MTH 3240 R Notes 4

4 Matrices

4.1 Creating and Examining Matrices

- *Matrices* are one way of storing data in a two-dimensional layout (i.e. in rows and columns). They're easily created and examined in R using the functions:

```
matrix()            # Create a matrix, from a vector, with nrow rows
                    # and ncol columns
dim()               # Returns the dimensions (number of rows and
                    # columns) of a matrix
nrow(); ncol()      # Number of rows, number of columns of a matrix
is.matrix()         # Indicates whether or not an object is a matrix
```

- Here's an example showing how to create a matrix from the vector of values `1:9` using `matrix()`:

```
x <- matrix(1:9, nrow = 3, ncol = 3)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
is.matrix(x)
```

```
## [1] TRUE
```

```
dim(x)
```

```
## [1] 3 3
```

- By default, the matrix is created by filling in its *columns*, left to right. An optional argument, `byrow`, can be set to `TRUE` if we want to create it by filling its *rows*:

```
x <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
x

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

- We can also create matrices using:

```
cbind()            # Create a matrix by "binding" vectors together
                   # in columns.
rbind()            # Like cbind(), but "binds" vectors together in
                   # rows.
```

- `cbind()` "binds" two or more vectors of the same length into *columns* of a matrix, and `rbind()` "binds" them into *rows*. For example:

```
x <- cbind(c(1,2,3), c(4,5,6), c(7,8,9))
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Section 4.1 Exercises

Exercise 1 Here's a matrix `x`:

```
x <- matrix(c(8, 8, 8, 4, 4, 4), nrow = 2, byrow = TRUE)
x

##      [,1] [,2] [,3]
## [1,]    8    8    8
## [2,]    4    4    4
```

Guess what the result of each of the following will be, then check your answers.

a) `dim(x)`

b) `nrow(x)`

c) `ncol(x)`

Exercise 2 Here's a matrix `x`:

```
##      [,1] [,2]
## [1,]    5    2
## [2,]    7    3
```

a) Create the matrix using `matrix()`. Report your R command(s).

b) Create the matrix using `cbind()`. Report your R command(s).

c) Create the matrix using `rbind()`. Report your R command(s).

4.2 General Matrix Operations

4.2.1 Matrix Arithmetic

- The operators '`+`', '`-`', '`*`', '`/`', and '`^`' operate on matrices *elementwise*. For example, using the matrices `x` and `y`,

```
x

##      [,1] [,2]
## [1,]    5    3
## [2,]    4    1

y

##      [,1] [,2]
## [1,]    1    2
## [2,]    4    6
```

we get:

```
x + y

##      [,1] [,2]
## [1,]    6    5
## [2,]    8    7
```

We can also perform arithmetic between a matrix and a single value. Again, R operates *elementwise* on the matrix:

```
x + 2

##      [,1] [,2]
## [1,]    7    5
## [2,]    6    3
```

- The built-in R functions that are vectorized, such as `log()`, `sqrt()`, `cos()`, etc. also operate elementwise on matrices.

4.2.2 Matrix Indexing Using `[ ]`

Accessing Matrix Elements, Rows, or Columns

- We access matrix elements, rows, or columns using square brackets:

```
[ , ]        # Access matrix elements via their row and column indices
             # (separated by a comma)
```

- To extract a specific element, specify its row and column indices in square brackets `[ ]`, separated by a comma. For example, using the matrix `x`:

```
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

if we want to extract the element in the 2nd row and 3rd column, we type:

```
x[2, 3]

## [1] 8
```

- An entire row is accessed by leaving the column index blank. For example the 2nd row of `x` is obtained via:

```
x[2, ]

## [1] 2 5 8
```

Likewise, an entire column is accessed by leaving the row index blank:

```
x[ , 3]

## [1] 7 8 9
```

Replacing Matrix Elements, Rows, or Columns

- The assignment operator `<-` can be used to assign a new value to a matrix element:

```
x[2, 3] <- 0
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    0
## [3,]    3    6    9
```

- Here's how to replace an entire column:

