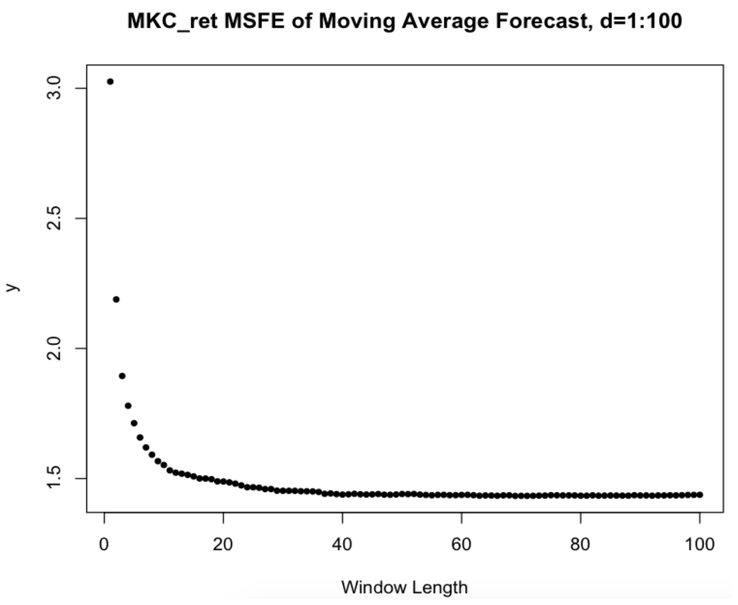
Marcus Crowder

CIS- 3920 Data-Mining

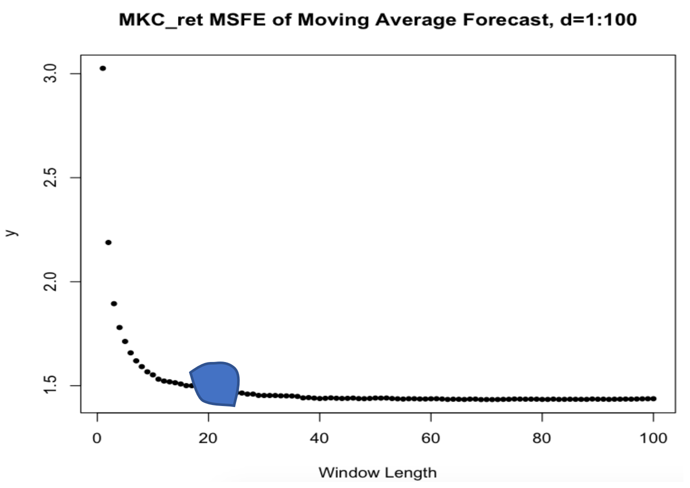
Learning Exercise 3

Q1. Using MSFE and VSFE R functions to plot data of the MKC stock.



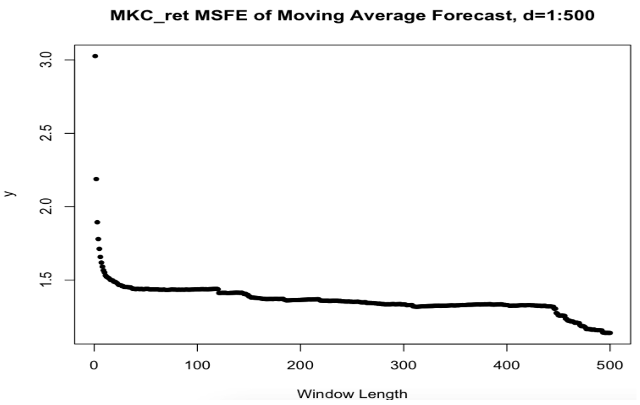
A. The graph on the right represents the MSFE of the MKC stock with Moving averages from 1 to 100. Notice as we get a larger window a lower error is achieved which also means a better forecast.

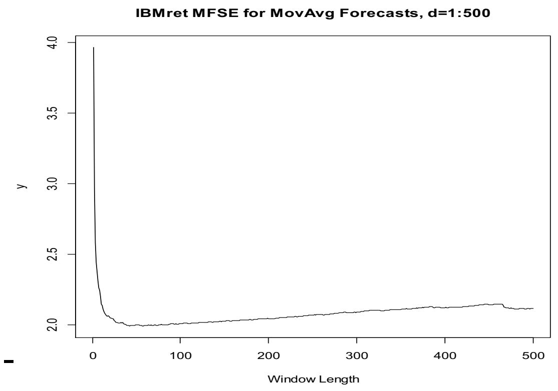
B. I choose a position around point 20. Or to be more precise I choose the point with the window length of 20 which has a MSFE or 1.5. It basically states that when we take the average of 20 previous points we end up with a standard MSFE or 1.5.



