Problem Set 4

Tim de Rooij

Instructions

First, save this notebook with a title consisting of "ProblemSet4" and then your last name, with no spaces. So, for Bob Bunny, the file should be called "ProblemSet4Bunny". Also, type your name in the subtitle on line 2 above. Each problem will have a problem statement, followed with space for you to answer. Type your work in the code chunk where it says, "Type your answer in here". If we ask a question that has a numeric answer (for example, "How much money does Bob Bunny owe?"), then you should write some text in the notebook with your answer; don't assume that just the R printout is sufficient.

When you're all done, knit the notebook to a PDF, and upload both the notebook and the PDF to Canvas. (Note: The notebook will have the file extension ".rmd".)

Before we get started, let's clear out the environment:

```
rm( list = ls() )
```

Problem 1: More Quadratic Equations

A *quadratic equation* is an equation of the form:

$$ax^2 + bx + cx = 0$$

From high-school algebra, we know that one solution of the quadratic equation is:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

The other solution has the form:

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

If you look at these formulas, you might notice that there is something important going on underneath the square root sign:

- If $b^2 > 4ac$, then the equation has two distinct real roots.
- Otherwise, if $b^2 < 4ac$, then the quantity under the square root sign is negative, and the solutions will be complex numbers, so that there will be no real roots.
- Finally, if $b^2 = 4ac$, then the equation has one real root.

Part a)

Write a function named `quadratic.roots()` that takes in the three coefficient of the quadratic equation and returns these values:

- If there are two distinct real roots, then the function returns the string "2 distinct real roots"
- If there is just one real root, then the function returns the string "1 real root"
- If there are no real roots, then the function returns the value "No real roots"

```r
setwd("/Users/TdR/Coding/R/IntroductiontoR")

# create function quadratic roots
quadratic.roots <- function( a, b, c ) {
  if( b^2 > 4 * a * c ) {
    message( "2 distinct real roots" )
  } else if( b^2 < 4 * a * c ) {
    message( "no real roots" )
  } else if( b^2 == 4 * a * c ) {
    message( "one real root" )
  }
}
```

Test this function on these equations:

$$x^2 - 6x - 16 = 0$$

```r
# give coefficients (1, -6, -16) as arguments to function
quadratic.roots( 1, -6, -16 )
```

```
## 2 distinct real roots
```

Now try it on this equation:

$$x^2 - 14x + 49 = 0$$

```r
# give coefficients (1, -14, -49) as arguments to function
quadratic.roots( 1, -14, 49 )
```

```
## one real root
```

Finally, try it on this equation:

$$x^2 - 8x + 20 = 0$$

The product is Not a Number (NaN). This happens because we are tyring to take the square root of a negative number, as negative numbers can't have square roots. In our case, -8^2 is -64, when we substract 80 (4 * 1 * 20) we get -144. Taking the square root of -144 results in NaN.

```r
# give coefficients (1, -8, 20) as arguments to function
quadratic.roots( 1, -8, 20 )
```

```
## no real roots
```

Problem 2: A Better Quadratic Solver

Part a)

Write a function named `quadratic.solver.2()` that takes the three coefficients from a quadratic equation and returns a vector:

- If there are two distinct roots x_1 and x_2, then the function should return a vector of length 2 containing the two distinct solutions x_1 and x_2.
- If there are no real roots, then the function should return a vector of length 0.
- If there is exactly one real root x_1, then the function should return a vector of length 1 containing this solution.

Thus, if you are given the equation $2x^2 - 10x + 8$, which has solutions $x_1 = 1$ and $x_2 = 4$, then your function call should be `quadratic.solver.2(2, -10, 8)`, and it should return a vector of length 2 containing the values 1 and 4.

Answer

```
# write the function
quadratic.solver.2 <- function( a, b, c ) {
  if( b^2 > 4 * a * c ) {
    message( "2 distinct real roots" )
    x1 <- ( -b + sqrt( b^2 - 4 * a * c ) ) / ( 2 * a )
    x2 <- ( -b - sqrt( b^2 - 4 * a * c ) ) / ( 2 * a )
    ( ( c( x1, x2 ) ) )
  } else if( b^2 < 4 * a * c ) {
    message( "No real roots" )
    ( ( as.numeric( 0 ) ) )
  } else if( b^2 == 4 * a * c ) {
    message( "1 real root" )
    x1 <- -b / (2 * a)
    ( ( c( x1 ) ) )
  }
}
```

