parameter
        """
        if self.method == self.method:
            self.N = list(args)
            pass
```

```
        return list(args)

    def FisherMethod(self,output):
        """
        Fishers method
        """
        if self.method == 'Fisher':
            self.output = output
            List = output
            temp = []
            for x in List:
                temp.append(mt.log(x))
            temp1 = -2* sum(temp)
            output = temp1 #-2SF is distributed chisquare 2 ddof

        return output

    def PearsonMethod(self,output):
        """
        Pearsons Method
        """
        if self.method == 'Pearson':
                self.output = output
                List = output
                temp = []
                for x in List:
                    temp.append(mt.log(1 - x))
                temp1 = -2* sum(temp)
                output = temp1 #-2SP is distributed chisquare 2 ddof

        return output

    def GeorgeMethod(self,output):
        """
        Georges Method
        """
        if self.method == 'George':
                self.output = output
                List = output
                temp = []
                for x in List:

                    temp.append(mt.log(x/(1 - x)))
                temp1 = -1 * sum(temp) #SF-SP
                output = temp1 #SG is distributed t distribution

        return output
```

```
    def EdMethod (self,output):
        """
        Edgington's Method
        """

        if self.method == 'Ed':
            self.output = output
            List = output
            temp = []
            for x in List:
                temp.append(x)
            temp1 = sum(temp)
            output = temp1 #SE is Gaussian Distribution

        return output

    def StoufferMethod(self,output):
        """
        StoufferMethod
        """

        if self.method == 'Stouffer':
            self.output = output
            List = output
            temp = []
            for x in List:
                temp.append(norm.ppf(x)) #inverse CDF
            temp1 = sum(temp)
            output = temp1 #SS is N(0,n)

        return output

    def TippettMethod(self,output):
        """
        Tippett Method
        """
        if self.method == 'Tippett':
            self.output = output
            List = output
            output = min(List) #ST is Beta(1,n)

        return output

    def CombinedPvalue(self,output):
        """
        Returns the p value of the combined pvalues based on the
        method selected
        """
```

```
        self.n = len(self.N)

        if self.method == 'Tippett':
            self.output = output
            output = 1-(1-output)**self.n#beta.pdf(output,a = 1, b = self.n) #ST is
Beta(1,n)
        elif self.method == 'Stouffer':
            self.output = output
            output = norm.pdf(output,scale = self.n) #SS is N(0,n)

        elif self.method == 'George':
            self.output = output
            output = t.pdf(output,self.n) #SG is Student t distribution (n)
        elif self.method == 'Ed':
            self.output = output
            output = gamma.pdf(output,a = self.n) #SE is Gamma(x,n)
        elif self.method == 'Pearson':
            self.output = output
            output = chi2.pdf(output,2*self.n) #SP is Chi-square df=2n
        else:
            self.output = output
            output = chi2.pdf(output,2*self.n) #SF is Chi-square df=2n

        return output


if __name__ == "__main__":

    A = CountPs('Tippett') #Fisher, Pearson, Ed, Stouffer, George, Tippett
    Output = A.InfinitePs(0.1,.3,.7)
    Final = A.TippettMethod(Output)
    SignOrNot = A.CombinedPvalue(Final)
    #print(Final, SignOrNot)

    #Test
    #random generator 10,12,15,18,20 N(mu,sigma^2) various values of mu and sigma^2
    mu = np.random.randint(1, 10 + 1)
    sigma = np.random.randint(0, 10 + 1)
    List = [10,12,15,18,20] #sample size
    PvalsFromPaper =
[0.585,0.76,0.365,0.905,0.08,0.265,0.405,0.76,0.1,0.25,0.185,0.115,0.525,0.035,0.65,0.
035,0.075,0.01,0.205,0.43,0.52,0.435,0.12]
    for x in List:
        Various = np.random.normal(mu, sigma, x)
        Pvalues = norm.cdf(Various)
        Output = A.InfinitePs(Pvalues)
        #Get P values and combine
        Final = A.StoufferMethod(Output[0])
```

```
        print(Final)
        SignOrNot = A.CombinedPvalue(Final)
        #print(SignOrNot)
    #Get P values and Combine

    #random generator 10,12,15,18,20 t-statistic N(0,sigma^2)
    #Based on t-statistic each sample to test mu = 0. get P values and combine
    #mu = 0
    #sigma = np.random.random_integers(low = 1,high = 10, size = 1)
    List = [10,12,15,18,20] #sample size
    for x in List:
        Various = t.rvs(x-1, size = x)
        Pvalues = t.cdf(Various,x-1)
        Output = A.InfinitePs(Pvalues)
        #Get P values and combine
        Final = A.StoufferMethod(Output[0])
        print(Final)
        SignOrNot = A.CombinedPvalue(Final)
        #print(SignOrNot)
    #Testing functionality based on P values provided by Cheng and Sheng paper