```
x[ , 3] <- c(0, 0, 0)
x

##      [,1] [,2] [,3]
## [1,]    1    4    0
## [2,]    2    5    0
## [3,]    3    6    0
```

  Replacing an entire row is similar, but the row index is specified *before* the comma in the square brackets.

Adding and Deleting Matrix Rows or Columns

- Negative indices in square brackets return all but that row or column of a matrix. For example, the following returns all but the 3rd column of `x`:

```
x[, -3]

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

- We can add a row or column to an existing matrix using `rbind()` and `cbind()`:

```
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
cbind(x, c(10, 10, 10))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   10
## [3,]    3    6    9   10
```

Rearranging Matrix Rows or Columns

- Square brackets can also be used to rearrange (permute) the rows or columns of a matrix. To do so, we specify the desired permutation before or after the comma. For example:

```
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

x[ , c(3, 1, 2)]

##      [,1] [,2] [,3]
## [1,]    7    1    4
## [2,]    8    2    5
## [3,]    9    3    6
```

  Above, the indices `c(3, 1, 2)` after the comma in square brackets `[ ]` tell R to return the 3rd column of `x` first, the 1st column second, and the 2nd column last.

Filtering on Matrices

- We can use a `"logical"` vector in square brackets to extract certain rows (or columns) from a matrix. For example:

```
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

x[c(TRUE, FALSE, TRUE), ]

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    3    6    9
```

Above, the `"logical"` vector is *before* the comma in square brackets `[ ]`, so the *rows* corresponding to `TRUE` in the `"logical"` vector are extracted from `x`.

(If it had come *after* the comma, the *columns* would have been extracted.)

- Now consider the following data set:

Title	Age	Years Experience	Paid Sick Days
Assembler	43	17	3
Machinist	22	4	7
Technician	37	6	2
Assembler	35	6	0
Engineer	31	3	0
Assembler	27	4	1
Engineer	55	16	0
Machinist	45	19	2
Assembler	35	7	3
Assembler	39	8	1
Machinist	40	14	0
Machinist	44	11	1
Technician	60	21	1

- One way to store the data is as a `"character"` vector of job titles and a numeric matrix containing the numeric values:

```
title
```

```
##  [1] "Assembler"  "Machinist"  "Technician" "Assembler"  "Engineer"   "Assembler"
##  [7] "Engineer"   "Machinist"  "Assembler"  "Assembler"  "Machinist"  "Machinist"
## [13] "Technician"
```

```
values
```

```
##       Age Years Experience Paid Sick Days
##  [1,]  43               17              3
##  [2,]  22                4              7
##  [3,]  37                6              2
##  [4,]  35                6              0
##  [5,]  31                3              0
##  [6,]  27                4              1
##  [7,]  55               16              0
##  [8,]  45               19              2
##  [9,]  35                7              3
## [10,]  39                8              1
## [11,]  40               14              0
## [12,]  44               11              1
## [13,]  60               21              1
```

(The column headings were added to the `values` matrix using the `colnames()` function.)

- To extract from `values` just the rows corresponding to Machinists, we can type:

```
values[title == "Machinist", ]   # title=="Machinist" is a "logical" vector
```

```
##      Age Years Experience Paid Sick Days
## [1,]  22                4              7
## [2,]  45               19              2
## [3,]  40               14              0
## [4,]  44               11              1
```

Note that the expression `title == "Machinist"` before the comma in square brackets `[ ]` is actually a `"logical"` vector:

```
title == "Machinist"
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE
```

Its `TRUE`s indicate which rows of the `values` matrix to extract.

Section 4.2 Exercises

Exercise 3 Consider the matrices `x` and `y`:

```
x <- matrix(c(1, 4, 9, 16), nrow = 2, ncol = 2)
x
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

```
y <- matrix(c(2, 1, 2, 1), nrow = 2, ncol = 2)
y
```

```
##      [,1] [,2]
## [1,]    2    2
## [2,]    1    1
```

Guess what the result of each of the following will be, then check your answers.

a) `x + 1`

b) `sqrt(x)`

c) `x * y`

Exercise 4 Consider the following matrix:

