EPIB-613 - Data Entry and Manipulation in R

Today we will cover:

1. Some useful data manipulation commands (rep, seq, cbind, rbind)
2. Reading in data files from txt, csv, Excel, STATA and SPSS files
3. Creating complete case data sets
4. Functions and missing data
5. Writing (saving) txt or csv files

1 Some data manipulation commands

R has some convenient and quick ways to create sequences of numbers that contain certain patterns. For example, the `rep` command repeats numbers as many times as indicated.

```
> rep(2, 10)
 [1] 2 2 2 2 2 2 2 2 2 2
> c(rep(2,10), rep(NA,3), 5, 1:10)
 [1]  2  2  2  2  2  2  2  2  2  2 NA NA NA  5  1  2  3  4  5  6  7  8  9 10
```

Similarly, the `seq` function creates strings of numbers with equal jumps between numbers:

```
> range1 <- seq(0, 1, by = 0.05)
> range1
 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65
[15] 0.70 0.75 0.80 0.85 0.90 0.95 1.00
> range2 <- seq(-5, 3, by = 0.5)
> range2
 [1] -5.0 -4.5 -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5
[15]  2.0  2.5  3.0
> range3 <- c(range1, range2)
> range3
 [1]  0.00  0.05  0.10  0.15  0.20  0.25  0.30  0.35  0.40  0.45  0.50  0.55
[13]  0.60  0.65  0.70  0.75  0.80  0.85  0.90  0.95  1.00 -5.00 -4.50 -4.00
[25] -3.50 -3.00 -2.50 -2.00 -1.50 -1.00 -0.50  0.00  0.50  1.00  1.50  2.00
[37]  2.50  3.00
```

The seq command has many uses, one main use we will see later is in creating x-axes for graphics.

Other useful commands are `cbind` and `rbind`, which, respectively, take a sequence of vector, matrix or data frames arguments and combine by columns or rows. Here are some examples:

First define a few objects to experiment on:

```
> x<-rnorm(10, mean=3, sd=2)
> x
 [1] 1.1747263 3.7092580 6.2271502 2.4388199 5.4313524 0.9660352 3.9457464
 [8] 1.1852031 1.1195718 2.2440396
```

```
>  y<-rbinom(10, size=1, prob=0.4)
> y
 [1] 1 0 1 1 0 0 0 0 0 1
> z<- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=T)
> z
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Now lets see what `cbind` and `rbind` do to these objects:

```
> cbind(x,y)
              x y
 [1,] 1.1747263 1
 [2,] 3.7092580 0
 [3,] 6.2271502 1
 [4,] 2.4388199 1
 [5,] 5.4313524 0
 [6,] 0.9660352 0
 [7,] 3.9457464 0
 [8,] 1.1852031 0
 [9,] 1.1195718 0
[10,] 2.2440396 1

> rbind(x,y)
      [,1]     [,2]    [,3]    [,4]     [,5]      [,6]     [,7]     [,8]
x 1.174726 3.709258 6.22715 2.43882 5.431352 0.9660352 3.945746 1.185203
y 1.000000 0.000000 1.00000 1.00000 0.000000 0.0000000 0.000000 0.000000
      [,9]    [,10]
x 1.119572 2.244040
y 0.000000 1.000000

> cbind(z,z)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    1    2    3
[2,]    4    5    6    4    5    6
[3,]    7    8    9    7    8    9

> rbind(z,z)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

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

> cbind(x,z)
            x
[1,] 1.174726 1 2 3
[2,] 3.709258 4 5 6
[3,] 6.227150 7 8 9
Warning message:
In cbind(x, z) :
  number of rows of result is not a multiple of vector length (arg 1)
```

Note the truncation and warning message in the last example, because x and z are of different sizes. Only the first three items of x were used.

2 Reading in data files from txt, csv, Excel, STATA and SPSS files

The following commands can read in data from files that are stored in specific formats:

- Text files: Plain ascii files can be read in via the read.table command.
- CSV files: Comma delimited files can be read in with the read.csv command.
- Excel files: Install and load the package called xlsReadWrite. The function read.xls reads in a standard excel formatted file.
- SPSS files can be read in with the read.spss command.
- STATA files can be read in with the function read.dta.

Many of these functions have some options for handling whether the first line contains data or is a header line with variable names, and so on. Here is one example of a series of sometimes useful arguments:

