Book Appendix
Introduction to R Programming

Learning objectives

- Provide numerous illustrations of basic R coding within the context of quantitative finance
- Introduce several useful R functions

See module *Ch Appendix A Introduction to R Programming*.

A1.1 R Basics Introduction.R

In this file, we cover a few coding insights. It is helpful to place the file name near the top along with a few comments.

```
# BA1.1 R Basics Introduction.R
# Insight: Getting help within R is easy, but there is a lot to know.
# #, cat(), help(), print(), ls(), rm(), getOption(), matrix(), list(), = v. <-
# This is a comment line -- Place cursor on this line and click <Run> above
# Comment lines show up in the Console window but not in the Environment window
# q() # Quit R, q(no) -- quit without saving
```

In code related to this book, we provide some cleanup code to avoid certain problems from repeatedly running the code. The next three lines clear the Environment window, the Console, and Plots tab respectively. It is recommended to run with every program you write.

```
rm(list = ls())  # Take out the Environment "trash"
cat("\014") # Clear Console, making error checking easier.
while (!is.null(dev.list()))  dev.off() # Clear old plots
```

The remainder of this file introduces you to several useful R features including cat(), help(), print(), assignment (<-), ls(), rm(), getOptions(), digits, and matrix() along with referencing matrix elements. The file concludes with an illustration of when the two assignment methods are not equivalent illustrated below.

```
# When = does not behave as <-
# Create an integer vector with values 1 to 10
x = 1:10
median(x) # Compute the median
rm(x)
median(x = 1:10)  # No vector x created and median calculated
x                 # x DOES NOT exist
median(y <- 1:10) # Vector y created and only then median calculated
y                 # y DOES exist
```

A1.2 R Basics Variables.R

Several introductory concepts are illustrated in this file. A few selected ones are repeated here.

```
StrikePrice = 100 # Vector of length 1, data type is double
StrikePrice # Console: [1] 100
# Display STRucture
str(StrikePrice) # Console: num 100, data type is double, value is 100
# Assignment of StockPrice to Price, = and <- perform same assignment
ExercisePrice <- StrikePrice[1]
# StockPrices will be a numerical vector of size 5 (following 2 lines)
StockPrices =       # Note math symbol MUST be on top line (not true in C++)
  c(90, 95, 100, 105, 110)    # New line but same equation
# Alternative for line continuation is leaving an open parenthesis
StockPrices = (
  c(9.0, 9.5, 10.0, 10.5, 11.0))
# A compact summary of the particular data structure displayed in the console
str(StockPrices) # Summary
typeof(StockPrices) # What data type
class(StockPrices) # What data class
# Historically, single floating point variables were stored with accuracy
#  around 7 digits whereas double floating point variables were about twice
#  the precision or 14 digits. Note every machine is different.
#  Details found when run .Machine.
.Machine # More details than you probably want to know
```

```
# Is it a double (R does not have single floating point)
is.double(StockPrices)
is.integer(StockPrices) # Not an integer
StockPrices = 100L # Force data type to be integer
typeof(StockPrices) # Note converted to integer
is.double(StockPrices)
is.integer(StockPrices)
```

Several other data types as well as simple plots are also reviewed. Finally, we introduce the concept of lists and ways to manage them.

```
# Example of a list rather than a vector
OptionName = 'Index Option'
isCall = TRUE
UnderlyingPrice = 105
StrikePrice = 100
InterestRate = 0.05
Volatility = 0.30
TimeToMaturity = 1
UnderlyingYield = 0.05
# Could use line continuation (+), but unnecessary
BSMOVM <- list(OptionName, isCall, UnderlyingPrice, StrikePrice, InterestRate,
+  Volatility, TimeToMaturity, UnderlyingYield)
BSMOVM <- list(OptionName, isCall, UnderlyingPrice, StrikePrice, InterestRate,
  Volatility, TimeToMaturity, UnderlyingYield)
is.list(BSMOVM)
is.list(OptionName) # Is not a list
attr(BSMOVM,"Option Name") <- "S&P 500 Index Option"
str(BSMOVM) # Summary
str(attributes(BSMOVM))
# Names
names(BSMOVM) <- c("Option Name", "Is Call?", "Underlying Price",
  "Strike Price", "Interest Rate", "Volatility", "Time To Maturity",
  "Underlying Yield")
str(BSMOVM) # Summary
str(attributes(BSMOVM)) # Now has two levels of attributes
BSMOVM
# Vector of zeros
Dividends = numeric(4) # Vector of zeros, dividends at each node
```

A1.3 R Basics Vocabulary.R

Several introductory functions are first reviewed. Quantitative finance problems often require taking first differences of various data series, such as stock prices.

```
?"diff"
a <- seq(1, 20, 2) # Sequence, increment by 2
b = diff(a, lag = 2) # Difference, two elements apart
c = diff(a, differences = 2) # Difference, one element apart, twice (difference of
difference)
d = diff(a, lag = 2, differences = 2)
lagt = 2 # Test lag
NV = 10 # Length of vector
e = a[(1+lagt):NV] - a[1:(NV-lagt)] # Difference lagged by lagt
f = diff(a, lag = 2)  # Difference again, lagged by two. NOTE: f = e
```

After reviewing several statistical functions, we examine more carefully vectors and matrices.

```
# Vectors and matrices
a = c(1:5) # Numerical vector of size 5, initiallized with 1, 2, 3, 4, 5
b = matrix(data = 0.0, nrow = 5, ncol = 4) # Matrix 5x5
for(i in 1:5){
  a[i] = i
  for(j in 1:4){
    b[i,j] = i + 2*j # i references row, j references column
  }
}
a; b
rm(list=ls())
```

```
# Lists and data.frames
StockPrice = 100.0; StrikePrice = 100.0; InterestRate = 5.0
Volatility = 30; Maturity = 1; Ticker = "AAPL"
# list(): Creates a generic vector
InputData <- list(StockPrice, StrikePrice, InterestRate, Volatility, Maturity, Ticker)
sapply(InputData, class) # Apply class function to every element of InputData
class(InputData) # Apply class to InputData matrix
InputData
# names(): Functions to assign names of R objects
names(InputData) <- c("StockPrice", "StrikePrice", "InterestRate", "Volatility",
  "Maturity", "Ticker")
sapply(InputData, class)
InputData
# Alternatively
InputData <- list(StockPrice, StrikePrice, InterestRate, Volatility, Maturity, Ticker)
InputData = array(data = InputData, dim = c(1, 6), dimnames = list(NULL,
   c("StockPrice", "StrikePrice", "InterestRate", "Volatility",
     "Maturity", "Ticker")))
sapply(InputData, class)
```

One important way to manage complex data is with the data.frame function.

```
# data.frame is a preferred way to collect variables, a fundamental data structure
InputData = data.frame(matrix(vector(), 1, 6, dimnames=list(c(),
  c("StockPrice", "StrikePrice", "InterestRate", "Volatility", "Maturity",
    "Ticker"))), stringsAsFactors=F)
InputData$StockPrice <- StockPrice
InputData$StrikePrice <- StrikePrice
InputData$InterestRate <- InterestRate
InputData$Volatility <- Volatility
InputData$Maturity <- Maturity
InputData$Ticker <- Ticker
sapply(InputData, class)
InputData
```

A1.4 R Basics Functions.R

The language R has numerous built-in functions as well as providing the ability to build your own. Many additional functions are available in various packages that can be added. After reviewing the basics, we illustrate installing the package lubridate and related implications.

```
# Some libraries modify existing functions, such as lubridate
# First, we need to be sure the package has been installed in the past.
# The next line only needs to ever be run once, but you are never sure where your R
program will end up.
if("lubridate" %in% rownames(installed.packages()) == FALSE)install.packages("lubridate")
library(lubridate)
# Note that "date" is masked, thus there is more than one date function.
# R will use the first available, depending on when packages were added.
# The order of packages can be found with the search function
search()
# At times you will want to remove some packages or detach them.
detach("package:lubridate", unload=TRUE) # Unattaching package
# Note that lubridate is removed.
search()
# if
if(is.R()){ # Functions with "is." are logical functions
  x <- 5
} else {
  x <- 10 # <ALT> <-> produces <-
}
# c() function which combines arguments
help("c")
NumberOfDraws = 1000
StockReturns = c(1:NumberOfDraws)
```

The stats package will be widely used in sample code.

```
# Function in stats (statistical package)
```

```
if("stats" %in% rownames(installed.packages()) == FALSE)install.packages("stats")
require(stats) # Likely not required if already installed
# help("stats")
# library(help = "stats") # Provides listing of functions
# rnorm function creates random numbers, normal(mean, sd)
help("rnorm")
ExpectedStockReturn = 0.15 # Expected stock return
SDStockReturn = 0.3 # Standard deviation of stock returns
# Vector of normally distributed random variables
StockReturns = rnorm(StockReturns, mean = ExpectedStockReturn, sd = SDStockReturn)
AverageStockReturns = mean(StockReturns)
Error = AverageStockReturns - ExpectedStockReturn
help("lognormal") # Functions are case sensitive, this does not workk
help("Lognormal")
StockPrice = 100
MaturityTime = 1
# Excessive use of parenthesis makes reading equations easier for all involved
Drift = log(StockPrice) + (ExpectedStockReturn - ((SDStockReturn^2)/2.0))*MaturityTime
Noise = SDStockReturn * sqrt(MaturityTime)
Epsilon = c(1:NumberOfDraws)
Epsilon = rnorm(Epsilon)
# Lognormally distributed vector of terminal stock prices
StockPrices = rlnorm(Epsilon, mean = Drift, sd = Noise)
AverageStockPrice = mean(StockPrices)
ExpectedStockPrice = StockPrice * exp(ExpectedStockReturn*MaturityTime)
# Monte Carlo simulations always have sample error
SampleError = AverageStockPrice - ExpectedStockPrice
```

