Our ultimate goal is to build a Shiny app that allows an end user to build a custom portfolio, simulate returns, and visualize the results.

Monte Carlo Shiny App Code

```{r setup, message = FALSE}

library(tidyverse)

library(highcharter)

library(tidyquant)

library(timetk)

library(scales)

```

Sidebar {.sidebar}

=====================================

```{r}

fluidRow(

column(6,

textInput("stock1", "Stock 1", "SPY")),

column(5,

numericInput("w1", "Portf. %", 25, min = 1, max = 100))

)

fluidRow(

column(6,

textInput("stock2", "Stock 2", "EFA")),

column(5,

numericInput("w2", "Portf. %", 25, min = 1, max = 100))

)

fluidRow(

column(6,

textInput("stock3", "Stock 3", "IJS")),

column(5,

numericInput("w3", "Portf. %", 20, min = 1, max = 100))

)

fluidRow(

column(6,

textInput("stock4", "Stock 4", "EEM")),

column(5,

numericInput("w4", "Portf. %", 20, min = 1, max = 100))

)

fluidRow(

column(6,

textInput("stock5", "Stock 5", "AGG")),

column(5,

numericInput("w5", "Portf. %", 10, min = 1, max = 100))

)

fluidRow(

column(7,

dateInput("date", "Starting Date", "2013-01-01", format = "yyyy-mm-dd"))

)

fluidRow(

column(5,

numericInput("sim\_months", "Months", 120, min = 6, max = 240, step = 6)),

column(5,

numericInput("sims", "Sims", 51, min = 31, max = 101, step = 10))

)

actionButton("go", "Submit")

```

```{r}

prices <- eventReactive(input$go, {

symbols <- c(input$stock1, input$stock2, input$stock3, input$stock4, input$stock5)

getSymbols(symbols, src = 'yahoo', from = input$date,

auto.assign = TRUE, warnings = FALSE) %>%

map(~Ad(get(.))) %>%

reduce(merge) %>%

`colnames<-`(symbols)

})

portfolio\_returns\_tq\_rebalanced\_monthly <- eventReactive(input$go, {

prices <- prices()

w <- c(input$w1/100, input$w2/100, input$w3/100, input$w4/100, input$w5/100)

portfolio\_returns\_tq\_rebalanced\_monthly <-

prices %>%

to.monthly(indexAt = "last", OHLC = FALSE) %>%

tk\_tbl(preserve\_index = TRUE, rename\_index = "date") %>%

gather(asset, returns, -date) %>%

group\_by(asset) %>%

mutate(returns = (log(returns) - log(lag(returns)))) %>%

tq\_portfolio(assets\_col = asset,

returns\_col = returns,

weights = w,

col\_rename = "returns",

rebalance\_on = "months")

})

mean\_port\_return <- eventReactive(input$go, {

portfolio\_returns\_tq\_rebalanced\_monthly <- portfolio\_returns\_tq\_rebalanced\_monthly()

mean(portfolio\_returns\_tq\_rebalanced\_monthly$returns)

})

stddev\_port\_return <- eventReactive(input$go, {

portfolio\_returns\_tq\_rebalanced\_monthly <- portfolio\_returns\_tq\_rebalanced\_monthly()

sd(portfolio\_returns\_tq\_rebalanced\_monthly$returns)

})

simulation\_accum\_1 <- function(init\_value, N, mean, stdev) {

tibble(c(init\_value, 1 + rnorm(N, mean, stdev))) %>%

`colnames<-`("returns") %>%

mutate(growth = accumulate(returns, function(x, y) x \* y)) %>%

select(growth)

}

sims <- eventReactive(input$go, {input$sims})

monte\_carlo\_sim\_51 <- eventReactive(input$go, {

sims <- sims()

starts <-

rep(1, sims) %>%

set\_names(paste("sim", 1:sims, sep = ""))

map\_dfc(starts, simulation\_accum\_1,

N = input$sim\_months, mean = mean\_port\_return(),

stdev = stddev\_port\_return()) %>%

mutate(month = seq(1:nrow(.))) %>%

select(month, everything()) %>%

`colnames<-`(c("month", names(starts))) %>%

gather(sim, growth, -month) %>%

group\_by(sim) %>%

mutate\_all(funs(round(., 2)))

})

```

Simulated Growth

=====================================

Row {data-height=600, .tabset}

-----------------------------------------------------------------------

### Sim Vis highcharter

```{r}

renderHighchart(

hchart( monte\_carlo\_sim\_51(),

type = 'line',

hcaes(y = growth,

x = month,

group = sim)) %>%

hc\_title(text = paste(sims(), "Simulations", sep = " ")) %>%

hc\_xAxis(title = list(text = "months")) %>%

hc\_yAxis(title = list(text = "dollar growth"),

labels = list(format = "${value}")) %>%

hc\_add\_theme(hc\_theme\_flat()) %>%

hc\_exporting(enabled = TRUE) %>%

hc\_legend(enabled = FALSE)

)

