CHAPTER 3

Matrices

So far we've only worked with vectors, which are simple lists of values. What if you need data in rows and columns? Matrices are here to help.

A matrix is just a fancy term for a 2-dimensional array. In this chapter, we'll show you all the basics of working with matrices, from creating them, to accessing them, to plotting them.

Matrices 3.1

Let's make a matrix 3 rows high by 4 columns wide, with all its fields set to `0`.

```
> matrix(0, 3, 4)
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

You can also use a vector to initialize a matrix's value. To fill a 3x4 matrix, you'll need a 12-item vector. We'll make that for you now:

```
> a <- 1:12
```

If we print the value of `a`, we'll see the vector's values, all in a single row:

```
> print(a)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

Now call `matrix` with the vector, the number of rows, and the number of columns:

```
> matrix(a, 3, 4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

The vector's values are copied into the new matrix, one by one. You can also re-shape the vector itself into a matrix. Create an 8-item vector:

```
> plank <- 1:8
```

The `dim` assignment function sets *dim*ensions for a matrix. It accepts a vector with the number of rows and the number of columns to assign.

Assign new dimensions to `plank` by passing a vector specifying 2 rows and 4 columns (`c(2, 4)`):

```
> dim(plank) <- c(2, 4)
```

If you print `plank` now, you'll see that the values have shifted to form 2 rows by 4 columns:

```
> print(plank)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

The vector is no longer one-dimensional. It has been converted, in-place, to a matrix.

Now, use the `matrix` function to make a 5x5 matrix, with its fields initialized to any values you like.

```
> matrix(1, 5, 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    1    1    1
[2,]    1    1    1    1    1
[3,]    1    1    1    1    1
[4,]    1    1    1    1    1
[5,]    1    1    1    1    1
```

Matrix Access 3.2

Getting values from matrices isn't that different from vectors; you just have to provide two indices instead of one.

Let's take another look at our `plank` matrix:

```
> print(plank)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

Try getting the value from the second row in the third column of `plank`:

```
> plank[2, 3]
[1] 6
```

Now, try getting the value from first row of the fourth column:

```
> plank[1, 4]
[1] 7
```

As with vectors, to set a single value, just assign to it. Set the previous value to `0`:

```
> plank[1, 4] <- 0
```

You can get an entire row of the matrix by omitting the column index (but keep the comma). Try retrieving the second row:

```
> plank[2,]
[1] 2 4 6 8
```

To get an entire column, omit the row index. Retrieve the fourth column:

```
> plank[, 4]
[1] 7 8
```

You can read multiple rows or columns by providing a vector or sequence with their indices. Try retrieving columns 2 through 4:

```
> plank[, 2:4]
     [,1] [,2] [,3]
[1,]    3    5    7
[2,]    4    6    8
```

Matrix Plotting 3.3

Text output is only useful when matrices are small. When working with more complex data, you'll need something better. Fortunately, R includes powerful visualizations for matrix data.

We'll start simple, with an elevation map of a sandy beach.

It's pretty flat - everything is 1 meter above sea level. We'll create a 10 by 10 matrix with all its values initialized to 1 for you:

```
> elevation <- matrix(1, 10, 10)
```

Oh, wait, we forgot the spot where we dug down to sea level to retrieve a treasure chest. At the fourth row, sixth column, set the elevation to `0`:

```
> elevation[4, 6] <- 0
```

You can now do a contour map of the values simply by passing the matrix to the `contour` function:

```
> contour(elevation)
```

Or you can create a 3D perspective plot with the `persp` function:

```
> persp(elevation)
```

The perspective plot looks a little odd, though. This is because persp automatically expands the view so that your highest value (the beach surface) is at the very top.

We can fix that by specifying our own value for the `expand` parameter.

```
> persp(elevation, expand=0.2)
```

Okay, those examples are a little simplistic. Thankfully, R includes some sample data sets to play around with. One of these is `volcano`, a 3D map of a dormant New Zealand volcano.

It's simply an 87x61 matrix with elevation values, but it shows the power of R's matrix visualizations.

Try creating a contour map of the volcano matrix:

```
> contour(volcano)
```

Try a perspective plot (limit the vertical expansion to one-fifth again):

```
> persp(volcano, expand=0.2)
```

The `image` function will create a heat map:

```
> image(volcano)
```

Chapter 3 Completed

Here we stand on the beach, at the end of Chapter 3. What's this, buried in the sand? It's another badge!

In this chapter, we learned how to create matrices from scratch, and how to re-shape a vector into a matrix. We learned how to access values within a matrix one-by-one, or in groups. And we saw just a few of the ways to visualize a matrix's data.

None of the techniques we've used so far will help you describe your data, though. We'll rectify that in the next chapter, where we'll talk about summary statistics.

Continue