Vectors in R

Sean Hellingman

shellingman@tru.ca

Data Visualization and Manipulation through Scripting (ADSC1010)

Fall 2023

THOMPSON RIVERS UNIVERSITY

Topics

Introduction

- Vectors are one of the simplest data structures in R
  - Vectors contain elements of the same type (logical, integer, double, character, complex, or raw).
  - Many R functions are designed to operate on vectors.

- Vectors in R are not assigned as row or column vectors
  - If we want a row or column vector we need to use a matrix.

Creating Vectors `c()`

- The simplest (and most important) way to create vectors in R is with the concatenate function `c()`.
- The command `x <- c(-2,0,3)` would produce a vector such that
  - First element is -2.
  - Second element is 0.
  - Third element is 3.
- Can also assign different data types to vectors

Creating Vectors `seq()`

- Sometimes it is useful to generate a vector that contains a regular sequence of values in steps of one.
  - This can be done using the `seq()` function.

- The following calls are equivalent:
  - `Vector.1 <- 1:5`
  - `Vector.2 <- seq(from = 1, to = 5, by = 1)`
  - These calls initializes a vector from 1 to 5 and saves it as a variable.

Creating Vectors `rep()`

- The `rep()` function allows the user to replicate values a specified number of times.

- Code to generate a vector containing five 3s:
  - `Vector.3 <- rep(3, times = 5)`
  - This can also be used to repeat non-numeric values.

Example 1

- Create a vector containing the values: 5, 8, and 6 and call it `x`.
- Create a vector that contains the elements of the sequence from 1 to 5 in steps of 0.5 and call it `z`.
- Create a vector `y` that repeats `x` five times.

Positional Indexing

- To extract elements based on their position we write the position within `[]`
- The first positional index is 1 in R (it is 0 in Python).
- Use `Vector.3[2]` to extract the second element of Vector.3.
- Negative indexing allows us to extract everything *except* the negatively indexed position.
  - Use `Vector.3[-2]` to extract every element of Vector.3 apart from position 2.

Example 2

- Extract the element in the 4^{th} position of $\mathtt{y}$.
- Extract the elements in the 2^{nd}, 4^{th}, and 5^{th} position of $\mathtt{y}$.
- Extract all elements from $\mathtt{y}$ excluding the 5^{th} and 7^{th} elements.

Logical Indexing

- Another way to extract data from a vector is to use a logical expression as an index.
  - Use `Vector.3[Vector.3 > 3]` to extract every element of Vector.3 greater than 3.

Example 3

- Extract the elements of `y` that are greater than or equal to 6.

Replacing Elements

- We can use the indexing methods in combination with the assignment operator <- to change some elements of a vector.
  - Use `Vector.3[4] <- 400` to replace the 4^{th} element of Vector.3 with 400.

- Logical expressions may also be used to change some elements of vectors.
  - Use `Vector.3[Vector.3 <= 3] <- 55` to replace elements that are less than or equal to 3 with 55.

Ordering Elements I

- Sort the values from lowest to highest value we can use the `sort()` function
  - `Vector.Sorted <- sort(Vector.1)` saves the sorted version of `Vector.1`.

- To reverse the sort, from highest to lowest, we can either include the `decreasing = TRUE` argument when using the `sort()` function or first sort the vector using the `sort()` function and then reverse the sorted vector using the `rev()` function.
  - `Vector.Sorted.Rev <- sort(Vector.1, decreasing = TRUE)`
  - `Vector.Sorted.Rev <- rev(sort(Vector.1))`

Ordering Elements II

- The `order()` function returns the order number of each element in a vector.
  - Running the `order()` function on `Vector.5 <- c(3,5,2)` would return 3, 1, 2.

- Used to return a permutation that simply orders or rearranges a vector in ascending or descending order by their index positions.

Example 4

- Order the vector $\mathbf{y}$ from highest to lowest, replace the 3^{rd} and 4^{th} elements with 2, and reorder the new vector from lowest to highest.

Operations

- Arithmetic operators (unless otherwise specified) will operate on all elements of a vector
  - Use `Vector.5 * 3` to multiply every element of `Vector.5` by 3.
- We can also apply these operators to the elements of two or more vectors.
  - **Careful** when using arithmetic operators on vectors of different lengths as R will quietly recycle the elements in the shorter vector

Example 5

- Create a vector `z` that contains the elements of the sequence from 1 to 2 in steps of 0.25.
- Create a vector `t` that contains the elements of the sequence from 1 to 3 in steps of 0.75.
- Add 0.5 to every element of `t`.
- Multiply `z` and `t`; What happens?

