Vectors and data frames in R

Quantitative Applications for Data Analysis

Alexey Fedoseev

January 16, 2025

Vectors

We know how to store a value in a variable. To view what value is stored in a variable we can simply type the name of the variable in R prompt.

```
> myapple <- "Big Red Delicious Apple"
> myapple
[1] "Big Red Delicious Apple"
```

Now we have a task to buy fruits and vegetables of the shopping list:

- Apple
- Orange
- Lemon
- Potato
- Cabbage

From what we already know, we can create a separate variable containing each item on our shopping list. However, there is a better way to store our list!

Vectors

We are going to use vectors to store our shopping list.

A vector is a sequence of elements of the same basic type.

In order to create a vector use the c command that combines values into a vector.

```
> fruits <- c("Apple", "Orange", "Lemon")
> str(fruits)
 chr [1:3] "Apple" "Orange" "Lemon"
> vegetables <- c("Potato", "Cabbage")
> str(vegetables)
 chr [1:2] "Potato" "Cabbage"
> shopping_list <- c(fruits, vegetables)
> str(shopping_list)
 chr [1:5] "Apple" "Orange" "Lemon" "Potato" "Cabbage"
```

Vectors

After we created the vector it is usually important to access its elements. Every element in a vector is referenced by the index. Adding the square brackets with the number inside to the variable name prints out the value stored in this variable at this index.

```
> fruits
[1] "Apple" "Orange" "Lemon"
> fruits[1]
[1] "Apple"
> fruits[2]
[1] "Orange"
> fruits[3]
[1] "Lemon"
> cat("I need to buy one", fruits[1], "\n")
I need to buy one Apple
```

Note that indices of the vector in R start with 1.

Vectors

We can access multiple elements in the vector by specifying the vector of indices. R provides us with a fast way to generate a vector of sequential numbers.

```
> fruits[c(1,3)]
[1] "Apple" "Lemon"
> 1:5
[1] 1 2 3 4 5
> fruits[1:2]
[1] "Apple" "Orange"
```

If we ask for the elements that are not in the vector, R will print out an `NA` value (Not Available/Missing Value).

```
> fruits[4]
[1] NA
> fruits[1:5]
[1] "Apple"  "Orange" "Lemon"  NA       NA
```

Vectors

When R encounters a negative index it skips the corresponding element. Notice that the `fruits` variable did not change.

```
> fruits
[1] "Apple"  "Orange" "Lemon"
> fruits[-1]
[1] "Orange" "Lemon"
> fruits
[1] "Apple"  "Orange" "Lemon"
```

Often it is useful to use logical values `TRUE` or `FALSE` to indicate which element to keep (`TRUE`) or remove (`FALSE`).

```
> fruits[c(TRUE, FALSE, TRUE)]
[1] "Apple" "Lemon"
```

Vectors

How many items are on my shopping list? Use the command `length` to count the number of elements in the vector.

```
> fruits
[1] "Apple"  "Orange" "Lemon"
> length(fruits)
[1] 3
> shopping_list
[1] "Apple"   "Orange"  "Lemon"   "Potato"  "Cabbage"
> length(shopping_list)
[1] 5
```

Do you have apples on your list? Check it using %in% command

```
> "Apple" %in% shopping_list
[1] TRUE
> "Mango" %in% shopping_list
[1] FALSE
```

Vectors

We can store numerical values and other data types in a vector as well.

```
> mynumbers <- c(1,5,3,7,5)
> str(mynumbers)
 num [1:5] 1 5 3 7 5
```

But, what happens if we create a vector of mixed data types?

```
> mymix <- c(FALSE, 1)
> str(mymix)
 num [1:2] 0 1
> mymix <- c(FALSE, 1, "Apple")
> str(mymix)
 chr [1:3] "FALSE" "1" "Apple"
```

R converts every element of the vector into the type that suits all values. At first, a boolean `FALSE` was converted in a numerical `0`. However, adding a string `"Apple"` converts every element into a string.

