Final Project

Paul Pawelec & Jeremy Gorman

University of Guelph

FIN 4100

Alex Maynard

04/08/2022

Feedback and Comments

In the bolded and italic text is the feedback we received, and the bullet points below them will address each concern mentioned in the feedback.

Assignment One Feedback

Thank you for collecting, describing and analyzing your data.

It is good that you have 3 risky assets and your descriptions are detailed.

Here are a few suggestions going forward:

1. ***It is good that you have three assets. One of them is a broad based asset and two are companies. You want to discuss questions of balancing diversification versus bets on particular companies, as well as selection of companies ex-post versus ex-ante. I am not asking your to change your data, but rather to address these issues in your write up.***
2. ***You are missing a table showing means, standard deviations, sharp values, annualizations, etc. This is important. Please add it.***
3. ***Please write it up as if it were part of a paper (it will be) and not as answers to an assignment question.***
4. ***I think you should separate descriptions of the data itself from descriptions of r-variable names and functions. The latter can be included as an appendix.***

- There are now comments about the issue of balancing diversification and balancing bets underneath the 'Section One – Data' heading.
- A table showing mean, standard deviations, and Sharpe values has been added under the 'Asset Descriptive Measures' heading for section one. The table is noted as table 1.2.
- The issue concerning the write up format has been fixed, we now have section one.
- The suggestion for separating the r-variable names and functions into a separate table has been noted, but due to time constraints and priorities we ultimately decided not to use this idea for the report.

Assignment Two Feedback

Thank you for your work on Assignment 2! I hope the feedback below with useful in revising for the final paper.

1) The report includes all main results for the baseline assignment.

2) The report does not appear to go beyond the baseline assignment.

3) Thank you for taking the time to write up and interpret your results. This will save time for your final paper and counts towards your assignment mark.

4) Thank you for including table and figure titles and numbers. The formatting also looks good. Note that Table 4 has an empty entry.

5) Please clarify the .adj. Does it mean adjusted for dividends.

6) Generally, Sharpe does not vary by risk aversion for the same stock. Yours vary a little but not by much. Please double check and maybe re-interpret.

7) As a minor point, the investor would earn the net return not the log return focused on in the report.
8) Enbridge return does not increase going from 75% to 90% weight. Is it a typo or do you need go out another digit?

- Table 4 (now called table 2.4) has been fixed to include the missing entry.
- The '.adj' does mean "adjusted for dividends", this detail has been added to the beginning of section two.
- Your mention of our Sharpe ratio declining slightly has been investigated and there doesn't seem to be any explicit issue with the code. I assume the change is resulting from some sort of rounding. We also made a note of this under section two's heading 'Equal Weighted Portfolio' and changed our interpretation.
- Your mention of the investor earning the net return and not the log return has been noted. Due to time constraints, priorities, and the minor variance between returns and log returns, it doesn't seem like a good use of our time to fix and re-format everything.
- Enbridge not increasing from 75% to 90% was due to rounding and poor formatting. It has been fixed in table 2.1

Assignment Three Feedback

Thank you for timely submission!

Thank you for labelling your figures and tables and keeping them pretty clear.

Thank you also for including a write up.

Thanks for conducting some sensitivity analysis on the lag lengths (using 1,7,30, 60, 90). Ideally, you would loop through a range, but what you did is useful.

90 and maybe even 60 seems a bit high. What was the maximum lag determined by the formula and how did these compare?

Thanks also for using the AIC, assuming you couldn't use MAIC. Please include in your table the lag-lengths that AIC selected.

Thank you for including both the intercept-only and trend cases

In addition to the ADF test, you may also want to report the GLS-ADF test results.
I appreciate that you have done a close visual inspection of the price trend, noting their possible exponential nature and possible breaks.

The exponential trend that you note in the prices may be (partially?) captured by taking logs and then modeling the trend (linear trend in ln P). You might want to see how that works.

A "sidways" trend essentially means no apparent trend. You might restate it.

As a minor point for the equations: state them with general gamma and then below indicate the null and alternative hypothesis. (Rather than combining as you do)

Another minor point. Recall that a failure to reject unit root, is simply that. It means that you cannot rule out the unit, but does not prove the unit. Still, we normally treat the data as having a unit root in this case, so your treatment is correct. It is just how you word it.

- Your mention of lags 90 and 60 being too high has been noted. I had missed the formula when looking over the assignment instructions and had just randomly chosen lags that I thought would be interesting. The max lags have been adjusted and can be seen in table 3.1, and tables 3.2 and 3.3 have been updated to reflect this development.
- You mention adding AIC values to our tables (3.2 and 3.3). A note has been made under each table pertaining to the value of the AIC (it was always returning 1).
- You mention restating our analysis of a "sideways" trend. It has been restated under section 3 for plot 3.2.
- You mention re-organizing our equations. I am unsure of what you mean and thus cannot fulfill the request. You don't seem to indicate they are incorrect, so I assume it's okay, but not perfect. I can live with that.
- You mention our explanation for 'failure to reject the unit root' is off, this has been adjusted for under the 'Our Unit Test Results and Model Selection' heading for section 3.

Assignment Four Feedback

Thank you for your assignment submission and the work that you put into it!

I hope that the comments below will be helpful as you revise for your final paper submission.

Exhibit 4.4 does not show lag 1. (Not showing lag 0 is helpful, but we need to see lag 1)

It may be possible to label your exhibits more clearly (more human readable). For examples the exhibits do not make clear which show ACF/PACF for raw data and which show it for the model residual. I can only figure out from the text, but should see it easily in the exhibit heading.

Not quite sure how the max lags table from the unit root tests made it into assignment 4. I think that max lag formula is specific to unit root testing.

Is Table 4.2 showing Ljung-Box tests for the data or for the model residual? Ideally, you will show both. I see eventually, that you do show, but clearer table titles are needed.

Thank you for showing the AIC and BIC model selection in Table 4.3. That is clear. It may also be helpful to see the full matrix of AIC/BIC results just to get a sense of how strongly AIC/BIC objects to other model choices. That would be better as a separate table, since 4.3 is nice as it is.

Thank you for making some additional efforts to improve upon the ACF of the BIC selected model.

Thank you for modelling price and return. It may be interesting also to model RF and RV.

Overall, you have most results, but the writing and table organization can be substantially sharpened and clarified.

Code is well organized.

- Exhibit 4.4 does not show lag 1 because it's incredibly small, I could adjust the x scale, but then I won't be able to fit everything cleanly.
- You mention to label our exhibits more clearly. This has been adjusted for.
- The max lag table from assignment 4 has been changed now due to a previous error. The formula used in assignment 5 should be correct, I took it directly from the project tips document.
- You mention that table 4.2 title should be clearer. This has been adjusted for in table 4.2 and future tables.
- You mention modeling RF and RV. I thought it would be interesting as well. However, I get some sort of error when finding the optimal model and I'm unable to resolve it due to time constraints.

Assignment Five Notes

There are two items of note for assignment five. First, I believe there may be an issue with the standard error calculations for table 5.2. Some of the numbers reported back as '#NUM!' which could either indicate an error in the calculation or some impossible number to record (infinity). Also, I wanted to continue trying to fit a better GARCH model for our SPX model (since the Ljung tests returned unfavorable) but due to time constraints this was not possible.

Executive Summary

The purpose of this project was to forecast returns of multiple risky equity securities and use the information to create optimal portfolios consisting of a risky asset and a risk-free asset. A first look at the data shows us that there may be some underlying trends in the price data, where the return data appears to be mostly white noise and volatile. We see that for both of our risky securities, Enbridge and Microsoft, the Sharpe ratio increases as the weight of the respective risky asset increases in the portfolio. We view this to mean that the risks taken with any portfolio choice including a risk-free asset and one of our risky securities will be a good investment.

To test for unit roots in our data, we use an Augmented Dickey-Fuller test given the complexity and size of our data. Our returns data is stationary, so we use autoregressive moving average models. Our prices data on the other hand all contain unit roots, and thus we use autoregressive integrated moving average models to achieve stationarity. When testing for autocorrelation, our prices are differenced once. Both prices and returns data have lag values outside of our acceptable confidence bands, and thus we move to optimize our portfolios using Akaike Information Criteria and Bayesian Information Criteria. The optimal models selected were run through the same autocorrelation tests and saw improvements across the board. Using Ljung tests, we also reject our null hypothesis that no autocorrelation exists for the first 5 lags of the return and prices variables.

When transitioning to generalized autoregressive conditional heteroskedasticity models, we only use Akaike Information Criteria as it appeared to perform better with previous results. In this case, we reject the null for the S&P 500 proxy security but fail to reject the null hypothesis mentioned above for both Enbridge and Microsoft, meaning we can use this model for Enbridge and Microsoft with confidence.

Table of Contents

Section One – Data

Our data is based on investors in the United States of America. For our risk-free rate, we use Ken French's data set and his daily risk-free return from the 1-month treasury bills. For our realized volatility series, we use the SPX index from the oxford-Man Institute of Quantitative Finance realized library site. The three risky assets we use are from Enbridge (ENB), Microsoft (MSFT), and the S&P 500 (SPX) close prices and adjusted prices (if possible).

These assets were chosen based on our interests and concerns of keeping a balanced portfolio with fewer assets. Microsoft is a well-known and reputable company and should serve as a good value stock over time. Enbridge is an energy company focused on gas, which we chose to possibly forecast its future which in the current state of the word is questioned given the demands for new renewable energies. Finally, the S&P 500 gives us a return based on the overall market which gives us a more robust

The original data was obtained from these sources below:

- "ENB.csv" : from yahoo finance for ENB (NYSE), from https://finance.yahoo.com/quote/ENB/history?period1=448156800&period2=1642464000&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true
- "MSFT.csv" : from yahoo finance for MSFT(NasdaqG5), from https://finance.yahoo.com/quote/MSFT/history?period1=511056000&period2=1642636800&interval=1d&filter=history&frequency=1d&includeAdjustedClose=true
- "F-F_Research_Data_Factors_Daily.CSV" : from Ken French's site download link for "Fama/French 3 Factors [Daily]", http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
- "oxfordmanrealizedvolatilityindices.csv" : from https://realized.oxford-man.ox.ac.uk/data/download, the top download link

Data Description

Table 1.1 covers additional data on each of the obtained data sets.

Table 1.1
Various measurements for each data set used for our project.

File Name	Measure	Description
ENB.csv		
	Observations	9539
	Date Range	3/15/1984 to 1/14/2022
	Frequency of Observations	Daily
	Unit of Observation	USD for close and adj close columns
MSFT.csv		
	Observations	9038
	Date Range	3/13/1986 to 1/19/2022
	Frequency of Observations	Daily
	Unit of Observation	USD for close and adj close columns
F-F_Research _Data_Factors_Daily.CSV		
	Observations	25126
	Date Range	1926/07/01 to 2021/11/30
	Frequency of Observations	Daily
	Unit of Observation	RF is a percentage
oxfordmanrealizedvolatilityindices.csv		
	Observations	156712
	Date Range	2000/01/03 to 2022-01/17
	Frequency of Observations	Daily
	Unit of Observation	USD for S&P 500 numbers that we pull, and r5 r10 are decimal variances.

Please refer to the script for a more detailed walkthrough of the applied transformations of our data. Below is a summary of the steps we took.

1. Downloaded data is untouched and then loaded into the environment as a list of data.frames.
2. We then pick components of each file out, such as the closing prices, dates, and rf returns for each file into separate variables
3. Each variable then undergoes additional formatting for their date columns to be registered as dates in R.
4. All variables are then combined using outer joins to capture all the observations possible for each variable. Then cutoff the dates to our oldest date for our risky assets, which would be for ENB stock assets at 1984-03-15.
5. At this point our data is prepared for our calculations and applied measures.

6. The first calculation we run is the returns for each close price column. This uses vector addition, subtract and division of our close price and close price (t –1). These returns are then bound onto our previously prepared data and returned.
7. The second calculation applies the measure, mean, standard deviation, and variance onto each return column. The measures are then annualized and bound with the daily measures and returned as a separate data.frame
8. The third calculation separates these daily and yearly (annualized) measures and separately calculates the Sharpe ratio using vectors of the return minus the proper risk-free rate divided by the vector of standard deviation.
9. This concludes the data manipulation for our exported document "Assignment One Data.xlsx"
10. We then do some additional data manipulation for our plots
    a. For our aggregate plots, we converted the final data frame into two long format data.frames for the returns and close prices which is subset for normal and log versions. This gives us four unique plots (All Return, All Log Return, All Closes, All Log Closes)
    b. The individual plots are simply using the date from our final data frame iterating with ggplot in a 'for' loop over each column in the data.frame.

Asset Descriptive Measures

Tables 1.2 and 1.3 show daily and yearly means, standard deviations, and variances for our riskless and riskless assets.

Table 1.2

Daily and Yearly results for Mean, Standard Deviation, and Variance for Risky/Riskless Assets

Assets	Daily.Mean	Daily.STD	Daily.Var	Yearly.Mean	Yearly.STD	Yearly.Var
RF	0.000128	0.000107	0.0000000114	0.032666	0.001696	0.000003
Return.ENB	0.000455	0.015324	0.000235	0.121388	0.243260	0.059176
Return.Adj.ENB	0.000623	0.015271	0.000233	0.169933	0.242427	0.058771
Return.MSFT	0.001119	0.021330	0.000455	0.325563	0.338597	0.114648
Return.Adj.MSFT	0.001169	0.021306	0.000454	0.342491	0.338220	0.114392
Return.SPX	0.000295	0.012289	0.000151	0.077286	0.195088	0.038059
Return.Log.ENB	0.000337	0.015360	0.000236	0.088606	0.243828	0.059452
Return.Log.Adj.ENB	0.000506	0.015298	0.000234	0.135971	0.242851	0.058977
Return.Log.MSFT	0.000890	0.021400	0.000458	0.251389	0.339713	0.115405
Return.Log.Adj.MSFT	0.000941	0.021374	0.000457	0.267527	0.339307	0.115129
Return.Log.SPX	0.000220	0.012309	0.000152	0.056942	0.195398	0.038180

Table 1.3
Daily and Yearly results for Sharpe Ratios for Risky/Riskless Assets

Assets	Daily.SharpeRatios	Yearly.SharpeRatios
Return.ENB	0.021350298	0.364719225
Return.Adj.ENB	0.032441714	0.566217592
Return.MSFT	0.046482943	0.865029813
Return.Adj.MSFT	0.048900981	0.9160455
Return.SPX	0.013661782	0.228714449
Return.Log.ENB	0.013632347	0.229422341
Return.Log.Adj.ENB	0.024739353	0.425381978
Return.Log.MSFT	0.035641738	0.643845955
Return.Log.Adj.MSFT	0.038065558	0.692179591
Return.Log.SPX	0.007492184	0.124236325

Time Plots of Our Data

In the below plots, we have included time-series line graphs of closing prices, log returns, and returns for Enbridge (NYSE: ENB), Microsoft (NASDAQ: MSFT), and the S&P 500 Index, with some going as far back as the 1980's.

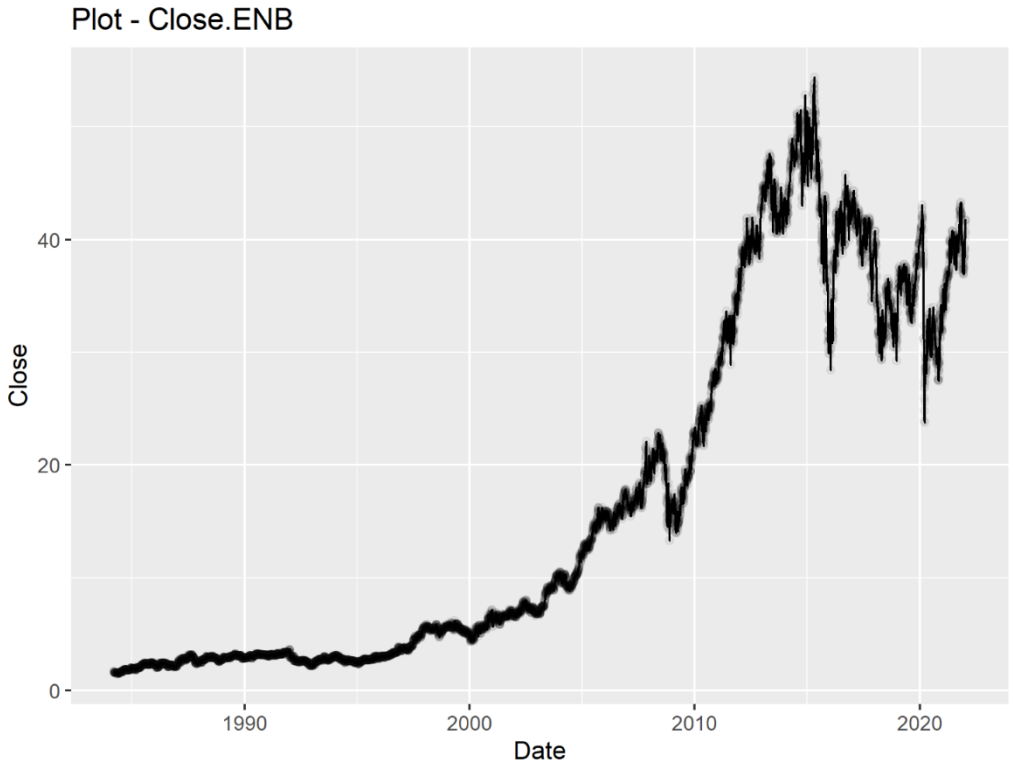
In exhibit 1.1, Enbridge's plots appear to resemble as we would expect for a Natural Resources company that is highly cyclical. We see that log and daily returns are quite volatile with some strong days, and some poor days.

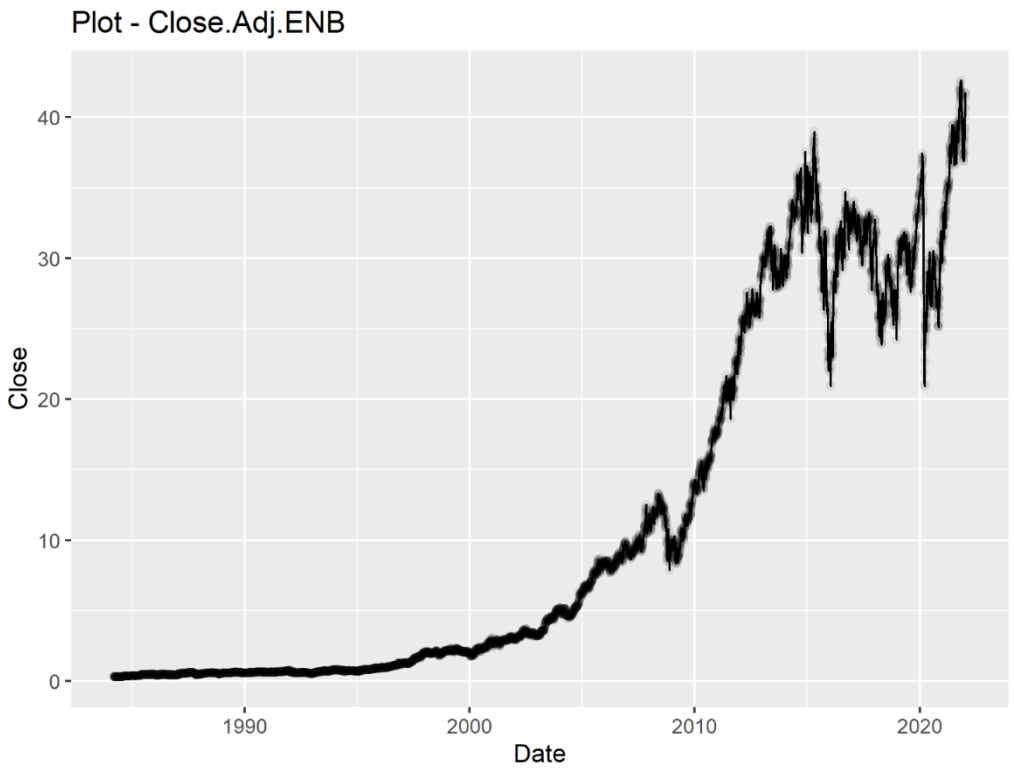
In exhibit 1.2, we see that results for Microsoft are as we would also expect. We see the log and normal returns lean more towards the upside.

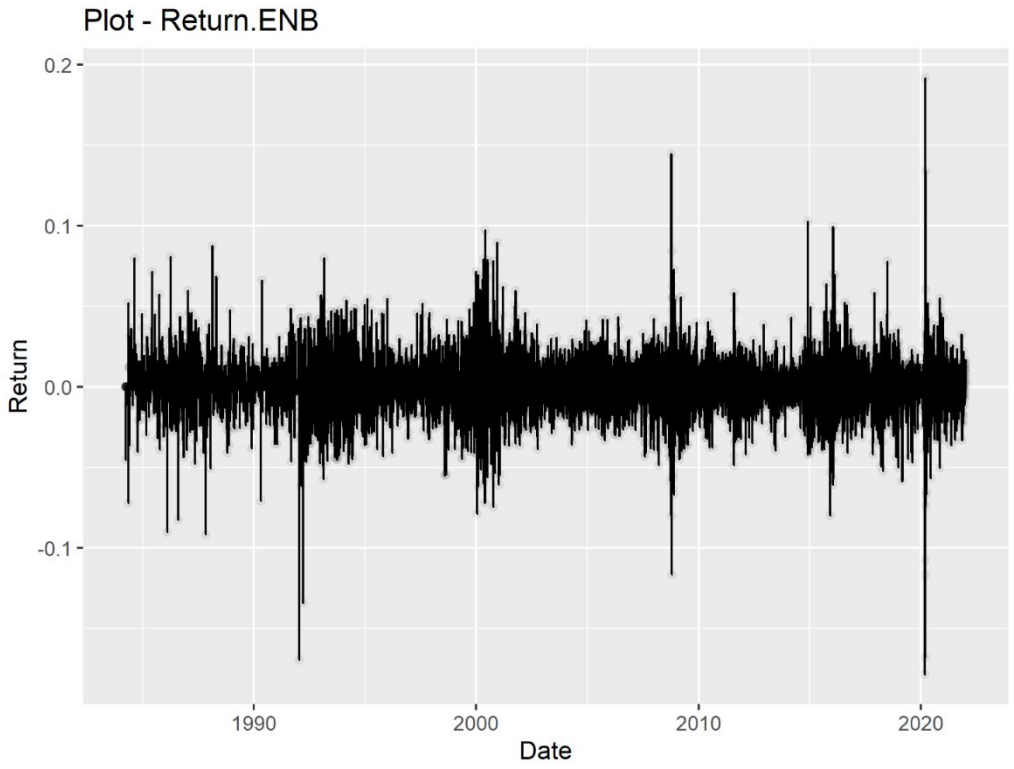
Exhibit 1.3 displays the plots for the S&P 500, which also tells the proper story of varying returns and retractions.

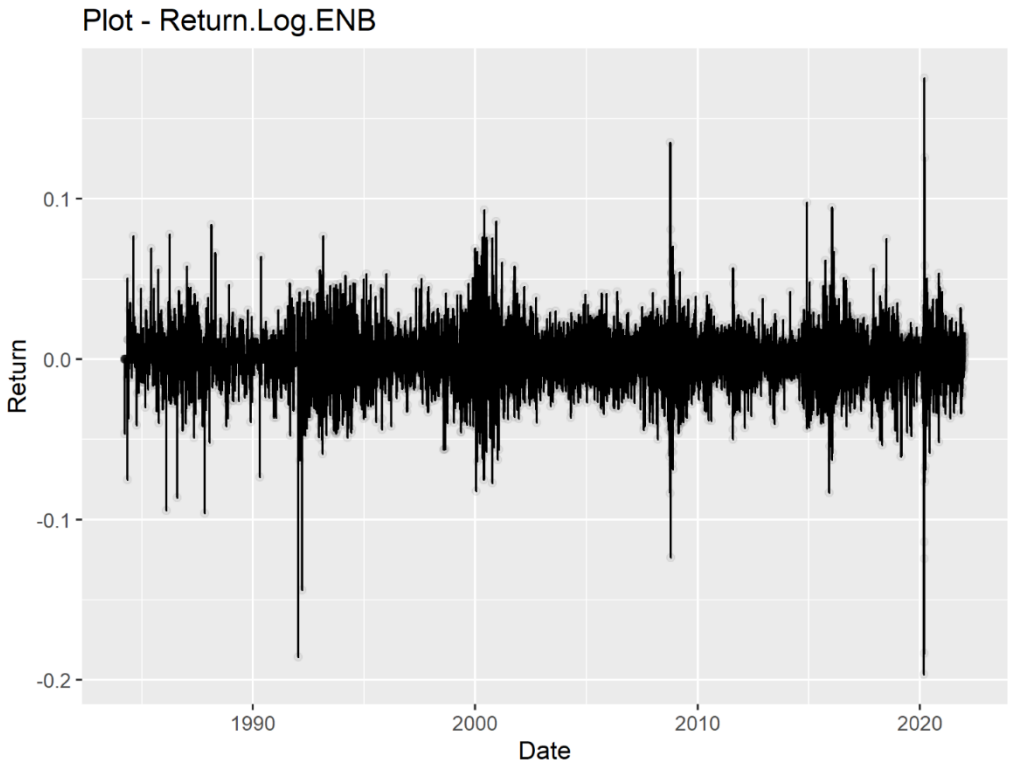
Lastly, we see Exhibit 1.4 comparing all our asset returns, with the risk-free rate as reference. The largest gainer is MSFT, which is expected.

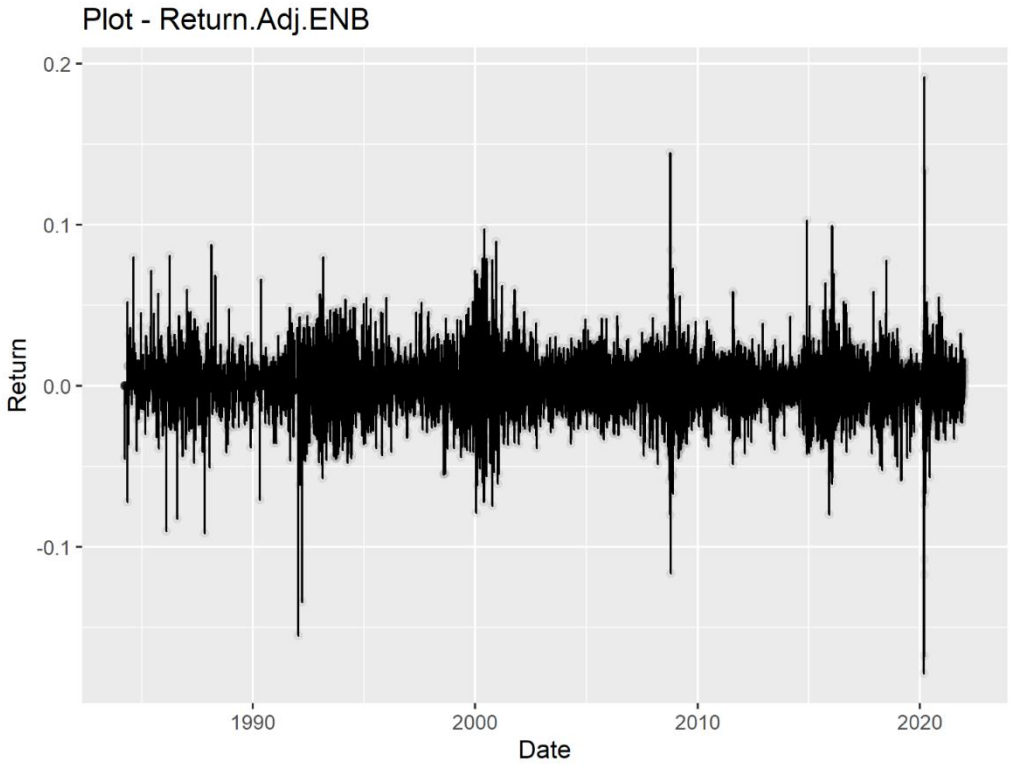
Exhibit 1.1: ENB






Plot - Return.ENB
0.2
0.1
0.0
-0.1
Return
1990
2000
2010
2020
Date


Plot - Return.Log.ENB
0.1
0.0
-0.1
-0.2
Return
1990
2000
2010
2020
Date


Plot - Return.Adj.ENB
0.2
0.1
0.0
-0.1
Return
1990
2000
2010
2020
Date

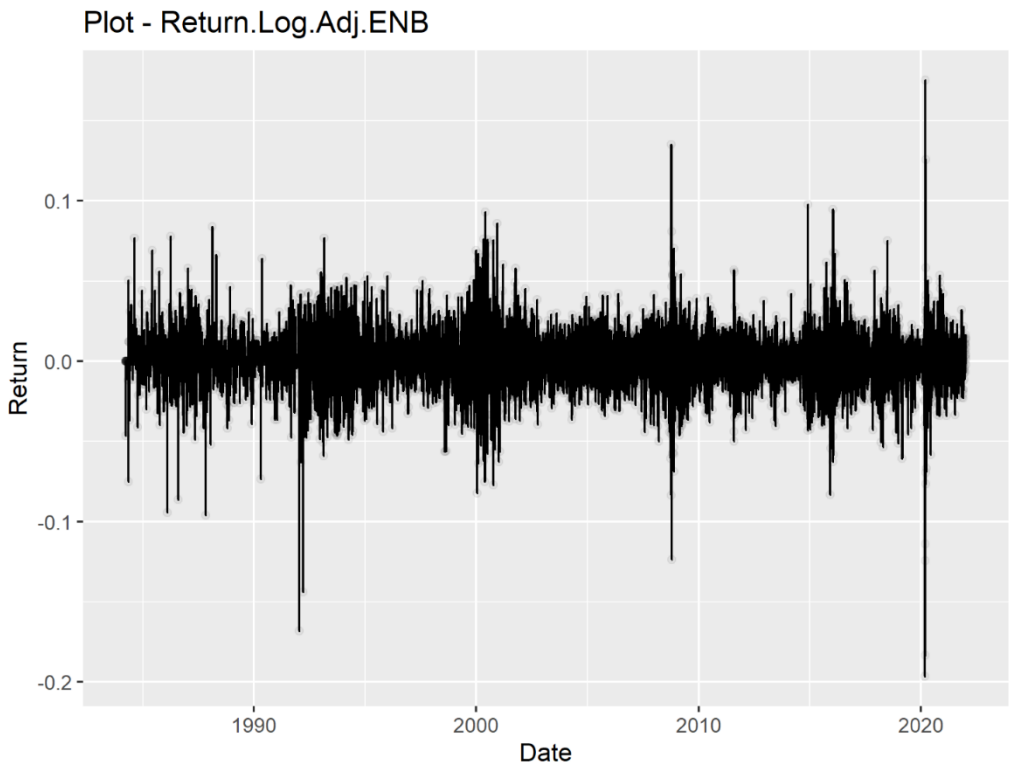
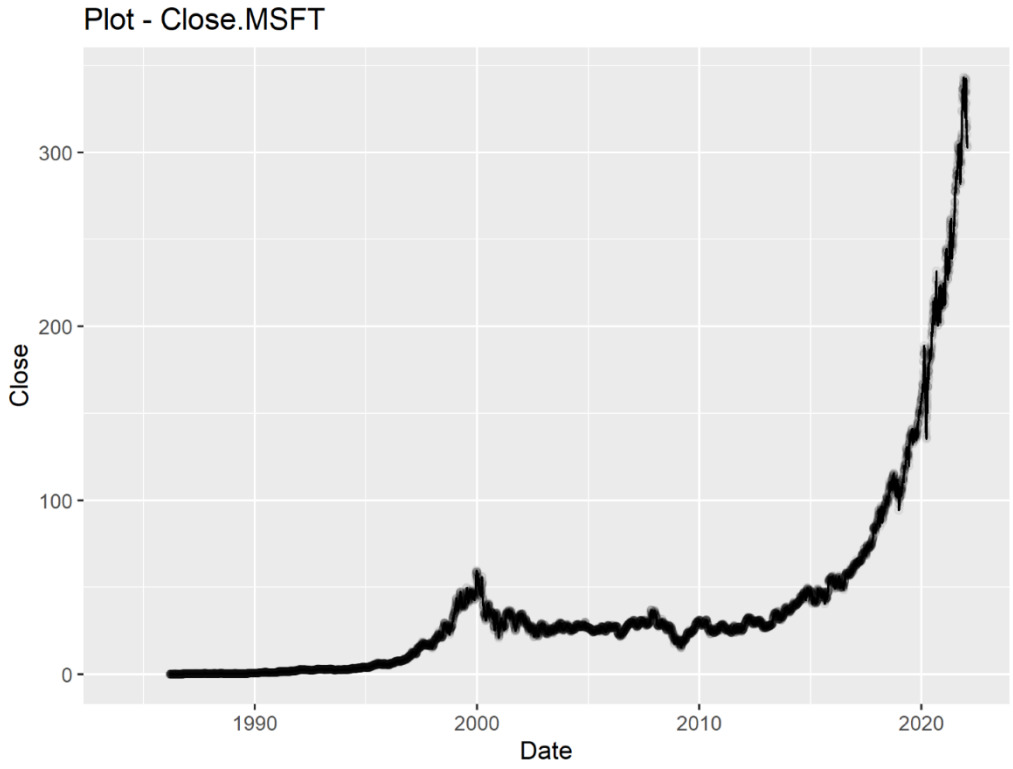
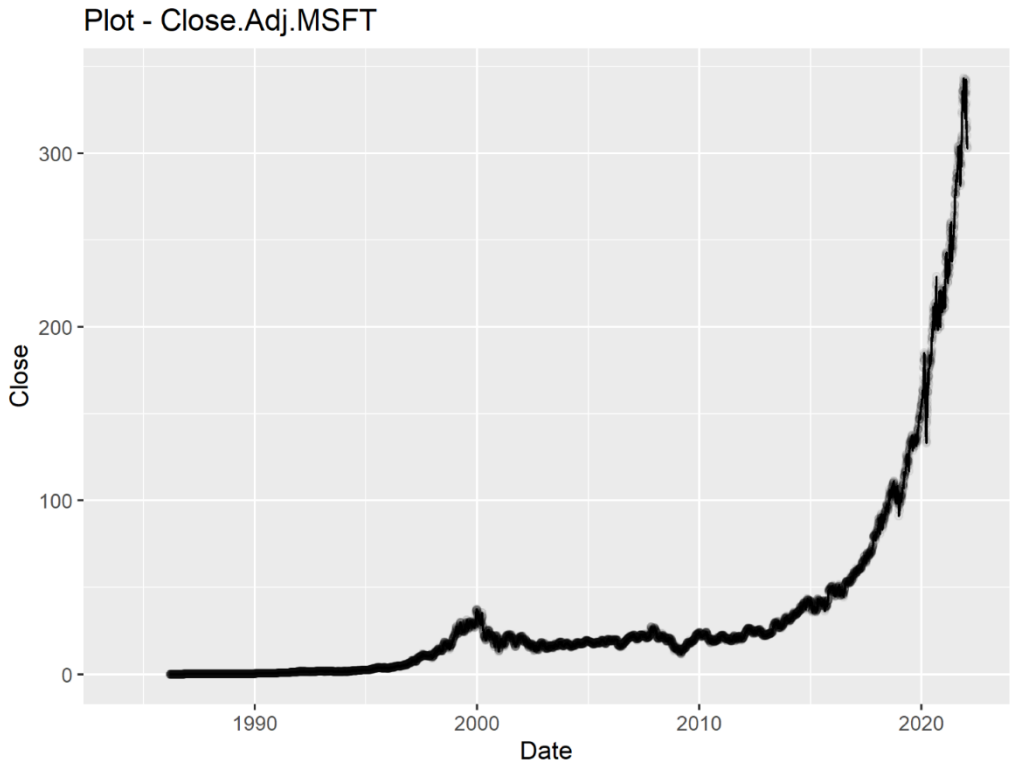
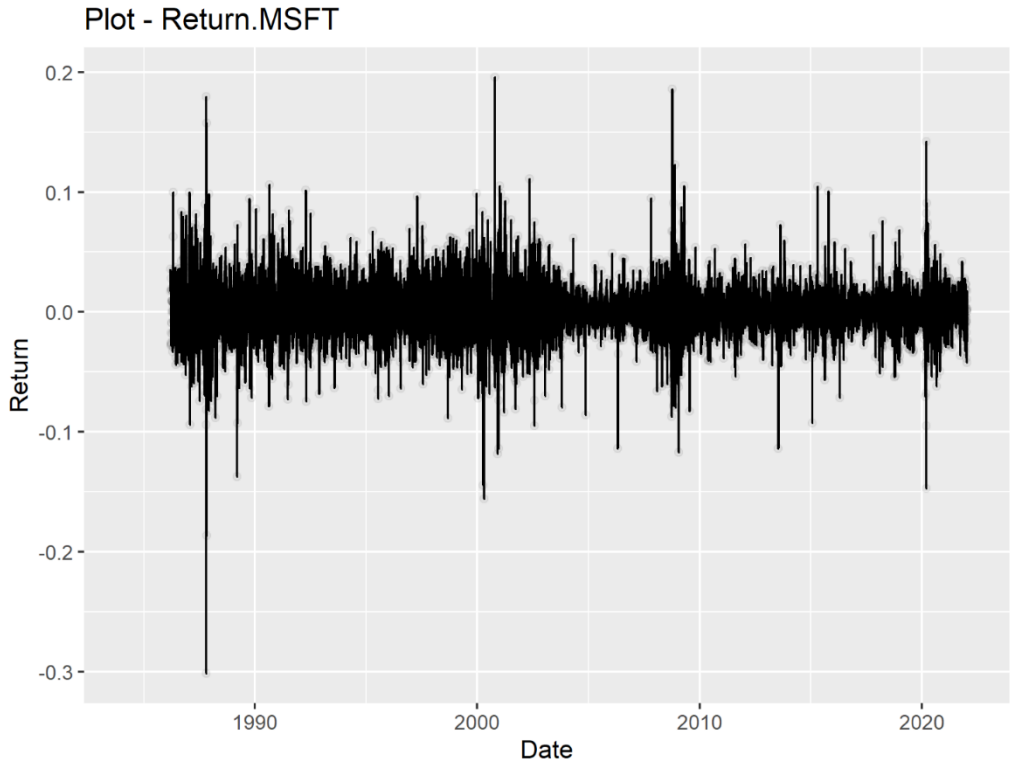
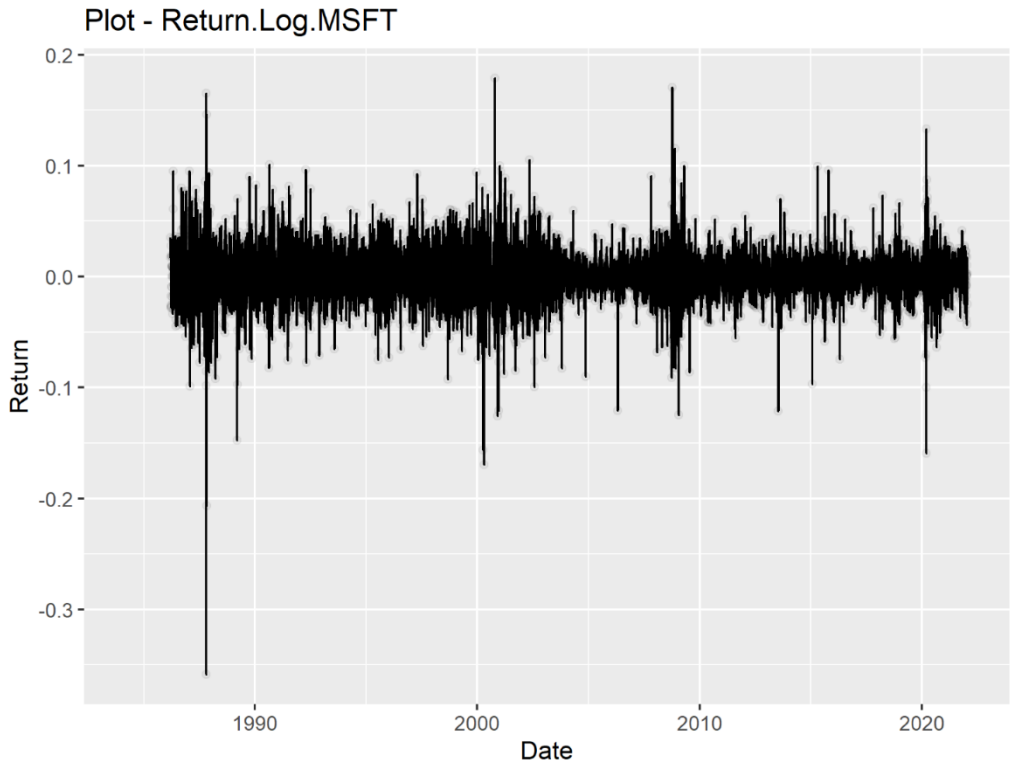


