The Bootstrap

John Clements

2/26/2022

Requirements

- `dplyr`: for general data manipulation
- `parallel`: for parallel computation
- `tidyquant`: for access to financial data
- `lubridate`: for date manipulation
- `hyfo`: for resampling frequency of time-series
- `ggplot2`: for data visualization

I. Introduction

When estimating a statistic of interest, we are not only interested in the point-value of that statistic. We also want to quantify our uncertainty about our estimate. For standard quantities of interest, like the mean or standard deviation, we have analytical results for the sampling distributions or can easily use large-sample approximations. Often times, we are interested in quantities where we do not know the sampling distribution and deriving it is difficult, if it is even possible. In these cases, the bootstrap is a valuable tool for quantifying uncertainty.

When bootstrapping, we treat our sample as the population. We repeatedly resample the same number of observations as the original sample *with replacement* and calculate the statistic of interest on those samples. We resample with replacement because the samples would not be independent when sampling without replacement. They would also exactly match our original sample, which would not be very helpful. This provides us with a simulated sampling distribution with which we can analyze.

II. Implementation

While there are libraries like `bootstrap` that do bootstrapping for us, I find coding lightweight functions a great learning method, even though I use well maintained libraries in practice.

To demonstrate bootstrapping, I wrote two main functions functions and two helpers.

A. Single Bootstrap Function

The function `one_bootstrap` is a helper function that takes in a data vector and a function to estimate a statistic of interest and computes the quantity of interest on a single bootstrapped sample.

```r
one_bootstrap <- function(my_vec, my_func){
  ###
  # This function retrieves one bootstrapped sample and returns the statistic
  # of interest.
  #
  # Args
  # ----
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  #
  # Returns
  # -------
  # double
  #   The statistic of interest computed on the bootstrapped sample.
  #
  ###
  bootstrapped_sample <- sample(my_vec, size=length(my_vec), replace=TRUE)
  return(my_func(bootstrapped_sample))
}
```

B. Bootstrapping using Looping

The function `bootstrap_replicate` takes in a data vector, function, and the number of bootstrap iterations to perform, **B**, and returns a list with the mean, standard error, and estimates themselves. It does this by calling `one_bootstrap` **B** times. R's `replicate` function performs loops in a faster manner than explicitly writing a for loop.

```r
bootstrap_replicate <- function(my_vec, my_func, B){
  ###
  # This function takes in a data vector, function, and the number of bootstrap
  # iterations and returns a list holding the mean and standard deviation of the
  # bootstrap estimates as well, as the vector of the bootstrap estimates. It
  # utilizes the replicate function for optimized looping.
  #
  # Args
  # ----
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  # B : int
  #   The number of bootstrapped samples to return.
  #
  # Returns
  # -------
  # output : list
  #   A list of the mean, and standard deviation of the estimates and a vector
  #   of the estimates.
  #
  ###
  estimates <- replicate(B, one_bootstrap(my_vec, my_func))
  output <- list(
    'mean' = mean(estimates),
    'se' = sd(estimates),
    'estimates' = estimates
  )
  return(output)
}
```

