Introduction to `IPTWtrim` R package

Xiaoya Wang

July 31, 2023

Contents

Welcome to the `IPTWtrim` package handbook! This vignette provides examples and usage instructions for the functions available in the `IPTWtrim` package. The package is designed to examine the performance of inverse probability of treatment weighting (IPTW) trimming on Average Treatment Effect (ATE) estimation. It enables users to estimate the Average Treatment Effect (ATE) using weight trimming and evaluate the performance of trimming for different trimming percentiles based on simulation studies in specific settings. We focus on scenarios where three confounding variables $\mathbf{X} = (X_1, X_2, X_3)$ follow a multivariate normal distribution. The treatment variable A is generated by a logistic regression model that includes all confounding variables $\mathbf{X}$, and the response variable Y is a linear combination of the confounding variables $\mathbf{X}$, treatment A, and a standard normal distributed error term.

1 Brief Introduction to IPTW Trimming

IPTW is a statistical method used for estimating the Average Treatment Effect (ATE) in observational studies with confounding variables. It assigns weights to observations based on the probability of treatment given observed confounders. However, sometimes extreme weights can lead to biased estimates. Some researchers have mentioned that IPTW trimming may address this issue by capping extreme weights at specific percentiles, thereby improving estimation reliability (Stuart, 2010).

2 Introduction to `IPTWtrim` R package

2.1 Overview

The `IPTWtrim` R package is a powerful tool designed for causal inference using Inverse Probability of Treatment Weighting (IPTW) with symmetric weight trimming. It provides a set of functions to estimate the Average Treatment Effect (ATE) and assess the estimation performance for different weighting percentiles in observational studies.

The `IPTWtrim` package has the following architecture:

```
## /Users/yaya/Desktop/IPTWtrim
## +-- DESCRIPTION
## +-- IPTWtrim.Rproj
## +-- LICENSE
## +-- LICENSE.md
## +-- NAMESPACE
## +-- R
## |   +-- IPTWtrim-package.R
## |   +-- dataGenFun.R
## |   +-- measureATEfun.R
## |   \-- trimSummary.R
## +-- README.md
## +-- man
## |   +-- IPTWtrim-package.Rd
## |   +-- dataGenFun.Rd
## |   +-- measureATEfun.Rd
## |   \-- trimSummary.Rd
## +-- tests
## |   +-- testthat
## |   \-- testthat.R
## \-- vignettes
##     +-- IPTWtrim_handbook.Rmd
##     +-- IPTWtrim_handbook.html
##     \-- references.bib
```

2.2 Basic examples of data generation

2.2.1 Installation

First, install and load the required R packages: `MASS` (Ripley et al., 2013), `ggplot2` (Wickham, 2011), `gridExtra` (Auguie et al., 2017).

```
install.packages("MASS")
install.packages("ggplot2")
install.packages("gridExtra")
library(MASS)
library(ggplot2)
library(gridExtra)
```

Next, we install and load the `IPTWtrim` package.

```
library(devtools)
```

```
## Loading required package: usethis
```

```
install_github("Dorayaya/IPTWtrim",
               build_vignettes = TRUE)
```

```
## Skipping install of 'IPTWtrim' from a github remote, the SHA1 (c1976115) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(IPTWtrim)
```

2.2.2 `dataGenFun`: Generate Data for Our Specific Setting

Now, we can use the `dataGenFun` function to generate data for a binary treatment (A) and continuous confounding variables (X_1, X_2, X_3). The confounding variables are assumed to be generated from a multivariate normal distribution with a specified mean and covariance matrix. The binary treatment A is modeled using a logistic link function with specified coefficients. The continuous outcome Y is generated based on the treatment and confounders using a linear model with normally distributed errors.

This table provides a description for each argument, and the default values are set according to the needs of my STAT900 final project.