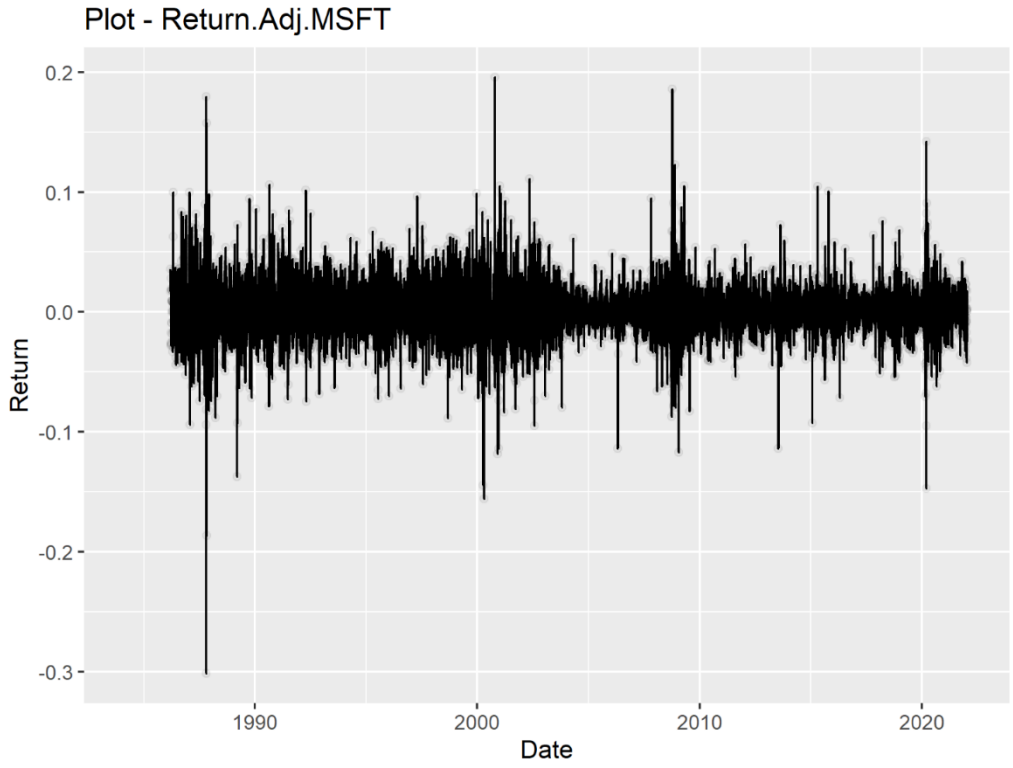
Exhibit 1.2: MSFT




Plot - Close.Adj.MSFT
Close
300
200
100
0
1990
2000
2010
2020
Date


Plot - Return.MSFT
Return
0.2
0.1
0.0
-0.1
-0.2
-0.3
1990
2000
2010
2020
Date


Plot - Return.Log.MSFT
0.2
0.1
0.0
-0.1
-0.2
-0.3
Return
1990
2000
2010
2020
Date


Plot - Return.Adj.MSFT
0.2
0.1
0.0
-0.1
-0.2
-0.3
Return
1990
2000
2010
2020
Date

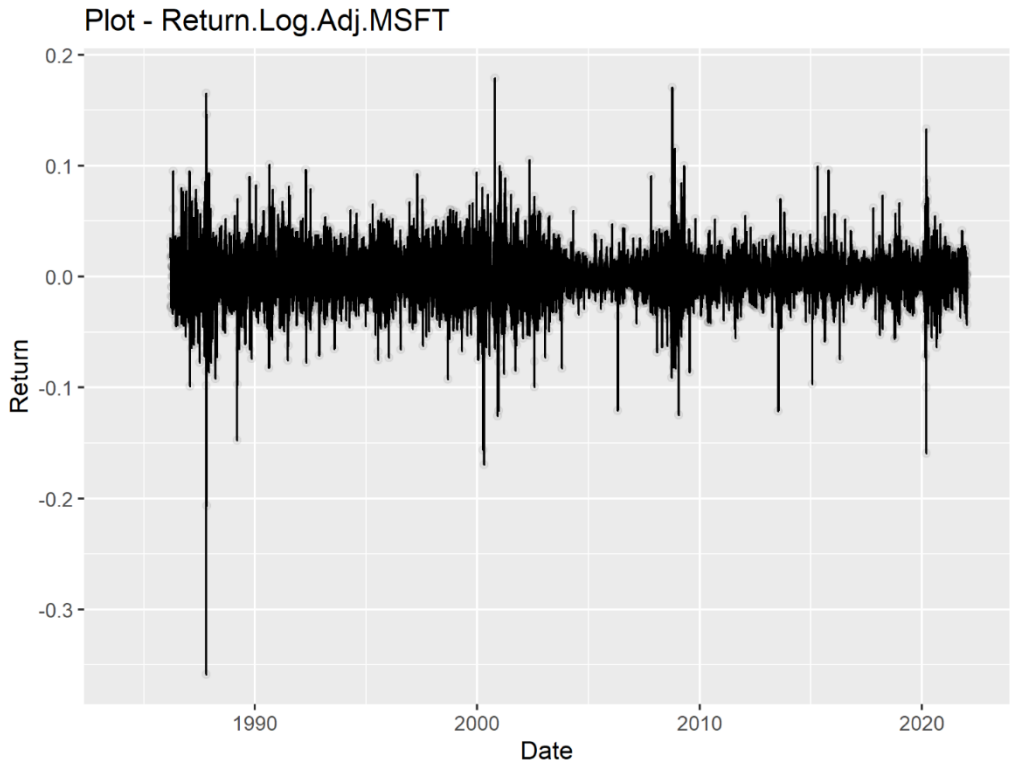
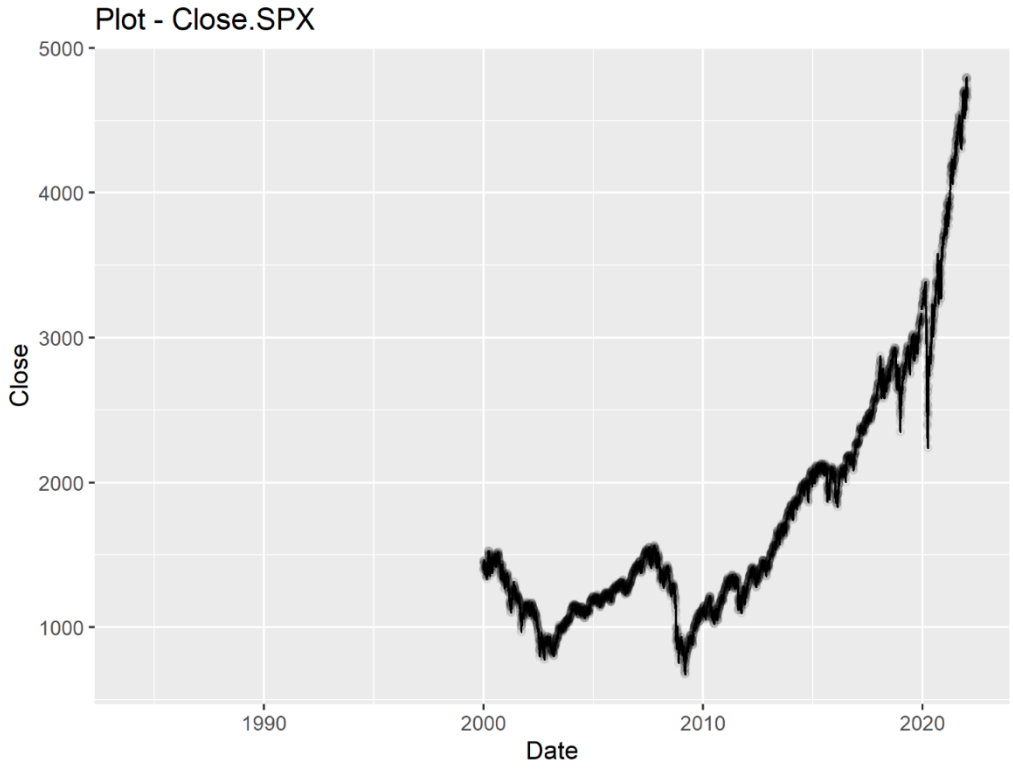
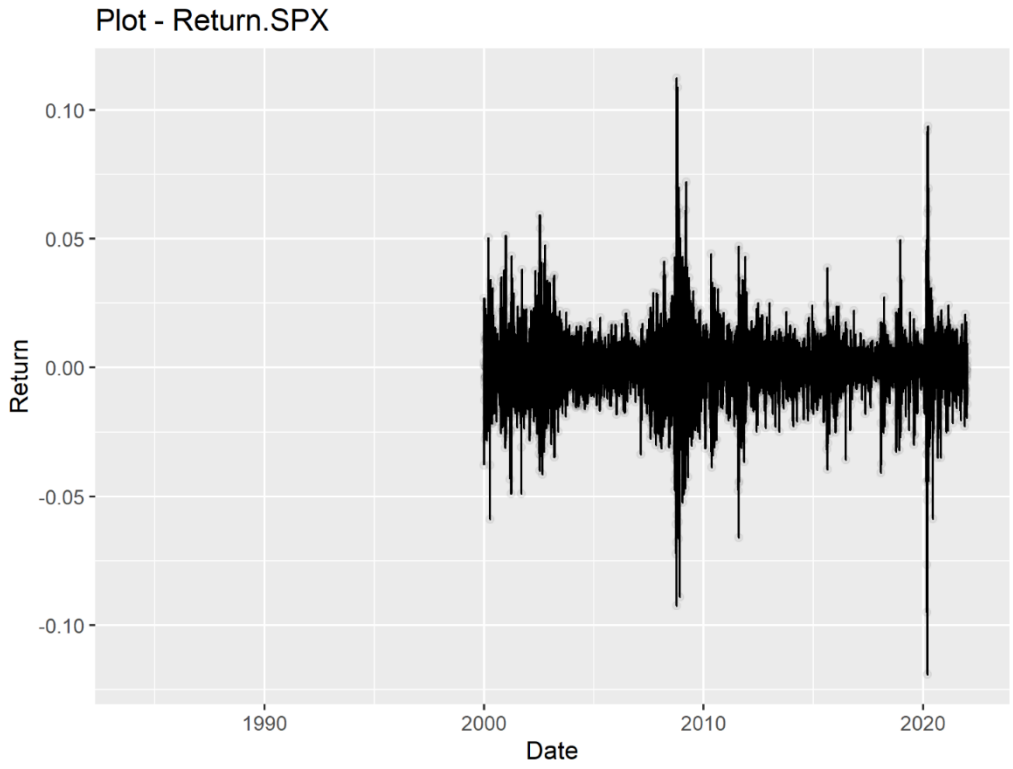


Exhibit 1.3: S&P 500 (SPX)




Plot - Return.SPX
0.10
0.05
0.00
-0.05
-0.10
Return
1990
2000
2010
2020
Date

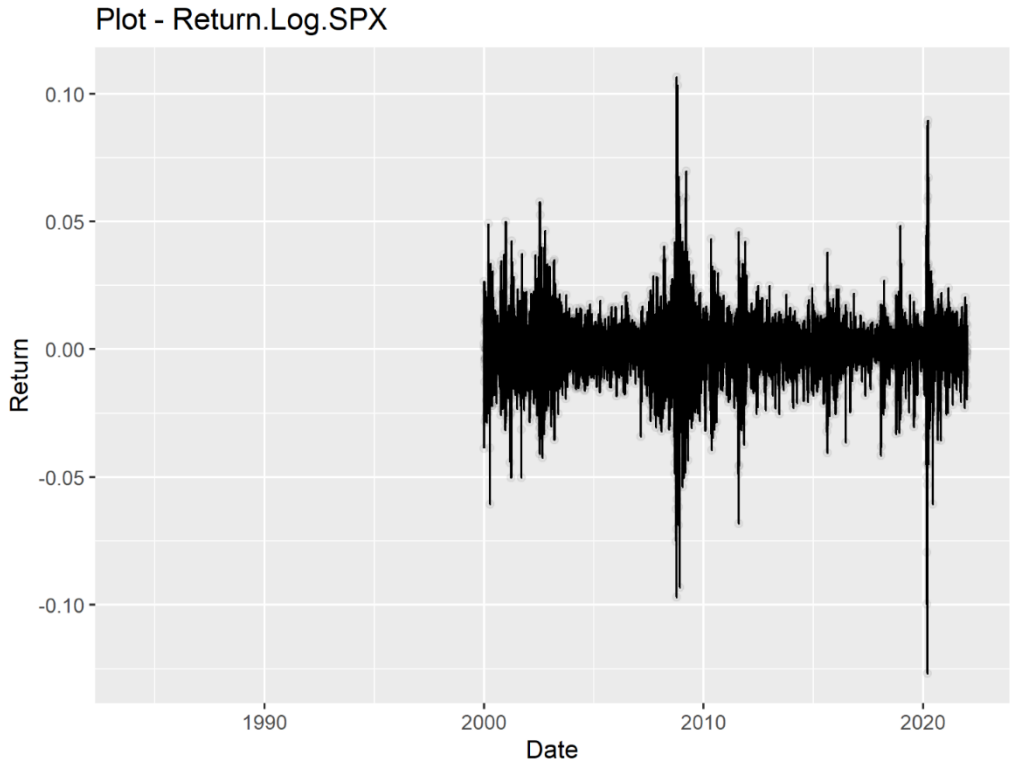
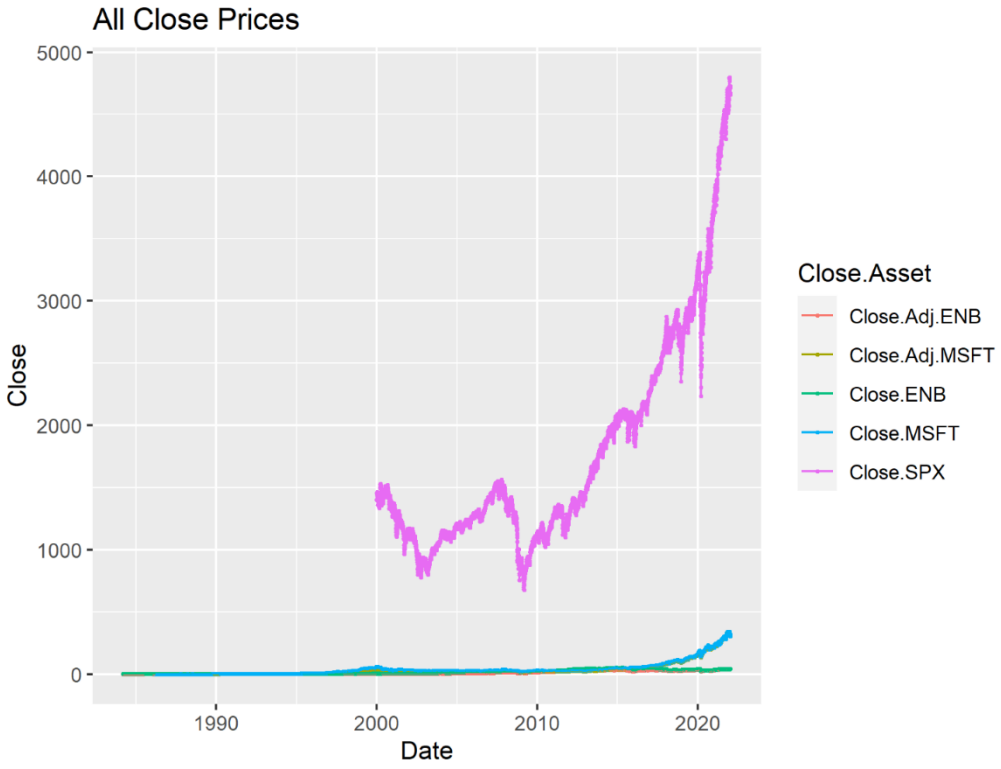
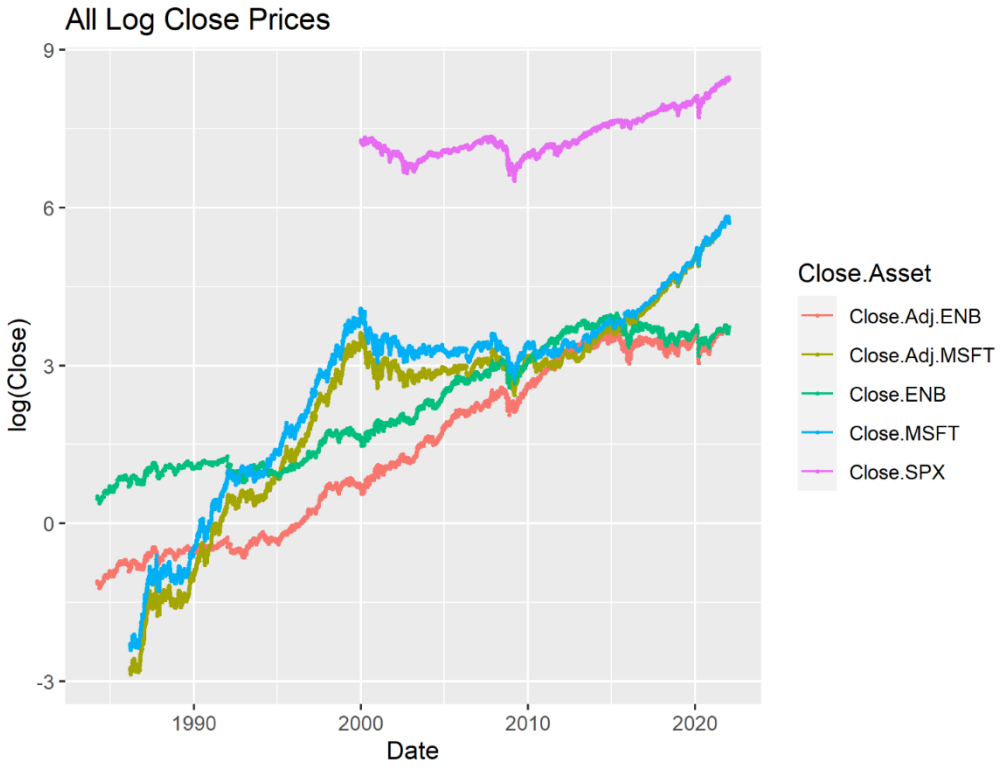
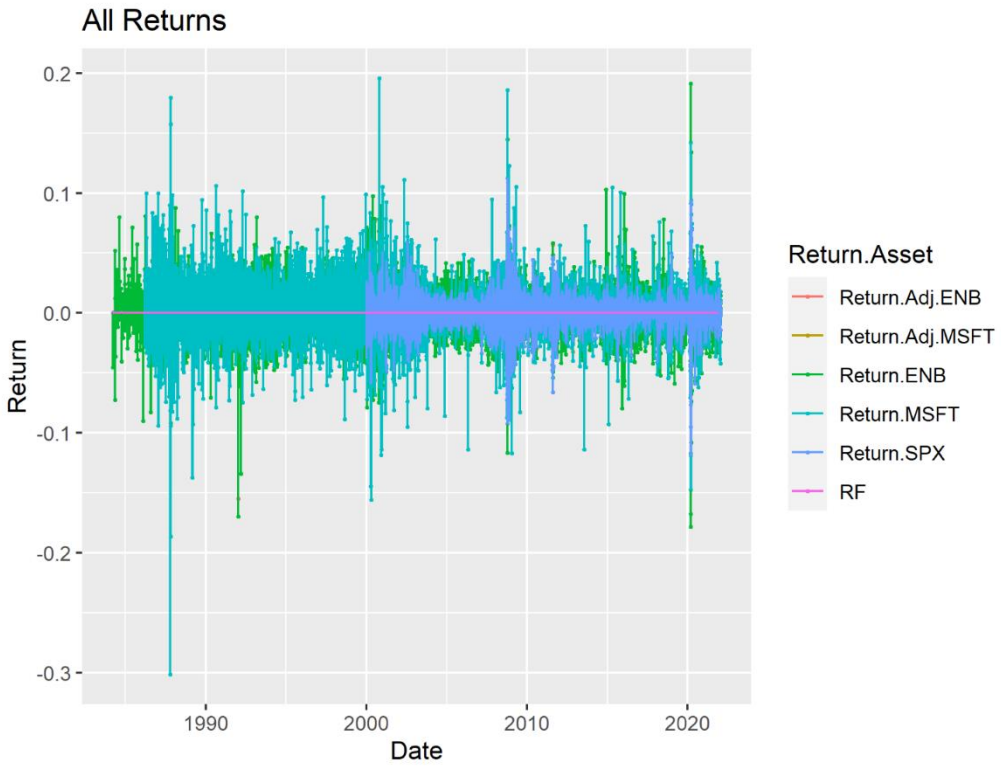
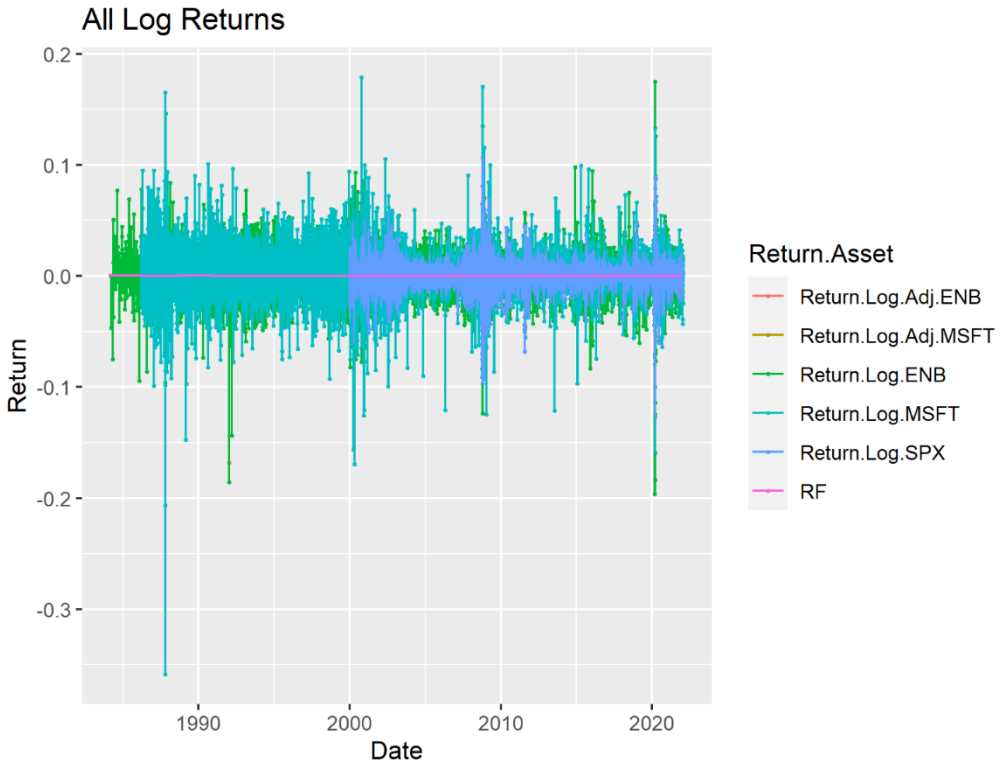

Plot - Return.Log.SPX
0.10
0.05
0.00
-0.05
-0.10
Return
1990
2000
2010
2020
Date

Exhibit 1.4: Risk-free & Aggregate Summary






All Returns
0.2
0.1
0.0
-0.1
-0.2
-0.3
Return
1990
2000
2010
2020
Date
Return.Asset
Return.Adj.ENB
Return.Adj.MSFT
Return.ENB
Return.MSFT
Return.SPX
RF


All Log Returns
0.2
0.1
0.0
-0.1
-0.2
-0.3
Return
1990
2000
2010
2020
Date
Return.Asset
Return.Log.Adj.ENB
Return.Log.Adj.MSFT
Return.Log.ENB
Return.Log.MSFT
Return.Log.SPX
RF

Code for Section One

Three separate functions were created to calculate returns, measures (mean, standard deviation, variance), and Sharpe ratios. The respective code can be found in figures 1.1, 1.2, and 1.3.

Figure 1.1

Function for calculating return, log return, and gross return for risky asset close prices.

```
Calculate_Returns = function(ClosePrice = CombinedData) {
  # This function calculates the return and log return for each close price

  # Variables
  ClosePriceLag = rbind(rep(NA, ncol(ClosePrice)), head(ClosePrice, -1))
  CloseIndex = which(grepl("Close", colnames(ClosePrice)) == TRUE) # Returns which columns are the closing prices.

  # Calculate Returns
  Return = (ClosePrice[,CloseIndex] - ClosePriceLag[,CloseIndex]) / ClosePriceLag[,CloseIndex]
  ReturnLog = log(ClosePrice[,CloseIndex]) - log(ClosePriceLag[,CloseIndex])
  Gross = ClosePrice[,CloseIndex] / ClosePriceLag[,CloseIndex]

  # Rename Columns
  colnames(Return) = gsub("Close", "Return", colnames(Return))
  colnames(ReturnLog) = gsub("Close", "Return.Log", colnames(ReturnLog))
  colnames(Gross) = gsub("Close", "Gross", colnames(Gross))

  # Combine Original Data and Returns into one data.frame
  Returns = cbind(ClosePrice, Gross, Return, ReturnLog)

  # End of Function
  return(Returns)
}
```

Figure 1.2

Function for calculating mean, standard deviation, and variance for close prices and returns with annualization.

```
Calculate_Measures = function(VariousReturns = CombinedData, k = 252) {
  # This function will subset for returns and calculate Mean, SD, and Variance
  # for daily and yearly times.

  # Variables
  ReturnIndex = which(grepl("Return|RF", colnames(VariousReturns)) == TRUE) # Returns which columns are the closing prices.
  VariousReturns = VariousReturns[,ReturnIndex] # Masks for columns using ReturnIndex
  Measures = c(mean, sd, var) # The measures we wish to apply to each return
  MeasuresNames = c("Mean", "StandardDeviation", "Variance") # For referencing in the later 'if' statement

  # Sub Functions
  ApplyMeasures = function(fun, Column, na_rm = TRUE){
    # This function applys a function onto a column

    return(fun(Column,na.rm = na_rm))

  }

  Annualized = function(x, type = "Mean", k = 1) {
    # This function actualizes any returns.

    # If statement to figure out how we want to annualize based on the measure.
    if (type == "Mean") {
      R = ((x+1)^(1/((1/k))))-1

    } else if (type == "StandardDeviation") {
      R = x*sqrt(k)

    } else if (type == "Variance") {
      R = x*k

    } else {
      return(stop("Something is Wrong :("))

    }

    # End of Function
    return(as.list(R))

  }
```

Figure 1.3

Function for calculating Sharpe ratios daily and yearly.

```
Calculate_SharpeRatios = function(DailyYearlyMeasures = A1.DailyYearlyMeasures) {
  # This calculates the Sharpe ratio using vectors

  # Variables - Index for Daily and Yearly means and standard deviations.
  ReturnIndex = grepl("Daily.Mean", colnames(DailyYearlyMeasures)) # Index for each mean column
  ReturnDaily = tail(DailyYearlyMeasures[,ReturnIndex], -1) # Removing the RF row

  ReturnIndex = grepl("Yearly.Mean", colnames(DailyYearlyMeasures)) # Index for each mean column
  ReturnYearly = tail(DailyYearlyMeasures[,ReturnIndex], -1)# Removing the RF row

  StandardDeviationIndex = grepl("Daily.StandardDeviation"
                                 , colnames(DailyYearlyMeasures)) # Index for each StandardDeviation column

  StandardDeviationDaily = tail(DailyYearlyMeasures[,StandardDeviationIndex], -1) # Removing the RF row

  StandardDeviationIndex = grepl("Yearly.StandardDeviation"
                                 , colnames(DailyYearlyMeasures)) # Index for each StandardDeviation column

  StandardDeviationYearly = tail(DailyYearlyMeasures[,StandardDeviationIndex], -1) # Removing the RF row

  RiskFreeDaily = DailyYearlyMeasures[1,grepl("Daily.Mean", colnames(DailyYearlyMeasures))]
    # Grabing the RiskFree daily, which should be in the first row

  RiskFreeYearly = DailyYearlyMeasures[1,grepl(
    "Yearly.Mean", colnames(DailyYearlyMeasures))] # Grabing the RiskFree daily, which should be in the first row

  # Calculate the SharpeRatio

  SharpeRatiosDaily = (ReturnDaily - RiskFreeDaily) / StandardDeviationDaily
    # Calculating the sharpe ratio for daily returns

  SharpeRatiosYearly = (ReturnYearly - RiskFreeYearly) / StandardDeviationYearly
    # Calculatin the sharpe ratio for yearly returns.

  SharpeRatios = matrix(c(SharpeRatiosDaily, SharpeRatiosYearly), nrow = length(SharpeRatiosDaily), ncol = 2) %>%
    as.data.frame # Combining all my ratio together

  colnames(SharpeRatios) = c("Daily.SharpeRatios", "Yearly.SharpeRatios")

  rownames(SharpeRatios) = DailyYearlyMeasures[-1,] %>% rownames()
    # to name the rows I need to make sure to get rid of the risk free row name

  # End of Function
  return(SharpeRatios)

}
```

Section Two – Weighted Portfolios

For this analysis, we assume our investors are in the United States of America and have access to financial markets required to own the securities discussed. ENB is viewed as a highly cyclical company in the Natural Gas industry, offering potentially high returns for investors who believe that the company will be able to transition their business model to take advantage of a "green energy" future. MSFT has long been viewed as an innovative tech giant, constantly introducing disruptive technology for the future of business, with consistent positive returns and high prospects of growth. Lastly, we view SPX as our benchmark to the US equity markets, acting as the level at which a passive but risk-tolerant investor would still choose to see growth in their investment. Our data for ENB and MSFT are downloaded from yahoo! Finance, while we use the data for the SPX index from the Oxford-Man Institute of Quantitative Finance realized library site. Furthermore, any indication of returns or price with the term ".adj" is referring to the stocks adjusted price/return with dividends.

Weighted Portfolio Results

In the tables 2.1 and 2.2 our means, standard deviations, variances, and Sharpe ratios are reported on a daily and yearly basis. For our yearly Sharpe ratios, rather than approximating by multiplying 252 or √252 onto our daily ratio, we used our yearly mean and standard deviation return and calculated it the same way as our daily ratio. We believe that this is more representative of the true data and is a more accurate measure of risk and risk-adjusted returns. We see that the adjusted-value Sharpe ratios for our portfolios represent what we would expect in a poorly diversified portfolio with a high-performing risky asset, such that they increase with the additional weight of the high-risk, high-return assets, but remain under the universal benchmark of 1.0. We note that the results for the SPX

index seem much lower than expected and will be cause for further investigation to determine the validity of those results.

Equal Weighted Portfolio

These are single asset portfolios that have been assigned weights of 50, 75, and 90 percent onto the risky asset, with the complement being covered by the riskless asset. In theory, as our risk aversion increases, we shouldn't see any change in our Sharpe ratio since it calculates return per unit of risk taken and thus does not change with leverage. However, in table 2.2, we find that as we increase the weight of the respective risky assets to their portfolio the Sharpe ratios remain constant relatively with extremely minor increases. This may allow us to interpret the choices of our risky assets as those that would be beneficial for a risk-averse investor, as the Sharpe ratios increase as we increase the level of risk that is taken. We say this as it is typically understood that a risk-averse investor does not wish to accept extra risk without commensurate return. However, it should be the case that as weight increases the Sharpe ratio is constant, the minor increase we see in table 2.2 is likely due to some error or rounding.

Table 2.1
Daily and Yearly results for Mean, Standard Deviation, and Variance for Equal Portfolio Returns

Row. Names	Weight (Risky)	D.Mean	D.STD	D.Var	Y.Mean	Y.STD	Y.Var
RF	0%	0.0128%	0.0107%	0.0000%	3.2666%	0.1696%	0.0003%
ENB	50%	0.0286%	0.7670%	0.0059%	7.4814%	12.1752%	1.4824%
Adj.ENB	50%	0.0371%	0.7643%	0.0058%	9.7915%	12.1334%	1.4722%
MSFT	50%	0.0624%	1.0672%	0.0114%	17.0334%	16.9407%	2.8699%
Adj.MSFT	50%	0.0650%	1.0660%	0.0114%	17.7815%	16.9216%	2.8634%
SPX	50%	0.0176%	0.6150%	0.0038%	4.5401%	9.7628%	0.9531%
Log.ENB	50%	0.0227%	0.7688%	0.0059%	5.8955%	12.2038%	1.4893%
Log.Adj.ENB	50%	0.0312%	0.7657%	0.0059%	8.1828%	12.1547%	1.4774%
Log.MSFT	50%	0.0510%	1.0707%	0.0115%	13.7068%	16.9962%	2.8887%
Log.Adj.MSFT	50%	0.0535%	1.0694%	0.0114%	14.4407%	16.9757%	2.8817%
Log.SPX	50%	0.0138%	0.6160%	0.0038%	3.5463%	9.7784%	0.9562%

ENB	75%	0.0366%	1.1504%	0.0132%	9.6530%	18.2624%	3.3352%
Adj.ENB	75%	0.0492%	1.1465%	0.0131%	13.2067%	18.1998%	3.3123%
MSFT	75%	0.0878%	1.6006%	0.0256%	24.7611%	25.4093%	6.4563%
Adj.MSFT	75%	0.0916%	1.5988%	0.0256%	25.9589%	25.3808%	6.4419%
SPX	75%	0.0235%	0.9226%	0.0085%	6.0914%	14.6451%	2.1448%
Log.ENB	75%	0.0277%	1.1531%	0.0133%	7.2351%	18.3052%	3.3508%
Log.Adj.ENB	75%	0.0404%	1.1485%	0.0132%	10.7279%	18.2318%	3.3240%
Log.MSFT	75%	0.0707%	1.6059%	0.0258%	19.4807%	25.4928%	6.4988%
Log.Adj.MSFT	75%	0.0745%	1.6040%	0.0257%	20.6391%	25.4622%	6.4832%
Log.SPX	75%	0.0178%	0.9240%	0.0085%	4.5823%	14.6686%	2.1517%
ENB	90%	0.0413%	1.3805%	0.0191%	10.9768%	21.9149%	4.8026%
Adj.ENB	90%	0.0565%	1.3758%	0.0189%	15.3063%	21.8398%	4.7698%
MSFT	90%	0.1031%	1.9207%	0.0369%	29.6394%	30.4906%	9.2968%
Adj.MSFT	90%	0.1076%	1.9186%	0.0368%	31.1343%	30.4564%	9.2759%
SPX	90%	0.0270%	1.1071%	0.0123%	7.0332%	17.5746%	3.0887%
Log.ENB	90%	0.0307%	1.3837%	0.0191%	8.0470%	21.9661%	4.8251%
Log.Adj.ENB	90%	0.0460%	1.3782%	0.0190%	12.2835%	21.8781%	4.7865%
Log.MSFT	90%	0.0825%	1.9270%	0.0371%	23.0843%	30.5909%	9.3580%
Log.Adj.MSFT	90%	0.0871%	1.9247%	0.0370%	24.5175%	30.5541%	9.3356%
Log.SPX	90%	0.0202%	1.1089%	0.0123%	5.2088%	17.6027%	3.0986%

* Note that “D” and “Y” in our column names stand for Daily and Yearly.

Table 2.2

Daily and Yearly results for Sharpe Ratios for Equal Portfolio Returns

Row.Names	Weight (Risky)	D.SharpeRatios	Y.SharpeRatios
ENB	50%	0.020702127	0.346179266
Adj.ENB	50%	0.031817662	0.537765199
MSFT	50%	0.046552741	0.812644497
Adj.MSFT	50%	0.048978882	0.857772175
SPX	50%	0.007909901	0.130445645
Log.ENB	50%	0.012978427	0.215414746
Log.Adj.ENB	50%	0.024109041	0.404466920
Log.MSFT	50%	0.035706870	0.614263379
Log.Adj.MSFT	50%	0.038138766	0.658239844
Log.SPX	50%	0.001742826	0.028605142
ENB	75%	0.020703499	0.349697251
Adj.ENB	75%	0.031819006	0.546161552

MSFT	75%	0.046901220	0.845925982
Adj.MSFT	75%	0.049327653	0.894071767
SPX	75%	0.011610277	0.192884592
Log.ENB	75%	0.012979697	0.216795940
Log.Adj.ENB	75%	0.024110405	0.409247375
Log.MSFT	75%	0.036053032	0.636025764
Log.Adj.MSFT	75%	0.038485274	0.682286776
Log.SPX	75%	0.005437688	0.089693355
ENB	90%	0.020703857	0.351822773
Adj.ENB	90%	0.031819300	0.551273474
MSFT	90%	0.047017293	0.864949455
Adj.MSFT	90%	0.049443817	0.915000689
SPX	90%	0.012843586	0.214320044
Log.ENB	90%	0.012980058	0.217623531
Log.Adj.ENB	90%	0.024110743	0.412143229
Log.MSFT	90%	0.036168353	0.647830266
Log.Adj.MSFT	90%	0.038600705	0.695516764
Log.SPX	90%	0.006669181	0.110335104

* Note that “D” and “Y” in our column names stand for Daily and Yearly.

Optimal Mean Variance (OMV) Portfolio

We now take our argument one step further to test the level of risk aversion without just a general assumption such as the weight of the risky asset. In reference to table 2.4 for our first section, an investor with low risk-aversion (k = 1) we have the highest results across the board. As we increase the level of risk-aversion (up to k = 7) we see the lowest results. This represents the common understanding that an investor who is more risk-averse would prefer to minimize their variance (subject to many other potential parameters which we will not touch upon) by increasing the weight on the riskless asset while taking weight off the risky asset. This results in a lower return mean, standard deviation, and variance. This concept is also reflected in table 5 containing our risky asset weights; risk aversion increases cause all our measures to decreases.

Table 2.3

Daily and Yearly results for Mean, Standard Deviation, and Variance for Optimal Mean Variance Returns

Row.Names	RA (K)	D.Mean	D.STD	D.Var	Y.Mean	Y.STD	Y.Var
RF	NA	0.01%	0.01%	0.00%	3.27%	0.17%	0.00%
ENB	Low, K = 1	0.014%	0.037%	0.000%	3.462%	0.589%	0.003%
Adj.ENB	Low, K = 1	0.014%	0.053%	0.000%	3.702%	0.841%	0.007%
MSFT	Low, K = 1	0.016%	0.098%	0.000%	4.187%	1.555%	0.024%
Adj.MSFT	Low, K = 1	0.017%	0.103%	0.000%	4.322%	1.633%	0.027%
SPX	Low, K = 1	0.007%	0.026%	0.000%	1.668%	0.406%	0.002%
Log.ENB	Low, K = 1	0.013%	0.027%	0.000%	3.353%	0.423%	0.002%
Log.Adj.ENB	Low, K = 1	0.014%	0.042%	0.000%	3.526%	0.664%	0.004%
Log.MSFT	Low, K = 1	0.014%	0.075%	0.000%	3.692%	1.194%	0.014%
Log.Adj.MSFT	Low, K = 1	0.015%	0.080%	0.000%	3.798%	1.273%	0.016%
Log.SPX	Low, K = 1	0.006%	0.019%	0.000%	1.606%	0.299%	0.001%
ENB	Med, K = 3	0.013%	0.016%	0.000%	3.331%	0.252%	0.001%
Adj.ENB	Med, K = 3	0.013%	0.020%	0.000%	3.411%	0.321%	0.001%
MSFT	Med, K = 3	0.013%	0.034%	0.000%	3.380%	0.540%	0.003%
Adj.MSFT	Med, K = 3	0.013%	0.036%	0.000%	3.425%	0.565%	0.003%
SPX	Med, K = 3	0.006%	0.011%	0.000%	1.559%	0.171%	0.000%
Log.ENB	Med, K = 3	0.013%	0.013%	0.000%	3.295%	0.212%	0.000%
Log.Adj.ENB	Med, K = 3	0.013%	0.017%	0.000%	3.353%	0.272%	0.001%
Log.MSFT	Med, K = 3	0.013%	0.027%	0.000%	3.216%	0.425%	0.002%
Log.Adj.MSFT	Med, K = 3	0.013%	0.028%	0.000%	3.252%	0.449%	0.002%
Log.SPX	Med, K = 3	0.006%	0.009%	0.000%	1.539%	0.145%	0.000%
ENB	High, K = 7	0.013%	0.012%	0.000%	3.294%	0.187%	0.000%
Adj.ENB	High, K = 7	0.013%	0.013%	0.000%	3.328%	0.205%	0.000%
MSFT	High, K = 7	0.012%	0.017%	0.000%	3.151%	0.271%	0.001%
Adj.MSFT	High, K = 7	0.012%	0.018%	0.000%	3.170%	0.280%	0.001%
SPX	High, K = 7	0.006%	0.008%	0.000%	1.528%	0.125%	0.000%
Log.ENB	High, K = 7	0.013%	0.011%	0.000%	3.278%	0.178%	0.000%
Log.Adj.ENB	High, K = 7	0.013%	0.012%	0.000%	3.303%	0.192%	0.000%
Log.MSFT	High, K = 7	0.012%	0.015%	0.000%	3.081%	0.230%	0.001%
Log.Adj.MSFT	High, K = 7	0.012%	0.015%	0.000%	3.096%	0.239%	0.001%
Log.SPX	High, K = 7	0.006%	0.007%	0.000%	1.519%	0.119%	0.000%

* Note that “RA” stands for risk aversion, and “D” and “Y” in our column names stand for Daily and Yearly.

