Problem Set 5

Tim de Rooij

Instructions

First, save this notebook with a title consisting of "ProblemSet5" and then your last name, with no spaces. So, for Bob Bunny, the file should be called "ProblemSet5Bunny". 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: Weird Missing Values, Part 1

Sometimes unusual values are used to represent missing data. R can handle missing data only if it represented as "NA", so any alternative representation will mean that R cannot use its mechanisms for dealing with missingness. Thus, if you encounter a dataset that uses these alternative representations, you will first have to change these into "NA" before you can work with the data.

I personally like to encode missing data using the string "Missing". However, this means that any vector that has missing data will be cast as a `character` vector, even if most of the values are numeric or logical. Write a function called `change.missing()` that takes two arguments, a vector of values named `input.vector`, and a parameter named `vector.class`, and returns a new vector with any "Missing" values changed to "NA", and with the class specified by the `vector.class` parameter. For this problem, you only have to consider two possibilities for the `vector.class` parameter: `numeric` and `logical`. Thus, suppose you make this function call:

> change.missing(x.vector, vector.class = "numeric")

Then the function will make a new vector of class `numeric` in which any elements in the input vector had the value "Missing" are now "NA". Similarly, suppose you make this function call:

> change.missing(x.vector, vector.class = "logical")

Then the function will make a new vector of class `logical` in which any elements in the input vector had the value "Missing" are now "NA".

Hint: you might find the functions `as.numeric()` and `as.logical()` to be very useful.

Answer

```
# create function
change.missing <- function( input.vector, vector.class ) {

   input.vector[ input.vector == "Missing" ] <- NA

   if( vector.class == "numeric" ) {
    as.numeric( input.vector )
```

```
  } else {
    as.logical( input.vector )
  }
}
```

Test your function on these vectors:

```
change.missing(
  c(5, 3, 4, 9, 7, "Missing", 1, "Missing"), vector.class = "numeric" )
```

```
## [1]  5  3  4  9  7 NA  1 NA
```

```
change.missing(
  c( TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, TRUE, "Missing", "Missing" ),
  vector.class = "logical"
)
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE    NA    NA
```

Problem 2: Weird Missing Values, Part 2

Another strategy that many people use is to encode missing data using a numeric value such as -9 or -99. Personally, I think this is the worst thing that you can do, and I strongly discourage it. Nonetheless this most unfortunate practice is all too common, and you might have to deal with it.

Write a function called `change.9()` that takes two arguments, a vector of values named `input.vector`, and a parameter named `vector.class`, and returns a new vector with any -9 values changed to "NA", and with the class specified by the `vector.class` parameter. For this problem, you only have to consider two possibilities for the `vector.class` parameter: `numeric` and `logical`. Thus, suppose you make this function call:

> change.9(x.vector, vector.class = "numeric")

Then the function will make a new vector of class `numeric` in which any elements in the input vector had the value "Missing" are now "NA". Similarly, suppose you make this function call:

> change.9(x.vector, vector.class = "logical")

Then the function will make a new vector of class `logical` in which any elements in the input vector had the value "Missing" are now "NA".

Hint: you might find the functions `as.numeric()` and `as.logical()` to be very useful.

Answer

```
# write function
change.9 <- function( input.vector, vector.class ) {

   input.vector[ input.vector == -9 ] <- NA

   if( vector.class == "numeric" ) {
    as.numeric( input.vector )
  } else {
    as.logical( input.vector )
  }
}
```

Test your function on these vectors:

```
change.9(
  c(5, 3, 4, 9, 7, -9, 1, -9), vector.class = "numeric" )
```

```
## [1]  5  3  4  9  7 NA  1 NA
```

```
change.9(
  c( TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, TRUE, -9, -9 ),
  vector.class = "logical"
)
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE    NA    NA
```

Problem 3: Extracting rows

In lecture we saw how to extract rows from a data frame using the strange notation with a row index followed by a comma followed by nothing. For instance, this will extract the seventh row from a data frame:

my.data.frame[7,]

We can use all the standard techniques for indexing that we saw for vectors. In particular, we can create a logical vector using some sort of condition, and then filter the rows based on that condition. This problem explores that technique.

First, run this code to create a data frame:

```
problem.3.data.frame <- data.frame(
  species = c(2, 2, 1, 2, 1, 1, 1, 2, 1, 2),
  bench.press = c(13.3, 19.8, 5.6, 10.6, 6.8, 7.2, 10.4, 12.0, 8.7, 14.4),
  mile.run = c(14.4, 10.8, 16.6, 15.3, 15.2, 15.8, 15.0, 13.6, 12.5, 16.7)
)

problem.3.data.frame
```

```
##    species bench.press mile.run
## 1        2        13.3     14.4
## 2        2        19.8     10.8
## 3        1         5.6     16.6
## 4        2        10.6     15.3
## 5        1         6.8     15.2
## 6        1         7.2     15.8
## 7        1        10.4     15.0
## 8        2        12.0     13.6
## 9        1         8.7     12.5
## 10       2        14.4     16.7
```