Argument	Description	Default Value
`n`	The number of observations to generate from the normal distribution. Should be a positive integer.	`500`
`mu`	The mean vector of the multivariate normal distribution for predictors x1, x2, x3.	`c(10, 1, 11)`
`sigma.mat`	The covariance matrix of the multivariate normal distribution for predictors x1, x2, x3.	`matrix(c(11^2, 4, 4, 4, 3^2, 3, 4, 3, 8^2), 3)`
`eta`	The coefficients of the logistic regression model for the binary outcome A.	`c(-1.5, 0.1, 0.3, 0.2)`
`theta`	The coefficients of the linear regression model for the continuous outcome Y.	`c(110, -12, -0.3, -0.8, -0.2)`

The function will return a data frame including A, X_1, X_2, X_3 and Y. Here is an example with default parameter values:

```r
dataExample <- dataGenFun()
head(dataExample)
```

```
##          x1           x2        x3 A         Y
## 1 20.623786  1.477271510 26.124102 1  86.23191
## 2 -5.188464  1.547415720  4.051226 0 108.99010
## 3  8.808266  6.369400675 16.783986 1  84.95688
## 4 -3.590845  0.008511589 14.243071 1  95.37275
## 5 15.346993 -0.880983394 10.150084 0 104.64925
## 6 28.510843  2.312267419  7.317765 1  87.61734
```

2.3 Basic examples of estimating the Average Treatment Effect (ATE) using `measureATEfun`

The `measureATEfun` function facilitates the estimation of the Average Treatment Effect (ATE) through Inverse Probability of Treatment Weighting (IPTW) with symmetric trimming on data generated from the `dataGenFun` function. It enables users to assess the causal effect of the binary treatment A on the continuous outcome Y in the presence of confounding variables ($x1$, $x2$, $x3$), while evaluating performance for a specific weight trimming percentile. The function employs IPTW to balance covariate distributions across treatment groups and offers options for trimming extreme weights.

The following table provides descriptions for each argument, and the default values are configured based on the simulation design used for my STAT900 final project.

Argument	Description	Default Value
`nsim`	The number of iterations to perform. A positive integer.	`1000`
`trim.p`	The percentile of extreme weights to be trimmed on both ends. A numeric value between 0.5 and 1.	`0.95`
`n`	The number of observations in each simulated dataset. A positive integer.	`500`
`mu`	The mean vector of the multivariate normal distribution for confounders x1, x2, x3.	`c(10, 1, 11)`
`sig.mat`	The covariance matrix of the multivariate normal distribution for confounders x1, x2, x3.	`matrix(c(11^2, 4, 4, 4, 3^2, 3, 4, 3, 8^2), 3)`
`eta`	The coefficients of the logistic regression model for the binary treatment A.	`c(-1.5, 0.1, 0.3, 0.2)`
`theta`	The coefficients of the linear regression model for the continuous outcome Y.	`c(110, -12, -0.3, -0.8, -0.2)`

This function returns a matrix containing measures related to ATE estimation using trimming, as well as summary statistics for the estimated weights. The columns include:

Measure	Description
`bias`	The difference between the estimated and true ATE.
`mse`	The mean squared error (MSE) of the ATE estimator.
`weight.mean`	The average of trimmed weights across all simulations.
`weight.ese`	The empirical standard error of the mean of trimmed weights.
`weight.asd`	The average standard deviation of trimmed weights across all simulations.
`weight.max`	The average maximum trimmed weight across all simulations.

Measure	Description
`weight.min`	The average minimum trimmed weight across all simulations.

Here is an example with default parameter values:

```
measureATE.example <- measureATEfun()
```

3 Simulation Studies using `trimSummary`

The `trimSummary` function conducts a simulation study to assess the performance of different weight trimming percentiles. For each specified trimming percentile in `trimperc.try`, the function calculates various measures such as bias, empirical standard error (EmpSE), and mean squared error (MSE) related to the estimation of Average Treatment Effect (ATE) using weight trimming. Additionally, the function computes summary statistics for the estimated weights, including the mean of trimmed weights, standard error of the mean of trimmed weights, average standard deviation of trimmed weights, average maximum trimmed weight, and average minimum trimmed weight.

The function generates three plots using the `ggplot2` (Wickham, 2011) and `gridExtra` (Auguie et al., 2017) packages to visualize the relationship between the weight trimming percentile and the resulting bias, EmpSE, and MSE.

The table provides a description for each argument, with default values set based on the simulation design for my STAT900 final project.

