This first section is meant to give a brief introduction to data manipulation in R that will support the work in future sections. It may not be exciting, but it's incredibly exciting.

Creating matrices

There are a variety of data objects in R, including numbers, vectors, matrices, strings, and dataframes. We will mainly be working with vectors and matrices, which are quick to create and manipulate in R. The `matrix` function will create a matrix, according to the supplied arguments.

```
matrix(1:6, ncol=2)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

The `ncol` option specifies the number of columns for the output matrix; and the default behavior of `matrix` is to cycle through by column. To cycle through by rows, you'll have to set the optional argument `byrow=TRUE`.

```
matrix(1:6, ncol=3, byrow=TRUE)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Suppose we wanted to check to see if the first matrix was equal to the transpose of the second. This is clearly the case — we can see that it is. But in code, it would be cumbersome to check this condition using the previous two commands. Instead, we can assign the matrices to variables for use in subsequent manipulations. The `<-` operator assigns the arbitrary object to the supplied variable:

```
A <- matrix(1:6, ncol=2)
B <- matrix(1:6, ncol=3, byrow=TRUE)
```

The `=` operator also assigns values, with a slightly different behavior; and it is common practice to use the `=` assignment for function arguments.[1] The `==` comparison operator will yield `TRUE` or `FALSE`:

```
A == t(B)
```

```
     [,1] [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
[3,] TRUE TRUE
```

Note that `t()` will return the tranpose of the supplied matrix. Each element is checked individually, and each is identical in matrix $\mathbf{A}$ and $\mathbf{B}'$. To check the truthiness of the statement that all elements are identical, we need only to employ the `all` function:

```
all(A == t(B))
```

```
[1] TRUE
```

We can get a list of all the object currently available in memory with the `ls()` function, which is useful as the assignments begin to accumulate:

```
ls()
```

Note that without assignment, the transpose of $\mathbf{B}$, or `t(B)`, is created on the fly, remaining anonymous.

[1]See the Google style sheet for a description of other standard practices in R.

Matrix operations

Matrix muliplication in `R` is bound to `%*%`, whereas scalar multiplication is bound to `*`. Consider the product $\mathbf{BA}$:

```
B %*% A
```

```
     [,1] [,2]
[1,]   14   32
[2,]   32   77
```

The dimensions have to line up properly for matrix multiplication to be appropriately applied, otherwise `R` returns an error, as is the case with the product $\mathbf{BA}'$:

```
B %*% t(A)
```

```
Error in B %*% t(A) : non-conformable arguments
```

If scalar multiplication is applied to matrices of exactly the same dimensions, then the result is element-wise multiplication. This type of operation is sometimes called the Hadamard product, denoted $\mathbf{B} \circ \mathbf{A}'$:

```
B * t(A)
```

```
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

More common, if we want to scale all elements by a factor of two, say, we just multiply a matrix by a scalar; but note that `class(2)` must be not be `matrix` but rather `numeric` so as to avoid a non-conformable error:

```
A * 2
```

```
     [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
```

```
A * matrix(2)
```

```
Error in A * matrix(2) : non-conformable arrays
```

Consider a more complicated operation, whereby each column of a matrix is multiplied element-wise by another, fixed column. We encounter this situation frequently in time series analysis to test for parameter instability. Here, each column of a particular matrix is multiplied in-place by a fixed column of residuals. Let $\mathbf{e}$ be a vector defined as an increasing sequence of length three:

```
e <- 1:3
```

Note first that the default sequence in `R` is a column vector, and not a row vector. We would like to `apply` a function to each column of $\mathbf{A}$, specifically a function that multiplies each column in-place by $\mathbf{e}$. We must supply a 2 to ensure that the function is applied to the second dimension (columns) of $\mathbf{A}$:

```
apply(A, 2, function(x) {x * e})
```

```
     [,1] [,2]
[1,]    1    4
[2,]    4   10
[3,]    9   18
```

The function that is applied is anonymous, but it could also be bound to a variable – just as a matrix is bound to a variable:

```
whoop <- function(x) {x * e}
apply(A, 2, whoop)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    4   10
[3,]    9   18
```

We will often need to define an identity matrix of dimension n, or $\mathbf{I}_n$. This is quick using `diag`:

```
I <- diag(5)
```

There are many ways to calculate the trace of $\mathbf{I}_5$. One method has been bundled into a function, called `tr()`, that is included in a packaged called `psych` which is not included in the base distribution of `R`. We will need to grab and call the library to have access to the function, installing it with the command `install.packages("psych")`. For this, you'll need an internet connection.

```
library(psych)
tr(I)
```

```
[1] 5
```