```

Row

-------------------------------------

### Min, Max, Median Sim

```{r}

renderHighchart({

sim\_summary <-

monte\_carlo\_sim\_51() %>%

summarise(final = last(growth)) %>%

summarise(

max = max(final),

min = min(final),

median = median(final))

mc\_max\_med\_min <-

monte\_carlo\_sim\_51() %>%

filter(

last(growth) == sim\_summary$max ||

last(growth) == sim\_summary$median ||

last(growth) == sim\_summary$min)

hchart(mc\_max\_med\_min,

type = 'line',

hcaes(y = growth,

x = month,

group = sim)) %>%

hc\_title(text = "Min Max Median Simulations") %>%

hc\_xAxis(title = list(text = "months")) %>%

hc\_yAxis(title = list(text = "dollar growth"),

labels = list(format = "${value}")) %>%

hc\_add\_theme(hc\_theme\_flat()) %>%

hc\_exporting(enabled = TRUE) %>%

hc\_legend(enabled = FALSE)

})

```

This post builds off the work we did previously. I won’t go through the logic again, but the code for building a portfolio, calculating returns, mean and standard deviation of returns, and using them for a simulation is here:

library(tidyquant)

library(tidyverse)

library(timetk)

library(broom)

library(highcharter)

symbols <- c("SPY","EFA", "IJS", "EEM","AGG")

prices <-

getSymbols(symbols, src = 'yahoo',

from = "2012-12-31",

to = "2017-12-31",

auto.assign = TRUE, warnings = FALSE) %>%

map(~Ad(get(.))) %>%

reduce(merge) %>%

`colnames<-`(symbols)

w <- c(0.25, 0.25, 0.20, 0.20, 0.10)

asset\_returns\_long <-

prices %>%

to.monthly(indexAt = "lastof", OHLC = FALSE) %>%

tk\_tbl(preserve\_index = TRUE, rename\_index = "date") %>%

gather(asset, returns, -date) %>%

group\_by(asset) %>%

mutate(returns = (log(returns) - log(lag(returns)))) %>%

na.omit()

portfolio\_returns\_tq\_rebalanced\_monthly <-

asset\_returns\_long %>%

tq\_portfolio(assets\_col = asset,

returns\_col = returns,

weights = w,

col\_rename = "returns",

rebalance\_on = "months")

mean\_port\_return <-

mean(portfolio\_returns\_tq\_rebalanced\_monthly$returns)

stddev\_port\_return <-

sd(portfolio\_returns\_tq\_rebalanced\_monthly$returns)

simulation\_accum\_1 <- function(init\_value, N, mean, stdev) {

tibble(c(init\_value, 1 + rnorm(N, mean, stdev))) %>%

`colnames<-`("returns") %>%

mutate(growth =

accumulate(returns,

function(x, y) x \* y)) %>%

select(growth)

}

That code allows us to run one simulation of the growth of a dollar over the next 10 years, with the simulation\_accum\_1() that we built for that purpose. Today, we will review how to run 51 simulations, though we could choose any number (and our Shiny application allows an end user to do just that).

First, we need an empty matrix with 51 columns, an initial value of $1, and intuitive column names.

We will use the rep() function to create 51 columns with a 1 as the value, and set\_names() to name each column with the appropriate simulation number.

sims <- 51

starts <-

rep(1, sims) %>%

set\_names(paste("sim", 1:sims, sep = ""))

Take a peek at starts to see what we just created, and how it can house our simulations.

head(starts)

sim1 sim2 sim3 sim4 sim5 sim6

1 1 1 1 1 1

tail(starts)

sim46 sim47 sim48 sim49 sim50 sim51

1 1 1 1 1 1

51 columns, with a value of 1 in one row. This is where we will store the results of the 51 simulations.

Now, we want to apply simulation\_accum\_1 to each of the 51 columns of the starts matrix, and we will do that using the map\_dfc() function from the purrr package.

map\_dfc() takes a vector – in this case, the columns of starts – and applies a function to it. By appending dfc() to the map\_ function, we are asking the function to store each of its results as the column of a data frame (map\_df() does the same thing, but stores results in the rows of a data frame). After running the code below, we will have a data frame with 51 columns, one for each of our simulations.