Table 2.4

Results for Mean, Standard Deviation, and Variance for Optimal Mean Variance Weights

Row.Names	RA (K)	Mean	STD	Var
Weight.ENB	Low, K = 1	2.1350%	0.6971%	0.0049%
Weight.Adj.ENB	Low, K = 1	3.2442%	0.6995%	0.0049%
Weight.MSFT	Low, K = 1	4.6483%	0.5009%	0.0025%
Weight.Adj.MSFT	Low, K = 1	4.8901%	0.5014%	0.0025%
Weight.SPX	Low, K = 1	1.3662%	0.8693%	0.0076%
Weight.Log.ENB	Low, K = 1	1.3632%	0.6955%	0.0048%
Weight.Log.Adj.ENB	Low, K = 1	2.4739%	0.6983%	0.0049%
Weight.Log.MSFT	Low, K = 1	3.5642%	0.4992%	0.0025%
Weight.Log.Adj.MSFT	Low, K = 1	3.8066%	0.4998%	0.0025%
Weight.Log.SPX	Low, K = 1	0.7492%	0.8679%	0.0075%
Weight.ENB	Med, K = 3	0.7117%	0.2324%	0.0005%
Weight.Adj.ENB	Med, K = 3	1.0814%	0.2332%	0.0005%
Weight.MSFT	Med, K = 3	1.5494%	0.1670%	0.0003%
Weight.Adj.MSFT	Med, K = 3	1.6300%	0.1671%	0.0003%
Weight.SPX	Med, K = 3	0.4554%	0.2898%	0.0008%
Weight.Log.ENB	Med, K = 3	0.4544%	0.2318%	0.0005%
Weight.Log.Adj.ENB	Med, K = 3	0.8246%	0.2328%	0.0005%
Weight.Log.MSFT	Med, K = 3	1.1881%	0.1664%	0.0003%
Weight.Log.Adj.MSFT	Med, K = 3	1.2689%	0.1666%	0.0003%
Weight.Log.SPX	Med, K = 3	0.2497%	0.2893%	0.0008%
Weight.ENB	High, K = 7	0.3050%	0.0996%	0.0001%
Weight.Adj.ENB	High, K = 7	0.4635%	0.0999%	0.0001%
Weight.MSFT	High, K = 7	0.6640%	0.0716%	0.0001%
Weight.Adj.MSFT	High, K = 7	0.6986%	0.0716%	0.0001%
Weight.SPX	High, K = 7	0.1952%	0.1242%	0.0002%
Weight.Log.ENB	High, K = 7	0.1947%	0.0994%	0.0001%
Weight.Log.Adj.ENB	High, K = 7	0.3534%	0.0998%	0.0001%
Weight.Log.MSFT	High, K = 7	0.5092%	0.0713%	0.0001%
Weight.Log.Adj.MSFT	High, K = 7	0.5438%	0.0714%	0.0001%
Weight.Log.SPX	High, K = 7	0.1070%	0.1240%	0.0002%

* Note that “RA” stands for risk aversion

Portfolio Plots

Note that the data for our plots have been filtered to only log adjusted returns for ENB and MSFT values, and the log returns for SPX. With our script, we calculate for various returns such as, net return, log return, adjusted return, and log adjusted return whenever possible. Given each of these returns, we know that log adjusted (when possible), or log returns should be the most useful since it emulates a compounding effect, normally disturbed, and considers the return from dividends. So, for the sake of cleaner and clearer graphs, we've decided to only include these latter returns.

Equal Portfolio Return Plots

Plotting the equal portfolio returns seen in figures 2.1 to 2.3, show three similar sideways moving graphs. The most noticeable difference between the three is the increase in magnitude on y axis as our weight on the risky asset increases. We also see each graph getting choppier near the end.

Optimal Mean Variance Return Plots

Plotting the optimal mean variance returns seen in figures 2.4 to 2.6 shows three different graphs.

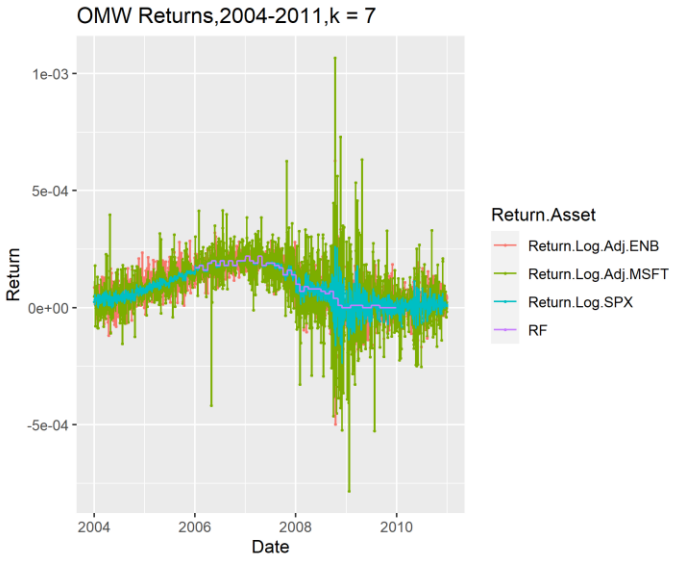
- Figure 2.4 shows a sideways trend that gets choppier near the end
- Figure 2.5 shows a slightly negative trend at the beginning and a sideways trend from the middle. It starts to get choppier near the end.
- Figure 2.6 shows a much clearer negative trend from the beginning, we still see a sideways trend closer to the end then before. It also gets choppier at the end.

What's interesting is that we see a stretching effect and numerous humps begin to form as *k* increases and our y-axis magnitude decreases. The most noticeable hump growth happens around the 2007-8 recession. If we take a closer look by subsetting for this date range (Plot 2.1) , the reasoning behind this hump becomes much clearer. Basically, higher risk aversion means higher weight onto our risk-free and around this time we saw major increases in our interest rates, hence why we see this exponential hump

growth across our figures as k increases. It's safe to say that all these humps are due to recessions, and most of the trend patterns we see are because of changes in the risk-free weight.

Plot 2.1

Subset of OMV with Added Risk-Free Rate



Optimal Mean Variance Weight Plots

Plotting the optimal mean variance weights seen in figures 2.7 to 2.9 shows three almost identical graphs all trending upwards. There appears to be a lot of stepped movement and numerous instances of flat, sideways movement. This can be explained by the way we solved for our weights by using the expected daily return. So, the only variable moving across time is our daily risk-free rate, which has many instances of constant and stepped behaviors in its return. The most noticeable difference between the three is the decrease in magnitude on the y axis as our k value increases.

Code for Section Two

Three separate functions were created to calculate equal portfolios, OMV weights, and OMV portfolios in the respective figures 2.10, 2.11, and 2.12.

Figure 2.10
Function for calculating equal portfolios

```
Create_Equal_Portfolio = function(Data = CombinedData, weight = 0.5) {
  # This function takes our dataset and returns a matrix containing the returns
  # for each asset return split equally with our risk free rate.

  # Variables
  Returns = Data[, grepl("Return", colnames(Data))]
  RiskFree = Data[,"RF"]

  # Calculate Equal Portfolio
  EqualPortfolio = (Returns*weight) + (RiskFree*(1-weight))

  # Add Date Column
  EqualPortfolio = cbind(Data$Date, EqualPortfolio)
  colnames(EqualPortfolio)[1] = "Date"

  # Rename Columns
  colnames(EqualPortfolio) = gsub("Return\\.", "Return.Equal.", colnames(EqualPortfolio))

  # End of Function
  return(EqualPortfolio)

}
```

Plot 2.11
Function for calculating OMV weights.

```
Create_OMV_Portfolio = function(Data = CombinedData, Measures = A1.DailyYearlyMeasures, k = 1) {
  # This function creates weights using data and K values

  # Variables
  RiskFree = Data[, grepl("RF", colnames(Data))]

  Returns = Measures[, grepl("Mean$", colnames(Measures))]
  Returns = Returns[grepl("Return", rownames(Returns)),]

  StandardDeviations = Measures[,grepl("StandardDeviation$", colnames(Measures))]
  StandardDeviations = StandardDeviations[grepl("Return", rownames(StandardDeviations)),]

  # Check
  if(all.equal(rownames(StandardDeviations), colnames(Returns)) == FALSE){
    stop("standard deviation row names do not match return column names")
  }

  # Calculation
  # I'll do it in three separate steps
  #   1. Do Asset Returns - RiskFree
  #   2. Divide by respective standard deviation
  #   3. multiply by 1/k

  Step1 = apply(Returns, 1, function(x) -(RiskFree - x[1]))
    # x[1] is used to index our daily mean returns

  Step2 = sweep(Step1, 2, StandardDeviations[,1], FUN = '/')
    # StandardDeviatons[,1] indexes for daily standard deviation
    # sweep() allows us to divide each column by the respective standard deviation

  Step3 = Step2 * 1/k # Apply our k value

  # Rename Columns
  colnames(Step3) = gsub("Return", "Return\\.Weight", colnames(Step3))

  # End of Function
  return(as.data.frame(Step3))

}
```

Plot 2.12

Function for calculating OMV portfolios using weights.

```
Apply_OMV_Weights = function(RiskyWeight = A2.OMVPortfolioWeights, Returns = CombinedData) {
  # This function applies our weights onto our returns and the risk free return

  # Variables
  RiskFreeWeight = 1 - RiskyWeight # Just the complement of our risky weights

  # Returns
  RiskyReturns = Returns[,grepl("Return", colnames(Returns))] * RiskyWeight
  RiskFreeReturns = Returns[,grepl("RF", colnames(Returns))] * RiskFreeWeight
  PortfolioReturn = RiskyReturns + RiskFreeReturns

  # End of function
  return(PortfolioReturn)
}
```

Section Three – Unit Root and ARIMA Selection

This section covers the results of our unit root tests and the initial ARMA and ARIMA model selection given our results. For our unit root tests, we use an augmented dickey fuller test, rather than just a normal dickey fuller given the complexity and size of our data. Furthermore, we will be using three kinds of models which are listed as, without intercept or trend, with intercept, and with intercept and trend as shown below. Furthermore, in tables 3.1 to 3.3 the model column refers to these three equations as 1, 2, and 3 respectively.

$$\Delta Pt = \gamma 1 Pt-1 + X\ p\ i=1\ \beta i \Delta Pt-i + \varepsilon t \quad (1)$$

$$\Delta Pt = \alpha 0 + \gamma 1 Pt-1 + X\ p\ i=1\ \beta i \Delta Pt-i + \varepsilon t \quad (2)$$

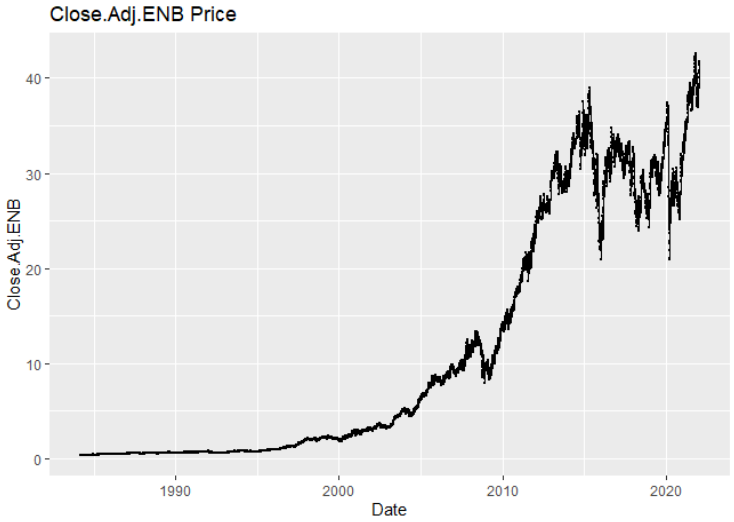
$$\Delta Pt = \alpha 0 + \alpha 1 t + \gamma 1 Pt-1 + X\ p\ i=1\ \beta i \Delta Pt-i + \varepsilon t \quad (3)$$

Finally, for these augmented dickey fuller tests, our null hypothesis (H_o) is the data contains a unit root and the alternative hypothesis (H_A) is that the data is stationary.

Reviewing Our Plots for Trends, Breaks, and Stochastic Trends

Plot 3.1

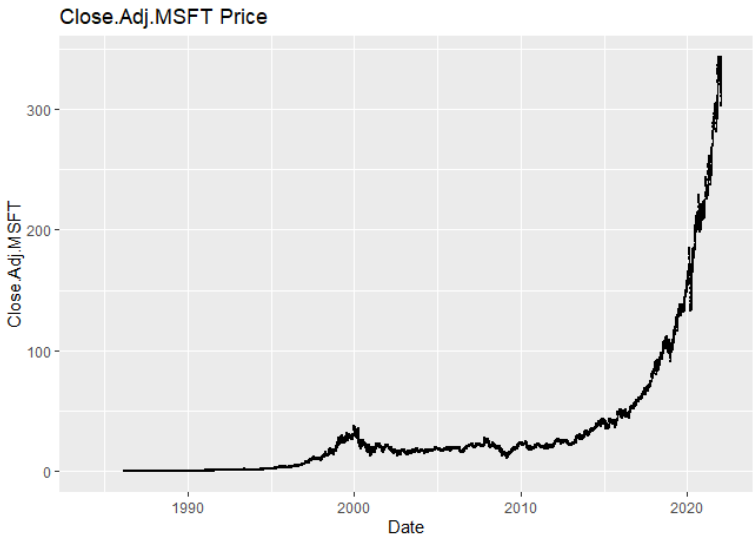
Time Series Plot of ENBs Closing Adjusted Price



For ENBs price, right up before 2010 it looks linear but there is a slight bend. At this point, there is a break and a strong upward linear trend, then we see another break trending sideways or a little bit downwards, and a break in 2020 going upwards. However, it is around the end of 2010 to 2020 where it seems very random or stochastically trending sideways or down.

Plot 3.2

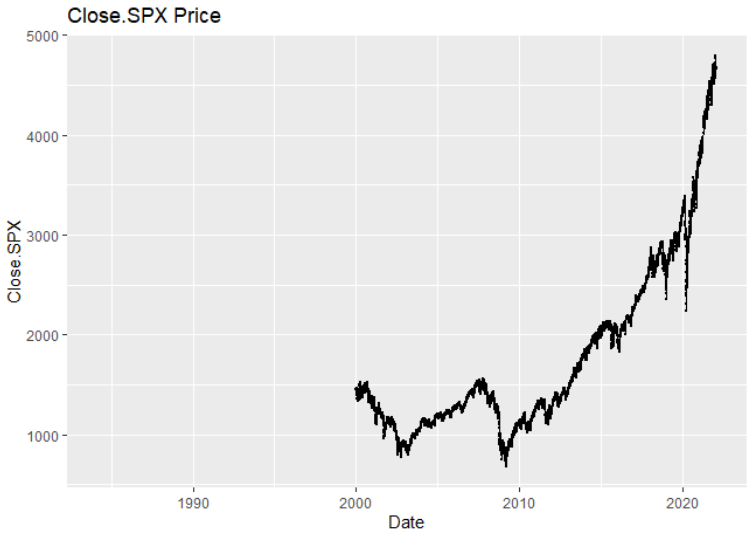
Time Series Plot of MSFTs Closing Adjusted Price



For MSFTs price, this looks like an exponential trend, we do see a break in the year 2000, it goes sideways (no apparent trend) until 2010 and begins to exponentially trend upwards.

Plot 3.3

Time Series Plot of SPXs Closing Price

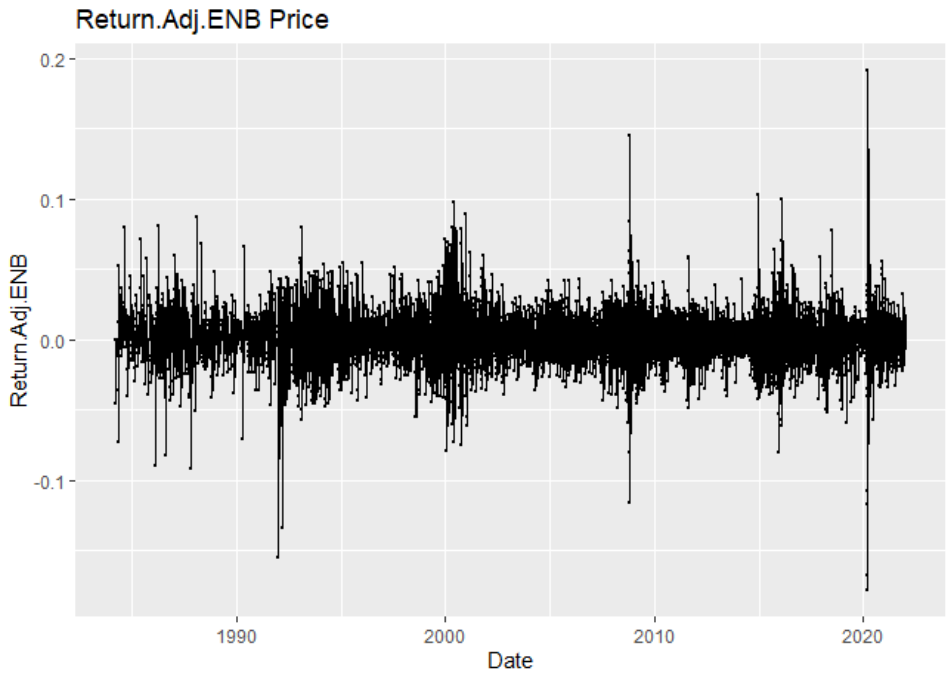


For SPXs price, starting in the year 2000, we see a downward trend, a break soon after trending upwards, then a break down and another final break when it linearly trends upward from 2010 to 2020.

For all our returns, they all look like stationary sideways trends with a lot of variances. It also looks a lot like random noise.

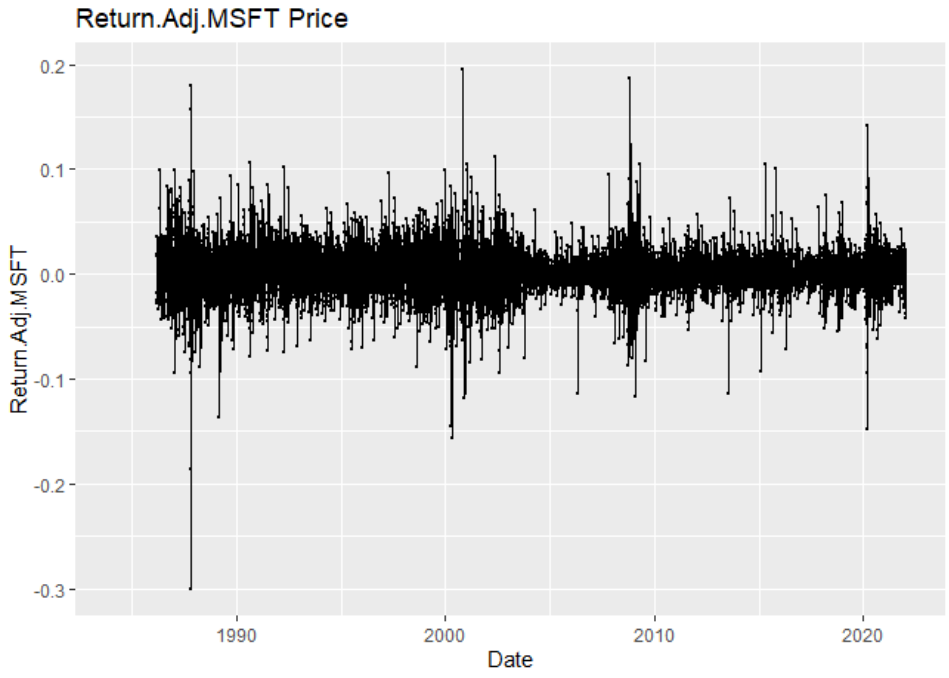
Plot 3.5

Time Series Plot of ENBs Returns



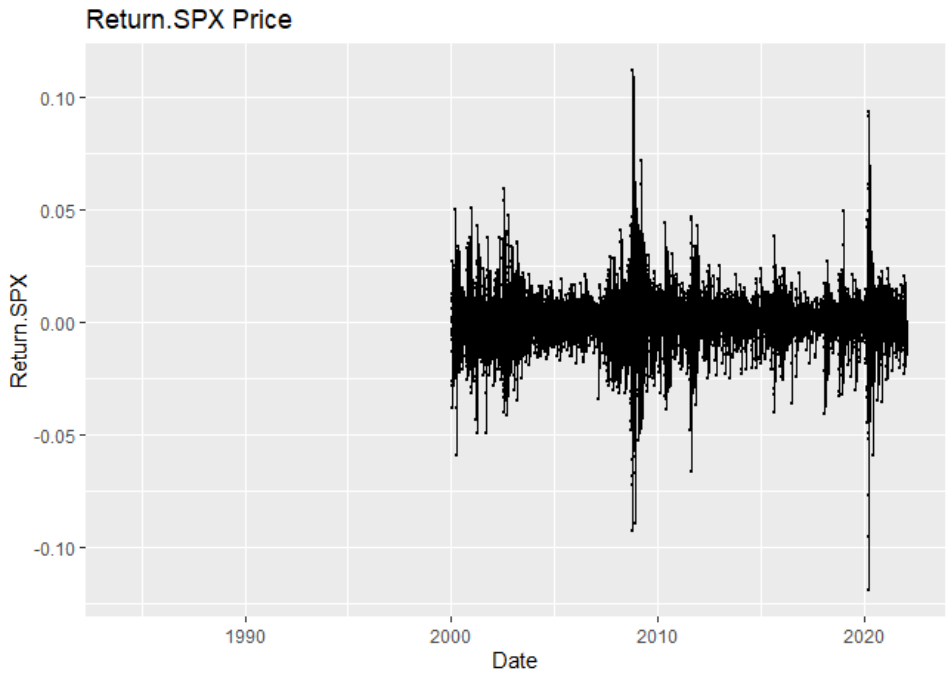
Plot 3.6

Time Series Plot of MSFTs Returns



Plot 3.7

Time Series Plot of SPX Returns



Our Unit Test Results and Model Selection

Given the results in tables 3.2 to 3.4 they indicate for all models based on close or adjusted close prices for ENB, MSFT, and SPX all fail to rule out a unit root and thus it's likely a unit root exists. For our models based on returns for ENB, MSFT, and SPX we see successful rejection of the null indicating they are stationary.

Based on our observations and unit root tests, for our returns we should just stick to a ARMA models since our data is already stationary. While for our prices we should stick with ARIMA and ARIMA + intercept models to insure we can integrate and difference the data enough to achieve stationarity.

Using the formula in figure 3.1, we found the respective lags for each of our prices and returns which is summarized in table 3.1. Table 3.2 contains our results using not only the max lags, but also various lags at 1,7, and 40 for robustness. Furthermore, the AIC picked a lag of 1 for each time series.

Figure 3.1
Max Lag Formula for Dickey Fuller Tests

$$p_{xmax} = \left[12 * (T/100)^{1/4}\right]$$

Table 3.1
Max Lags for Each Price and Return Column

Time Series	Max Lag
Close.Adj.ENB	37
Close.Adj.MSFT	36
Close.SPX	32
Return.Adj.ENB	37
Return.Adj.MSFT	36
Return.SPX	32

Table 3.2
Augmented Dickey Fuller Tests for Price and Return – Model One

Time Series	Lag	Model	Type	Test	Critical Value	Reject Null?
Close.Adj.ENB	1	1	none	1.36001	-1.616	No
Close.Adj.ENB	7	1	none	1.31819	-1.616	No
Close.Adj.ENB	37	1	none	1.88565	-1.616	No
Close.Adj.ENB	40	1	none	1.78466	-1.616	No
Close.Adj.ENB	1	1	none	1.36001	-1.616	No
Close.Adj.MSFT	1	1	none	6.92935	-1.616	No
Close.Adj.MSFT	7	1	none	6.64451	-1.616	No
Close.Adj.MSFT	37	1	none	9.36290	-1.616	No
Close.Adj.MSFT	40	1	none	7.97982	-1.616	No
Close.Adj.MSFT	1	1	none	6.92935	-1.616	No
Close.SPX	1	1	none	3.23633	-1.616	No
Close.SPX	7	1	none	3.07026	-1.616	No
Close.SPX	37	1	none	3.63903	-1.616	No

Close.SPX	40	1	none	3.31227	-1.616	No
Close.SPX	1	1	none	3.23633	-1.616	No
Return.Adj.ENB	1	1	none	-73.17788	-5.567	Yes, at 1%
Return.Adj.ENB	7	1	none	-36.59183	-5.567	Yes, at 1%
Return.Adj.ENB	37	1	none	-16.20969	-5.567	Yes, at 1%
Return.Adj.ENB	40	1	none	-15.46604	-5.567	Yes, at 1%
Return.Adj.ENB	1	1	none	-73.17788	-5.567	Yes, at 1%
Return.Adj.MSFT	1	1	none	-70.11090	-5.567	Yes, at 1%
Return.Adj.MSFT	7	1	none	-34.45383	-5.567	Yes, at 1%
Return.Adj.MSFT	37	1	none	-15.22321	-5.567	Yes, at 1%
Return.Adj.MSFT	40	1	none	-14.39370	-5.567	Yes, at 1%
Return.Adj.MSFT	1	1	none	-70.11090	-5.567	Yes, at 1%
Return.SPX	1	1	none	-56.36348	-5.567	Yes, at 1%
Return.SPX	7	1	none	-28.35244	-5.567	Yes, at 1%
Return.SPX	37	1	none	-13.55666	-5.567	Yes, at 1%
Return.SPX	40	1	none	-11.67993	-5.567	Yes, at 1%
Return.SPX	1	1	none	-56.36348	-5.567	Yes, at 1%

*Critical values are based on the critical value tables for Dickey Fuller and Models which can be found in our appendix under tables 3.4 to 3.6

*AIC picked a lag of 1 for each time series.

Table 3.3

Augmented Dickey Fuller Tests for Price and Return – Model Two

Time Series	Lag	Model	Type	Test	Critical Value	Reject Null?
Close.Adj.ENB	1	2	drift	0.33099	-2.568	No
Close.Adj.ENB	7	2	drift	0.29890	-2.568	No
Close.Adj.ENB	37	2	drift	0.73798	-2.568	No
Close.Adj.ENB	40	2	drift	0.66369	-2.568	No
Close.Adj.ENB	1	2	drift	0.33099	-2.568	No
Close.Adj.MSFT	1	2	drift	6.06657	-2.568	No
Close.Adj.MSFT	7	2	drift	5.83384	-2.568	No
Close.Adj.MSFT	37	2	drift	8.69977	-2.568	No
Close.Adj.MSFT	40	2	drift	7.39749	-2.568	No
Close.Adj.MSFT	1	2	drift	6.06657	-2.568	No
Close.SPX	1	2	drift	2.59896	-2.568	No
Close.SPX	7	2	drift	2.45434	-2.568	No
Close.SPX	37	2	drift	3.13164	-2.568	No
Close.SPX	40	2	drift	2.79560	-2.568	No
Close.SPX	1	2	drift	2.59896	-2.568	No

Return.Adj.ENB	1	2	drift	-73.38001	-3.434	Yes, at 1%
Return.Adj.ENB	7	2	drift	-36.94380	-3.434	Yes, at 1%
Return.Adj.ENB	37	2	drift	-17.08180	-3.434	Yes, at 1%
Return.Adj.ENB	40	2	drift	-16.35830	-3.434	Yes, at 1%
Return.Adj.ENB	1	2	drift	-73.38001	-3.434	Yes, at 1%
Return.Adj.MSFT	1	2	drift	-70.43679	-3.434	Yes, at 1%
Return.Adj.MSFT	7	2	drift	-35.00789	-3.434	Yes, at 1%
Return.Adj.MSFT	37	2	drift	-16.29594	-3.434	Yes, at 1%
Return.Adj.MSFT	40	2	drift	-15.52256	-3.434	Yes, at 1%
Return.Adj.MSFT	1	2	drift	-70.43679	-3.434	Yes, at 1%
Return.SPX	1	2	drift	-56.41868	-3.434	Yes, at 1%
Return.SPX	7	2	drift	-28.44980	-3.434	Yes, at 1%
Return.SPX	37	2	drift	-13.77476	-3.434	Yes, at 1%
Return.SPX	40	2	drift	-11.91506	-3.434	Yes, at 1%
Return.SPX	1	2	drift	-56.41868	-3.434	Yes, at 1%

*Critical values are based on the critical value tables for Dickey Fuller and Models which can be found in our appendix under tables 3.4 to 3.6

*AIC picked a lag of 1 for each time series.

Table 3.4

Augmented Dickey Fuller Tests for Price and Return – Model Three

Time Series	Lag	Model	Type	Test	Critical Value	Reject Null?
Close.Adj.ENB	1	3	trend	-2.06073	-3.128	No
Close.Adj.ENB	7	3	trend	-2.06073	-3.128	No
Close.Adj.ENB	37	3	trend	-2.06073	-3.128	No
Close.Adj.ENB	40	3	trend	-2.06073	-3.128	No
Close.Adj.ENB	1	3	trend	-2.06073	-3.128	No
Close.Adj.MSFT	1	3	trend	4.11842	-3.128	No
Close.Adj.MSFT	7	3	trend	3.97211	-3.128	No
Close.Adj.MSFT	37	3	trend	6.90258	-3.128	No
Close.Adj.MSFT	40	3	trend	5.80365	-3.128	No
Close.Adj.MSFT	1	3	trend	4.11842	-3.128	No
Close.SPX	1	3	trend	0.01875	-3.128	No
Close.SPX	7	3	trend	-0.10498	-3.128	No
Close.SPX	37	3	trend	0.45753	-3.128	No
Close.SPX	40	3	trend	0.25599	-3.128	No
Close.SPX	1	3	trend	0.01875	-3.128	No
Return.Adj.ENB	1	3	trend	-73.37621	-3.963	Yes, at 1%

Return.Adj.ENB	7	3	trend	-36.94215	-3.963	Yes, at 1%
Return.Adj.ENB	37	3	trend	-17.08374	-3.963	Yes, at 1%
Return.Adj.ENB	40	3	trend	-16.35939	-3.963	Yes, at 1%
Return.Adj.ENB	1	3	trend	-73.37621	-3.963	Yes, at 1%
Return.Adj.MSFT	1	3	trend	-70.47329	-3.963	Yes, at 1%
Return.Adj.MSFT	7	3	trend	-35.08099	-3.963	Yes, at 1%
Return.Adj.MSFT	37	3	trend	-16.43917	-3.963	Yes, at 1%
Return.Adj.MSFT	40	3	trend	-15.67494	-3.963	Yes, at 1%
Return.Adj.MSFT	1	3	trend	-70.47329	-3.963	Yes, at 1%
Return.SPX	1	3	trend	-56.45471	-3.963	Yes, at 1%
Return.SPX	7	3	trend	-28.52424	-3.963	Yes, at 1%
Return.SPX	37	3	trend	-13.91038	-3.963	Yes, at 1%
Return.SPX	40	3	trend	-12.07343	-3.963	Yes, at 1%
Return.SPX	1	3	trend	-56.45471	-3.963	Yes, at 1%

*Critical values are based on the critical value tables for Dickey Fuller and Models which can be found in our appendix under tables 3.4 to 3.6

*AIC picked a lag of 1 for each time series.

GLS – ADF Tests

For our GLS – ADF tests, we see very similar results from before. However, we do see some indication of the null rejecting MSFT returns for our constant model, and SPX returns rejecting for both the constant and trend model at their max lag and at lag 40; while neither show rejection for smaller lags. This could be a possible indication of a unit root in our returns for MSFT and SPX, but all our prior results and smaller lags suggest otherwise. So, no changes will be made to our model selection from before.

Table 3.5

Generalized Least Squares Dickey Fuller Tests for Price and Return – Model Constant

Time Series	Lag	Type	Test	Critical Value	Reject Null?
Close.Adj.ENB	1	constant	1.34005	-1.62	No
Close.Adj.ENB	7	constant	1.29815	-1.62	No
Close.Adj.ENB	37	constant	1.86551	-1.62	No
Close.Adj.ENB	40	constant	1.76454	-1.62	No
Close.Adj.MSFT	1	constant	6.93102	-1.62	No
Close.Adj.MSFT	7	constant	6.64615	-1.62	No
Close.Adj.MSFT	37	constant	9.37137	-1.62	No
Close.Adj.MSFT	40	constant	7.98691	-1.62	No
Close.SPX	1	constant	3.24017	-1.62	No
Close.SPX	7	constant	3.05659	-1.62	No
Close.SPX	37	constant	3.75492	-1.62	No
Close.SPX	40	constant	3.35687	-1.62	No
Return.Adj.ENB	1	constant	-73.17894	*-2.57*	Yes, at 1%
Return.Adj.ENB	7	constant	-36.59362	*-2.57*	Yes, at 1%
Return.Adj.ENB	37	constant	-16.21379	*-2.57*	Yes, at 1%
Return.Adj.ENB	40	constant	-15.47023	*-2.57*	Yes, at 1%
Return.Adj.MSFT	1	constant	-23.20367	*-2.57*	Yes, at 1%
Return.Adj.MSFT	7	constant	-6.54752	*-2.57*	Yes, at 1%
Return.Adj.MSFT	37	constant	-1.34442	*-1.62*	No
Return.Adj.MSFT	40	constant	-1.20066	*-1.62*	No
Return.SPX	1	constant	-9.56862	*-2.57*	Yes, at 1%
Return.SPX	7	constant	-2.66744	*-2.57*	Yes, at 1%
Return.SPX	37	constant	-0.56856	*-1.62*	No
Return.SPX	40	constant	-0.46243	*-1.62*	No

**Critical values for this model are -2.57, -1.94, -1.62 for the respective 1, 5, and 10% significance levels.*

Table 3.6

Generalized Least Squares Dickey Fuller Tests for Price and Return – Model Trend

Time Series	Lag	Type	Test	Critical Value	Reject Null?
Close.Adj.ENB	1	trend	-1.26209	-2.57	No
Close.Adj.ENB	7	trend	-1.29317	-2.57	No
Close.Adj.ENB	37	trend	-0.90539	-2.57	No
Close.Adj.ENB	40	trend	-0.96594	-2.57	No
Close.Adj.MSFT	1	trend	2.43626	-2.57	No
Close.Adj.MSFT	7	trend	2.24428	-2.57	No

Close.Adj.MSFT	37	trend	3.06897	-2.57	No
Close.Adj.MSFT	40	trend	2.28332	-2.57	No
Close.SPX	1	trend	0.47468	-2.57	No
Close.SPX	7	trend	0.36993	-2.57	No
Close.SPX	37	trend	0.67485	-2.57	No
Close.SPX	40	trend	0.46234	-2.57	No
Return.Adj.ENB	1	trend	-73.26834	*-3.48*	Yes, at 1%
Return.Adj.ENB	7	trend	-36.74408	*-3.48*	Yes, at 1%
Return.Adj.ENB	37	trend	-16.56261	*-3.48*	Yes, at 1%
Return.Adj.ENB	40	trend	-15.83028	*-3.48*	Yes, at 1%
Return.Adj.MSFT	1	trend	-40.67359	*-3.48*	Yes, at 1%
Return.Adj.MSFT	7	trend	-12.70763	*-3.48*	Yes, at 1%
Return.Adj.MSFT	37	trend	-3.12898	*-2.89*	Yes, at 10%
Return.Adj.MSFT	40	trend	-2.88541	*-2.57*	Yes, at 5%
Return.SPX	1	trend	-18.45271	*-3.48*	Yes, at 1%
Return.SPX	7	trend	-5.41723	*-3.48*	Yes, at 1%
Return.SPX	37	trend	-1.77243	*-2.57*	No
Return.SPX	40	trend	-1.75258	*-2.57*	No

**Critical values for this model are -3.48, -2.89, -2.57 for the respective 1, 5, and 10% significance levels.*

Code for Section Three

Figures 3.2, 3.3, and 3.4 cover code for adjusted dickey and generalized least squares dickey fuller tests. Figure 3.4 shows an example of how to call/use functions and the next steps in returning a readable table.