C. I would suggest using a window length of 100 just from looking at the plotting of the graph. The higher window length seems to produce a lower Squared Error. This is without accounting for outside factors and the Volatility of the stock market.

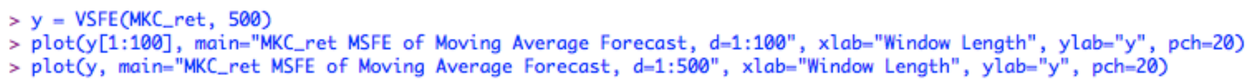
D. With my Plot Above that of Professor Tatum's there are a few differences. One difference is the stocks MKC vs IBM. Another big difference is the line the stocks make as MKC uses a larger window the MSFE continually decreases. On the other hand as the IBM stock uses a larger window the MSFE goes up after a 35-40 window.





E. Because of Momentum should the MSFE for a window of length 1 be smaller or larger than for a window of length 10? Why? No even if momentum exists I would not recommend using a window of size 1 because of the volatility of the stock market. And just by looking at the graphs we can see a MSFE of window length 1 is a lot higher than that of MSFE 10.

R Commands used for Creating VSFE of return and plotting both graphs

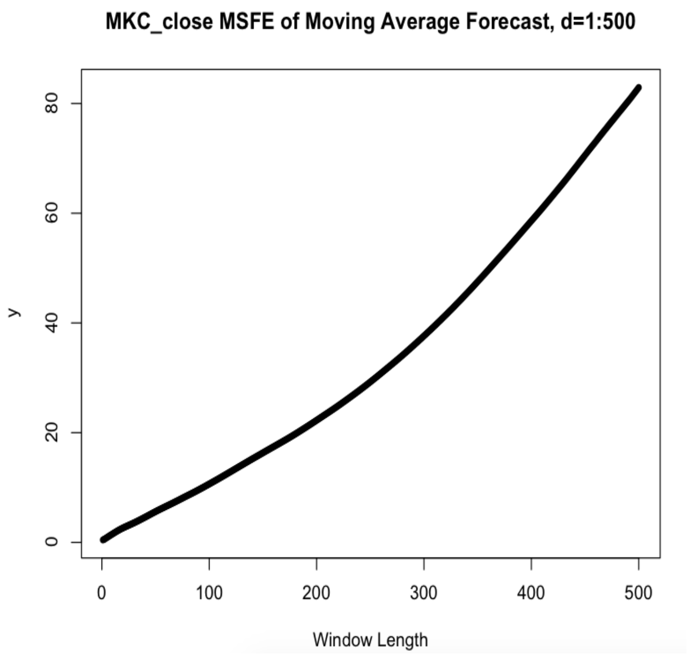


```
> y = VSFE(MKC_ret, 500)
> plot(y[1:100], main="MKC_ret MSFE of Moving Average Forecast, d=1:100", xlab="Window Length", ylab="y", pch=20)
> plot(y, main="MKC_ret MSFE of Moving Average Forecast, d=1:500", xlab="Window Length", ylab="y", pch=20)
```

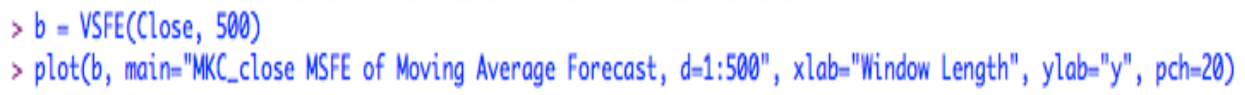
Q2.VSFE of Closing Prices for MCK Stock.

Plotting the MSFE of closing prices was a lot different than plotting returns.

The closing prices continued to rise over time. Which is great for this company but horrible for our MSFE forecast. This can be due to Inflation and the rise of stock prices in general over the years but the line is unexpected in a good way. It teaches us to not always depend on the MSFE or a large window.