Now let's test this on the example in the problem statement:

```
quadratic.solver.2( 2, -10, 8 )
```

```
## 2 distinct real roots
```

```
## [1] 4 1
```

Here are some other equations to test:

$$x^2 - 3x - 10 = 0$$

This function has two roots 5 and -2. Run your `quadratic.solver.2()` function on this equation, and see if you get those values:

```
# run the function with coefficients
quadratic.solver.2( 1, -3, -10 )
```

```
## 2 distinct real roots
```

```
## [1]  5 -2
```

Now let's consider this equation:

$$x^2 - 6x + 9 = 0$$

This equation has exactly one real root, $x = 3$. What do you get when you run `quadratic.solver.2()` on this equation?

```
# run the function with coefficients
quadratic.solver.2( 1, -6, 9 )
```

```
## 1 real root
```

```
## [1] 3
```

Next, consider this equation:

$$x^2 - 3.5x - 30 = 0$$

This equation has two roots: 7.5 and -4. What do you get when you run your `quadratic.solver.2()` on this equation?

```
# run the function with coefficients
quadratic.solver.2( 1, -3.5, -30 )
```

```
## 2 distinct real roots
```

```
## [1]  7.5 -4.0
```

Next, try this equation:

$$x^2 - 8x + 20 = 0$$

This equation has no real roots, so your function should return an empty vector of length 0.

```
# run the function with coefficients
quadratic.solver.2( 1, -8, 20 )
```

```
## No real roots
```

```
## [1] 0
```

Finally, this equation has two roots: 5 and -5:

$$x^2 - 25 = 0$$

This equation has two roots: 5 and -5. What do you get when you run `quadratic.solver()`:

```
# run the function with coefficients
quadratic.solver.2( 1, 0, -25 )
```

```
## 2 distinct real roots
```

```
## [1]  5 -5
```

Problem 3: Financial Functions

Suppose we loan a certain amount of money, denoted PV (which means "Present Value"), for T years at an interest rate of r. Then the final value of the loan, denoted FV, is:

$$FV = PV \times (1 + r)^T$$

This is an equation in four variables, so if we know 3 of them, we can solve for the fourth. For instance:

$$\begin{aligned} PV &= \frac{FV}{(1+r)^T} \\ r &= \sqrt[T]{\frac{FV}{PV}} - 1 \\ T &= \frac{\log FV - \log PV}{\log(1+r)} \end{aligned}$$

Part a)

Write a function called `calculate.FV()` that takes the present value `PV`, interest rate `r`, and time duration `T` as arguments, and returns the future value of the loan.

Answer

```r
# write function for FV
calculate.FV <- function( PV, r, t ) {
  message( "Future Value equals $", round( PV * ( 1 + r )^t, 2 ) )
}
```

Suppose Taylor lends Jamey 180 dollars, at a 5.5% interest rate, for 3 years. Using your function `calculate.FV()`, what is the final value of the loan?

```r
# run the function with input arguments
calculate.FV( 180, 0.055, 3 )
```

```
## Future Value equals $211.36
```

The future value of the loan equals $211.36.

Part b)

Write a function called `calculate.PV()` that takes the future value `FV`, interest rate `r`, and time duration `T` as arguments, and returns the present value of the loan.

Answer

```r
# write function for PV
calculate.PV <- function( FV, r, t ) {
  message( " Present Value equals $", round( FV / ( 1 + r )^t, 2 ) )
}
```

Suppose Elvis borrows some money from Obie at a 4.5% interest rate so that in 4 years the final value will be 298.1297. Using your function `calculate.PV()`, what is the present value of the loan?

```r
# run PV function
calculate.PV( 298.1297, 0.045, 4 )
```

```
##  Present Value equals $250
```

The present value of the loan equals $250.

Part c)

Write a function called `calculate.interest.rate()` that takes the present value `PV`, future value `FV`, and time duration `T` as arguments, and returns the interest rate of the loan.

Answer

```r
# write function for interest rate calculation
calculate.interest.rate <- function( PV, FV, t ) {
  message( "Interest rate equals ", round( ( ( FV / PV -1 )^1 / t ) * 100, 2 ), "%" )
}
```

Suppose Gazelle lends Tom Gravy 1000 dollars today and in 2.5 years the future value will be 1184.294 dollars. Using your function `calculate.interest.rate()`, what is the interest rate that Gazelle is charging Tom?

```r
# run function with input arguments
calculate.interest.rate( 1000, 1184.294, 2.5 )
```

```
## Interest rate equals 7.37%
```

Part d)

Write a function called `calculate.loan.duration()` that takes the present value `PV`, the future value `FV`, and the interest rate `r`, and returns the time duration of the loan.