Illustration of a histogram.

```
# Quick histogram to check simulation
hist(StockPrices, main="Lognormal Distribution", xlab="Prices", ylab="Frequency",
border="blue",
  col="green", xlim=c(0, 500), las=1, breaks=20)
#
# Nasty feature of R
# WRAPPED LINES MUST END WITH APPROPRIATE SIGN, NOT AT BEGINNING (z2 is wrong below)
#
rm(list = ls())
cat("\014") # Clear Console, making error checking easier.
dev.off() # Clear any old Plots
X = 100
Y = 200
z1 = X + Y
z2 = X         # Due to the absence of semicolons in R (vs. C++)
  + Y          # WRONG!!!
z3 = X +       # RIGHT: Always terminate line with required symbol
  Y
```

Now it is time to start building your own functions.

```
# Function that computes N + 1 (over simple to illustrate functions)
AddOne = function(N) {
  N + 1  # Alternatively: return(N+1)
}
attr(AddOne, "comment") <- "AddOne simply adds one to a number"
attr(AddOne, "help") <- "AddOne expects one number and it will add 1 to that number"
attributes(AddOne)
attr(AddOne, "comment")
attr(AddOne, "srcref") # Print R source code: SouRCe REFerences
N = 10
M = AddOne(N) # Call function AddOne and pass N. Then return results to M.
AddOne # What is this function? See the Console below.
```

Finally, when running long programs, it helps to have some sort of beep to let you know when it is finished.

```
install.packages("beepr")
if("beepr" %in% rownames(installed.packages()) == FALSE)install.packages("beepr")
library(beepr)
beep(sound = 2, print('Finished')) # Helpful when running long program
```

```
# Illustration of other available sounds
beep(sound = 1, print('Finished')) # Helpful when running long program
beep(sound = 3, print('Finished')) # Helpful when running long program
beep(sound = 4, print('Finished')) # Helpful when running long program
beep(sound = 5, print('Finished')) # Helpful when running long program
beep(sound = 6, print('Finished')) # Helpful when running long program
beep(sound = 7, print('Finished')) # Helpful when running long program
beep(sound = 8, print('Finished')) # Helpful when running long program
beep(sound = 9, print('Finished')) # Helpful when running long program
beep(sound = 10, print('Finished')) # Helpful when running long program
beep(sound = 11, print('Finished')) # Helpful when running long program
beep() # Default of one
```

A1.5 R Basics File Management.R

Managing data files is one strength of the R language. Several aspects of data management are illustrated here. First, we need to be sure we have the necessary packages in our library.

```
if("openxlsx" %in% rownames(installed.packages()) == FALSE)install.packages("openxlsx")
if("date" %in% rownames(installed.packages()) == FALSE)install.packages("date")
library(openxlsx)
library(date)
```

Input and output of comma separated files.

```
#
# Example 1: File provided. read.csv is part of the utils package
#  within the base package
#
OptionExample <- read.csv("A1.5a Options.csv")
tail(OptionExample, 5) # Last five observations
OptionExample$X <- NULL # Remove column X
tail(OptionExample, 5) # Note column X is removed
# Removes observations prior to 8/30/2015 (ends 8/31/2015)
OptionExample<-OptionExample[!(OptionExample$date < 20150830),]
# Write data to csv file
write.table(OptionExample, file = 'A1.5a Temp File.csv', quote = FALSE,
  row.names = FALSE, sep = ",", col.names = TRUE)
OptionTable <- read.csv("A1.5a Temp File.csv")
tail(OptionTable, 5)
# Save as a text file
write.table(OptionExample, file = 'A1.5a Temp File 2.txt')
OptionText <- read.table("A1.5a Temp File 2.txt")
tail(OptionText, 5)
?"write.table"
rm(list = ls())
file.remove('A1.5a Temp File.csv', 'A1.5a Temp File 2.txt') # Delete files
```

Input data from the internet (often unstable and constantly changing).

```
#
# Example 2: Online files using quantmod
#
# quantmod - pulling stock data from internet (may or may not work)
#   as internet protocols constantly change
if("quantmod" %in% rownames(installed.packages()) == FALSE)
  install.packages("quantmod")
library(quantmod)
Security1 <- "AAPL" # Ticker
# Input S1Data from Yahoo (or other available vendor) with
#  getSymbols (quantmod function)
# load historical data into an xts object
S1Data <- getSymbols(Security1, auto.assign = FALSE)
# xts denotes eXtensible Time-Series object useful for managing time series
# data (details later in 2.9 Zoo and Natural Gas Futures.R)
S1Data[1:5,] # Alternative way to see first five observations
# Helpful to make xts data generic for further processing (need to eliminate
# ticker symbol). Now referenced as Security 1 (S1)
# Convert xts to numeric, strips index and returns only observation
S1GenericData <- coredata(S1Data)
```

```
tail(S1GenericData, 5) # Note loss of date
# Must use S1GenericData so AdjClose is not overwritten
S1AdjPrices <- data.frame("Date" = index(S1Data),
  "AdjClose" = S1GenericData[,6])
head(S1AdjPrices, 5)
class(S1AdjPrices$Date)
# Eliminate data before a certain date
StartDate <- as.Date('2010-01-01')
class(StartDate)
S1AdjPrices <- S1AdjPrices[S1AdjPrices$Date >= StartDate,]
write.table(S1AdjPrices, file = 'A1.5a Temp Stock Data.txt', quote = FALSE,
  row.names = FALSE, sep = ",", col.names = TRUE)
S1AdjPrices1 <- read.table('A1.5a Temp Stock Data.txt', sep = ',',
  header = TRUE)
getwd() # Get working directory just to be sure where to locate files
?"save"
# Save as a binary R data set
save(S1AdjPrices, file = 'A1.5b Stock 1 Adjusted Prices.Rdata')
remove(S1AdjPrices) # Note S1AdjPrices is removed from the Environment
rm(list = ls())
# Upload a binary R data set
# Note S1AdjPrices is in the Environment
load('A1.5b Stock 1 Adjusted Prices.Rdata')
head(S1AdjPrices, 5)
rm(list = ls())
file.remove('A1.5b Stock 1 Adjusted Prices.Rdata') # Alternative way to delete
```

Input and output of text files.

```
#
# Example 3: Files on your computer (txt)
#
# If file is in the project path (relative path)
Rel_S1 <- read.table('A1.5a Temp Stock Data.txt', sep = ',', header = TRUE)
Rel_S1[1:5,]
# If file is not in the project path (absolute path)
Abs_S1 <- read.table('/QFRepository/Ch Appendix/A1.5a Temp Stock Data.txt',
  sep = ',', header = TRUE)
Abs_S1[1:5,]
TestData <- read.table('A1.5a Temp Stock Data.txt', sep = ",", header = TRUE,
  skip = 10, nrows = 20, col.names = c("Date", "AdjClose"))
TestData # 20 rows of data, starting at observation 11
# If need to access a particular path several times
Path <- '/QFRepository/Ch Appendix/'
File <- 'A1.5a Temp Stock Data.txt'
FileLocation <- paste(Path, File, sep = "")
Abs_S1 <- read.table(FileLocation, sep = ',', header = TRUE)
Abs_S1[1:5,]
rm(list = ls())
# Relative path using delim for delimiter
Del_S1 <- read.delim('A1.5a Temp Stock Data.txt', sep = ',', header = TRUE)
Del_S1[1:5,]
# Relative path using table, note header = TRUE is the default
# Note using table introduces heading V1-V7 as well as numerical precision
# What is displayed is what is actually stored
Tab_S1 <- read.table('A1.5a Temp Stock Data.txt', sep = ',')
Tab_S1[1:5,]
# Replace 2.5 Temp Stock Data.txt with space delimited file
write.table(Del_S1, file = 'A1.5b Temp Stock Data.txt', quote = FALSE,
  row.names = FALSE, sep = " ", col.names = TRUE)
Space_S1 <- read.delim('A1.5b Temp Stock Data.txt', sep = ' ', header = TRUE)
head(Space_S1, 5)
rm(list = ls())
# Read in lines using readLines, just messing around
# Each line is a character vector
RL_S1 <- readLines('A1.5b Temp Stock Data.txt')
```

```
head(RL_S1,5)
length(RL_S1)
# Transpose and split character vector
RL1_S1 <- t(sapply(strsplit(RL_S1[1], " "),unlist)) # Read first line
head(RL1_S1, 5)
length(RL1_S1)
RL2_S1 <- t(sapply(strsplit(RL_S1[2], " "),unlist)) # Read second line
head(RL2_S1, 5)
length(RL2_S1)
# Clean up files
file.remove('A1.5b Temp Stock Data.txt')
rm(list = ls())  # Clean up to illustrate xlsx
#
```