Part a)

Now create a new data frame by selecting the rows for hedgehogs (species = 1); call this data frame `problem.3.hedgehog`:

Answer

```
# select rows that satisfy logical test
problem.3.hedgehog <- data.frame(
  problem.3.data.frame[ problem.3.data.frame[, 1 ] == 1, ] )
```

Let's display the data so Karen can be sure you got the right rows:

```
problem.3.hedgehog
```

```
##   species bench.press mile.run
## 3       1         5.6     16.6
## 5       1         6.8     15.2
## 6       1         7.2     15.8
## 7       1        10.4     15.0
## 9       1         8.7     12.5
```

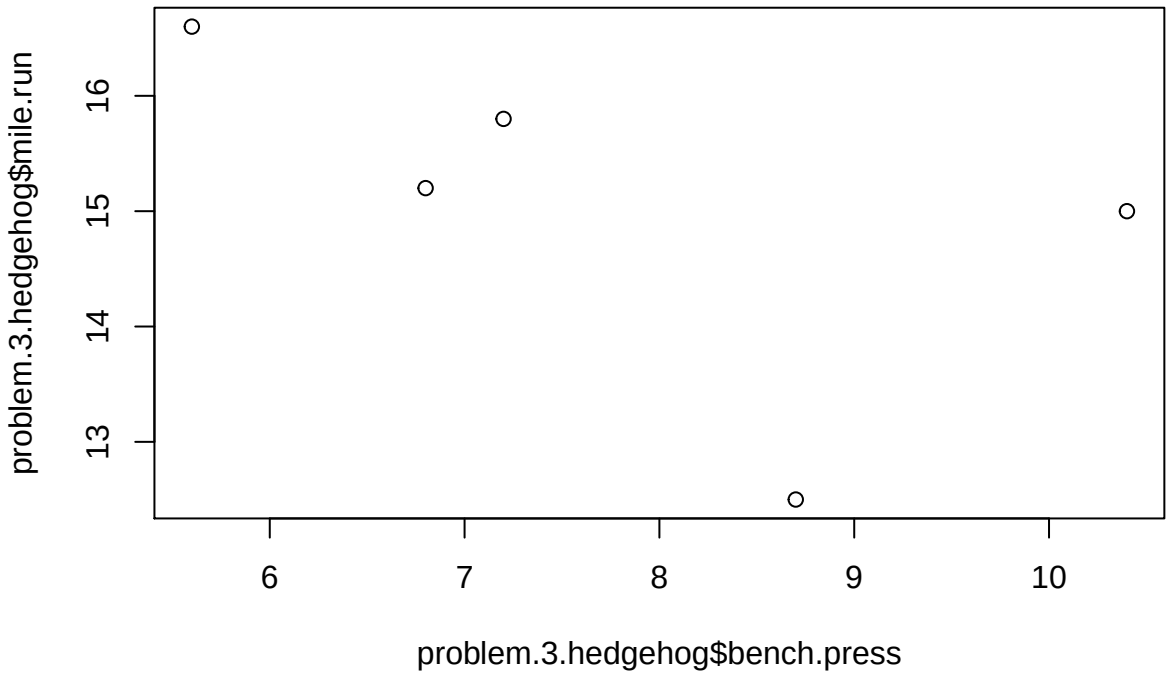
Part b)

What is the correlation between bench press weight and mile run time, for hedgehogs only?

Answer

When we visualize the data there seems to be no obvious relationship.

```
# visualise the data
plot( x = problem.3.hedgehog$bench.press, y = problem.3.hedgehog$mile.run )
```



When we run a correlation test we can observe that the data suggest a negative correlation of -0.5641379 between bench press weight (x) and mile run time (y). This suggests that hedgehogs that can press more weight have lower mile run times (i.e. run faster). However, The p-value is very large (p-value = 0.3219), indicating low significance. Therefore, the suggested correlation can also be caused by randomness instead of an existing relationship between these two variables.

```
# run cor.test() function to test for correlation
cor.test( x = problem.3.hedgehog$bench.press, y = problem.3.hedgehog$mile.run )
```

```
##
##  Pearson's product-moment correlation
##
## data:  problem.3.hedgehog$bench.press and problem.3.hedgehog$mile.run
## t = -1.1834, df = 3, p-value = 0.3219
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9657375  0.6333688
## sample estimates:
##        cor
## -0.5641379
```

Part c)

Now create a new data frame by selecting the rows for bunny rabbits (species = 2); call this data frame problem.3.bunny:

Answer

```
# select rows that satisfy logical test
problem.3.bunny <- data.frame(
  problem.3.data.frame[ problem.3.data.frame[, 1 ] == 2, ] )
```

Display your result so Karen can be sure you got the right result:

```
problem.3.bunny
```

```
##    species bench.press mile.run
## 1        2        13.3     14.4
## 2        2        19.8     10.8
## 4        2        10.6     15.3
## 8        2        12.0     13.6
## 10       2        14.4     16.7
```