Non-Numeric Vectors

- Vectors do not have to have numerical entries
  - `Midfield <- c("Stones", "Rodri", "Silva", "De Bruyne", "Gündoğan", "Grealish") #character`
  - `Remaining <- c(TRUE, TRUE, TRUE, TRUE, FALSE, TRUE) #logical`
  - Use `class()` to determine what class the elements in a vector are.
- All entries of a vector must be of the same type.
  - Can't have both logicals and characters in the same vector.
  - Objects of different types can be stored in lists (will cover later).
- If we try to mix data types, R will convert everything to the same type using an internal hierarchy.

Named Vectors

- Used to assign meaningful names to vector components
  - Named refers to vectors for which labels have been assigned.
  - Unnamed otherwise.
  - Being able to do things like `grades[c("Lab","Exam")]` makes code-writing much more efficient.

names

- Suppose that:
  - `x` is a vector (of any type).
  - `y` is a vector of character strings.
  - `x` and `y` are of equal length.

- `names(x) <- y` will label the components of `x` using the strings in `y`.

Example 6

- Create the vector `grades <- c(0.75,0.85,0.80,0.65)`
- Use `print(grades)` to print the grades
- Create the vector `tasks <- c("Test","Assignment","Lab","Exam")`
- Assign labels to the grades using `names(grades) <- tasks`
- Print the labeled grades using `print(grades)`

Exercise 1

- Suppose `x <- c(1,-4,6)` and `x <- c(-3,3)`. What do you expect the output of each of the following commands to be?
  - `z <- c(x,y)`
  - `z <- c(y,x)`
  - `z <- c(2,x)`
  - `z <- c(x,2)`
  - `z <- c(1,x,4)`
  - `z <- c(1,2,x,4,y,7)`
  - `z <- c(x,x,y,c(1,3),y,-2)`
  - `c(c(1,2,-4),1)`
  - `c(c(1,2,-4),c(1))`
- Execute each command and confirm your suspicions.

Exercise 2

- Use `seq` to create each of the following vectors.
  - $(2.25, 3.25, ..., 9.25, 10.25)$
  - $(-3, -4, ..., -9, -10)$
- Use `c` and `seq` to create each of the following:
  - $(3, 2, 1, 1, 2, 3)$
  - $(0, 0.5, 1, 1.5, 2, 1.5, 1, 0.5, 0)$

Exercise 3

- Use `rep` to create each of the following vectors.
  - $(1, 0, -1, 1, 0, -1, 1, 0, -1)$.
  - $(2, 0, 2, 0, 2, 0, 2, 0, 2, 0)$.
- Use `c` and `rep` to create each of the following:
  - $(1, 1, 1, 1, 3, 3, 3, 3, 3, 3)$.
  - $(1, 1, 1, 7, 2, 2, 2, 2, 2, 3)$.

Exercise 4

- Use the following vectors to determine R's hierarchy.
  - `test.vec1 <- c(33,FALSE, "Kane")`
  - `test.vec2 <- c(33,FALSE)`
  - `test.vec3 <- c(33,4.7,9L,FALSE)`

Exercise 5

- Create the vector $(1, 2^{1/2}, 3^{1/3}, 4^{1/4}, ..., 100^{1/100})$.
  - Use `seq` and `^` to complete

Exercise 6

- Consider the output of each of the following.
  - `x <- "Thiago Silva"`
  - `y <- c("Thiago", "Silva")`
  - `length(x)`
  - `nchar(x)`
  - `length(y)`
  - `nchar(y)`
- Use the results to explain how to interpret `length(z)` and `nchar(z)` for a character vector `z`.

Exercise 7

- In R, assign the following positions: Forward, Midfielder, Defender, and Goalkeeper as labels for the following players: Kane, Silva, Mings, Ederson.

References & Resources

1. Douglas, A., Roos, D., Mancini, F., Couto, A., & Lusseau, D. (2023). *An introduction to R*. Retrieved from https://intro2r.com/

- https://alexd106.github.io/intro2R/index.html
- https://www.datamentor.io/r-programming/vector