Input and output of spreadsheet files.

```
# Excel file management examples
#
# Example 1: Read in an existing data file, 2.7 AAPL_Yahoo.xlsx
#
AAPL_From_Excel <- read.xlsx(xlsxFile = "A1.5a AAPL_Yahoo.xlsx", sheet = 1,
  skipEmptyRows = FALSE)
is.data.frame(AAPL_From_Excel) # Inputted as data frame
# Note that date is in "Julian" format (large integer number)
head(AAPL_From_Excel, 5)
tail(AAPL_From_Excel, 5)
# Note: attach() is a useful function that makes the data frame local in scope
# Do not have to use AAPL_From_Excel$ on right hand side
attach(AAPL_From_Excel)
AAPL_From_Excel$Date2 = date.mmddyyyy(Date) # Date adjustment
detach(AAPL_From_Excel) # Revert back
# Excel Julian dates start from 12/30/1899
AAPL_From_Excel$Date2 = date.mmddyyyy(AAPL_From_Excel$Date) # Date adjustment
# But R defaults to 1/1/1960, thus the dates are off by 21,916 days
head(AAPL_From_Excel, 5)
tail(AAPL_From_Excel, 5)
# Date adjustment
AAPL_From_Excel$Date3 = date.mmddyyyy(AAPL_From_Excel$Date - 21916)
head(AAPL_From_Excel,5)
tail(AAPL_From_Excel,5)
x1 = as.numeric(AAPL_From_Excel$Date)
x2 = as.date(AAPL_From_Excel$Date2) # Now Julian
AAPL_From_Excel$Open <- NULL
AAPL_From_Excel$High <- NULL
AAPL_From_Excel$Low <- NULL
AAPL_From_Excel$Close <- NULL
AAPL_From_Excel$Volume <- NULL
head(AAPL_From_Excel,5)
tail(AAPL_From_Excel,5)
AAPL_From_Excel$Date2 = NULL
AAPL_From_Excel$Date = AAPL_From_Excel$Date3
AAPL_From_Excel$Date3 = NULL
head(AAPL_From_Excel,5)
tail(AAPL_From_Excel,5)
#
# Example 2: Write out an existing data, AAPL_From_Excel
#
write.xlsx(AAPL_From_Excel, "A1.5b AAPL_AdjPrices.xlsx")
AAPLAdjPrices <- read.xlsx(xlsxFile = "A1.5b AAPL_AdjPrices.xlsx", sheet = 1,
  skipEmptyRows = FALSE)
tail(AAPLAdjPrices,5)
FLength1 = length(AAPLAdjPrices)
FLength2 = ncol(AAPLAdjPrices)
FSize = nrow(AAPLAdjPrices)
TDate = as.date('06/01/2015') # Note in Julian format
Date1 = as.date(AAPLAdjPrices$Date)
```

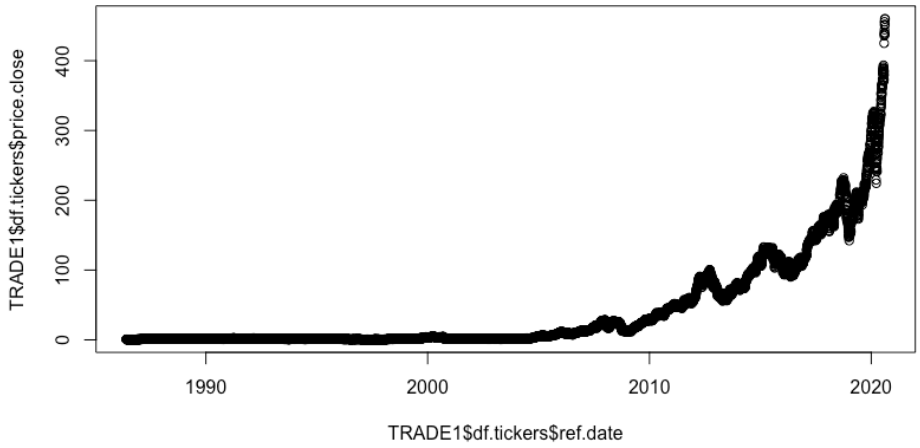
```
j = 0
# Count number of days after TDate
for (i in 1:FSize){
  if(Date1[i] > TDate){
    j = j + 1
  }
}
TotalDates = j
# ARRAY PROBLEMS: All variables would be character, thus dfReturns[,5]
# is error. Avoid using array.
dfReturns = array(data = -99, dim = c(TotalDates, 7), dimnames = list(NULL,
  c("Date", "Month", "Day", "Year", "Price", "Return", "Ticker")))
class(dfReturns) # data type for dfReturns
sapply(dfReturns, class) # data type for each variable within dfReturns
Ticker = 'AAPL'
for (i in 1:TotalDates){
  if(Date1[i] > TDate){
    dfReturns[i,5] = as.numeric(AAPLAdjPrices$Adj.Close[i])
    if(i > 1)dfReturns[i,6] = (dfReturns[i,5]/dfReturns[i-1,5]) - 1
    # If next line is uncommented, the for loop throws an error
    #    dfReturns[i,7] = Ticker
# Assigning character converts all variables to character!
  }
}
# Note that all variables are character; hence, division fails
#  (when Ticker assigned)
head(dfReturns, 3)
dfReturns[,7] <- Ticker
head(dfReturns, 3)
# data.frame is a preferred way to collect variables,
#  a fundamental data structure
dfReturns = data.frame(matrix(vector(), TotalDates, 7, dimnames=list(c(),
  c("Date", "Month", "Day", "Year", "Price", "Return", "Ticker"))),
  stringsAsFactors=F)
sapply(dfReturns, class)
Ticker = 'AAPL'
Price = c(1:TotalDates)
NDate = c(1:TotalDates)
for (i in 1:TotalDates){
  if(Date1[i] > TDate){
    temp = date.mdy(Date1[i])
    dfReturns[i,1] = Date1[i] + 21916
    dfReturns[i,2] = temp$month
    dfReturns[i,3] = temp$day
    dfReturns[i,4] = temp$year
    # dfReturns[i,5] = as.numeric(AAPLAdjPrices$Adj.Close[i])
    dfReturns[i,5] = AAPLAdjPrices$Adj.Close[i]
    if(i > 1)dfReturns[i,6] = (dfReturns[i,5]/dfReturns[i-1,5]) - 1
    dfReturns[i,7] = Ticker
  }
}
sapply(dfReturns, class) # Variable types preserved
head(dfReturns,5)
write.xlsx(dfReturns, "A1.5b AAPL_Returns.xlsx")
# Clean up
file.remove('A1.5b AAPL_AdjPrices.xlsx')
file.remove('A1.5b AAPL_Returns.xlsx')
```

Another method of inputting data from the internet.

```
#
# BatchGetSymbols method of online data extraction
#
if("BatchGetSymbols" %in% rownames(installed.packages()) == FALSE)
  install.packages("BatchGetSymbols")
library(BatchGetSymbols)
```

```
library(zoo)
library(date)
library(openxlsx)
# Single stock
TRADE1 <- BatchGetSymbols('AAPL', # Symbol desired
  # src = "google", # Source: Yahoo or Google
# '1950-01-01', # Sys.Date()-365, start date in Date format
  first.date = '1986-06-01',
  last.date = Sys.Date()) # End date in Date format
head(TRADE1, 5)
head(TRADE1$df.tickers$price.close, 5)
head(TRADE1$df.control$ticker, 5)
head(TRADE1$df.control, 5)
head(TRADE1$df.tickers, 5)
plot(TRADE1$df.tickers$ref.date, TRADE1$df.tickers$price.close)
```

Figure A1.1 Closing prices for Apple (AAPL) based on plot()




```
#
# Portfolio
#
Tickers = c('SPY', 'XLB', 'XLE', 'XLF', 'XLI', 'XLK', 'XLP', 'XLRE',
  'XLU', 'XLV', 'XLY')
PORTFOLIO1 <- BatchGetSymbols(tickers = Tickers, # Symbol desired
  bench.ticker = "^GSPC", # Benchmark (SP 500 for dates ?)
  #  src = "yahoo", # Source: Yahoo or Google
  first.date = as.Date('2010-01-01'), # Sys.Date()-365, start date in Date format
  last.date = Sys.Date(),
  thresh.bad.data = 0.9) # Percent of bad data allowed, if less, out
is.data.frame(PORTFOLIO1)
is.numeric(PORTFOLIO1)
print(PORTFOLIO1$df.control)
print(PORTFOLIO1$df.tickers)
head(PORTFOLIO1, 5)
head(PORTFOLIO1$df.tickers$price.close, 5)
head(PORTFOLIO1$df.control$ticker, 5)
Value <- PORTFOLIO1$df.tickers$price.adjusted[1]
head(Value, 5)
class(PORTFOLIO1)
str(PORTFOLIO1)
x1 <- PORTFOLIO1[[1]]
x2 <- as.data.frame(PORTFOLIO1[[2]])
x3 <- x2[x2$ticker == 'SPY',]
```

A1.6 R Basics Plot Example.R

Plotting is a strength of the R language. Prior to illustrating several techniques, we illustrate a useful way to install numerous libraries at the same time.

```
PackagesToLibrary <- c("tidyverse", "ggplot2", "scales", "stats", "grDevices",
  "quantmod","png","grDevices") # Libraries
if (length(setdiff(PackagesToLibrary, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(PackagesToLibrary, rownames(installed.packages())))
} # Make sure libraries are installed on this computer
# Load and attach libraries
lapply(PackagesToLibrary, library, character.only = TRUE)
rm(PackagesToLibrary)
```

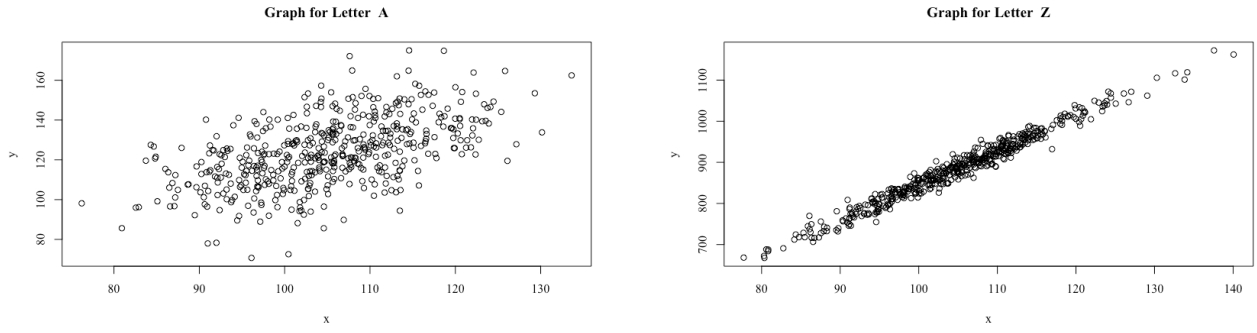
Managing the font of output from the R language makes incorporating the results into other documents easier.

```
# Globally set fonts for graphs: Great for consistency in documents
par(family = 'Times New Roman')
```