We also need to choose how many months to simulate (the N argument to our simulation function) and supply the distribution parameters as we did before. We do not supply the init\_value argument because the init\_value is 1, that same 1 that is already in the 51 columns.

monte\_carlo\_sim\_51 <-

map\_dfc(starts,

simulation\_accum\_1,

N = 120,

mean = mean\_port\_return,

stdev = stddev\_port\_return)

tail(monte\_carlo\_sim\_51 %>% select(growth1, growth2,

growth49, growth50), 3)

# A tibble: 3 x 4

growth1 growth2 growth49 growth50

1 1.81 1.38 2.32 1.80

2 1.84 1.37 2.38 1.84

3 1.82 1.33 2.39 1.82

Have a look at the results. We now have 51 simulations of the growth of a dollar, and we simulated that growth over 120 months; however, the results are missing a piece that we need for visualization, namely a month column.

Let’s add that month column with mutate() and give it the same number of rows as our data frame. These are months out into the future. We will use mutate(month = seq(1:nrow(.))), and then clean up the column names. nrow() is equal to the number of rows in our object. If we were to change to 130 simulations, that would generate 130 rows, and nrow() would be equal to 130.

monte\_carlo\_sim\_51 <-

monte\_carlo\_sim\_51 %>%

mutate(month = seq(1:nrow(.))) %>%

select(month, everything()) %>%

`colnames<-`(c("month", names(starts))) %>%

mutate\_all(funs(round(., 2)))

tail(monte\_carlo\_sim\_51 %>% select(month, sim1, sim2,

sim49, sim50), 3)

# A tibble: 3 x 5

month sim1 sim2 sim49 sim50

1 119 2.16 1.81 1.46 2.32

2 120 2.28 1.84 1.46 2.38

3 121 2.26 1.82 1.46 2.39

We have accomplished our goal of running 51 simulations, and could head to data visualization now, but let’s explore an alternative method using the the rerun() function from purrr. As its name implies, this function will “rerun” another function, and we stipulate how many times to do that by setting .n = number of times to rerun. For example to run the simulation\_accum\_1 function 5 times, we would set the following:

monte\_carlo\_rerun\_5 <-

rerun(.n = 5,

simulation\_accum\_1(1,

120,

mean\_port\_return,

stddev\_port\_return))

That returned a list of 5 data frames, or 5 simulations. We can look at the first few rows of each data frame by using map(..., head).

map(monte\_carlo\_rerun\_5, head)

[[1]]

# A tibble: 6 x 1

growth

1 1

2 0.983

3 0.965

4 0.946

5 0.967

6 0.962

[[2]]

# A tibble: 6 x 1

growth

1 1

2 0.980

3 0.975

4 0.964

5 0.969

6 0.914

[[3]]

# A tibble: 6 x 1

growth

1 1

2 1.03

3 0.997

4 0.979

5 1.04

6 1.04

[[4]]

# A tibble: 6 x 1

growth

1 1

2 0.974

3 0.962

4 0.943

5 0.942

6 0.963

[[5]]

# A tibble: 6 x 1

growth

1 1

2 0.990

3 1.02

4 1.06

5 1.12

6 1.13

Let’s consolidate that list of data frames to one tibble. We start by collapsing to vectors with simplify\_all(), then add nicer names with the names() function and finally coerce to tibble with as\_tibble(). Let’s run it 51 times to match our previous results.

reruns <- 51

monte\_carlo\_rerun\_51 <-

rerun(.n = reruns,

simulation\_accum\_1(1,

120,

mean\_port\_return,

stddev\_port\_return)) %>%

simplify\_all() %>%

`names<-`(paste("sim", 1:reruns, sep = " ")) %>%

as\_tibble() %>%

mutate(month = seq(1:nrow(.))) %>%

select(month, everything())

tail(monte\_carlo\_rerun\_51 %>% select(`sim 1`, `sim 2`,

`sim 49`, `sim 50`), 3)

# A tibble: 3 x 4

`sim 1` `sim 2` `sim 49` `sim 50`

1 1.99 1.97 3.66 2.20

2 2.00 1.86 3.68 2.25

3 2.02 1.95 3.78 2.18

Now we have two objects holding the results of 51 simulations, monte\_carlo\_rerun\_51 and monte\_carlo\_sim\_51.

Each has 51 columns of simulations and 1 column of months. Note that we have 121 rows because we started with an initial value of 1, and then simulated returns over 120 months.

Now let’s get to visualization with ggplot() and visualize the results in monte\_carlo\_sim\_51. The same code flows for visualization would also apply to monte\_carlo\_rerun\_51, but we will run them for only monte\_carlo\_sim\_51 here.

