Intermediate R for Finance

Matt Nason

June 23, 2017

Chapter 1: Dates

1.1 What Day Is It?

```
# What is the current date?
Sys.Date()
## [1] "2017-06-24"

# What is the current date and time?
Sys.time()
## [1] "2017-06-24 03:47:25 UTC"

# Create the variable today
today <- Sys.Date()


# Confirm the class of today
class(today)
## [1] "Date"
```

1.2 From Char To Date

1.3 Many Dates

```
# Create dates from "2017-02-05" to "2017-02-08" inclusive.
dates <- c("2017-02-05","2017-02-06","2017-02-07","2017-02-08")

# Add names to dates
names(dates) <- c("Sunday","Monday","Tuesday","Wednesday")

# Subset dates to only return the date for Monday
dates["Monday"]
##       Monday
## "2017-02-06"
```

1.4 Date Formats (1)

```r
# "08,30,30"
as.Date("08,30,1930", format = "%m, %d, %Y")
## [1] "1930-08-30"

# "Aug 30,1930"
as.Date("Aug 30,1930", format = "%b %d, %Y")
## [1] "1930-08-30"

# "30aug1930"
as.Date("30aug1930", format = "%d%b%Y")
## [1] "1930-08-30"
```

1.5 Date Formats (2)

```r
# char_dates
char_dates <- c("1jan17", "2jan17", "3jan17", "4jan17", "5jan17")

# Create dates using as.Date() and the correct format
dates <- as.Date(char_dates, format= "%d%b%y")

# Use format() to go from "2017-01-04" -> "Jan 04, 17"
format(dates, format="%b %d, %y")
## [1] "Jan 01, 17" "Jan 02, 17" "Jan 03, 17" "Jan 04, 17" "Jan 05, 17"

# Use format() to go from "2017-01-04" -> "01,04,2017"
format(dates, format="%m,%d,%Y")
## [1] "01,01,2017" "01,02,2017" "01,03,2017" "01,04,2017" "01,05,2017"
```

1.6 Subtraction of Dates

```
# Dates
dates <- as.Date(c("2017-01-01", "2017-01-02", "2017-01-03"))

# Create the origin
origin <- as.Date("1970-01-01")

# Use as.numeric() on dates
as.numeric(dates)
## [1] 17167 17168 17169

# Find the difference between dates and origin
dates - origin
## Time differences in days
## [1] 17167 17168 17169
```

1.7 Months() and Weekdays() and Quarters(),Oh My!

```
# dates
dates <- as.Date(c("2017-01-02", "2017-05-03", "2017-08-04", "2017-10-17"))

# Extract the months
months(dates)
## [1] "January" "May"     "August"  "October"

# Extract the quarters
quarters(dates)
## [1] "Q1" "Q2" "Q3" "Q4"

# dates2
dates2 <- as.Date(c("2017-01-02", "2017-01-03", "2017-01-04", "2017-01-05"))

# Assign the weekdays() of dates2 as the names()
names(dates2) <- weekdays(dates2)

# Print dates2
dates2
##       Monday      Tuesday    Wednesday     Thursday 
## "2017-01-02" "2017-01-03" "2017-01-04" "2017-01-05"
```

Chapter 2: If Statements and Operators

2.1 Relational Practice

```r
# Stock prices
apple <- 48.99
micr <- 77.93

# Apple vs Microsoft
apple > micr
## [1] FALSE

# Not equals
apple != micr
## [1] TRUE

# Dates - today and tomorrow
today <- as.Date(Sys.Date())
tomorrow <- as.Date(Sys.Date() + 1)

# Today vs Tomorrow
tomorrow < today
## [1] FALSE
```

2.2 Vectorized Operations

2.3 And/Or

2.4 Not!

2.5 Logicals and Subset()

2.6 All Together Now!

2.7 If This

```
# micr
micr <- 48.55

# Fill in the blanks
if( micr < 55) {
    print("Buy!")
}
## [1] "Buy!"
```

2.8 If This, Else That

```
# micr
micr <- 57.44

# Fill in the blanks
if( micr < 55 ) {
    print("Buy!")
} else {
    print("Do nothing!")
}
## [1] "Do nothing!"
```

2.9 If This, Else If That, Else That Other Thing

```r
# micr
micr <- 105.67

# Fill in the blanks
if( micr < 55 ) {
    print("Buy!")
} else if( micr >= 55 & micr <75 ){
    print("Do nothing!")
} else {
    print("Sell!")
}
## [1] "Sell!"
```

2.10 Can You If Inside An If?

```r
# micr
micr <- 105.67
shares <- 1

# Fill in the blanks
if( micr < 55 ) {
    print("Buy!")
} else if( micr >= 55 & micr < 75 ) {
    print("Do nothing!")
} else {
    if( shares >=1 ) {
        print("Sell!")
    } else {
        print("Not enough shares to sell!")
    }
}
## [1] "Sell!"
```

2.11 ifelse()

Chapter 3: Loops

3.1 Repeat, Repeat, Repeat

```
# Stock price
stock_price <- 126.34

repeat {
  # New stock price
  stock_price <- stock_price * runif(1, .985, 1.01)
  print(stock_price)

  # Check
  if(stock_price < 125) {
    print("Stock price is below 125! Buy it while it's cheap!")
    break
  }
}
## [1] 124.5415
## [1] "Stock price is below 125! Buy it while it's cheap!"
```