Several different plotting techniques as well as plot management are illustrated in the remainder of this file. First, we generate 26 plots with alphabetic titles, random slopes, but the same intercept.

```
names = LETTERS[1:26] # Gives a sequence of the letters of the alphabet
beta1 = rnorm(26, 5, 2) # N(5,2), a vector of slopes (one for each letter)
beta0 = 10 ## A common intercept
# 26 plots are produced, output to screen and files (see current directory)
for(i in 1:26){ # Standard for loop, runs from 1 to 26 by 1
  x = rnorm(500, 105, 10) # N(105, 10)
  y = beta0 + beta1[i]*x + 15*rnorm(500) # Fitted OLS with noise
  mytitle = paste("Graph for Letter ", names[i])
  plot(x, y, main = mytitle) # Plot to screen
# Save plots to Plot_A.jpg
  PlotName <- file.path(paste("Plot_", names[i], ".jpg", sep = ""))
  jpeg(file=PlotName) # Produce plots in files
  mytitle = paste("File Plot ", names[i]) # Plot to file
  plot(x, y, main = mytitle) # Plot to file
  dev.off() # Shuts down producing plots in files
}
```

Figure A1.2. Randomly generated plots with plot()



…

Clearing old plots is straightforward.

```
# unlink('*.jpg') # Rather dangerous, deletes all files with extension jpg
# Safer way to delete files (one at a time, cannot accidentally delete files)
for(i in 1:26){
  PlotName <- file.path(paste("Plot_", names[i], ".jpg", sep = ""))
  file.remove(PlotName) # Alternative way to delete
}
```

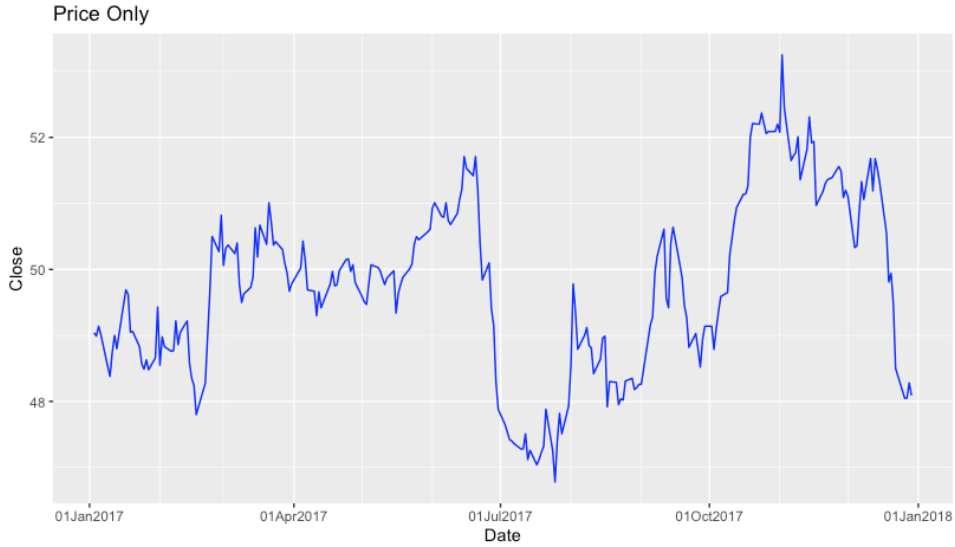
…

Based on stock prices, we can generate fancier plots with ggplot.

```
#
# Fancier Plots
#
# Globally set fonts for graphs: Great for consistency in documents
```

```
par(family = 'Times New Roman')
# Insight: Generic calculations and plots can be separated from main code
# Inputs
StartDate <- as.Date("01/01/2017", format="%m/%d/%Y")
EndDate <- as.Date("01/01/2018", format="%m/%d/%Y")
Security1 = "SO" # Southern Company ticker
# Input S1Data from Yahoo (or other available vendor) with
#  getSymbols (quantmod function)
# load historical data into an xts object
TCKR <- getSymbols(Security1, auto.assign = FALSE)
# TCKRDate <- TCKR
tail(TCKR, 5) #
TCKRGenericData <- coredata(TCKR)
tail(TCKRGenericData, 5) # Note loss of date
# Must use S1GenericData so AdjClose is not overwritten
TCKRAdjPrices <- data.frame("Date" = index(TCKR),
  "Close" = TCKRGenericData[,4], "AdjClose" = TCKRGenericData[,6])
head(TCKRAdjPrices, 5)
class(TCKRAdjPrices$Date)
# # Convert to date format
# JDate is number of days since 1970-01-01
TCKRAdjPrices$JDate <- as.integer(TCKRAdjPrices$Date)
# dplyr method to manipulate dates
TCKRAdjPrices = TCKRAdjPrices %>% filter(Date<=EndDate, Date>=StartDate) %>%
  mutate(Day = as.integer(format(Date, "%d")),
  Month = as.integer(format(Date, "%m")),
  Year = as.integer(format(Date, "%Y") )) %>% arrange(Date)
# Plot
x = TCKRAdjPrices$Date
y = TCKRAdjPrices$Close
Title1 = "Price Only"
yTitle = "Price"
xTitle = "Date"
plot(x, y, xaxt="n", type="l", col="blue", pch = 1, cex = 0.5, main=Title1,
     xlab=xTitle, ylab=yTitle, las=1)
xat = seq(min(x), max(x), by="month")
xat.format = format(xat, "%d%b%Y")
axis(1,               # side=1 is x-axis
     at=xat,             # dates for ticks
     labels=xat.format,  # formatted date for printing
     las=1,              # las=2 for vertical, las=1 for horizontal
     cex.axis=0.8)        # text size
# Plot based on ggplot2
ggplot(TCKRAdjPrices) + geom_line(aes(Date, Close), color="blue", size=0.5) +
  ggtitle("Price Only") + scale_x_date(labels=date_format("%d%b%Y"),
                                       date_minor_breaks="1 month")
```

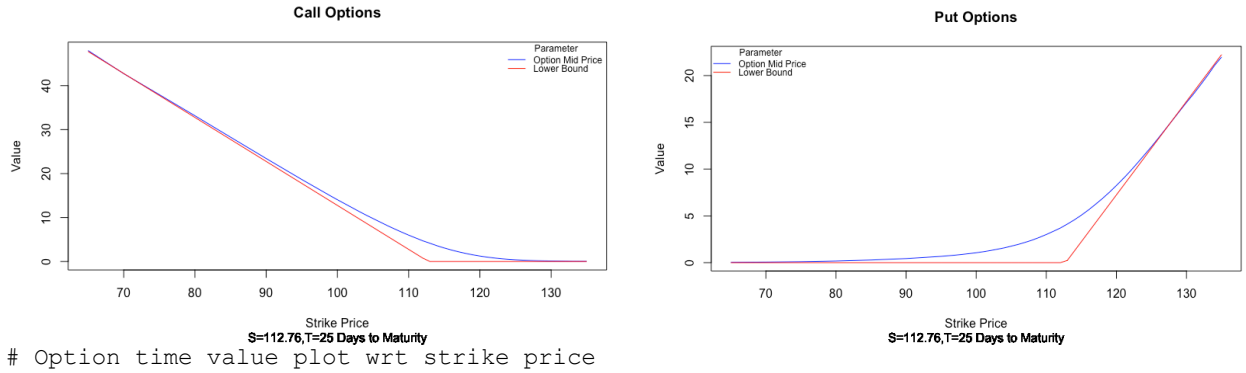
Figure A1.3. Price plot with ggplot()



After a bit of data processing, we illustrate a few more plots.

```
…
yTitle = "Value"
xTitle = "Strike Price"
lTitle = "Parameter"
plot(dfTCKR_Options$strike_price, dfTCKR_Options$MidPrice, type = "l",
  main = mTitle, sub = sTitle, xlab = xTitle, ylab = yTitle, col = "blue",
  xlim = xlim1, ylim = ylim1, pch = 1, cex = 0.5)
lines(dfTCKR_Options$strike_price, dfTCKR_Options$IntrinsicValue, type = "l",
  col ="red", xlim = xlim1, ylim = ylim1, pch = 2, cex = 0.5)
if(OptionType == 1){
  legend("topright", legtxt, cex = 0.75, lwd = c(1, 1), lty = c(1, 1),
         col = c("blue","red"), pch = c(NA,NA), bty = "n", title = lTitle)
} else {
  legend("topleft", legtxt, cex = 0.75, lwd = c(1, 1), lty = c(1, 1),
         col = c("blue","red"), pch = c(NA,NA), bty = "n", title = lTitle)
}
```