We start with a chart of all 51 simulations, and assign a different color to each one by setting ggplot(aes(x = month, y = growth, color = sim)). ggplot() will automatically generate a legend for all 51 time series, but that gets quite crowded. We will suppress the legend with theme(legend.position = "none").

monte\_carlo\_sim\_51 %>%

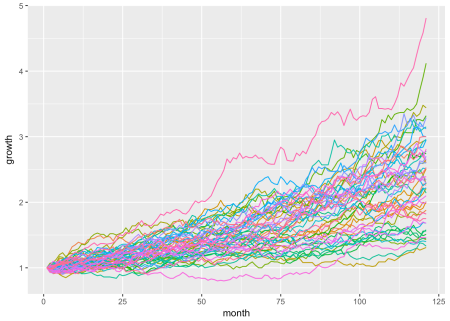
gather(sim, growth, -month) %>%

group\_by(sim) %>%

ggplot(aes(x = month, y = growth, color = sim)) +

geom\_line() +

theme(legend.position="none")



We can check the minimum, maximum and median simulation with the summarise() function here.

sim\_summary <-

monte\_carlo\_sim\_51 %>%

gather(sim, growth, -month) %>%

group\_by(sim) %>%

summarise(final = last(growth)) %>%

summarise(

max = max(final),

min = min(final),

median = median(final))

sim\_summary

# A tibble: 1 x 3

max min median

1 4.81 1.31 2.3

We can clean up our original visualization by including only the max, min and median that were just calculated.

monte\_carlo\_sim\_51 %>%

gather(sim, growth, -month) %>%

group\_by(sim) %>%

filter(

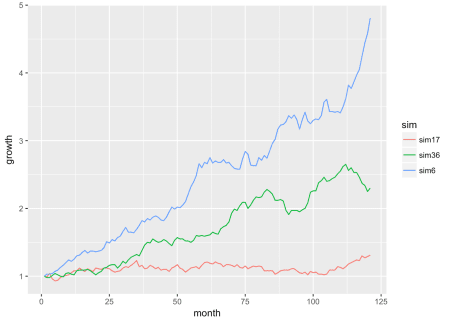
last(growth) == sim\_summary$max ||

last(growth) == sim\_summary$median ||

last(growth) == sim\_summary$min) %>%

ggplot(aes(x = month, y = growth)) +

geom\_line(aes(color = sim))



Now let’s port our results over to highcharter, but in a major departure from our usual code flow, we will pass a tidy tibble instead of an xts object.

Our first step is to convert the data from wide to long tidy format with the gather() function.

mc\_gathered <-

monte\_carlo\_sim\_51 %>%

gather(sim, growth, -month) %>%

group\_by(sim)

head(mc\_gathered)

# A tibble: 6 x 3

# Groups: sim [1]

month sim growth

1 1 sim1 1

2 2 sim1 0.99

3 3 sim1 1.01

4 4 sim1 1.06

5 5 sim1 1.08

6 6 sim1 1.1

We can now pass this tibble directly to the hchart() function, specify the type of chart as line, and then work with a similar grammar to ggplot(). The difference is we use hcaes, which stands for highcharter aesthetic, instead of aes.

# This takes a few seconds to run

hchart(mc\_gathered,

type = 'line',

hcaes(y = growth,

x = month,

group = sim)) %>%

hc\_title(text = "51 Simulations") %>%

hc\_xAxis(title = list(text = "months")) %>%

hc\_yAxis(title = list(text = "dollar growth"),

labels = list(format = "${value}")) %>%

hc\_add\_theme(hc\_theme\_flat()) %>%

hc\_exporting(enabled = TRUE) %>%

hc\_legend(enabled = FALSE)

We just plotted 51 lines in highcharter using a tidy tibble. For tidy data fans out there, this is a big deal because we can stay in the grammar of the tidyverse but also use highcharter.

Very similar to what we did with ggplot, let’s isolate the maximum, minimum and median simulations and save them to an object called mc\_max\_med\_min.

mc\_max\_med\_min <-

mc\_gathered %>%

filter(

last(growth) == sim\_summary$max ||

last(growth) == sim\_summary$median ||

last(growth) == sim\_summary$min) %>%

group\_by(sim)

Now we pass that filtered object to hchart().

hchart(mc\_max\_med\_min,

type = 'line',

hcaes(y = growth,

x = month,

group = sim)) %>%

hc\_title(text = "Min, Max, Median Simulations") %>%

hc\_xAxis(title = list(text = "months")) %>%

hc\_yAxis(title = list(text = "dollar growth"),

labels = list(format = "${value}")) %>%

hc\_add\_theme(hc\_theme\_flat()) %>%

hc\_exporting(enabled = TRUE) %>%

hc\_legend(enabled = FALSE)

That concludes our visualization of Monte Carlo simulations.