Answer

```
# write function for duration
calculate.loan.duration <- function( PV, FV, r ) {
  message( "Loan duration equals ", round( ( log( FV ) - log( PV ) ) / log( 1 + r ), 1 ),
           " years" )
}
```

Goldilocks borrows 800 dollars from Baby Bear at 5.5% interest, and when she pays it back it will be worth 1,045.568 dollars. Using your function `calculate.loan.duration()`, how long will it be before she pays back the loan?

```
# run function with input arguments
calculate.loan.duration( 800, 1045.568, 0.055 )
```

```
## Loan duration equals 5 years
```

Problem 4: Missing Value Indices

Write a function named `missing.value.indices()` that takes two arguments:

- First, a vector of any class.
- Second, a named parameter called `print.message`, which has the default value FALSE.

The function `missing.value.indices()` returns the indices of the locations with missing values in the input vector. That is, if there are missing values at positions 2, 5, and 7 in the input vector, then `missing.value.indices()` should return a numeric vector with the values 2, 5, and 7. Also, if the named parameter `print.message` is set to TRUE, then the function will print out a short sentence for each missing value such as "There were a missing value at position 2"

Answer

```
# write function
missing.value.indices <- function( x, print.message = FALSE ) {
  for( i in 1:length( x ) ) {
    if( is.na( x )[i] & print.message == TRUE ) {
       message( "There is a missing value at position ", i)
    } else if( i == length(x) & print.message & all( !is.na( x ) ) == TRUE ) {
      message( "There are no missing values" )
    }
  }
  print( which( is.na( x ) ) )
}
```

Test your function on these vectors:

```
problem.4.test.vector.1 <- c(1, 6, 3, 7, NA, 12, -15, NA, 6, 8, NA)
problem.4.test.vector.2 <- c(FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE)
```

First, call the function on `problem.4.test.vector.1` without printing a message:

```
missing.value.indices( problem.4.test.vector.1 )
```

```
## [1]  5  8 11
```

Next, call the function on `problem.4.test.vector.1` this time printing a message:

```
missing.value.indices( problem.4.test.vector.1, TRUE )
```

```
## There is a missing value at position 5
## There is a missing value at position 8
## There is a missing value at position 11
## [1]  5  8 11
```

Now call the function on `problem.4.test.vector.2` without printing a message:

```
missing.value.indices( problem.4.test.vector.2 )
```

```
## integer(0)
```

Next, call the function on `problem.4.test.vector.2` this time printing a message:

```
missing.value.indices( problem.4.test.vector.2, TRUE )
```

```
## There are no missing values
## integer(0)
```

Problem 5: Missing Value Imputer

Write a function named `impute.missing.values()` that takes a numeric vector as input and returns a vector such that all the missing values of the input vector have been imputed using the vector mean.

Answer

```
# write function
impute.missing.values <- function( x ) {
  vec.mean <- mean( x, na.rm = TRUE )
  if( any( is.na( x ) ) == TRUE ) {
    x[ which( is.na( x ) == TRUE ) ] <- vec.mean
  }
  print( x )
}
```

Let's test your code. First, we'll define a test vector, and check its mean:

```
problem.5.test.vector <- c( 4.3, 2.8, 12.9, NA, -3.5, 8.8, NA, 4.7 )
```

Now run your `impute.missing.values()` on this test vector:

```
impute.missing.values( problem.5.test.vector )
```

```
## [1]  4.3  2.8 12.9  5.0 -3.5  8.8  5.0  4.7
```

Notice that `problem.5.test.vector` remains unchanged:

```
problem.5.test.vector
```

```
## [1]  4.3  2.8 12.9   NA -3.5  8.8   NA  4.7
```

Problem 6: Vector Summarizer

Write a function named `vector.summarizer()` that takes a vector and prints out various summary statistics, depending on the class of the vector.