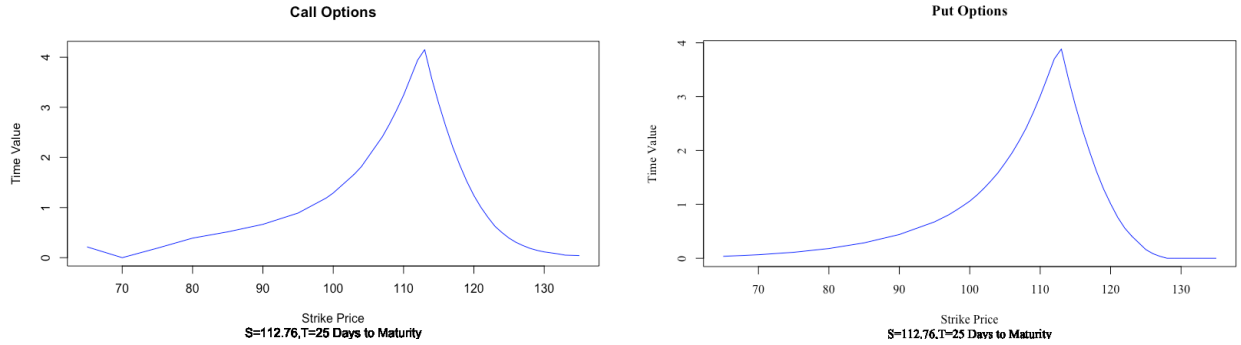
Figure A1.4 Option price with respect to strike price using plot()




```
# Option time value plot wrt strike price
dfTCKR_Options <- dfTCKR_Options[order(dfTCKR_Options$strike_price),]
MaxValue = max(dfTCKR_Options$TimeValue)
MinValue = min(dfTCKR_Options$TimeValue)
ylim1 = c(1:2); ylim1[1] = MinValue; ylim1[2] = MaxValue
MaxValue = max(dfTCKR_Options$strike_price)
MinValue = min(dfTCKR_Options$strike_price)
xlim1 = c(1:2); xlim1[1] = MinValue; xlim1[2] = MaxValue
if(OptionType == 1){
  mTitle = "Call Options"
```

```
} else {
  mTitle = "Put Options"
}
yTitle = "Time Value"
xTitle = "Strike Price"
plot(dfTCKR_Options$strike_price, dfTCKR_Options$TimeValue, type = "l",
  main = mTitle, sub = sTitle, xlab = xTitle, ylab = yTitle, col = "blue",
  xlim = xlim1, ylim = ylim1, pch = 1, cex = 0.5)
```

Figure A1.5 Option time value with respect to strike price using plot()



A1.6 R Generic Color Plot Example.R

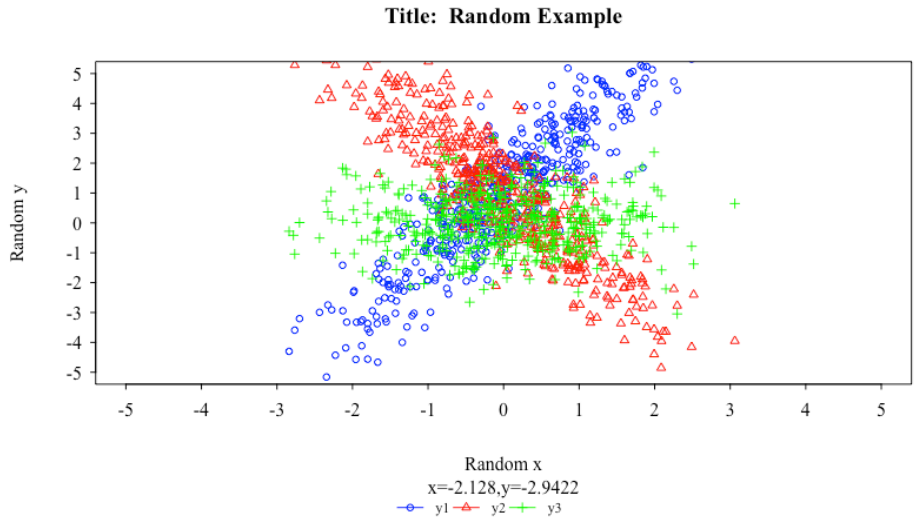
If you wish to generate consistent plots throughout a presentation, it is helpful to place all of your particular plotting preferences into a single file.

```
#
# Generic Plot: Inputs required
#
par(defaultpar) # Reset to default parameters
LineType <- "p" # "p" points; "l" lines; "b" both
LineSize <- 0.75
# Set x variable here:
isxDate <- FALSE
xT <- rnorm(500, 0, 1) # N(0, 1)
FixAxisX <- TRUE
MinValueX <- -5.0
MaxValueX <- 5.0
if(isxDate){
  xIT <- as.integer(x) # Dates are tricky: Convert to integer only (Julian)
} else {
  xIT <- xT
}
# Number of y variables and definitions
Ny <- 3
y1T <- 1.0 + 2.0*xT + rnorm(500) # Simple example
y2T <- 1.0 - 2.0*xT + rnorm(500)
y3T <- 0.0 + 0.0*xT + rnorm(500)
FixAxisY <- TRUE
MinValueY <- -5.0
MaxValueY <- 5.0
# Titles and other information
Title1 <- "Title: "
legtxt <- c("y1","y2","y3")
mTitle <- paste0(Title1," Random Example")
xTitle <- "Random x"
yTitle <- "Random y"
lTitle <- "Parameter"
# Plot footers: Insert subtitles here
Tx <- paste0('x=', round(xT[1],4))
Ty1 <- paste0(',y=', round(y1T[1],4))
```

```
sTitle <- paste0(Tx, Ty1)
# Note Generic Plots.R can be reused as needed
source("Generic Plots.R")
```
Thus, Generic Plots.R contains the particulars related to the plots generated.
```
# Generic Plots.R
x <- xT
xI <- xIT
if(Ny == 1){
  y1 <- y1T
  MaxYValue <- max(y1, na.rm = TRUE)
  MinYValue <- min(y1, na.rm = TRUE)
}
if(Ny == 2){
  y1 <- y1T
  y2 <- y2T
  MaxYValue <- max(y1, y2, na.rm = TRUE)
  MinYValue <- min(y1, y2, na.rm = TRUE)
}
if(Ny > 2){
  y1 <- y1T
  y2 <- y2T
  y3 <- y3T
  MaxYValue <- max(y1, y2, y3, na.rm = TRUE)
  MinYValue <- min(y1, y2, y3, na.rm = TRUE)
}
if(FixAxisY){
  MaxYValue <- MaxValueY
  MinYValue <- MinValueY
}
ylim1 <- c(1:2); ylim1[1] <- MinYValue; ylim1[2] <- MaxYValue
MaxXValue <- max(xI); MinXValue <- min(xI)
if(FixAxisX){
  MaxXValue <- MaxValueX
  MinXValue <- MinValueX
}
xlim1 <- c(1:2); xlim1[1] <- MinXValue; xlim1[2] <- MaxXValue
# Illustrate more control over plots
#  Want legend outside of box to avoid overwriting data points
par(omi = c(0.2, 0, 0, 0)) # Set outer margins at bottom to 0.2 inches
plot(xI, y1, type = LineType, main = mTitle, axes = FALSE,
  sub = sTitle, xlab = xTitle, ylab = yTitle, col = "blue", xlim = xlim1,
  ylim = ylim1, pch = 1, cex = LineSize)
lines(xI, y2, type = LineType, col ="red", xlim = xlim1, ylim = ylim1,
  pch = 2, cex = LineSize)
lines(xI, y3, type = LineType, col ="green", xlim = xlim1, ylim = ylim1,
  pch = 3, cex = LineSize)
box() # create a wrap around the points plotted
# Format x-axis
IncrementX <- round((as.numeric(MaxXValue) - as.numeric(MinXValue))/10.0,1)
TickMarksX <- c(seq(from = as.numeric(MinXValue), to = as.numeric(MaxXValue),
  by=IncrementX))
if(isxDate){
  lblX <- as.Date(TickMarksX, origin = "1960-01-01")
  lblX <- format.Date(lblX, "%b-%Y")
} else {
  lblX <- paste0(format(TickMarksX, trim = TRUE, digits = 4,
    justify = c("right"), width = 0, big.mark = ","))
}
axis(side = 1, labels = NA, tck = -0.015, at = TickMarksX)
axis(side = 1, lwd = 0, line = -0.4, at = TickMarksX, label = lblX)
# Format y-axis
IncrementY <- round((as.numeric(MaxYValue) - as.numeric(MinYValue))/10.0, 1)
TickMarksY <- c(seq(from=MinYValue,to=MaxYValue,by=IncrementY))
lblY = paste0(format(TickMarksY, trim = TRUE, digits = 4,
```

```
  justify = c("right"), width = 0, big.mark = ","))
axis(side = 2, labels = NA, tck = -0.015, at = TickMarksY)
axis(side = 2, lwd = 0, line = -0.4, las = 1, at = TickMarksY, label = lblY)
# Overlay with invisible plot solely for legend
par(fig = c(0, 1, 0, 1), oma = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),new = TRUE)
plot(0, 0, type = "n", bty = "n", xaxt = "n", yaxt = "n",
     sub = sTitle)
legend("bottom", legtxt, horiz = TRUE, cex = 0.75, lwd = c(1,1,1),
  lty = c(1,1,1), col = c("blue","red","green"), bty = "n", pch = c(1, 2, 3),
  inset = c(0, 0))
par(defaultpar) # Reset to default parameters
```