C. Parallelized Bootstrapping

Processing each bootstrapped sample does not depend on the other samples, as the samples are independent of one another. This allows bootstrapping to be easily parallelized, which is what `bootstrap_parallel` does.

```
bootstrap_replicate_par <- function(B, my_vec, my_func){
  ###
  # This function is a helper function for the parallized bootstrapping function.
  # It takes in a vector whose length determines the number of bootstrap samples
  # to take, a data vector, and a function. It returns the list of bootstrapped
  # estimates from my_func.
  #
  # Args
  # ----
  # B : vector
  #   A vector whose length determines of bootstrapped samples to return.
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  #
  # Returns
  # -------
  # estimates : vector
  #   A vector of the estimates.
  #
  ###
  estimates <- replicate(length(B), one_bootstrap(my_vec, my_func))
  return(estimates)
}
```

```
bootstrap_parallel <- function(my_vec, my_func, B){
  ###
  # This function takes in a data vector, function, and the number of bootstrap
  # iterations and returns a list holding the mean and standard deviation of the
  # bootstrap estimates as well, as the vector of the bootstrap estimates. It
  # utilizes parallel computing.
  #
  # Args
  # ----
  # my_vec : numeric vector
  #   A vector of numbers of which to compute the statistic of interest.
  # my_func : function
  #   Function which computes the statistic of interest.
  # B : int
  #   The number of bootstrapped samples to return.
  #
  # Returns
  # -------
  # output : list
  #   A list of the mean, and standard deviation of the estimates and a vector
  #   of the estimates.
  #
  ###

  # Count the cores and make a cluster from leaving one core free.
  cores <- detectCores()
  cluster <- makeCluster(cores - 1)

  # Create a vector that will be split up and determine the number of bootstrap
  # samples to get on each core.
  boot_vec <- 1:B

  # Export functions to the cluster.
  clusterExport(
    cluster,
    list("boot_vec", "one_bootstrap", "bootstrap_replicate_par", "my_vec",
         "my_func"),
    envir=environment()
    )

  estimates <- parSapply(
    cluster,
    boot_vec,
    FUN=bootstrap_replicate_par,
    my_vec=my_vec,
    my_func=my_func
    )

  stopCluster(cluster)

  output <- list(
    'mean' = mean(estimates),
    'se' = sd(estimates),
    'estimates' = estimates
  )

  return(output)
}
```

With our functions defined, we can move onto applications

III. Applications

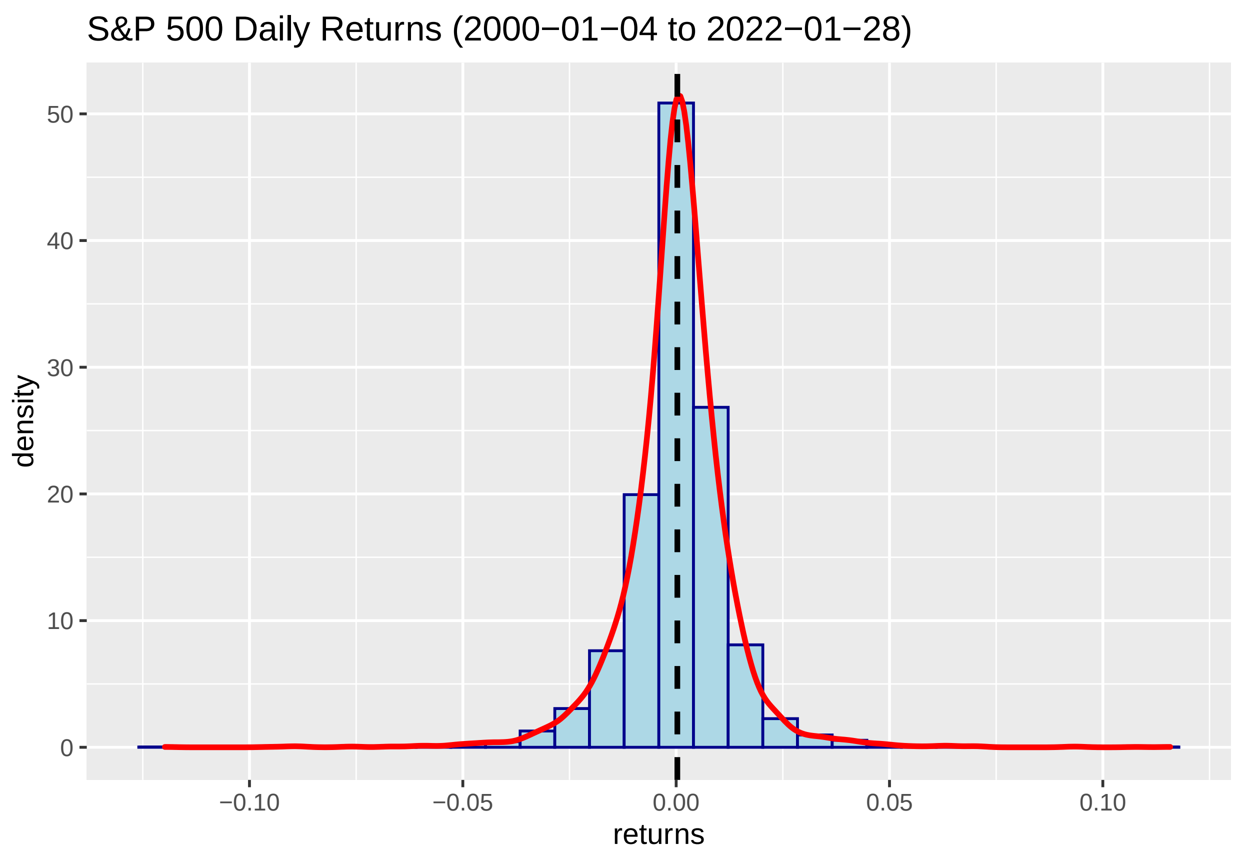
To demonstrate bootstrapping, I will apply it to the Sharpe ratio, Value at Risk VaR, and Expected Shortfall ES, sometimes known as Conditional Value at Risk or CVaR.