3.2 When To Break?

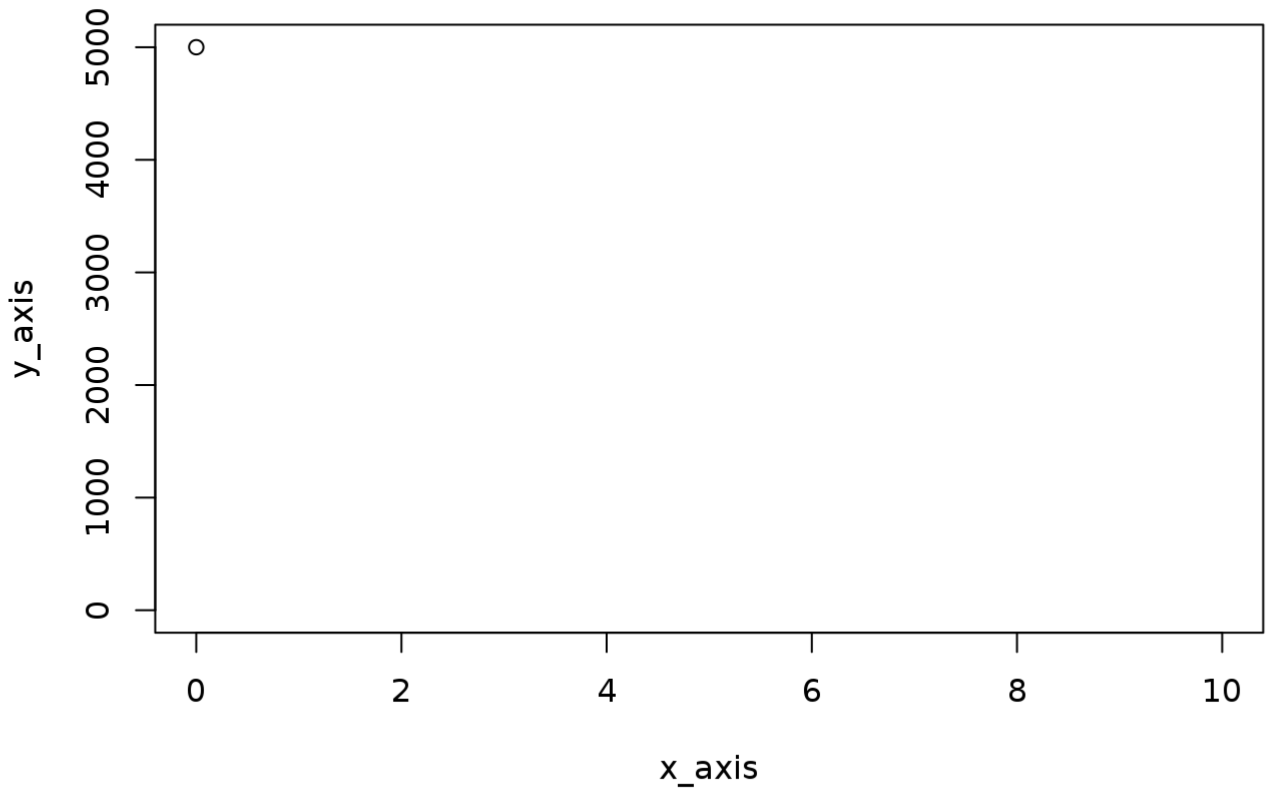
```r
# Stock price
stock_price <- 67.55

repeat {
  # New stock price
  stock_price <- stock_price * .995


  # Check
  if(stock_price < 66) {
    print("Stock price is below 66! Buy it while it's cheap!")
    break
  }
  print(stock_price)
}
## [1] 67.21225
## [1] 66.87619
## [1] 66.54181
## [1] 66.2091
## [1] "Stock price is below 66! Buy it while it's cheap!"
```

3.3 While With A Print

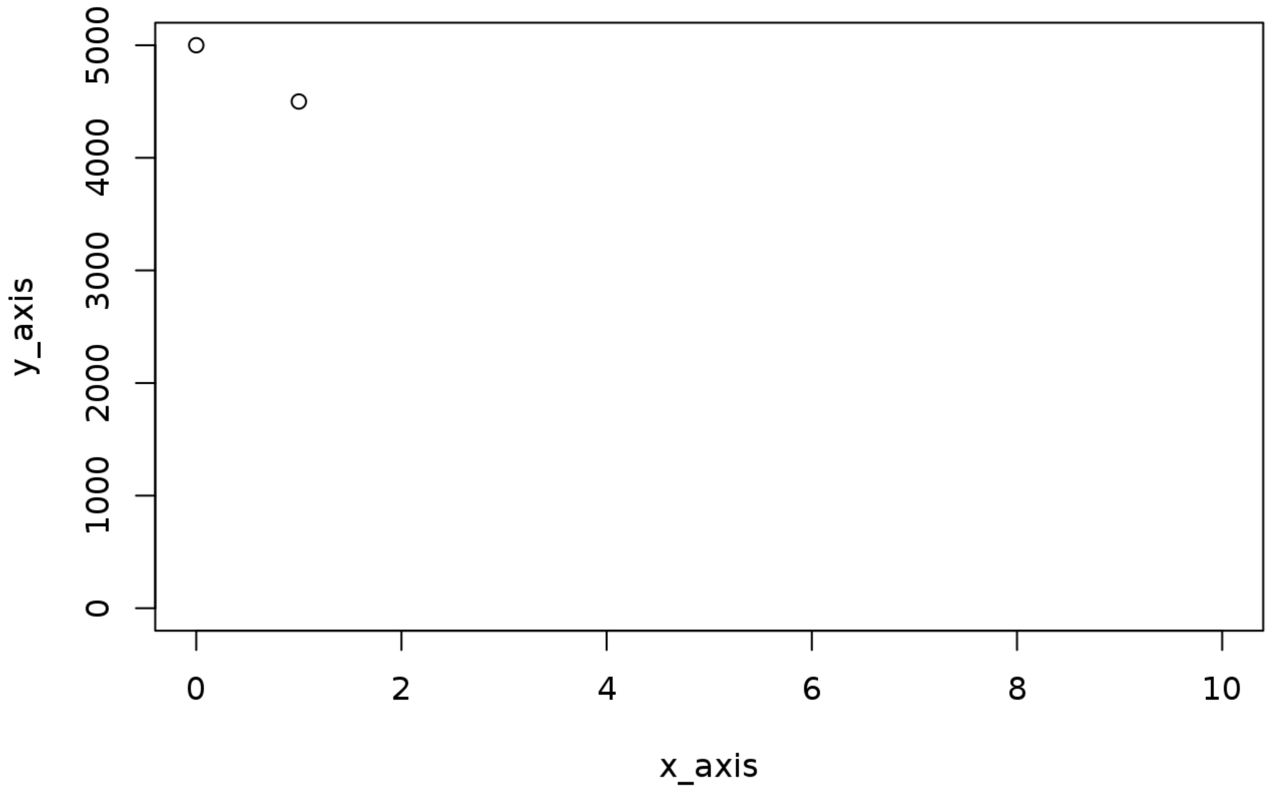
```
# Initial debt
debt <- 5000

# While loop to pay off your debt
while (debt > 0) {
  debt <- debt - 500
  print(paste("Debt remaining", debt))
}
## [1] "Debt remaining 4500"
## [1] "Debt remaining 4000"
## [1] "Debt remaining 3500"
## [1] "Debt remaining 3000"
## [1] "Debt remaining 2500"
## [1] "Debt remaining 2000"
## [1] "Debt remaining 1500"
## [1] "Debt remaining 1000"
## [1] "Debt remaining 500"
## [1] "Debt remaining 0"
```

