## Matrix Operations

Addition, subtraction, and multiplication. These are all things you already know how to do with scalars. What happens, though, if you want to multiply two different matrices together. Does that simple, scalar operation still translate if you have a $2x3$ matrix and a $3x2$ matrix? If you said, "Yes!", or if words like matrix and scalar make you break out in a sweat, then this chapter is for you! Maybe you've encountered these concepts before, possibly in the first 3 weeks of a graduate statistics course; you left that class confused, angry, and wondering why you would be subjected to such nonsense. If that sounds familiar, this chapter is for you. If you found Linear Algebra easy, then you can comfortably skip this chapter and see if Chapter 2 might be where you want to start.

Matrix operations, especially multiplication, are critical for understanding what comes throughout the rest of the book. Knowing the underlying mechanics of matrix operations helps to demystify several issues that you might run into with your models. You'll frequently see model output tell you how many records were deleted due to missingness. You'll then find yourself wondering why is missingness such a problem that entire rows get deleted from your model? As we progress through the next several chapters, those issues and more will become much clearer.

Before we get into any operations, though, let's make sure we are together on some concepts.

A *scalar* is a single value. It might help if you think about a scalar as a single block @ref(fig:numbered_block)

```
scalar_example <- 1
```

```
scalar_example = 1
```

Just like we can line blocks up on the floor, we can put our scalars together to form a *vector* @ref(fig:vector_blocks). A vector is a collection of scalars with a length of **n**.

```
vector_example <- 1:5
```

This vector containing values from 1 to 5 would have a length of 5.

```
vector_example = range(0, 5)
```

Now, we can take a few of our block vectors and assemble them into a *matrix*. A matrix is a 2 dimensional collection of vectors.

@ref(fig:matrix_blocks)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

If you think about most tables you've ever seen, you'll see that the simple matrix looks remarkably familiar!

A matrix has 2 dimensions, rows and columns. When we talk about the dimensions of a matrix, we always make note of the rows first, followed by the columns. This matrix has 2 rows and 3 columns; therefore, we have a $2x3$ matrix.

## Addition

Matrix addition, along with subtraction, is the easiest concept when dealing with matrices. While it is easy to grasp, you will not find it featured as prominently as matrix multiplication.

There is one rule for matrix addition: the matrices need to have the same dimensions.

Let's check out these two matrices:

$$\underset{\text{Matrix A}}{\begin{bmatrix} 1_{11} & 2_{12} & 3_{13} \\ 4_{21} & 5_{22} & 6_{23} \end{bmatrix}} \quad \underset{\text{Matrix B}}{\begin{bmatrix} 7_{11} & 8_{12} & 9_{13} \\ 9_{21} & 8_{22} & 7_{23} \end{bmatrix}}$$

You probably noticed that we gave each scalar within the matrix a label associated with its row and column position. We can use these to see how we will produce the new matrix:

Now, we can set this up as an addition problem to produce Matrix C:

$$\underset{\text{Matrix A}}{\begin{bmatrix} 1_{11} & 2_{12} & 3_{13} \\ 4_{21} & 5_{22} & 6_{23} \end{bmatrix}} + \underset{\text{Matrix B}}{\begin{bmatrix} 7_{11} & 8_{12} & 9_{13} \\ 9_{21} & 8_{22} & 7_{23} \end{bmatrix}} = \underset{\text{Matrix C}}{\begin{bmatrix} A_{11}+B_{11} & A_{12}+B_{12} & A_{13}+B_{13} \\ A_{21}+B_{21} & A_{22}+B_{22} & A_{23}+B_{23} \end{bmatrix}}$$

Now we can pull in the real numbers:

$$\underset{\text{Matrix A}}{\begin{bmatrix} 1_{11} & 2_{12} & 3_{13} \\ 4_{21} & 5_{22} & 6_{23} \end{bmatrix}} + \underset{\text{Matrix B}}{\begin{bmatrix} 7_{11} & 8_{12} & 9_{13} \\ 9_{21} & 8_{22} & 7_{23} \end{bmatrix}} = \underset{\text{Matrix C}}{\begin{bmatrix} 1+7 & 2+8 & 3+9 \\ 4+9 & 5+8 & 6+7 \end{bmatrix}}$$

