Part 2: Vectors

Introduction to R

This assignment consists of six parts:

- Intro to Basics
- Vectors (this document)
- Matrices
- Factors
- Data frames
- Lists

If you haven't already, download the answer sheet for this document (**Note:** right-click and save the file with the `.Rmd` extension to your `hw02` folder.)

Open a new script and save it as `hw02-2.R`. Enter comments with your name and the assignment name like you did in Part 1. Commit and push your new R script to your remote repo. See Part 0 for instructions. Do *not* push this document.

Note: The skills with vectors you learn in this exercise apply to other data types so be sure you really understand the techniques. The next exercises will be easier to follow and complete.

2.1 Vectors explained

Vectors are one-dimensional arrays of one data type such as numerics or text strings. Examples of vectors would be the letters of the alphabet or the numbers 1 through 10.

In R, you create vectors with the `c()` function, which **c**ombines multiple values of the same data type together. For example, the following code creates a vector for the first six letters of the alphabet and for the first six positive integers.

```
alphabet <- c("A", "B", "C", "D", "E", "F")

integers <- c(1, 2, 3, 4, 5, 6)
```

Notice the use of quotation marks around the letters to indicate they are text strings. What do you think would happen if you ran this code?

```
alphabet <- c(A, B, C, D, E, F)
```

```
## Error in eval(expr, envir, enclos): object 'A' not found
```

Without the quotation marks, R expects to find variables named `A`, `B`, and so on. If you defined the variables first, then you could assign them to a vector, like so.

```
A <- "A"
B <- "B"

# Note the use of quotes for the last for letters but
# not for the two variables defined above.
```

```
alphabet <- c(A, B, "C", "D", "E", "F")

alphabet
```

```
## [1] "A" "B" "C" "D" "E" "F"
```

The variables `A` and `B` were defined as text strings so they can be combined with the other letters.

Recall the different data types from Intro to Basics exercise. What would happen if you ran the following code? The variable `G` is defined as an numeric class variable.

```
A <- "A"
B <- "B"
G <- 7 #  Define G as a numeric class variable.

# Vectors must have the same data type.
alphabet <- c(A, B, "C", "D", "E", "F", G)

alphabet
```

```
## [1] "A" "B" "C" "D" "E" "F" "7"
```

```
class(alphabet)
```

```
## [1] "character"
```

```
class(G)
```

```
## [1] "numeric"
```

Notice the quotation marks around the 7. `G` is a numeric class variable still but `alphabet` is character class. When R created the `alphabet` vector, it copied the value stored in `G` and changed the data type to match the data type of the rest of the vector. The change of data type by R (and other languages) is called *coercion* and is something to be careful of. It's always best to be sure you—the programmer—control the data type to ensure you get the results you expect.

2.2 Vector types

Vectors contain data of the same data type. You just saw examples of vectors for numeric and character data.

Instructions

- Add the two lines of code for the alphabet and integers vectors to your scripts.
- Add a line to your script to make a `logical_vector` variable that has three boolean elements: `FALSE`, `FALSE` and `TRUE` (in that order). Remember than boolean values are not text strings.

```
alphabet <- c("A", "B", "C", "D", "E", "F")

integers <- c(1, 2, 3, 4, 5, 6)

# Logical vector


#
```

2.3 Biological vectors (pun intended)

In 1963, George Kenny and Mary Pollock (Journal of Infectious Diseases 112: 7–16) studied growth of mammalian cells in cultures contaminated with pleuropneumonia-like organisms (PPLO), a persistent problem for lab-grown cell cultures. T

They counted cells every two days in cultures without and with PPLO contaminants. The values below are the average number of cells (log, base 10) in each culture type.