3.4 While With A Plot

```
debt <- 5000    # initial debt
i <- 0          # x axis counter
x_axis <- i     # x axis
y_axis <- debt  # y axis

# Initial plot
plot(x_axis, y_axis, xlim = c(0,10), ylim = c(0,5000))
```

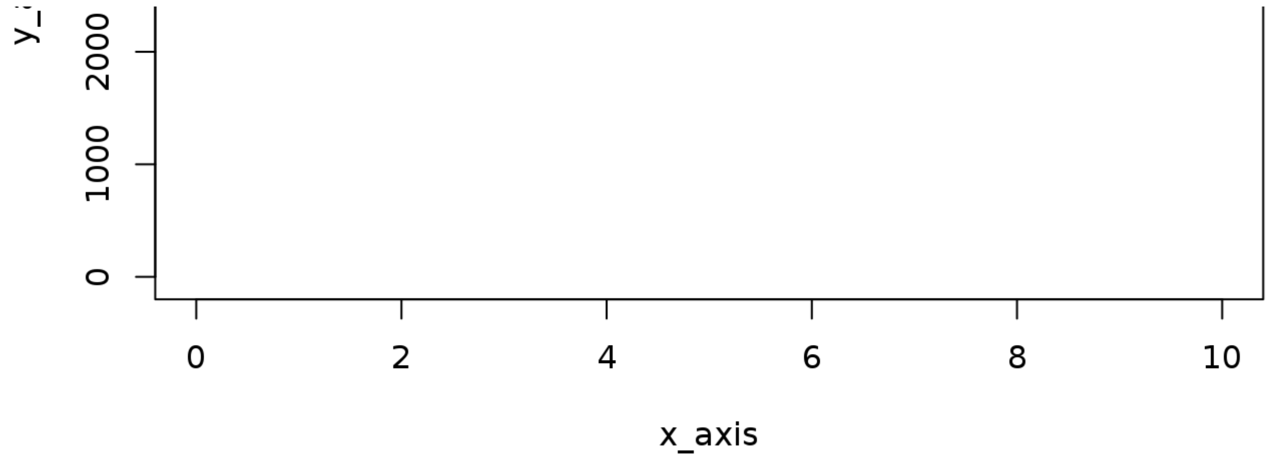

y_axis
0
1000
2000
3000
4000
5000
0
2
4
6
8
10
x_axis

```r
# Graph your debt
while (debt > 0) {

  # Updating variables
  debt <- debt - 500
  i <- i + 1
  x_axis <- c(x_axis, i)
  y_axis <- c(y_axis, debt)

  # Next plot
  plot(x_axis, y_axis, xlim = c(0,10), ylim = c(0,5000))
}
```