Figure 3.2

Function for Dickey Fuller Test

```
Run_DF_Test = function(VectorName, lags, type) {
  # This function wraps around the ur.af function from urca

  # Variables
  Vector = CombinedData %>% select(VectorName) # Grab all my prices and returns
  Vector = Vector[!is.na(Vector), ] # omit na rows.

  # Create Empty List for Results
  ADF_Tests = vector(mode = "list", length(lags)+1)

  # Run Dickey Fuller Tests for Each Vector
  for (i in 1:length(lags)) {
    ADF_Tests[[i]] = ur.df(Vector, lags = lags[i], type = type, selectlags = c("Fixed"))
  }

  # Append Dickey Fuller test with lags selected by AIC
  ADF_Tests[[length(ADF_Tests)]] =  ur.df(Vector, type = type, selectlags = c("AIC"))

  # End of Function
  return(ADF_Tests)
}
```

Figure 3.3

Function for Generalized Least Squares Dickey Fuller Tests

```
Run_DF_Test_GLS = function(VectorName, lags, type) {
  # This function wraps around the ur.ers function from urca, this is to do GLS-ADF test

  # Variables
  Vector = CombinedData %>% select(VectorName)# Grab all my prices and returns
  Vector = Vector[!is.na(Vector), ] # omit na rows.

  # Create Empty List for Results
  ADF_Tests = vector(mode = "list", length(lags))

  # Run Dickey Fuller Tests for Each Vector
  for (i in 1:length(lags)) {
    ADF_Tests[[i]] = ur.ers(Vector, lag.max = lags[i], type = c("DF-GLS"), model = type)
  }

  # End of Function
  return(ADF_Tests)
}
```

Figure 3.4

Example for Running and Returning Dickey Fuller Results

```
A3.ADF_Tests = mapply(function(x,y) {Run_DF_Test(VectorName = x
                                                 , lags =  c(1,7,y,40)
                                                 , type = "none")
}, x = VectorNames
, y = A3.Max.Lags
, SIMPLIFY = FALSE)

A3.ADF_Tests1 = lapply(A3.ADF_Tests, function(x) mapply(function(y) summary(y)@teststat, y = x))

A3.ADF_Tests_TValues1 = do.call(rbind, A3.ADF_Tests1) %>% data.frame
```

**Use mapply() to pass in vectors for vectors and max lags for each price/return. Then index and return summary results using lapply() nested into mapply(). Finally, bind list into a data.frame for export.*

Section Four – Optimal ARIMA Models

Max Lags

For our autocorrelation (ACF) and partial autocorrelation (PACF) tests, we display the max lags in Table 4.1, using equation (1) for vectors:

$$h_{max} = [10 * LogN] \quad (1)$$

Table 4.1
Max lags calculated for each named vector

Column Name	Max_Lags
Close.Adj.ENB	91
Close.Adj.MSFT	91
Close.SPX	86
Return.Adj.ENB	91
Return.Adj.MSFT	91
Return.SPX	86

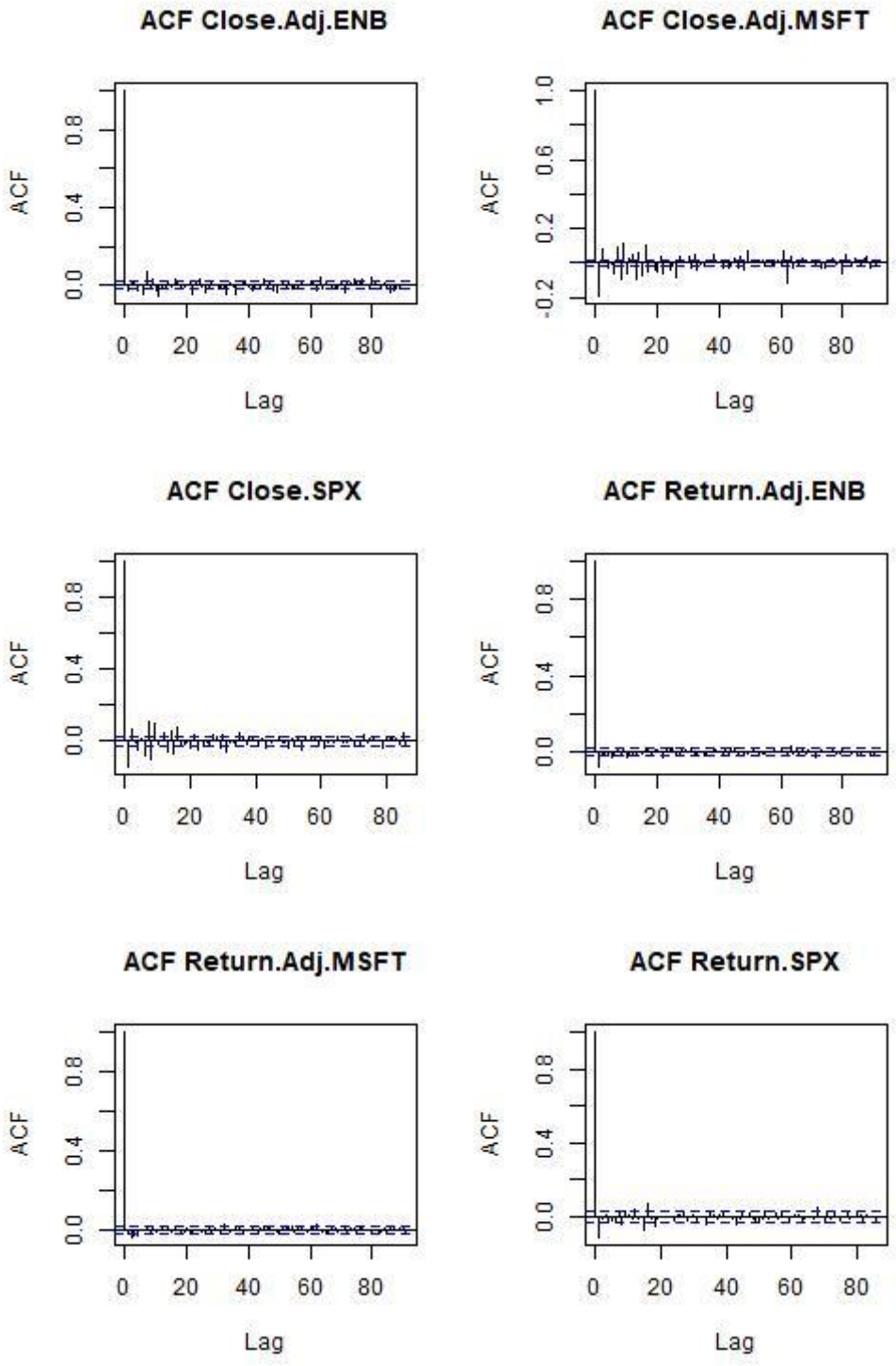
Model Selection

As previously found, there are unit roots present in our closing price vectors. For this reason, we have first differenced these for our ACF, PACF, and Ljung tests, displayed in the below exhibits. In Exhibit 4.1, we see that many of the lags do find themselves outside of the confidence bands. The term when h=0 is included to display how minimal the autocorrelation is, despite being outside of the bands on some occasions. In Exhibit 4.2 we put a limit on the y-axis to better view the ACF. Exhibit 4.3 is similar to 4.1, however we plot the PACF rather than ACF. In Exhibit 4.4 we use a limit on the x-axis to only include the first 10 lags. We can see some sort of trend during the first 5 days, with some oscillation afterwards. This indicates we may have a seasonal trend and will need to be further investigated, with additional analysis discussed below in Exhibits 4.13 and 4.14 and Tables 4.7 and 4.7.

Exhibits 4.5 through 4.12 use the optimal models as selected using Akaike Information Criteria (AIC) or Bayesian Information Criteria (BIC). We see that these models have improved results from those in Exhibits 4.1 to 4.4. Despite this, we still view some lags that reach outside of the confidence bands.

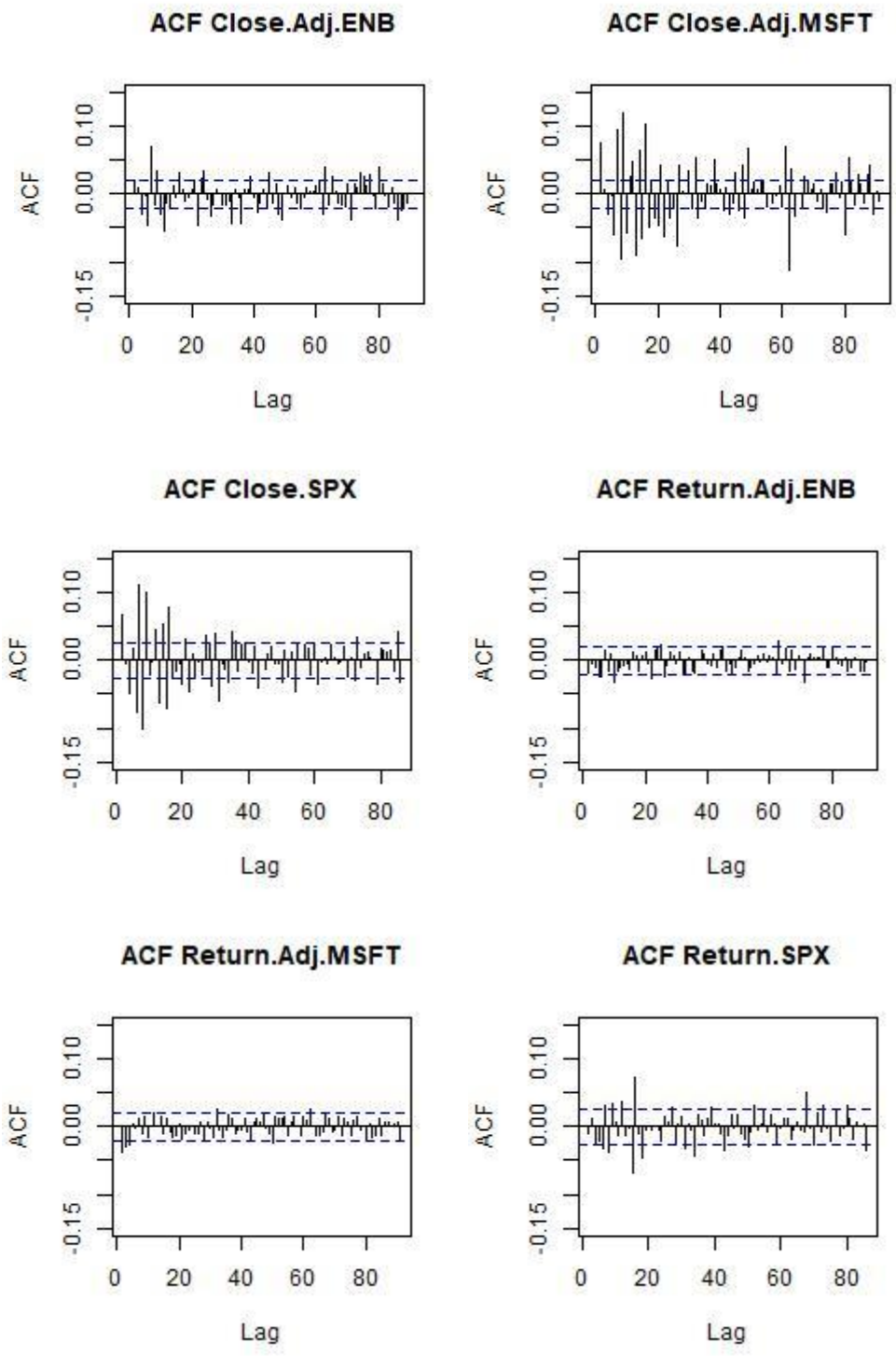
Exhibit 4.1

ACF Plots for Various Columns



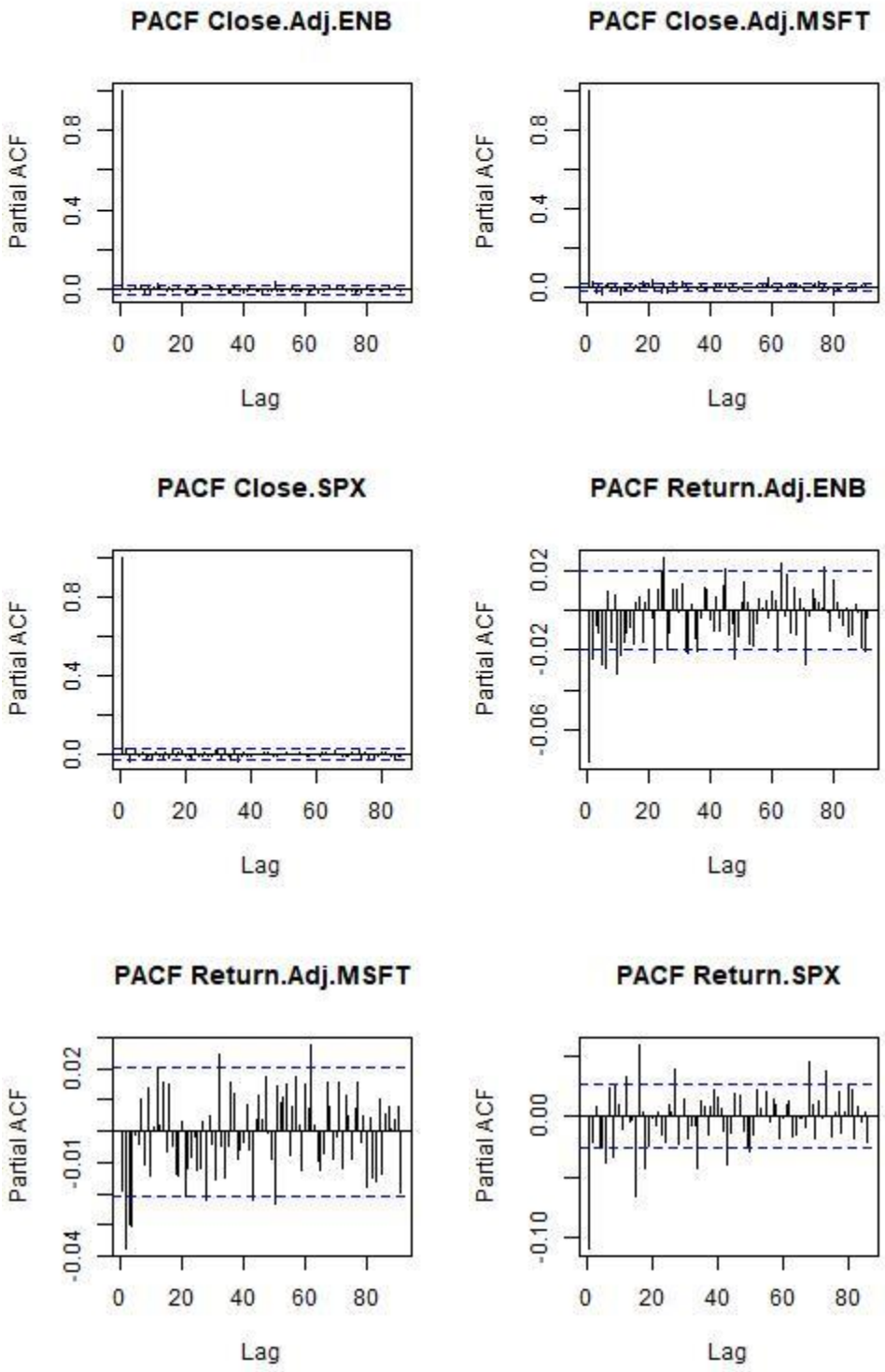
*These are ACF tests on our asset returns and asset close prices that have been first differenced to find visually indications of unit root.

Exhibit 4.2
ACF Plots for Various Columns with a y limit



*These are ACF tests on our asset returns and asset close prices that have been first differenced to find visually indications of unit root. There has also been an adjustment to the scale.

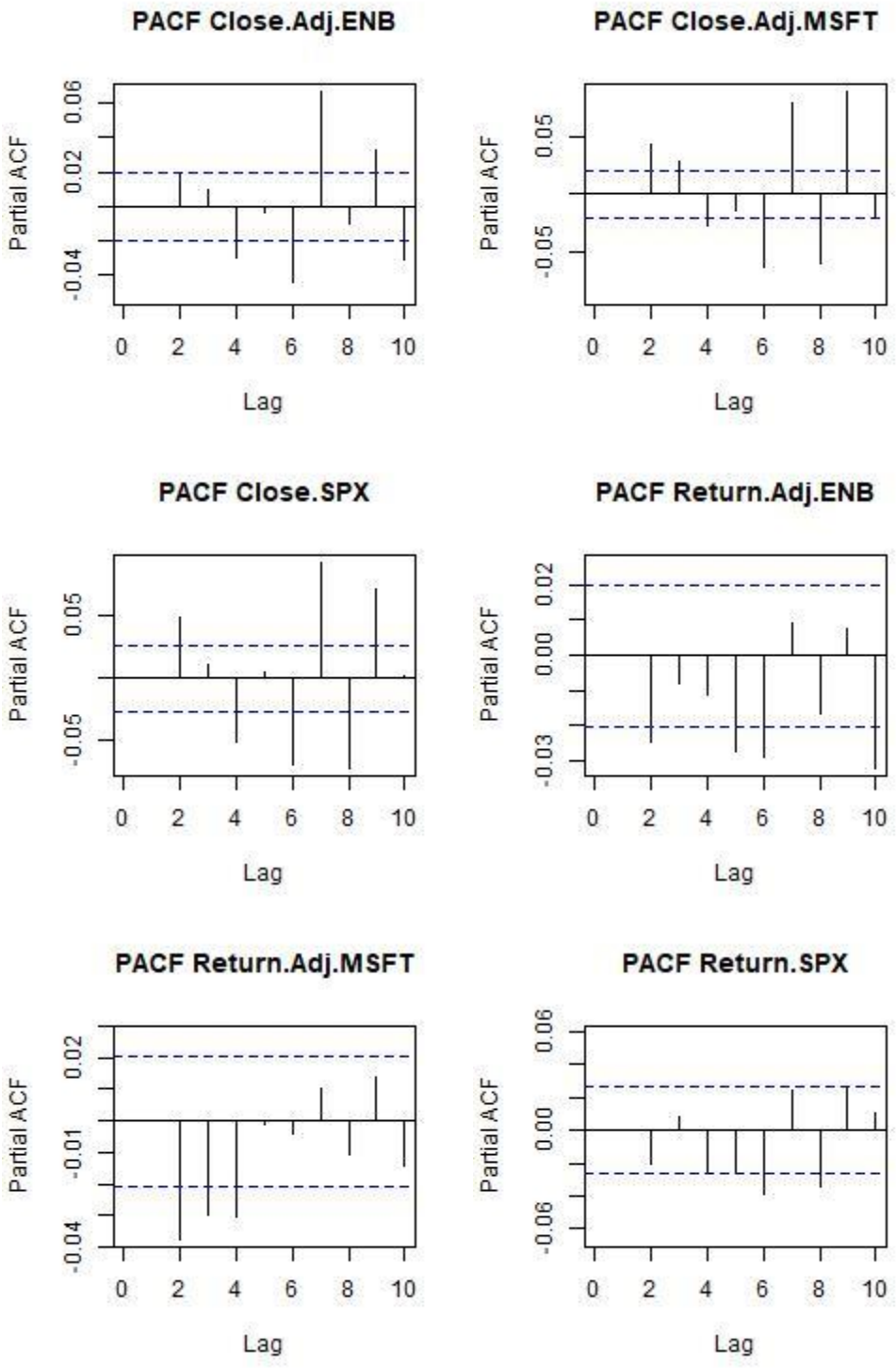
Exhibit 4.3
PACF Plots for Various Columns



*These are PACF tests on our asset returns and asset close prices that have been first differenced to find visually indications of unit root.

Exhibit 4.4

PACF Plots for Various Columns with a x limit



*These are PACF tests on our asset returns and asset close prices that have been first differenced to find visually indications of unit root. These have an adjustment to their scale

Table 4.2

Ljung P Values for our Returns and First Differenced Prices

Lag	Close.Adj.ENB	Close.Adj.MSFT	Close.SPX	Return.Adj.ENB	Return.Adj.MSFT	Return.SPX
1	0.007017	0	0	1.35E-13	0.0724	6.66E-16
2	0.004933	0	0	2.46E-13	0.0004	5.88E-15
3	0.010392	0	0	1.38E-12	4.28E-05	2.64E-14
4	0.000525	0	0	4.44E-12	5.58E-06	1.95E-14
5	0.001295	0	0	1.01E-12	1.64E-05	2.93E-14

The results from table 4.2 suggest that we can reject our null hypothesis that no autocorrelation exists for the first 5 lags: most p-values are close to zero or are less than 5%. The exception to this is for Return.Adj.MSFT as Lag 1 has a p-value of 7%, meaning the lowest level of confidence which we can reject this would be at 93%. This however would not be a problem if we aim to only test at 90%, meaning there are some circumstances which all 5 lags would allow us to reject the null hypothesis.

Table 4.3

AIC and BIC Tests for our Returns and First Differenced Prices

VectorNames	AIC	P D Q	BIC	P D Q
Close.Adj.ENB	1180.915	5 1 4	1245.843	4 1 4
Close.Adj.MSFT	27573.82	5 1 4	27644.91	5 1 4
Close.SPX	49367.88	5 1 4	49431.35	4 1 4
Return.Adj.ENB	-52775	5 0 4	-52731.2	1 0 1
Return.Adj.MSFT	-43936.5	2 0 1	-43904.2	1 0 1
Return.SPX	-32904.3	5 0 5	-32845	1 0 1

Optimal Model Selection

Twenty-five variations of ARIMA models were tested for each named vector, with P and Q orders ranging from 1 to 5. Table 4.3 displays the best model that was found when using AIC or BIC tests. Generally, the closing price models fit best to an ARIMA(5,1,4) model, or a very close variation: each of the six models given are only differenced once, and two models only differ such that they take

“P” order of 4 instead of 5. Due to previously established stationarity, the return models were best fitted for an ARMA model. The BIC tests established that ARMA(1,1) models were the most appropriate, whereas the AIC test gave a greater selection of models, with “P” orders ranging from 2 to 5, and “Q” orders ranging from 1 to 5.

Exhibit 4.5

Optimal ARIMA Models chosen using AIC, ACF Plots for Prices and Returns.

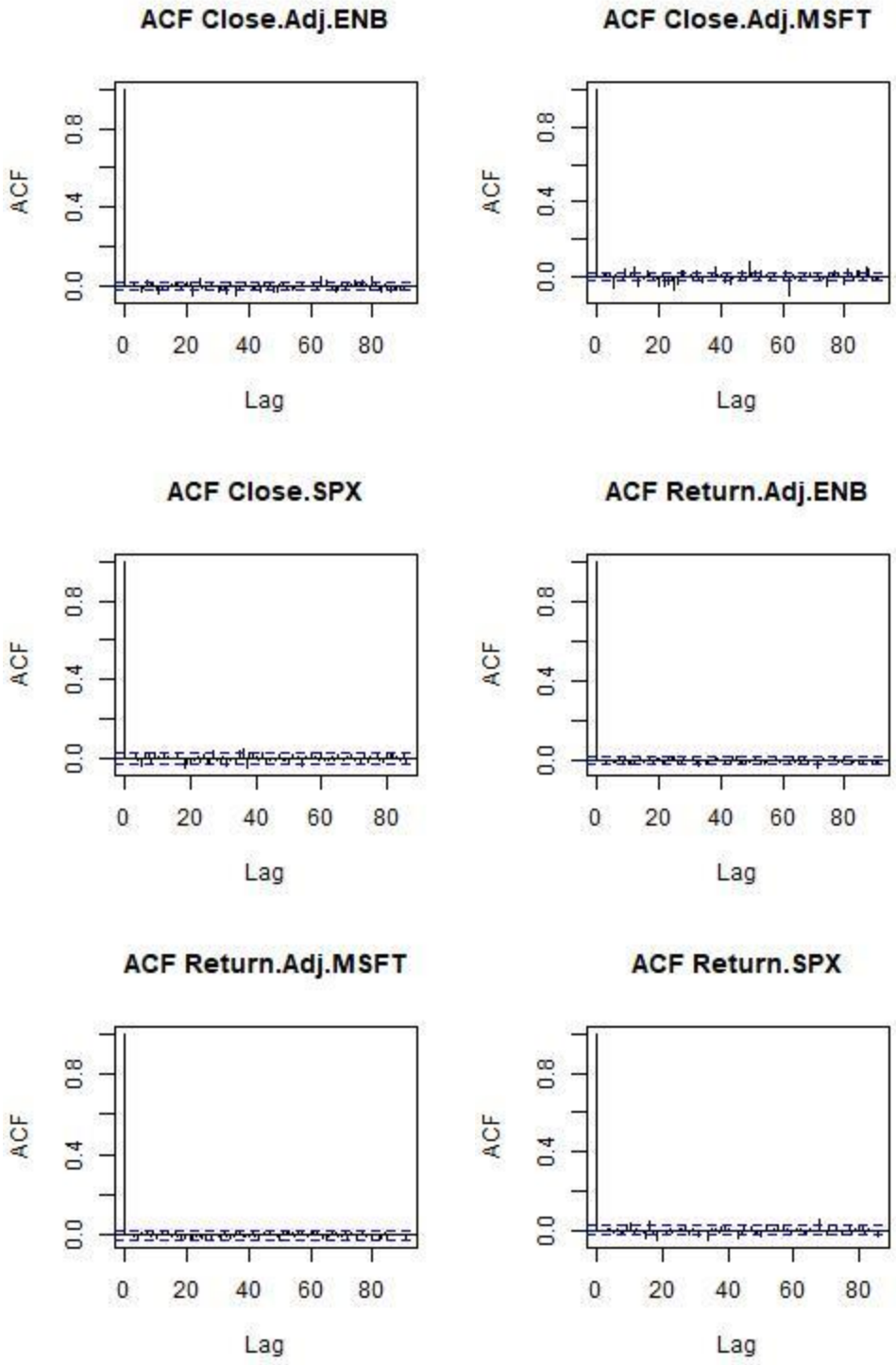
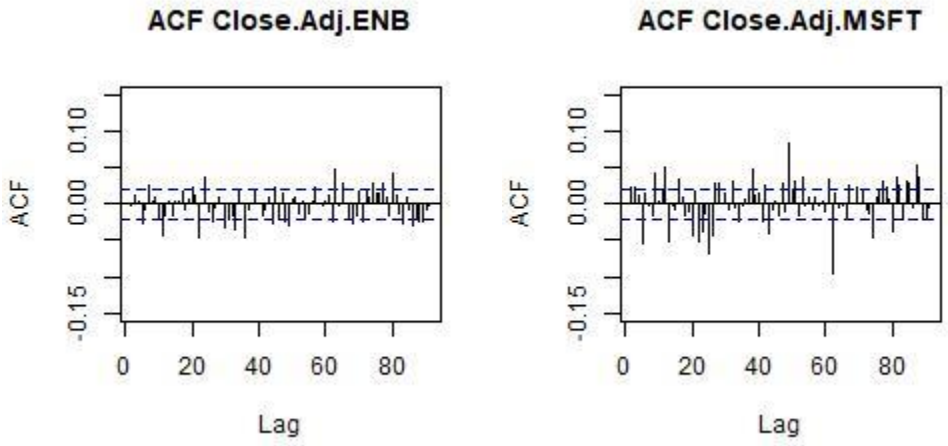
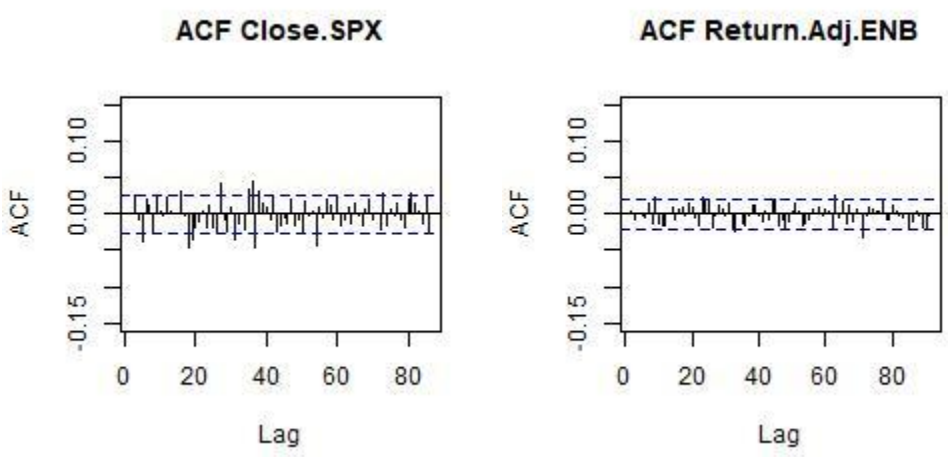


Exhibit 4.6

Optimal ARIMA Models chosen using AIC, ACF Plots for Prices and Returns, with scaled down y axis





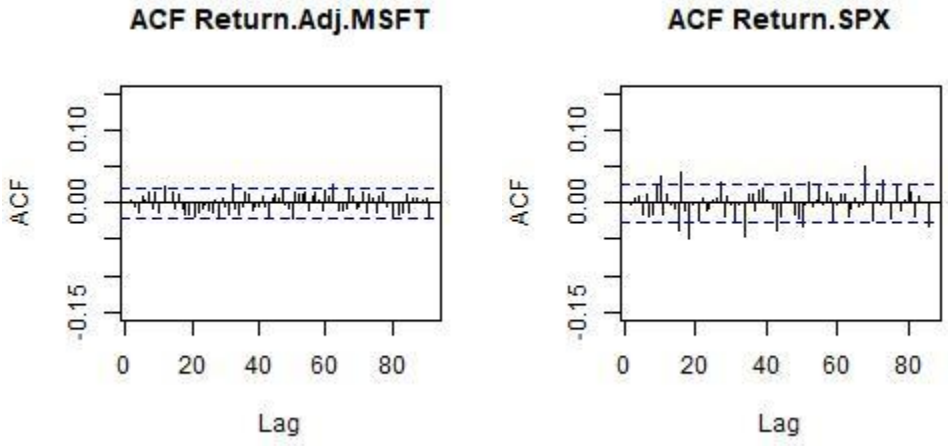
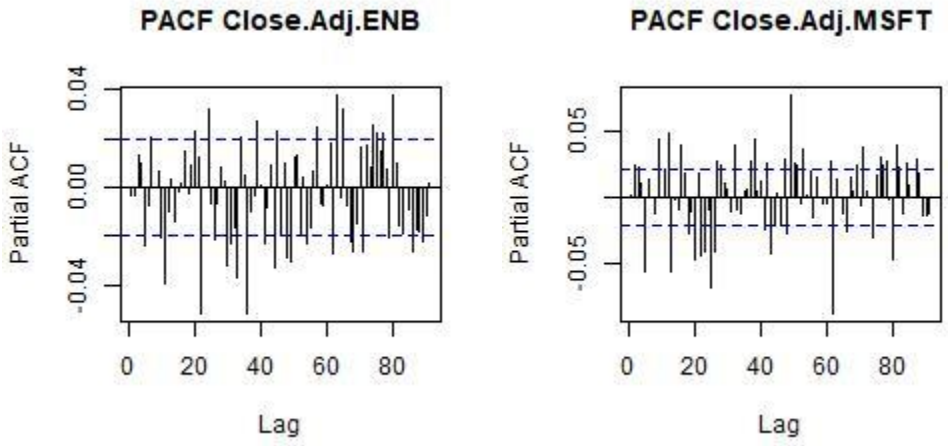
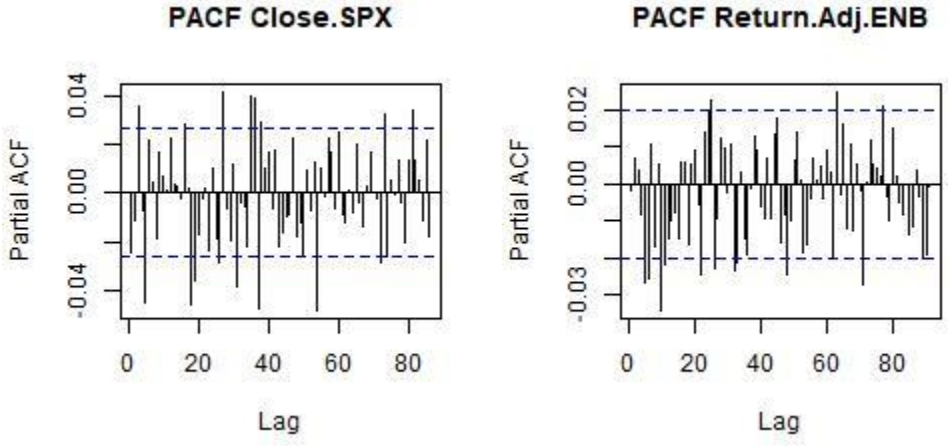


Exhibit 4.7

Optimal ARIMA Models chosen using AIC, PACF Plots for Prices and Returns.





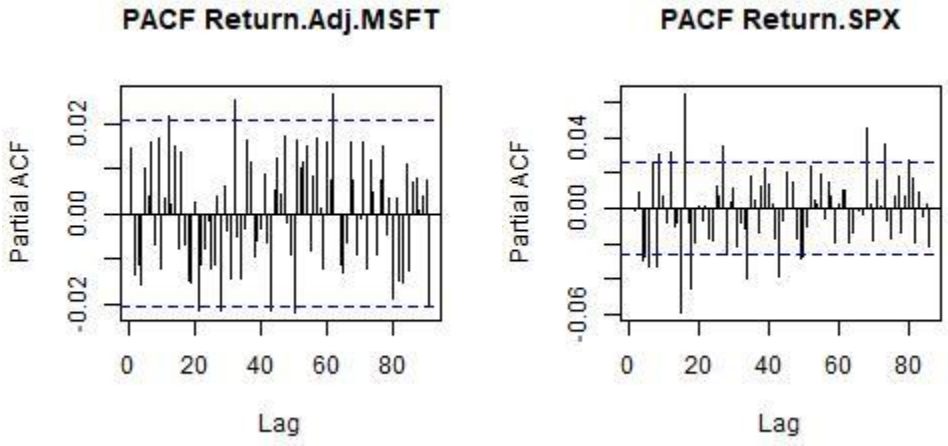
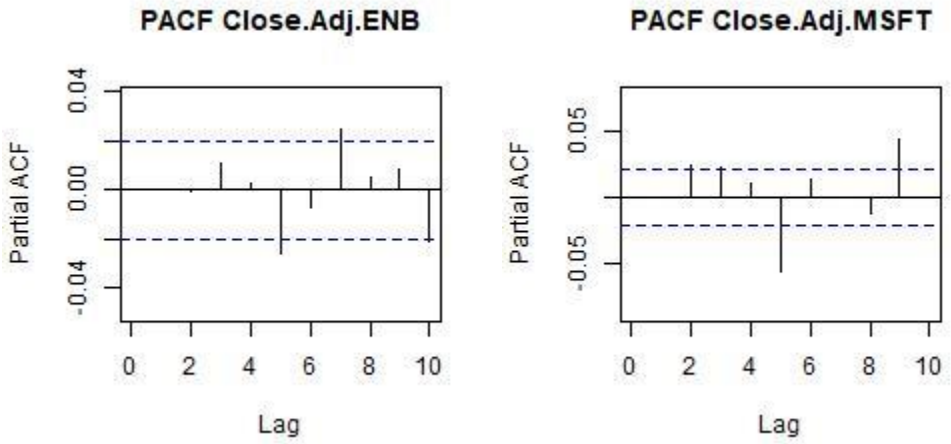
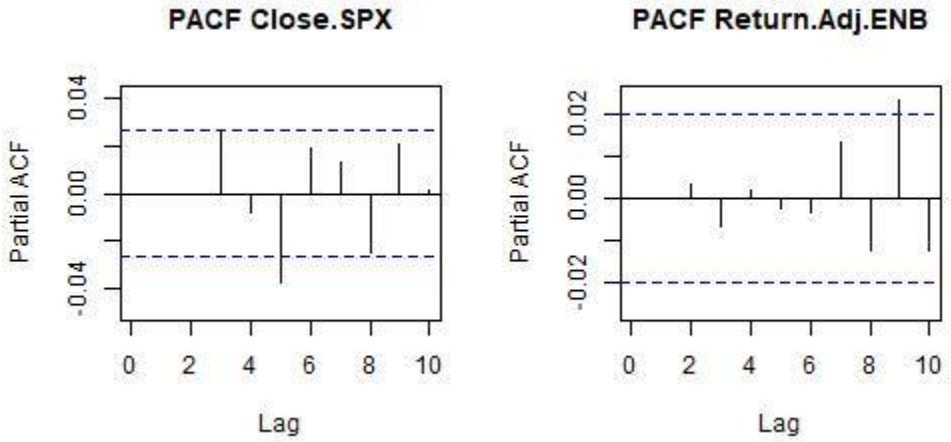


Exhibit 4.8

Optimal ARIMA Models chosen using AIC, PACF Plots for Prices and Return, with x axis scaled down.





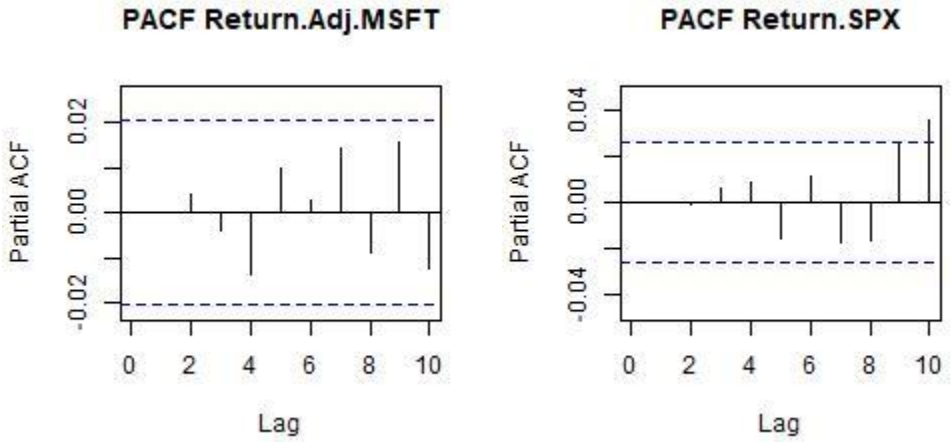


Exhibit 4.9

Optimal ARIMA Models chosen using BIC, ACF Plots for Prices and Return.

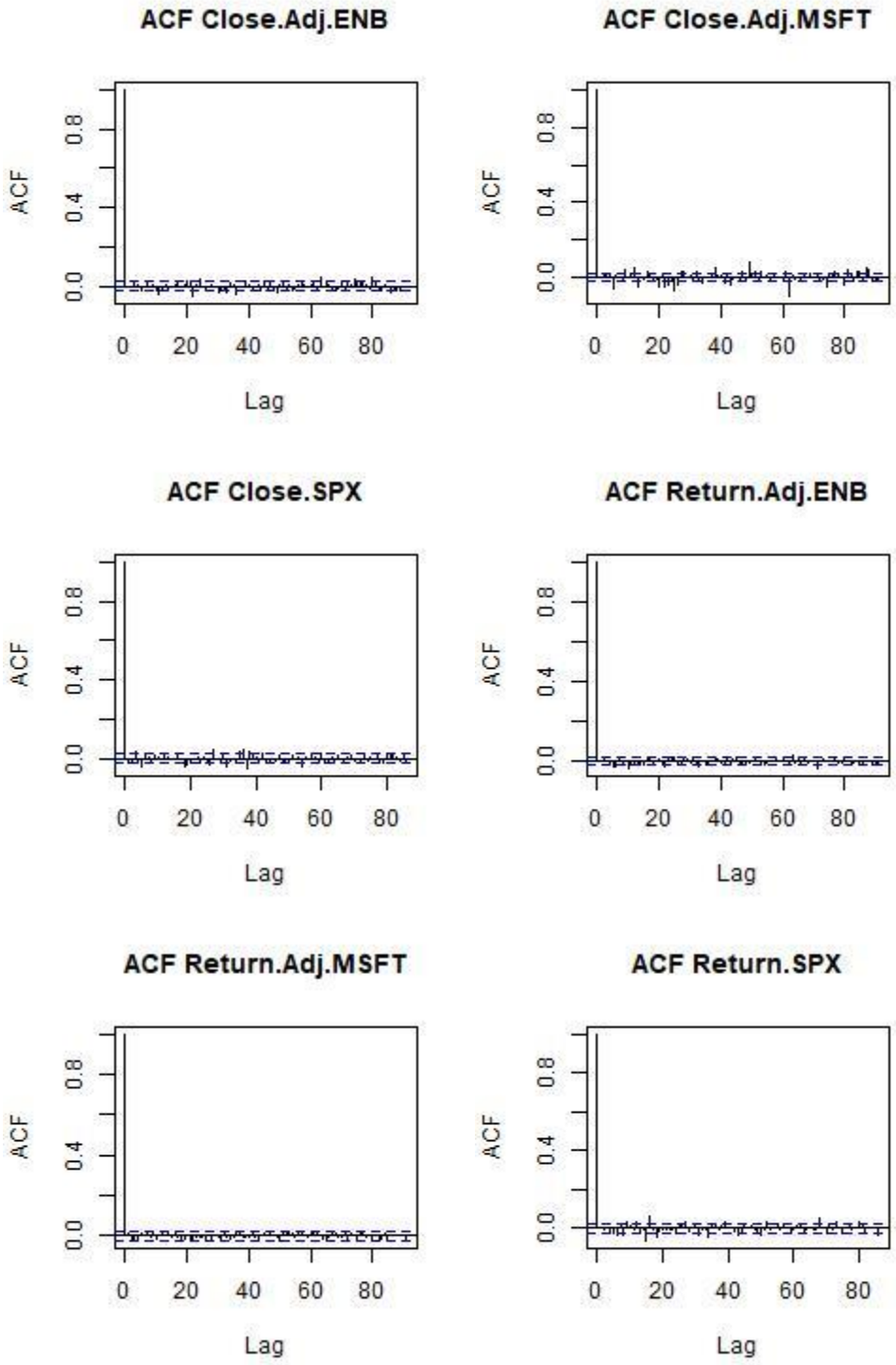
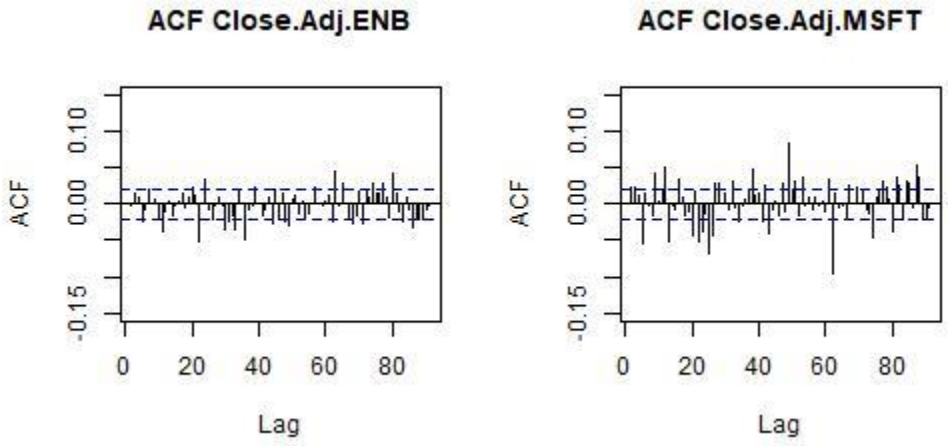
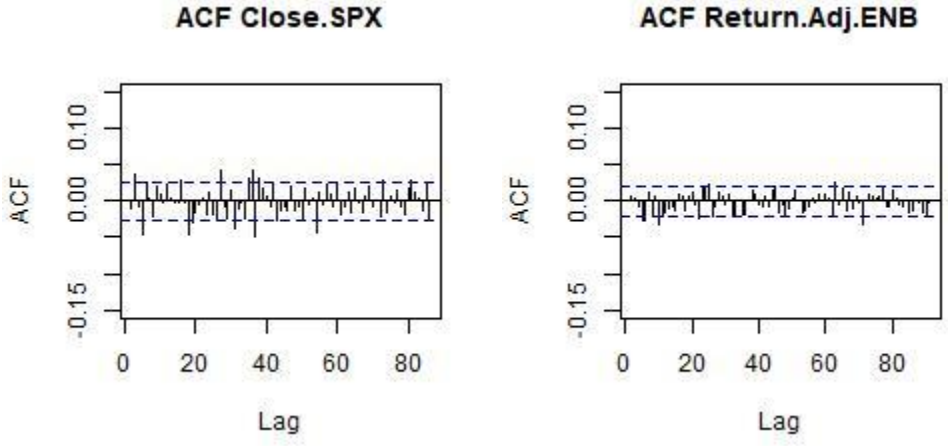


Exhibit 4.10

Optimal ARIMA Models chosen using BIC, ACF Plots for Prices and Return, with y axis scaled down.