Figure A1.6 Color plots using Generic Plots.R



A1.7 R Basics Calendar Challenges.R

One of the most tedious and frustrating elements of quantitative finance is managing the numerous aspects related to the calendar. The key to understanding dates is conversion to and from Julian dates, named after Julius Caesar who converted the lunar calendar to the solar calendar. Today, a Julian date is a sequential integer value assigned to very calendar date, based on the Gregorian calendar that is a modified version of the Julian calendar. Merely counting the number of days between two dates is relatively straightforward, but there remain significant issues. We introduce just a few here.

```
…
# From Gregorian to/from Julian via build-in date functions
TestMonth <- 1
TestDay <- 10
TestYear <- 2005
sapply(TestMonth, class) # TestMonth has the "numeric" attribute
# Julian date, Jan 1, 1960 baseline date
# Function mdy.date(), defined in date package converts M, D, Y to Julian
TestDate <- mdy.date(TestMonth, TestDay, TestYear) # 16,446
sapply(TestDate, class) # TestDate has the "character" attribute
# TestDate formatted as "Class 'date' int 16446
is.integer(TestDate) # TRUE
is.date(TestDate) # TRUE
BaseDate <- mdy.date(1, 1, 1960) # BaseDate = 0
BaseDate <- mdy.date(1, 1, 1970) # BaseDate = 3,653 (10 years since 1/1/60)
DaysSinceBaseDate <- TestDate - BaseDate # 12,793 (+ 3,653 = 16,446)
# as.Date, in this context, expecting only integer. Thus, changing origin
# has no effect (?)
TestDate2 <- as.Date(TestDate, origin = "1970-01-01") # 2005-01-10
sapply(TestDate2, class) # TestDate2 has the "Date" attribute
```

```
is.character(TestDate2) # FALSE
TestDate2Int <- as.integer(TestDate2) # 12,793
# TestDate2 is same as original input, even with new origin
# Converting to solely integer, changes the date (roughly ten years later)
TestDate3 <- as.Date(TestDate, origin = "1970-01-01") # 2005-01-10
# Converting to integer adds a day
TestDate31 <- as.integer(TestDate) # 16,446
TestDate311 <- as.Date(TestDate31, origin = "1970-01-01") # 2015-01-11 (?)
sapply(TestDate3, class) # TestDate3 has the "Date" attribute
# Alternative coercion of character data to Julian date
CharDate = "01-10-2005"
TestDate4 <- as.date(CharDate) # 16,446
sapply(TestDate4, class) # TestDate4 has the "character" attribute
# Here just for completeness
TestDate5 <- as.POSIXlt(as.Date(TestDate4))
sapply(TestDate5, class) # TestDate5 has many variables with different attributes
TestDate5$sec
TestDate5$min
TestDate5$hour
TestDate5$mday
TestDate5$mon + 1
TestDate5$year + 1900
TestDate5$wday
TestDate5$yday + 1
# But lose a day if
TestDate
TestDate6 <- as.POSIXlt(TestDate)
TestDate6$sec
TestDate6$min
TestDate6$hour # Why 18???
TestDate6$mday # Lost one day
TestDate6$mon + 1
TestDate6$year + 1900
TestDate6$wday
TestDate6$yday + 1
# Current time
TestDate7 <- as.POSIXlt(Sys.time(), "GMT")
TestDate7$sec # Seconds, 0-61 (leap second ???)
TestDate7$min # Minutes, 0-59
TestDate7$hour # Hours, 0-23
TestDate7$mday # Day of month, 1-31
TestDate7$mon # January = 0; months after the first of the year
TestDate7$year # Years since 1900
TestDate7$wday # Day of week, 0-6, starting on Sunday
TestDate7$yday # Day of year, 0-365
# A look at 2005-01-10
TestDate8 <- as.POSIXlt(as.Date(16446, origin = "1960-01-01"))
TestDate8$sec
TestDate8$min
TestDate8$hour
TestDate8$mday
TestDate8$mon + 1
TestDate8$year + 1900
TestDate8$wday
TestDate8$yday + 1
#
# Example 1 (as.date): MLK day (Monday, is holiday)
#
IMonth <- 1
IDay <- 21
IYear <- 2019
TestDateJulian1 <- mdy.date(IMonth, IDay, IYear, nineteen=TRUE)
CDate1 <- '1/21/2019'
TestDateJulian2 <- as.date(CDate1, order = "mdy")
```

```
CDate2 <- '1-21-2019'
TestDateJulian3 <- as.date(CDate2, order = "mdy")
CDate3 <- '21Jan2019'
TestDateJulian4 <- as.date(CDate3, order = "dmy") # Has no effect (dmy)
CDate4 <- 'January 21 2019'
TestDateJulian5 <- as.date(CDate4, order = "mdy")
CDate5 <- '21 1 2019'
TestDateJulian6 <- as.date(CDate5, order = "dmy") # Has effect
CDate6 <- '1212019'
TestDateJulian7 <- as.date(CDate6, order = "mdy") # Do not lead month w 0
CDate7 <- '2019-01-21'
TestDateJulian8 <- as.date(CDate7, order = "ymd")
#
# Date format helps:
#
# %y: 2-digit year (07)
# %Y: 4-digit year (2007)
# %b: abbreviated month (Jan)
# %B: Unabbreviated month (January)
# %m: month as number (00-12 (?))
# %a: abbreviated weekday (Mon)
# %A: Unabbreviated weekday (Monday)
# %d: day as a number (0-31 (?))
today <- Sys.Date()
is.date(today)
class(today)
is.Date(today)
Today1 <- format(today, format="%B %d %Y")
Today1
Today2 <- format(today, format="%m/%d/%Y")
Today2
```

A1.8 R Basics Optimization.R

In this simple R code, we illustrate using the optim() function within the stats package. Quantitative finance often requires solving for embedded parameters.

```
# Example 1: Minimize joint function (say option value and implied volatility)
#   Simple function assumed here for illustration, useful in Merton model related to
#   credit risk.
OptionValue = 14.23
Volatility = 30
# Specify the variables over which we optimize as the first input (a vector)
# Specify other parameters as second, third, etc. inputs
Solver1 <- function(variables, OptionValue, Volatility){
  variable1 <- variables[1]
  variable2 <- variables[2]
  return((variable1 - OptionValue)^2 + (variable2 - Volatility)^2)
}
# Minimization procedure
# First argument in optim is the vector of initial values of the variable optimized over,
#   assumed zero here
# Second argument is the function to be minimized
# Third argumnet is the gradient of the function being minimized (can be NULL)
# Remaining argument are the parameters to be passed to fn and gr
# The following minimizes Solver1. To check the optimization procedure: we expect
#   the result to be simply [14.23, 30]
OptimizationResults = optim(par = c(0,0), fn = Solver1, gr = NULL, method = "BFGS",
  OptionValue, Volatility)
OptimizationResults
# Output: par - best set of parameters found
#         value - value of the function (fn) with parameters, par
#         convergence - 0 if successful, several potential error codes
ImpliedOptionValue = OptimizationResults$par[1]
ImpliedVolatility = OptimizationResults$par[2]
```

```
OptimiumValue = OptimizationResults$convergence
ImpliedOptionValue; ImpliedVolatility; OptimiumValue
```

A1.9 R Basics VaR and Graphs.R

Note that this program requires several packages.

```
# Libraries
# ggplot2 - creating quality plots
# quantmod - pulling stock data
# KableExtra - aid in formatting some of our results
# Knitr - create RMarkdown file producing a record of our results
# openxlsx - Read and write to excel files
Packages <- c("ggplot2", "quantmod", "KableExtra", "Knitr", "openxlsx")
if(length(setdiff(Packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(Packages, rownames(installed.packages())))
} # Make sure libraries are installed on this computer
lapply(Packages, library, character.only=TRUE) # Load and attach libraries
rm(Packages)
```

Basic data to pull stock information from the internet using quantmod.

```
# For illustration purposes only, it is not important to fully understand
#   value-at-risk (VaR) at this point.
# The goal with this program is merely illustrative.
ConfidenceLevel <- 0.95 # Define the confidence interval
NumberOfObservations <- 252
Security1 <- "AMZN" # Ticker
Import <- TRUE # If TRUE, then import from yahoo, if FALSE, read in file
# Import historical stock data via quantmod (Security1 abbreviated S1)
if(Import == TRUE){
  S1Data <- getSymbols(Security1, auto.assign = FALSE)
# Though the line below evaluates to TRUE, contains many other features
  is.numeric(S1Data)
# Note data is type xts, an time series class like zoo,
#  data in ascending order
  head(S1Data, 5)
  tail(S1Data, 5)
# Convert xts to numeric, strips index and returns only observation
  S1GenericData <- coredata(S1Data)
  is.numeric(S1GenericData)
  head(S1GenericData, 5)
  tail(S1GenericData, 5)
# Must use S1GenericData so AdjClose is not overwritten
  S1AdjPrices <- data.frame("Date" = index(S1Data),
    "AdjClose" = S1GenericData[,6])
# Note the column is generic (no AAPL reference)
  head(S1AdjPrices, 5)
  tail(S1AdjPrices, 5)
# MS Excel will store date as Julian (base date is 12/30/1899)
  write.xlsx(S1AdjPrices, "A1.9 S1AdjPrices.xlsx")
} else {
  S1AdjPrices <- read.xlsx(xlsxFile = "A1.9 S1AdjPrices.xlsx", sheet = 1,
    skipEmptyRows = FALSE)
# MS Excel base year is 12/30/1899
  S1AdjPrices$Date <- as.Date(S1AdjPrices$Date, origin = "1899-12-30")
}
# Trim data based on NumberOfObservations
S1AdjPrices1 <- tail(S1AdjPrices, NumberOfObservations)
# Current price is used in analytic value-at-risk below
CurrentPrice <- S1AdjPrices1$AdjClose[NumberOfObservations]
# Convert to zoo object for analysis
S1AdjPrices1z <- zoo(S1AdjPrices1, as.Date(S1AdjPrices1$Date))
S1AdjPrices1z$Date <- NULL # Delete Date from zoo object
```