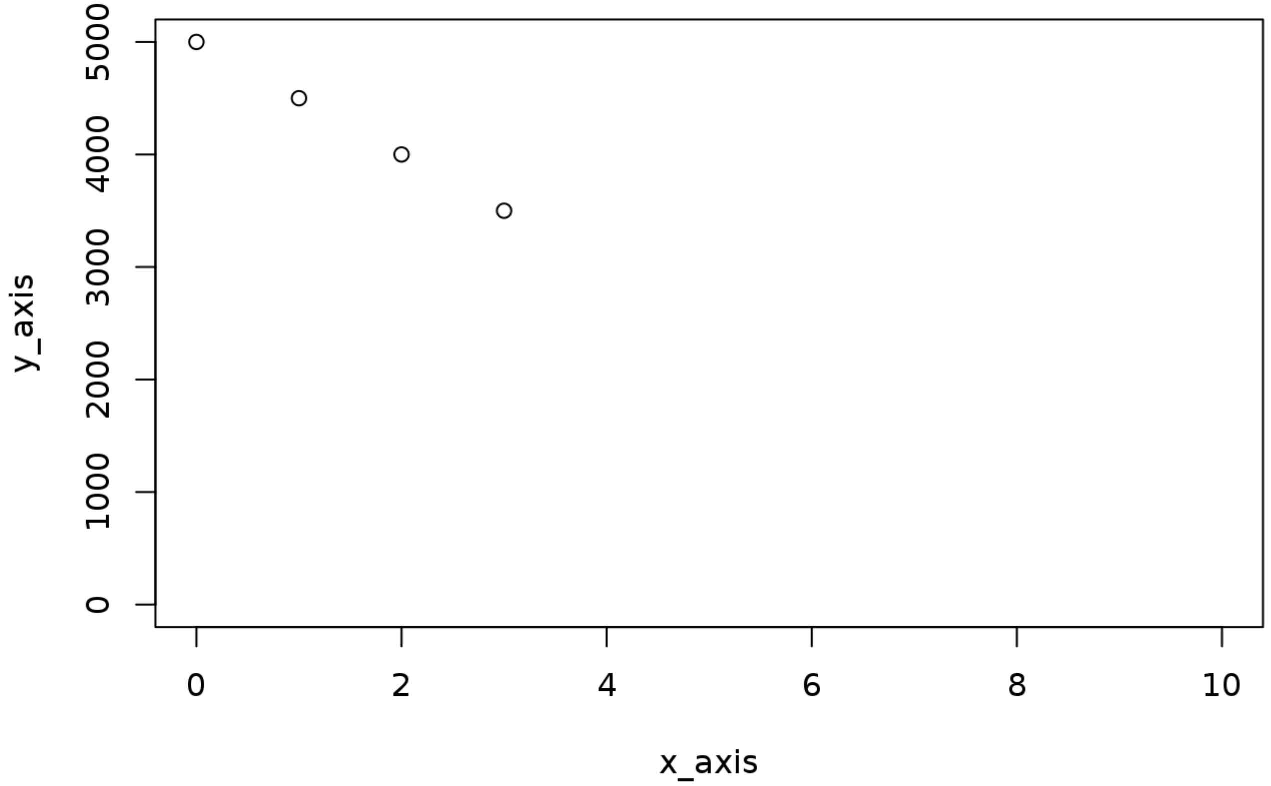


y_axis
x_axis

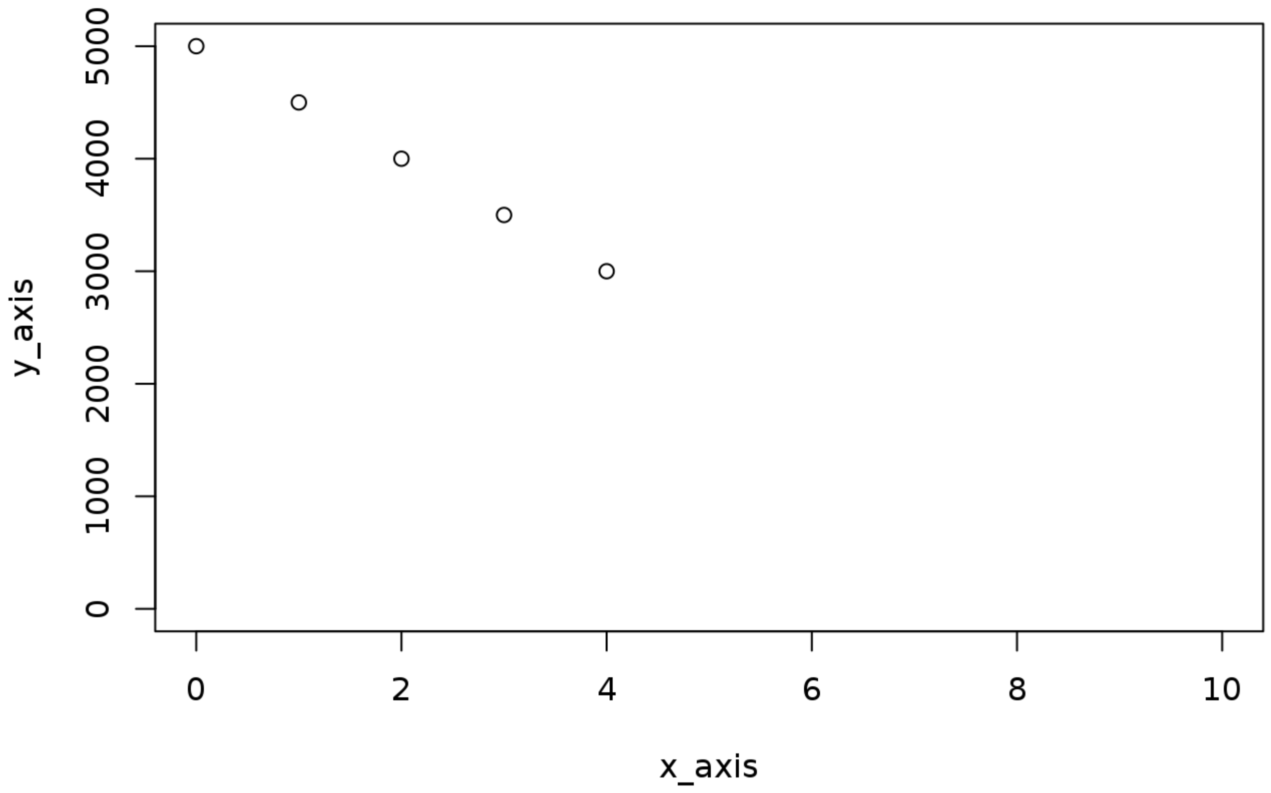
axis

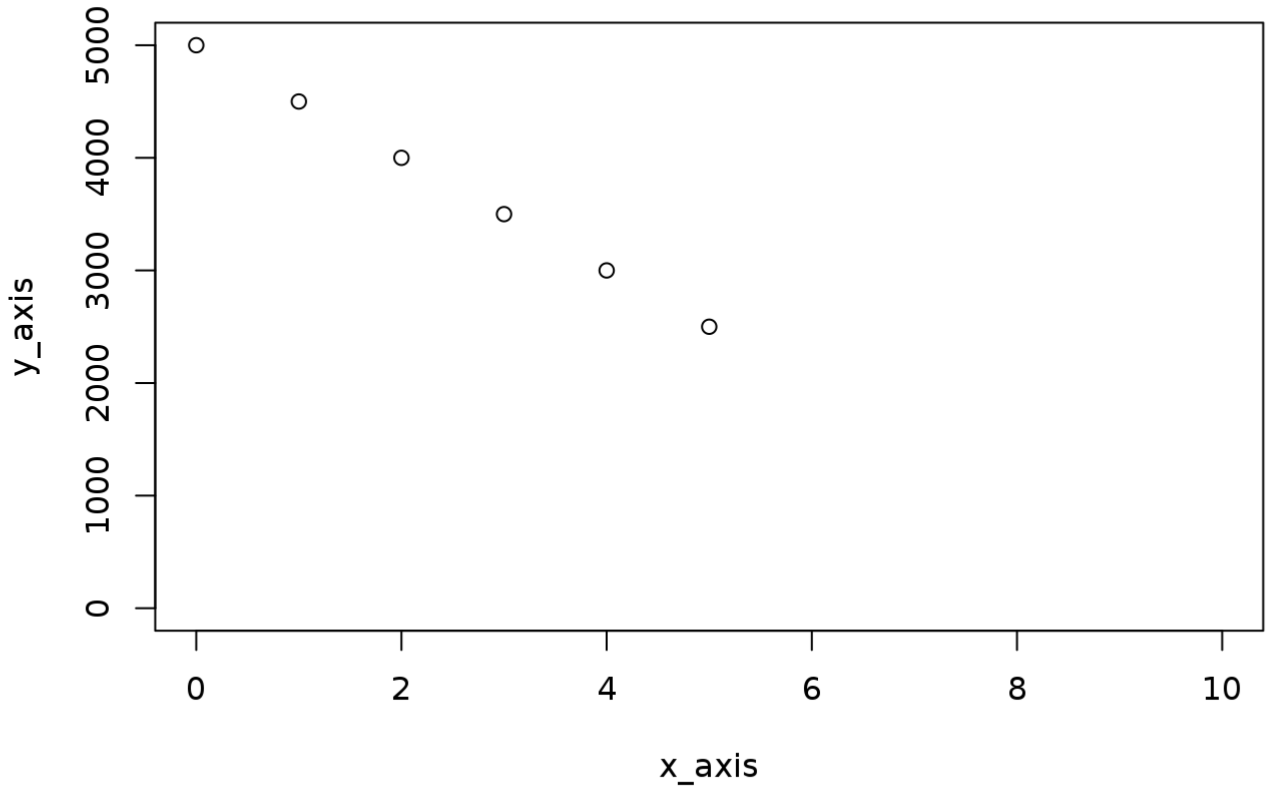


y_
2000
1000