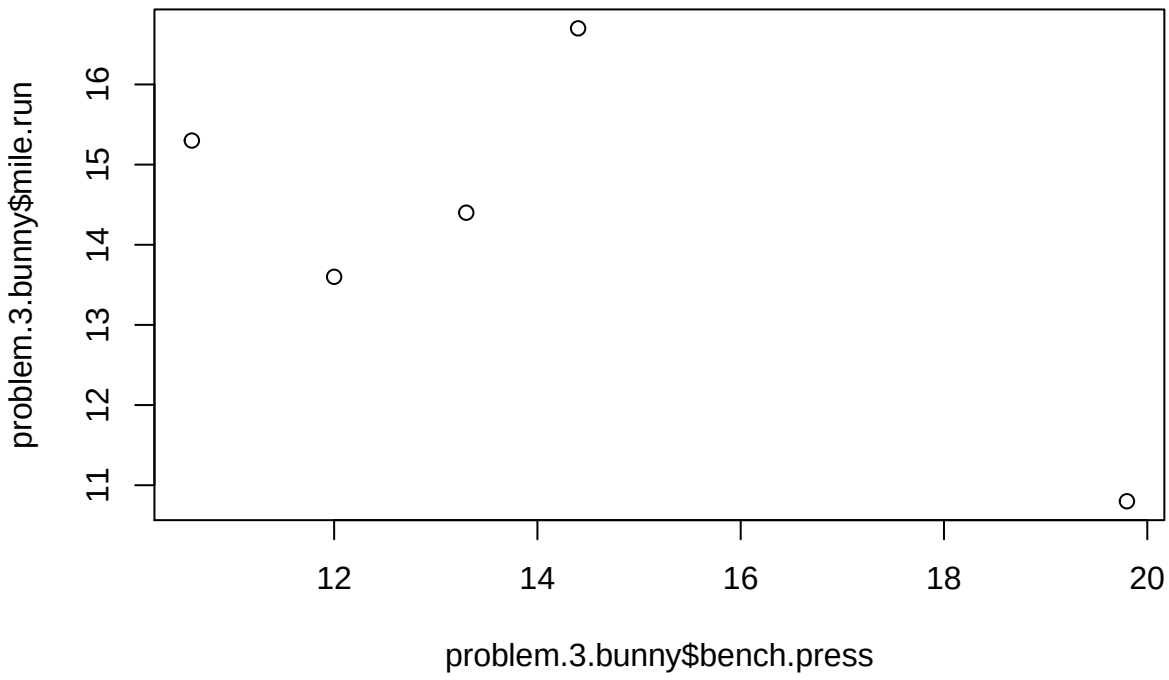
Part d)

What is the correlation between bench press weight and mile run time, for bunnies only?

Answer

When we plot bench press weight and mile run time there does not seem to be an obvious relationship.

```
# visualise the data
plot( x = problem.3.bunny$bench.press, y = problem.3.bunny$mile.run )
```



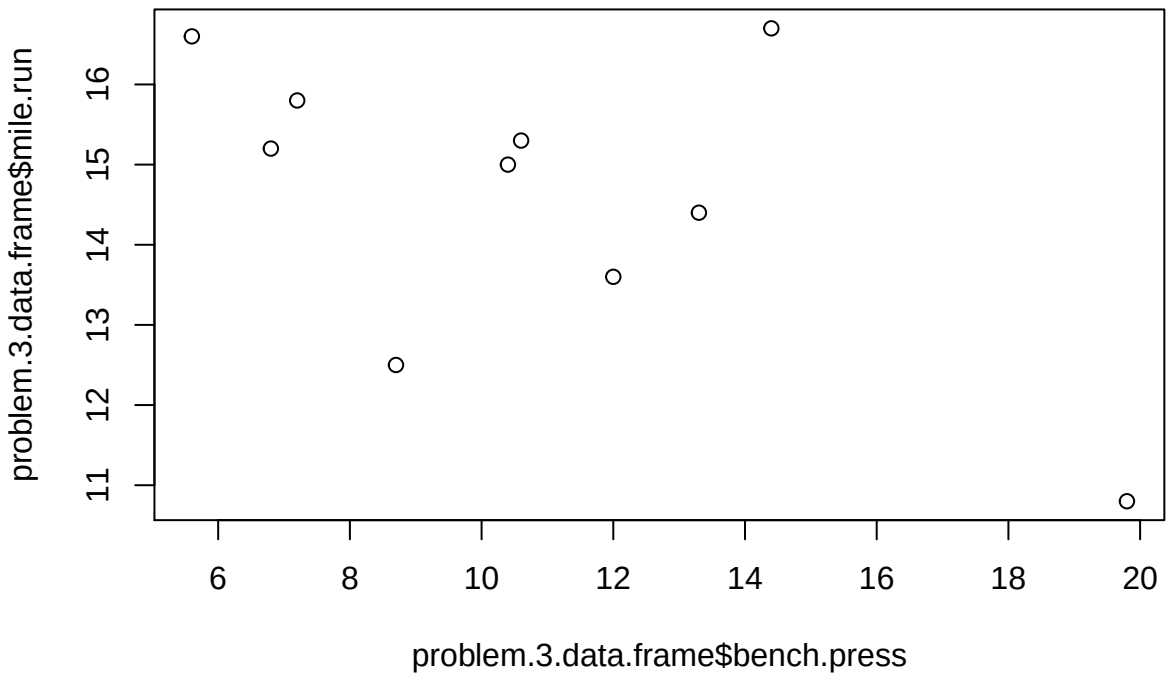
When we run a correlation test, we get a negative correlation of -0.6878181. Again, the results are insignificant at a 95% confidence level (p-value is greater than 0.05).

```
# run cor.test() function to test for correlation
cor.test( x = problem.3.bunny$bench.press, y = problem.3.bunny$mile.run )
```

```
##
##  Pearson's product-moment correlation
##
## data:  problem.3.bunny$bench.press and problem.3.bunny$mile.run
## t = -1.6412, df = 3, p-value = 0.1993
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9771263  0.4945767
## sample estimates:
##        cor
## -0.6878181
```