The data set I am using for this demonstration is the daily returns of the S&P 500 from the beginning of 2000 to the end of January 2022. I lose a day at the very beginning due to calculating returns. I also downloaded the 3-month Treasure interest rates, converting BPS to decimals and converted annual interest rates to daily rates to use for the Sharpe Ratio demonstration.

I am assuming returns are *iid* for simplicity, although this is a dubious assumption due to the possibility of correlations in returns and volatility clustering.

Below is a plot of daily returns. We can see daily returns are peaked slightly to the right of 0. The tails are long, with the left-tail a bit longer than the right, meaning there have been bigger losses on the worst days than the biggest gains on the best days, which is common in financial data.

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.

## [1] "^GSPC"
```



Now let's look at the Sharpe ratio!

B. Sharpe Ratio

The Sharpe ratio is the expected returns in excess of the risk-free rate-of-return per unit of risk. Risk is measured by the standard deviation of returns. We can estimate the Sharpe Ratio with the sample mean of excess returns divided by the standard deviations of the returns.

$$SR = \frac{E[R - r_f]}{\sigma_R}$$

The standard error of the Sharpe Ratio is computed as:

$$\hat{SE}(SR) = \sqrt{\frac{1 + \frac{1}{2}\hat{SR}^2}{n}}$$

The function below is calculates the Sharpe ratio given the returns and risk-free rate.

```r
sharpe_ratio <- function(returns, rf=GSPC$rf){
  return(mean(returns-rf)/sd(returns))
}
```

The way I wrote it is not conducive to use with my parallelized function because it cannot export the the GSPC dataframe to the cluster, so we will use the bootstrapping function utilizing optimized looping.