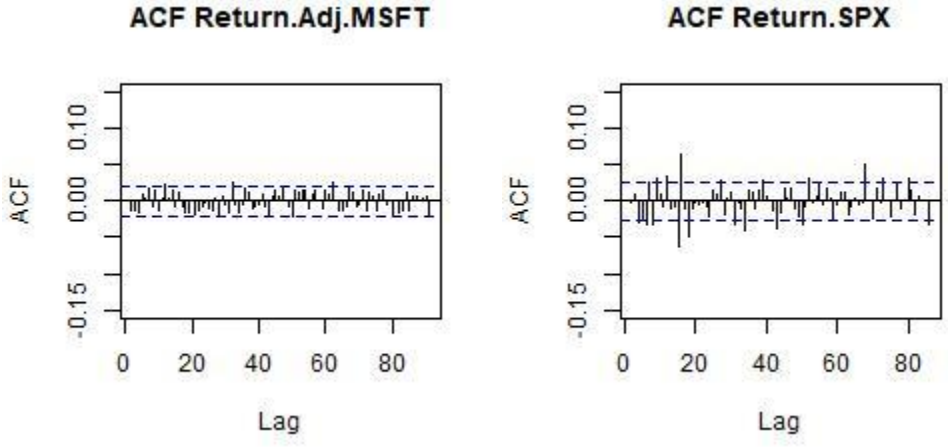


Exhibit 4.11

Optimal ARIMA Models chosen using BIC, PACF Plots for Prices and Return.

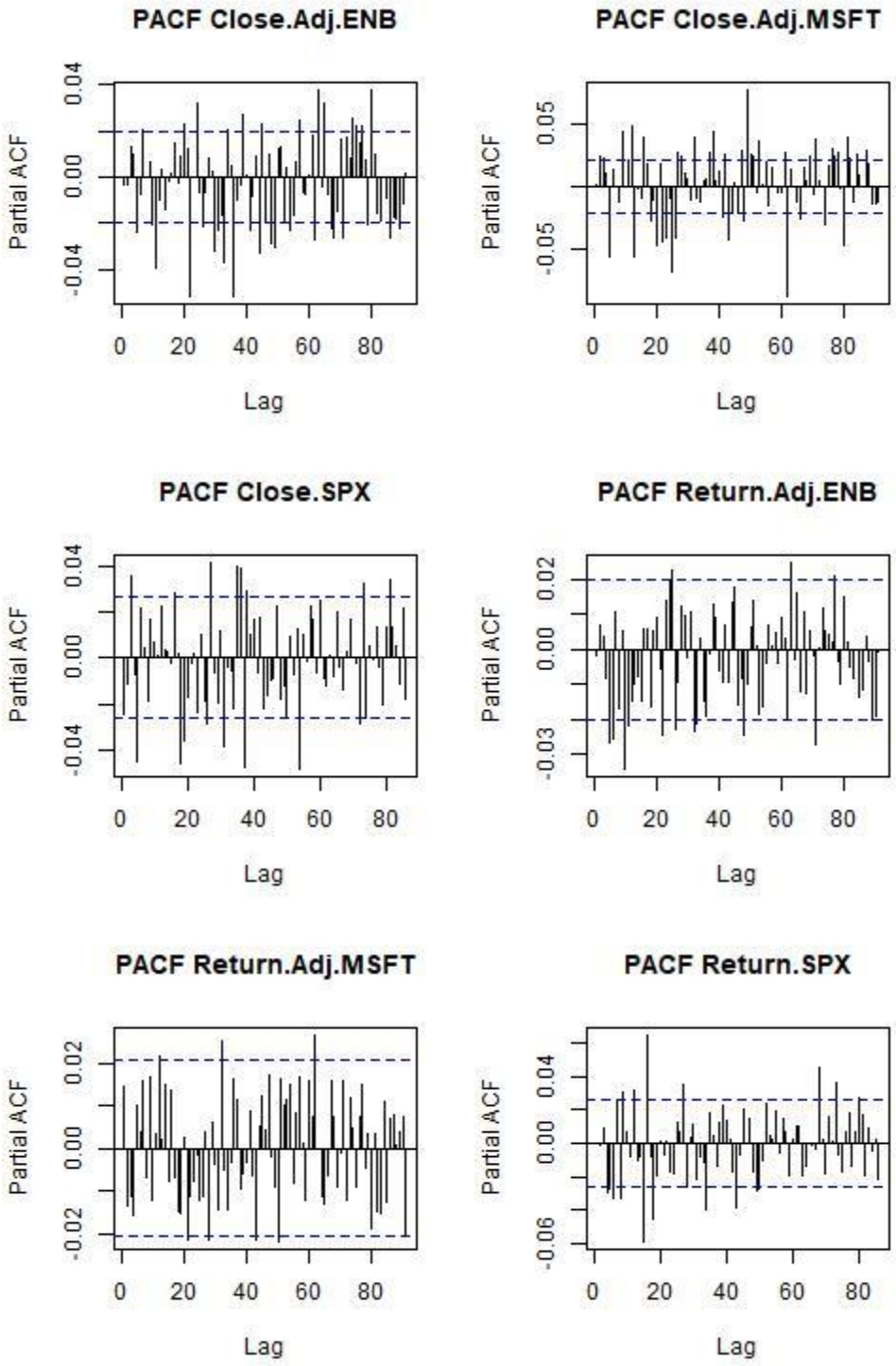
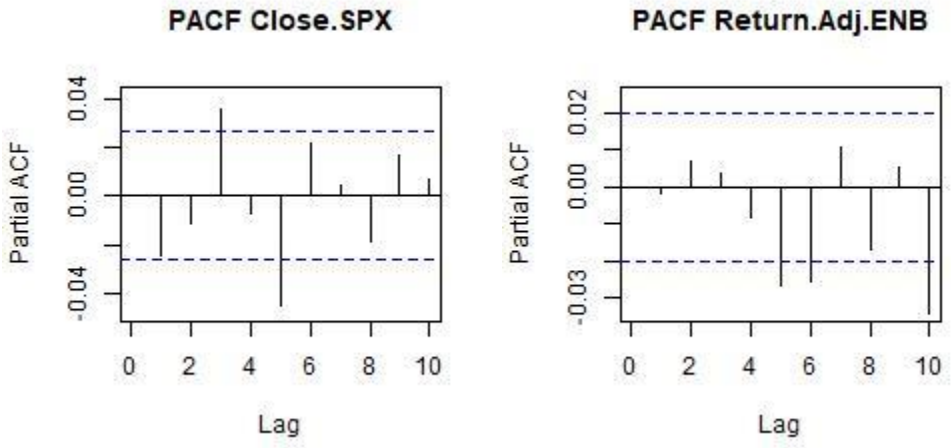
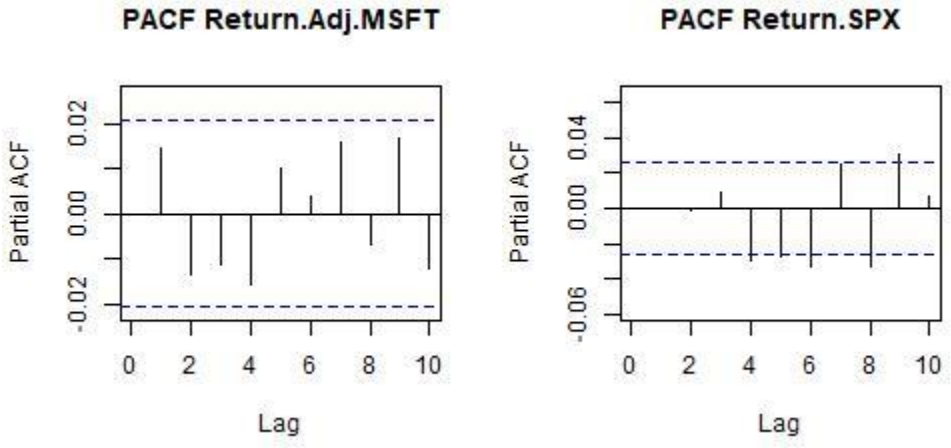


Exhibit 4.12

Optimal ARIMA Models chosen using BIC, PACF Plots for Prices and Return, with x axis scaled down.





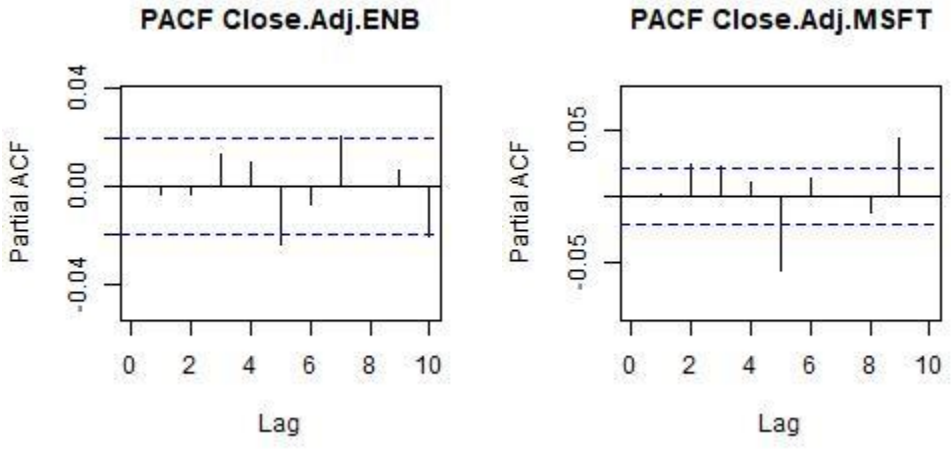


Table 4.4
Ljung Tests for optimal ARIMA models chosen using AIC

Lag	Close.Adj.ENB	Close.Adj.MSFT	Close.SPX	Return.Adj.ENB	Return.Adj.MSFT	Return.SPX
1	0.978129	0.863357	0.949243	0.909192	0.981317	0.845537
2	0.997086	0.078248	0.997938	0.936496	0.92492	0.979588
3	0.759999	0.019116	0.270353	0.908851	0.962132	0.96639
4	0.86954	0.025741	0.372865	0.966079	0.731075	0.949937
5	0.174216	2.72E-07	0.034841	0.986134	0.722529	0.840385

Table 4.5
Ljung Tests for optimal ARIMA models chosen using BIC

Lag	Close.Adj.ENB	Close.Adj.MSFT	Close.SPX	Return.Adj.ENB	Return.Adj.MSFT	Return.SPX
1	0.766203	0.863357	0.068785	0.851123	0.16162	0.986342
2	0.909726	0.078248	0.139391	0.776421	0.17308	0.99748
3	0.587968	0.019116	0.011801	0.8904	0.190263	0.933049
4	0.592479	0.025741	0.022349	0.863867	0.137248	0.275122
5	0.138062	2.72E-07	0.000392	0.16077	0.161536	0.104413

Model Conclusions and Improvements

Overall, for our Ljung tests in Tables 4.4 and 4.5 we cannot reject the null hypothesis and therefore can assume our autocorrelations are independent from each other in most cases. However, our prices for MSFT and SPX seem to reject the null hypothesis over increased lag periods.

Attempts were made to improve the ARIMA models for MSFT and SPX to further reduce the residual patterns seen and we see some promise in adding a seasonal portion to our models. For these attempts we focused only on our AIC results since these models tend to be better than the models chosen by BIC and chose a simple seasonal order for p, d, and q.

Table 4.6
AIC Results for optimal Seasonal ARIMA model for MSFT and SPX

VectorNames	AIC	P D Q	Seasonal P D Q
Close.Adj.MSFT	27731.92	3 1 5	1 1 0
Close.SPX	49379.76	5 1 5	1 1 0

For exhibits 4.13 and 4.14 we do see some improvement in estimations as our residual correlations are closer to our confidence bands. However, there still seems to be a pattern remaining but it's not as obvious as before.

Exhibit 4.13
Optimal Seasonal ARIMA model chosen using AIC for MSFT: Contains ACF and PACF with various x and y limits

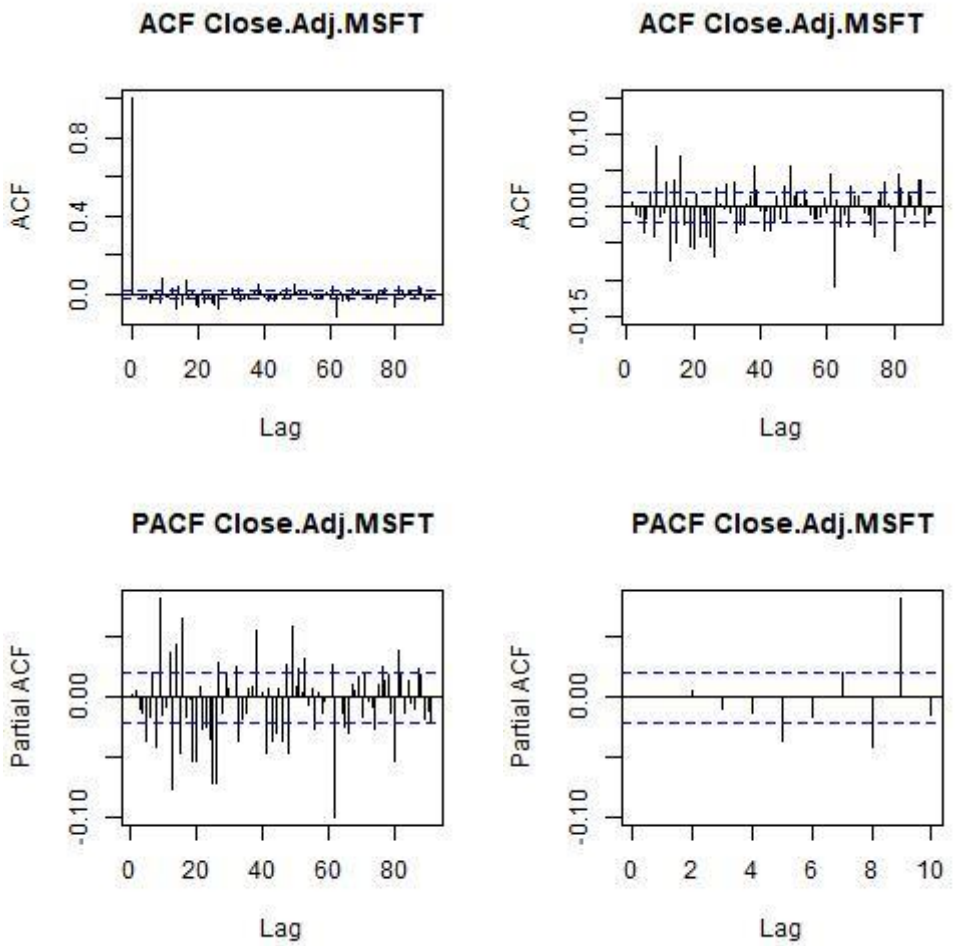


Exhibit 4.14

Optimal Seasonal ARIMA model chosen using AIC for SPX: Contains ACF and PACF with various x and y limits

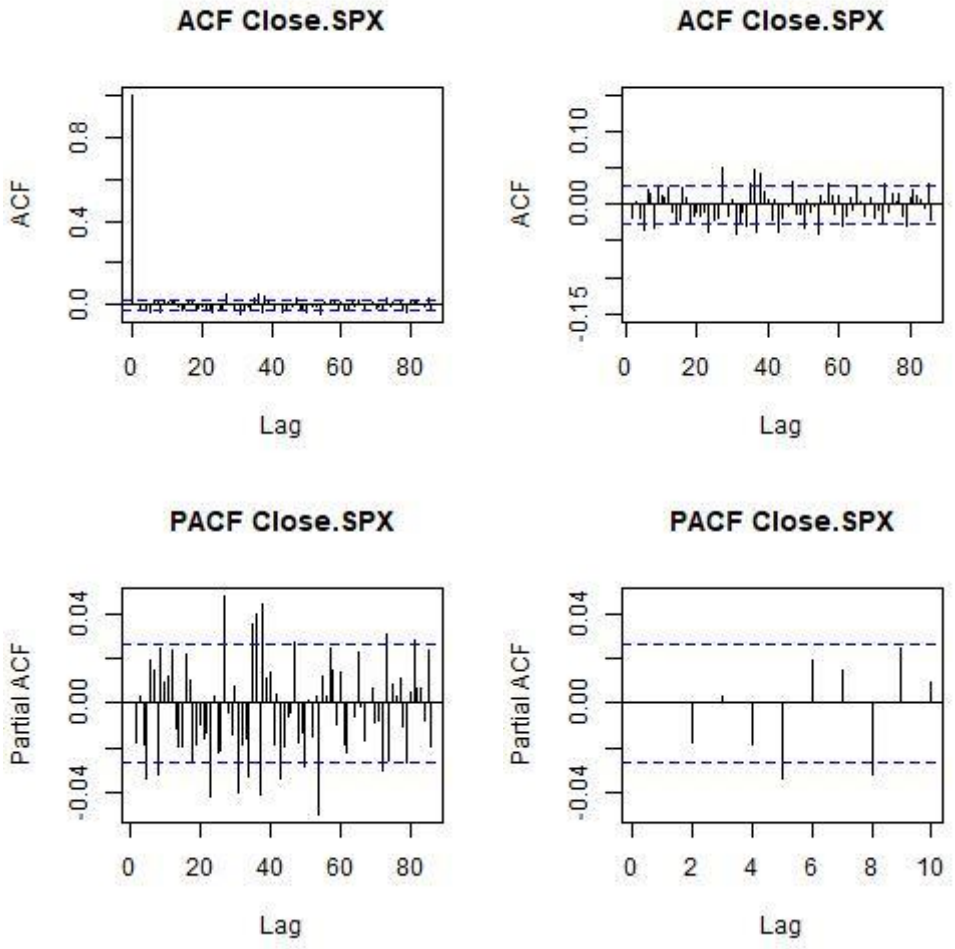


Table 4.7

Ljung Tests for Optimal Seasonal ARIMA model chosen with AIC

Lag	Close.Adj.MSFT	Close.SPX
1	0.822964	0.99511
2	0.864106	0.415405
3	0.783424	0.613766
4	0.657089	0.463005
5	0.014472	0.077217

For these new seasonal ARIMA models we do see significant improvement in our Ljung tests. However, by lag five we do see a drop and we could see further improvements if we were to exhaust other seasonal pdq models.

Code for Section Four

Figures 4.1, 4.2, and 4.3 show how we got our ACF, PACF, Ljung, ARIMA, and AIC/BIC results.

Figure 4.1

Function for running ACF, PACF, and Ljung Tests

```
Run_ACF_PACF_Ljung = function(Vector, Max_Lags){
  # This function receives a list of vectors and respective max lags to calculate
  # ACF, PACF, and Ljung pvalues

  # ACF and PACF
  ACF = mapply(function(x,y) acf(unlist(x), lag.max = y), x = Vector, y = Max_Lags, SIMPLIFY = FALSE)
  PACF = mapply(function(x,y) acf(unlist(x),type = "partial", lag.max = y), x = Vector, y = Max_Lags, SIMPLIFY = FALSE)

  # Ljung Test
  #   - Setup a blank matrix
  #   - repeat test for 1:5 lags
  #   - format Ljung object into a table containing only p=values

  Ljung.pvalues = matrix(,0,5) # column for each lag (I transpose later)

  for( i in 1:5) {
    # For loop to apply Ljung to different lags

    Ljung = lapply(Vector, function(x) Box.test(unlist(x), lag = i, type = "Ljung-Box", fitdf = 0))
    Ljung.pvalues = append(Ljung.pvalues, t(unlist(mapply("[", Ljung)[3,])))
    rm(Ljung)

  }

  # Format
  Ljung.pvalues = t(matrix(Ljung.pvalues,length(Max_Lags),5))
  colnames(Ljung.pvalues) = names(Vector)

  # List Our three variables
  A4.ACF.PACF.Ljung = list(ACF
                           , PACF
                           , Ljung.pvalues
  )

  # End of Function
  return(A4.ACF.PACF.Ljung)
}
```

Figure 4.2

Function for running ARIMA models with various settings

```
Run_ARIMA = function(Vector, diff) {
  # Setup empty list for all our models
  Models = vector("list", length(Vector)) # setup empty list for each vector
  Models = lapply(Models, function(x) vector("list", 5*5)) # setup empty lists for each model

  # Run ARIMA function in a loop for various p and q numbers
  for (i in 1:length(Vector)) {
    Index = 1
    print(paste("Starting...", names(Vector)[Index]))

    for (p in 1:5) {
      for(q in 1:5) {

        print(paste(i,p,q))

        model = arima(Vector[[i]], c(p,diff[i],q), method = "ML")
        order = paste(p,diff[i],q)

        Models[[i]][[Index]] = list(model,order)

        Index = Index + 1

      }
    }
  }

  # Name list at index 1
  names(Models) = names(Vector)

  # End of Function
  return(Models)
}
```

*When running an ARIMA with seasonality, we would use the exact same setup, but add an additional seasonal argument to the arima() function for our ‘for’ loop.

Figure 4.3

Function for passing through ARIMA models and returning AIC and BIC results

```
Run_AIC_BIC = function(Models) {
  # This function runs the AIC and BIC calculations for our ARIMA models in Section 4.

  # AIC
  AIC = lapply(Models, function(x) sapply(x, function(y) AIC(y[[1]])))
  AIC = do.call(rbind, AIC)
  AIC.MIN = apply(AIC, 1, function(x) min(x))
  AIC.MIN.Index = apply(AIC, 1, function(x) which(x == min(x)))

  # BIC
  BIC = lapply(Models, function(x) sapply(x, function(y) AIC(y[[1]], k = log(length(y[[1]]$residuals)))))
  BIC = do.call(rbind, BIC)
  BIC.MIN = apply(BIC, 1, function(x) min(x))
  BIC.MIN.Index = apply(BIC, 1, function(x) which(x == min(x)))

  # Create Lists for Both
  AIC_List = list(AIC
                  , AIC.MIN
                  , AIC.MIN.Index
  )
  names(AIC_List) = c("Result", "Min", "Index")


  BIC_list = list(BIC
                  , BIC.MIN
                  , BIC.MIN.Index
  )
  names(BIC_list) = c("Result", "Min", "Index")

  # Combine Lists
  AIC.BIC = list(AIC_List, BIC_list)
  names(AIC.BIC) = c("AIC", "BIC")



  # End of Function
  return(AIC.BIC)
}
```

Section Five – Optimal GARCH/ARCH Models

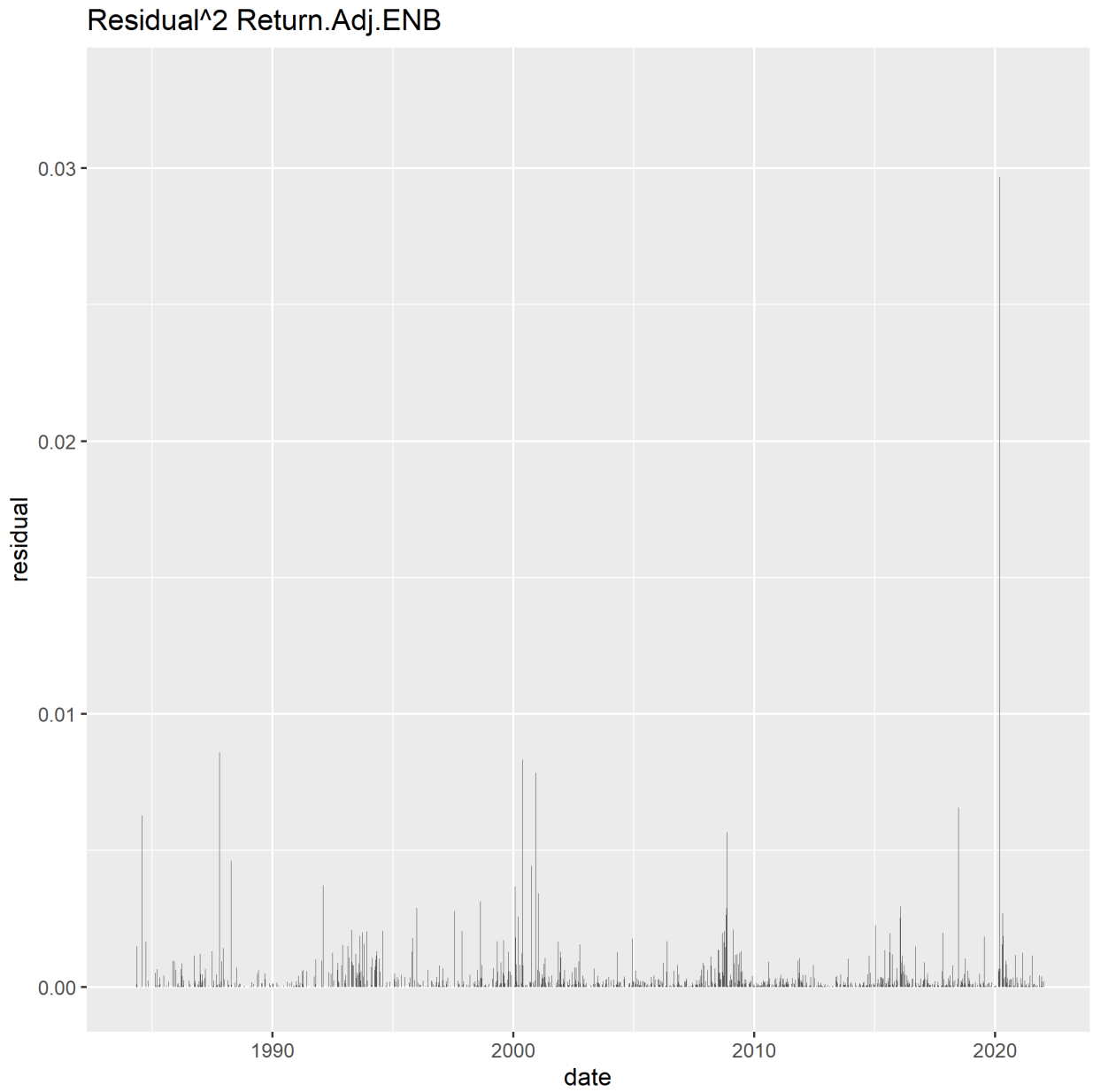
Optimal ARMA Squared Residuals Test

Using our results from table 4.3, we pulled our optimal AIC models for our returns and pulled out their residuals and squared them. We choose just the AIC models because they performed better than our BIC models. We can then see each time plot of the squared residuals in plots 5.1, 5.2, and 5.3. In general, we can see some obvious points of high variability. This is probably related to peak gains or loses with the risky asset potentially during events such as covid or recession periods.

In exhibits 5.1 and 5.2 with can see our ACF and PACF tests on our squared residuals. In general, for each return under ACF and PACF, we see serious correlation between lags at the beginning, but it exponentially decreases as the lags increase.

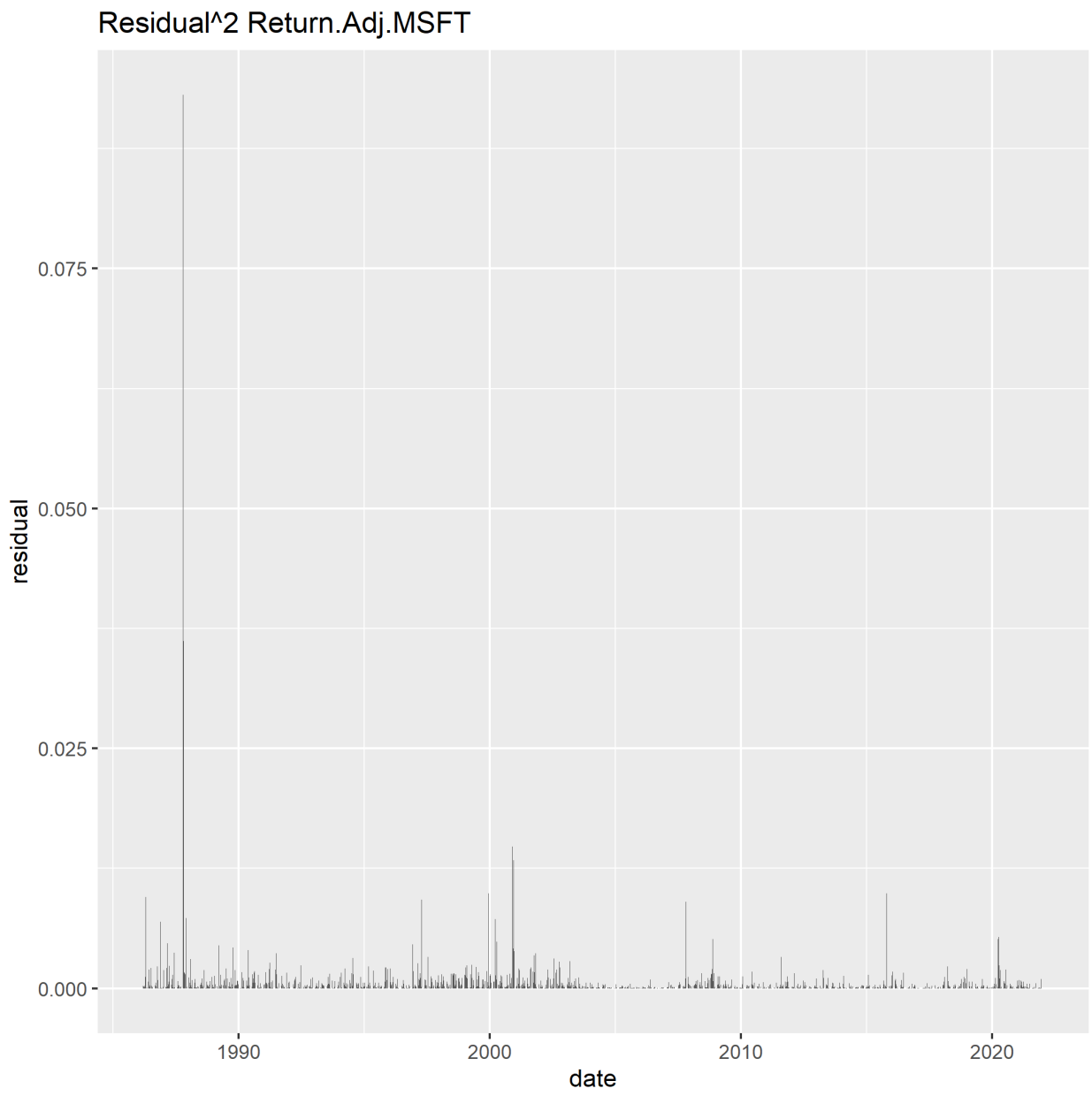
Plot 5.1

Residuals^2 for Return.Adj.ENB from AIC Optimal Model



Plot 5.2

Residuals^2 for Return.Adj.MSFT from AIC Optimal Model



Plot 5.3

Residuals^2 for Return.SPX from AIC Optimal Model

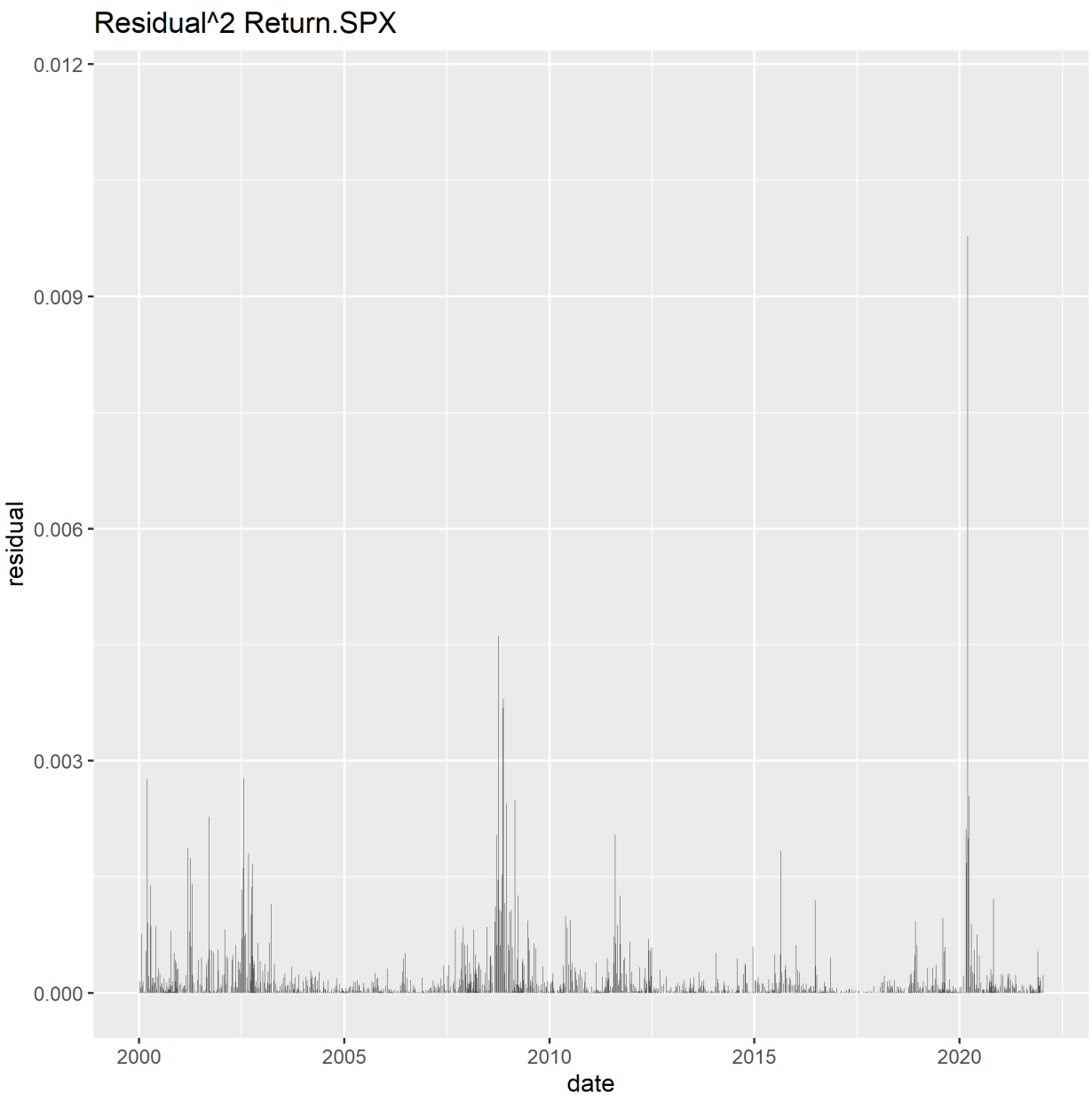


Exhibit 5.1

Residuals^2 Returns (ENB, MSFT, and SPX) with ACF plotted for Optimal AIC model

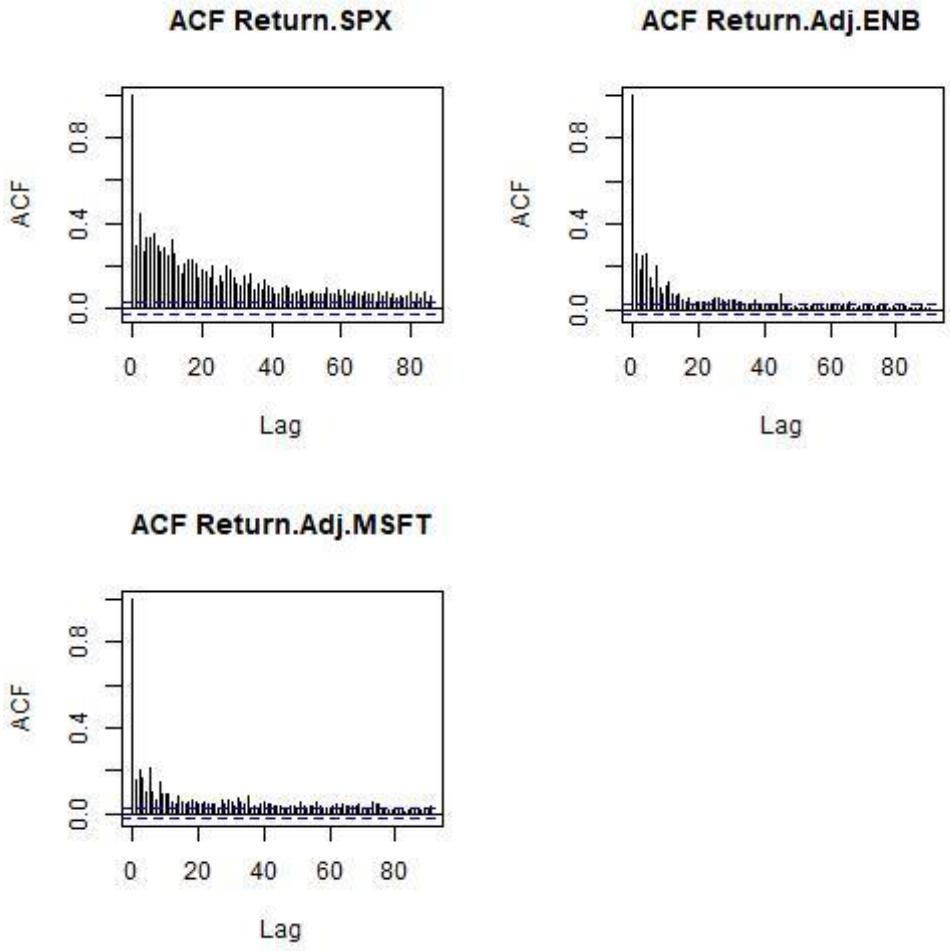


Exhibit 5.2

Residuals^2 Returns (ENB, MSFT, and SPX) with PACF plotted for Optimal AIC model

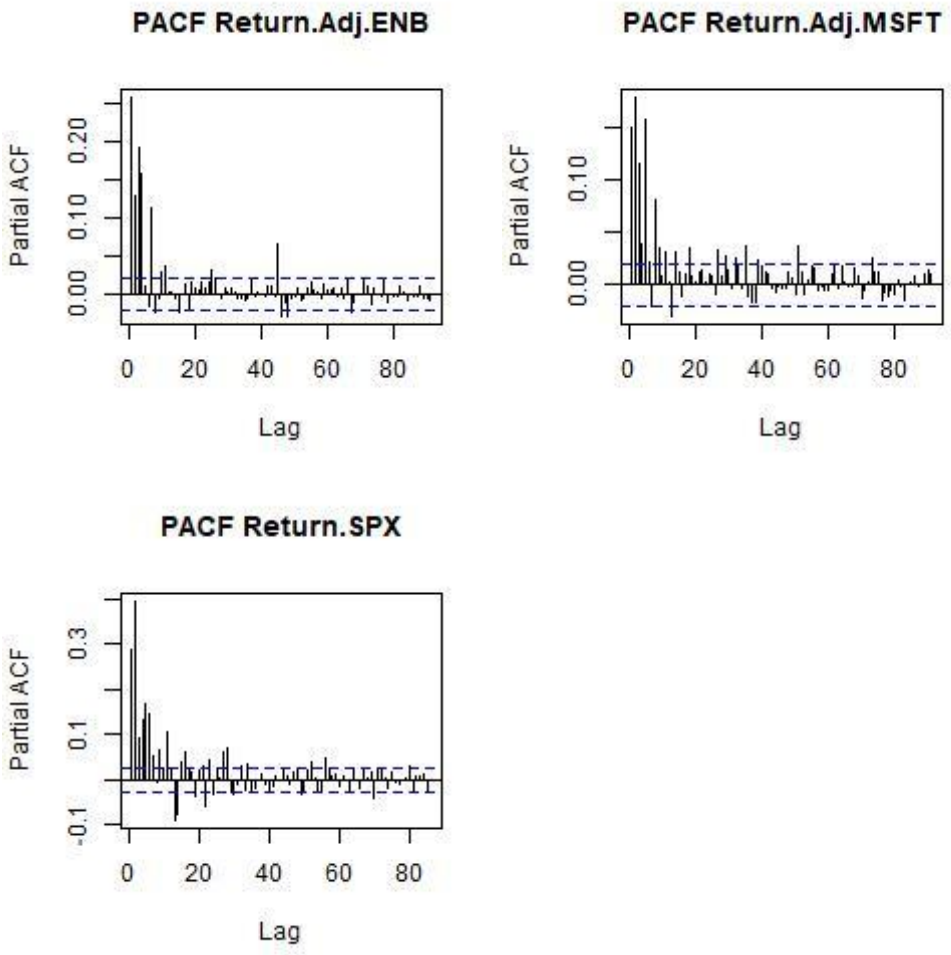


Table 5.1

Ljung Tests for Residuals^2 Returns (ENB, MSFT, and SPX) with PACF plotted for Optimal AIC model

Lag	Return.Adj.ENB	Return.Adj.MSFT	Return.SPX
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0

For our Ljung tests we can reject the null and assume autocorrelation between our lags for our residual squared returns

GARCH Diagnostic

Using just the optimal ARMA models we ran various p and q values for our GARCH and returned the results in table 5.2. Overall, AIC and BIC tests show similar results and identical optimal models.