Argument	Description	Default Value
`nsim`	The number of iterations to perform for each weight trimming percentile. A positive integer.	`1000`
`n`	The number of observations in each simulated dataset. An integer.	`500`
`mu`	The mean vector of the multivariate normal distribution for confounders x1, x2, x3.	`c(10, 1, 11)`
`sig.mat`	The covariance matrix of the multivariate normal distribution for confounders x1, x2, x3.	`matrix(c(11^2, 4, 4, 4, 3^2, 3, 4, 3, 8^2), 3)`
`eta`	The coefficients of the logistic regression model for the binary outcome A.	`c(-1.5, 0.1, 0.3, 0.2)`
`theta`	The coefficients of the linear regression model for the continuous outcome Y.	`c(110, -12, -0.3, -0.8, -0.2)`
`trimperc.try`	A numeric vector specifying the weight trimming percentiles to evaluate.	`c(1, 0.99, 0.95, 0.9, 0.75, 0.5)`
`seed`	An optional seed value for reproducibility of simulations. Default to `NULL`.	`20739377` (if not specified)

The function performs a simulation study for each specified weight trimming percentile, generates three plots to visualize the relationship between the weight trimming percentile and bias, EmpSE, and MSE. Also, it calculates the following metrics for each percentile.

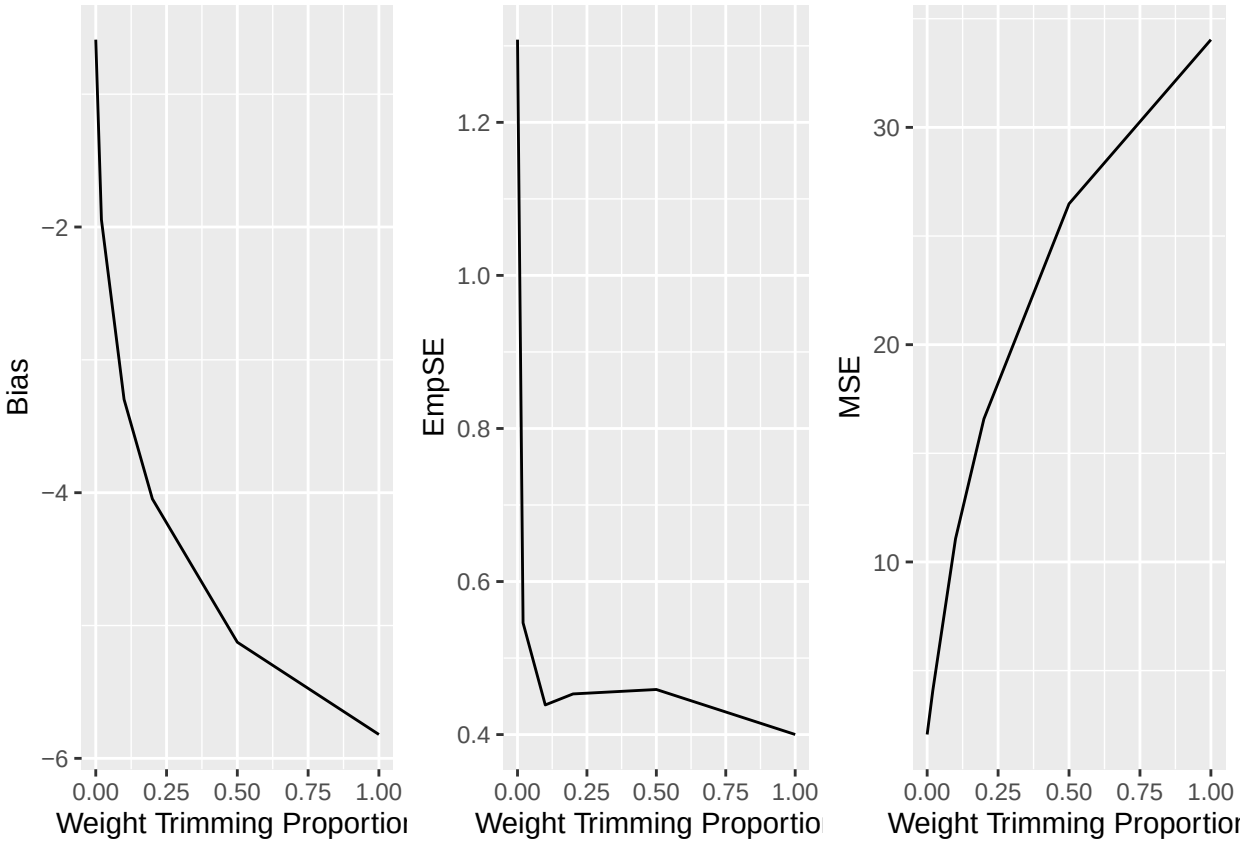
Metrics	Description
`trim.perc`	Weight trimming percentile evaluated (e.g. 1, 0.99, 0.95, 0.9, 0.75, 0.5).
`trim.prop`	Weight trimming proportion calculated as (1 - `trim.perc`) * 2.
`weight.mean`	The average of trimmed weights across all simulations.
`weight.ese`	The empirical standard error of the mean of trimmed weights.

