# Stat 2080: Statistical Modeling

## Introduction to R: 3

**Matrices**

Recall that a vector is an R object with one dimension (one row or column of values). A **matrix** is similar to a vector with the addition of another dimension (multiple rows and columns of values). A matrix is described by how many rows and columns it has. For example, a 3 x 2 matrix has 3 rows and 2 columns. This structure allows us to create data sets with multiple variables (columns) for multiple observations (rows). A matrix can be created using the matrix() function.

```
> m <- matrix(c(1.2, 3.5, 4.7, 1.8, -6.4, 5.3), nrow=3, ncol=2, byrow=F)
> m
     [,1] [,2]
[1,]  1.2  1.8
[2,]  3.5 -6.4
[3,]  4.7  5.3
```

The arguments for this function, in order are: the data that is to fill in the matrix, the number of rows it will have, the number of columns it will have, and whether the spots should be filled in by going across the rows or down the columns.

Note: You must have enough data points to fill the matrix based on the rows and columns you want, if too few or too many values are included the matrix will not be created.

The functions dim() and dimnames() may be used to determine the size of a matrix or assign names to corresponding rows/columns.

```
> dim(m)
[1] 3 2
> dimnames(m)
NULL
> dimnames(m) <- list(c("Row 1", "Row 2", "Row 3"), c("Col 1", "Col 2"))
> m
      Col 1 Col 2
Row 1   1.2   1.8
Row 2   3.5  -6.4
Row 3   4.7   5.3
```

These are useful to create names for variables and subjects directly in your matrix. You can assign names to vectors in a similar way using the names() function.

Exercise 1: Write code to produce the following matrix in R. Name each of the rows as Subject 1, Subject 2, etc. and the columns as Length, Height, and Weight.

$$\begin{bmatrix} 1 & 2 & 5 \\ 4 & 0 & 8 \\ 7 & 4 & 6 \\ 9 & 9 & 10 \end{bmatrix}$$

**Matrix Operations**

There are a variety of different matrix computations that we can perform in R, many of which are more complex mathematical techniques. Without getting into anything too complex, here are a few common functions and operators.

rbind() and cbind() allow you to add rows or columns to a matrix, or combine two whole matrices (as long as they share the same number of rows or columns)

```
> rm(m)
> mdata <- c(1.2,3.5,4.7,1.8,-6.4,5.3)
> m <- matrix(mdata, ncol=3, byrow=T)
> m
     [,1] [,2] [,3]
[1,]  1.2  3.5  4.7
[2,]  1.8 -6.4  5.3
> m1 <- rbind(1:3, m)
> m1
     [,1] [,2] [,3]
[1,]  1.0  2.0  3.0
[2,]  1.2  3.5  4.7
[3,]  1.8 -6.4  5.3
> m2 <- cbind(2, m1)
> m2
     [,1] [,2] [,3] [,4]
[1,]    2  1.0  2.0  3.0
[2,]    2  1.2  3.5  4.7
[3,]    2  1.8 -6.4  5.3
```

You can add and subtract matrices *only* when they match dimensions. When multiplying matrices, you cannot use the * symbol, that is reserved for multiplying a *scalar*, or one constant value to each element. To multiply two matrices, you must use %*%, and the dimensions must be compatible. (The number of columns in the first matrix must be the same as the number of rows in the second)

```
> m1*2
     [,1]  [,2] [,3]
[1,]  2.0   4.0  6.0
[2,]  2.4   7.0  9.4
[3,]  3.6 -12.8 10.6
> m1%*%m2
     [,1]  [,2]   [,3]  [,4]
[1,] 12.0  8.80 -10.20 28.30
[2,] 18.8 13.86 -15.43 44.96
[3,]  1.4  3.66 -52.72  3.41
```

Exercise 2: What does the function t() do for a matrix. Try using it on the previous matrix examples and using the help tab to find the term for this operator.

`t(m) =`

`t(m1) =`

`t(m2) =`

`t() stands for _____`

Note: Matrices do not get “squared”, they get multiplied by their transpose.

Exercise 3: Perform the following matrix operations.

Add the row (6, 8, 3) to the bottom of m and call it m3

Add a column $\begin{matrix}6\\3\\1\end{matrix}$ to the end of m2.

Calculate m3 – m1

Calculate m2 %*% t(m2)

**Indexing and Subscripting**

**Vectors:**

The elements of a vector can be extracted using an index vector enclosed in square brackets. This is called *subscripting*. The use of subscripts is convenient when you only want to access, view, or manipulate part of a vector.

```
> hh
 [1] 15.1 11.3  7.0  9.0  0.0  0.0  0.0  0.0 15.1 11.3  7.0  9.0
> hh[10]
[1] 11.3
> hh[1:5]
[1] 15.1 11.3  7.0  9.0  0.0
> hh[c(1,5,8)]
[1] 15.1  0.0  0.0
> hh[-1]
 [1] 11.3  7.0  9.0  0.0  0.0  0.0  0.0 15.1 11.3  7.0  9.0
> hh[-c(1,5,8)]
[1] 11.3  7.0  9.0  0.0  0.0 15.1 11.3  7.0  9.0
```

