Data Structures in R

Killian Anderson

January 23, 2026

1 Vectors: The Atomic Structure

In R, a vector is a sequence of elements of the same type. In other words, we may treat this sequence of elements $\mathbf{v}$ mathematically as an element of a n-dimensional space $\mathbf{v} = \{x_i \mid i = 1, 2, \ldots, n\}$. Do not get the definition of a vector in R, i.e. a data structure, confused with the mathematical definition of a vector in Euclidean geometry.

1.1 Formal Definition of a Vector

A vector can be defined by the set:

$$\mathbf{v} = \{x_i \mid i = 1, 2, \ldots, n\} = [x_1, x_2, ..., x_n]$$

There are three types of vector that can be created by concatenating data using `c()` known as the combine function.

- Numeric Vector: `num_vec <- c(1, 2, 3)`
- Character Vector: `char_vec <- c("apples", "bananas", "pralines")`
- Logical Vector: `log_vec <- c(TRUE, TRUE, FALSE)`

Special creation functions include:

- **Sequence Operator (:):** Generates an arithmetic progression where $a, b \in \mathbb{R}$.
$$a : b = \{a, a + 1, a + 2, \ldots, b\}$$
- **Repetition (`rep`):** Creates a vector by repeating a constant or a pattern.
$$\text{rep}(k, n) = \underbrace{(k, k, \ldots, k)}_{n \text{ times}}$$

1.2 Manipulation via Indexing

Let $\mathbf{v}$ be a vector of length n. Manipulation is defined by a mapping function $f : I \to \mathbf{v}$, where I is a set of indices.

- **Subsetting:** $\mathbf{v}_S = \{x_i \in \mathbf{v} \mid i \in S\}$ where $S \subseteq \{1, \ldots, n\}$.
- **Logical Masking:** Given a condition P, the subset is $\mathbf{v}_p = \{x_i \in \mathbf{v} \mid P(x_i) = \text{TRUE}\}$.

1.2.1 Accessing Data in a Vector

There are various ways to call the data in R. Let us create the following vector `x <- 1:10` and showcase its respective manipulation:

- Size: `length(x)`, n
  ```
  length(x)
  [1] 10
  ```
- i-th Element: `x[i]`, $\{x_i\}$
  ```
  x[2]
  [1] 2
  ```
- Exclusion: `x[-i]`, $\{x_j \mid j \in \{1, \ldots, n\}, j \neq i\}$
  ```
  x[-2]
  [1] 1  3  4  5  6  7  8  9 10
  ```
- Multiple Exclusion: `x[-c(i, j)]`, $\{x_k \mid k \in \{1, \ldots, n\}, k \notin \{i, j\}\}$
  ```
  x[-c(1, 2)]
  [1] 3 4 5 6 7 8 9 10
  ```
- Element Sequence: `x[j:k]` $(x_j, \ldots, x_k)$
  ```
  x[3:6]
  [1] 3 4 5 6
  ```
- Logic Filtering: `x[x > n]`, $\{x_i \in \mathbf{x} \mid x_i > n\}$
  ```
  x[x > 8]
  [1] 9 10
  ```
- Compound Filtering:

  ```
  x[x < j | x > k]
  ```

  $\{x_i \in \mathbf{x} \mid x_i < j \vee x_i > k, j < k\}$

  ```
  x[x < 2 | x > 8]

  [1] 1 9 10
  ```

2 Data Frames: The Relational Structure

A data frame is a heterogeneous two dimension structure. Mathematically, it is best represented as a named list of vectors of equal length.

2.1 Matrix-Like Representation

Let a data frame D be a collection of m column vectors $\mathbf{c}_j$:

$$D = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m]$$

Note that while a matrix requires all $\mathbf{c}_j$ to be of the same type, a data frame allows the following:

$$\text{type}(\mathbf{c}_j) \neq \text{type}(\mathbf{c}_k) \text{ for } j \neq k$$

We may create a data frame from vectors if we have not loaded one from a file.