```r
set.seed(1)
B <- 5000

sharpe_est_boot <- bootstrap_replicate(GSPC$returns, sharpe_ratio, B)
```

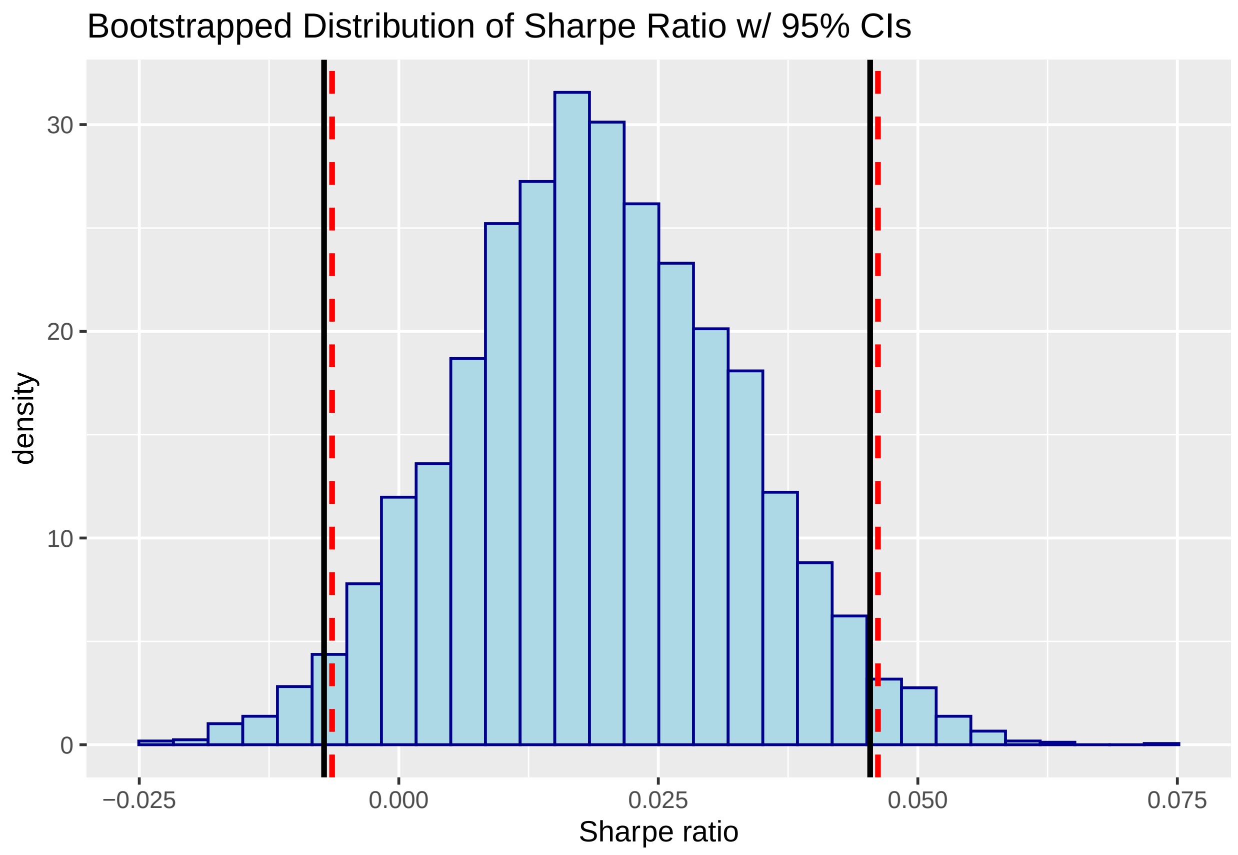
Below is a table comparing the estimates for the Sharpe Ratio, and the estimated standard error and 95% Confidence Intervals (CIs). The CIs are only approximate because the CI in the first row rely on the Central Limit Theorem and the CI in the second row are from the bootstrap estimates quantiles.

Table 1: Comparison of Estimates of Sharpe Ratio w/ 95 % CIs

Method	Est	SE	Lower	Upper
MLE	0.0191024	0.0134195	-0.0071993	0.0454042
Bootstrapped	0.0191407	0.0134148	-0.0064260	0.0461585

The values are very close, but the estimate of the Sharpe ratio and the CI is slightly shifted right for the bootstrapped estimate. This difference is negligible.

The histogram below contains the bootstrapped estimates of the Sharpe Ratio with the approximate CI (solid black lines) and the bootstrapped CI (dashed red line) over-layed. We see the that the bootstrapped CI is not that different than the approximate CI.



Now let's look at VaR and ES!

B. Value at Risk and Expected Shortfall

VaR and ES are in depth topics that I can not explore in this article, so I am only giving a basic overview.