Giving us Matrix C:

$$\underset{\text{Matrix A}}{\begin{bmatrix}1_{11} & 2_{12} & 3_{13}\\ 4_{21} & 5_{22} & 6_{23}\end{bmatrix}} + \underset{\text{Matrix B}}{\begin{bmatrix}7_{11} & 8_{12} & 9_{13}\\ 9_{21} & 8_{22} & 7_{23}\end{bmatrix}} = \underset{\text{Matrix C}}{\begin{bmatrix}8 & 10 & 12\\ 13 & 13 & 13\end{bmatrix}}$$

### Subtraction

Take everything that you just saw with addition and replace it with subtraction!

Just like addition, every matrix needs to have the same dimensions if you are going to use subtraction.

Let's see those two matrices again and cast it as subtraction problem:

$$\underset{\text{Matrix A}}{\begin{bmatrix}1_{11} & 2_{12} & 3_{13}\\ 4_{21} & 5_{22} & 6_{23}\end{bmatrix}} - \underset{\text{Matrix B}}{\begin{bmatrix}7_{11} & 8_{12} & 9_{13}\\ 9_{21} & 8_{22} & 7_{23}\end{bmatrix}} = \underset{\text{Matrix C}}{\begin{bmatrix}A_{11}-B_{11} & A_{12}-B_{12} & A_{13}-B_{13}\\ A_{21}-B_{21} & A_{22}-B_{22} & A_{23}-B_{23}\end{bmatrix}}$$

And now we can substitute in the real numbers:

$$\underset{\text{Matrix A}}{\begin{bmatrix}1_{11} & 2_{12} & 3_{13}\\ 4_{21} & 5_{22} & 6_{23}\end{bmatrix}} - \underset{\text{Matrix B}}{\begin{bmatrix}7_{11} & 8_{12} & 9_{13}\\ 9_{21} & 8_{22} & 7_{23}\end{bmatrix}} = \underset{\text{Matrix C}}{\begin{bmatrix}1-7 & 2-8 & 3-9\\ 4-9 & 5-8 & 6-7\end{bmatrix}}$$

And end with this matrix:

$$\underset{\text{Matrix A}}{\begin{bmatrix}1_{11} & 2_{12} & 3_{13}\\ 4_{21} & 5_{22} & 6_{23}\end{bmatrix}} - \underset{\text{Matrix B}}{\begin{bmatrix}7_{11} & 8_{12} & 9_{13}\\ 9_{21} & 8_{22} & 7_{23}\end{bmatrix}} = \underset{\text{Matrix C}}{\begin{bmatrix}-6 & -6 & -6\\ -5 & -3 & -1\end{bmatrix}}$$

Adding and subtracting matrices in R and Python is pretty simple.

In R, we can create a matrix a few ways: with the matrix function or by row binding numeric vectors.

```
matrix_A <- rbind(1:3,
                  4:6)

# The following is an equivalent
# to rbind:
# matrix_A <- matrix(c(1:3, 4:6),
#                    nrow = 2,
#                    ncol = 3, byrow = TRUE)
```

```
matrix_B <- rbind(7:9,
                  9:7)
```

Once we have those matrices created, we can use the standard `+` and `-` signs to add and subtract:

```
matrix_A + matrix_B
```

```
     [,1] [,2] [,3]
[1,]    8   10   12
[2,]   13   13   13
```

```
matrix_A - matrix_B
```

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

The task is just as easy in Python. We will import `numpy` and then use the `matrix` method to create the matrices:

```
import numpy as np

matrix_A = np.matrix('1 2 3; 4 5 6')

matrix_B = np.matrix('7 8 9; 9 8 7')
```

Just like R, we can use `+` and `-` on those matrices.

```
matrix_A + matrix_B
```

```
matrix([[ 8, 10, 12],
        [13, 13, 13]])
```

```
matrix_A - matrix_B
```

```
matrix([[-6, -6, -6],
        [-5, -3, -1]])
```

