SPEC Lab REU R Resources: Intro to R - Vectors & Tibbles

Alix Ziff, Gaea Morales, Zachary Johnson

Summer 2024

Welcome to the second walkthrough document in our Introduction to R series. Building on what we covered previously—basic arithmetic operations and foundational R concepts—this document will take you further into the essential elements of R, focusing specifically on vectors and tibbles. These are key skills that will lay the groundwork for everything you'll do in R, from data analysis to advanced statistical modeling.

Vectors: The Building Blocks of Data in R

A **vector** is the simplest type of data you can work with in R. A **vector** is the simplest type of data you can work with in R. "A vector or a one-dimensional array simply represents a collection of information stored in a specific order" (Imai 2016: 6). It is essentially a list of data of a single type (either numerical, character, or logical). (See also, Minions). Another way to think about it is as a column of data in a table (or dataframe in R–more on this later).

Variable types

It's important to understand the different types of variables you will be working with. There are four main variable types:

- **Numerical**: Any type of number, which can include:
  - **Numeric (num)**: Refers to any number, including decimals. These values are also known as **doubles** when they include decimals.
  - **Integer (int)**: Whole numbers without decimal points.
- **Character**: Alphanumeric data that is not ordered as a character vector.
- **Logical**: A collection of `TRUE` and `FALSE` values.
- **Factor**: Represents categorical variables that have specific levels, such as ordinal variables.

Checking Data Types

There are several ways to check the data type of an object in R:

- **Global Environment:** The abbreviation (e.g., num or int) appears next to the object name.
- Use the `str()` command to get a compact display of the structure of an R object.
- Use the `class()` command to displays the class of the R object, revealing its data type.

Working with Vectors

To create a vector, you use the `c()` function, which stands for "concatenate," to combine separate data points. The general format for creating a vector in R is as follows:

```
name_of_vector <- c("What you want to put into the vector")
```

For example, let's suppose that we have population data (in millions) for the five most populous countries in 2016. We will create a vector with these numerical values separated by commas, called `pop1`.

```
# Create vector with the population data
pop1 <- c(1379, 1331, 325, 258, 207)

# Display the contents of vector pop1
pop1
```

```
## [1] 1379 1331  325  258  207
```

You can also combine vectors using the `c()` function. For example, let's say we have data for 5 more countries, which we will store in a vector named `pop2`, and we want to merge it with `pop1`.

```
# Creating another vector with the population data
pop2 <- c(194, 187, 161, 142, 127)

# Combining pop1 and pop2 into a single vector
pop <- c(pop1, pop2)

# Display the contents of vector pop
pop
```

```
##  [1] 1379 1331  325  258  207  194  187  161  142  127
```

Exploring and Modifying Vectors

To learn more about a vector, such as its type or structure, you can use the `str()` function.

```
# Check the structure of vector pop
str(pop)
```

```
##  num [1:10] 1379 1331 325 258 207 ...
```

This command tells us that the object `pop` is of class numeric and has the dimensions `[1:10]`(10 elements in one dimension).

Indexing Vectors

How do we retrieve the nth value from a vector? The **index** is the number that indicates the position of the value you're interested in (e.g., an index of 1 corresponds to the 1st value in the vector), and **indexing** allows you to access specific elements in a vector.

Let's use the `pop2` vector for the following example. What's the third value in `pop2`?

```
# Extract the 3rd value from pop2
pop2[3]
```

```
## [1] 161
```

What if we wanted to get rid of the last country in the set of 5 (i.e., the 5th element)?

```
# Remove the 5th value from pop2
pop2[-5]
```

```
## [1] 194 187 161 142
```

What if we wanted to find out multiple values from the same vector at once, e.g. the 2nd, 3rd, and 4th values? We can use either the `c()` function or a colon (`:`) to specify a range.

```
# Extract the 2nd, 3rd, and 4th values from pop2
pop2[c(2,3,4)] # or
```

```
## [1] 187 161 142
```

```
pop2[2:4]
```

```
## [1] 187 161 142
```

```
## Both lines return the same result
```

Can you think of what the code would look like if we wanted to **drop** values 2-4 instead?

```
# Remove the 2nd to 4th values from pop2
pop2[-c(2,3,4)]  # or
```

```
## [1] 194 127
```

```
pop2[-2:-4]
```

```
## [1] 194 127
```

```
## Both lines return the same result
```

Using Functions with Vectors

This section revisits concepts from the earlier walkthrough on calculations. We will explore several special functions that operate on vectors, enabling us to calculate measures of central tendency and variability.

	Function
`min()`	Finds the minimum value.
`max()`	Finds the maximum value.
`sum()`	Sums all the values.
`length()`	Returns the number of elements in the vector.
`mean()`	Calculates the average.
`median()`	Finds the median value.
`var()`	Computes the variance.
`sd()`	Computes the standard deviation.