When we test on the entire set, we observe a negative correlation (in line with expectations) at a smaller value for p: p = 0.07, which would be significant at a 90% confidence level.

```
plot( y = problem.3.data.frame$mile.run, x = problem.3.data.frame$bench.press )
```




```
cor.test( y = problem.3.data.frame$mile.run, x = problem.3.data.frame$bench.press )
```

```
##
##  Pearson's product-moment correlation
##
## data:  problem.3.data.frame$bench.press and problem.3.data.frame$mile.run
## t = -2.0423, df = 8, p-value = 0.0754
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.88780000  0.07003644
## sample estimates:
##        cor
## -0.5854043
```

Problem 4: Quadratic Equations, One Last Time

In this problem, we'll see why it's useful to have the list data structure. We won't use lists very much in our course, but I want you to have some sense of how they work, and what they're good for.

First, recall that a *quadratic equation* is an equation of the form:

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

We know from high-school algebra that one solution of the quadratic equation is:

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

The other solution has the form:

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

We also know that the expression under the square root sign tells us about the number of solutions:

- 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.

For this problem, write an R function named `quadratic.solver.3` that takes the three coefficients a, b, and c of the quadratic equation and returns a **list** consisting of two values:

- The first value is a string, which is either "2 real solutions", "1 real solution", or "No real solutions", depending on the value of $b^2 - 4ac$.
- The second value is vector consisting of all the solutions to the equation; this vector will have 2, 1, or 0 elements in it, depending on how many distinct real solutions of the equation there are.

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.3(2, -10, 8)`, and it should return a list consisting of the string "2 real roots" and a vector `(4,1)`.

To be clear: don't print out any messages. Instead, the whole point of this problem is for you to write a function that returns a list, and the list contains a rich data structure that encodes a complex set of information.

Answer

```
quadratic.solver.3 <- function( a, b, c ) {

  if( b^2 > 4 * a * c) {
    output.string <- "2 real solutions"
    x1 <- ( -b + sqrt( b^2 - 4 * a * c ) ) / ( 2 * a )
    x2 <- ( -b - sqrt( b^2 - 4 * a *  c ) ) / ( 2 * a )
    output.list <- list( output.string, c( x1, x2 ) )
  }
  else if( b^2 < 4 * a * c) {
    output.string <- "No real solutions"
    x1 <- as.numeric( 0 )
    output.list <- list( output.string, c( x1 ) )
  }
  else {
    output.string <- "1 real solution"
    x1 <- ( -b / ( 2 * a ) )
    output.list <- list( output.string, c( x1 ) )
  }
  output.list
}
```

Test your function on the equation $2x^2 - 10x + 8$:

```
quadratic.solver.3(2, -10, 8)
```

```
## [[1]]
## [1] "2 real solutions"
##
## [[2]]
```

```
## [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.3()` function on this equation, and see if you get those values:

```
quadratic.solver.3(1, -3, -10)
```

```
## [[1]]
## [1] "2 real solutions"
##
## [[2]]
## [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.3()` on this equation?

```
quadratic.solver.3(1, -6, 9)
```

```
## [[1]]
## [1] "1 real solution"
##
## [[2]]
## [1] 3
```

```
( -6 )^2 < 1 * 4 * 9
```

```
## [1] FALSE
```

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.3()` on this equation?

```
quadratic.solver.3( 1, -3.5, -30)
```

```
## [[1]]
## [1] "2 real solutions"
##
## [[2]]
## [1]  7.5 -4.0
```

Next, try this equation:

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

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

```
quadratic.solver.3( 1, -8, 20 )
```

```
## [[1]]
## [1] "No real solutions"
##
## [[2]]
## [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.3()`:

```
quadratic.solver.3(1, 0, -25)
```

```
## [[1]]
## [1] "2 real solutions"
##
## [[2]]
## [1]  5 -5
```

Problem 5: Creating New Columns in a Data Frame

I mentioned in lecture that it was easy to create new columns in a data frame: all you have to do is to reference the column that you wished existed, and then R will create it for you. In this problem we're going to explore this by building a table of trigonometric function values. You don't have to remember anything about trigonometry to do this problem; all you need to know is that the sine function is `sin()` and the cosine function is `cos()`, and that the arguments for both of these functions have to be in what is called *radian* measure, which I'll calculate for you.

First, run this code to create the data frame:

```
problem.5.degrees.vector <- seq(0, 90, 10)
problem.5.radians.vector <- problem.5.degrees.vector * 2 * pi / 360
problem.5.trig.table <- data.frame(
  degrees = problem.5.degrees.vector,
  radians = problem.5.radians.vector
)

problem.5.trig.table
```

```
##    degrees   radians
## 1        0 0.0000000
## 2       10 0.1745329
## 3       20 0.3490659
## 4       30 0.5235988
## 5       40 0.6981317
## 6       50 0.8726646
## 7       60 1.0471976
## 8       70 1.2217305
## 9       80 1.3962634
## 10      90 1.5707963
```

Part a)

Now create a new column named `sine` that consists of the sine values of the `radians` variable. In other words, for the first row, the radian measure is 0, so the first row for the `sine` column should be the value of the sine function for 0 radians.

Answer

```
problem.5.trig.table$sine <- sin( problem.5.trig.table$radians )
```

Display your table so Karen can check it:

```
problem.5.trig.table
```

```
##    degrees   radians      sine
## 1        0 0.0000000 0.0000000
## 2       10 0.1745329 0.1736482
## 3       20 0.3490659 0.3420201
## 4       30 0.5235988 0.5000000
## 5       40 0.6981317 0.6427876
## 6       50 0.8726646 0.7660444
## 7       60 1.0471976 0.8660254
## 8       70 1.2217305 0.9396926
## 9       80 1.3962634 0.9848078
## 10      90 1.5707963 1.0000000
```

Part b)

Now add a new column named `cosine` to the data frame consisting of the values of the cosine of the `radian` variable. In other words, for the first row, the radian measure is 0, so the first row for the `cosine` column should be the value of the cosine function for 0 radians.

Answer

See table output below.

```
problem.5.trig.table$cosine <- cos( problem.5.trig.table$radians )
problem.5.trig.table
```

```
##    degrees   radians      sine       cosine
## 1        0 0.0000000 0.0000000 1.000000e+00
## 2       10 0.1745329 0.1736482 9.848078e-01
## 3       20 0.3490659 0.3420201 9.396926e-01
## 4       30 0.5235988 0.5000000 8.660254e-01
## 5       40 0.6981317 0.6427876 7.660444e-01
## 6       50 0.8726646 0.7660444 6.427876e-01
## 7       60 1.0471976 0.8660254 5.000000e-01
## 8       70 1.2217305 0.9396926 3.420201e-01
## 9       80 1.3962634 0.9848078 1.736482e-01
## 10      90 1.5707963 1.0000000 6.123234e-17
```

Problem 6: Formatting Strings

This problem will familiarize you with the string functions `formatC()` and `sprintf()`.

Start by running this code:

```
problem.6.number.1 <- exp( 3.8 )
problem.6.number.2 <- exp( 12.4 )
problem.6.number.3 <- exp( -13.8 )
```

Part a)

Use `formatC()` to create a string so that each number has two decimal places and the string has a total width of 7. Note: you'll need to specify some named parameters for this problem; don't forget to set `format = "f"`.

Answer

First, for `problem.6.number.1`:

```
formatC( problem.6.number.1, digits = 2, width = 7, format = "f" )
```

```
## [1] "  44.70"
```

Now for 'problem.6.number.2':

```
formatC( problem.6.number.2, digits = 2, width = 7, format = "f" )
```

```
## [1] "242801.62"
```

Now for 'problem.6.number.3':

```
formatC( problem.6.number.3, digits = 2, width = 7, format = "f" )
```

```
## [1] "   0.00"
```

Part b)

Now format the numbers with 8 decimal places. You don't specify the width of the string, so you can let R determine that.

Answer

First, for `problem.6.number.1`:

```
formatC( problem.6.number.1, digits = 8, format = "f" )
```

```
## [1] "44.70118449"
```

Now for 'problem.6.number.2':

```
formatC( problem.6.number.2, digits = 8, format = "f" )
```

```
## [1] "242801.61749832"
```

Now for 'problem.6.number.3':

```
formatC( problem.6.number.3, digits = 8, format = "f" )
```

```
## [1] "0.00000102"
```

Part c)

Now we're going to print out a readable message. For each number, create a string that starts with the message "Number X is:", followed by the number with 4 decimal places. Format the number using `formatC()`, and use either `cat()` or `paste()` to combine everything into one string. Make sure that there is just one space between the colon in "Number X is:" and the number.

Answer

First, for `problem.6.number.1`:

```
paste( "Number X is:", formatC( problem.6.number.1,
                                digits = 4, format = "f" ) )
```

```
## [1] "Number X is: 44.7012"
```

Now for 'problem.6.number.2':