- For every type of vector, `vector.summarizer()` should print out the class of the vector e.g. "numeric" or "logical".
- If the vector is numeric, then `vector.summarizer()` should print out the mean, median, standard deviation, maximum value, and minimum value.
- If the vector is logical, then `vector.summarizer()` should print out the number of TRUE values, the number of FALSE values, and the number of missing values.

Make your printout messages readable – don't just print the number, but have a complete sentence e.g. "The mean of the vector is 6.7."

Suggestion: it can be nice to indent your summary statistics statements, and you can do this by using a "$\widehat{\ }$" in any strings in `message()`.

Answer

```
# write function
vector.summarizer <- function( x ) {
  print( class( x ) )
  if( class( x ) == "numeric" ) {
    message( "The mean of the vector is ", "\t", "\t", "\t",
             mean( x, na.rm = TRUE ) )
    message( "The median of the vector is ", "\t", "\t", "\t",
             median( x, na.rm = TRUE ) )
    message( "The standard dev. of the vector is ", "\t",
             round( sd(x, na.rm = TRUE ), 2 ) )
    message( "The maximum value of the vector is ", "\t", max( x ) )
    message( "The minimum value of the vector is ", "\t", min( x ) )
  } else if( class( x ) == "logical" ) {
    message( "The total number of TRUE values equals ",  "\t", "\t",
             sum( x, na.rm = TRUE ) )
    message( "The total number of FALSE values equals ", "\t", "\t",
             sum( x == FALSE, na.rm = TRUE ) )
    message( "The total number of missing values equals ", "\t",
             sum( is.na( x ) ) )
  } else {
    summary( x )
  }
}
```

Test your function by running these test cases, and make sure that you get the right answer:

```
vector.summarizer( c(4, 6, 7, 2, -8, 6, -12, 25 ) )
```

```
## [1] "numeric"
## The mean of the vector is            3.75
## The median of the vector is          5
## The standard dev. of the vector is   11.09
## The maximum value of the vector is   25
## The minimum value of the vector is   -12
```

```
vector.summarizer( c( TRUE, TRUE, TRUE, FALSE, NA, FALSE, FALSE, TRUE))
```

```
## [1] "logical"
## The total number of TRUE values equals       4
## The total number of FALSE values equals      3
## The total number of missing values equals    1
```

Problem 7: Data Frame Summarizer

Write a function named `data.frame.summarizer()` that takes a data frame as its input argument and prints out a summary of each variable. Use your `summarizer()` function that you built in Problem 7 to do the actual work of printing out any messages. HINT: use a `for` loop across the columns.

Answer

```
# write function
data.frame.summarizer <- function( x ) {

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

    message( "The class of variable ", i, " is " , class( x[, i] ), " \n" )

    if( class( x[, i] ) == "numeric" ) {
      message( "The mean is ", "\t", "\t", "\t", "\t",
               round( mean( x[, i], na.rm = TRUE ), 2 ) )
      message( "The median is ", "\t", "\t", "\t",
               round( median( x[, i], na.rm = TRUE ), 2 ) )
      message( "The standard dev. is ", "\t",
               round( sd(x[, i], na.rm = TRUE ), 2 ) )
      message( "The maximum value is ", "\t",
               round( max( x[, i] ), 2 ) )
      message( "The minimum value is ", "\t",
               round( min( x[, i] ), 2 ), " \n" )
    }
    else if( class( x[, i] ) == "logical" ) {
      message( "The total number of TRUE values equals ",  "\t", "\t",
               sum( x[, i], na.rm = TRUE ) )
      message( "The total number of FALSE values equals ", "\t", "\t",
               length( x[, i] ) - sum ( x[, i], na.rm = TRUE ) )
      message( "The total number of missing values equals ", "\t",
               sum( is.na( x[, i] ) ), " \n" )
    }
    else {
      message( "Summary:" )
      print( summary( x[, i] ) )
    }
  }
}
```

Test your function on the built-in data frame `iris`:

```
data.frame.summarizer( iris )
```

```
## The class of variable 1 is numeric
## The mean is              5.84
## The median is            5.8
## The standard dev. is     0.83
## The maximum value is     7.9
## The minimum value is     4.3
## The class of variable 2 is numeric
## The mean is              3.06
## The median is            3
## The standard dev. is     0.44
## The maximum value is     4.4
## The minimum value is     2
## The class of variable 3 is numeric
## The mean is              3.76
## The median is            4.35
## The standard dev. is     1.77
## The maximum value is     6.9
## The minimum value is     1
## The class of variable 4 is numeric
## The mean is              1.2
## The median is            1.3
## The standard dev. is     0.76
## The maximum value is     2.5
## The minimum value is     0.1
## The class of variable 5 is factor
## Summary:
##     setosa versicolor  virginica
##         50         50         50
```

Problem 8: Improved Data Frame Summarizer

The data frame summarizer that we built in Problem 7 is nice, but we can make it even better. Now write a new function named `improved.data.frame.summarizer()` that does these things:

- First, it prints out the index of the column, along with the column name. So, the first line might be something like "Column 1 has name: ID".
- Next, you should print out the number of non-missing and missing values for the column.