The use of negative subscripts causes all values except those specified in the index vector to be extracted. The use of logical values as indices is perhaps the most useful of all operations involving subscripts. If an index vector consisting of TRUE and FALSE values is used as a subscript, the values in the vector for which the subscript is TRUE are returned.

```
> hh > 0
 [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> hh[hh > 0]
[1] 15.1 11.3  7.0  9.0 15.1 11.3  7.0  9.0
> attach(chickwts)
> weight
 [1] 179 160 136 227 217 168 108 124 143 140 309 229 181 141 260 203 148
[18] 169 213 257 244 271 243 230 248 327 329 250 193 271 316 267 199 171
[35] 158 248 423 340 392 339 341 226 320 295 334 322 297 318 325 257 303
[52] 315 380 153 263 242 206 344 258 368 390 379 260 404 318 352 359 216
[69] 222 283 332
> index <- weight < 300
> index
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[23]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
[34]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
[45] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE
[56]  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
[67] FALSE  TRUE  TRUE  TRUE FALSE

> weight[index]
 [1] 179 160 136 227 217 168 108 124 143 140 229 181 141 260 203 148 169
[18] 213 257 244 271 243 230 248 250 193 271 267 199 171 158 248 226 295
[35] 297 257 153 263 242 206 258 260 216 222 283
> weight[weight > mean(weight)]
 [1] 309 271 327 329 271 316 267 423 340 392 339 341 320 295 334 322 297
[18] 318 325 303 315 380 263 344 368 390 379 404 318 352 359 283 332
```

Three very useful functions associated with subscripting are order(), rank(), and sort(). The order function will return a vector with the location of the smallest to largest value in the original vector. Rank will provide a vector with the rank from smallest to largest of each element in its original spot. A demonstration of these two functions will help:

```
> g <- c(7, 3.4, 6.1, 8.9, 2.5)
> g
[1] 7.0 3.4 6.1 8.9 2.5
> order(g)
[1] 5 2 3 1 4
> rank(g)
[1] 4 2 3 5 1
> sort(g)
[1] 2.5 3.4 6.1 7.0 8.9
```

Think of the order function as always returning a vector that starts at the position of the lowest value and ends with the position of the highest value. In the example above, the first element is 5, meaning the smallest value in the vector g is in the fifth position, which is correct. The next element is 2, indicating the next smallest value in g is in the second position, which would be 3.4. And so on for the rest of the values.

When using the rank function, picture the order of g fixed, and the resulting vector will give the rank from smallest to largest of each element in g. The first value is 4, meaning that the first element in g, 7, is the fourth smallest number. The next value is 2, indicating that 3.4 is the second smallest number in g.

The sort function is a little more straightforward, it will automatically rearrange the values in a vector from smallest to largest.

**Matrices**

Subscripting with matrices follows the same rules and methods as with vectors. The only difference is the additional dimension, now we have Rows and Columns. So the index after the name of a matrix will look like this [Row, Column]. When indexing a matrix it is important to remember that the first number always indicates the row(s) and the second number always indicates the column(s).

```
> mdata <- c(1.2,3.5,4.7,1.8,-6.4,5.4,-1.9,2.7,3.4,-2,7.2,4.5)
> m <- matrix(mdata, 3, 4, byrow=T)
> m
     [,1] [,2] [,3] [,4]
[1,]  1.2  3.5  4.7  1.8
[2,] -6.4  5.4 -1.9  2.7
[3,]  3.4 -2.0  7.2  4.5
> m[,2]
[1]  3.5  5.4 -2.0
```

```
> m[2,3]
[1] -1.9
> m[2, 2:3]
[1]  5.4 -1.9
> m[2:3, c(1,3)]
     [,1] [,2]
[1,] -6.4 -1.9
[2,]  3.4  7.2
> m[2,2] <- 5.5
> m
     [,1] [,2] [,3] [,4]
[1,]  1.2  3.5  4.7  1.8
[2,] -6.4  5.5 -1.9  2.7
[3,]  3.4 -2.0  7.2  4.5
```

In the last example, you can see how it is possible to replace values in an already created matrix by assigning that location a new number.

Just as in the case of a vector, if the matrix is compared to a scalar (number), a matrix of logical values (TRUE or FALSE) can be created. This matrix can be used as a subscript to index the elements of the matrix that correspond to the TRUE values.

```
> m < 0
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE FALSE FALSE
[2,]  TRUE FALSE  TRUE FALSE
[3,] FALSE  TRUE FALSE FALSE
> m[m < 0]
[1] -6.4 -2.0 -1.9
> m[m < 0] <- 0
> m
     [,1] [,2] [,3] [,4]
[1,]  1.2  3.5  4.7  1.8
[2,]  0.0  5.5  0.0  2.7
[3,]  3.4  0.0  7.2  4.5
```