```
paste( "Number X is:", formatC( problem.6.number.2,
                                digits = 4, format = "f" ) )
```

```
## [1] "Number X is: 242801.6175"
```

Now for 'problem.6.number.3':

```
paste( "Number X is:", formatC( problem.6.number.3,
                                digits = 4, format = "f" ) )
```

```
## [1] "Number X is: 0.0000"
```

Part d)

Now we're going to print out the numbers using `sprintf()`. For each number, use `sprintf()` to create a string that starts with the message "Number X is:", followed by the number with 4 decimal places.

Answer

First, for `problem.6.number.1`:

```
sprintf( "Number X is: %.4f", problem.6.number.1 )
```

```
## [1] "Number X is: 44.7012"
```

Now for 'problem.6.number.2':

```
sprintf( "Number X is: %.4f", problem.6.number.2 )
```

```
## [1] "Number X is: 242801.6175"
```

Now for 'problem.6.number.3':

```
sprintf( "Number X is: %.4f", problem.6.number.3 )
```

```
## [1] "Number X is: 0.0000"
```

Problem 7: Cleaning Data

In this problem, you'll get some practice with some basic data cleaning. Don't try to write general functions to solve these problems, but instead fix each problem individually.

Part a)

Oh no! The computer that your team uses for data entry has a weird mechanical malfunction – occasionally, when a digit is typed, the computer will repeat the digit an arbitrary number of timmes. For instance, if a '3' was typed, then the computer might actually end up storing the value '3333'.

You have been supplied with a set of data for a score which you know must be an integer between 0 and 9, but was entered using the malfunctioning computer:

```
bad.numeric.vector <- c( 6, 3, 222, 888, 4, 55, 2, 4, 3, 77)
```

Caclulate the correct mean of these values, manually fixing any problems that might have been caused by the malfunction.

Answer

One manual cleaning method is to substitute the erroneous values individually by the correct values.

```
bad.numeric.vector
```

```
##  [1]   6   3 222 888   4  55   2   4   3  77
```

```
bad.numeric.vector[3] <- 2
bad.numeric.vector[4] <- 8
bad.numeric.vector[6] <- 5
bad.numeric.vector[10] <- 7
bad.numeric.vector
```

```
##  [1] 6 3 2 8 4 5 2 4 3 7
```

We can then calculate the mean of the data in the cleaned vector.

```
mean( bad.numeric.vector )
```

```
## [1] 4.4
```

Part b)

It also turns out that the computer sometimes stores alphabetic characters with the incorrect case. Thus, if someone typed "bunny rabbit", the computer might store "buNnY RabBIt". Note that in this problem, the only source of error is the case. Fix the mistakes in this vector:

```
bad.character.vector <-
  c( "hedgehog", "BUnNY raBbiT", "BunNy rAbbIt", "HEdGehoG", "bunnY RaBbiT",
     "HEDGEhOg", "hedgehog", "bunnY RabBit", "hEdGeHoG", "bUnnY RabBit" )
```

Answer

Using the tolower() function to convert elements in vector to lowercase.

```
bad.character.vector <- tolower( bad.character.vector )
```

Display the vector so Karen can be sure you did it right:

```
bad.character.vector
```

```
##  [1] "hedgehog"     "bunny rabbit" "bunny rabbit" "hedgehog"
##  [5] "bunny rabbit" "hedgehog"     "hedgehog"     "bunny rabbit"
##  [9] "hedgehog"     "bunny rabbit"
```

Here's a little tip: the function `unique()` takes a vector and returns a vector consisting of just the unique values, so for this problem if you run `unique()` on your answer you should get a vector with precisely two values.

```
unique( bad.character.vector )
```

```
## [1] "hedgehog"     "bunny rabbit"
```

Part c)

The crazy computer has been fixed! But now you have some data on the average number of hours of sleep for 10 subjects:

```
sleep.vector <- c( 7.5, "I think 8", 8, 9, "Usually 8", 6.5, "Always 7",
                   "I'm pretty sure it's 9", 8.5, "On average 7" )
```

Calculate the mean of the hours of sleep for the 10 subjects.

Hint: in order to use the `mean()` function, the vector must be numeric.

Answer

The mean hours of sleep for the 10 subjects is 7.85.

```
# assess the vector values
sleep.vector
```

```
##  [1] "7.5"                   "I think 8"
##  [3] "8"                     "9"
##  [5] "Usually 8"             "6.5"
##  [7] "Always 7"              "I'm pretty sure it's 9"
##  [9] "8.5"                   "On average 7"
```

```
# correct the input values
sleep.vector[2] <- 8
sleep.vector[5] <- 8
sleep.vector[7] <- 7
sleep.vector[8] <- 9
sleep.vector[10] <- 7