Metrics	Description
`weight.asd`	The average standard deviation of trimmed weights across all simulations.
`weight.max`	The average maximum trimmed weight across all simulations.
`weight.min`	The average minimum trimmed weight across all simulations.
`bias`	The difference between the estimated and true ATE.
`ese`	Empirical standard error of the estimator, computed as the square root of mse - biasˆ2.
`mse`	The mean squared error (MSE) of the ATE estimator.

3.1 Simulation results

Let us run the simulation with default parameter values and observe the results.

```
trimSummary.example <- trimSummary()
```




```
trimSummary.example
```

```
##   trim.perc trim.prop weight.mean weight.ese weight.asd weight.max weight.min
## 1      1.00      0.00    1.979704 0.30420924  5.2797583  94.756169   1.000179
## 2      0.99      0.02    1.725863 0.06685995  1.9245078  14.248130   1.000777
## 3      0.95      0.10    1.502412 0.05494706  0.8727856   4.369460   1.003206
## 4      0.90      0.20    1.382307 0.05224444  0.5282690   2.631357   1.007117
## 5      0.75      0.50    1.207139 0.03977363  0.1893395   1.483407   1.027051
## 6      0.50      1.00    1.121103 0.02864148  0.0000000   1.121103   1.121103
##         bias       ese       mse
## 1 -0.5895103 1.3080832  2.058604
## 2 -1.9446404 0.5459651  4.079704
## 3 -3.2975645 0.4387043 11.066393
```

```
## 4 -4.0470055 0.4531161 16.583568
## 5 -5.1256602 0.4588987 26.482981
## 6 -5.8204212 0.4001030 34.037385
```

4 Package Testings

We use `testthat` package to do tests.

```
library(testthat)
```

```
##
## Attaching package: 'testthat'
```

```
## The following object is masked from 'package:devtools':
##
##     test_file
```

4.1 Testing `dataGenFun`

The tests for the `dataGenFun` function were designed to ensure that the function behaves correctly and handles invalid inputs appropriately. The first five tests check for specific error messages when invalid input is provided, such as a negative sample size, an invalid mean vector, an invalid covariance matrix, an invalid eta vector, or an invalid theta vector. These tests ensure that the function throws appropriate errors and provides meaningful error messages when the input is not in the correct format.

The sixth test verifies that the `dataGenFun` function returns a data frame as expected. This ensures that the function generates the output in the correct format.

The seventh test confirms that the generated data frame has the correct column names: "x1", "x2", "x3", "A", and "Y".

