Introduction to R and RStudio

Part 2: Extended Introduction to R

Rob Cribbie
Department of Psychology
York University
Although with most software packages learning about a “matrix” is unnecessary, in R many of the concepts explored while learning about matrices apply to dealing with multiple variables, dataset operations, etc.

We will start with a brief discussion about matrices before moving more specifically into a discussion about datasets.
Matrices

> x <- matrix(data=c(1,4,5,2,4,5,6,2), nrow=2, ncol=4)
> x

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

> x[2,4]
[1] 2

> rowSums(x)
[1] 16 13

Notice that by default the Matrix was created “by Columns” (1st col, 2nd col, etc.)

x[row, column]
Matrices, cont’d

> x <- matrix(data=c(1,4,5,2,4,5,6,2), nrow=2, ncol=4, byrow=TRUE)
> x

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

> mean(x[,1])
[1] 3

> var(x[,3])
[1] 0.5

This time the data was entered “by row”

When a row or column is blank, all elements are used
Matrices, cont’d

> x <- rbind(x, apply(X=x, MARGIN=2, FUN=mean))
> x

\[
\begin{array}{cccc}
[1,] & 1.0 & 5.0 & 4.0 & 6 \\
[2,] & 4.0 & 2.0 & 5.0 & 2 \\
[3,] & 2.5 & 3.5 & 4.5 & 4 \\
\end{array}
\]

> rownames(x) <- c(1,2,'Mean')
> x

\[
\begin{array}{cccc}
[1] & 1.0 & 5.0 & 4.0 & 6 \\
2 & 4.0 & 2.0 & 5.0 & 2 \\
Mean & 2.5 & 3.5 & 4.5 & 4 \\
\end{array}
\]
Matrix Shortcuts

\[ a <- \text{matrix}(c(2,1,2,4,3,5), \text{nrow}=2) \]

\[
> a
\]

\[
\begin{bmatrix}
[1,] & 2 & 2 & 3 \\
[2,] & 1 & 4 & 5 \\
\end{bmatrix}
\]

\[ b <- \text{matrix}(c(2,4,6,1,1,3), \text{nrow}=3) \]

\[
> b
\]

\[
\begin{bmatrix}
[1,] & [2] \\
[1,] & 2 & 1 \\
[2,] & 4 & 1 \\
[3,] & 6 & 3 \\
\end{bmatrix}
\]

When elements are left out of a function, the function uses the defaults or figures out the required number of columns.

Defaults to "byrow = FALSE"... defaults can be found in the help file.

Figures out that there needs to be 3 columns.
Introduction to Matrix Operations

> t(a)

\[
\begin{array}{cc}
[1,] & [2,] \\
[1,] & 2 & 1 \\
[2,] & 2 & 4 \\
[3,] & 3 & 5 \\
\end{array}
\]

Transpose (reverse rows and columns)

> a*b

Error in a * b : non-conformable arrays

> a%*%b

\[
\begin{array}{cc}
[1,] & [2,] \\
[1,] & 30 & 13 \\
[2,] & 48 & 20 \\
\end{array}
\]

* = Scalar multiplication

Matrix Multiplication
You can enter your data directly into R

```r
> x <- rep(x = c(1, 2), each = 4)
## same as > x <- c(1, 1, 1, 1, 2, 2, 2, 2)
> y <- c(3, 4, 5, 2, 6, 6, 5, 4)
> data <- data.frame(x, y)
> data
  x y
1 1 3
2 1 4
3 1 5
4 1 2
5 2 6
6 2 6
7 2 5
8 2 4
```

The first column contains the row numbers.
The most important difference is that a dataset can contain a mixture of non-numeric and numeric variables, where a matrix cannot (all elements must be of the same form)

```r
> mat <- matrix(data=c("a","b","c","d", 1, 2, 3, 5), nrow=1)
> mat
[1,]   "a"  "b"  "c"  "d"   "1"  "2"  "3"  "5"
```

Numeric elements were converted to character elements
> a <- data.frame(matrix(c(1,3,1,4,2,6,2,8),ncol=2,byrow=TRUE))

> a
      X1 X2
1     1  1  3
2     1  4
3     2  6
4     2  8

> X1
Error: object 'X1' not found

> a$X1  # variable 'X1' in dataset 'a'
[1] 1 1 2 2

The variable names (X1,X2) were automatically assigned

The $ separates the dataset name from the variable name
You can also relate variables to each other without them being in the same matrix/dataset

```r
> iv<-c(1,1,1,1,2,2,2,2)
> dv<-rnorm(8)
> dv
[1] 1.4671261 -1.0878009 -1.0487529 -0.521493
[5] 1.9040232  0.5586128  2.8512241 -1.0348406

> mean(dv[iv==1])
[1] -0.2977303
```
The Problem of “=” and “==”

> x <- c(2, 5, 4, 2, 8, 6)
> y <- c(1, 1, 1, 2, 2, 2)

> x[y == 1]
[1] 2 5 4

Note that `x[y == 1]` represents the first instance of `x` where `y = 1`, whereas `x[y == 1]` represents ‘all x where y = 1'
Opening Existing Datasets

