Data Structures

July 16, 2023 V5

Popular Data Structures

The R programming language includes a number of data structures that are frequently employed in data analysis and statistical modeling. These are some of the most popular data structures in R:

1. **Vector**: A vector is a one-dimensional array that stores identical data types, such as numeric, character, or logical. The `c()` function can be used to create vectors, and indexing can be used to access individual vector elements.
2. **Factor**: A factor is a vector representing **categorical** data, with each distinct value or category represented as a level. Using indexing, individual levels of a factor can be accessed using the `factor()` function.
3. **Dataframe**: A data frame is a two-dimensional table-like structure similar to a spreadsheet, that can store various types of data in columns. The `data.frame()` function can be used to construct data frames, and individual elements can be accessed using row and column indexing.
4. **Matrix**: A matrix is a two-dimensional array of data with identical rows and columns. The `matrix()` function can be used to construct matrices, and individual elements can be accessed using row and column indexing.
5. **Array**: An array is a multidimensional data structure that can contain data of the same data type in user-specified dimensions. Arrays can be constructed using the `array()` function, and elements can be accessed using multiple indexing.
6. **List**: A list is an object that may comprise elements of various data types, including vectors, matrices, data frames, and even other lists. The `list()` function can be used to construct lists, while indexing can be used to access individual elements.

These data structures are helpful for storing and manipulating data in R, and they can be utilized in numerous applications, such as statistical analysis and data visualization. We will focus our attention on Vectors, Factors and Dataframes, since we believe that these are the three most useful data structures.

[1]

Vectors

1. A vector is a fundamental data structure in R that can hold a sequence of values of the same data type, such as integers, numeric, character, or logical values.
2. A vector can be created using the `c()` function.
3. R supports two forms of vectors: atomic vectors and lists. Atomic vectors are limited to containing elements of a single data type, such as numeric or character. Lists, on the other hand, can contain elements of various data types and structures. [1]

Vectors in R

1. The following R code creates a numeric vector, a character vector and a logical vector respectively.

```r
# Read data into vectors
names <- c("Ashok", "Bullu", "Charu", "Divya")
ages <- c(72, 49, 46, 42)
females <- c(FALSE, TRUE, TRUE, TRUE)
```

2. The `c()` function is employed to combine the four character elements into a single vector.
3. Commas separate the elements of the vector within the parentheses.
4. Individual elements of the vector can be accessed via indexing, which utilizes square brackets []. For instance, names[1] returns "Ashok", while names[3] returns "Charu".
5. We can also perform operations such as categorizing and filtering on the entire vector. For instance, `sort(names)` returns a vector of sorted names, whereas names[names!= "Bullu"] returns a vector of names excluding "Bullu."

Vector Operations

Vectors can be used to perform the following vector operations:

1. **Accessing Elements:** We can use indexing with square brackets to access individual elements of a vector. To access the second element of the "names" vector, for instance, we can use:

```
names[2]
```

```
[1] "Bullu"
```

This returns “Bullu”, the second element of the “names” vector.

2. **Concatenation:** The “c()” function can be used to combine multiple vectors into a single vector. For instance, to combine the “names” and “ages” vectors into the “people” vector, we can use:

```
persons <- c(names, ages)
persons
```

```
[1] "Ashok" "Bullu" "Charu" "Divya" "72"    "49"    "46"    "42"
```

This generates an eight-element vector containing the names and ages of the four people.

3. **Subsetting:** We can use indexing with a logical condition to construct a new vector that contains a subset of elements from an existing vector. For instance, to construct a new vector named “female_names” containing only the females’ names, we can use:

```
female_names <- names[females == TRUE]
female_names
```

```
[1] "Bullu" "Charu" "Divya"
```

This generates a new vector comprising three elements containing the names of the three females (“Bullu”, “Charu”, and “Divya”).

4. **Arithmetic Operations:** We can perform element-wise arithmetic operations on vectors. To calculate the sum of the “ages” vector, for instance, we can use:

```
sum(ages)
```

```
[1] 209
```

This returns 209, the sum of the four ages.

5. **Logical Operations:** We can perform logical operations on vectors, which are also executed element-by-element. To create a new vector titled “middle_age” that indicates whether each individual is 45 to 55 years old, for instance, we can use:

```r
middle_age <- (ages >= 45) & (ages <= 55)
middle_age
```

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

This generates a new vector with four elements containing logical values indicating whether each person is between 45 and 55 years of age.