    #Stouffer's test
    A = CountPs('Stouffer')
    Test = A.InfinitePs(PvalsFromPaper)
    print(Test[0])
    StouffersOut = A.StoufferMethod(copy.deepcopy(Test[0]))
    SignOrNot = A.CombinedPvalue(StouffersOut)
    print("Stouffer",StouffersOut, SignOrNot)

    #input metap_beckerp.csv data

    #Fishers Test against metap_beckerp.csv data used in R tests
    A = CountPs('Fisher')
    Input = A.InfinitePs(0.016,0.067,0.25,0.405,0.871)
    FishersOut = A.FisherMethod(Input)
    SignOrNot = A.CombinedPvalue(FishersOut)
    print("Fisher",FishersOut, SignOrNot)

    A = CountPs('Pearson')
    Input = A.InfinitePs(0.016,0.067,0.25,0.405,0.871)
    FishersOut = A.PearsonMethod(Input)
    SignOrNot = A.CombinedPvalue(FishersOut)
    print("Pearson",FishersOut, SignOrNot)

    A = CountPs('Ed')
    Input = A.InfinitePs(0.016,0.067,0.25,0.405,0.871)
    FishersOut = A.EdMethod(Input)
```

```
SignOrNot = A.CombinedPvalue(FishersOut)
print("Ed",FishersOut, SignOrNot)


A = CountPs('Tippett')
Input = A.InfinitePs(0.016,0.067,0.25,0.405,0.871)
FishersOut = A.TippettMethod(Input)
SignOrNot = A.CombinedPvalue(FishersOut)
print("Tippett",FishersOut, SignOrNot)

A = CountPs('George')
Input = A.InfinitePs(0.016,0.067,0.25,0.405,0.871)
FishersOut = A.GeorgeMethod(Input)
SignOrNot = A.CombinedPvalue(FishersOut)
print("George",FishersOut, SignOrNot)

A = CountPs('Stouffer')
Input = A.InfinitePs(0.016,0.067,0.25,0.405,0.871)
FishersOut = A.StoufferMethod(Input)
SignOrNot = A.CombinedPvalue(FishersOut)
print("Stouffer",FishersOut, SignOrNot)

#scipy.stats results
```

R Code:

```
#Purpose: Combining P-values methodology
#Author: Breya McGlown
#Math Master's Thesis


library(chi)
library(EmpiricalBrownsMethod)
require(utils)


#' Class
#'
#' Functions within this class allow for multiple p values
#' to be defined within each method below
#' Method options include: Fisher, Pearson, Edgington, Stouffer, George, and
#' Tippett
#'
#' @param infile Path to the input file
#' @return Functions within this class allow for multiple p values
#' @export
PickMethod <- function(x){
    "Functions within this class allow for multiple p values
    to be defined within each method below
    Method options include: Fisher, Pearson, Ed, Stouffer, George, Tippett"

    self.method<- x
    structure(class = "PickMethod", list(
        #methods
        #FishersMethod

#' SumOfPs
#'
#' Input n p-values
#' n = 2,3,...,k
#'
#' @param infile Path to the input file
#' @return n p-values used as input
#' @export
        SumOfPs = function(x,...){
            kwargs<-list(...)
            pos <- 1
            envir = as.environment(pos)
            output <- (c(x,kwargs))
            assign("output",output, envir = envir)
```

```
            return(output)
        },

#' Fishers Method
#'
#' Combination p-value method that uses Fishers statistic
#' Summation i=1 to n log of pi where p equals p-value
#' input is SumOfPs
#' output is test statistic
#'
#' @param infile Path to the input file
#' @return Combination p-value using Fishers method
#' @export
        FishersMethod = function(x) {
            if (self.method == "Fisher"){
            k <- 1
            Len<- length(x)
            temp <-vector("list",Len)
            for (i in x) {
            temp[[k]]<-log(i)
            k <- k + 1
            }
            temp1 <- Reduce("+",temp)
            output <- -2 * temp1
            return(output)
            }
        },
#' Pearson
#'
#' Combination p-value method that uses Pearson statistic
#' -Summation i= 1 to n log(1-pi) where p equals p value
#' input is SumOfPs
#' output is combined p value
#'
#' @param infile Path to the input file
#' @return Combination p-value using Pearson method
#' @export
        #PearsonsMethod
        PearsonsMethod = function(x) {
            if (self.method == "Pearson"){
            k <- 1
            Len<- length(x)
            temp <-vector("list",Len)
            for (i in x) {
            temp[[k]]<- ((log(1-i)))
            k <- k + 1
            }
            temp1 <- Reduce("+",temp)
```