For `cultures_without_pplo`:

- Day 0: 4.6
- Day 2: 4.8
- Day 4: 5.1
- Day 6: 5.5
- Day 8: 5.8

For `cultures_with_pplo`:

- Day 0: 4.6
- Day 2: 4.7
- Day 4: 4.8
- Day 6: 4.9
- Day 8: 4.8

Note: The variable names describe their contents. Although they are long, descriptive variable names make your code much easier to read. Plus, you may have noticed that RStudio uses code completion to match variables and function names, which

Instructions

- Add the `cultures_without_pplo` vector shown below to your code.
- Add another line for the `cultures_with_pplo` with the data listed above.

```
# Cultures without pplo contaminant
cultures_without_pplo <- c(4.6, 4.8, 5.1, 5.5, 5.8)

# Cultures with pplo contaminant


#
```

2.4 Name your vectors

You should always be able to interpret your data quickly to ease your analysis and understanding. But, it can be hard to remember what each element (individual values) in a long vector represents.

The vectors in the previous section contain counts obtained on five days but the vectors themselves are not clear which day each value was frpm. You can name the elements so that when you view the vector you see the name and the value, so you can interpret your data easily.

In R, you use the `names()` function to name each element in a vector, as shown in this example code. Read at this example.

```
name_vector <- c("Michael", "Taylor")
names(name_vector) <- c("First Name", "Last Name")
```

```
name_vector
```

```
## First Name  Last Name
##  "Michael"   "Taylor"
```

This code creates the `name_vector` and then names each element in the vector. The first element "Michael" is named `First Name`, while the second element "Taylor" is named `Last Name`.

Use a vector to hold your names.

If you think about it, the `c("First Name", "Last Name")` code used to name the vector elements is the same code you use to *create* a vector. Thus, you can use *a variable* to store a vector of names to use again and again. Study this example.

```
# Two vectors of staff members from different business divisions
financial <- c("Jessica", "Smith", "Payroll", "Supervisor")
facilities <- c("Stephen", "Jones", "Electrical", "Technician")

# A vector of names needed to name the elements
names_vector <- c("First Name", "Last Name", "Department", "Position")

# Name the elements of each division
names(financial) <- names_vector
names(facilities) <- names_vector

financial
```

```
##   First Name    Last Name   Department     Position
##    "Jessica"      "Smith"    "Payroll" "Supervisor"
```

```
facilities
```

```
##   First Name    Last Name   Department     Position
##    "Stephen"      "Jones" "Electrical" "Technician"
```

Note: Store information you use multiple times in a variable. Use the variable to access the information when you need it. **Work smart, not hard.**

Instructions

Add the following to your script.

- Create a `days_sampled` vector with `Day 0`, `Day 2`, `Day 4`, `Day 6`, and `Day 8`.
- Use the `days_sampled` vector to name the elements of the `cultures_without_pplo` and `cultures_with_pplo` data vectors.
- Code to show that the elements of each data vector were named properly.
- Did you use comments to make sections in your code that correspond to the sections of this assignment? If not, go back and do so now. *I won't remind you again but I will expect you to use comments appropriately.*

```
# Create a `days_sampled` vector.


# Name your two data vectors with the `days_sampled` vector.
```

```
# Check that your two data vectors were properly named.


#
```

2.5 Calculations with vectors

R is designed to work efficiently with vectors. Study this example.

```
# Six numbers
six_numbers <- c(1, 2, 3, 5, 8, 13)

# Add 10 to each value in the vector
six_numbers <- six_numbers + 10

# Display the values in my_vector
six_numbers

# Use the square root function to obtain the square root of each value
six_numbers <- sqrt(six_numbers)

# Display the values
six_numbers

# Calculate the sum and mean of the values in my_vector
sum(six_numbers)

mean(six_numbers)
```

Notice that 10 was added to *each* element in the `six_numbers` vector and that the square root of *each* element was calculated. The `sum()` and `mean()` functions used *all* elements in the vector.

R has many built-in functions for numeric vectors including

- `sum()` adds together all of the elements,
- `mean()` calculates the average of the elements,
- `min()` finds the minimum value,
- `max()` finds the maximum value,
- `sqrt()` calculates the square root of each element, and
- `log10()` calculates the logarithm (base 10) of each element.