VaR(α) is a measure of risk of an asset or portfolio. If VaR(0.05) is $10,000, there is a 5% probability that we see a loss of *at least* $10,000 in a given time period. You can assume a distribution of returns and estimate VaR based on that probabilistic model, or you can use the sample quantile if you have sufficient data. You really want to have enough data points to fully cover the left-tail of the returns distribution if taking this approach, which is the one I am taking. Choosing too small of an α relative to your sample size can lead to large underestimates of risk.

This non-parametric VaR formula is below:

$$V\hat{a}R(\alpha) = -S * \hat{q}(\alpha)$$

where S is the holdings in the asset or portfolio and $\hat{q}(\alpha)$ is the sample quantile at α.

ES is related to VaR. It is the expected return conditional that it is in the worst α% of cases.

This non-parametric ES formula is below:

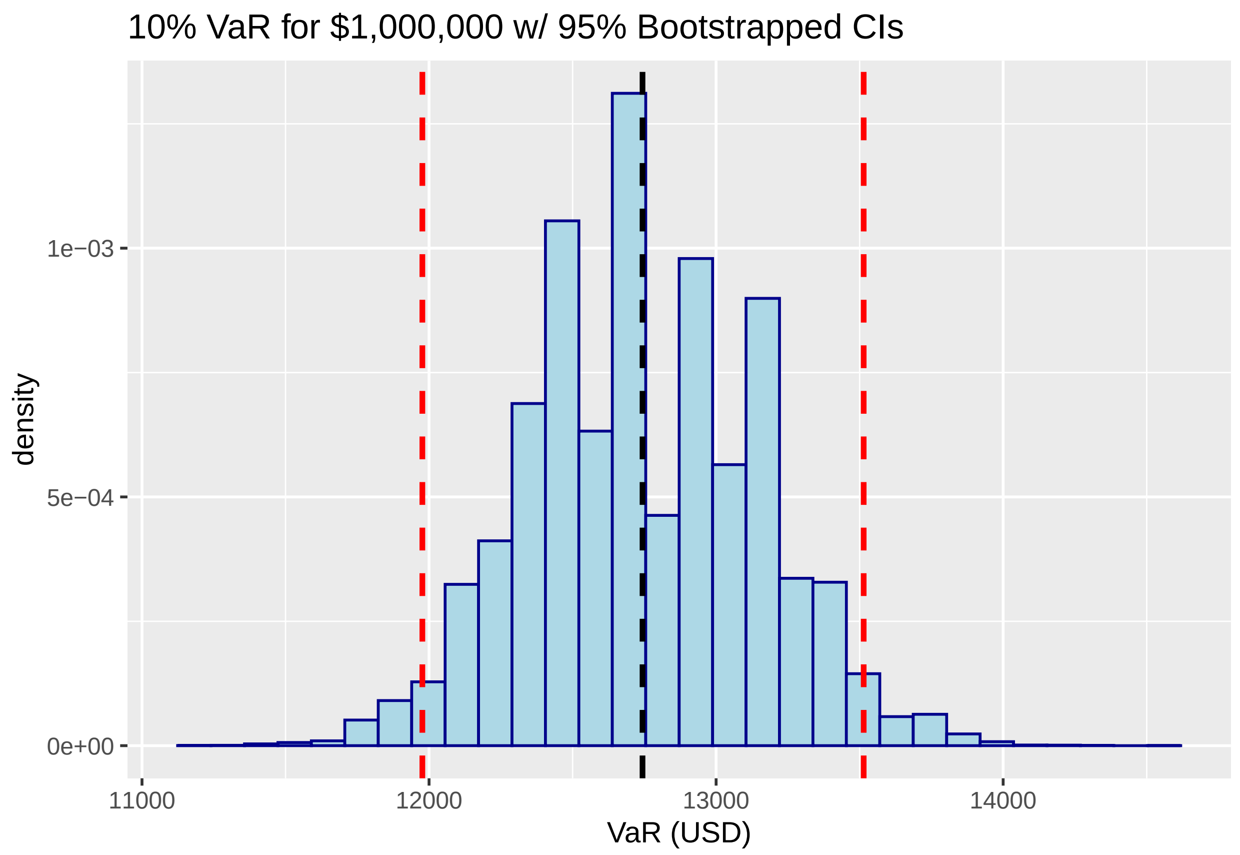
$$\hat{ES}(\alpha) = -S * \frac{\Sigma_{i=1}^{n} R_i * I(R_i < \hat{q}(\alpha))}{\Sigma_{i=1}^{n} I(R_i < \hat{q}(\alpha))}$$

where S is the holdings in the asset or portfolio, R_i are returns, $\hat{q}(\alpha)$ is the sample quantile at α, and I is an indicator function that is 1 if true and 0 if false.

With that background out of the way, let's apply bootstapping to VaR and ES. I will look at the 10% VaR and ES on a $1,000,000 portfolio indexed to the S&P 500 for our period interest. Since I am using daily returns, these risk measures apply to a time period of a day.