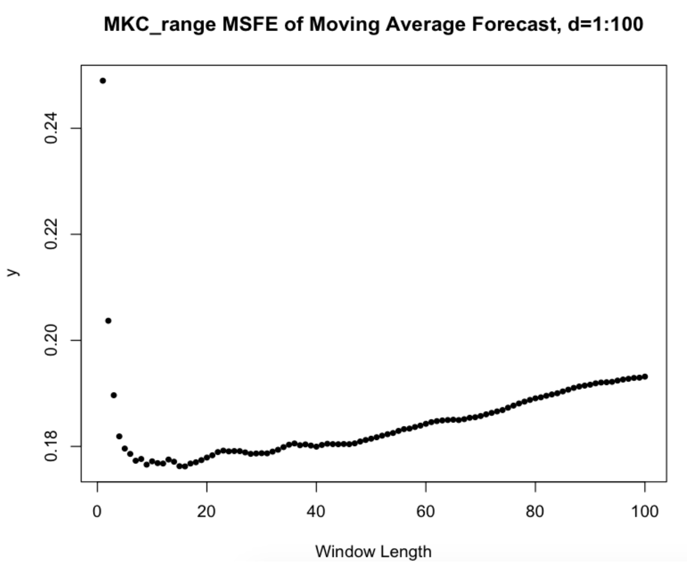
R commands used to create VSFE



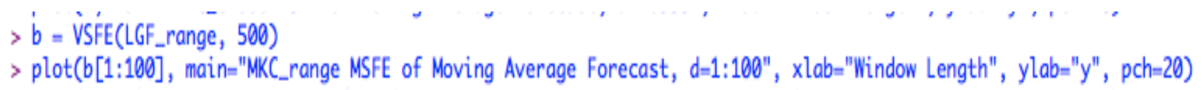
```
> b = VSFE(Close, 500)
> plot(b, main="MKC_close MSFE of Moving Average Forecast, d=1:500", xlab="Window Length", ylab="y", pch=20)
```

Q3.The Daily Range of the MKC stock with MSFE graph

The daily range which was given by subtracting the high of the day minus the low produced a lower than ever MSFE. The best choice for a forecasting range seems to be around the 18-19 window. This is the case because it has the lowest forecasting error before eventually rising again without sight of stoppage.