0
0
2
4
6
8
10
x_axis

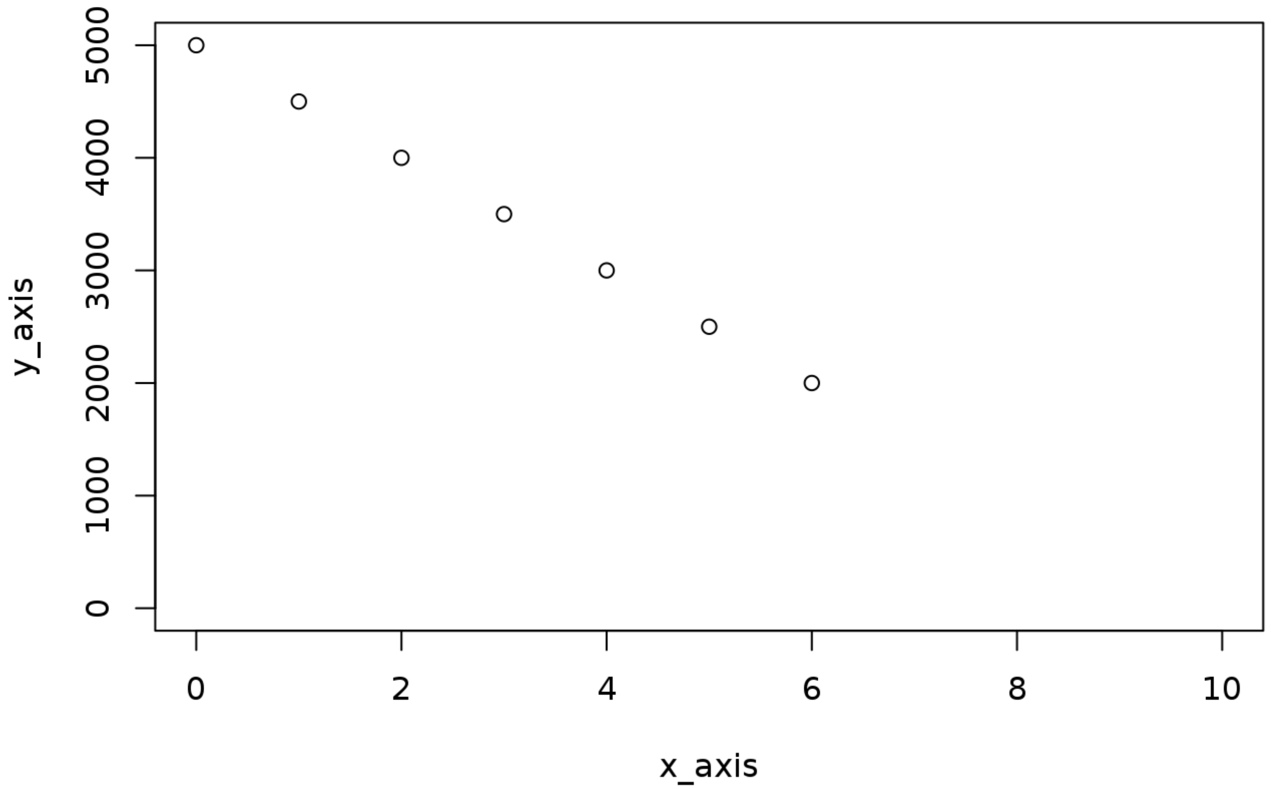


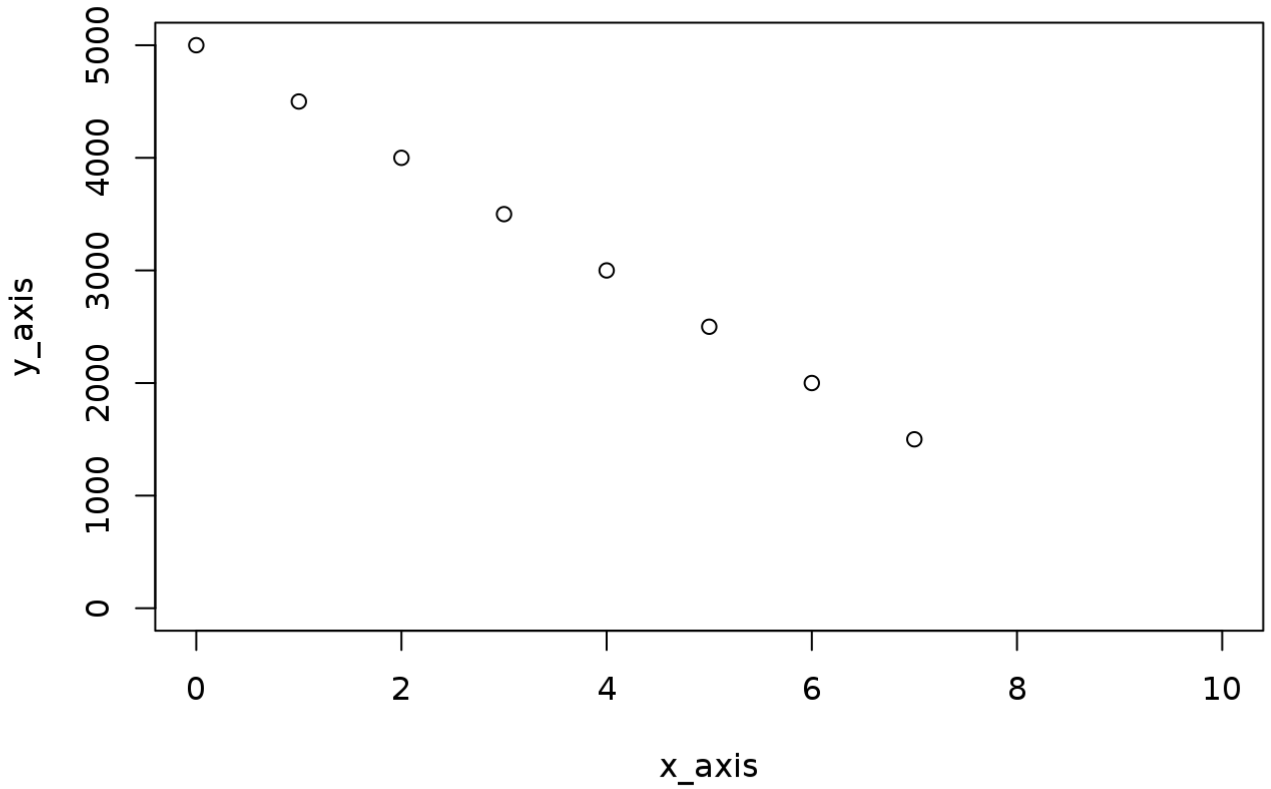
y_axis
5000
4000
3000
2000
1000
0
0
2
4
6
8
10
x_axis