1. `x <- 1:3`
2. `y <- c("apples", "bananas", "pralines")`
3. `df <- data.frame(x, y)`
4. `df`

```
  x        y
1 1   apples
2 2  bananas
3 3 pralines
```

2.1.1 Manipulation and Transformations

- **Selection:** Access to a specific element $x_{i,j}$ (row i, column j).
- **Linear Transformation:** Create a new column $\mathbf{c}_{new}$ based on existing columns:

$$\mathbf{c}_{new} = g(\mathbf{c}_j, \mathbf{c}_k)$$

Continuing with the aforementioned example, we may add another column to our data frame.

1. `df$z <- c("yes", "no", "yes")`
2. `df`

```
  x        y   z
1 1   apples yes
2 2  bananas  no
3 3 pralines yes
```

3. `df$z`

```
[1] "yes" "no"  "yes"
```

2.2 Accessing Data in a Data Frame

Using the data frame from above, we get the following.

1. Dimensions: `dim(df)`, (n, m)
   `dim(df)`

   ```
   [1] 3  3
   ```

2. Specific Element by Index: `df[i, j]`, $x_{i,j}$
   `df[2, 1]`

   ```
   [1] 2
   ```

3. Row Selection by Index: `df[i, ]`, $\{x_{i,j} \mid j \in \{1, \ldots, m\}\}$
   `df[2, ]`

   ```
     x       y  z
   2 2 bananas no
   ```

4. Column Selection by Index: `df[, j]`, $\{x_{i,j} \mid i \in \{1, \ldots, n\}\}$
   `df[, 3]`

   ```
   [1] "yes" "no"  "yes"
   ```

5. Column Selection by Name: `df$name` or `df[["name"]]`
   `df$x`

   ```
   [1] 1 2 3
   ```

6. Logical Row Filtering: `df[df$var > n, ]`, $\{\mathbf{r}_i \mid x_{i,var} > n\}$
   `df[df$x > 2, ]`

   ```
     x        y   z
   3 3 pralines yes
   ```

7. Column Exclusion: `df[, -j]`, $\{\mathbf{c}_k \mid k \in \{1, \ldots, m\}, k \neq j\}$
   `df[, -1]`

   ```
           y   z
   1   apples yes
   2  bananas  no
   3 pralines yes
   ```

8. Row Exclusion: `df[-i,]`, $\{\mathbf{r}_j \mid j \in \{1, \ldots, n\}, j \neq i\}$
   `df[-1,]`

```
  x        y   z
2 2  bananas  no
3 3 pralines yes
```

9. Multiple Column Exclusion: `df[, -c(j, k)]`, $\{\mathbf{c}_h \mid h \in \{1, \ldots, m\}, h \notin \{j, k\}\}$
```
df[, -c(1, 2)]
```

```
[1] "yes" "no"  "yes"
```

10. Multiple Row Exclusion: `df[-c(i, j), ]`, $\{\mathbf{r}_h \mid h \in \{1, \ldots, n\}, h \notin \{i, j\}\}$
```
df[-c(1, 2), ]
```

```
  x        y   z
3 3 pralines yes
```

3 Accessing Information via Graphical Interface

We may access information in a graphical format by supposed x to be vector or a data frame has been created by using the following:

- Edit Spreadsheet: `data.entry(x)`
- Edit spreadsheet without save: `x = de(x)`
- Edit R command:`x = edit(x)`

4 Input/Output Operations

The transition between disk storage and memory is handled via:

1. **Read:** f : CSV File $\rightarrow$ Data Frame
2. **Write:** f : Data Frame $\rightarrow$ CSV File

Putting everything together, we may use a vector to create a data frame, then subsequently write that to a file on the machine and read it back.

1. ```
   vec <- c(10, 20, 30, 40)
   vec_sub <- vec[vec >= 20]
   ```
2. `df <- data.frame(id = 1:3, vec_sub)`
3. `write.csv(df, "data.csv", row.names = FALSE)`
4. `read.csv("data.csv")`
   ```

4.1 Working Directory

The working directory is where R will interact with files and you may find out the default or set the working directory yourself in the following way.

1. Get working directory: `getwd()`
2. Set working directory: `setwd("your_file_path")`

4.2 Getting Help

The user of R may get help by typing writing a function into the console and placing a question mark in front of the command such as the following.

```
?mean()
```