- You can also read in external data files.
- To read in a dataset from SPSS there are multiple options, here are a couple of the popular options:
  - Use the ‘read.spss’ command in the ‘foreign’ package (not recommended as it is very fussy with variable names, formats, and you need to data.frame the dataset after creating it).
  - Convert the SPSS file to a comma separated file (.csv) and then open the file using read.csv(file=" ")
    - This is recommended since it is easy to convert to .csv in SPSS and is very flexible.
In general, it is recommended that you convert datasets to .csv before opening them in R

- Almost all spreadsheet and statistical software packages will let you save your file as a .csv file

To browse your directories for a file use the ‘file.choose()’ option (with a read statement)

- newdata<-read.csv(file.choose())

Also note that if you are specifying the exact file name, that the slashes are backward to Windows

- > newdata<-read.csv(file="C:/My Documents/Robs Work/SCS/R Course/testdata2.csv")
Example

In this example a researcher is interested in exploring whether sex (male, female) or community size (small, large) relate to the amount of recycling performed by individuals.
After running this line a ‘select file’ box appears

‘head’ function gives the first six lines of the dataset

Variable Names

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>male</td>
<td>small</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>male</td>
<td>small</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>male</td>
<td>large</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>male</td>
<td>large</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>female</td>
<td>small</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>female</td>
<td>small</td>
<td>3</td>
</tr>
</tbody>
</table>

> newdat<-read.csv(file.choose())
> head(newdat)

> names(newdat)

[1] "Sex" "Commssize" "Recycle"
Editing a Dataset

- The following command will pop up a spreadsheet that allows you to change dataset values, variable names, etc.
  > edit (dat1)
- However, when you close the spreadsheet nothing is saved because you are not saving (putting, <-) your changes into an object
- In order to save our changes we must specify the name of the dataset that will receive the changes
Editing a Dataset

> dat2 <- edit(dat1)
In some instances we want to work with just a subset of the original dataset
  ◦ This could be done with indexing, but subsetting is very straightforward

> dat2 <- subset(dat1, Sex=='Male')
> dat2

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Male</td>
<td>Small</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Male</td>
<td>Large</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Male</td>
<td>Small</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Male</td>
<td>Large</td>
<td>4</td>
</tr>
</tbody>
</table>
Basic Statistics on a Dataset

- `mean(newdat$Recycle)`
  
  `[1] 4.25`

- `mean(newdat$Recycle[newdat$Sex=="male" & newdat$Commsize=="small"])`
  
  `[1] 3`

- `var(newdat$Recycle[newdat$Sex=="female" & newdat$Commsize=="large"])`
  
  `[1] 4.5`
Simple Plot

- `> hist(newdat$Recycle, col="blue")`
Check normality and variance homogeneity assumptions

- `> shapiro.test(newdat$Recycle)`
  Shapiro–Wilk normality test, data: newdat$Recycle
  W = 0.959, p-value = 0.8006

- `> library(car)`
- `# ‘car’ is a package that I previously installed`

- `> leveneTest(newdat$Recycle,newdat$Sex)`
  Levene's Test for Homogeneity of Variance (center = median)
  Df  F value  Pr(>F)
  group 1 1.5 0.2666
  6
What if we wanted to verify that the distributions in each commsize are normal in shape (this would not make much sense with n=4)

- `> tapply(newdat$Recycle,newdat$Commsize,shapiro.test)`

$large
Shapiro–Wilk normality test data: X[[1L]]
W = 0.9714, p-value = 0.85

$small
Shapiro–Wilk normality test data: X[[2L]]
W = 0.9714, p-value = 0.85

‘tapply’ is very handy any time you need to look at a statistic (e.g., mean) across multiple levels or cells of other variables
One of the nice features of R is that when a variable is designated as a “factor”, R performs some operations that are either safeguards or helpful to the user.

A couple examples are:
- Not permitting numeric operations on factors
  - E.g., mean of factor
- Automatically assigning dummy variables in regression
Working with Factors

- > is.factor(newdat$Sex)
  [1] TRUE

- > levels(newdat$Sex)
  [1] "female" "male"

- > cor(newdat$Sex, newdat$Recycle)
  Error in cor(newdat$Sex, newdat$Recycle) :
    'x' must be numeric

- > library(ltm) #package I installed
- > biserial.cor(y=newdat$Sex, x=newdat$Recycle)
  [1] 0.4493585
Saving an R Dataset or Matrix

> a

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

> write.csv(a, file='c:/Documents and Settings/Rob/My Documents/RCourse/newdat.csv', row.names=FALSE)
> write.csv(a, file='newdat.csv', row.names=FALSE)

Don’t forget the forward slashes (/)

If you have already set your working directory

It is important to add row.names=FALSE if you want to open the dataset in other software programs
One of the main advantages of R is its flexibility. For example, R makes it very easy to write your own functions. Here is a (completely unnecessary) function to take the mean of a set of observations:

```r
robsmean <- function (x) {
  result <- sum(x)/length(x)
  return(result)
}
> robsmean(c(3,2,4))
[1] 3
```