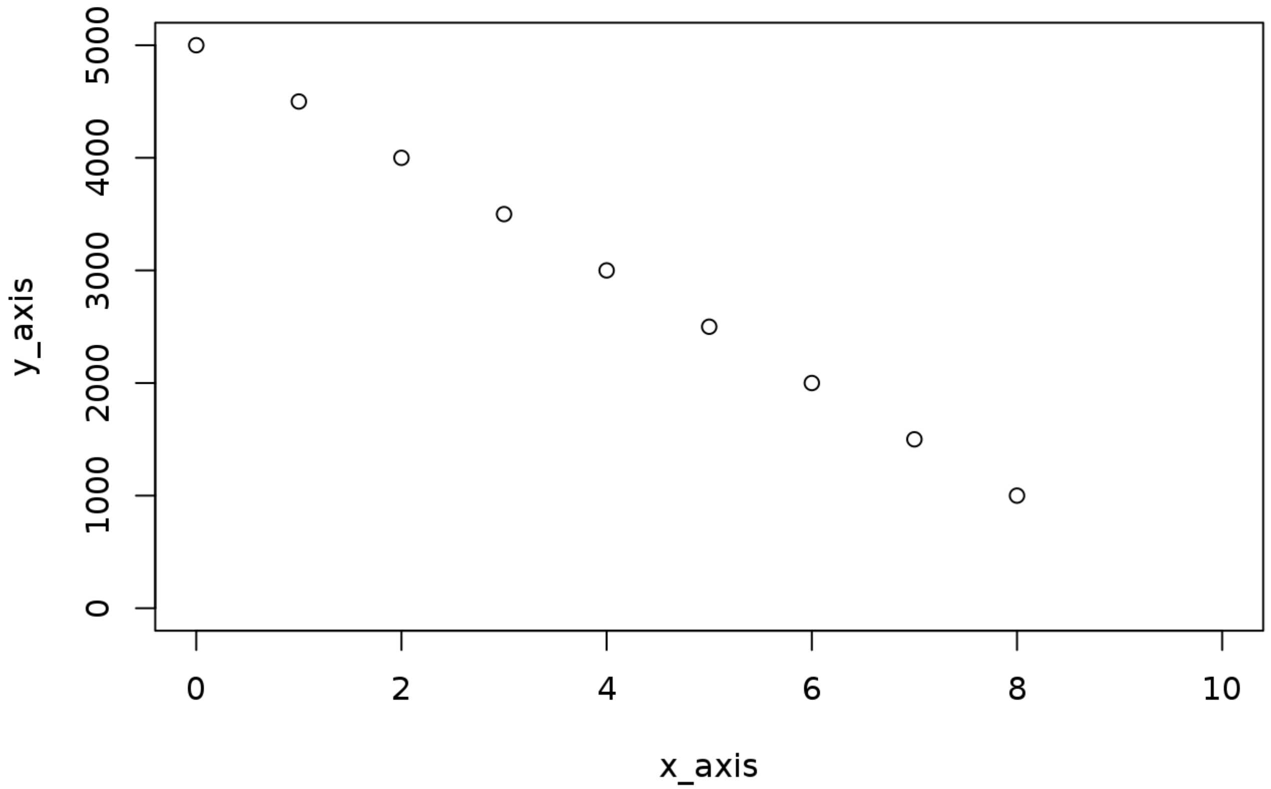

y_axis
0
1000
2000
3000
4000
5000
0
2
4
6
8
10
x_axis



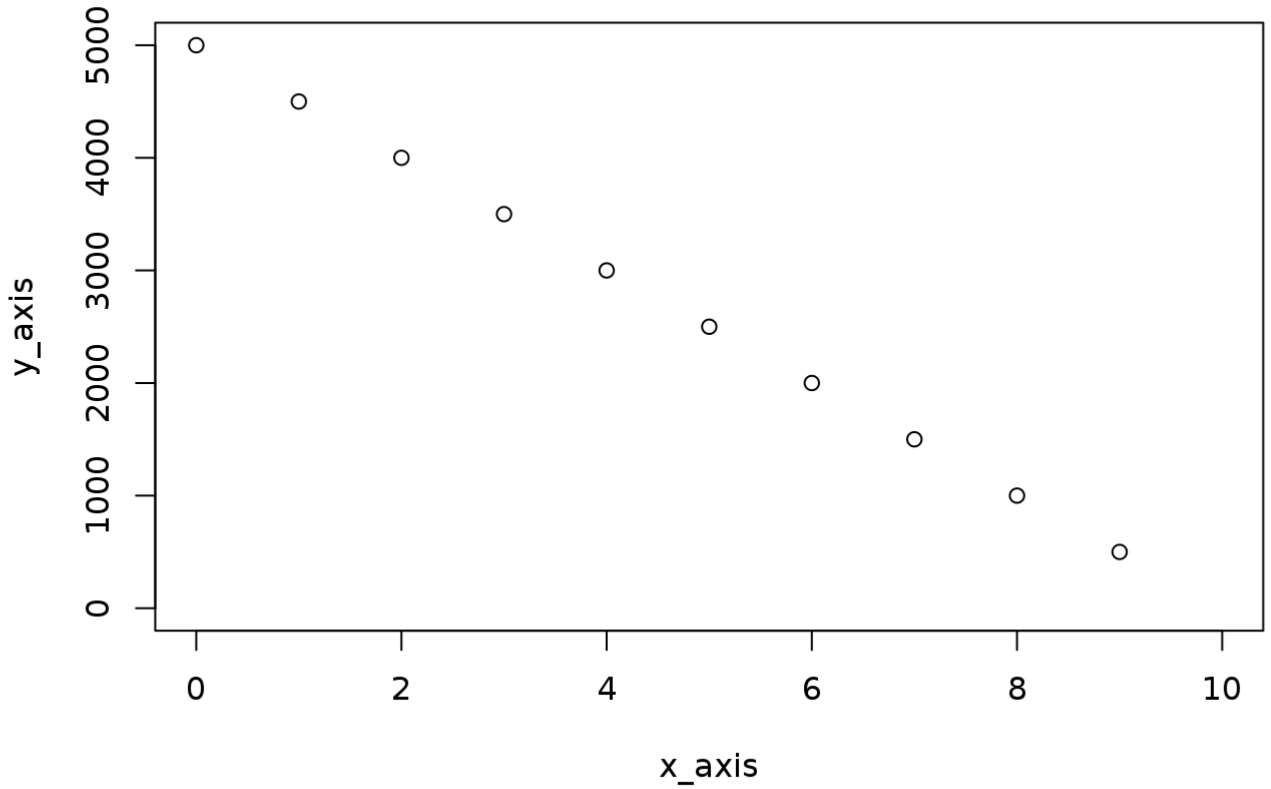
y_axis
0
1000
2000
3000
4000
5000
0
2
4
6
8
10
x_axis