### Transpose

As you progress through this book, you might see a matrix denoted as $A^T$; here the superscripted T stands for *transpose*. If we transpose a matrix, all we

are doing is flipping the rows and columns along the matrix's main diagonal. A visual example is much easier:

$$\underset{\text{Matrix A}}{\begin{bmatrix} 1_{11} & 2_{12} & 3_{13} \\ 4_{21} & 5_{22} & 6_{23} \end{bmatrix}} -> \underset{\text{Matrix A}^T}{\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}}$$

Like any matrix operation, a transpose is pretty easy to do when the matrix is small; you're best bet is to rely on software to do anything beyond a few rows or columns.

In R, all we need is the `t` function:

```
t(matrix_A)
```

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

In Python, we can use numpy's `transpose` method:

```
matrix_A.transpose()
```

```
matrix([[1, 4],
        [2, 5],
        [3, 6]])
```

### Multiplication

Now, you probably have some confidence in doing matrix operations. Just as quickly as we built that confidence, it will be crushed when learning about matrix multiplication.

When dealing with matrix multiplication, we have a huge change to our rule. No longer can our dimensions be the same! Instead, the matrices need to be *conformable* – the first matrix needs to have the same number of columns as the number of rows within the second matrix. In other words, the inner dimensions must match.

Look one more time at these matrices:

$$\underset{\text{Matrix A}}{\begin{bmatrix} 1_{11} & 2_{12} & 3_{13} \\ 4_{21} & 5_{22} & 6_{23} \end{bmatrix}} \cdot \underset{\text{Matrix B}}{\begin{bmatrix} 7_{11} & 8_{12} & 9_{13} \\ 9_{21} & 8_{22} & 7_{23} \end{bmatrix}}$$

Matrix A has dimensions of $2x3$, as does Matrix B. Putting those dimensions side by side – $2x3 * 2x3$ – we see that our inner dimensions are 3 and 2 and do not match.

What if we *transpose* Matrix B?

$$\begin{matrix}\text{Matrix B}^T\\ \begin{bmatrix}7_{11} & 9_{12}\\ 8_{21} & 8_{22}\\ 9_{31} & 7_{32}\end{bmatrix}\end{matrix}$$

Now we have something that works!

$$\begin{matrix}\text{Matrix A}\\ \begin{bmatrix}1_{11} & 2_{12} & 3_{13}\\ 4_{21} & 5_{22} & 6_{23}\end{bmatrix}\end{matrix} \cdot \begin{matrix}\text{Matrix B}^T\\ \begin{bmatrix}7_{11} & 9_{12}\\ 8_{21} & 8_{22}\\ 9_{31} & 7_{32}\end{bmatrix}\end{matrix} = \begin{matrix}\text{Matrix C}\\ \begin{bmatrix}. & .\\ . & .\end{bmatrix}\end{matrix}$$

Now we have a $2x3 * 3x2$ matrix multiplication problem! The resulting matrix will have the same dimensions as our two matrices' outer dimensions: $2x2$

Here is how we will get at $2x2$ matrix:

$$\begin{matrix}\text{Matrix A}\\ \begin{bmatrix}1_{11} & 2_{12} & 3_{13}\\ 4_{21} & 5_{22} & 6_{23}\end{bmatrix}\end{matrix} \cdot \begin{matrix}\text{Matrix B}^T\\ \begin{bmatrix}7_{11} & 9_{12}\\ 8_{21} & 8_{22}\\ 9_{31} & 7_{32}\end{bmatrix}\end{matrix} =$$

$$\begin{matrix}\text{Matrix C}\\ \begin{bmatrix}(A_{11} * B_{11}) + (A_{12} * B_{21}) + (A_{13} * B_{31}) & (A_{11} * B_{12}) + (A_{12} * B_{22}) + (A_{13} * B_{32})\\ (A_{21} * B_{11}) + (A_{22} * B_{21}) + (A_{23} * B_{31}) & (A_{21} * B_{12}) + (A_{22} * B_{22}) + (A_{23} * B_{32})\end{bmatrix}\end{matrix}$$

That might look like a horrible mess and likely isn't easy to commit to memory. Instead, we'd like to show you a way that might make it easier to remember how to multiply matrices. It also gives a nice representation of why your matrices need to be conformable.