# convert vector to numeric and calculate the mean
mean( as.numeric( sleep.vector ) )
```

```
## [1] 7.85
```

Problem 8: Putting It All Together

Now we're going to put these skills together. Your job is to calculate the correlation between bench press weight and mile run time for the entire group, and then perform a subgroup for hedgehogs and bunny rabbits individually. Unfortunately, the data entry team had to use a malfunctioning computer, so the data might not be perfectly clean, and there might be various kinds of mistakes. You'll have to make some judgement calls, so report them when you write up your solution.

The specs for the dataset are:

- The first column specifies which species the subject is. There are only two species in the data set: hedgehogs and bunny rabbits.
- The bench press data and the mile run data are numeric. The values -99 or "Missing" were used by the data entry team to represent missing data.

First, run this code to create the dataset:

```
problem.8.data.frame <- data.frame(
  species = c( "hedgeHog", "BunnY rABBiT", "buNY RaBBiT", "Hedghog",
               "bunny rabbbit", "hedGeHOG", "buNNY Rabbt", "hedgehog",
               "hEdghog", "bunny rabbit"),
```

```
  bench.press = c( 145, 58, 67, -99, 84, 123, 95, 148, 145555555, 74),
  mile.run = c(18.8, 8.3, 7.5, 15.2, 6.9, 12.3, "Missing", 13.7, 14.5, 7.2),
  stringsAsFactors = FALSE
)

problem.8.data.frame
```

```
##          species bench.press mile.run
## 1       hedgeHog         145     18.8
## 2   BunnY rABBiT          58      8.3
## 3    buNY RaBBiT          67      7.5
## 4        Hedghog         -99     15.2
## 5  bunny rabbbit          84      6.9
## 6       hedGeH0G         123     12.3
## 7    buNNY Rabbt          95  Missing
## 8       hedgehog         148     13.7
## 9        hEdghog   145555555     14.5
## 10  bunny rabbit          74      7.2
```

Part a)

Clean up any mistakes in `problem.8.data.frame` so that it's ready for analysis.

```
# convert all strings in species column to lowercase
problem.8.data.frame$species <- tolower( problem.8.data.frame$species )
problem.8.data.frame
```

```
##          species bench.press mile.run
## 1       hedgehog         145     18.8
## 2   bunny rabbit          58      8.3
## 3    buny rabbit          67      7.5
## 4        hedghog         -99     15.2
## 5  bunny rabbbit          84      6.9
## 6       hedgeh0g         123     12.3
## 7    bunny rabbt          95  Missing
## 8       hedgehog         148     13.7
## 9        hedghog   145555555     14.5
## 10  bunny rabbit          74      7.2
```

```
# correct spelling mistakes
problem.8.data.frame$species
```

```
##  [1] "hedgehog"      "bunny rabbit"  "buny rabbit"   "hedghog"
##  [5] "bunny rabbbit" "hedgeh0g"      "bunny rabbt"   "hedgehog"
##  [9] "hedghog"       "bunny rabbit"
```

```
unique( problem.8.data.frame$species )
```

```
## [1] "hedgehog"      "bunny rabbit"  "buny rabbit"   "hedghog"
## [5] "bunny rabbbit" "hedgeh0g"      "bunny rabbt"
```

```
problem.8.data.frame$species[3] <- "bunny rabbit"
problem.8.data.frame$species[4] <- "hedgehog"
problem.8.data.frame$species[5] <- "bunny rabbit"
problem.8.data.frame$species[6] <- "hedgehog"
problem.8.data.frame$species[7] <- "bunny rabbit"
```

```
problem.8.data.frame$species[9] <- "hedgehog"
unique( problem.8.data.frame$species )
```

```
## [1] "hedgehog"     "bunny rabbit"
```

```
# change erroneous values in bench.press column
problem.8.data.frame$bench.press
```

```
##  [1]       145        58        67       -99        84       123        95
##  [8]       148 145555555        74
```

```
problem.8.data.frame$bench.press[4] <- NA
problem.8.data.frame$bench.press[9] <- 145
problem.8.data.frame
```

```
##         species bench.press mile.run
## 1      hedgehog         145     18.8
## 2  bunny rabbit          58      8.3
## 3  bunny rabbit          67      7.5
## 4      hedgehog          NA     15.2
## 5  bunny rabbit          84      6.9
## 6      hedgehog         123     12.3
## 7  bunny rabbit          95  Missing
## 8      hedgehog         148     13.7
## 9      hedgehog         145     14.5
## 10 bunny rabbit          74      7.2
```

```
# change "Missing" in mile.run column to NA
problem.8.data.frame$mile.run
```

```
##  [1] "18.8"    "8.3"     "7.5"     "15.2"    "6.9"     "12.3"    "Missing"
##  [8] "13.7"    "14.5"    "7.2"
```

```
problem.8.data.frame$mile.run[7]  <- NA
problem.8.data.frame
```

```
##         species bench.press mile.run
## 1      hedgehog         145     18.8
## 2  bunny rabbit          58      8.3
## 3  bunny rabbit          67      7.5
## 4      hedgehog          NA     15.2
## 5  bunny rabbit          84      6.9
## 6      hedgehog         123     12.3
## 7  bunny rabbit          95     <NA>
## 8      hedgehog         148     13.7
## 9      hedgehog         145     14.5
## 10 bunny rabbit          74      7.2
```