Vectors

We can easily perform operations on the whole vector.

```
> mynumbers
[1] 1 5 3 7 5
> mynumbers + 10
[1] 11 15 13 17 15
> mynumbers * mynumbers
[1]  1 25  9 49 25
```

Very often we want to establish what elements of the vector satisfy a particular condition. This is called conditional slicing or subsetting.

```
> (mynumbers > 2) & (mynumbers < 6)
[1] FALSE  TRUE  TRUE FALSE  TRUE
> mynumbers[(mynumbers > 2) & (mynumbers < 6)]
[1] 5 3 5
```

Vectors

To make our shopping list useful we need to add quantities. We want our shopping list to look like a Excel spreadsheet with rows and columns. Let us use already defined variable `mynumbers` as our quantities.

```
> mynumbers
[1] 1 5 3 7 5
> shopping.cart <- cbind(shopping_list, mynumbers)
> shopping.cart
     shopping_list mynumbers
[1,] "Apple"       "1"
[2,] "Orange"      "5"
[3,] "Lemon"       "3"
[4,] "Potato"      "7"
[5,] "Cabbage"     "5"
```

Notice how R converted all elements into strings. The resulting `shopping.cart` is represented as a **matrix**.

Matrices

A **matrix** is a collection of elements of the same data type arranged in a two-dimensional rectangular layout. You can think of a matrix as a mentioned earlier Excel spreadsheet.

Using the command `cbind` we can add more columns with details to our shopping cart.

```
> locations <- c("Costco", "Walmart", "Walmart", "Zehrs", "Walmart")
> shopping.cart <- cbind(shopping.cart, locations)
> shopping.cart
     shopping_list mynumbers locations
[1,] "Apple"       "1"       "Costco"
[2,] "Orange"      "5"       "Walmart"
[3,] "Lemon"       "3"       "Walmart"
[4,] "Potato"      "7"       "Zehrs"
[5,] "Cabbage"     "5"       "Walmart"
```

Column names

We can improve our `shopping.cart` by renaming our columns using `colnames` command.

```
> colnames(shopping.cart)
[1] "shopping_list" "mynumbers" "locations"
> colnames(shopping.cart) <- c("Items", "Quantities", "Locations")
> shopping.cart
     Items     Quantities Locations
[1,] "Apple"   "1"        "Costco"
[2,] "Orange"  "5"        "Walmart"
[3,] "Lemon"   "3"        "Walmart"
[4,] "Potato"  "7"        "Zehrs"
[5,] "Cabbage" "5"        "Walmart"
```

Data frames

Our shopping cart looks good, however quantities should be numbers instead of strings. It is important as we can perform mathematical operations on numbers and not on strings. *Data frames* allow us to store various data types in one variable. Since data frame type is more suitable for our shopping cart, we will overwrite the variable `shopping.cart`.

```
> shopping.cart <- data.frame(shopping_list, mynumbers, locations)
> shopping.cart
  shopping_list mynumbers locations
1         Apple         1    Costco
2        Orange         5   Walmart
3         Lemon         3   Walmart
4        Potato         7     Zehrs
5       Cabbage         5   Walmart
```

How to choose between a matrix or a data frame? You should use matricies if you perform a lot of mathematical operations on a very large amount of numbers. This will ensure your calculations to be more efficient. Otherwise, use data frames.

Data frames

Renaming the columns in a data frame as easy as in a matrix.

```
> colnames(shopping.cart) <- c("Items", "Quantities", "Locations")
> shopping.cart
    Items Quantities Locations
1   Apple          1    Costco
2  Orange          5   Walmart
3   Lemon          3   Walmart
4  Potato          7     Zehrs
5 Cabbage          5   Walmart
```

Data frames

We can verify that our quantities in fact are numbers by using the command `str`.

```
> str(shopping.cart)
'data.frame':   5 obs. of  3 variables:
 $ Items     : Factor w/ 5 levels "Apple","Cabbage",..: 1 4 3 5 2
 $ Quantities: num  1 5 3 7 5
 $ Locations : Factor w/ 3 levels "Costco","Walmart",..: 1 2 2 3 2
```

Notice that Items and Locations have a factor data type which we have not seen before.

Note: in newer versions on R you need to add stringsAsFactors=TRUE to the list of parameters for the command `data.frame` to get the same output, as the default behavior of this command has changed. For example:

```
shopping.cart <- data.frame(shopping_list, mynumbers,
                            locations, stringsAsFactors=TRUE)
```

Factors

Factors help us to find out the categories in a vector and how are they all doing.