```
            output <- -2 * temp1
            return(output)
            }
        },
#' George Method
#'
#' Combination p-value method that uses George statistic
#' Summation i=1 to n log(pi/(1-pi)) where p equals p-value
#' input is SumOfPs
#' output is test statistic
#'
#' @param infile Path to the input file
#' @return Combination p-value using George method
#' @export
        #GeorgeMethod
        GeorgeMethod = function(x) {
            if (self.method == "George"){
            k <- 1
            Len<- length(x)
            temp <-vector("list",Len)
            for (i in x) {
            temp[[k]]<- log(i/(1-i))
            k <- k + 1
            }
            temp1 <- Reduce("+",temp)
            output <- -2* temp1
            return(output)
            }
        },
#' Edgington Method
#'
#' Combination p-value method that uses Edgington statistic
#' Summation i=1 to n pi where p equals p-value
#' input is SumOfPs
#' output is test statistic
#'
#' @param infile Path to the input file
#' @return Combination p-value using Edgington method
#' @export
        #EdMethod
        EdMethod = function(x) {
            if (self.method == "Ed"){
            k <- 1
            Len<- length(x)
            temp <-vector("list",Len)
            for (i in x) {
            temp[[k]]<- i
            k <- k + 1
```

```
            }
            temp1 <- Reduce("+",temp)
            output <- temp1
            return(output)
            }
        },
#' Stouffer Method
#'
#' Combination p-value method that uses Stouffer statistic
#' Summation i=1 to n inverse CDF of N(0,1)(pi) where p equals p-value
#' input is SumOfPs
#' output is test statistic
#'
#' @param infile Path to the input file
#' @return Combination p-value using Stouffer method
#' @export
        #StoufferMethod
        StoufferMethod = function(x){
            if (self.method == "Stouffer"){
            k <- 1
            Len<- length(x)
            temp <-vector("list",Len)
            for (i in x) {
            temp[[k]]<- qnorm(i) #inverse CDF
            k <- k + 1
            }
            temp1 <- Reduce("+",temp)
            output <- temp1
            return(output)
            }
        },
#' Tippett Method
#'
#' Combination p-value method that uses Tippett statistic
#' min(p1,...,pn), n= 2,3,...,k where p equals p-value
#' input is SumOfPs
#' output is test statistic
#'
#' @param infile Path to the input file
#' @return Combination p-value using Tippett method
#' @export
        #TippettMethod
        TippettMethod = function(x){
            if (self.method == "Tippett"){
            temp <- Reduce(min,x)
            output <- temp
            return(output)
            }
```

```
        },

#' CombinedPValue Method
#'
#' Return the combined pvalue
#' output is combined p-value
#'
#' @param infile Path to the input file
#' @return Combination p-value using Tippett method
#' @export
#CombinedPValue
CombinedPValueMethod = function(x,name){
    output = get("output",envir = .GlobalEnv)
    n <- length(output) #how to get the length from SumOfPs into this function


    if (name == "Tippett"){
        outputs <- 1-(1-x)**n #pbeta(x, shape1 = 1, shape2 = n)
        return(outputs)
    }
    else if (name == "George"){
        outputs <- dt(x,df=n)
        return(outputs)
    }
    else if (name == "Pearson"){
        outputs <- dchisq(x,2*n)
        return(outputs)
    }
    else if (name == "Ed"){
        outputs <- dgamma(x,shape=n)
        return(outputs)
    }
    else if (name == "Stouffer"){
        outputs <- dnorm(x,sd=n)
        return(outputs)
    }
    else if (name == "Fisher"){
        outputs <- dchisq(x,2*n)
        return(outputs)
    }
}

    ))
}

#test __main__
#Fisher, Pearson, Ed, Stouffer, George, Tippett
my_object <- PickMethod("Perason")
```

```
Output <- my_object$SumOfPs(0.1,0.3,.7)
print(my_object$PearsonsMethod(Output))

#Test
my_object <- PickMethod("Stouffer")
#random generator 10,12,15,18,20 N(mu,sigma^2) various values of mu and sigma^2
    mu <- sample(0:10,1)
    sigma <- sample(0:10,1)
    List <- list(10,12,15,18,20) #sample size
    for (x in List){
        Various <- rnorm(x, mean=mu, sd=sigma)
        Pvalues = pnorm(Various)
        Output = my_object$SumOfPs(Pvalues)
        #Get P values and combine
        Final <- my_object$StoufferMethod(Output)
        #print(Final)
        #SignOrNot = A.DetermineSig(Final)
    }
    #Get P values and Combine