```
> args(read.xls)
```

```
function (file, colNames = TRUE, sheet = 1, type = "data.frame",
    from = 1, rowNames = NA, colClasses = NA, checkNames = TRUE,
    dateTime = "numeric", naStrings = NA, stringsAsFactors = default.stringsAsFactors())
```

3 Creating complete case data sets

It is extremely easy to create a data set of complete cases in R. Suppose you have a data frame called x that has some variables with missing items. Then run these commands:

```
> ok <- complete.cases(x)
> x.no.missing <- x[ok,]
```

The data set x.no.missing is now a data frame of complete cases only. The names ”ok” and ”x.no.missing” are of course arbitrary. Here is an example:

```
> y<- c(rep(1, 3), rep(NA, 3), rep(2, 3))
> y
[1]  1  1  1 NA NA NA  2  2  2
> z<- c(1,2,3,4,5,6,7, NA, 9)
> z
[1]  1  2  3  4  5  6  7 NA  9
> x<- data.frame(y,z)
> x
   y  z
1  1  1
2  1  2
3  1  3
4 NA  4
5 NA  5
6 NA  6
7  2  7
8  2 NA
9  2  9
> ok <- complete.cases(x)
> x.no.missing <- x[ok,]
> x.no.missing
  y z
1 1 1
2 1 2
```

```
3 1 3
7 2 7
9 2 9
```

Note that only rows with no missing values have been kept in the final data set.

4 Functions and missing data

The universal symbol for missing data in R is NA. Most functions have some way to deal with missing data, but by default some may fail, others may work.

Keeping the above definitions for x, y, and z, here are some examples:

```
> mean(y)
[1] NA

> mean(y, na.rm=T)
[1] 1.5

> summary(x)
       y              z        
 Min.   :1.0   Min.   :1.000  
 1st Qu.:1.0   1st Qu.:2.750  
 Median :1.5   Median :4.500  
 Mean   :1.5   Mean   :4.625  
 3rd Qu.:2.0   3rd Qu.:6.250  
 Max.   :2.0   Max.   :9.000  
 NA's   :3.0   NA's   :1.000  

> summary(lm(y ~ z))

Call:
lm(formula = y ~ z)

Residuals:
      1       2       3       7       9 
 0.1186 -0.0339 -0.1864  0.2034 -0.1017 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)   0.7288     0.1438   5.068   0.0148 *
z             0.1525     0.0268   5.692   0.0107 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.1841 on 3 degrees of freedom
  (4 observations deleted due to missingness)
Multiple R-squared: 0.9153,     Adjusted R-squared: 0.887
F-statistic:  32.4 on 1 and 3 DF,  p-value: 0.01075

> cor(y,z,use="complete.obs")
[1] 0.9566892
```

Check the help for each function to see how NAs are handled each time. Note that by default mean and cor do not work with missing data, but by setting an option, both can be made to work, essentially by ignoring the missing data items. Regression needs no special option, but you should realize that by default, all rows (patients) with one or more missing data items are deleted.

5 Writing (saving) txt or csv files

Storing data frames as text or csv files is easily done. The command write.table prints its required argument x (after converting it to a data frame if it is not one nor a matrix) to a file. Similarly, write.csv saves to a csv file. Taking the data frame from section 4, x, we can type:

```
> args(write.table)
function (x, file = "", append = FALSE, quote = TRUE, sep = " ",
    eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
    qmethod = c("escape", "double"))
> write.table(x, file="c:\\temp\\x.txt")
> write.csv(x, file = "c:\\temp\\x.csv")
```

You can check that the appropriate files get saved in the correct formats. The text file looks like this:

```
      "y" "z"
"1" 1   1
```

```
"2" 1   2
"3" 1   3
"4" NA  4
"5" NA  5
"6" NA  6
"7" 2   7
"8" 2   NA
"9" 2   9
```

The csv file looks like this (as expected, commas replace spaces compared to the previous text file):

```
"","y","z"
"1",1,1
"2",1,2
"3",1,3
"4",NA,4
"5",NA,5
"6",NA,6
"7",2,7
"8",2,NA
"9",2,9
```