What is the average population size of these countries?

```
# Calculate the mean (average) population in pop
mean(pop)
```

```
## [1] 431.1
```

How many people total live in all these places?

```
# Calculate the total population in pop
sum(pop)
```

```
## [1] 4311
```

How many elements are in our vector? (We already know the answer, but this is useful to know for when we are working with larger vectors).

```
# Count the number of elements (countries) in pop
length(pop)
```

```
## [1] 10
```

Combining Functions and Arithmetic

You can also combine functions and perform calculations within vectors. For instance, to find values above the median:

```
# Extracting values from pop that are greater than the median population
pop[pop > median(pop)]  # or
```

```
## [1] 1379 1331  325  258  207
```

```
## Calculate it separately:
median(pop) # First calculate the median
```

```
## [1] 200.5
```

```
pop[pop > 200.5] # Then you extract the values above the median
```

```
## [1] 1379 1331  325  258  207
```

Now count how many values are above the mean:

```
# Count how many values are above the median in pop
length(pop[pop > median(pop)])
```

```
## [1] 5
```

Adding Character Data

Suppose you want to associate the country names with the population data. To save time, we will create a character vector only with the country codes of the five most populous countries.

```
# Create a character vector named cname with the country codes of the five most populous countries
cname <- c("CHN", "IND", "USA", "IDN", "BRA")

# Check the structure of cname
str(cname)
```

```
##  chr [1:5] "CHN" "IND" "USA" "IDN" "BRA"
```

Creating Logical and Factor Variables

Now, let's code a logical variable that shows whether the country is in Asia or not.

```
# Create a logical vector named asia indicating whether each country is in Asia
asia <- c(TRUE, TRUE, FALSE, TRUE, FALSE)
## TRUE = in Asia
## FALSE = not in Asia

# Check the structure of asia
str(asia)
```

```
##  logi [1:5] TRUE TRUE FALSE TRUE FALSE
```

Lastly, we will define a factor variable for the regime type that can take one of four values (based on Economist Intelligence Unit): Full Democracy, Flawed Democracy, Hybrid Regimes, Autocracies.

```
# Create a factor vector named regime representing the regime type of each country
regime <- c("Autocracy", "FlawedDem", "FullDem", "FlawedDem", "FlawedDem")

# Converting the regime vector into a factor
regime <- as.factor(regime)
```

```r
# Check the structure of regime
str(regime)
```

```
##  Factor w/ 3 levels "Autocracy","FlawedDem",..: 1 2 3 2 2
```

Introducing Tibbles

As social scientists, the data we work with rarely consist of single vectors. For example, a list of populations is not incredibly informative if we do not know that they are referring to specific countries, and in what year the data were collected. We might also be curious about other characteristics of these countries, such as what region they are in, or their regime type or levels of democracy.

Now that we have some background on vectors, we can begin to think about the next level: tibbles. Tibbles are an enhanced version of data frames in R, often used to store and organize data in a tabular format. Tibbles are particularly useful when working with data that involves multiple related vectors. In other words, tibbles are named lists of **vectors of the same length**. Equal lengths are important because tibbles are organized sequentially: the first element in vector `a` will be paired with the first element in vectors `b` and `c`, the second element in `a` to the second element in `b` and `c`, and so on.

Creating Tibbles

We can create a tibble using the four vectors describing the five most populous countries in 2016 we were working with (`cname`, `pop1`, `asia`, `regime`) and the `tibble()` command. We can re-specify the information in each of our vectors using the concatenate (`c()`) function, or more simply type in the object names as we have the data saved in the environment.

```r
# Creating a tibble using vectors cname, pop1, asia, and regime
# Option 1: re-specify the information in each of our vectors using the c() function
tib1 <- tibble::tibble(
  cname = c("CHN", "IND", "USA", "IDN", "BRA"),
  pop1 = c(1379, 1331, 325, 258, 20),
  asia = c(TRUE, TRUE, F, T, F),
  regime = c("Autocracy", "FlawedDem", "FullDem", "FlawedDem", "FlawedDem"))

# Option 2 (Shortcut): type in the object names as we have the data saved in the environment
tib2 <- tibble::tibble(cname, pop1, asia, regime)
```

Both approaches yield the same result, grouping the vectors into a tibble stored in the global environment.

Ensuring Consistent Vector Lengths

When creating tibbles, it's essential that all vectors have the same length. If they don't, R will return an error. Just to demonstrate, let's see what happens when we try to put together vectors of different lengths by using `pop` instead of `pop1`.

```r
# Attempting to create a tibble with incompatible vector lengths, which will produce an error
tib_test <- tibble::tibble(
  cname = c("CHN", "IND", "USA", "IDN", "BRA"),
  pop = c(1379, 1331, 325, 258, 20, 194, 187, 161, 142, 127),
  asia = c(TRUE, TRUE, F, T, F),
  regime = c("Autocracy", "FlawedDem", "FullDem", "FlawedDem", "FlawedDem"))
```

```
## Error in `tibble::tibble()`:
## ! Tibble columns must have compatible sizes.
## * Size 5: Existing data.
```

```
## * Size 10: Column `pop`.
## i Only values of size one are recycled.
```

As we expected, the `tibble()` command produces an error saying that "tibble columns must have compatible sizes" because the `pop` vector has more elements than the other vectors. What this means, is that tibbles must have the same length.

Later on, we'll delve into working with data frames, which offer even greater flexibility for data manipulation, including merging datasets, renaming variables, and more. For now, understanding vectors and tibbles will give you a strong foundation for managing and analyzing data in R.