Calculate required information for sorting and plotting.

```
# Calculate continuously compounded returns
S1R = diff(log(as.numeric(S1AdjPrices1z$'AdjClose'))) # CC rates of return
```

```
S1FD = diff(as.numeric(S1AdjPrices1z$'AdjClose')) # First difference
# Lose one observation
A = NA # Need to append one due to loss of one in calculating returns
# Need date for appending first observation below
TDate <- as.Date(index(S1AdjPrices1z))
# zoo object same size as S1AdjPrices1z
S1R <- zoo(append(S1R, A, after = 0), TDate)
# zoo object same size as S1AdjPrices1z
S1FD <- zoo(append(S1FD, A, after = 0), TDate)
# Add the return column
S1z <- cbind(S1AdjPrices1z, S1R, S1FD)
LengthS1z = length(S1z$S1R)
for(i in 1:LengthS1z){
  if(i==1){
    S1z$S1TR[1] = 1.0
    S1z$S1TD[1] = 0.0
  } else {
    S1z$S1TR[i] = as.numeric(S1z$S1TR[i-1]) * exp(as.numeric(S1z$S1R[i]))
    S1z$S1TD[i] = as.numeric(S1z$S1TD[i-1]) + as.numeric(S1z$S1FD[i])
  }
}
# Extract security 1 returns and first differences from zoo
S1R <- as.numeric(S1z$S1R)
S1FD <- as.numeric(S1z$S1FD)
# Remove first observation (it is NA)
S1R <- S1R[-1]
S1FD <- S1FD[-1]
# Compute daily mean, convert to numeric and ignore NAs
S1RMean <- mean(S1R, na.rm = TRUE)
S1RSD <- sd(S1R, na.rm = TRUE)
S1FDMean <- mean(S1FD, na.rm = TRUE)
S1FDSD <- sd(S1FD, na.rm = TRUE)
# Lose one observation at the beginning
NumberOfReturns <- length(S1R)
```

Preparing for histogram display.

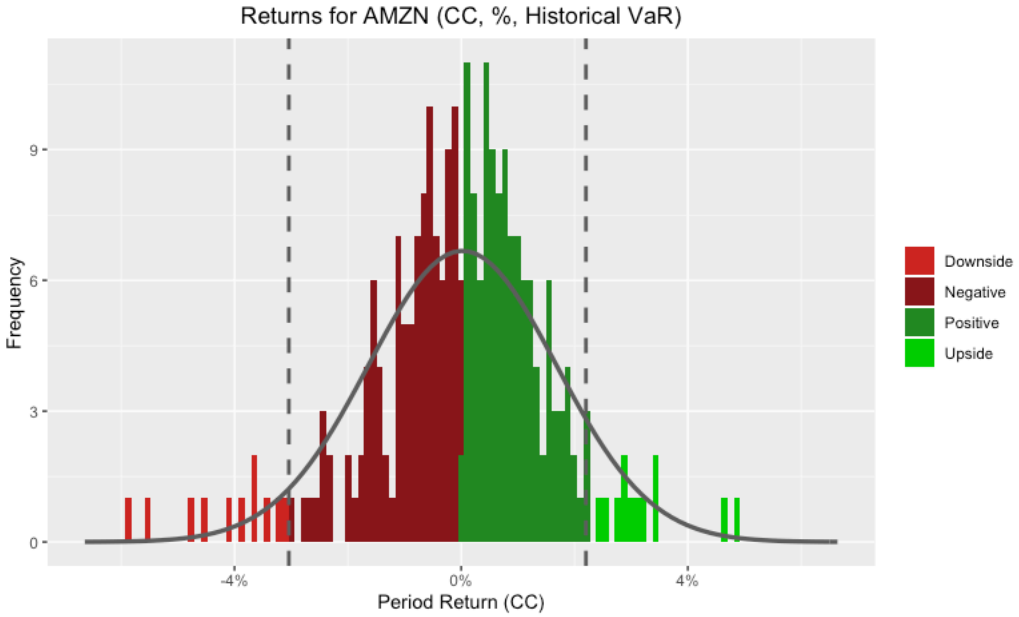
```
# Width of each bin for histogram
S1RBinWidth <- S1RSD/15
S1FDBinWidth <- S1FDSD/15
# For Historical VaR, we must sort all of our observations
S1R <- sort(S1R)
S1FD <- sort(S1FD)
# VaR as a percent
HistoricalVaRR <- as.numeric(quantile(S1R, ConfidenceLevel))
# VaR as a Dollar Value
HistoricalVaRD <- as.numeric(quantile(S1FD, ConfidenceLevel))
# Illustrate text conversions as functions
# Note: NumberOfDigits = 2 sets the default values if nothing is passed
DecimalToPercentInText <- function(x, NumberOfDigits = 2) {
  stopifnot(is.numeric(x)) # Make sure it is numeric
  x = x * 100 # Convert decimal to percent
  y = round(x, NumberOfDigits) # Round, not truncate, at appropriate level
# Numeric to text, with number of digits on RHS
  z = format(y, nsmall = NumberOfDigits)
  return(paste0(z, "%"))
}
DollarToDollarInText <- function(x, NumberOfDigits = 2) {
  stopifnot(is.numeric(x))
  y = round(x, NumberOfDigits)
  z = format(y, nsmall = NumberOfDigits)
  return(paste0("$", z))
}
TextHistoricalVaRR <- DecimalToPercentInText(HistoricalVaRR, 4)
TextHistoricalVaRD <- DollarToDollarInText(HistoricalVaRD, 4)
# Print values
```

```
print(TextHistoricalVaRR)
print(TextHistoricalVaRD)
# Returns graphed
# Categorize outccomes based on VaR confidence level (Return)
Segment <- NULL
for (i in 1:NumberOfReturns){
  if(S1R[i] < S1R[(1-ConfidenceLevel) * NumberOfReturns]){
    Segment[i] <- "Downside"
  }
  else if (S1R[i] > S1R[ConfidenceLevel * NumberOfReturns]){
    Segment[i] <- "Upside"
  }
  else if (S1R[i] >= 0 & S1R[i] <= S1R[ConfidenceLevel * NumberOfReturns]){
    Segment[i] <- "Positive"
  }
  else if (S1R[i]<=0 & S1R[i]>=S1R[(1-ConfidenceLevel) * NumberOfReturns]){
    Segment[i] <- "Negative"
  }
}
# Create data frame for graphing returns (GP denotes graph parameters)
S1RGP <- data.frame(CCReturn = S1R, Segment)
```

Produce several histograms.

```
# Illustrate with histogram.
ggplot(data = S1RGP) +
  geom_histogram(aes(S1RGP[,1], fill = Segment), binwidth = S1RBinWidth) +
  geom_vline(xintercept = S1R[ConfidenceLevel * NumberOfReturns],
    linetype = "dashed", color = "gray38", size = 1) +
  geom_vline(xintercept = S1R[(1-ConfidenceLevel) * NumberOfReturns],
    linetype = "dashed", color = "gray38", size = 1) +
  scale_fill_manual(values=
    c('firebrick3', 'firebrick4', 'forestgreen', 'green3')) +
  labs(x = "Period Return (CC)", y = "Frequency",
    title = paste0("Returns for ", Security1,
    " (CC, %, Historical VaR)")) +
  theme(legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(labels = scales::percent, limits=c(-4*S1RSD, 4*S1RSD)) +
  scale_y_continuous(labels = scales::comma) +
  stat_function(
    fun = function(x, mean, sd, n, bw){
      dnorm(x = x, mean = mean, sd = sd) * n * bw
    }, args = c(mean = S1RMean, sd = S1RSD, n = NumberOfReturns,
      bw = S1RBinWidth),
    color = "gray38", size = 1.2)
```