Table 5.2
AIC and BIC Tests returning optimal GARCH models

VectorNames	AIC	p	q	BIC	p	q
Return.Adj.ENB	-12.4896	4	5	-12.4798	4	5
Return.Adj.MSFT	-11.2205	1	4	-11.2134	1	4
Return.SPX	-14.0928	1	2	-14.0844	1	2

For our GARCH models, in exhibit 5.3 we can see our ACF results which look like an improvement over the previous ACF tests in exhibit 5.1. Exhibit 5.4 shows our PACF which also shows an improvement over the previous PACF tests in exhibit 5.4. However, once we see our Ljung test we fail to reject the null hypothesis for our ENB and MSFT returns indicating a lack of evidence for autocorrelation, and unfortunately reject the null for our SPX return. Our GARCH models can be confidently used with ENB and MSFT, but we'll have to think of another feasible model for SPX returns.

Exhibit 5.3

Returns (ENB, MSFT, and SPX) using GARCH with ACF plotted for Optimal AIC model

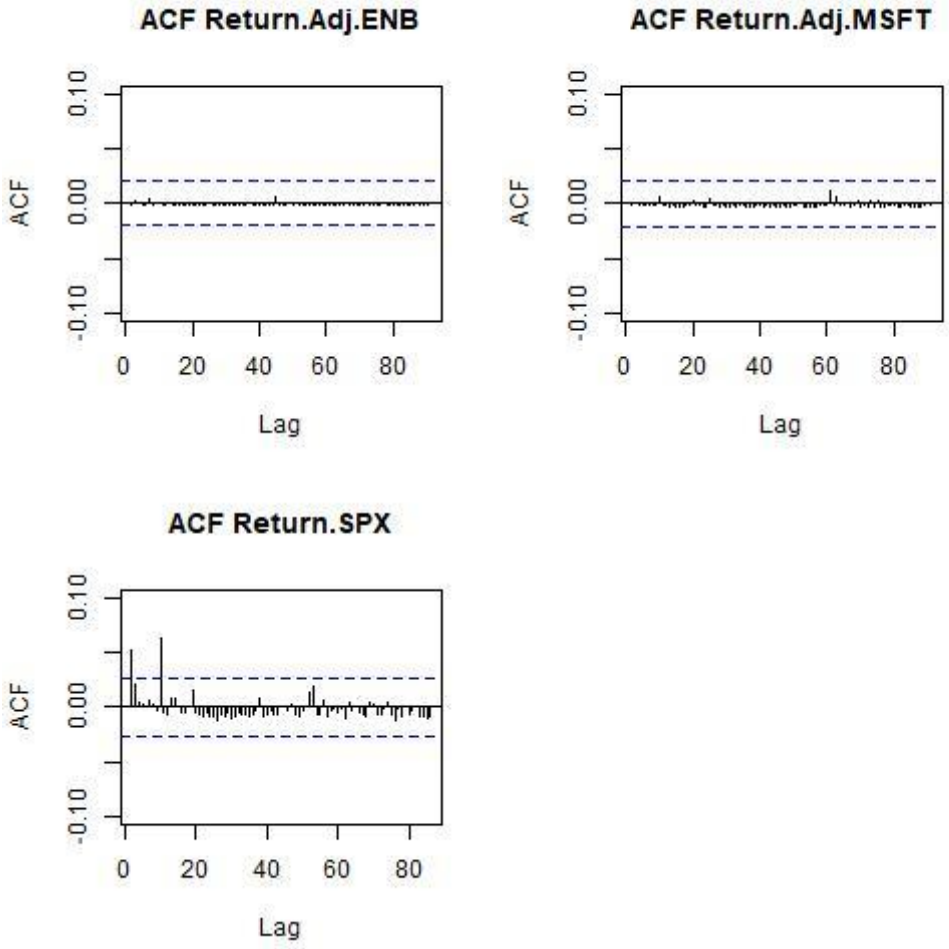


Exhibit 5.4

Residuals^2 Returns (ENB, MSFT, and SPX) using GARCH with PACF plotted for Optimal AIC model

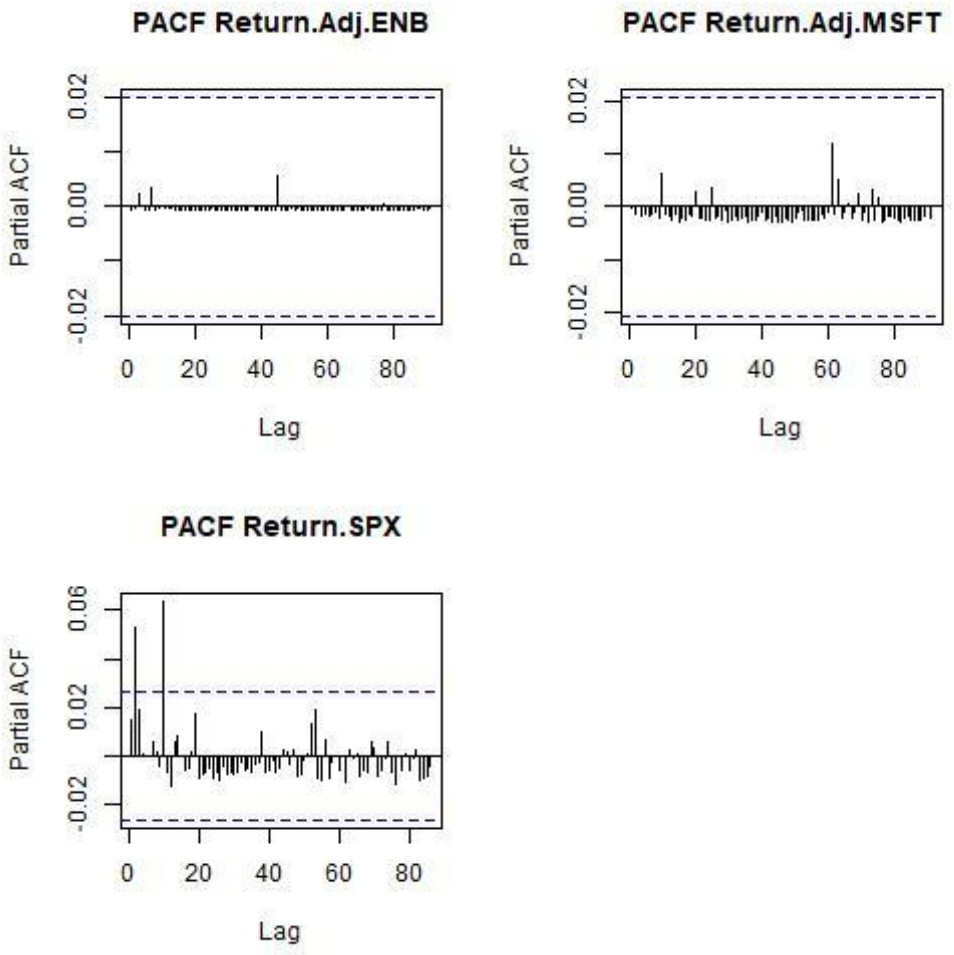


Table 5.1

Ljung Tests for Residuals^2 Returns (ENB, MSFT, and SPX) using GARCH for Optimal AIC models

Lag	Return.Adj.ENB	Return.Adj.MSFT	Return.SPX
1	0.959853	0.978978	0.284596
2	0.997858	0.991371	0.000259
3	0.996213	0.999395	0.000298
4	0.999561	0.999772	0.000816
5	0.99995	0.999942	0.001967

GARCH Estimation

Some error may be present in table 5.2 as indicated by various standard errors calculation returning null making it difficult to understand our results. However, most standard errors seen are relatively low, which generally indicates that volatility may not be highly persistent or clustering. This is seen across our three returns.

Table 5.2
Coefficients and Standard Errors for GARCH model Estimation for each return.

Names	COEF/SE	mu	ar1	ma1	omega	alpha1
Return.Adj.ENB	COEF	0.00018	0.97895	(0.94956)	0.00000	0.14791
Return.Adj.ENB	SE	0.00001	0.00335	0.00487	0.00000	0.00931
Return.Adj.MSFT	COEF	0.00025	0.93077	(0.81364)	0.00000	0.05629
Return.Adj.MSFT	SE	#NUM!	0.00013	#NUM!	0.00000	0.00001
Return.SPX	COEF	0.00015	0.96623	(0.81757)	0.00000	0.05000
Return.SPX	SE	0.00003	#NUM!	0.02146	0.00000	0.00315

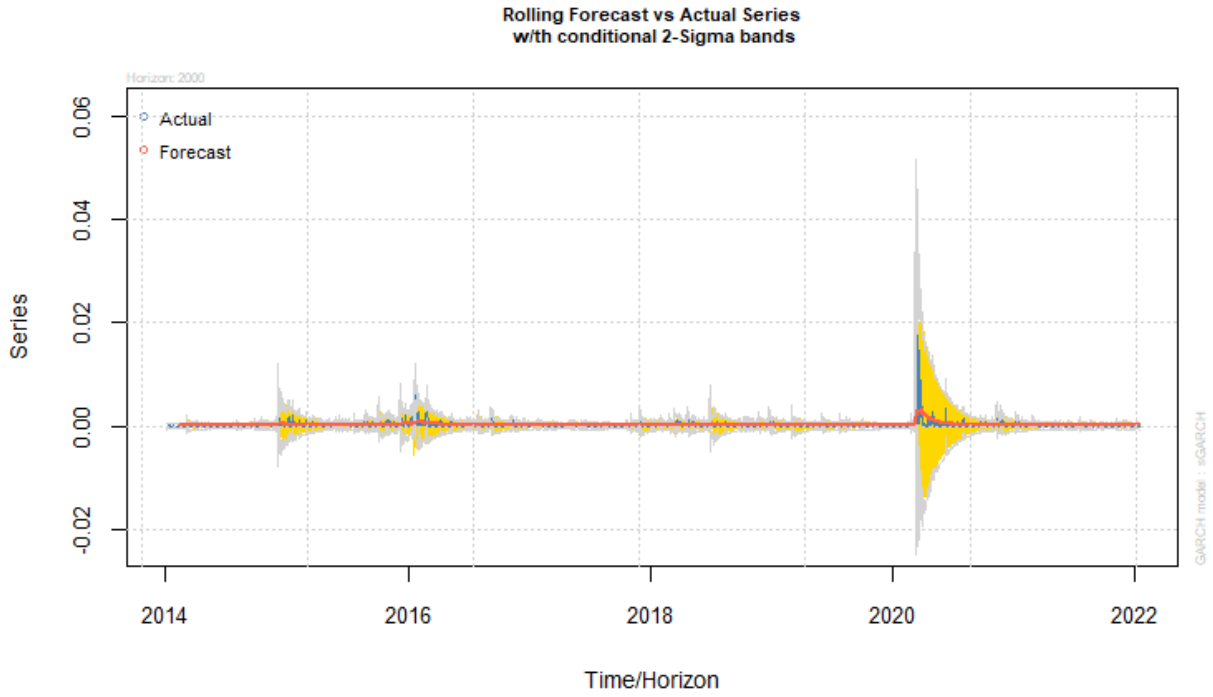
alpha2	alpha3	alpha4	beta1	beta2	beta3	beta4	beta5
0.03153	0.02565	-	0.01655	-	-	0.34559	0.44271
0.00427	0.00235	#NUM!	#NUM!	#NUM!	#NUM!	0.01693	#NUM!
0.22382	0.22703	0.23067	0.23404	0.00025	0.93077	(0.81364)	0.00000
0.00002	#NUM!	0.00003	0.00004	#NUM!	0.00013	#NUM!	0.00000
0.45000	0.45000	0.00015	0.96623	(0.81757)	0.00000	0.05000	0.45000
0.05530	0.04878	0.00003	#NUM!	0.02146	0.00000	0.00315	0.05530

*COEF/SE column stands for coefficient and standard error.

Looking at plots 5.4 to 5.6, we do not see much change in our volatility over time, for most of the series it seems constant. However, across all three we do see points where volatility jumps especially during 2020 from the initial waves of covid. There are also various events of higher volatility around 2016 and 2018.

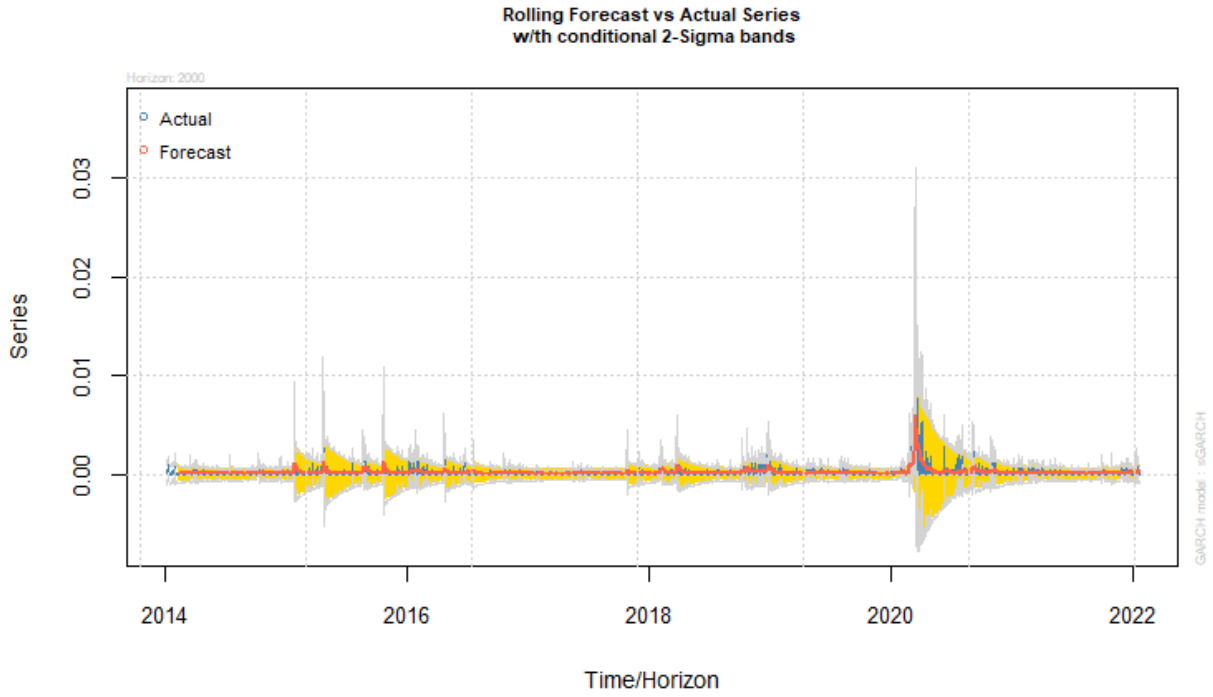
Plot 5.4

Forecast Volatility Estimation for ENB GARCH forecast



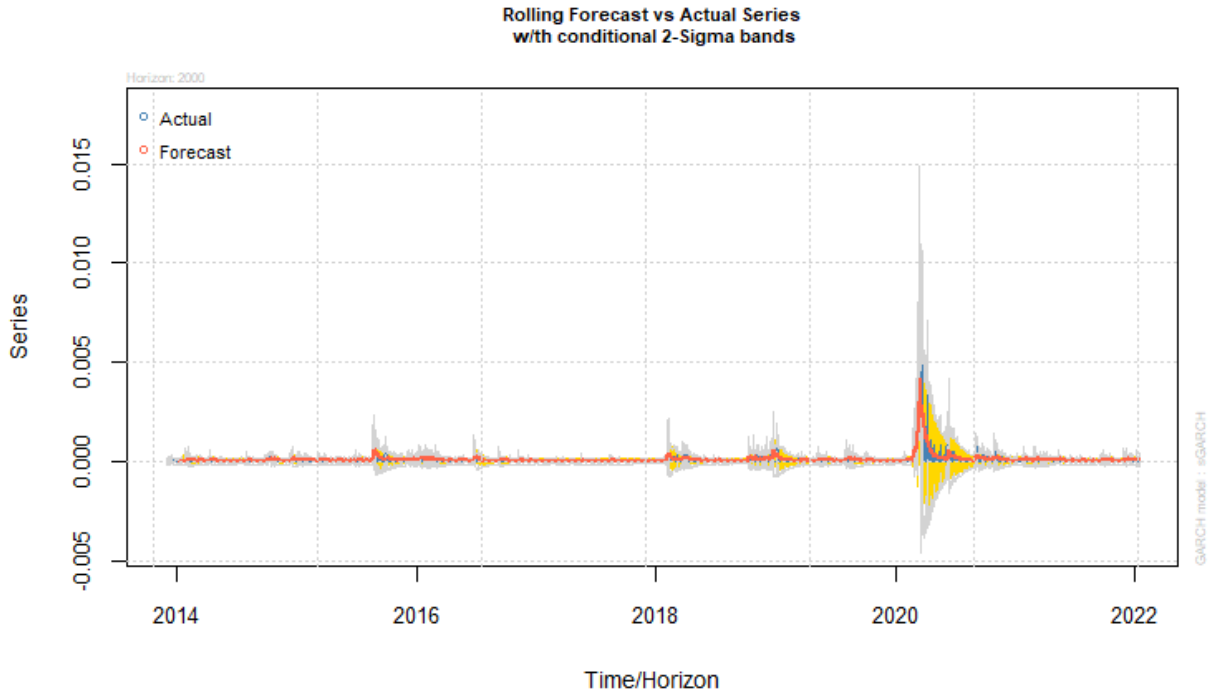
Plot 5.5

Forecast Volatility Estimation for MSFT GARCH forecast



Plot 5.6

Forecast Volatility Estimation for SPX GARCH forecast



Portfolio Allocation and Analysis

Table 5.3

Portfolio Allocation Adjusted by Aversion and Max/Min Risky Asset Volatility Forecasts

Risky Asset	K*	K = 1	k = 5	k = 7
Return.Adj.ENB	Var Riskless	2.124335935	0.424867187	0.303476562
Return.Adj.ENB	Max volatility	0.035480014	0.007096003	0.005068573
Return.Adj.ENB	Min volatility	1.973506264	0.394701253	0.281929466
Return.Adj.MSFT	Var Riskless	2.295192536	0.459038507	0.327884648
Return.Adj.MSFT	Max volatility	0.047770082	0.009554016	0.006824297
Return.Adj.MSFT	Min Volatility	4.069744098	0.81394882	0.581392014
Return.SPX	Var Riskless	1.111672836	0.222334567	0.158810405
Return.SPX	Max volatility	0.060736944	0.012147389	0.008676706
Return.SPX	Min volatility	3.12163256	0.624326512	0.445947509

In table 5.3, when risk aversion increases (from k = 1, to k = 7), we decrease the weight on the risky assets across all scenarios (Variance Riskless, Max Volatility, and Min Volatility). When we use minimum volatility, we allow higher weights on the risky asset while using maximum volatility decreases the allowed weights on risky assets. This is seen across each of our risky assets using each instance of volatility. This makes sense considering the expectation of returns given certain volatilities. If volatility is high, we ideally adjust weights lower to cap their portfolio influence, while lower volatility allows us to use more of these risky assets without fear of losing returns.

Code for Section Five

Figures 5.1 and 5.2 cover creating, fitting, and forecasting with GARCH models. Figure 5.3 covers how standard errors were returned, and figure 5.4 covers our portfolio allocation using GARCH volatility forecasts.

Figure 5.1
Example 'for' loop for running GARCH models

```
for (i in 1:length(A5.Residuals.Squared.DF)) {
  # Loop to run and format various GARCH models

  # Variables
  Index = 1
  print(paste("Starting...", names(A5.Residuals.Squared.DF)[Index]))

  # For loop to run various p and q
  for (p in 1:5) {

    for(q in 1:5) {

      print(paste(i,p,q))

      # Model fitted to a package specific class
      model = ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(p,q))
                         , mean.model = list(aeamaOrder = A5.Order[[i]], include.mean = TRUE))


      # Fit Model to Data
      A5.GARCH.Models[[i]][[Index]] = list(ugarchfit(model, unlist(A5.XTS[[i]]), solver = "hybrid")) # Fi
      A5.GARCH.Models[[i]][[Index]][[2]] = c(p,q)

      Index = Index + 1

    }
  }

}
```

*I pass in various P and Q values while iterating over our optimal ARMA orders from the prior section.

Figure 5.2
Example with Anonymous function to calculate GARCH forecasts

```
A5.GARCH.Forecast = mapply(function(x,y,z) {
  # Function to run GARCH forecasts

  # Setup Variables
  optimal = x[[y]][[1]] # return optimal model based on AIC/BIC
  spec = getspec(optimal) # convert model to specific package class
  setfixed(spec) = as.list(coef(optimal)) # format for package to use
  forecast = ugarchforecast(spec, data = z, n.ahead = 1, n.roll = 2000, out.sample = 2000) # run forecast w optimal GARCH
  condVarForecast = sigma(optimal)

  return(list(spec,forecast,condVarForecast))

}
  , x = A5.GARCH.Models
  , y = AIC.BIC.3[[1]][[3]], SIMPLIFY = FALSE
  , z = A5.XTS[1:6]
)
```

* I pass in all my GARCH models, subset for the optimal ones using AIC/BIC and then match them when their specific squared residual data to then forecast the results.

Figure 5.3
Returning coefficients and calculating standard errors from optimal GARCH models

```
# Coefficients and Standard Errors

A5.GARCH.COEF = mapply(function(x,y) coef(x[[y]][[1]]), x = A5.GARCH.Models, y = AIC.BIC.3[[1]][[3]], SIMPLIFY = FALSE)
A5.GARCH.VCOV = mapply(function(x,y) vcov(x[[y]][[1]]), x = A5.GARCH.Models, y = AIC.BIC.3[[1]][[3]], SIMPLIFY = FALSE)
A5.GARCH.SE   = lapply(A5.GARCH.VCOV, function(x) sqrt(diag(x)))

A5.GARCH.COEF = data.frame(do.call(rbind, A5.GARCH.COEF))
A5.GARCH.SE = data.frame(do.call(rbind, A5.GARCH.SE))

GARCH.Data = list(A5.GARCH.COEF, A5.GARCH.SE)
names(GARCH.Data) = c("COEF", "SE")
```

Figure 5.4
Calculating portfolio allocations with new variance forecasts.

```
# Portfolio Allocation

mrisky = CombinedData[["Return.Adj.ENB"]] %>% na.omit() %>% mean()
mriskless = CombinedData[["RF"]] %>% na.omit() %>% mean()
k = 1
vrisky = CombinedData[["Return.Adj.ENB"]] %>% na.omit() %>% var()

optimalweightriskless = (mrisky - mriskless) / (k*vrisky)


A5.Return.Vectors = lapply(VectorNames[4:6], function(x) as.ts(CombinedData[!is.na(CombinedData[x]), x]))

A5.Porfolio.Allocation = mapply(function(x,y) {
  # Function to use GARCH forecasted volatility to create OMV portfolios

  # Risky and Riskless Assets
  mean_risky = x %>% mean()
  mean_riskless = CombinedData[["RF"]] %>% na.omit() %>% mean()

  # Risk Aversions
  k = c(1,5,7)

  # Variances
  variance_risky = x %>% var()
  max_volitility = y[[3]] %>% max()
  min_volitility = y[[3]] %>% min()

  # Optimal Weights Calculations
  Optimal_Weights_Var = (mean_risky - mean_riskless) / (k*variance_risky)
  Optimal_Weights_Max = (mean_risky - mean_riskless) / (k*max_volitility)
  Optimal_Weights_Min = (mean_risky - mean_riskless) / (k*min_volitility)
```