y_axis
x_axis
0
1000
2000
3000
4000
5000
0
2
4
6
8
10

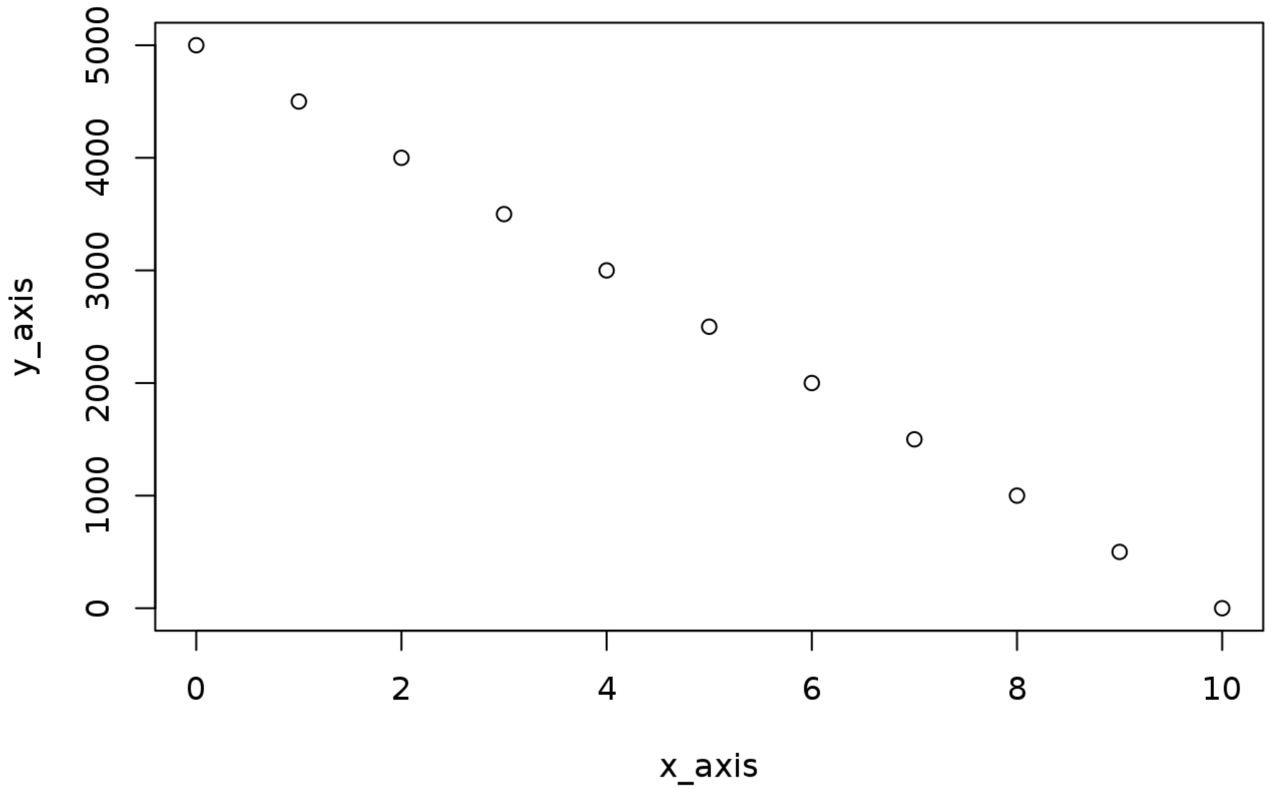

y_axis
x_axis
0
1000
2000
3000
4000
5000
0
2
4
6
8
10



y_axis
x_axis
0
1000
2000
3000
4000
5000
0
2
4
6
8
10



y_axis
0
1000
2000
3000
4000
5000
x_axis
0
2
4
6
8
10



### 3.5 Break It

```
# debt and cash
debt <- 5000
cash <- 4000

# Pay off your debt...if you can!
while (debt > 0) {
  debt <- debt - 500
  cash <- cash - 500
  print(paste("Debt remaining:", debt, "and Cash remaining:", cash))

  if (cash == 0) {
    print("You ran out of cash!")
    break
  }
}
## [1] "Debt remaining: 4500 and Cash remaining: 3500"
## [1] "Debt remaining: 4000 and Cash remaining: 3000"
## [1] "Debt remaining: 3500 and Cash remaining: 2500"
## [1] "Debt remaining: 3000 and Cash remaining: 2000"
## [1] "Debt remaining: 2500 and Cash remaining: 1500"
## [1] "Debt remaining: 2000 and Cash remaining: 1000"
## [1] "Debt remaining: 1500 and Cash remaining: 500"
## [1] "Debt remaining: 1000 and Cash remaining: 0"
## [1] "You ran out of cash!"
```

3.6 Loop Over A Vector

```r
# Sequence
seq <- c(1:10)

# Print Loop
for (value in seq) {
    print(value)
}
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10

# A sum variable
sum <- 0

# Sum Loop
for (value in seq) {
    sum <- value + sum
    print(sum)
}
## [1] 1
## [1] 3
## [1] 6
## [1] 10
## [1] 15
## [1] 21
## [1] 28
## [1] 36
## [1] 45
## [1] 55
```

3.7 Loop Over Data Frame Rows

3.8 Loop Over Matrix Elements

3.9 Break And Next

```r
# Print apple
apple
## [1] 48.99

# Loop through apple. Next if NA. Break if above 117.
for (value in apple) {
    if(is.na(value)) {
        print("Skipping NA")
        next
    }

    if(value > 117) {
        print("Time to sell!")
        break
    } else {
        print("Nothing to do here!")
    }
}
## [1] "Nothing to do here!"
```

