**Creating and Managing Tables in R**

To read a text file (e.g. “data.txt”) containing a table:

my.table = read.delim(“data.txt”, sep=”\t”, stringsAsFactors=FALSE)

Or

my.table = read.table(“data.txt”, sep=”\t”, stringsAsFactors=FALSE)

The 2 commands are pretty much equivalent. The commands above assume that columns in your tables are separated by ‘tabs’, as in the case you copy an excel table and paste it into a simple txt file (for example a TextWrangler doc). If the columns are separated by something else, then change the parameter sep

For example:

my.table = read.table(“data.txt”, sep=”,”, stringsAsFactors=FALSE)

# if columns separated by commas

my.table = read.table(“data.txt”, sep=” ”, stringsAsFactors=FALSE)

# if columns separated by a white space

By default this command interprets the first row of your table as the header, i.e. it must contains the names of columns. For simplicity, make sure the column names **do not contain spaces** and **do not start with a number**.

If you do not have a header:

my.table = read.table(“data.txt”, sep=”\t”, stringsAsFactors = FALSE, header = FALSE)

If you want the first column to be interpreted as the name of each row then:

my.table = read.table(“data.txt”, sep=”\t”, stringsAsFactors=FALSE, row.names = 1)

Once you have your table loaded, check if it is correctly formatted. E.g.:

My.table[1:10, 1:10]

Will display the first 10 rows and the first 10 columns of your table

A table (or data.frame as it is called in R) is indexed as a matrix (see my previous notes on matrices) so **[ x , y ]** indicates the row **x** and column **y**.

You can use special character to sub-select:

# display the first 2 rows and all columns

my.table[1:2 , ]

# display the row 1 and 4 and all columns

my.table[c(1,4) , ]

# display the row from 1 to 6 and columns 1, 5, 6, and 7

my.table[ 1:6, c(1, 5:7) ]

# or

my.table[ 1:6, c(1, 5, 6, 7) ]

If your table has a header, you can access its columns by name.

To display the column names of your table:

colnames(my.table)

Let’s imagine that your table has 4 columns called: ‘C1, ‘C2, ‘C3, ‘C4, then the command above will give you this:

> colnames(my.table)

[1] "C1"  "C2" "C3" "C4"

If you want to change the names, for example instead of ‘C4 you want ‘C\_gene, then:

colnames(my.table) = c("C1", "C2", "C3", "C\_gene")

or

colnames(my.table)[4] = “C\_gene”

# i.e. you ONLY change the 4th element of the vector returned by colnames(my.table)

Now, to look at all the values in the column “C1” you can use the name of the column preceded by the character $:

my.table$C1

To look at the first 5 values of the column ‘C1:

my.table$C1[1:5]

Now, let’s extract a sub-table from the original table

# to create a sub-table with only the first 5 rows and the columns “C1” and “C2” you can do:

sub.table = my.table[ 1:5, c(“C1”, “C2”) ]

# or

sub.table = my.table[ 1:5, 1:2 ]

Now the object sub.table is table with only the first 5 rows and 2 columns.

Sub-setting can be useful in a large table to extract only the rows (or columns) of certain element.

For example, let’s assume you have a large table of beta-values where each row is a probe and the second column contains the gene name corresponding to that probe, something like:

probeID gene sample1 sample2 sample3 …

CG1 MYC 0.5 0.2 0.6

CG2 MYC 0.3 0.1 0.6

CG3 IRS1 0.5 0.2 0.6

CG4 IRS1 0.3 0.1 0.6

CG5 IRS1 0.5 0.2 0.6

CG6 MYCN 0.3 0.1 0.6

CG7 MYCN 0.3 0.1 0.6

Let’s say that you want to extract ONLY the rows containing probes mapping to IRS1, then:

sub.table = my.table[ my.table$gene == “IRS1”, ]

Now sub.table will be:

probeID gene sample1 sample2 sample3 …

CG3 IRS1 0.5 0.2 0.6

CG4 IRS1 0.3 0.1 0.6

CG5 IRS1 0.5 0.2 0.6

Let’s re-examine how the sub-setting work:

# inside the square brackets you can select the rows and columns that you want (as in the examples before)

# the value BEFORE the comma indicate which rows you want

# the value AFTER the comma indicate which columns you want

# you can select rows and columns by imposing a condition

sub.table =

my.table [ <here you input the condition - only rows that satisfy the condition are selected>     ,  ]

Hence:

my.table$gene == “IRS1”

Is a condition satisfied ONLY by rows where the values of the column “gene” are equal to “IRS1”

Note that we used $gene to access the column “gene” and used the symbol == which indicates “is it equal?” (the symbol “=” is used instead to assign a value)

You can impose all kind of conditions, just be careful to use the correct expression.

To indicate 2 conditions that BOTH need to be satisfied, use “&” (i.e. logical AND)

To indicate 2 conditions such that at least one of the two needs to be satisfied, use “|” (i.e. logical OR)

Examples:

# you want to extract ONLY the rows containing probes mapping to IRS1 and  where sample1 is less than 0.5, then:

sub.table = my.table[ my.table$gene == “IRS1” & my.table$sample1 < 0.5 , ]

**Output:**

probeID gene sample1 sample2 sample3 …

CG4 IRS1 0.3 0.1 0.6

#  you want to extract ONLY the rows containing probes mapping to either IRS1 or MYCN

sub.table = my.table[ my.table$gene == “IRS1” | my.table$gene == “MYCN” , ]

**Output:**

probeID gene sample1 sample2 sample3 …

CG3 IRS1 0.5 0.2 0.6

CG4 IRS1 0.3 0.1 0.6

CG5 IRS1 0.5 0.2 0.6

CG6 MYCN 0.3 0.1 0.6

CG7 MYCN 0.3 0.1 0.6

# you want to extract ONLY the rows containing probes mapping to either IRS1 and ONLY the first 3 samples

sub.table = my.table[ my.table$gene == “IRS1” , c(3, 4, 5) ]

# note that column 1 is ‘probeID’ and column 2 is ‘gene’, hence the first 3 samples are in columns 3, 4, and 5