By running these tests, we can be more confident that the `dataGenFun` function behaves as intended, handles invalid inputs appropriately, and produces the expected output. Passing these tests provides evidence that the function is working correctly.

```
# Test 1: Test that dataGenFun returns an error with negative sample size
test_that("dataGenFun throws error with negative n", {
  expect_error(dataGenFun(n = -5,  mu = c(10, 1, 11), sigma.mat = matrix(c(11^2, 4, 4, 4, 3^2, 3, 4, 3,
                          eta = c(-1.5,0.1, 0.3, 0.2), theta =  c(110, -12, -0.3, -0.8, -0.2)), "sample
})
```

```
## Test passed
```

```
# Test 2: Test that dataGenFun returns an error with invalid mu
test_that("dataGenFun throws error with invalid mu", {
  expect_error(dataGenFun(n = 500,  mu = c(10, 1, 11,20), sigma.mat = matrix(c(11^2, 4, 4, 4, 3^2, 3, 4
                          eta = c(-1.5,0.1, 0.3, 0.2), theta =  c(110, -12, -0.3, -0.8, -0.2)), "the mea
})
```

```
## Test passed
```

```
# Test 3: Test that dataGenFun returns an error with invalid covariance matrix
test_that("dataGenFun throws error with invalid covariance matrix", {
  expect_error(dataGenFun(n = 500,  mu = c(10, 1, 11), sigma.mat = matrix(c(11^2, 4, 4, 4, 3^2, 3, -4, -
                          eta = c(-1.5,0.1, 0.3, 0.2), theta =  c(110, -12, -0.3, -0.8, -0.2)), "the co
})
```

```
## Test passed
```

```r
# Test 4: Test that dataGenFun returns an error with invalid eta
test_that("dataGenFun throws error with invalid eta", {
  expect_error(dataGenFun(n = 500,  mu = c(10, 1, 11), sigma.mat = matrix(c(1, 4, 4, 4, 1, 3, 4, 3, 1),
                          eta = 1.5, theta =  c(110, -12, -0.3, -0.8, -0.2)),"eta must be a numeric vec
})
```

```
## Test passed
```

```r
# Test 5: Test that dataGenFun returns an error with invalid theta
test_that("dataGenFun throws error with invalid eta", {
  expect_error(dataGenFun(n = 500,  mu = c(10, 1, 11), sigma.mat = matrix(c(1, 4, 4, 4, 1, 3, 4, 3, 1),
                          eta = c(0,0.1, 0.3, 1), theta =  c(110, -12, 200)),"theta must be a numeric v
})
```

```
## Test passed
```

```r
# Test 6: Test that dataGenFun returns a data frame
test_that("dataGenFun returns a data frame", {
  data <- dataGenFun()
  expect_true(is.data.frame(data))
})
```

```
## Test passed
```

```r
# Test 7: Test that dataGenFun generates data with correct column names
test_that("dataGenFun generates data with correct column names", {
  data <- dataGenFun()
  expect_equal(names(data), c("x1", "x2", "x3", "A", "Y"))
})
```

```
## Test passed
```

4.2 Testing `measureATEfun`

The tests for the `measureATEfun` function aim to ensure its proper functionality and handling of various inputs. The first seven tests focus on identifying potential errors and providing meaningful error messages when invalid inputs are provided. These tests check for negative values or inappropriate data types for parameters `nsim`, `trim.p`, `n`, `mu`, `sig.mat`, `eta`, and `theta`. Ensuring these errors are properly handled helps guarantee that the function responds robustly to different inputs and provides helpful feedback to users when necessary.

The eighth test verifies that `measureATEfun` returns a numeric matrix. This test ensures that the output format is consistent and reliable, which is essential for further analysis and interpretation.

The ninth test checks that the returned matrix has the expected dimensions, with one row and seven columns. This confirms that the function is generating the correct output structure.

The tenth test focuses on evaluating the accuracy of the bias and mean squared error (MSE) calculations performed by `measureATEfun`. By comparing the results with the expected values, this test helps ensure the correctness of the calculations and the reliability of the function's output.