```
x <- matrix(c(3, 5, 8, 2, 9, 7, 6, 1, 4), nrow = 3, ncol = 3)
x
```

```
##      [,1] [,2] [,3]
## [1,]    3    2    6
## [2,]    5    9    1
## [3,]    8    7    4
```

Recall that indices specified *before* the comma in square brackets `[ ]` refer to *rows*, and indices *after* the comma refer to *columns*. Guess what the result of each of the following will be, then check your answers.

a) `x[1, 3]`

b) `x[1, ]`

c) `x[, 3]`

d) `x[, -1]`

Exercise 5 Consider the following matrix:

```
x <- matrix(1:6, nrow = 2, ncol = 3)
x
```

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

Recall that a vector of indices in square brackets `[ ]` can be use to rearrange rows (or columns) of a matrix. Guess what the result of each of the following will be, then check your answers.

a) `x[, c(3, 1, 2)]`

b) `x[c(2, 1), ]`

Exercise 6 Consider the following matrix:

```
x <- matrix(c(4, 2, 7, 8, 1, 9), nrow = 3, ncol = 2)
x

##      [,1] [,2]
## [1,]    4    8
## [2,]    2    1
## [3,]    7    9
```

Guess what the result of the following will be, then check your answer.

```
x[c(TRUE, TRUE, FALSE), ]
```

Exercise 7 Consider the following data set:

Gender	Age	Number of Siblings	Number of Pets	Age of Oldest Pet
Male	11	1	2	5
Male	8	2	1	3
Female	12	1	2	2
Female	9	0	0	NA
Male	12	3	0	NA
Female	10	4	1	6

These data can be stored as a `"character"` vector and numeric matrix:

```
gender <- c("Male", "Male", "Female", "Female", "Male", "Female")
values <- cbind(c(11, 8, 12, 9, 12, 10), c(1, 2, 1, 0, 3, 4),
                c(2, 1, 2, 0, 0, 1), c(5, 3, 2, NA, NA, 6))
```

To add column headings to the `values` matrix, type:

```
colnames(values) <- c("Age", "Number of Siblings", "Number of Pets",
                      "Age of Oldest Pet")
```

Guess what the following command returns, then check your answer.

```
values[gender == "Male", ]
```

4.3 The `apply()` Function

- Sometimes we need to apply a function separately to each row (or each column) of a matrix. To do so, we use:

```
apply()         # Apply a function separately to each row (or each
                # column) of a matrix
```

- `apply()` has three main arguments, `x`, a matrix, `FUN`, the function to be applied to the rows (or columns) of `x`, and `MARGIN` (`MARGIN = 1` for rows and `MARGIN = 2` for columns).
- For example, below, `apply()` is used to compute the sum of each row:

```
x

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

apply(X = x, MARGIN = 1, FUN = sum)

## [1] 12 15 18
```

Note that `apply()` returns a vector, each element of which is the sum of the corresponding row in `x`.

Section 4.3 Exercises

Exercise 8 Consider the following matrix `x`:

```
x <- matrix(c(8, 6, 3, 6, 5, 7, 2, 1, 9, 4, 7, 6, 8, 3, 4, 3),
            nrow = 4, ncol = 4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    8    5    9    8
## [2,]    6    7    4    3
## [3,]    3    2    7    4
## [4,]    6    1    6    3
```

Guess what the result of each of the following will be, then check your answers.

a) `apply(x, MARGIN = 1, FUN = min)`

b) `apply(x, MARGIN = 2, FUN = min)`

Exercise 9 R has several built-in datasets, one of which is `USPersonalExpenditure`, a matrix with five rows and five columns consisting of U.S. personal expenditures (in

billions of dollars) in the categories: food and tobacco, household operation, medical and health, personal care, and private education for the years 1940, 1945, 1950, 1955 and 1960.

Type

```
USPersonalExpenditure
```

to view the matrix.

a) Use `apply()`, with `FUN = mean`, to find the mean expenditure for each of the five categories (rows). Report your R command(s). **Hint**: What should `MARGIN` be set to?

b) Use `apply()`, with `FUN = sum`, to find the total (sum) of the expenditures for each of the five years (columns). Report your R command(s). **Hint**: See the hint for Part *a*.