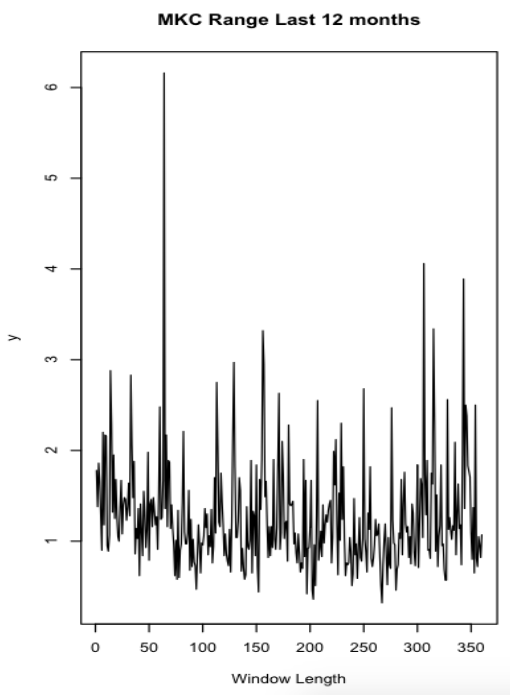
R commands for Range plot

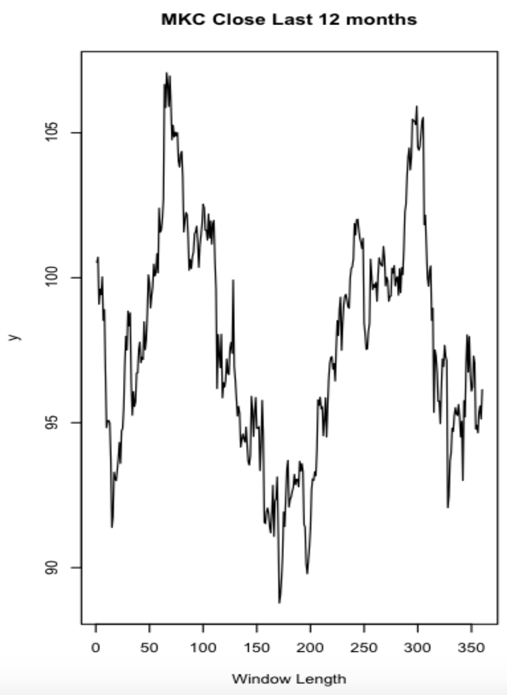


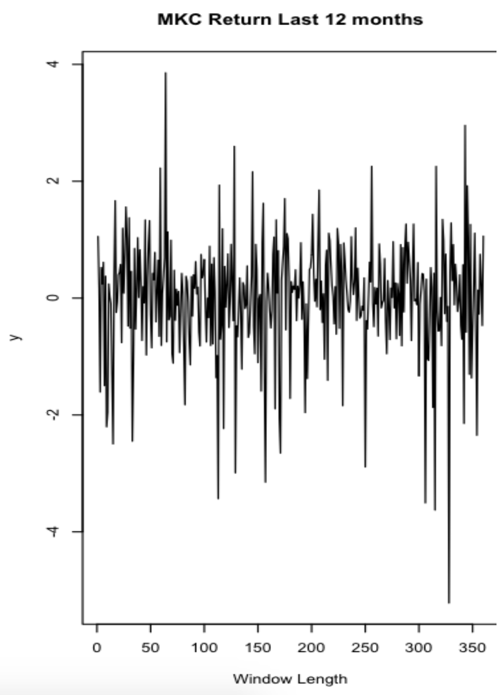
```
> b = VSFE(LGF_range, 500)
> plot(b[1:100], main="MKC_range MSFE of Moving Average Forecast, d=1:100", xlab="Window Length", ylab="y", pch=20)
```

4. We are working with three different types of time series data in this section. First, we can have time series with independent values, where knowing what happened at time t-1 and at time t+1 tells you nothing what happened at time t. Second, we have time series containing a random walk, where knowing what happened at time t-1 is our best forecast of what will happen at time t. Third, we can have time series with positive autocorrelation, like the high temperature series on page 3. There, we see that the temperature for days which are near in time will tend to be similar in value, while when the days are far apart in time, then the difference in the temperatures will tend to be larger, too. As a consequence, data with positive autocorrelation will tend to wander away from its underlying average and then gradually return to its average, only to wander off again.

   Examine times series plots for the last 12 months of (a) your daily range, (b) your daily close, and (c) your daily stock return. Which time series type does each belong to? What is your reasoning?







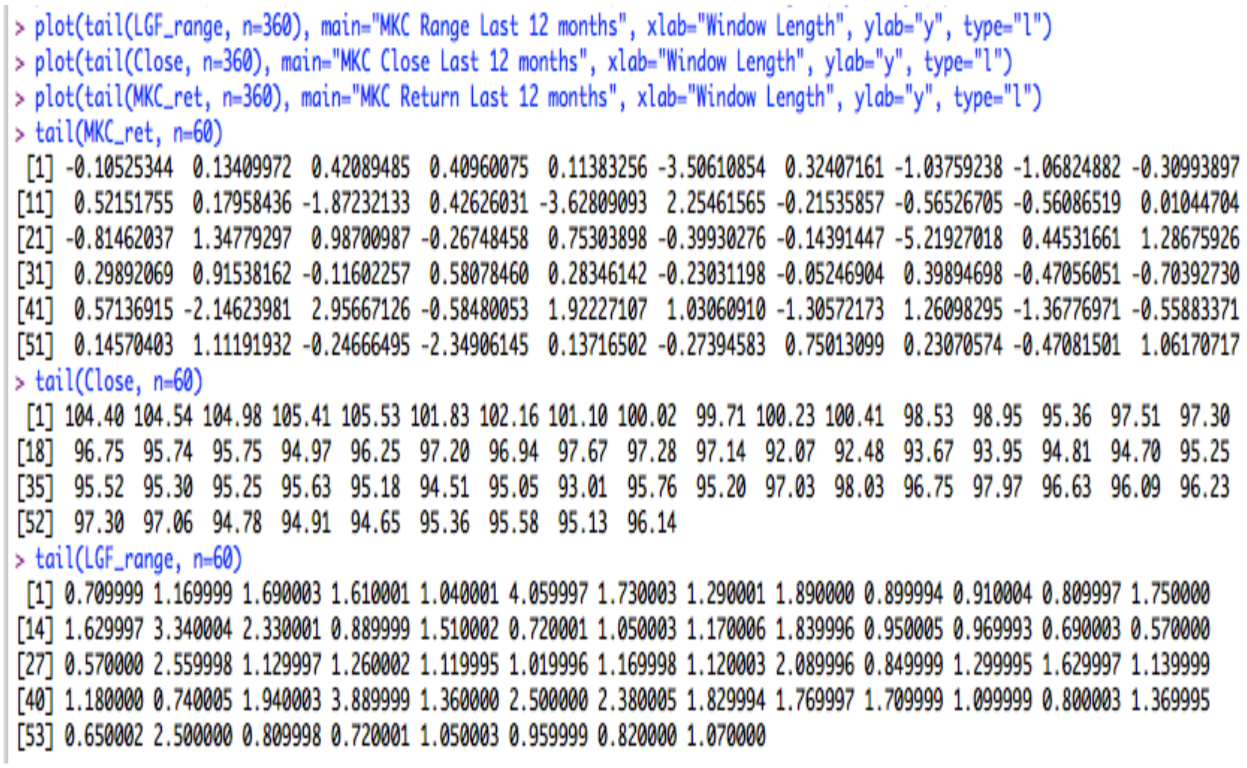
Plot1: The first plot is of the MKC range over the last 12 months which looks like a time series data that gives no real information. The values seem independent on a day to day basis. This is because stocks rise and fall throughout the day before ending at a certain closing price. The rise and fall of stocks happen for many reasons such as news reports. By knowing the range of one day's stock price it would be hard to predict the range of another day's stock price because no one knows what news will happen or the impact it has on high volume trading.