Passing these tests confirms the reliability and correctness of the `measureATEfun` function.

```r
# Test 1: Test that measureATEfun returns an error with negative nsim
test_that("measureATEfun throws error with negative nsim", {
  expect_error(measureATEfun(nsim = -5), "the number of iterations must be a positive integer")
})
```

```
## Test passed
```

```r
# Test 2: Test that measureATEfun returns an error with invalid trim.p
test_that("measureATEfun throws error with invalid trim.p", {
  expect_error(measureATEfun(trim.p = 1.1), "the trimming percentile must be a number between 0.5 and 1
})
```

```
## Test passed
```

```r
# Test 3: Test that measureATEfun returns an error with negative n
test_that("measureATEfun throws error with negative n", {
  expect_error(measureATEfun(n = -500), "the sample size must be a positive integer")
})
```

```
## Test passed
```

```r
# Test 4: Test that measureATEfun returns an error with invalid mu
test_that("measureATEfun throws error with invalid mu", {
  expect_error(measureATEfun(mu = c(10, 1, 11, 20)), "the mean vector must be a numeric vector of lengt
})
```

```
## Test passed
```

```r
# Test 5: Test that measureATEfun returns an error with invalid covariance matrix
test_that("measureATEfun throws error with invalid covariance matrix", {
  expect_error(measureATEfun(sig.mat = matrix(c(11^2, 4, 4, 4, 3^2, 3, -4, -3, 8^2), 3)),
               "the covariance matrix sigma must be a numeric symmetric 3 by 3 matrix")
})
```

```
## Test passed
```

```r
# Test 6: Test that measureATEfun returns an error with invalid eta
test_that("measureATEfun throws error with invalid eta", {
  expect_error(measureATEfun(eta = 1.5), "eta must be a numeric vector of length 4")
})
```

```
## Test passed
```

```r
# Test 7: Test that measureATEfun returns an error with invalid theta
test_that("measureATEfun throws error with invalid theta", {
  expect_error(measureATEfun(theta = c(110, -12, -0.3)), "theta must be a numeric vector of length 5")
})
```

```
## Test passed
```

```r
# Test 8: Test that measureATEfun returns a numeric matrix
test_that("measureATEfun returns a numeric matrix", {
  result <- measureATEfun()
  expect_true(is.matrix(result))
})
```

```
## Test passed
```

```r
# Test 9: Test that measureATEfun returns a matrix with 1 row and 7 columns
test_that("measureATEfun returns a matrix with 1 row and 7 columns", {
  result <- measureATEfun()
  expect_equal(nrow(result), 1)
  expect_equal(ncol(result), 7)
})
```

```
## Test passed
```

```
# Test 10: Test that measureATEfun calculates the bias and mse correctly
test_that("measureATEfun calculates the bias and mse correctly", {
  result <- measureATEfun(nsim = 100, n = 500)
  expect_true(!is.na(result[1, "bias"]))
  expect_true(!is.na(result[1, "mse"]))
})
```

```
## Test passed
```

4.3 Testing `trimSummary`.

The `trimSummary` function has been thoroughly tested to ensure its correctness and reliability. The series of tests cover various scenarios, including negative and invalid inputs for `nsim`, `n`, `mu`, `sig.mat`, `eta`, `theta`, and `trimperc.try`. The tests are designed to validate the function's behavior when dealing with incorrect or unexpected inputs, making sure it provides meaningful error messages and gracefully handles invalid cases. Additionally, the tests verify that the function produces the correct output format, returning a data frame with the expected number of rows and columns.

```
# Test 1: Test that trimSummary returns an error with negative nsim
test_that("trimSummary throws error with negative nsim", {
  expect_error(trimSummary(nsim = -5), "the number of iterations must be a positive integer")
})
```

```
## Test passed
```

```
# Test 2: Test that trimSummary returns an error with negative n
test_that("trimSummary throws error with negative n", {
  expect_error(trimSummary(n = -500), "the sample size must be a positive integer")
})
```

```
## Test passed
```

```
# Test 3: Test that trimSummary returns an error with invalid mu
test_that("trimSummary throws error with invalid mu", {
  expect_error(trimSummary(mu = c(10, 1, 11, 20)), "the mean vector must be a numeric vector of length 3
})
```

```
## Test passed
```

```
# Test 4: Test that trimSummary returns an error with invalid covariance matrix
test_that("trimSummary throws error with invalid covariance matrix", {
  expect_error(trimSummary(sig.mat = matrix(c(11^2, 4, 4, 4, 3^2, 3, -4, -3, 8^2), 3)),
               "the covariance matrix sigma must be a numeric symmetric 3 by 3 matrix")
})
```

```
## Test passed
```

```
# Test 5: Test that trimSummary returns an error with invalid eta
test_that("trimSummary throws error with invalid eta", {
  expect_error(trimSummary(eta = 'happy'), "eta must be a numeric vector of length 4")
})
```

```
## Test passed
```