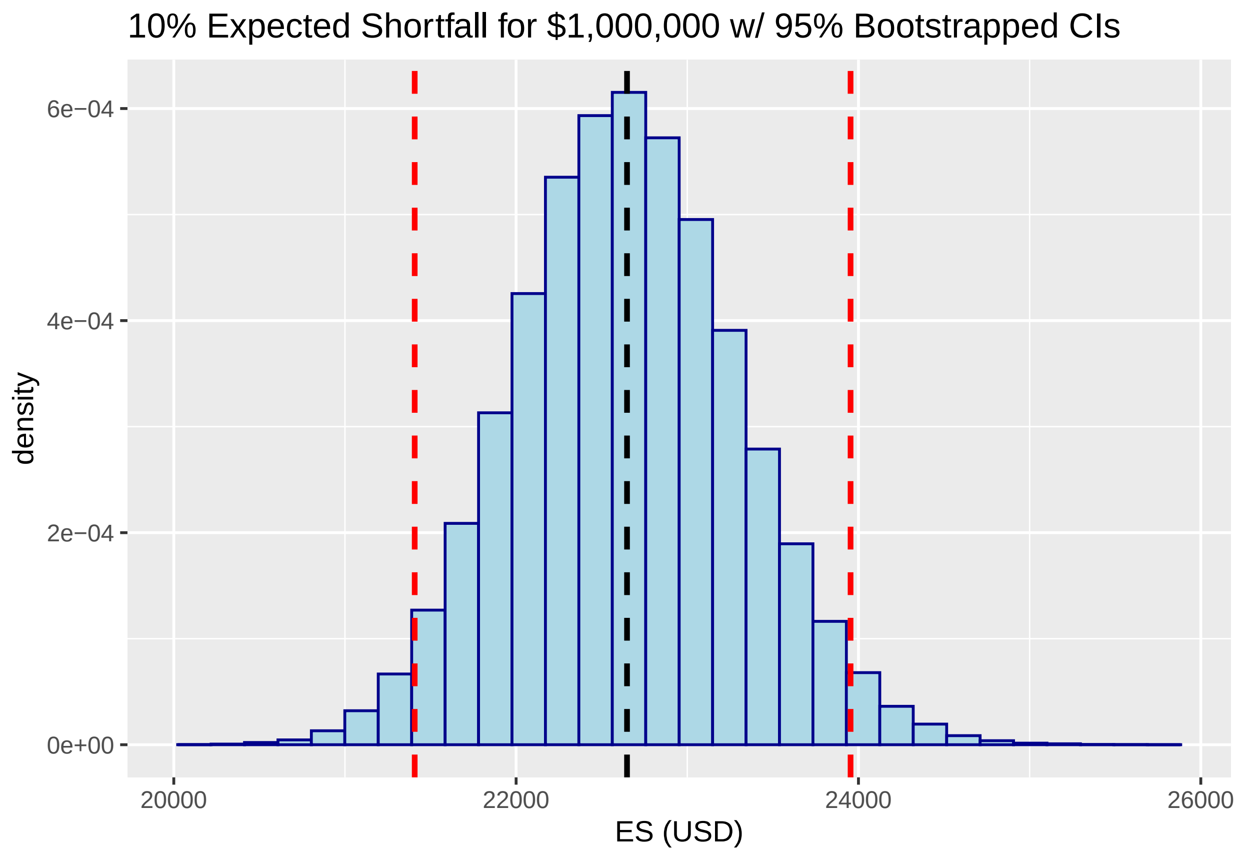
```r
non_parametric_var <- function(returns, prob=0.1, holdings=1000000){
  return(as.numeric(-holdings*quantile(returns, probs=prob)))
}

non_parametric_es <- function(returns, prob=0.1, holdings=1000000){
  below_quantile <- returns < quantile(returns, probs=prob)
  numer <- sum(returns[below_quantile])
  denom <- sum(below_quantile)
  return(-holdings * (numer / denom))
}

var_est <- bootstrap_parallel(returns, non_parametric_var, 100000)
es_est <- bootstrap_parallel(returns, non_parametric_es, 100000)
```

The distribution of bootstrapped estimates of VaR(0.1) on our portfolio are below. Our mean estimate of VaR(0.1) is 1.27436×10^4 with a standard error of 404.17. The bootstrapped 95% CI is 1.1977×10^4 to 1.3514×10^4.



The histogram is a little lumpy, so perhaps there is not enough data to fill out the left-tail sufficiently. Let's look at ES.

The distribution of bootstrapped estimates of ES(0.1) on our portfolio are below. Our mean estimate of ES(0.1) is 2.26478×10^4 with a standard error of 649.18. The bootstrapped 95% CI is 2.1407×10^4 to 2.3954×10^4.



Our simulated sampling distribution of ES has a nice approximate bell-curve with a bit of a skew towards the right.

I hope this has clarified what bootstrapping is and how to apply it. Thanks for reading!