Instructions

Add code to your script to do the following.

- Find the maximum value stored in the `cultures_without_pplo` and `cultures_with_pplo` vectors. Store the results in `max_without_pplo` and `max_with_pplo` variables.
- Find the minimum values stored in each vector. Replace `max` in the variable names with `min`.

Math refresher: Before continuing with the instructions, let's recall that the cell counts in your vectors are logarithms (base 10). Logs are exponents (power) that raise the base (10 in this case) to get specific values. For example, if you have a logarithm value of 2, then $10^2 = 100$.

The first value (Day 0) in each of your vectors is 4.6; therefore the average number of cells in both culture types on Day 0 is $10^{4.6}$ or about 39,811 cells.

- Use exponentiation (^) to raise 10 to each value in your vectors to calculate the actual number of cells. Store the results in `cell_counts_without_pplo` and `cell_counts_with_pplo` variables, respectively.
- Calculate the average number of cells for `cell_counts_without_pplo` and `cell_counts_with_pplo`. You do not have to save these results to variables.

```
# Find the maximum values in cultures_without_pplo and cultures_without_pplo
# Store in max_without_pplo and max_with_pplo, respectively.


# Find the minimum values in cultures_without_pplo and cultures_without_pplo
# Store in min_without_pplo and min_with_pplo, respectively.


# Use 10^ to calculate the actual number of cells for each culture.
# Store in cell_counts_without_pplo and cell_counts_with_pplo


# Calculate the average number of cell counts for each vector.
# You do not have to save these values to variables


#
```

As a check, the results for `cell_counts_without_pplo` should be about 39810.72, 63095.73, 125892.54, 316227.77, 630957.34.

2.6 Extract individual elements from a vector

R provides many ways to get a subset of values from a vector, as demonstrated below.

```
# Same example from above
financial <- c("Jessica", "Smith", "Payroll", "Supervisor")

names_vector <- c("First Name", "Last Name", "Department", "Position")

names(financial) <- names_vector
```

R keeps tracks of elements by position. The first element in a vector is position 1, the second element is position 2, and so on. You can put the position number in square brackets to get a single element.

```
# Show the second element
financial[2]
```

```
## Last Name
##   "Smith"
```

You can specify multiple positions inside the square brackets. You can also put the positions in a vector and then use the vector. You can indicate a range of elements to extract by separating the first and last element positions with a : (colon).

```
# Show the first, second, and fourth element
# Notice that you have to use the c() function.
financial[c(1, 2, 4)]
```

```
##   First Name    Last Name     Position
##    "Jessica"      "Smith" "Supervisor"
```

```r
# same result using a vector of positions
position_vector <- c(1, 2, 4)
financial[position_vector]
```

```
##   First Name    Last Name     Position
##    "Jessica"      "Smith" "Supervisor"
```

```r
# Extract elements 2 through 4
financial[2:4]
```

```
##    Last Name   Department     Position
##      "Smith"    "Payroll" "Supervisor"
```

You can also use element names in a similar fashion as numbers. Names are much easier to remember than positions. Notice that the names have to be in quotes because they are text strings. But, you can't do ranges using element names.

```r
# Extract the last name
financial["Last Name"]
```

```
## Last Name
##   "Smith"
```

```r
# Extract first and last names, and position
financial[c("First Name", "Last Name", "Position")]
```

```
##   First Name    Last Name     Position
##    "Jessica"      "Smith" "Supervisor"
```

```r
# Can't use the names in a range as it generates an error
financial["Last Name":"Position"]
```

```
## Error in "Last Name":"Position": NA/NaN argument
```

Instructions

Add code to your script to do the following. You do not have to save the results to new variables for this section. Just enter code to show the elements indicated.