Appendix

Figure 2.1

All Equal Portfolio Return Weights

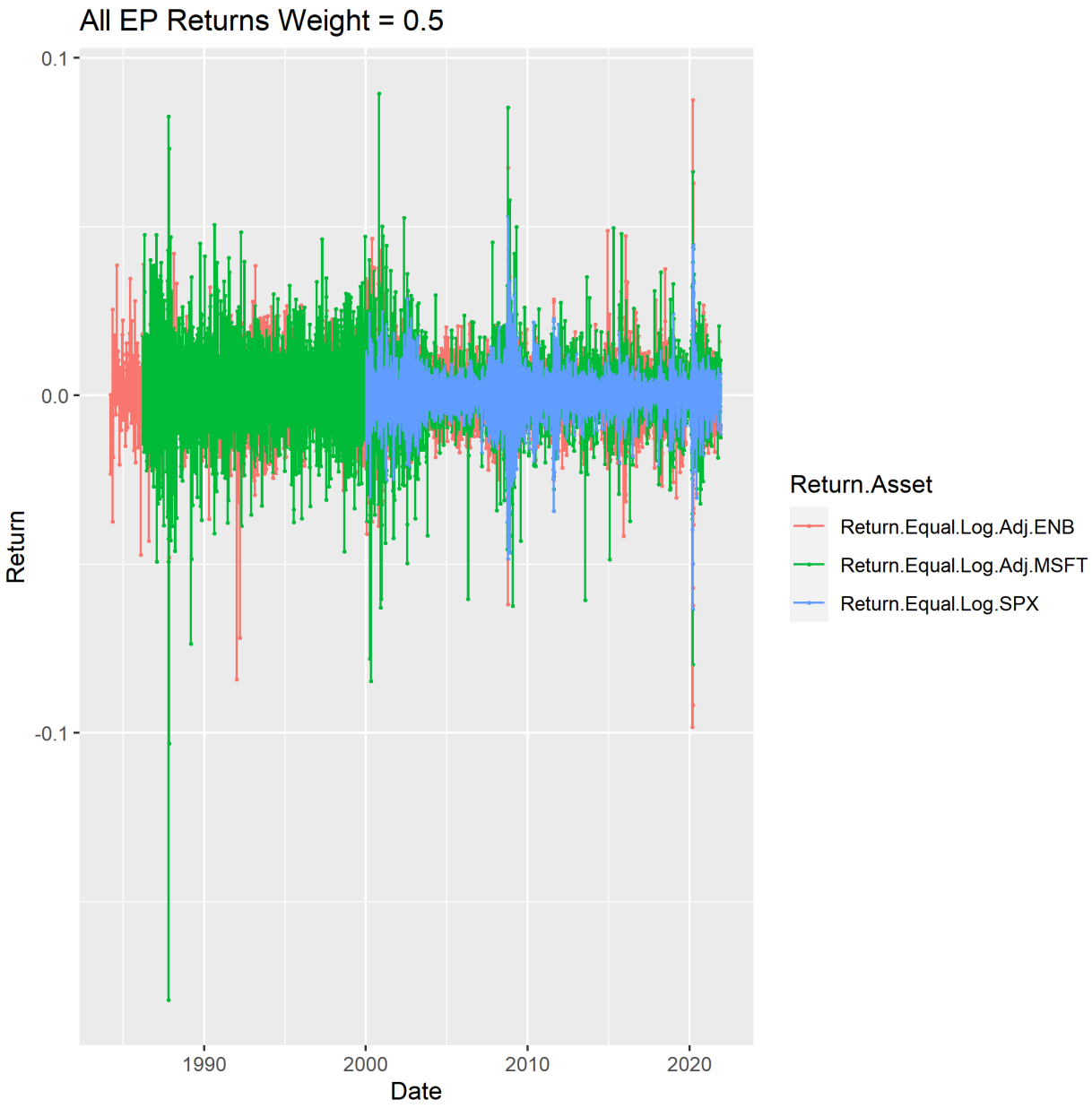


Figure 2.2

All Equal Portfolio Return Weights

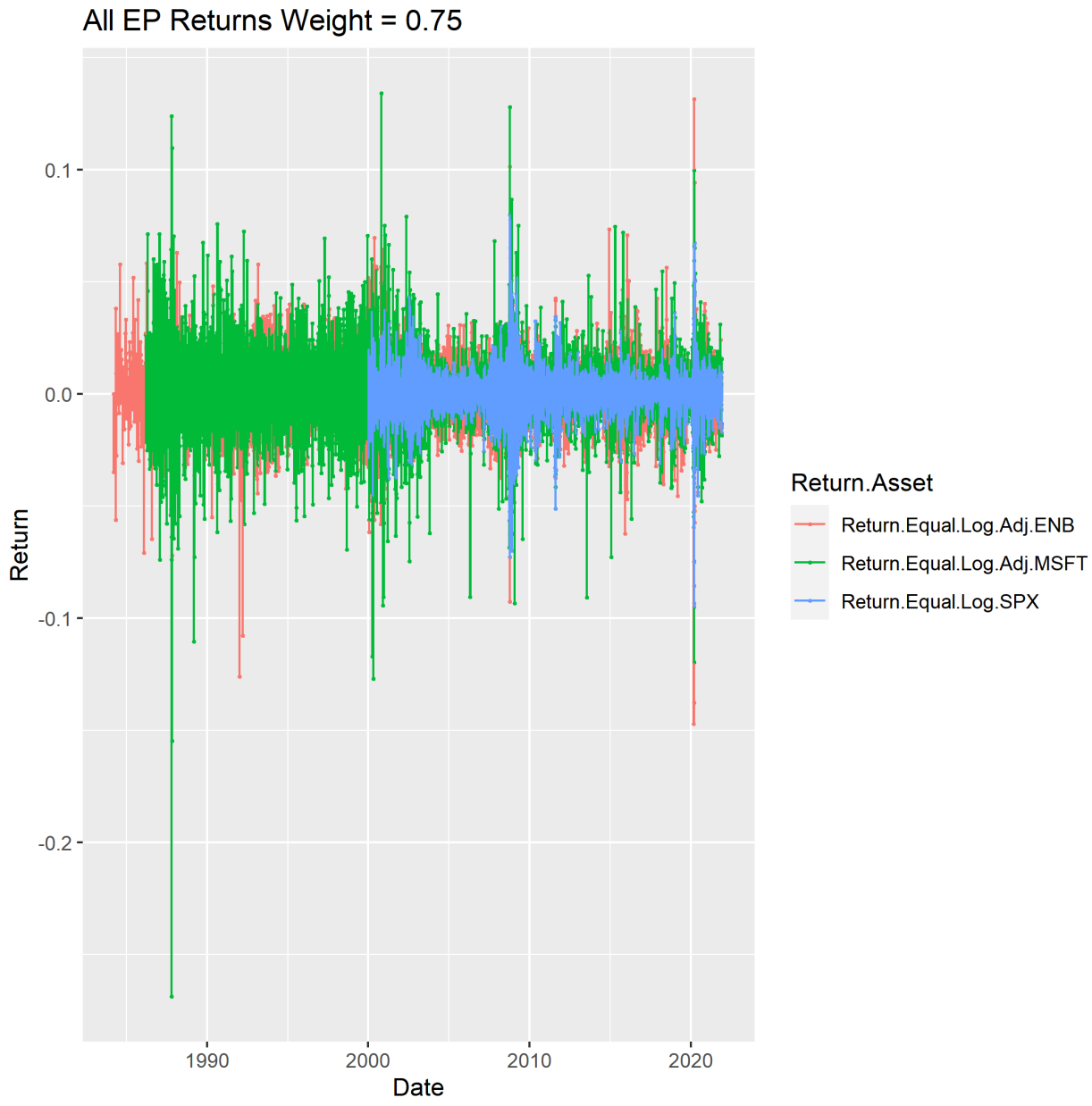


Figure 2.3

All Equal Portfolio Return Weights

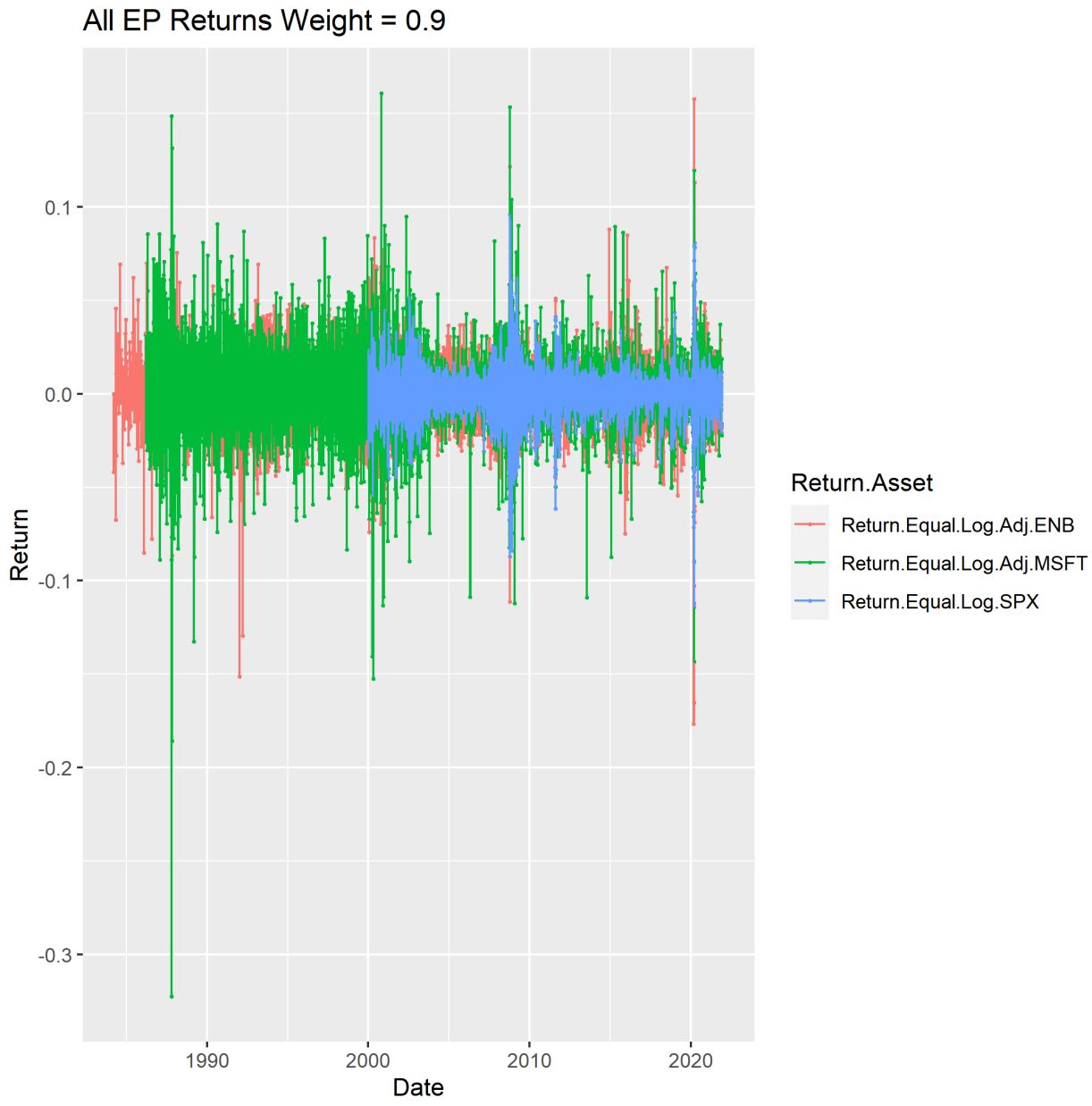


Figure 2.4

All Optimal Mean Variance Portfolio Returns

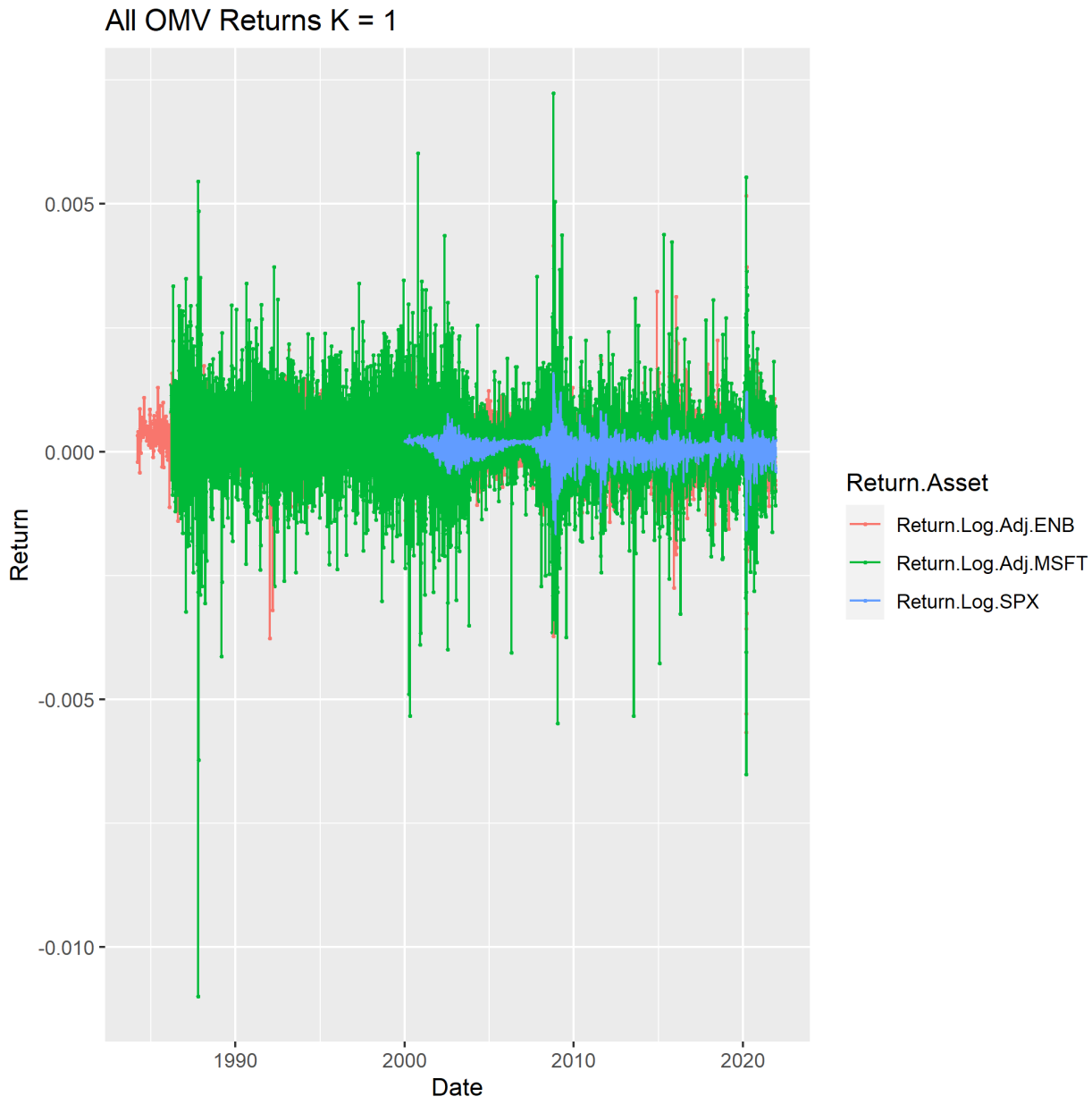


Figure 2.5

All Optimal Mean Variance Portfolio Returns

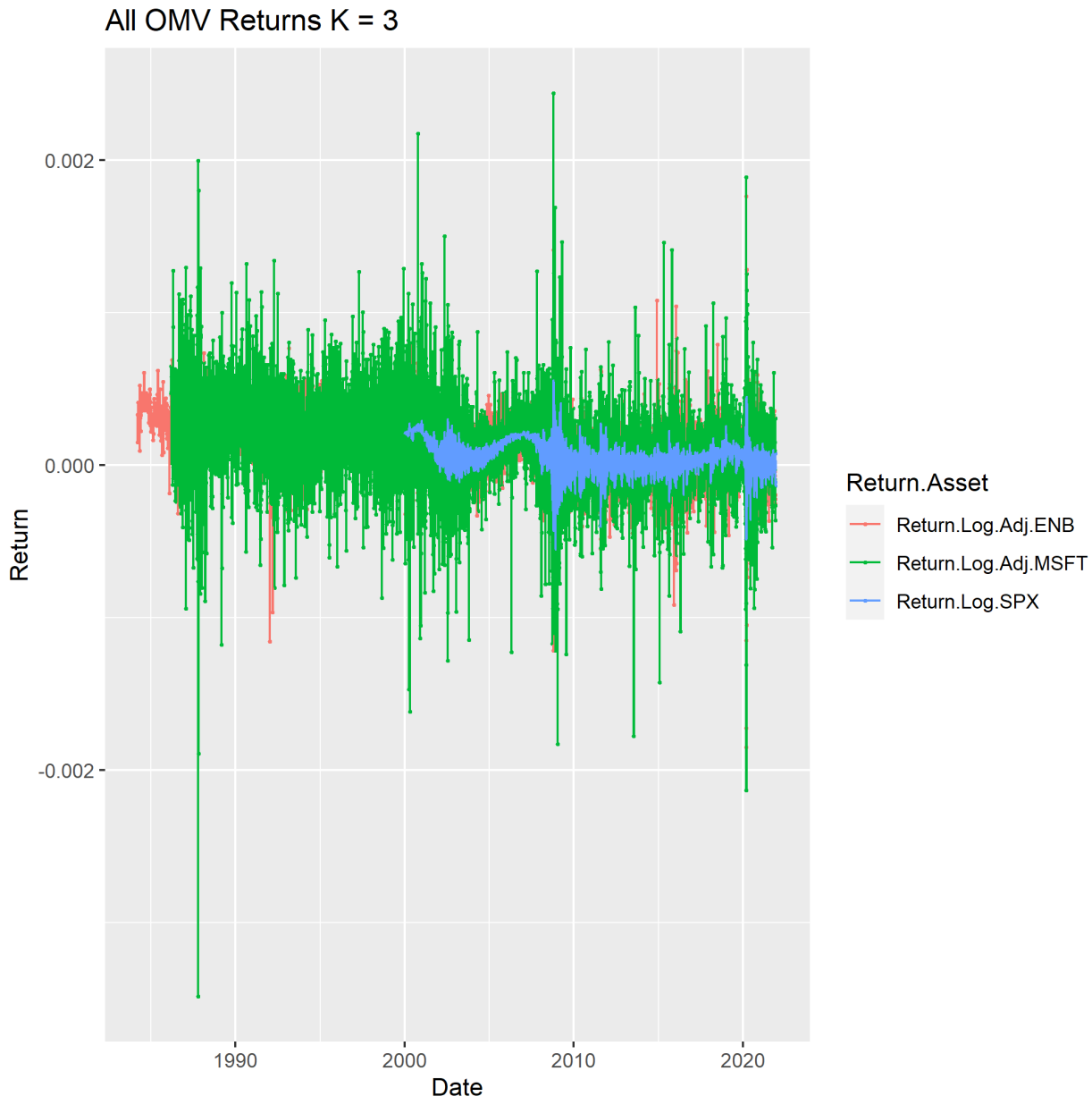


Figure 2.6

All Optimal Mean Variance Portfolio Returns

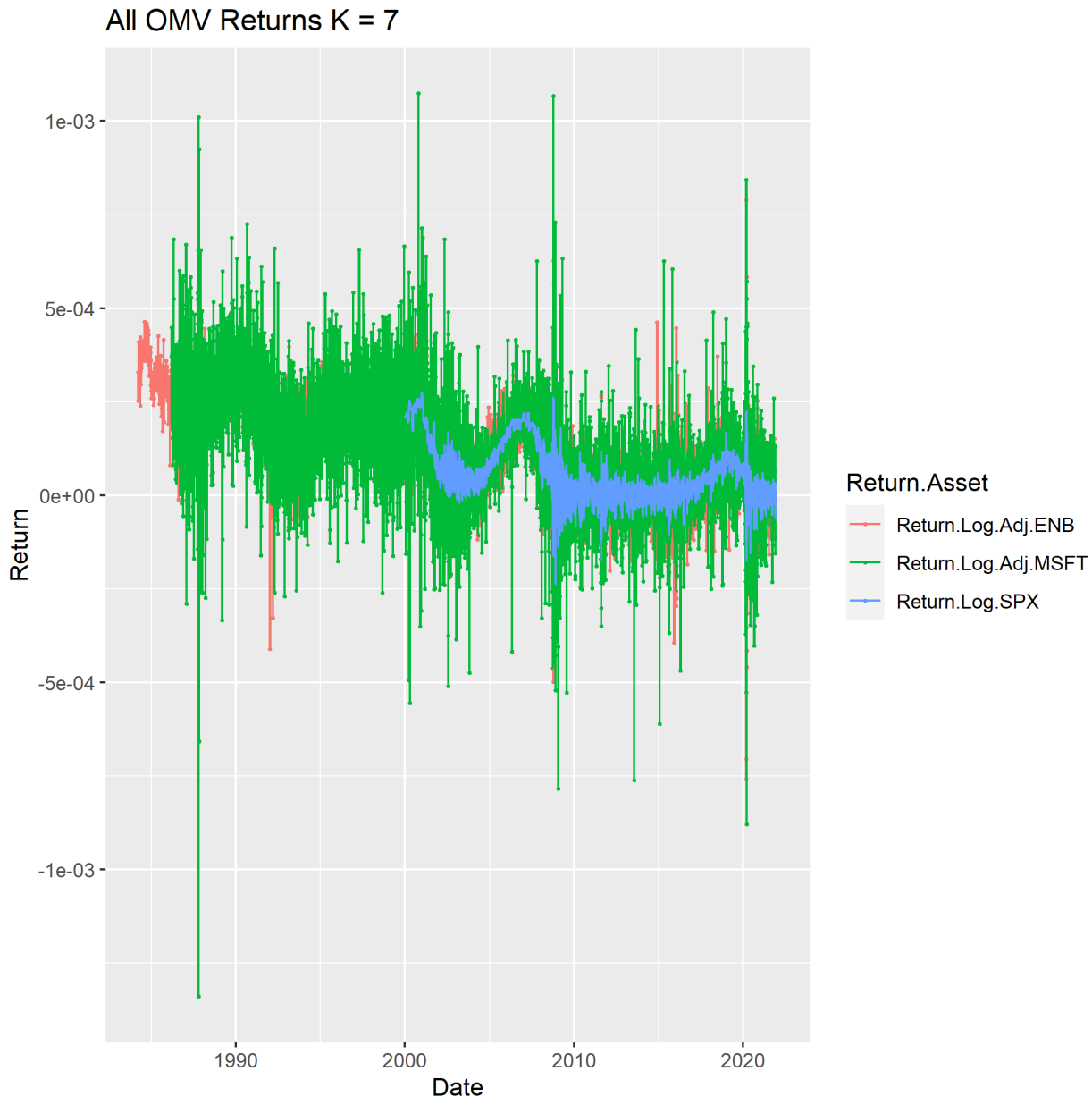


Figure 2.7

All Optimal Mean Variance Portfolio Weights

Figure 2.8

All Optimal Mean Variance Portfolio Weights

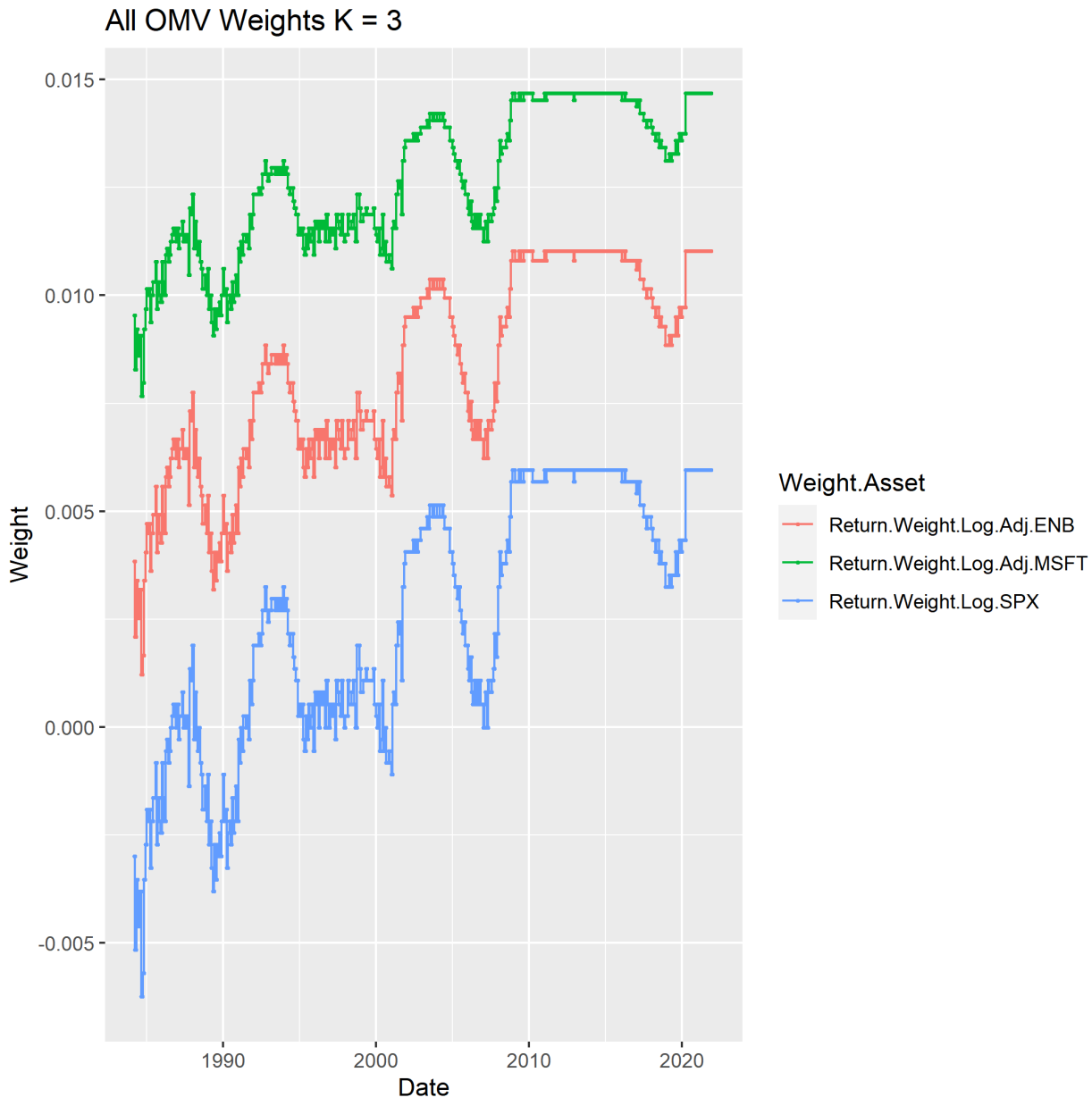


Figure 2.9

All Optimal Mean Variance Portfolio Weights

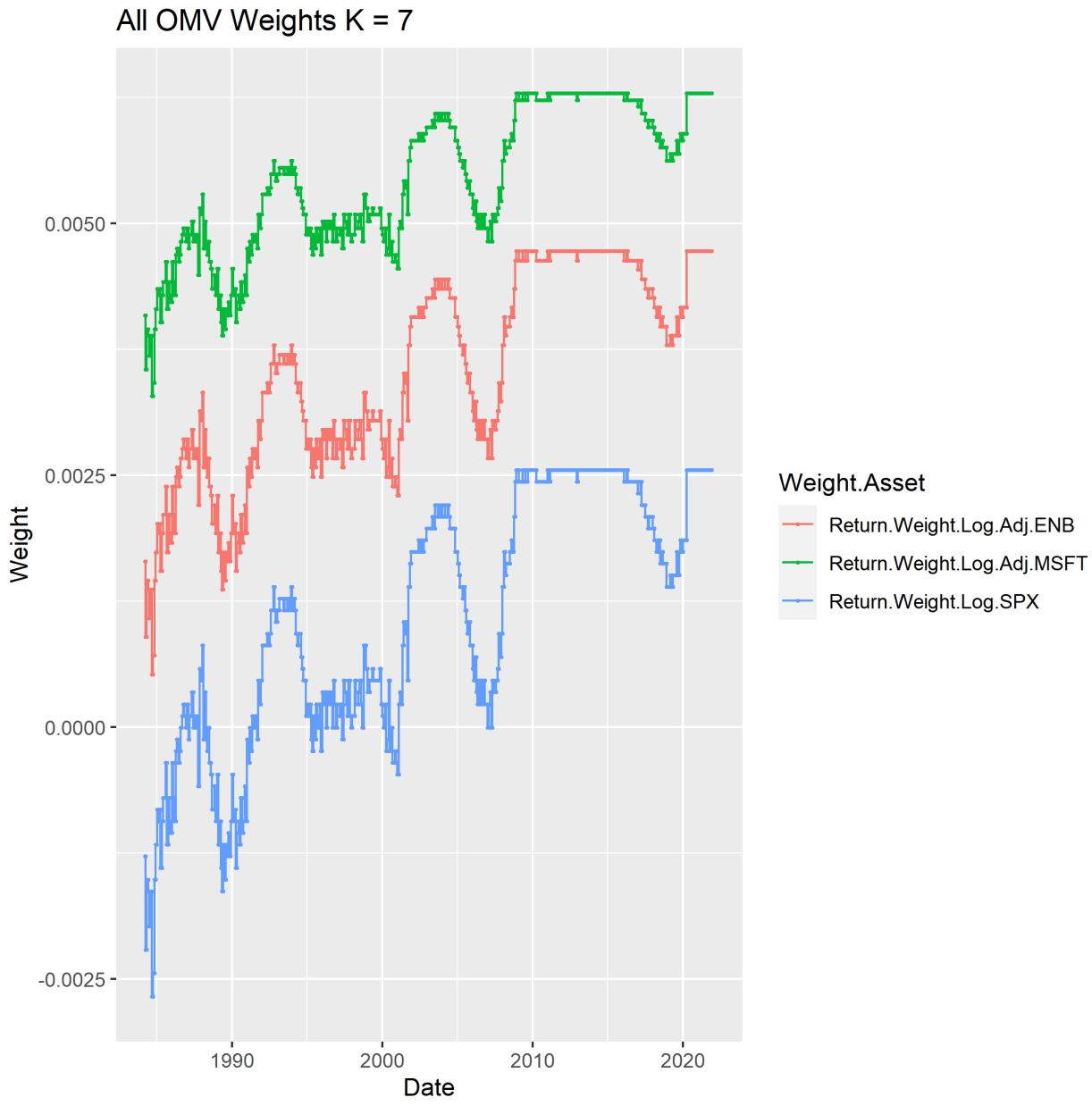


Table 3.4

Critical Values for Dickey Fuller Test – Model One

N	0.01	0.025	0.05	0.10
25	-2.661	-2.273	-1.955	-1.609
50	-2.612	-2.246	-1.947	-1.612
100	-2.588	-2.234	-1.944	-1.614
250	-2.575	-2.227	-1.942	-1.616
500	-2.570	-2.224	-1.942	-1.616
>500	-5.567	-2.223	-1.941	-1.616

Table 3.5

Critical Values for Dickey Fuller Test – Model Two

N	0.01	0.025	0.05	0.10
25	-3.724	-3.318	-2.986	-2.633
50	-3.568	-3.213	-2.921	-2.599
100	-3.498	-3.164	-2.891	-2.582
250	-3.457	-3.1336	-2.873	-2.573
500	-3.443	-3.127	-2.867	-2.570
>500	-3.434	-3.120	-2.863	-2.568

Table 3.6

Critical Values for Dickey Fuller Test – Model Three

N	0.01	0.025	0.05	0.10
25	-4.375	-3.943	-3.589	-3.238
50	-4.152	-3.791	-3.495	-3.181
100	-4.052	-3.722	-3.452	-3.153
250	-3.995	-3.683	-3.427	-3.137
500	-3.977	-3.670	-3.419	-3.132
>500	-3.963	-3.660	-3.413	-3.128

