Week 3 Walkthrough: Vectors

James Robertson

2025-01-17

Big 3 Object Types

Coming to you ~~live~~ asynchronously from my couch, it's **ST 308 Content Walkthroughs**! As always, I'm your host, James Robertson.

This week we're all about **Object Types**. Our stars for today are the sleek **Vector**, the flexible **List**, and the handy **Data Frame**. Before we get into the technical details, let's start with some vocabulary.

- **Collection**: A group of things that we're considering all together. The objects inside my grocery cart, for example, form a **collection**. The students in this class also form a **collection**! The **object types** we're looking at today are all special types of **collections**!
- **Elements**: The *things* inside of a **Collection** are **Elements**. My **grocery collection** always has eggs and milk in it, so eggs is an **Element** and so is milk.
- **Homogeneous**: Every **element** into the same category. Going to the grocery store and only buying milk, cheese, yogurt, and ice cream would lead to me having a pretty **homogeneous** diet as everything is in the same category (dairy in this case).
- **Heterogeneous**: **Elements** can fall into different categories! Getting meat, dairy, fruits, and vegetables at the grocery store is what the surgeon general and food pyramid recommend, so a **heterogeneous** (varied) diet is generally considered ideal. A new (tastier) strain of brussels sprouts entered the mainstream in the last decade so eating leafy greens is only getting easier!

Whew, that was a lot. Now that we're all set to get into the exciting stuff, let's meet the beautiful **Vectors**!

Vectors

Name:	Vector
Related to:	Character Vector, Numeric Vector, Logical Vector, Matrix
Likes:	Order, names, math, square brackets, eating the same meal 3 times a day
Dislikes:	Variety
Bio:	Born from `c()` in a small town you haven't heard of, Vector always dreamed of settling down right in that same town and maintaining the same daily routine forever.

- **Vectors** are **ordered collections** of **homogeneous elements**.
  - *Jargon!* Put another (better) way, **vectors** consist of multiple **elements**. All these **elements** have to be of the same type (**numeric**, **character**, or **logical**) and there's a known order to them.

The order in a **vector** could be something like alphabetical (apple, banana, capers, etc.) or completely inscrutable (Greg, Stanley, Adam, Jarvis). When you make a vector, *you choose the order*! Think Jarvis should be before Adam? Make it so! Hate the letter "C" and want it at the end of the alphabet? Do it! Change your mind later? You can **reorder** vectors!

Vectors can be made with the `c()` (short for **combine** or concatenate) function by putting the elements which will combine to form the vector inside the `c()` function (i.e. having them as functional **arguments**) separated by commas, like below.

```r
aVector<-c(1, 2, 5, 7, 0)
print(aVector)
```

```
## [1] 1 2 5 7 0
```

```r
typeof(aVector)
```

```
## [1] "double"
```

Numbers go in, numbers come out! And in the same order, to boot! As you see, the `typeof()` function returns "double" which is essentially computer-speak for numbers (more technical details available in office hours!). We can actually even do math with our (numeric) vectors!

```r
print(aVector + 3)
```

```
## [1]  4  5  8 10  3
```

```r
print(aVector * 3)
```

```
## [1]  3  6 15 21  0
```

```r
print(aVector + aVector)
```

```
## [1]  2  4 10 14  0
```

But I mentioned that vectors are **homogeneous**, so what happens if we try to make a vector with both numeric *and* character elements?

```r
bVector<-c(1, 2, "three", 4)
print(bVector)
```

```
## [1] "1"     "2"     "three" "4"
```

```r
typeof(bVector)
```

```
## [1] "character"
```

When we print `bVector`, everything is in quotation marks! Why? Because all **elements** have been converted (or **coerced**) into **character** values. This is because while you and I may read "three" and see that it's the same as 3, `R` doesn't know that! And what if the value was "Sputnik"? Is there an obvious numerical value for that? The easiest thing is to just treat *everything* as text when *anything* is. **Vectors** just don't like variety, so they force everything to be the same! If your **vector** is **character** and you don't understand why, try to find the traitorous **element** turning the whole **vector** non-numeric.

Now is where everything starts to get more intense: **Indexing**. From here on the **vector** content moves on to *using* **vectors** rather than their properties. If you don't want to dive into this just yet, head on to the **Lists** section! You can always come back later.

A brief review of **indexing** first! When **indexing**, we use square brackets [ ] with another vector in the brackets like `vectorThing[indexVector]`. Indexes start at 1 and end at however many elements we have. We can do *a lot* with indexing and, in my opinion, **mastery of indexing *is* mastery of `R`**.

The videos mentioned that you can **subset** vectors by **indexing**, but *there's so much more.* Let's look at our `aVector` again. If we wanted just the first two **elements**, we could do `aVector[1:2]` or `aVector[-(1:2)]` (*Note: the colon operator generates a numeric vector from one number to the other, more details available upon request*). If we wanted the first and third **elements**, we would do `aVector[c(1,3)]`. Nothing fancy or new here. What if we wanted to reverse the order? Well, we can put the indices in reverse.

```r
print(aVector)
```

```
## [1] 1 2 5 7 0
```

```r
print(aVector[5:1])
```

```
## [1] 0 7 5 2 1
```

What if I want *everything* to be 7's? AND what if I want the vector to be longer than `aVector`? Then just keep going with the same index!

```r
print(aVector[c(4,4,4,4,4,4,4,4,4,4,4,4,4,4)])
```

```
##  [1] 7 7 7 7 7 7 7 7 7 7 7 7 7 7
```

Something else about **vectors** is that you can **name** the **elements**. You can then use these **names** for indexing as well! This is very relevant when there isn't a "natural" numerical index, such as baseball team statistics. Below I've made a **vector** of *some* of the Durham Bulls' stats, but if we were to include *every* stat the team has the numeric **index** would shift around wildly. The much easier thing is to know what you want and **index** it by **name**!

```r
bullsStats<-c(782,1318,257,29,203,723)
names(bullsStats)<-c('R','H','2B','3B','HR','RBI')
bullsStats[c('H','HR')]
```

```
##    H   HR
## 1318  203
```

And all the same rules apply!

The **indexing** rabbithole continues deeper, but we'll leave it there for the sake of "brevity" (though I've more a reputation for long-windedness). If you want to go down the **indexing** rabbithole, stop by my office hours and we can dig into it 'til the cows come home! As a last bit of trivia, **matrices** (and arrays) are just special types of **vectors**!