We can leave Matrix A exactly where it is, flip Matrix $B^T$, and stack it right on top of Matrix A:

$$\begin{bmatrix}9_b & 8_b & 7_b\\ 7_b & 8_b & 9_b\\ \\ 1_a & 2_a & 3_a\\ 4_a & 5_a & 6_a\end{bmatrix}$$

Now, we can let those rearranged columns from Matrix $B^T$ "fall down" through the rows of Matrix A:

$$\begin{bmatrix} 9_b & 8_b & 7_b \\ \\ 1_a * 7_b & 2_a * 8_b & 3_a * 9_b \\ 4_a & 5_a & 6_a \end{bmatrix} = \begin{matrix} \text{Matrix C} \\ \begin{bmatrix} 50 & . \\ . & . \end{bmatrix} \end{matrix}$$

Adding those products together gives us 50 for $C_{11}$.

Let's move that row down to the next row in the Matrix A, multiply, and sum the products.

$$\begin{bmatrix} 9_b & 8_b & 7_b \\ \\ 1_a & 2_a & 3_a \\ 4_a * 7_b & 5_a * 8_b & 6_a * 9_b \end{bmatrix} = \begin{matrix} \text{Matrix C} \\ \begin{bmatrix} 50 & . \\ 122 & . \end{bmatrix} \end{matrix}$$

We have 122 for $C_{21}$. That first column from Matrix $B^T$ won't be used any more, but now we need to move the second column through Matrix A.

$$\begin{bmatrix} 1_a * 9_b & 2_a * 8_b & 3_a * 7_b \\ 4_a & 5_a & 6_a \end{bmatrix} = \begin{matrix} \text{Matrix C} \\ \begin{bmatrix} 50 & 46 \\ 122 & . \end{bmatrix} \end{matrix}$$

That gives us 46 for $C_{12}$.

And finally:

$$\begin{bmatrix} 1_a & 2_a & 3_a \\ 4_a * 9_b & 5_a * 8_b & 6_a * 7_b \end{bmatrix} = \begin{matrix} \text{Matrix C} \\ \begin{bmatrix} 50 & 46 \\ 122 & 118 \end{bmatrix} \end{matrix}$$

We have 118 for $C_{22}$.

Now that you know how these work, you can see how easy it is to handle these tasks in R and Python.

In R, we need to use a fancy operator: **`%*%`**. This is just R's matrix multiplication operator. We will also use the transpose function: `t`.

```
matrix_A %*% t(matrix_B)
```

```
     [,1] [,2]
[1,]   50   46
[2,]  122  118
```

In Python, we can just use the regular multiplication operator and the transpose method:

```
matrix_A * matrix_B.transpose()
```

```
matrix([[ 50,  46],
        [122, 118]])
```

You can see that whether we do this by hand, R, or Python, we come up with the same answer! While these small matrices can definitely be done by hand, we will always trust the computer to handle larger matrices.

### Inversion

You might want to think of *matrix inversion* as the reciprocal of the matrix, usually noted as $A^{-1}$. The biggest reason that we might invert a matrix is because there is no matrix division.

Inversion can only be performed on square matrices (e.g., $2x2$, $3x3$, $4x4$) and the *determinant* of a matrix cannot be 0. Since the determinant is important for finding the inverse, we should probably have an idea about how to find the determinant.

You can find a lot of examples online on how to do $2x2$ and $3x3$ matrix inversions, mostly because they are the easiest to do.

How do you know that you properly inverted your matrix? You multiply the original matrix by the inverse matrix and you will get an *identity* matrix.

We have a nice figure in Figure @ref(fig:hello), and also a table in Table @ref(tab:iris).