    #random generator 10,12,15,18,20 t-statistic N(0,sigma^2)
    #Based on t-statistic each sample to test mu = 0. get P values and combine
    #mu <- 0
    #sigma <- sample(1:10,1)
    List <- list(10,12,15,18,20) #sample size
    PvalsFromPaper <-
list(0.585,0.76,0.365,0.905,0.08,0.265,0.405,0.76,0.1,0.25,0.185,0.115,0.525,0.035,0.6
5,0.035,0.075,0.01,0.205,0.43,0.52,0.435,0.12)

    my_object <- PickMethod("Tippett")
    for (x in List){
        Various <- rt(x, x-1)
        Pvalues = pt(Various, x-1)
        Output = my_object$SumOfPs(Pvalues)
        #print(Output)
        #Get P values and combine
        Final <- my_object$StoufferMethod(Output)
        print(Final)
        #SignOrNot = A.DetermineSig(Final)
    }
    print("start")
    #Testing new Paper Sheng and Cheng
    input <- read.csv(file = '~/Desktop/Thesis2021/DataSets/metap_beckerp.csv')
    my_object <- PickMethod("Stouffer")
    Final <- my_object$StoufferMethod(input$x)
    print("Stouffer")
    print(Final)
    Combined <- CombinedPValueMethod(Final,"Stouffer")
```

```
print(Combined)

my_object <- PickMethod("Pearson")
Final <- my_object$PearsonsMethod(input$x)
print("Pearson")
print(Final)
Combined <- CombinedPValueMethod(Final,"Pearson")
print(Combined)

my_object <- PickMethod("George")
Final <- my_object$GeorgeMethod(input$x)
print("George")
print(Final)
Combined <- CombinedPValueMethod(Final,"George")
print(Combined)

my_object <- PickMethod("Ed")
Final <- my_object$EdMethod(input$x)
print("Ed")
print(Final)
Combined <- CombinedPValueMethod(Final,"Ed")
print(Combined)

my_object <- PickMethod("Tippett")
Final <- my_object$TippettMethod(input$x)
print("Tippett")
print(Final)
Combined <- CombinedPValueMethod(Final,"Tippett")
print(Combined)

my_object <- PickMethod("Fisher")
Final <- my_object$FishersMethod(input$x)
print("Fisher")
print(Final)
Combined <- CombinedPValueMethod(Final,"Fisher")
print(Combined)

my_object <- PickMethod("Fisher")
Final <- my_object$FishersMethod(PvalsFromPaper)
print(Final)
#Data dataframes from the EmpricialBrownsMethod package
#print(try(data(package = "metap") ))
```

Appendix B

```
library(testthat)        # load testthat package
library(combinationpvalues)       # load our package
library(spatstat.utils)

# Test whether the output is a data frame
test_that("test whether SumOfPs returns a list", {
  output<- SumOfPs(0.2)
  expect_type(output, "list")
})

# Test whether SumOfPs returns length >= 2
test_that("test whether SumOfPs returns length >= 2", {
  output<- SumOfPs(0.2,0.1,0.9)
  expect_gte(length(output), 2)
})

#expect_condition function
#created a function to check whether list of values are within a given range
expect_condition<- function(object) {
  # 1. Capture object and label
  act <- quasi_label(rlang::enquo(object), arg = "object")

  # 2. Call expect()
  rr <- c(0, 1)
  act$val <-lapply(act$val, as.numeric)
  #print(act)
  for (x in act$val) {
    act$range <-inside.range(x, rr)
    #print(act)
  }
  expect(
    act$range,
    sprintf("%s has range %e,%f, not range 0 to 1.", act$lab,
min(unlist(act$val))[1],max(unlist(act$val))[1])
  )

  # 3. Invisibly return the value
  invisible(act$range)
}
```

```r
# Test whether SumOfPs inputs are within range of 0 to 1 inclusive
test_that("test whether SumOfPs inputs are within range of 0 to 1 inclusive", {
  output<- SumOfPs(0.2,0.1,0.9)
  expect_condition(output)
})
```

```
==> Testing R file using 'testthat'

ℹ Loading combinationpvalues

══ Testing combinationpvalues.R ═══════════════════════════
═══════════════════════════════════════
[ FAIL 0 | WARN 0 | SKIP 0 | PASS 5 ] Done!

Test complete
```