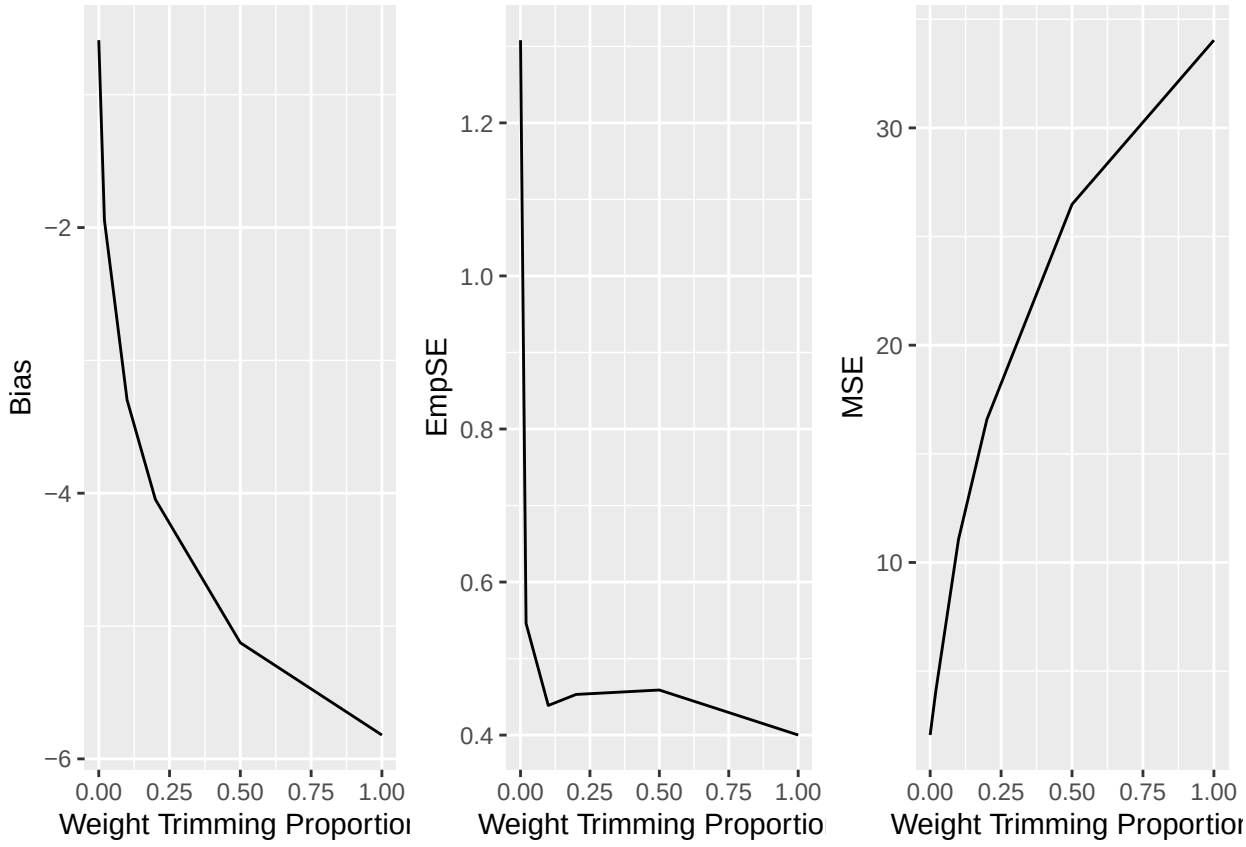
```
# Test 6: Test that trimSummary returns an error with invalid theta
test_that("trimSummary throws error with invalid theta", {
  expect_error(trimSummary(theta = "c(110, -12, -0.3)"), "theta must be a numeric vector of length 5")
})
```

```
## Test passed
```

```
# Test 7: Test that trimSummary returns an error with invalid trimperc.try
test_that("trimSummary throws error with invalid trimperc.try", {
  expect_error(trimSummary(trimperc.try = c(1.1, 0.5, -0.9)),
               "trimperc.try must be a numeric list, each number in the list must between 0.5 to 1")
})
```