- Select the third element of `cultures_without_pplo` using position number.
- Select the odd numbered elements of `cell_counts_with_pplo` using a vector of position numbers.
- Select the elements for `Day 2` and `Day 4` by name from `cultures_with_pplo`.

```r
# Select the third element from cultures_without_pplo by position number


# Select the odd numbered elements of cell_counts_with_pplo using a vector of position numbers.



# Select the elements for `Day 2` and `Day 4` by name from cultures_with_pplo


#
```

2.7 Extraction by logical comparison

You can also use logical (boolean) operators to extract elements from a vector. R has several logical operators for comparison.

- > for greater than
- < for less than
- >= for greater than or equal to
- <= for less than or equal to
- == for equal to each other (*two* equal signs; a very common mistake is to use only one equal sign for comparison.)
- != not equal to each other

Comparisons using the logical operators return TRUE or FALSE for results. Study these examples.

```
# TRUE. 2 is greater than 1
2 > 1

# FALSE. 2 is not less than 1
2 < 1

# TRUE. 3 is not equal to 4
3 != 4

# TRUE
5 == 5   # Two equal signs!

# FALSE. You can compare strings.
"ABC" == "AB C"

# TRUE. Sum of 2 and 3 equals 5.
sum(c(2,3)) >= 5

# Also TRUE. Sum of 1, 2, and 3 equals 6, which is greater than 5.
sum(c(1, 2, 3)) >= 5

# Returns a vector of TRUE, TRUE, FALSE. The first two values are equal
# to or less than 2.  Remember that R works very well with vectors.
c(1, 2, 3) <= 2
```

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

You must remember that logical comparisons return TRUE or FALSE values so you can not use comparisons to obtain elements from a vector directly. You have to create a variable with your boolean values, then use the variable to extract the element(s) that meet your logical conditions. Study this example.

```
# Here's a short cut to make a vector with integers 1 through 10.
# Notice the use of the colon to indicate a range.
ten_integers <- c(1:10)
```

```
ten_integers
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# Select integers less than or equal to 5
low_integers <- ten_integers <= 5

# Vector of boolean values
low_integers
```

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

```
# Extract the elements
ten_integers[low_integers]
```

```
## [1] 1 2 3 4 5
```

```
# Boolean vector for even numbers
# Modulo is the remainder after division. Even numbers divided
# by two have a remainder of zero.
even_numbers <- ten_integers %% 2 == 0

# Vector of boolean values
even_numbers
```

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

```
# Extract the even numbers
ten_integers[even_numbers]
```

```
## [1]  2  4  6  8 10
```

```
# Ampersand (&) is a logical AND. This code selects any element that
# is greater than or equal to 4 AND less than or equal to 7.
number_range <- ten_integers >=4 & ten_integers <= 7

number_range
```

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

```
ten_integers[number_range]
```

```
## [1] 4 5 6 7
```

Instructions

Add code to your script to do the following.

- Use `cell_counts_without_pplo` to create a logical vector for cell counts greater than 100,000 (do not use the comma in your code; use 100000).
- Use that vector to show the days and log values from `cultures_without_pplo`. Yes, you can create a logical array from one vector and use it with another vector. You do not have to store this result in a new variable.
- Use `cell_counts_with_pplo` to create a logical vector for cells counts greater than 50,000 and less than 75,000. Use the logical `&` as shown in the example above.
- Use that logical vector to show the days and log values from `cultures_with_pplo`. You do not have to store this result in a new variable.

```
# Use `cell_counts_without_pplo` to create a logical vector for cell counts greater than 100,000 (do no

# Use that vector to show the days and log values from `cultures_without_pplo`.

# Use `cell_counts_with_pplo` and `&` to create a logical vector for cells counts greater than 50,000 a

# Use that logical vector to show the days and log values from `cultures_with_pplo`.

#
```

Save to GitHub

Be sure you have used comments throughout your code to identify sections and to describe what you are doing, then commit and push your `hw02-2.R` script to GitHub.