To test whether any of the elements in the “ages” vector are greater than 50, we can use:

```r
any(ages > 50)
```

```
[1] TRUE
```

6. **Unique Values:** We can find the unique values in a vector using the “unique()” function. For example, to find the unique values in the “ages” vector, we can use:

```r
unique(ages)
```

```
[1] 72 49 46 42
```

7. **Sorting:** We can sort a vector in ascending or descending order using the “sort()” function. For example, to sort the “ages” vector in descending order, we can use:

```r
sort(ages, decreasing = TRUE)
```

```
[1] 72 49 46 42
```

Statistical Operations on Vectors

1. **Length**: The length represents the count of the number of elements in a vector.

```r
length(ages)
```

```
[1] 4
```

2. **Maximum** and **Minimum**: The maximum and minimum values are the vector's greatest and smallest values, respectively.
3. **Range**: The range is a measure of the spread that represents the difference between the maximum and minimum values in a vector.

```r
min(ages)
```

```
[1] 42
```

```r
max(ages)
```

```
[1] 72
```

```r
range(ages)
```

```
[1] 42 72
```

4. **Mean**: The mean is a central tendency measure that represents the average value of a vector's elements.
5. **Standard Deviation**: The standard deviation is a measure of dispersion that reflects the amount of variation in a vector's elements.

```r
mean(ages)
```

```
[1] 52.25
```

```r
sd(ages)
```

```
[1] 13.47529
```

6. **Median**: The median is a measure of central tendency that represents the middle value of a sorted vector.

```r
median(ages)
```

```
[1] 47.5
```

7. **Quantiles**: The quantiles are a set of cut-off points that divide a sorted vector into equal-sized groups.

```r
quantile(ages)
```

```
   0%   25%   50%   75%  100% 
42.00 45.00 47.50 54.75 72.00 
```

This will return a set of five values, representing the minimum, first quartile, median, third quartile, and maximum of the four ages.

Thus, we note that the R programming language provides a wide range of statistical operations that can be performed on vectors for data analysis and modeling. Vectors are clearly a potent and versatile data structure that can be utilized in a variety of ways.

Strings

Here are some common string operations that can be conducted using the provided vector examples.

1. **Substring**: The `substr()` function can be used to extract a substring from a character vector. To extract the first three characters of each name in the "names" vector, for instance, we can use:

```r
substr(names, 1, 3)
```

```
[1] "Ash" "Bul" "Cha" "Div"
```

This returns a new character vector containing the initial three letters of each name (“Ash”, “Bul”, “Cha”, and “Div”).

2. **Concatenation**: Using the `paste()` function, we can concatenate two or more character vectors into a singular vector. To create a new vector containing the names and ages of the individuals, for instance, we can use:

```
persons <- paste(names, ages)
persons
```

```
[1] "Ashok 72" "Bullu 49" "Charu 46" "Divya 42"
```

This will generate a new eight-element character vector containing the name and age of each individual, separated by a space.

3. **Case Conversion:** The `toupper()` and `tolower()` functions can be used to convert the case of characters within a character vector. To convert the “names” vector to uppercase letters, for instance, we can use:

```
toupper(names)
```

```
[1] "ASHOK" "BULLU" "CHARU" "DIVYA"
```

This will generate a new character vector with all of the names converted to uppercase.

4. **Pattern Matching:** Using the `grep()` and `grepl()` functions, we can search for a pattern within the elements of a character vector. To find the names in the “names” vector that contain the letter “a”, for instance, we can use:

```
grep("a", names)
```

```
[1] 3 4
```

This returns a vector containing the indexes of the “names” vector elements that contain the letter “a.”

5. **Regular Expressions:** We can use regular expressions with the `grep()` and `grepl()` functions to search for patterns in the elements of a character vector. To find the names in the “names” vector that begin with the letter “C”, for instance, we can use:

```r
grep("^C", names)
```

```
[1] 3
```

This returns a vector containing the indexes of the elements in "names" that begin with the letter "C." [1]

References

[1]

R Core Team. (2021). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. **https://www.R-project.org/**

R Core Team. (2022). Vectors, Lists, and Arrays. R Documentation. https://cran.r-project.org/doc/manuals/r-release/R-intro.html#vectors-lists-and-arrays

Wickham, H., & Grolemund, G. (2016). R for data science: Import, tidy, transform, visualize, and model data. O'Reilly Media, Inc.