- Print out the class of the column vector.
- If the column vector is numeric, print out lines for the mean, median, standard deviation, min, and max.
- If the column vector is logical, print out the proportion of non-missing values that are TRUE.
- Finally, at the end, put in an extra carriage return, so there is a space between the results for each column.

You won't be able to use the function that you defined in Problem 6. But you're welcome to copy and paste anything that you wrote for that problem, and then to modify it.

Answer

```
# write function
improved.data.frame.summarizer <- function( x ) {

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

    message( "\n" )
    message( "Column ", i, " has name ", names( x[i] ) )
    message( "Column ", i, " has ", sum( is.na( x[, i] ) ), " missing values" )
    message( "Column ", i, " has ", sum( !is.na( x[, i] ) ),
             " non-missing values" )
    message( "Column ", i, " class is " , class( x[, i] ) )
    message( "Column ", i, " summary:" )

    if( class( x[, i] ) == "numeric" ) {
      message( "The mean is ", "\t", "\t", "\t", "\t",
               round( mean( x[, i], na.rm = TRUE ) ) )
      message( "The median is ", "\t", "\t", "\t",
               median( x[, i], na.rm = TRUE ) )
      message( "The standard dev. is ", "\t",
               round( sd(x[, i], na.rm = TRUE ), 2 ) )
      message( "The maximum value is ", "\t", max( x[, i] ) )
      message( "The minimum value is ", "\t", min( x[, i] ) )
    }
    else if( class( x[, i] ) == "logical" ) {
      message( "The total number of TRUE values equals ",  "\t", "\t",
               sum( x[, i], na.rm = TRUE ) )
      message( "The proportion of TRUE values equals ",
               sum( x[, i], na.rm = TRUE ) / length( x[, i] ) )
      message( "The total number of FALSE values equals ", "\t",
               length( x[, i] ) - sum ( x[, i], na.rm = TRUE ) )
      message( "The proportion of FALSE values equals ",
               sum( x[, i], na.rm = FALSE ) / length( x[, i] ) )
      message( "The total number of missing values equals ", "\t",
               sum( is.na( x[, i] ) ) )
    }
    else {
      print( summary( x[, i] ) )
    }
  }
}
```

Test your function on the built-in data frame `iris`:

```
improved.data.frame.summarizer( iris )
```

```
##
## Column 1 has name Sepal.Length
## Column 1 has 0 missing values
## Column 1 has 150 non-missing values
## Column 1 class is numeric
## Column 1 summary:
## The mean is              6
## The median is            5.8
## The standard dev. is     0.83
## The maximum value is     7.9
## The minimum value is     4.3
##
## Column 2 has name Sepal.Width
## Column 2 has 0 missing values
## Column 2 has 150 non-missing values
## Column 2 class is numeric
## Column 2 summary:
## The mean is              3
## The median is            3
## The standard dev. is     0.44
## The maximum value is     4.4
## The minimum value is     2
##
## Column 3 has name Petal.Length
## Column 3 has 0 missing values
## Column 3 has 150 non-missing values
## Column 3 class is numeric
## Column 3 summary:
## The mean is              4
## The median is            4.35
## The standard dev. is     1.77
## The maximum value is     6.9
## The minimum value is     1
##
## Column 4 has name Petal.Width
```

```
## Column 4 has 0 missing values
## Column 4 has 150 non-missing values
## Column 4 class is numeric
## Column 4 summary:
## The mean is              1
## The median is            1.3
## The standard dev. is     0.76
## The maximum value is     2.5
## The minimum value is     0.1
##
## Column 5 has name Species
## Column 5 has 0 missing values
## Column 5 has 150 non-missing values
## Column 5 class is factor
## Column 5 summary:
##     setosa versicolor  virginica
##         50         50         50
```