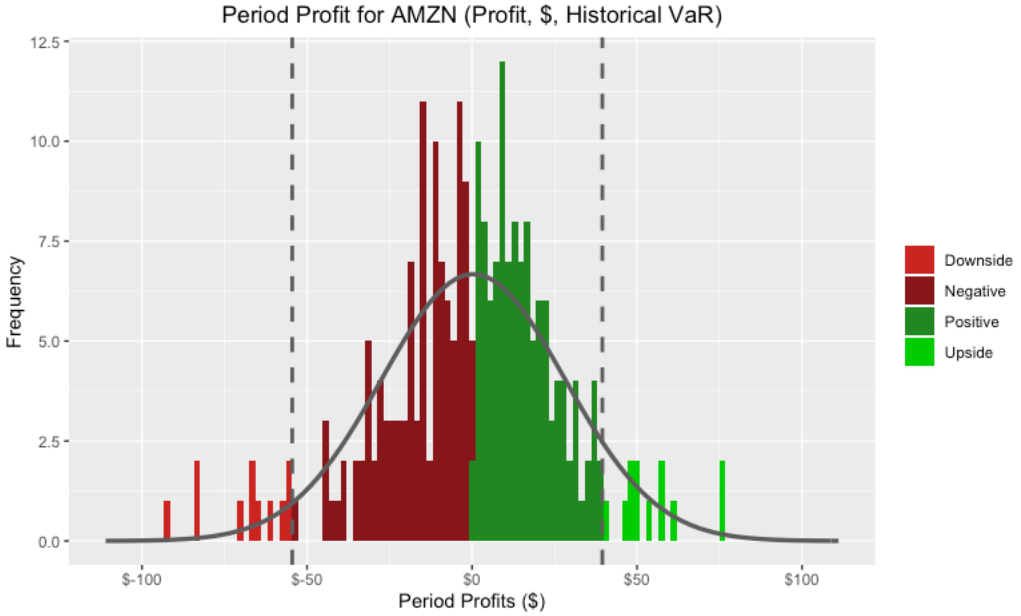
Figure A1.7 Continuously compounded returns and historical VaR for Amazon




```
# Dollar profits graphed
# Categorize outccomes based on VaR confidence level (Return)
Segment <- NULL
for (i in 1:NumberOfReturns){
  if(S1FD[i] < S1FD[(1-ConfidenceLevel) * NumberOfReturns]){
    Segment[i] <- "Downside"
  }
  else if (S1FD[i] > S1FD[ConfidenceLevel * NumberOfReturns]){
    Segment[i] <- "Upside"
  }
  else if (S1FD[i]>=0 & S1FD[i]<=S1FD[ConfidenceLevel * NumberOfReturns]){
    Segment[i] <- "Positive"
  }
  else if (S1FD[i]<=0 & S1FD[i]>=S1FD[(1-ConfidenceLevel) * NumberOfReturns]){
    Segment[i] <- "Negative"
  }
}
# Create data frame for graphing returns (GP denotes graph parameters)
S1FDGP <- data.frame(CCReturn = S1FD, Segment)
# Illustrate with histogram.
ggplot(data = S1FDGP) +
  geom_histogram(aes(S1FDGP[,1], fill = Segment), binwidth = S1FDBinWidth) +
  geom_vline(xintercept = S1FD[ConfidenceLevel * NumberOfReturns],
    linetype = "dashed", color = "gray38", size = 1) +
  geom_vline(xintercept = S1FD[(1-ConfidenceLevel) * NumberOfReturns],
    linetype = "dashed", color = "gray38", size = 1) +
  scale_fill_manual(values=c('firebrick3', 'firebrick4', 'forestgreen',
    'green3')) +
  labs(x = "Period Profits ($)", y = "Frequency",
    title = paste0("Period Profit for ", Security1,
    " (Profit, $, Historical VaR)")) +
  theme(legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(labels = scales::dollar,
    limits = c(-4*S1FDSD, 4*S1FDSD)) +
# Need to manipulate y labels eventually
  scale_y_continuous(labels = scales::comma) +
  stat_function(
    fun = function(x, mean, sd, n, bw){
      dnorm(x = x, mean = mean, sd = sd) * n * bw
```

```
    }, args = c(mean = S1FDMean, sd = S1FDSD, n = NumberOfReturns,
      bw = S1FDBinWidth),
    color = "gray38", size = 1.2)
```

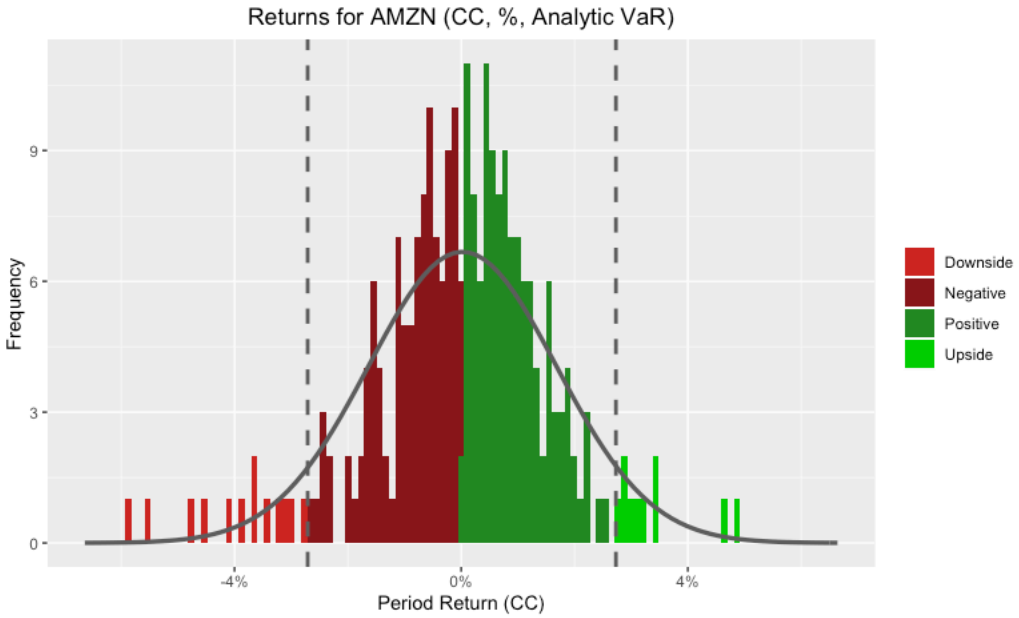
Figure A1.8 Dollar returns and historical VaR for Amazon




```
#
# Work on analytic value-at-risk based on normal distribution
#
Lambda = qnorm(ConfidenceLevel, 0.0, 1.0, lower.tail = TRUE, log.p = FALSE)
# Note: Unit of measure is daily; hence, Horizon = 1 implies period is one
#  trading day
Horizon = 1
ReturnVaRMean = S1RSD*(Horizon^0.5)*Lambda
DollarVaRMean = CurrentPrice * ReturnVaRMean
AnalyticVaRR = -S1RMean*Horizon + ReturnVaRMean
AnalyticVaRD = CurrentPrice * AnalyticVaRR
TextAnalyticVaRR <- DecimalToPercentInText(AnalyticVaRR, 4)
TextAnalyticVaRD <- DollarToDollarInText(AnalyticVaRD, 4)
# Print values
print(TextAnalyticVaRR)
print(TextAnalyticVaRD)
# Categorize outccomes based on Analytic VaR confidence level
Segment <- NULL
for (i in 1:NumberOfReturns){
  if(S1R[i] < -AnalyticVaRR){
    Segment[i] <- "Downside"
  }
  else if (S1R[i] > S1RMean*Horizon + ReturnVaRMean){
    Segment[i] <- "Upside"
  }
  else if (S1R[i] >= 0 & S1R[i] <= S1RMean*Horizon + ReturnVaRMean){
    Segment[i] <- "Positive"
  }
  else if (S1R[i] <= 0 & S1R[i] >= -AnalyticVaRR){
    Segment[i] <- "Negative"
  }
}
# Create data frame for graphing returns (GP denotes graph parameters)
S1RGP <- data.frame(CCReturn = S1R, Segment)
# Illustrate with histogram.
ggplot(data = S1RGP) +
```

```
  geom_histogram(aes(S1RGP[,1], fill = Segment), binwidth = S1RBinWidth) +
  geom_vline(xintercept = S1RMean*Horizon + AnalyticVaRR,
    linetype = "dashed", color = "gray38", size = 1) +
  geom_vline(xintercept = -AnalyticVaRR, linetype = "dashed",
    color = "gray38", size = 1) +
  scale_fill_manual(values=c('firebrick3', 'firebrick4', 'forestgreen',
    'green3')) +
  labs(x = "Period Return (CC)", y = "Frequency",
       title = paste0("Returns for ", Security1, " (CC, %, Analytic VaR)")) +
  theme(legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(labels = scales::percent,
    limits = c(-4*S1RSD, 4*S1RSD)) +
  # Need to manipulate y labels eventually
  scale_y_continuous(labels = scales::comma) +
  stat_function(
    fun = function(x, mean, sd, n, bw){
      dnorm(x = x, mean = mean, sd = sd) * n * bw
    }, args = c(mean = S1RMean, sd = S1RSD, n = NumberOfReturns,
      bw = S1RBinWidth),
      color = "gray38", size = 1.2)
```

Figure A1.9 Continuously compounded returns and analytic VaR for Amazon




```
# Categorize outcomes based on Analytic VaR confidence level
Segment <- NULL
for (i in 1:NumberOfReturns){
  if(S1FD[i] < -AnalyticVaRD){
    Segment[i] <- "Downside"
  }
  else if (S1FD[i] > S1FDMean*Horizon + AnalyticVaRD){
    Segment[i] <- "Upside"
  }
  else if (S1FD[i] >= 0 & S1FD[i] <= S1FDMean*Horizon + AnalyticVaRD){
    Segment[i] <- "Positive"
  }
  else if (S1FD[i] <= 0 & S1FD[i] >= -AnalyticVaRD){
    Segment[i] <- "Negative"
  }
}
# Create data frame for graphing returns (GP denotes graph parameters)
S1FDGP <- data.frame(DProfit = S1FD, Segment)
```

```
# Illustrate with histogram.
ggplot(data = S1FDGP) +
  geom_histogram(aes(S1FDGP[,1], fill = Segment), binwidth = S1FDBinWidth) +
  geom_vline(xintercept = S1FDMean*Horizon + AnalyticVaRD,
    linetype = "dashed", color = "gray38", size = 1) +
  geom_vline(xintercept = -AnalyticVaRD, linetype = "dashed",
    color = "gray38", size = 1) +
  scale_fill_manual(values=c('firebrick3', 'firebrick4', 'forestgreen',
    'green3')) +
  labs(x = "Period Profit ($)", y = "Frequency",
       title = paste0("Profit for ", Security1,
         " (Profit, $, Analytic VaR)")) +
  theme(legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(labels = scales::dollar,
    limits = c(-4*S1FDSD, 4*S1FDSD)) +
  # Need to manipulate y labels eventually
  scale_y_continuous(labels = scales::comma) +
  stat_function(
    fun = function(x, mean, sd, n, bw){
      dnorm(x = x, mean = mean, sd = sd) * n * bw
    }, args = c(mean = S1FDMean, sd = S1FDSD, n = NumberOfReturns,
      bw = S1FDBinWidth),
    color = "gray38", size = 1.2)
```

Figure A1.10 Dollar returns and analytic VaR for Amazon