Chapter 4: Functions

4.1 Function Help And Documentation

```r
# subset help
?subset()

# Sys.time help
?Sys.time()
```

4.2 Optional Arguments

```r
# Round 5.4
round(5.4)
## [1] 5

# Round 5.4 with 1 decimal place
round(5.4, digits=1)
## [1] 5.4

# numbers
numbers <- c(.002623, pi, 812.33345)

# Round numbers to 3 decimal places
round(numbers, digits=3)
## [1]   0.003   3.142 812.333
```

4.3 Functions In Functions

4.4 Your First Function

```r
# Percent to decimal function
percent_to_decimal <- function(percent) {
    percent / 100
}

# Use percent_to_decimal() on 6
percent_to_decimal(6)
## [1] 0.06

# Example percentage
pct <- 8

# Use percent_to_decimal() on pct
percent_to_decimal(pct)
## [1] 0.08
```

4.5 Multiple Arguments (1)

```r
# Percent to decimal function
percent_to_decimal <- function(percent, digits = 2) {
    decimal <- percent / 100

    round(decimal, digits)
}

# percents
percents <- c(25.88, 9.045, 6.23)

# percent_to_decimal() with default digits
percent_to_decimal(percents)
## [1] 0.26 0.09 0.06

# percent_to_decimal() with digits = 4
percent_to_decimal(percents, digits=4)
## [1] 0.2588 0.0904 0.0623
```

4.6 Multiple Arguments (2)

```
# Present value function
pv <- function(cash_flow, i, year) {

    # Discount multiplier
    mult <- 1 + percent_to_decimal(i)

    # Present value calculation
    cash_flow * mult ^ -year
}

# Calculate a present value
pv(1200, 7, 3)
## [1] 979.5575
```

4.7 Function Scope (1)

percent_to_decimal <- function(percent) { decimal <- percent / 100 decimal }

percent_to_decimal(6) [1] 0.06

Answer= Error ### 4.8 Function Scope (2) hundred <- 100

percent_to_decimal <- function(percent) { percent / hundred }

Answer= 0.06 ### 4.9 Tidyquant Package

Chapter 5: Apply

5.1 lapply() On A List

5.2 lapply() On A Data Frame

5.3 FUN Arguments

5.4 sapply() VS lapply()

5.5 Failing To Simplify

```
# Market crash with as.Date()
market_crash <- list(dow_jones_drop = 777.68,
                     date = as.Date("2008-09-28"))

# Find the classes with sapply()
sapply(market_crash, class)
## dow_jones_drop           date
##      "numeric"         "Date"

# Market crash with as.POSIXct()
market_crash2 <- list(dow_jones_drop = 777.68,
                      date = as.POSIXct("2008-09-28"))

# Find the classes with lapply()
lapply(market_crash2, class)
## $dow_jones_drop
## [1] "numeric"
##
## $date
## [1] "POSIXct" "POSIXt"

# Find the classes with sapply()
sapply(market_crash2, class)
## $dow_jones_drop
## [1] "numeric"
##
## $date
## [1] "POSIXct" "POSIXt"
```

5.6 vapply() VS sapply()

5.7 More vapply()

5.8 Anonymous Functions

Quiz 1 Answers

1. Compare vectors and matrices. What are similarities and differences?

Vectors are a collection of data that are a single data type, such as the stock market. A matrix is a 2D vector, with multiple vectors put into rows and columns using cbind and rbind.

2. Compare matrices and data frames. What are similarities and differences?

Again, matrices are multiple vectors put into rows and columns. Data frames are similar in that data is put into eows and columns to form a grid, but data frames can have multiple data types, such as character and numerical, in a single data frame. A matrix cannot.

3. Create your first vector, matrix, data frame, factor, and list. Do this within a R code chunk.

Vector

```r
# Another numeric vector
ibm_stock <- c(159.82, 160.02, 159.84)

# Another character vector
finance <- c("stocks", "bonds", "investments")

# A logical vector
logic <- c(TRUE,FALSE,TRUE)
```

Matrix

```
# A vector of 9 numbers
my_vector <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)

# 3x3 matrix
my_matrix <- matrix(data = c(1,2,3,4,5,6,7,8,9), nrow = 3, ncol = 3)

# Print my_matrix
my_matrix
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

# Filling across using byrow = TRUE
matrix(data = c(2, 3, 4, 5), nrow = 2, ncol = 2, byrow = TRUE)
##      [,1] [,2]
## [1,]    2    3
## [2,]    4    5
```

Data Frame

```
# Variables
company <- c("A", "A", "A", "B", "B", "B", "B")
cash_flow <- c(1000, 4000, 550, 1500, 1100, 750, 6000)
year <- c(1, 3, 4, 1, 2, 4, 5)

# Data frame
cash <- data.frame(company, cash_flow, year)

# Print cash
cash
##   company cash_flow year
## 1       A      1000    1
## 2       A      4000    3
## 3       A       550    4
## 4       B      1500    1
## 5       B      1100    2
## 6       B       750    4
## 7       B      6000    5
```

Factor

```r
# credit_rating character vector
credit_rating <- c("BB", "AAA", "AA", "CCC", "AA", "AAA", "B", "BB")

# Create a factor from credit_rating
credit_factor <- factor (credit_rating)

# Print out your new factor
credit_factor
## [1] BB  AAA AA  CCC AA  AAA B   BB
## Levels: AA AAA B BB CCC

# Call str() on credit_rating
str(credit_rating)
##  chr [1:8] "BB" "AAA" "AA" "CCC" "AA" "AAA" "B" "BB"

# Call str() on credit_factor
str(credit_factor)
##  Factor w/ 5 levels "AA","AAA","B",..: 4 2 1 5 1 2 3 4
```

List

```
# List components
name <- "Apple and IBM"
apple <- c(109.49, 109.90, 109.11, 109.95, 111.03)
ibm <- c(159.82, 160.02, 159.84, 160.35, 164.79)
cor_matrix <- cor(cbind(apple, ibm))

# Create a list
portfolio <- list(name,apple,ibm,cor_matrix)

# View your first list
portfolio
## [[1]]
## [1] "Apple and IBM"
##
## [[2]]
## [1] 109.49 109.90 109.11 109.95 111.03
##
## [[3]]
## [1] 159.82 160.02 159.84 160.35 164.79
##
## [[4]]
##           apple       ibm
## apple 1.0000000 0.9131575
## ibm   0.9131575 1.0000000
```

Quiz 2 Answers