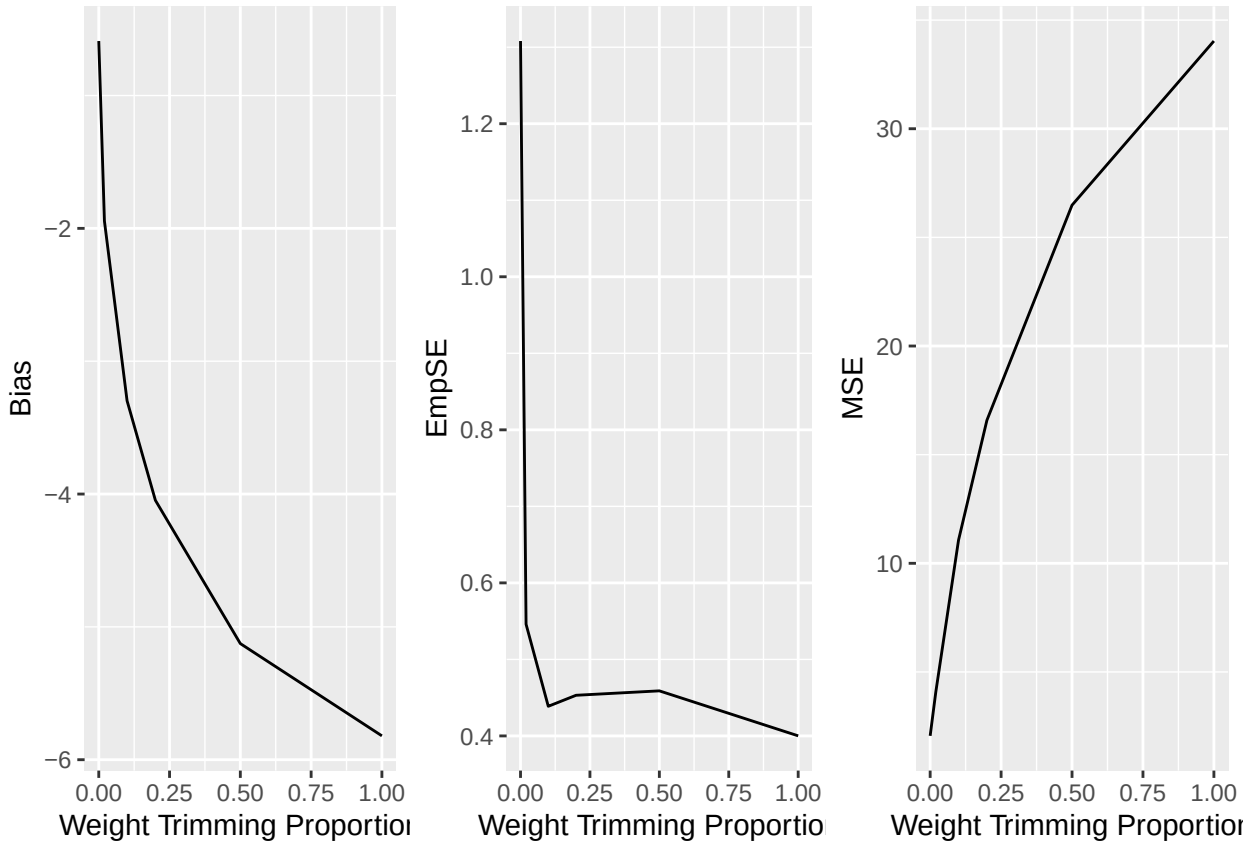
```
## Test passed
```

```
# Test 8: Test that trimSummary returns a data frame
test_that("trimSummary returns a data frame", {
  result <- trimSummary()
  expect_true(is.data.frame(result))
})
```




```
## Test passed
```

```
# Test 9: Test that trimSummary returns a data frame with expected number of rows and columns
test_that("trimSummary returns a data frame with expected number of rows and columns", {
  result <- trimSummary()
  expect_equal(nrow(result), length(c(1, 0.99, 0.95, 0.9, 0.75, 0.5)))
  expect_equal(ncol(result), 10)
})
```




```
## Test passed
```

References

Auguie, B., Antonov, A., and Auguie, M. B. (2017). Package 'gridextra'. *Miscellaneous Functions for "Grid" Graphics*.

Ripley, B., Venables, B., Bates, D. M., Hornik, K., Gebhardt, A., Firth, D., and Ripley, M. B. (2013). Package 'mass'. *Cran r*, 538:113–120.

Stuart, E. A. (2010). Matching methods for causal inference: A review and a look forward. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 25(1):1.

Wickham, H. (2011). ggplot2. *Wiley interdisciplinary reviews: computational statistics*, 3(2):180–185.