Since we only shop in Costco, Walmart and Zehrs, they will be our categories or `levels`.

```
> locations
[1] "Costco"  "Walmart" "Walmart" "Zehrs"   "Walmart"
> places <- factor(locations)
> places
[1] Costco  Walmart Walmart Zehrs   Walmart
Levels: Costco Walmart Zehrs
> summary(places)
 Costco Walmart   Zehrs
      1       3       1
```

The command `summary` allows us to see how often each level appears in the vector.

summary

The summary command actually gives us more information if we use it on our shopping.cart.

```
> summary(shopping.cart)
     Items      Quantities    Locations
 Apple  :1   Min.   :1.0   Costco :1
 Cabbage:1   1st Qu.:3.0   Walmart:3
 Lemon  :1   Median :5.0   Zehrs  :1
 Orange :1   Mean   :4.2
 Potato :1   3rd Qu.:5.0
             Max.   :7.0
```

The column Quantities shows us the statistics that are useful for the data analysis. For the columns with factors summary gives us the frequencies of the levels.

Selecting columns

There are three ways to select a whole column in a data frame that are all equivalent to each other. Additionally, you can refer to the column by its index (which is less descriptive but useful sometimes).

```
> shopping.cart[["Quantities"]]
[1] 1 5 3 7 5
> shopping.cart[,"Quantities"]
[1] 1 5 3 7 5
> shopping.cart$Quantities
[1] 1 5 3 7 5
> shopping.cart[,2]
[1] 1 5 3 7 5
```

If you are using the dollar sign $ to extract the actual column and the column name has special characters, it **must** be surrounded by quotes (`shopping.cart$"Price/kg"`) or backticks (`shopping.cart$`Price/kg``).

Adding columns to a data frame

There is a convinient way to add a column to the data frame.

```
> prices.kg <- c(3.27, 3.9, 3.97, 3.49, 2.14)
> shopping.cart$Price <- prices.kg
> colnames(shopping.cart)
[1] "Items"      "Quantities" "Locations"  "Price"
> colnames(shopping.cart)[4] <- "Price/kg"
> shopping.cart
    Items Quantities Locations Price/kg
1   Apple          1     Costco     3.27
2  Orange          5    Walmart     3.90
3   Lemon          3    Walmart     3.97
4  Potato          7      Zehrs     3.49
5 Cabbage          5    Walmart     2.14
```

Selecting rows

You can extract rows from the data frame using slicing.

```
> shopping.cart[c(2,4),]
   Items Quantities Locations Price/kg
2 Orange          5   Walmart     3.90
4 Potato          7     Zehrs     3.49
> shopping.cart[c(2,4), c("Items", "Price/kg")]
   Items Price/kg
2 Orange     3.90
4 Potato     3.49
> shopping.cart[2,"Items"]
[1] Orange
Levels: Apple Cabbage Lemon Orange Potato
```

Notice how selecting rows preserves all levels for the factors.

Conditional slicing or subsetting data

While looking at your shopping list you want to know if you need to bring the car and what items you are buying at Walmart.

```
> shopping.cart$Quantities > 4
[1] FALSE  TRUE FALSE  TRUE  TRUE
> shopping.cart[shopping.cart$Quantities > 4,]
    Items Quantities Locations Price/kg
2  Orange          5   Walmart     3.90
4  Potato          7     Zehrs     3.49
5 Cabbage          5   Walmart     2.14
> shopping.cart$Location == "Walmart"
[1] FALSE  TRUE  TRUE FALSE  TRUE
> shopping.cart[shopping.cart$Location == "Walmart",]
    Items Quantities Locations Price/kg
2  Orange          5   Walmart     3.90
3   Lemon          3   Walmart     3.97
5 Cabbage          5   Walmart     2.14
```

Adding columns

You are also curious how much money you need to take with you to the store.

```
> shopping.cart$Total <- shopping.cart$Quantities * shopping.cart$`Price/kg`
> shopping.cart
    Items Quantities Locations Price/kg Total
1   Apple          1    Costco     3.27  3.27
2  Orange          5   Walmart     3.90 19.50
3   Lemon          3   Walmart     3.97 11.91
4  Potato          7     Zehrs     3.49 24.43
5 Cabbage          5   Walmart     2.14 10.70
```

Using R command `sum` we can easily calculate how much money we need to have on us.

```
> cat("Total amount:", sum(shopping.cart$Total), "\n")
Total amount: 69.81
```