Plot2: The Closing prices show a time series of positive autocorrelation. The Ckise fluctuates but seem to follow an average. It also eventually wanders off.

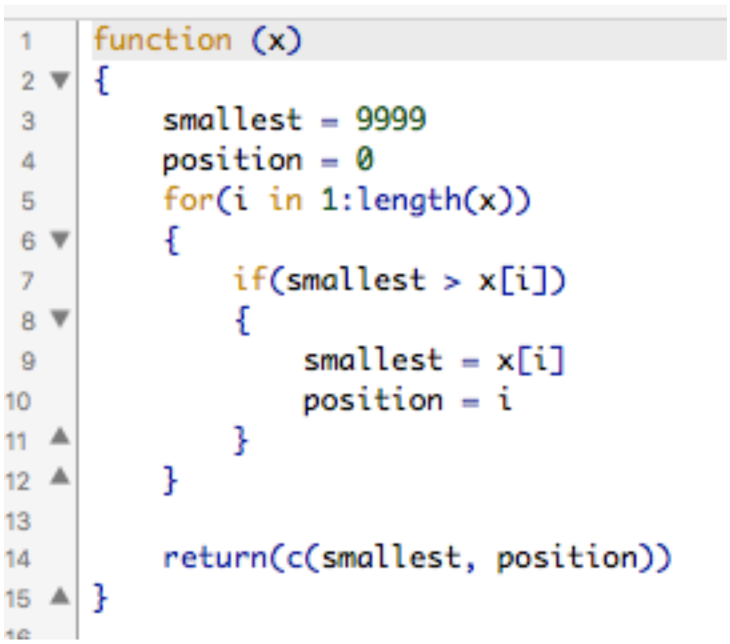
Plot3: The Return plot seems to indicate a random walk by knowing what happened at time t-1 is our best shot at predicting what will happen at time t. Just looking at the graph it is evident that a moving average would not be very effective on this graph (similar to MSFE Close price above). The lowest error would come from looking at the value or forecast of the day before. The stock is too volatile for a moving average.

Plot3: The Return prices show a time series of positive autocorrelation. The return goes fluctuates but seem to follow an average. It also eventually wanders off.