```
# convert columns class to numeric
str( problem.8.data.frame )
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ species    : chr  "hedgehog" "bunny rabbit" "bunny rabbit" "hedgehog" ...
##  $ bench.press: num  145 58 67 NA 84 123 95 148 145 74
##  $ mile.run   : chr  "18.8" "8.3" "7.5" "15.2" ...
```

```
problem.8.data.frame$mile.run <- as.numeric( problem.8.data.frame$mile.run )
str( problem.8.data.frame )
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ species    : chr  "hedgehog" "bunny rabbit" "bunny rabbit" "hedgehog" ...
##  $ bench.press: num  145 58 67 NA 84 123 95 148 145 74
##  $ mile.run   : num  18.8 8.3 7.5 15.2 6.9 12.3 NA 13.7 14.5 7.2
```

I took the following steps to clean the data:

- convert species column to lowercase
- correct spelling mistakes in species names
- correct benchpress values
- changed -99 to NA
- changed 145555555 to 145
- Replaced "Missing" in milerun column by NA
- Note: another option is to substitute by mean of the column but given the small sample size I would argue it is better to use NA
- convert column class of mile.run from character to numeric

Part b)

Calculate the overall correlation between bench press weight and mile run time for the entire study cohort. Note: in order to handle any missing values, include the named parameter value "use ="complete.obs" in your code.

Answer

We are interested in the relation between mile run time (y variable) and the bench press weight (x variable).

The correlation is 0.8967417 (strong positive relationship) with a p-value smaller than 0.05 (p-value = 0.002544) which indicates a significant relationship at a 95% confidence level.

This outcome suggests that runners that lift heavier weights have longer run times and hence are slower than runners that lift lower weights.

```
cor.test( y = problem.8.data.frame$mile.run,
          x = problem.8.data.frame$bench.press,
          use = "complete.obs" )
```

```
##
##  Pearson's product-moment correlation
##
## data:  problem.8.data.frame$bench.press and problem.8.data.frame$mile.run
## t = 4.9634, df = 6, p-value = 0.002544
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5217977 0.9813133
## sample estimates:
##       cor
## 0.8967417
```

Part c)

Extract a data frame named `problem.8.hedgehog` consisting only of observations on hedgehogs. Display this data frame, and then calculate the correlation between bench press weights and mile run times for hedgehogs only.

Answer

```
problem.8.hedgehog <- data.frame(
  problem.8.data.frame[ problem.8.data.frame[, 1 ] == "hedgehog", ] )
```

Display the data frame so Karen can be sure that you extracted the correct rows:

```
problem.8.hedgehog
```

```
##    species bench.press mile.run
## 1 hedgehog         145     18.8
## 4 hedgehog          NA     15.2
## 6 hedgehog         123     12.3
## 8 hedgehog         148     13.7
## 9 hedgehog         145     14.5
```

Now calculate the correlation between bench press weights and mile run time for hedgehogs only (remember to use the named parameter "use ="complete.obs"):

```
cor.test( y = problem.8.hedgehog$mile.run,
          x = problem.8.hedgehog$bench.press,
          use = "complete.obs" )
```

```
##
##  Pearson's product-moment correlation
##
## data:  problem.8.hedgehog$bench.press and problem.8.hedgehog$mile.run
## t = 0.89734, df = 2, p-value = 0.4642
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.8768032  0.9880753
## sample estimates:
##       cor
## 0.5357625
```

The correlation is 0.5357625 (positive relationship) with a p-value greater than 0.05 (p-value = 0.4642) which indicates that this correlation is statistically insignificant.

Part d)

Extract a data frame named `problem.8.bunny` consisting only of observations on hedgehogs. Display this data frame, and then calculate the correlation between bench press weights and mile run times for hedgehogs only.

Answer

```
problem.8.bunny <- data.frame(
  problem.8.data.frame[ problem.8.data.frame[, 1 ] == "bunny rabbit", ] )
```

Display the data frame so Karen can be sure that you extracted the correct rows:

```
problem.8.bunny
```

```
##         species bench.press mile.run
## 2  bunny rabbit          58      8.3
## 3  bunny rabbit          67      7.5
## 5  bunny rabbit          84      6.9
## 7  bunny rabbit          95       NA
## 10 bunny rabbit          74      7.2
```

Now calculate the correlation between bench press weights and mile run time for hedgehogs only (remember to use the named parameter "use ="complete.obs"):

```
cor.test( y = problem.8.bunny$mile.run,
          x = problem.8.bunny$bench.press,
          use = "complete.obs" )
```

```
##
##  Pearson's product-moment correlation
##
## data:  problem.8.bunny$bench.press and problem.8.bunny$mile.run
## t = -5.0465, df = 2, p-value = 0.0371
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9992503 -0.0243701
## sample estimates:
##        cor
## -0.9629042
```

The correlation is -0.9629042 (strong negative relationship) with a p-value smaller than 0.05 (p-value = 0.0371) which indicates statistical significance.