Figure 6.0

Full Script for our Final Project, divided into 5 sections for calculations and plotting, with additional sections for exporting, functions, packages, directories, and data manipulation.

```
# Final Project Script
#==============================================================================
# Culmination of our efforts.

#==============================================================================
# Packages
#==============================================================================
# You will need the below packages to run the entire script, openxlsx is optional,
# however, it's recommended if you wish to export any of the tables with the existing script.

# Packages we need
Packages = c("tidyverse", "lubridate", "openxlsx", "urca", "rugarch", "xts")

# This will load and install the packages if required
PackagesCheck = lapply(
  Packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
       install.packages(x, dependicies = TRUE)
       library(x, character.only = TRUE)
    }
  }
)

# Clean up
rm(Packages, PackagesCheck)


#==============================================================================
# Functions
#==============================================================================

# Excel Functions

Excel_Import = function(FileDir = FileDir) {
  # This function takes our FileDir list and uses it to import all our csv's

  # Variables
  List = vector(mode = "list", length = length(FileDir)) # Creating a blank list to store our imports
```

```
  skip = c(0,0,4,0) # Need to skip 4 rows for french's data to avoid an error

  # For Loop - Iterates over FileDir and pulls each file from my directory.
  for (i in 1:length(FileDir)) {
    List[[i]] = as.data.frame(read.csv(file = FileDir[[i]], skip = skip[i]))
  }

  # End of Function
  return(List)

}

Excel_Export = function(FileExport = FileExport, Directory = Dir, FileName = "Assignment One Data.xlsx")
{
  # This function Writes data.frames (most formats should work) to excel

  # Variables
  wb = createWorkbook(,title = FileName) # This is setting up a workbook in our environment to write to.

  # For Loop - Iterates over FileExport, adds a worksheet to workbook, writes our data.
  for (i in 1:length(FileExport)) {
    addWorksheet(wb, sheetName = names(FileExport)[i])
    writeDataTable(wb,sheet = names(FileExport)[[i]], FileExport[[i]], rowNames = TRUE)
  }

  # End of Function - Saves our workbook to our directory.
  saveWorkbook(wb, file = paste0(Directory,FileName))

}

# Calculation Functions
#   - These are functions I wrote from scratch

Calculate_Returns = function(ClosePrice = CombinedData) {
  # This function calculates the return and log return for each close price

  # Variables
  ClosePriceLag = rbind(rep(NA, ncol(ClosePrice)), head(ClosePrice, -1))
  CloseIndex = which(grepl("Close", colnames(ClosePrice)) == TRUE) # Returns which columns are the
closing prices.

  # Calculate Returns
  Return = (ClosePrice[,CloseIndex] - ClosePriceLag[,CloseIndex]) / ClosePriceLag[,CloseIndex]
  ReturnLog = log(ClosePrice[,CloseIndex]) - log(ClosePriceLag[,CloseIndex])
```

```
  Gross = ClosePrice[,CloseIndex] / ClosePriceLag[,CloseIndex]

  # Rename Columns
  colnames(Return) = gsub("Close", "Return", colnames(Return))
  colnames(ReturnLog) = gsub("Close", "Return.Log", colnames(ReturnLog))
  colnames(Gross) = gsub("Close", "Gross", colnames(Gross))

  # Combine Original Data and Returns into one data.frame
  Returns = cbind(ClosePrice, Gross, Return, ReturnLog)

  # End of Function
  return(Returns)
}

Calculate_Measures = function(VariousReturns = CombinedData, k = 252) {
  # This function will subset for returns and calculate Mean, SD, and Variance
  # for daily and yearly times.

  # Variables
  ReturnIndex = which(grepl("Return|RF", colnames(VariousReturns)) == TRUE) # Returns which columns
are the closing prices.
  VariousReturns = VariousReturns[,ReturnIndex] # Masks for columns using ReturnIndex
  Measures = c(mean, sd, var) # The measures we wish to apply to each return
  MeasuresNames = c("Mean", "StandardDeviation", "Variance") # For referencing in the later 'if'
statement

  # Sub Functions
  ApplyMeasures = function(fun, Column, na_rm = TRUE){
    # This function applys a function onto a column

    return(fun(Column,na.rm = na_rm))

  }

  Annualized = function(x, type = "Mean", k = 1) {
    # This function actualizes any returns.

    # If statement to figure out how we want to annualize based on the measure.
    if (type == "Mean") {
      R = ((x+1)^(1/((1/k))))-1

    } else if (type == "StandardDeviation") {
      R = x*sqrt(k)
```

```
  } else if (type == "Variance") {
    R = x*k

  } else {
    return(stop("Something is Wrong :("))

  }

  # End of Function
  return(as.list(R))

 }

 # Measures Daily - apply() Measures list over our Index for our VariousReturns columns.
 MeasuresDaily = lapply(Measures, function(x) apply(VariousReturns, 2, function(y) ApplyMeasures(fun
= x, Column = y)))
 names(MeasuresDaily) = paste0("Daily.",MeasuresNames)

 # Measures Yearly - mapply() iterates over MeasuresDaily and Measures name to annualize every
measure.
 MeasuresYearly = mapply(function(x, y) Annualized(x, y, k), x = MeasuresDaily, y = MeasuresNames,
SIMPLIFY = TRUE) # For this function to work, MeasuresDaily needs to be in a list.
 colnames(MeasuresYearly)=paste0("Yearly.",MeasuresNames)

 # Formatting - Making sure it's the proper format.
 MeasuresDaily = do.call(cbind, MeasuresDaily) # Combine MeasuresDaily into a data.frame
 AllMeasures = cbind(MeasuresDaily, MeasuresYearly) %>% as.data.frame() # Combine both Daily and
Yearly Measures
 rownames(AllMeasures) = gsub("(Return)\\.", "", rownames(AllMeasures), perl = TRUE) # Fix rownames
to be correct.

 AllMeasures = sapply(AllMeasures, function(x) unlist(x)) %>%  as.data.frame() # Each column is actually
a list now, so this fixes it.

 # End of Function
 return(AllMeasures)

}

Calculate_SharpeRatios = function(DailyYearlyMeasures = A1.DailyYearlyMeasures) {
 # This calculates the Sharpe ratio using vectors

 # Variables - Index for Daily and Yearly means and standard deviations.
 ReturnIndex = grepl("Daily.Mean", colnames(DailyYearlyMeasures)) # Index for each mean column
```

```
  ReturnDaily = tail(DailyYearlyMeasures[,ReturnIndex], -1) # Removing the RF row

  ReturnIndex = grepl("Yearly.Mean", colnames(DailyYearlyMeasures)) # Index for each mean column
  ReturnYearly = tail(DailyYearlyMeasures[,ReturnIndex], -1)# Removing the RF row

  StandardDeviationIndex = grepl("Daily.StandardDeviation"
                                 , colnames(DailyYearlyMeasures)) # Index for each StandardDeviation column

  StandardDeviationDaily = tail(DailyYearlyMeasures[,StandardDeviationIndex], -1) # Removing the RF
row

  StandardDeviationIndex = grepl("Yearly.StandardDeviation"
                                 , colnames(DailyYearlyMeasures)) # Index for each StandardDeviation column

  StandardDeviationYearly = tail(DailyYearlyMeasures[,StandardDeviationIndex], -1) # Removing the RF
row

  RiskFreeDaily = DailyYearlyMeasures[1,grepl("Daily.Mean", colnames(DailyYearlyMeasures))]
    # Grabing the RiskFree daily, which should be in the first row

  RiskFreeYearly = DailyYearlyMeasures[1,grepl(
    "Yearly.Mean", colnames(DailyYearlyMeasures))] # Grabing the RiskFree daily, which should be in the
first row

  # Calculate the SharpeRatio

  SharpeRatiosDaily = (ReturnDaily - RiskFreeDaily) / StandardDeviationDaily
    # Calculating the sharpe ratio for daily returns

  SharpeRatiosYearly = (ReturnYearly - RiskFreeYearly) / StandardDeviationYearly
    # Calculatin the sharpe ratio for yearly returns.

  SharpeRatios = matrix(c(SharpeRatiosDaily, SharpeRatiosYearly), nrow = length(SharpeRatiosDaily),
ncol = 2) %>%
    as.data.frame # Combining all my ratio together

  colnames(SharpeRatios) = c("Daily.SharpeRatios", "Yearly.SharpeRatios")

  rownames(SharpeRatios) = DailyYearlyMeasures[-1,] %>% rownames()
    # to name the rows I need to make sure to get rid of the risk free row name

  # End of Function
  return(SharpeRatios)
```

```
}

Create_Equal_Portfolio = function(Data = CombinedData, weight = 0.5) {
  # This function takes our dataset and returns a matrix containing the returns
  # for each asset return split equally with our risk free rate.

  # Variables
  Returns = Data[, grepl("Return", colnames(Data))]
  RiskFree = Data[,"RF"]

  # Calculate Equal Portfolio
  EqualPortfolio = (Returns*weight) + (RiskFree*(1-weight))

  # Add Date Column
  EqualPortfolio = cbind(Data$Date, EqualPortfolio)
  colnames(EqualPortfolio)[1] = "Date"

  # Rename Columns
  colnames(EqualPortfolio) = gsub("Return\\.", "Return.Equal.", colnames(EqualPortfolio))

  # End of Function
  return(EqualPortfolio)

}

Create_OMV_Portfolio = function(Data = CombinedData, Measures = A1.DailyYearlyMeasures, k = 1) {
  # This function creates weights using data and K values

  # Variables
  RiskFree = Data[, grepl("RF", colnames(Data))]

  Returns = Measures[, grepl("Mean$", colnames(Measures))]
  Returns = Returns[grepl("Return", rownames(Returns)),]

  StandardDeviations = Measures[,grepl("StandardDeviation$", colnames(Measures))]
  StandardDeviations = StandardDeviations[grepl("Return", rownames(StandardDeviations)),]

  # Check
  if(all.equal(rownames(StandardDeviations), colnames(Returns)) == FALSE){
    stop("standard deviation row names do not match return column names")
  }

  # Calculation
  # I'll do it in three separate steps
```

```
  #   1. Do Asset Returns - RiskFree
  #   2. Divide by respective standard deviation
  #   3. multiply by 1/k

  Step1 = apply(Returns, 1, function(x) -(RiskFree - x[1]))
    # x[1] is used to index our daily mean returns

  Step2 = sweep(Step1, 2, StandardDeviations[,1], FUN = '/')
    # StandardDeviatons[,1] indexes for daily standard deviation
    # sweep() allows us to divide each column by the respective standard deviation

  Step3 = Step2 * 1/k # Apply our k value

  # Rename Columns
  colnames(Step3) = gsub("Return", "Return\\.Weight", colnames(Step3))

  # End of Function
  return(as.data.frame(Step3))

}

Apply_OMV_Weights = function(RiskyWeight = A2.OMVPortfolioWeights, Returns = CombinedData) {
  # This function applies our weights onto our returns and the risk free return

  # Variables
  RiskFreeWeight = 1 - RiskyWeight # Just the complement of our risky weights

  # Returns
  RiskyReturns = Returns[,grepl("Return", colnames(Returns))] * RiskyWeight
  RiskFreeReturns = Returns[,grepl("RF", colnames(Returns))] * RiskFreeWeight
  PortfolioReturn = RiskyReturns + RiskFreeReturns

  # End of function
  return(PortfolioReturn)
}

# Wrapper Functions
#   - These are functions wrapped around a given function (ARIMA, DF, ACF,...)

Run_DF_Test = function(VectorName, lags, type) {
  # This function wraps around the ur.af function from urca

  # Variables
  Vector = CombinedData %>% select(VectorName) # Grab all my prices and returns
```

```
  Vector = Vector[!is.na(Vector), ] # omit na rows.

  # Create Empty List for Results
  ADF_Tests = vector(mode = "list", length(lags)+1)

  # Run Dickey Fuller Tests for Each Vector
  for (i in 1:length(lags)) {
    ADF_Tests[[i]] = ur.df(Vector, lags = lags[i], type = type, selectlags = c("Fixed"))
  }

  # Append Dickey Fuller test with lags selected by AIC
  ADF_Tests[[length(ADF_Tests)]] =  ur.df(Vector, type = type, selectlags = c("AIC"))

  # End of Function
  return(ADF_Tests)
}

Run_DF_Test_GLS = function(VectorName, lags, type) {
  # This function wraps around the ur.ers function from urca, this is to do GLS-ADF test

  # Variables
  Vector = CombinedData %>% select(VectorName)# Grab all my prices and returns
  Vector = Vector[!is.na(Vector), ] # omit na rows.

  # Create Empty List for Results
  ADF_Tests = vector(mode = "list", length(lags))

  # Run Dickey Fuller Tests for Each Vector
  for (i in 1:length(lags)) {
    ADF_Tests[[i]] = ur.ers(Vector, lag.max = lags[i], type = c("DF-GLS"), model = type)
  }

  # End of Function
  return(ADF_Tests)
}

Run_ACF_PACF_Ljung = function(Vector, Max_Lags){
  # This function receives a list of vectors and respective max lags to calculate
  # ACF, PACF, and Ljung pvalues

  # ACF and PACF
  ACF = mapply(function(x,y) acf(unlist(x), lag.max = y), x = Vector, y = Max_Lags, SIMPLIFY = FALSE)
  PACF = mapply(function(x,y) acf(unlist(x),type = "partial", lag.max = y), x = Vector, y = Max_Lags,
SIMPLIFY = FALSE)
```

```
 # Ljung Test
 #   - Setup a blank matrix
 #   - repeat test for 1:5 lags
 #   - format Ljung object into a table containing only p=values

 Ljung.pvalues = matrix(,0,5) # column for each lag (I transpose later)

 for( i in 1:5) {
  # For loop to apply Ljung to different lags

  Ljung = lapply(Vector, function(x) Box.test(unlist(x), lag = i, type = "Ljung-Box", fitdf = 0))
  Ljung.pvalues = append(Ljung.pvalues, t(unlist(mapply("[", Ljung)[3,])))
  rm(Ljung)

 }

 # Format
 Ljung.pvalues = t(matrix(Ljung.pvalues,length(Max_Lags),5))
 colnames(Ljung.pvalues) = names(Vector)

 # List Our three variables
 A4.ACF.PACF.Ljung = list(ACF
                  , PACF
                  , Ljung.pvalues
 )

 # End of Function
 return(A4.ACF.PACF.Ljung)
}

Run_ARIMA = function(Vector, diff) {
 # Setup empty list for all our models
 Models = vector("list", length(Vector)) # setup empty list for each vector
 Models = lapply(Models, function(x) vector("list", 5*5)) # setup empty lists for each model

 # Run ARIMA function in a loop for various p and q numbers
 for (i in 1:length(Vector)) {
  Index = 1
  print(paste("Starting...", names(Vector)[Index]))

  for (p in 1:5) {
   for(q in 1:5) {
```

```
      print(paste(i,p,q))

      model = arima(Vector[[i]], c(p,diff[i],q), method = "ML")
      order = paste(p,diff[i],q)

      Models[[i]][[Index]] = list(model,order)

      Index = Index + 1

    }
   }
  }

  # Name list at index 1
  names(Models) = names(Vector)

  # End of Function
  return(Models)
}

Run_AIC_BIC = function(Models) {
  # This function runs the AIC and BIC calculations for our ARIMA models in Section 4.

  # AIC
  AIC = lapply(Models, function(x) sapply(x, function(y) AIC(y[[1]])))
  AIC = do.call(rbind, AIC)
  AIC.MIN = apply(AIC, 1, function(x) min(x))
  AIC.MIN.Index = apply(AIC, 1, function(x) which(x == min(x)))

  # BIC
  BIC = lapply(Models, function(x) sapply(x, function(y) AIC(y[[1]], k = log(length(y[[1]]$residuals)))))
  BIC = do.call(rbind, BIC)
  BIC.MIN = apply(BIC, 1, function(x) min(x))
  BIC.MIN.Index = apply(BIC, 1, function(x) which(x == min(x)))

  # Create Lists for Both
  AIC_List = list(AIC
                  , AIC.MIN
                  , AIC.MIN.Index
  )
  names(AIC_List) = c("Result", "Min", "Index")


  BIC_list = list(BIC
```

```
            , BIC.MIN
            , BIC.MIN.Index
  )
  names(BIC_list) = c("Result", "Min", "Index")

  # Combine Lists
  AIC.BIC = list(AIC_List, BIC_list)
  names(AIC.BIC) = c("AIC", "BIC")



  # End of Function
  return(AIC.BIC)
}

Run_AIC_BIC_2 = function(Models) {
  # This function runs the AIC and BIC calculations for our GARCH models in Section 5.

  # AIC
  AIC = lapply(Models, function(x) sapply(x, function(y) infocriteria(y[[1]])[1,1]))
  AIC = do.call(rbind, AIC)
  AIC.MIN = apply(AIC, 1, function(x) min(x))
  AIC.MIN.Index = apply(AIC, 1, function(x) which(x == min(x)))

  # BIC
  BIC = lapply(Models, function(x) sapply(x, function(y) infocriteria(y[[1]])[2,1]))
  BIC = do.call(rbind, BIC)
  BIC.MIN = apply(BIC, 1, function(x) min(x))
  BIC.MIN.Index = apply(BIC, 1, function(x) which(x == min(x)))

  # Create Lists for Both
  AIC_List = list(AIC
            , AIC.MIN
            , AIC.MIN.Index
  )
  names(AIC_List) = c("Result", "Min", "Index")


  BIC_list = list(BIC
            , BIC.MIN
            , BIC.MIN.Index
  )
  names(BIC_list) = c("Result", "Min", "Index")
```

```
  # Combine Lists
  AIC.BIC = list(AIC_List, BIC_list)
  names(AIC.BIC) = c("AIC", "BIC")

  # End of Function
  return(AIC.BIC)

}

# Plotting Functions
#   - These function help me plot my data structures easier

GroupByKey = function(x, KeyName = "Return") {
  # This function converts our data into long format by a ke

  # Variables
  Colnames = colnames(x[,grepl(KeyName, colnames(x))])

  # Convert Data
  PlotingData = gather(x, Key, Key.Value, Colnames)
  PlotingData = PlotingData[order(PlotingData$Key),]

  # Column Names
  colnames(PlotingData) = c("Date", paste0(KeyName, ".Asset"), KeyName)

  # End of Function
  return(PlotingData)

}

PlotbyList = function(PlotData, x, y, color, title, index) {
  # This function takes in long data in a list and creates/exports graphs

  gtitle = paste0(title,index)
  gxtitle = paste0("Plot - ",title,index, ".png")

  Plot = ggplot(data = PlotData, mapping = aes_string(x = x, y = y, color = color)) +
    geom_point(size = 0.5) +
    geom_line(size = 0.5) +
    ggtitle(gtitle)

  paste0("Printing ", gxtitle, "...") %>% print()
  PlotData %>% head %>% print
```

```
  ggsave(device = "png", plot = Plot, filename = gxtitle)

}

PlotResidualsSquared = function(data, names) {
  # This function plots and exports residuals^2 in my lists


  title = paste0("Residual^2 ",names)

  Plot = ggplot(data, aes(x = date, y = residual)) +
    geom_bar(stat='identity', width = 1.5) +
    ggtitle(title)


  ggsave(device = "png", plot = Plot, filename = paste0("Residual ", names, ".png"))

}

#================================================================================
# Directories
#================================================================================
# Below is the data you will need to run this file.
#   - "ENB" : from yahoo finance for ENB (NYSE),
https://finance.yahoo.com/quote/ENB/history?period1=448156800&period2=1642464000&interval=1d
&filter=history&frequency=1d&includeAdjustedClose=true
#   - "MSFT" : from yahoo finance for MSFT(NasdaqG5),
https://finance.yahoo.com/quote/MSFT/history?period1=511056000&period2=1642636800&interval=1
d&filter=history&frequency=1d&includeAdjustedClose=true
#   - "F-F_Research_Data_Factors_Daily.CSV" : from Ken French's site download link for "Fama/French 3
Factors [Daily]", http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
#   - "oxfordmanrealizedvolatilityindices.csv" : from https://realized.oxford-
man.ox.ac.uk/data/download, the top download link

# Set your working directory
Dir = "E:\\My Stuff\\Main School Folder\\4_Year Four\\Semester 8\\FIN 4100 - Financial
Econometrics\\Assignments\\Final Project\\"
setwd(Dir)

# These are the default names of the files we need to download, do not change.
FileDir = list(
  RiskyAsset1 = "ENB.csv",
  RiskyAsset2 = "MSFT.csv",
  RiskFree = "F-F_Research_Data_Factors_daily.CSV",
```

```
  RealizedVolatility = "oxfordmanrealizedvolatilityindices.csv"
)


#================================================================================
# Data Import
#================================================================================

# Call my Excel_)Import function and list every file we need.
FileImports = Excel_Import(FileDir)
names(FileImports) = names(FileDir)


#================================================================================
# Data Manipulation
#================================================================================

# 1. For each data set we need to subset certain columns.
#   - RiskyAsset1: Date, Close price, and Adj Close Price
#   - RiskyAsset2: Date, Close price, and Adj Close Price
#   - RiskFree: Date and RF, and remove the copyright at the bottom
#   - RealizedVolatility: Date, SPX from Symbol Col, close price, rv10, and rv5

RiskyAsset1 = as.data.frame(FileImports[[1]][[1]])
RiskyAsset1[,2] = FileImports[[1]][[5]]
RiskyAsset1[,3] = FileImports[[1]][[6]]
names(RiskyAsset1) = c("Date", "Close.ENB", "Close.Adj.ENB")

RiskyAsset2 = as.data.frame(FileImports[[2]][[1]])
RiskyAsset2[,2] = FileImports[[2]][[5]]
RiskyAsset2[,3] = FileImports[[2]][[6]]
names(RiskyAsset2) = c("Date", "Close.MSFT", "Close.Adj.MSFT")

RiskFree = as.data.frame(FileImports[[3]][[1]])
RiskFree[,2] = as.data.frame(FileImports[[3]][[5]])
RiskFree = RiskFree[-nrow(RiskFree),1:2] # This is to delete the last row containing the copyright
names(RiskFree) = c("Date", "RF")

RealizedVolatility = as.data.frame(FileImports[[4]][[1]])
RealizedVolatility[,2] = as.data.frame(FileImports[[4]][[2]])
RealizedVolatility[,3] = as.data.frame(FileImports[[4]][[5]])
RealizedVolatility[,4] = as.data.frame(FileImports[[4]][[10]])
RealizedVolatility[,5] = as.data.frame(FileImports[[4]][[19]])
names(RealizedVolatility) = c("Date","Symbol","Close.SPX","rv10","rv5")
```

```
# 2. Now we format each data set.
#   - RiskyAsset1: Date just needs to be formatted
#   - RiskFree: Date needs to be formatted and divide RiskFree$RF by 100 to turn it into a decimal
#   - RealizedVolatility: Date format and symbol subset, I'm also going to drop
#   - the Symbol Column, I won't be needing it anymore.

RiskyAsset1[["Date"]] = as.Date(RiskyAsset1[["Date"]])

RiskyAsset2[["Date"]] = as.Date(RiskyAsset2[["Date"]])

RiskFree[["Date"]] = ymd(RiskFree[["Date"]])
RiskFree$RF = RiskFree$RF/100

RealizedVolatility[["Date"]] = as.Date(RealizedVolatility[["Date"]])
RealizedVolatility = RealizedVolatility[grepl("\\.SPX",
RealizedVolatility[["Symbol"]]),1:ncol(RealizedVolatility)] # Logical mask for S&P 500 rows.
RealizedVolatility = subset(RealizedVolatility, select = -2)


# 4. Combine 4 Data Sets onto common dates.
#   - I'm going to include as much data as I can for each data set
#   but I'll make a cut off point at the oldest date for my two stocks which is ENB/RiskyAsset1 at 1984-03-
15.
#   - Then I'll just rm these datasets from the environment to clean it up.

CombinedData = merge(RiskyAsset1,RiskyAsset2,by.x="Date",by.y="Date",sort=TRUE, all = TRUE)
CombinedData = merge(CombinedData,RealizedVolatility,by.x="Date",by.y="Date",sort=TRUE, all =
TRUE)
CombinedData = merge(CombinedData,RiskFree,by.x="Date",by.y="Date",sort=TRUE, all = TRUE)
CombinedData = CombinedData[CombinedData[["Date"]] >= RiskyAsset1[[1]][[1]],]

# Clean Up - won't be needing these anymore
rm(RiskyAsset1, RiskyAsset2, RiskFree, RealizedVolatility, FileDir, FileImports)

#================================================================================
# Calculations for Section One
#================================================================================

# 1. Calculate the Returns for each closing Price
#   - Using our CombinedData variable, I'll subset for a Closing price matrix
#   and calculate for returns and log returns.
```

```
CombinedData = Calculate_Returns(CombinedData)


# 2. Calculate over our Returns using some Measures and annulize it.
#   - We're going to apply the functions mean, sd, and var to each return column
#   - Then we annualize it and put it all into a matrix

A1.DailyYearlyMeasures = Calculate_Measures(CombinedData)


# 3. Calculate the Sharpe Ratio for each return against RF
# Simply take each return vector, subtract RF, and divide by standard deviation vector

A1.SharpeRatios = Calculate_SharpeRatios(A1.DailyYearlyMeasures)

#================================================================================
# Calculations for Section Two
#================================================================================

# 1. Create equal weighted portfolios between our risky and risk free asset.
# Creating an equal portfolio with each possible asset using various weights.

# Variables - This is a vector containing what weights we'd like to apply
A2.EqualPortfolioWeighting = c(0.5, 0.75, 0.9)

# Function Calls - Iterate over weighting to create returns, measures, and Sharpe ratios.
A2.EqualPortfolios          = lapply(A2.EqualPortfolioWeighting, function(x)
Create_Equal_Portfolio(CombinedData, x))
A2.EqualPortfolioMeasures     = lapply(A2.EqualPortfolios, function(x)
Calculate_Measures(cbind(CombinedData$RF,x))) # Include RF to include into the Sharpe ratio
calculations.
A2.EqualPortfolioSharpeRatios = lapply(A2.EqualPortfolioMeasures, function(x)
Calculate_SharpeRatios(x))


# 2. Create Optimal Mean Variance Portfolios
# Each portfolio is based on a single asset and our risk free rate

# Variables
# This is the different k values we will run
K = c(1,3,7)

# Function Calls - Iterate over K to create weights for our returns
# First we'll create our OMW portfolio weights
```

```
# Then we'll apply these weights to create our portfolio returns.

A2.OMVPortfolioWeights = lapply(K, function(x) Create_OMV_Portfolio(CombinedData,
A1.DailyYearlyMeasures, k = x))
A2.OMVPortfolioWeightsMeasures = lapply(A2.OMVPortfolioWeights, function(x)
Calculate_Measures(x)) # We can reuse an Section one function to create our measures

A2.OMVPortfolioReturns = lapply(A2.OMVPortfolioWeights, function(x)
Apply_OMV_Weights(RiskyWeight = x, Returns = CombinedData))
A2.OMVPortfolioReturns = lapply(A2.OMVPortfolioReturns, function(x) { # This is to change the
colnames for each list item.
   colnames(x) = gsub("Weight.", "", colnames(x))
   return(x)
  }
)

A2.OMVPortfolioReturnsMeasures = lapply(A2.OMVPortfolioReturns, function(x)
Calculate_Measures(x))

#================================================================================
# Calculations for Section Three
#================================================================================

VectorNames = c("Close.Adj.ENB"
          , "Close.Adj.MSFT"
          , "Close.SPX"
          , "Return.Adj.ENB"
          , "Return.Adj.MSFT"
          , "Return.SPX")

# Find Max Lags

A3.Max.Lags = sapply(VectorNames, function(x) {
  T = nrow(CombinedData[x] %>% na.omit())
  pmax = 12 * (T/100)**(1/4)

  return(floor(pmax))

})


# Run Dickey Fuller Tests
```

```
# This Code will run our wrapper function and list all our tests: We run this three times for each type; none, drift, and trend.

# A3.ADF_Tests = lapply(VectorNames
#                , function(x) Run_DF_Test(VectorName = x
#                                          , lags =  c(1,7,30,60,90)
#                                          , type = "none"))
#
# names(A3.ADF_Tests) = VectorNames

# A3.ADF_Tests = mapply(function(x,y) {Run_DF_Test(VectorName = x
#                                           , lags =  c(1,7,y,40)
#                                           , type = "none")
# }, x = VectorNames
# , y = A3.Max.Lags
# , SIMPLIFY = FALSE)
#
# names(A3.ADF_Tests) = VectorNames


### Normal DF tests

A3.ADF_Tests = mapply(function(x,y) {Run_DF_Test(VectorName = x
                                          , lags =  c(1,7,y,40)
                                          , type = "none")
}, x = VectorNames
, y = A3.Max.Lags
, SIMPLIFY = FALSE)

A3.ADF_Tests1 = lapply(A3.ADF_Tests, function(x) mapply(function(y) summary(y)@teststat, y = x))

A3.ADF_Tests_TValues1 = do.call(rbind, A3.ADF_Tests1) %>% data.frame



A3.ADF_Tests = mapply(function(x,y) {Run_DF_Test(VectorName = x
                                          , lags =  c(1,7,y,40)
                                          , type = "drift")
}, x = VectorNames
, y = A3.Max.Lags
, SIMPLIFY = FALSE)

A3.ADF_Tests2 = lapply(A3.ADF_Tests, function(x) mapply(function(y) summary(y)@teststat, y = x))
```

```
A3.ADF_Tests_TValues2 = do.call(rbind, A3.ADF_Tests2) %>% data.frame


A3.ADF_Tests = mapply(function(x,y) {Run_DF_Test(VectorName = x
                                  , lags =  c(1,7,y,40)
                                  , type = "trend")
}, x = VectorNames
, y = A3.Max.Lags
, SIMPLIFY = FALSE)

A3.ADF_Tests3 = lapply(A3.ADF_Tests, function(x) mapply(function(y) summary(y)@teststat, y = x))

A3.ADF_Tests_TValues3 = do.call(rbind, A3.ADF_Tests3) %>% data.frame



### GLS ADF Tests


A3.GLS.ADF_Tests = mapply(function(x,y) {Run_DF_Test_GLS(VectorName = x
                                  , lags =  c(1,7,y,40)
                                  , type = "constant")
}, x = VectorNames
, y = A3.Max.Lags
, SIMPLIFY = FALSE)

A3.ADF_Tests_GLS1 = lapply(A3.GLS.ADF_Tests, function(x) mapply(function(y) summary(y)@teststat, y
= x))
A3.ADF_Tests_GLS_TValues1 = do.call(rbind, A3.ADF_Tests_GLS1) %>% data.frame

A3.GLS.ADF_Tests = mapply(function(x,y) {Run_DF_Test_GLS(VectorName = x
                                    , lags =  c(1,7,y,40)
                                    , type = "trend")
}, x = VectorNames
, y = A3.Max.Lags
, SIMPLIFY = FALSE)

A3.ADF_Tests_GLS2 = lapply(A3.GLS.ADF_Tests, function(x) mapply(function(y) summary(y)@teststat, y
= x))
A3.ADF_Tests_GLS_TValues2 = do.call(rbind, A3.ADF_Tests_GLS2) %>% data.frame


#=============================================================================
# Calculations for Section Four
```

```
#=============================================================================

# Variables
VectorNames = c("Close.Adj.ENB"
            , "Close.Adj.MSFT"
            , "Close.SPX"
            , "Return.Adj.ENB"
            , "Return.Adj.MSFT"
            , "Return.SPX"
            , "RF"
            , "rv10")


A4.Vectors = lapply(VectorNames, function(x) as.ts(CombinedData[!is.na(CombinedData[x]), x])) # Time Series Data for ARIMA and ARMA
names(A4.Vectors) = VectorNames


# Find Maximum Lags for Each Vector (Risky Asset Prices and Returns)

A4.Max.Lags = sapply(VectorNames, function(x) {
  as.numeric(floor(10 * log(length(CombinedData[!is.na(CombinedData[[x]]), x]))))
})

# Calculate ACF, PACF, Ljung-Box

A4.ACF.PACF.Ljung.1 = Run_ACF_PACF_Ljung(A4.Vectors, A4.Max.Lags) # This is using our initial risky asset prices and returns


# Setup Each ARMA and ARIMA Model

A4.Models.1 = Run_ARIMA(A4.Vectors, c(1,1,1,0,0,0,1,0))


# Find AIC BIC

AIC.BIC.1 = Run_AIC_BIC(A4.Models.1)


# According to our AIC and BIC, these are the best models for each vector

AIC.Model = mapply(function(x,y) x[[y]][[2]], x = A4.Models.1, y = AIC.BIC.1[[1]][[3]] , SIMPLIFY = FALSE)
BIC.Model = mapply(function(x,y) x[[y]][[2]], x = A4.Models.1, y = AIC.BIC.1[[2]][[3]], SIMPLIFY = FALSE)
```

```
AIC.Model = do.call(rbind, AIC.Model)
BIC.Model = do.call(rbind, BIC.Model)

A4.AIC.BIC.1 = cbind(names(A4.Models.1),AIC.BIC.1[[1]][[2]], AIC.Model, AIC.BIC.1[[2]][[2]], BIC.Model)
colnames(A4.AIC.BIC.1) = c("VectorNames", "AIC", "AIC.Model", "BIC", "BIC.Model")


# Residual Vectors from our best models

A4.AIC.Residuals = mapply(function(x,y) x[[y]][[1]][["residuals"]], x = A4.Models.1, y = AIC.BIC.1[[1]][[3]],
SIMPLIFY = FALSE)
A4.BIC.Residuals = mapply(function(x,y) x[[y]][[1]][["residuals"]], x = A4.Models.1, y = AIC.BIC.1[[2]][[3]],
SIMPLIFY = FALSE)


# Run ACF, PACF, and Ljung for AIC and BIC models

A4.ACF.PACF.Ljung.AIC.1 = Run_ACF_PACF_Ljung(A4.AIC.Residuals, A4.Max.Lags)
A4.ACF.PACF.Ljung.BIC.1 = Run_ACF_PACF_Ljung(A4.BIC.Residuals, A4.Max.Lags)


# Refine Our ARIMA models for MSFT and SPX prices

# Variables

A4.Models.2 = vector("list", 2) # setup empty list for each vector
A4.Models.2 = lapply(A4.Models.2, function(x) vector("list", 5*5)) # setup empty lists for each model
names(A4.Models.2) = VectorNames[2:3]

A4.Vectors = lapply(VectorNames[2:3], function(x) as.ts(CombinedData[!is.na(CombinedData[x]), x]))


# Create Models Using Seasonal Argument for ARIMA

for (i in 1:2) {
  Index = 0

  for (p in 1:5) {

    for(q in 1:5) {

      print(paste(i,p,q))
```

```
    Index = Index + 1
    model = arima(A4.Vectors[[i]], c(p,1,q), seasonal = c(1,1,0))
    order = paste(p,1,q)

    A4.Models.2[[i]][[Index]] = list(model,order)

   }
  }
}


# Tests, AIC and BIC

AIC.BIC.2 = Run_AIC_BIC(A4.Models.2)

AIC.Model = mapply(function(x,y) x[[y]][[2]], x = A4.Models.2, y = AIC.BIC.2[[1]][[3]], SIMPLIFY = FALSE)
BIC.Model = mapply(function(x,y) x[[y]][[2]], x = A4.Models.2, y = AIC.BIC.2[[2]][[3]], SIMPLIFY = FALSE)

AIC.Model = do.call(rbind, AIC.Model)
BIC.Model = do.call(rbind, BIC.Model)

A4.AIC.BIC.2 = cbind(VectorNames[2:3],AIC.BIC.2[[1]][[2]], AIC.Model, AIC.BIC.2[[2]][[2]], BIC.Model)
colnames(A4.AIC.BIC.2) = c("VectorNames", "AIC", "AIC.Model", "BIC", "BIC.Model")

A4.AIC.Seasonal.Residuals = mapply(function(x,y) x[[y]][[1]][["residuals"]], x = A4.Models.2, y =
AIC.BIC.2[[1]][[3]], SIMPLIFY = FALSE)
A4.BIC.Seasonal.Residuals = mapply(function(x,y) x[[y]][[1]][["residuals"]], x = A4.Models.2, y =
AIC.BIC.2[[2]][[3]], SIMPLIFY = FALSE)

A4.ACF.PACF.Ljung.AIC.2 = Run_ACF_PACF_Ljung(A4.AIC.Seasonal.Residuals, A4.Max.Lags[2:3])
A4.ACF.PACF.Ljung.BIC.2 = Run_ACF_PACF_Ljung(A4.BIC.Seasonal.Residuals, A4.Max.Lags[2:3])

# Some Cleanup

rm(AIC.Model, BIC.Model, p, q, i, Index, order)


#=============================================================================
# Calculations for Section Five
#=============================================================================

# We need to take our residuals from our earlier chosen models
#   - Our AIC models preformed better so we'll just be reusing those.
#   - We'll also take our seasonal AIC model residuals
```

```
A5.Residuals = A4.AIC.Residuals

# A5.Residuals[2:3] = A4.AIC.Seasonal.Residuals

# Create our Squared Residuals

A5.Residuals.Squared = lapply(A5.Residuals, function(x) x**2)

# Convert from time series to data frames, for plotting purposes.

A5.Residuals.Squared.DF = mapply(function(x,y) {
  # Function to Create Residual Squares

  # Setup Variables to proper format
  df = x %>% as.numeric()
  df = df %>% data.frame()

  # Add proper dates to residuals
  date = CombinedData %>% select(y, "Date") %>%  na.omit()
  df$date = date[,2]

  # Rename columns
  colnames(df) = c("residual", "date")

  # End of Function
  return(df)
}
, x = A5.Residuals.Squared
, y = VectorNames
, SIMPLIFY = FALSE

)

# ACF PACF and Ljung for my Residuals Squared
A5.ACF.PACF.Ljung.1 = Run_ACF_PACF_Ljung(A5.Residuals.Squared, A4.Max.Lags)

# Optimal ARIMA Model Specification
#   - I need to grab my optimal orders from prior tests (A4.AIC.BIC.1 and 2)

A5.Order = lapply(A4.AIC.BIC.1[,3], function(x) {
  c(substr(x,1,1)
    , substr(x,5,5)) %>% as.numeric()
```

```
})

# A5.Order.temp = lapply(A4.AIC.BIC.2[,3], function(x) {
#   c(substr(x,1,1)
#     , substr(x,5,5)) %>% as.numeric()
#
# })
#
# A5.Order[2:3] = A5.Order.temp
# rm(A5.Order.temp)

A5.Optimal.ARIMA = lapply(A5.Order, function(x) arfimaspec(mean.model = list(armaOrder =
as.vector(x), include.mean = TRUE)))
A5.Optimal.Fit = mapply(function(x,y) arfimafit(x,y), x = A5.Optimal.ARIMA, y = A5.Residuals.Squared)

# ARCH and GARCH
#   - Setup xts data
#   - Run loop for various p and q combinations for GARCH model

# A5.Vectors = lapply(VectorNames, function(x) data.frame(x = CombinedData[!is.na(CombinedData[x]),
x], Date = CombinedData[!is.na(CombinedData[x]), "Date"])) # Time Series Data for ARIMA and ARMA
# names(A5.Vectors) = VectorNames

A5.XTS = lapply(A5.Residuals.Squared.DF, function(x) xts(x[,1], order.by = x[,2]))

A5.GARCH.Models = vector("list", length(A5.Residuals.Squared.DF)) # setup empty list for each vector
A5.GARCH.Models = lapply(A5.GARCH.Models, function(x) vector("list", 5*5)) # setup empty lists for
each model

for (i in 1:length(A5.Residuals.Squared.DF)) {
  # Loop to run and format various GARCH models

  # Variables
  Index = 1
  print(paste("Starting...", names(A5.Residuals.Squared.DF)[Index]))

  # For loop to run various p and q
  for (p in 1:5) {

    for(q in 1:5) {

      print(paste(i,p,q))

      # Model fitted to a package specific class
```

```
    model = ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(p,q))
                       , mean.model = list(aeamaOrder = A5.Order[[i]], include.mean = TRUE))


    # Fit Model to Data
    A5.GARCH.Models[[i]][[Index]] = list(ugarchfit(model, unlist(A5.XTS[[i]]), solver = "hybrid")) # Fi
    A5.GARCH.Models[[i]][[Index]][[2]] = c(p,q)

    Index = Index + 1

   }
  }

}


# Find AIC, BIC, and best models.
A5.GARCH.Models.backup = A5.GARCH.Models
A5.GARCH.Models = A5.GARCH.Models[1:6]


AIC.BIC.3 = Run_AIC_BIC_2(A5.GARCH.Models)

AIC.Model = mapply(function(x,y) x[[y]][[2]], x = A5.GARCH.Models, y = AIC.BIC.3[[1]][[3]], SIMPLIFY =
FALSE)
BIC.Model = mapply(function(x,y) x[[y]][[2]], x = A5.GARCH.Models, y = AIC.BIC.3[[2]][[3]], SIMPLIFY =
FALSE)

AIC.Model = do.call(rbind, AIC.Model)
BIC.Model = do.call(rbind, BIC.Model)

A5.AIC.BIC.1 = cbind(VectorNames[1:6], AIC.BIC.3[[1]][[2]], AIC.Model, AIC.BIC.3[[2]][[2]], BIC.Model)
colnames(A5.AIC.BIC.1) = c("VectorNames", "AIC", "p", "q", "BIC", "p", "q")

A5.AIC.GARCH.Residuals = mapply(function(x,y) residuals(x[[y]][[1]], standardize = TRUE)**2, x =
A5.GARCH.Models, y = AIC.BIC.3[[1]][[3]], SIMPLIFY = FALSE)
#A5.BIC.GARCH.Residuals = mapply(function(x,y) residuals(x[[y]][[1]], standardize = TRUE)**2, x =
A5.GARCH.Models, y = AIC.BIC.3[[2]][[3]], SIMPLIFY = FALSE)

A5.ACF.PACF.Ljung.AIC.GARCH = Run_ACF_PACF_Ljung(A5.AIC.GARCH.Residuals, A4.Max.Lags[1:6])
#A5.ACF.PACF.Ljung.BIC.GARCH = Run_ACF_PACF_Ljung(A5.BIC.GARCH.Residuals, A4.Max.Lags[1:6])

# Coefficients and Standard Errors
```

```
A5.GARCH.COEF = mapply(function(x,y) coef(x[[y]][[1]]), x = A5.GARCH.Models, y = AIC.BIC.3[[1]][[3]],
SIMPLIFY = FALSE)
A5.GARCH.VCOV = mapply(function(x,y) vcov(x[[y]][[1]]), x = A5.GARCH.Models, y = AIC.BIC.3[[1]][[3]],
SIMPLIFY = FALSE)
A5.GARCH.SE   = lapply(A5.GARCH.VCOV, function(x) sqrt(diag(x)))

A5.GARCH.COEF = data.frame(do.call(rbind, A5.GARCH.COEF))
A5.GARCH.SE = data.frame(do.call(rbind, A5.GARCH.SE))

GARCH.Data = list(A5.GARCH.COEF, A5.GARCH.SE)
names(GARCH.Data) = c("COEF", "SE")

Excel_Export(GARCH.Data, FileName = "coefVcovSe.xlsx")

# GARCH Forecast

A5.GARCH.Forecast = mapply(function(x,y,z) {
  # Function to run GARCH forecasts

  # Setup Variables
  optimal = x[[y]][[1]] # return optimal model based on AIC/BIC
  spec = getspec(optimal) # convert model to specific package class
  setfixed(spec) = as.list(coef(optimal)) # format for package to use
  forecast = ugarchforecast(spec, data = z, n.ahead = 1, n.roll = 2000, out.sample = 2000) # run forecast w
optimal GARCH
  condVarForecast = sigma(optimal)

  return(list(spec,forecast,condVarForecast))

}
  , x = A5.GARCH.Models
  , y = AIC.BIC.3[[1]][[3]], SIMPLIFY = FALSE
  , z = A5.XTS[1:6]
)

A5.GARCH.Forecast[[6]][[2]] %>% plot


# Portfolio Allocation

mrisky = CombinedData[["Return.Adj.ENB"]] %>% na.omit() %>% mean()
mriskless = CombinedData[["RF"]] %>% na.omit() %>% mean()
k = 1
vrisky = CombinedData[["Return.Adj.ENB"]] %>% na.omit() %>% var()
```

```
optimalweightriskless = (mrisky - mriskless) / (k*vrisky)


A5.Return.Vectors = lapply(VectorNames[4:6], function(x) as.ts(CombinedData[!is.na(CombinedData[x]),
x]))

A5.Porfolio.Allocation = mapply(function(x,y) {
  # Function to use GARCH forecasted volatility to create OMV portfolios

  # Risky and Riskless Assets
  mean_risky = x %>% mean()
  mean_riskless = CombinedData[["RF"]] %>% na.omit() %>% mean()

  # Risk Aversions
  k = c(1,5,7)

  # Variances
  variance_risky = x %>% var()
  max_volitility = y[[3]] %>% max()
  min_volitility = y[[3]] %>% min()

  # Optimal Weights Calculations
  Optimal_Weights_Var = (mean_risky - mean_riskless) / (k*variance_risky)
  Optimal_Weights_Max = (mean_risky - mean_riskless) / (k*max_volitility)
  Optimal_Weights_Min = (mean_risky - mean_riskless) / (k*min_volitility)

  # End of Function
  return(list(Optimal_Weights_Var, Optimal_Weights_Max, Optimal_Weights_Min))
}
, x = A5.Return.Vectors
, y = A5.GARCH.Forecast[4:6]
, SIMPLIFY = FALSE
)


A5.Porfolio.Allocation.Matrix = lapply(A5.Porfolio.Allocation, function(x) do.call(rbind, x))
A5.Porfolio.Allocation.Matrix = do.call(rbind, A5.Porfolio.Allocation.Matrix)
A5.Porfolio.Allocation.Matrix = data.frame(A5.Porfolio.Allocation.Matrix)

write.csv(A5.Porfolio.Allocation.Matrix, "port allo matx.csv")


#================================================================================
```

```
# Plots for Section One
#================================================================================

# 1. Setting Up Long Data
#   - For 1 and 2, I'm setting up long data to capture each Return or Close price
#   as a separate class for my gather() function.

Colnames1 = colnames(CombinedData[,grepl("Return|RF", colnames(CombinedData))])
Colnames2 = colnames(CombinedData[,grepl("Close", colnames(CombinedData))])

PlotData = gather(CombinedData, Return.Asset, Return, Colnames1)
PlotData = PlotData[order(PlotData$Return.Asset),]

PlotData2 = gather(CombinedData, Close.Asset, Close, Colnames2)
PlotData2 = PlotData2[order(PlotData2$Close.Asset),]


# 2. Plots for Wider Data
#   - Returns: Split into regular returns and log returns.
#   - Close Prices: Graph one normally and use log on the other.

LogIndex = grepl("Log", PlotData$Return.Asset)
RFIndex = grepl("RF", PlotData$Return.Asset)

ggplot(data = PlotData[which(!LogIndex == TRUE | RFIndex == TRUE),], mapping = aes(x = Date, y =
Return, color = Return.Asset)) +
  geom_point(size = 0.5) +
  geom_line(size = 0.5) +
  ggtitle("All Returns")
  ggsave(filename = "Plot - All Returns.png")

ggplot(data = PlotData[which(LogIndex == TRUE | RFIndex == TRUE),], mapping = aes(x = Date, y =
Return, color = Return.Asset)) +
  geom_point(size = 0.5) +
  geom_line(size = 0.5) +
  ggtitle("All Log Returns")
  ggsave(filename = "Plot - All Log Returns.png")

ggplot(data = PlotData2, mapping = aes(x = Date, y = Close, color = Close.Asset)) +
  geom_point(size = 0.5) +
  geom_line(size = 0.5) +
  ggtitle("All Close Prices")
  ggsave(filename = "Plot - All Closes.png")
```

```
ggplot(data = PlotData2, mapping = aes(x = Date, y = log(Close), color = Close.Asset)) +
  geom_point(size = 0.5) +
  geom_line(size = 0.5) +
  ggtitle("All Log Close Prices")
  ggsave(filename = "Plot - All Log Closes.png")


# 3. Plots for Each column from our CombinedData variable and export to our working directory.

for(i in 1:ncol(CombinedData)) {

  if(i == ncol(CombinedData)) {
    i = 1
    next()

  } else {

    PlotName = paste0("Plot - ", colnames(CombinedData)[i+1], ".png")
    yaxis = {
      if(grepl("Close", colnames(CombinedData)[i+1])) {
        "Close"

      } else if (grepl("Return|RF", colnames(CombinedData)[i+1])) {
        "Return"
      }
    }

    print(ggplot(data = CombinedData, mapping = aes(x = Date, y = CombinedData[,(i+1)])) +
          geom_point(alpha = 0.05) +
          geom_line()) +
      ylab(label = yaxis) +
      ggtitle(paste0("Plot - ", colnames(CombinedData)[i+1]))
    ggsave(file=PlotName)

  }
}


#============================================================================
# Plots for Section Two
#============================================================================

# 1. Plot Portfolio Returns and Weights
```

```
# Optimal Mean Variance Returns

PlotDataOMV = lapply(OMVPortfolioReturns, function(x)
GroupByKey(cbind(CombinedData$Date,x),"Return"))
PlotDataOMV = lapply(PlotDataOMV, function(x) { # Clean for just log.adj and log.spx
  x = x[grepl("Log.Adj|Log.SPX", x[,2]),]
  return(x)
})


for (i in seq_along(PlotDataOMV)) {

  ggplot(data = PlotDataOMV[[i]], mapping = aes(x = Date, y = Return, color = Return.Asset)) +
    geom_point(size = 0.5) +
    geom_line(size = 0.5) +
    ggtitle(paste0("All OMV Returns ", "K = ", K[i]))

  ggsave(filename = paste0("Plot - All OMV Returns ", "K = ", K[i], ".png"))

}

# Optimal Mean Variance Weights

PlotDataOMVWeight = lapply(A2.OMVPortfolioWeights, function(x)
GroupByKey(cbind(CombinedData$Date,x %>% as.data.frame()), "Weight"))
PlotDataOMVWeight = lapply(PlotDataOMVWeight, function(x) { # Clean for just log.adj and log.spx
  x = x[grepl("Log.Adj|Log.SPX", x[,2]),]
  return(x)
})


for (i in seq_along(PlotDataOMVWeight)) {

  ggplot(data = PlotDataOMVWeight[[i]], mapping = aes(x = Date, y = Weight, color = Weight.Asset)) +
    geom_point(size = 0.5) +
    geom_line(size = 0.5) +
    ggtitle(paste0("All OMV Weights ", "K = ", K[i]))

  ggsave(filename = paste0("Plot - All OMV Weights ", "K = ", K[i], ".png"))

}

# Equal Portfolio Returns
```

```
PlotDataEPReturns = lapply(A2.EqualPortfolios, function(x) GroupByKey(x,"Return"))
PlotDataEPReturns = lapply(PlotDataEPReturns, function(x) { # Clean for just log.adj and log.spx
  x = x[grepl("Log.Adj|Log.SPX", x[,2]),]
  return(x)
})


for (i in seq_along(A2.EqualPortfolios)) {

  ggplot(data = PlotDataEPReturns[[i]], mapping = aes(x = Date, y = Weight, color = Weight.Asset)) +
    geom_point(size = 0.5) +
    geom_line(size = 0.5) +
    ggtitle(paste0("All EP Returns ", "Weight = ", A2.EqualPortfolioWeighting[i]))

  ggsave(filename = paste0("Plot - All EP Returns ", "Weight = ", A2.EqualPortfolioWeighting[i], ".png"))

}


# New Alternative for Plotting

mapply(function(l, m)
    PlotbyList(
      PlotData = l,
      x = "Date",
      y = "Weight",
      color = "Weight.Asset",
      title ="All OMW Weights, K = ",
      index = m),
  l = PlotDataOMVWeight,
  m = K,
  SIMPLIFY = FALSE
)

# If I want to edit the data going in I can use the below code for the data argument in ggplot
# PlotDataOMVWeight[[i]][grepl("ENB$", PlotDataOMVWeight[[i]][,"Weight.Asset"]),]


#==============================================================================
# Plots for Section Three
#==============================================================================

# Price Plots
```

```
ggplot(CombinedData, aes(x = Date, y = Close.Adj.ENB)) +
  geom_point(size = 0.5) + geom_line() +
  ggtitle("Close.Adj.ENB Price")

ggplot(CombinedData, aes(x = Date, y = Close.Adj.MSFT)) +
  geom_point(size = 0.5) + geom_line() +
  ggtitle("Close.Adj.MSFT Price")

ggplot(CombinedData, aes(x = Date, y = Close.SPX)) +
  geom_point(size = 0.5) + geom_line() +
  ggtitle("Close.SPX Price")

# Return Plots

ggplot(CombinedData, aes(x = Date, y = Return.Adj.ENB)) +
  geom_point(size = 0.5) + geom_line() +
  ggtitle("Return.Adj.ENB Price")

ggplot(CombinedData, aes(x = Date, y = Return.Adj.MSFT)) +
  geom_point(size = 0.5) + geom_line() +
  ggtitle("Return.Adj.MSFT Price")

ggplot(CombinedData, aes(x = Date, y = Return.SPX)) +
  geom_point(size = 0.5) + geom_line() +
  ggtitle("Return.SPX Price")


#================================================================================
# Plots for Section Four
#================================================================================

# Plots for ACF and PACF with and without xlim = 10

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
    return({
      jpeg(paste0("ACF.", y,".jpg"), width = 250, height = 250)
      plot(x, main = paste0("ACF ", y))
      dev.off()
    })

  }
  , SIMPLIFY = FALSE
  , x = A4.ACF.PACF.Ljung.1[[1]]
```

```
  , y = names(A4.ACF.PACF.Ljung.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
    return({
      jpeg(paste0("PACF.", y,".jpg"), width = 250, height = 250)
      plot(x, main = paste0("PACF ", y))
      dev.off()
    })

  }
  , SIMPLIFY = FALSE
  , x = A4.ACF.PACF.Ljung.1[[2]]
  , y = names( A4.ACF.PACF.Ljung.1[[2]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("ACF with ylim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.15, 0.15))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x =  A4.ACF.PACF.Ljung.1[[1]]
, y = names(A4.ACF.PACF.Ljung.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("PACF with xlim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("PACF ", y), xlim = c(0, 10))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.1[[2]]
, y = names(A4.ACF.PACF.Ljung.1[[2]])
)
```

```
# Plots for ACF and PACF with and without xlim = 10 for our Optimal Models under AIC

# AIC

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC ACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("ACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.1[[1]]
, y = names(A4.ACF.PACF.Ljung.AIC.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC PACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("PACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.1[[2]]
, y = names(A4.ACF.PACF.Ljung.AIC.1[[2]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC ACF with ylim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.15, 0.15))
    dev.off()
  })

}
, SIMPLIFY = FALSE
```

```
, x = A4.ACF.PACF.Ljung.AIC.1[[1]]
, y = names(A4.ACF.PACF.Ljung.AIC.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC PACF with xlim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("PACF ", y), xlim = c(0,10))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.1[[2]]
, y = names(A4.ACF.PACF.Ljung.AIC.1[[2]])
)

# BIC


mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("BIC ACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("ACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.1[[1]]
, y = names(A4.ACF.PACF.Ljung.BIC.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("BIC PACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("PACF ", y))
    dev.off()
  })

}
```

```
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.1[[2]]
, y = namers(A4.ACF.PACF.Ljung.BIC.1[[2]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("BIC ACF with ylim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.15, 0.15))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.1[[1]]
, y = names(A4.ACF.PACF.Ljung.BIC.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("BIC PACF with xlim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("PACF ", y), xlim = c(0, 10))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.1[[2]]
, y = names(A4.ACF.PACF.Ljung.BIC.1[[2]])
)

# Plots for AIC BIC for New Seasonal Model

# AIC


mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC ACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("ACF ", y))
    dev.off()
```

```
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.2[[1]]
, y = names(A4.ACF.PACF.Ljung.AIC.2[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC PACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("PACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.2[[2]]
, y = names(A4.ACF.PACF.Ljung.AIC.2[[2]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC ACF with ylim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.15, 0.15))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.2[[1]]
, y = names(A4.ACF.PACF.Ljung.AIC.2[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC PACF with xlim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("PACF ", y), xlim = c(0,10))
    dev.off()
  })
```

```
}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.AIC.2[[2]]
, y = names(A4.ACF.PACF.Ljung.AIC.2[[2]])
)


# BIC


mapply(function(x,y) {
 # Function to unlist, plot, and export graphs
 return({
  jpeg(paste0("BIC ACF.", y,".jpg"), width = 250, height = 250)
  plot(x, main = paste0("ACF ", y))
  dev.off()
 })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.2[[1]]
, y = names(A4.ACF.PACF.Ljung.BIC.2[[1]])
)

mapply(function(x,y) {
 # Function to unlist, plot, and export graphs
 return({
  jpeg(paste0("BIC PACF.", y,".jpg"), width = 250, height = 250)
  plot(x, main = paste0("PACF ", y))
  dev.off()
 })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.2[[2]]
, y = namers(A4.ACF.PACF.Ljung.BIC.2[[2]])
)

mapply(function(x,y) {
 # Function to unlist, plot, and export graphs
 return({
  jpeg(paste0("BIC ACF with ylim ", y,".jpg"), width = 250, height = 250)
  plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.15, 0.15))
  dev.off()
```

```
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.2[[1]]
, y = names(A4.ACF.PACF.Ljung.BIC.2[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("BIC PACF with xlim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("PACF ", y), xlim = c(0, 10))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A4.ACF.PACF.Ljung.BIC.2[[2]]
, y = names(A4.ACF.PACF.Ljung.BIC.2[[2]])
)

#================================================================================
# Plots for Section Five
#================================================================================

# Plot my Residuals Squared with Dates

mapply(function(x,y) PlotResidualsSquared(x,y), x = A5.Residuals.Squared.DF, y = VectorNames)


# ACF and PACF plots for Residual Squared data

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("ACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("ACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A5.ACF.PACF.Ljung.1[[1]]
```

```
, y = names(A5.ACF.PACF.Ljung.1[[1]])
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("PACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("PACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A5.ACF.PACF.Ljung.1[[2]]
, y = names(A5.ACF.PACF.Ljung.1[[2]])
)

# Plot GARCH Standardized Residuals

# AIC

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC ACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("ACF ", y))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A5.ACF.PACF.Ljung.AIC.GARCH[[1]]
, y = VectorNames[1:6]
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC PACF.", y,".jpg"), width = 250, height = 250)
    plot(x, main = paste0("PACF ", y))
    dev.off()
  })

}
```

```
, SIMPLIFY = FALSE
, x = A5.ACF.PACF.Ljung.AIC.GARCH[[2]]
, y = VectorNames[1:6]
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC ACF with ylim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.10, 0.10))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A5.ACF.PACF.Ljung.AIC.GARCH[[1]]
, y = VectorNames
)

mapply(function(x,y) {
  # Function to unlist, plot, and export graphs
  return({
    jpeg(paste0("AIC PACF with xlim ", y,".jpg"), width = 250, height = 250)
    plot(x[2:91], main = paste0("PACF ", y), xlim = c(0,10))
    dev.off()
  })

}
, SIMPLIFY = FALSE
, x = A5.ACF.PACF.Ljung.AIC.GARCH[[2]]
, y = VectorNames
)


# BIC

# mapply(function(x,y) {
#   # Function to unlist, plot, and export graphs
#   return({
#     jpeg(paste0("BIC ACF.", y,".jpg"), width = 250, height = 250)
#     plot(x, main = paste0("ACF ", y))
#     dev.off()
#   })
#
```

```
# }
# , SIMPLIFY = FALSE
# , x = A5.ACF.PACF.Ljung.BIC.GARCH[[1]]
# , y = VectorNames
# )
#
# mapply(function(x,y) {
#   # Function to unlist, plot, and export graphs
#   return({
#     jpeg(paste0("BIC PACF.", y,".jpg"), width = 250, height = 250)
#     plot(x, main = paste0("PACF ", y))
#     dev.off()
#   })
#
# }
# , SIMPLIFY = FALSE
# , x = A5.ACF.PACF.Ljung.BIC.GARCH[[2]]
# , y = VectorNames
# )
#
# mapply(function(x,y) {
#   # Function to unlist, plot, and export graphs
#   return({
#     jpeg(paste0("BIC ACF with ylim ", y,".jpg"), width = 250, height = 250)
#     plot(x[2:91], main = paste0("ACF ", y), ylim = c(-0.15, 0.15))
#     dev.off()
#   })
#
# }
# , SIMPLIFY = FALSE
# , x = A5.ACF.PACF.Ljung.BIC.GARCH[[1]]
# , y = VectorNames
# )
#
# mapply(function(x,y) {
#   # Function to unlist, plot, and export graphs
#   return({
#     jpeg(paste0("BIC PACF with xlim ", y,".jpg"), width = 250, height = 250)
#     plot(x[2:91], main = paste0("PACF ", y), xlim = c(0, 10))
#     dev.off()
#   })
#
# }
# , SIMPLIFY = FALSE
```

```
# , x = A5.ACF.PACF.Ljung.BIC.GARCH[[2]]
# , y = VectorNames
# )


#================================================================================
# Excel Export
#================================================================================

# Section One Exports

FileExport = list(CombinedData
            , A1.DailyYearlyMeasures
            , A1.SharpeRatios
            )

names(FileExport) = c("CombinedData"
                , "DailyYearlyMeasures"
                , "SharpeRatios"
                )

Excel_Export(FileExport, FileName = "Section 1 Data.xlsx", Dir)


# Section Two Exports

#   1. Optimal Mean Variance Portfolio Data

FileExport = c(OMVPortfolioReturns,A2.OMVPortfolioReturnsMeasures, A2.OMVPortfolioWeights,
A2.OMVPortfolioWeightsMeasures)

names(FileExport) = c(
  paste0(rep("OMV ", 3), "Return ", seq(1,3,1)),
  paste0(rep("OMV ", 3), "ReturnMeasures ", seq(1,3,1)),
  paste0(rep("OMV ", 3), "Weights ", seq(1,3,1)),
  paste0(rep("OMV ", 3), "WeightsMeasures ", seq(1,3,1))
)

Excel_Export(FileExport, FileName = "OMV Data.xlsx")


#   2. Equal Portfolio Data

FileExport = c(A2.EqualPortfolios, A2.EqualPortfolioMeasures, A2.EqualPortfolioSharpeRatios)
```

```
names(FileExport) = c(
  paste0(rep("EP ", 3), "Return ", seq(1,3,1)),
  paste0(rep("EP ", 3), "ReturnMeasures ", seq(1,3,1)),
  paste0(rep("EP ", 3), "SharpeRatios ", seq(1,3,1))
)

Excel_Export(FileExport, FileName = "EP Data.xlsx")




# Section Three Exports

write.xlsx(A3.ADF_Tests_TValues1, file = "ADF Test Values1.xlsx")
write.xlsx(A3.ADF_Tests_TValues2, file = "ADF Test Values2.xlsx")
write.xlsx(A3.ADF_Tests_TValues3, file = "ADF Test Values3.xlsx")

write.csv(A3.ADF_Tests_GLS1, file = "ADF GLS Test Values1.csv")
write.csv(A3.ADF_Tests_GLS2, file = "ADF GLS Test Values2.csv")

# Section Four Exports

write.csv(data.frame(A4.Max.Lags), "vec_lag.csv") # table for max lags
write.csv(unlist(A4.Ljung.pvalues), "Ljung_pvalue.csv") # table for max lags
write.csv(A4.AIC.BIC.Test, "AIC_BIC_Test.csv")
write.csv(A4.ARIMA.AIC.Ljung.pvalues, "ARIMA_AIC_LJUNG.csv")
write.csv(A4.ARIMA.BIC.Ljung.pvalues, "ARIMA_BIC_LJUNG.csv")
write.csv(AIC.MIN.Model, "AIC.MIN.MODEL.csv")
write.csv(A4.Best.Ljung.pvalues, "Ljung.best.csv")

# Section Five Exports

write.csv(A5.ACF.PACF.Ljung.1[[3]], "a5 ljung test for sqr res.csv")
write.csv(A5.AIC.BIC.1, "a5 aic and bic.csv")
write.csv(A5.ACF.PACF.Ljung.AIC.GARCH[[3]], "a5 garch ljung.csv")
#================================================================================
# End of Script
#================================================================================
```