R commands used to produce Plots



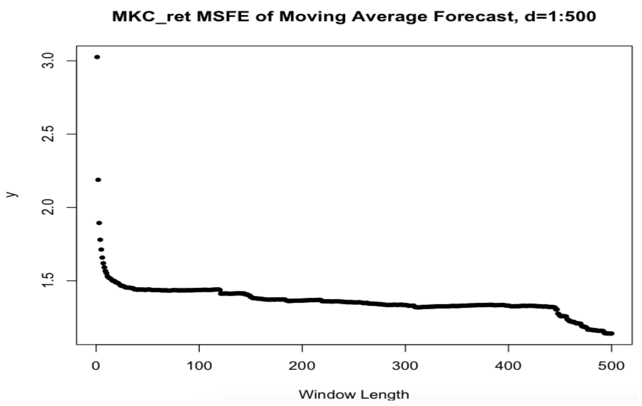
```
> plot(tail(LGF_range, n=360), main="MKC Range Last 12 months", xlab="Window Length", ylab="y", type="l")
> plot(tail(Close, n=360), main="MKC Close Last 12 months", xlab="Window Length", ylab="y", type="l")
> plot(tail(MKC_ret, n=360), main="MKC Return Last 12 months", xlab="Window Length", ylab="y", type="l")
> tail(MKC_ret, n=60)
 [1] -0.10525344  0.13409972  0.42089485  0.40960075  0.11383256 -3.50610854  0.32407161 -1.03759238 -1.06824882 -0.30993897
[11]  0.52151755  0.17958436 -1.87232133  0.42626031 -3.62809093  2.25461565 -0.21535857 -0.56526705 -0.56086519  0.01044704
[21] -0.81462037  1.34779297  0.98700987 -0.26748458  0.75303898 -0.39930276 -0.14391447 -5.21927018  0.44531661  1.28675926
[31]  0.29892069  0.91538162 -0.11602257  0.58078460  0.28346142 -0.23031198 -0.05246904  0.39894698 -0.47056051 -0.70392730
[41]  0.57136915 -2.14623981  2.95667126 -0.58480053  1.92227107  1.03060910 -1.30572173  1.26098295 -1.36776971 -0.55883371
[51]  0.14570403  1.11191932 -0.24666495 -2.34906145  0.13716502 -0.27394583  0.75013099  0.23070574 -0.47081501  1.06170717
> tail(Close, n=60)
 [1] 104.40 104.54 104.98 105.41 105.53 101.83 102.16 101.10 100.02  99.71 100.23 100.41  98.53  98.95  95.36  97.51  97.30
[18]  96.75  95.74  95.75  94.97  96.25  97.20  96.94  97.67  97.28  97.14  92.07  92.48  93.67  93.95  94.81  94.70  95.25
[35]  95.52  95.30  95.25  95.63  95.18  94.51  95.05  93.01  95.76  95.20  97.03  98.03  96.75  97.97  96.63  96.09  96.23
[52]  97.30  97.06  94.78  94.91  94.65  95.36  95.58  95.13  96.14
> tail(LGF_range, n=60)
 [1] 0.709999 1.169999 1.690003 1.610001 1.040001 4.059997 1.730003 1.290001 1.890000 0.899994 0.910004 0.809997 1.750000
[14] 1.629997 3.340004 2.330001 0.889999 1.510002 0.720001 1.050003 1.170006 1.839996 0.950005 0.969993 0.690003 0.570000
[27] 0.570000 2.559998 1.129997 1.260002 1.119995 1.019996 1.169998 1.120003 2.089996 0.849999 1.299995 1.629997 1.139999
[40] 1.180000 0.740005 1.940003 3.889999 1.360000 2.500000 2.380005 1.829994 1.769997 1.709999 1.099999 0.800003 1.369995
[53] 0.650002 2.500000 0.809998 0.720001 1.050003 0.959999 0.820000 1.070000
```



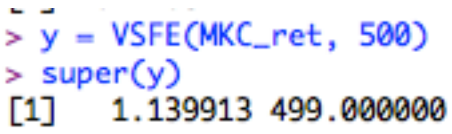
```
function (x)
{
    smallest = 9999
    position = 0
    for(i in 1:length(x))
    {
        if(smallest > x[i])
        {
            smallest = x[i]
            position = i
        }
    }

    return(c(smallest, position))
}
```

5. I created my own R function with a loop named super. Because it is super powerful saves time instead of having to look at the graph to find where the exact placement of the best moving average is. The function sets Smallest to 9999 so it would not be lower than any low MSFE values. Then a variable called position is created to find the position of the lowest MSFE. The loop goes from 1 to the length of the vector placed into it. After looping through the vector and finding the smallest MSFE the function returns smallest and the position in a vector. This function saves time when looking at a graph and trying to decide which moving average is optimal.



Here is an example of me using it to find the optimal moving average to forecast my Returns it happens to be 1.139913 at the 499 position which can also be seen by looking at the graph plot from question



```
> y = VSFE(MKC_ret, 500)
> super(y)
[1]   1.139913 499.000000
```