BINF6200/8200: Statistics for Bioinformatics

Lab 1-2
To install the test package in your home computer:

--Make sure you have installed R first http://cran.r-project.org
--within R, do the following

```r
> source("http://bioconductor.org/biocLite.R")

> biocLite("multtest")  #download the multtest package to your system
```

To test if the download is successful and to use the packages, try these steps:

```r
> library(multtest)  #load the package

> data(golub)  #make the data available for usage
```

If you want to install other packages from Bioconductor:

```r
> biocLite("packagename")

> biocLite (groupName="all")  #not recommended!!!
```
# what version of R do I have?
> R.version.string

# what R packages do I have?
> library()

# How to install a new package?
> install.packages("package_name")

# How to remove packages?
> remove.packages("package_name")

# where are R installed?
> path.packages()
> .libPaths() #default path for installing packages
Introduction to R

To use R

• Start R
• Type a command and press Enter
• R executes this command (often printing the result)
• R waits for more input
• Type `q()` to exit

Getting help:
--If you know the function name

```
help(sum) or ?sum
help(sd) or ?sd
?plot
```

--If you don’t know the function name

```
help.search ("standard deviation")
```
Where am I?

Where to find the file?

> `getwd()`
[1] "/Users/jguo4"

> `setwd("/Users/jguo4/Documents/R_test")`

> `getwd()`
Basic Arithmetic Operations in R

Try:
> mytest <- rnorm(50)
> mytest

***It is perfectly ok if you don’t understand these commands at this point!!
Built-in Functions in R

\[ \log(a), \sqrt{b} \ldots \]

> \log(100)

> \log(100, base=10)

> \log(100, 10)

> \log10(100)

!!!It is very important to know the default parameters of the R functions. If you are not sure, think about:

? 

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt(x)</td>
<td>Square root of x</td>
</tr>
<tr>
<td>abs(x)</td>
<td>Absolute value</td>
</tr>
<tr>
<td>sin(x), tan(x), cos(x)</td>
<td>Trigonometric functions</td>
</tr>
<tr>
<td>exp(x)</td>
<td>Exponential</td>
</tr>
<tr>
<td>log(x)</td>
<td>Natural logarithm</td>
</tr>
<tr>
<td>log10(x)</td>
<td>Base 10 logarithm</td>
</tr>
<tr>
<td>ceiling(x)</td>
<td>Closest integer not less than x</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Closest integer not greater x</td>
</tr>
<tr>
<td>round(x)</td>
<td>Closest integer to the element</td>
</tr>
</tbody>
</table>
Variable Names and Assignments

Variable names can be very flexible. However,
• cannot start with a digit, cannot start with a period followed by a digit
• names are case-sensitive
• avoid using system names in R, e.g. c, q, t, C, D, F, I, T, diff….

R has symbolic variables which can be assigned values.

' <- ' operator ***no space between “<” and “-”

> x <- 3 #Assign the value 3 to x

> y <- -0.5 #Assign the value -0.5 to y

> x
[1] 3

> y
[1] -0.5
Variable Names and Assignments

Type the this command and see what happens:

> mya <- a

Then try the next two:

> myc <- "a"
> mys <- "This is BINF6200!"

R is an object-oriented language, meaning everything in R is an object belonging to some class

> class(x)
> class(myc)
> class(mys)
> class(rnorm)

"mode()" function

> remove(x)  or > rm(x)    # to remove the object
**mode() vs. class()**

Difference between `mode()` and `class()`

`mode()` is a classification of objects according to their basic structure, including numeric, character, logical, list and function etc.

*An object has one and only one mode.*

`class()` is a property assigned to an object that determines how generic functions operate with it. **If an object does not have specific classes assigned to it, e.g. a simple numeric vector, its class is usually the same as its mode.**

```r
> x <- 5:100
> mode(x)
[1] "numeric"
> class(x)
[1] "integer"

> y <- matrix(1:12, nrow=3, byrow=T)
> mode(y)
[1] "numeric"
> class(y)
[1] "matrix"
```
The elementary data types in R are all vectors.

1. The `c()` construct is used to create vectors “concatenate”

   ```r
   # numerical vector
   > weight <- c(60, 72, 57, 90, 95, 72) OR >weight <- scan()
   > weight
   [1] 60 72 57 90 95 72
   
   # character vector
   > binf6200 <- c("JT", "Guo", "Jun", "Tao") OR >binf6200 <- scan(what=" ")
   > binf6200
   [1] "JT" "Guo" "Jun" "Tao"
   
   # logical vector
   > biostudent <- c(F, F, T, F)
   > biostudent
   [1] FALSE FALSE  TRUE FALSE
   ```

   Vector elements must all have the same mode!!!
vector in R

Other functions that can be used to create vectors:

2. `seq()` “sequence”, used for equidistant series of numbers

```r
> seq(5, 10)
[1] 5 6 7 8 9 10

> seq(4, 16, 2)          # the same as > seq(4, 16, by=2)
[1] 4 6 8 10 12 14 16

> seq(0, 4, 0.5)
[1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0

# If the step size is 1, there is a simple way to generate the numbers
> 5:10
[1] 5 6 7 8 9 10       # Which is the same as `seq(5, 10)`.

Very useful for graphics!!
```

Try `seq(10, 5), 10:5`
Other functions that can be used to create vectors (continued):

3. `rep()` “replicate”, used to generate repeated values

```r
> rep(1, 3)  # If the second argument is a number n, repeat the first argument n times
[1] 1 1 1

> rep(1:3, 3)  # the same as rep(1:3, times=3)
[1] 1 2 3 1 2 3 1 2 3

> rep(1:3, each=3)
[1] 1 1 1 2 2 2 3 3 3

# If the second argument is a vector of numbers
> rep(8:10, 1:3)
[1]  8  9  9 10 10 10
```

Very useful for generating groups!!
vector in R

Other functions that can be used to create vectors (continued):

3. \texttt{rep()} “replicate”, used to generate repeated values

**Exercise:**

1. Generate a vector of numbers with 8 “1” and 10 “2”.

2. Generate a vector of 5 “man” and 8 “woman”

3. Generate an array with one 1, two 2s, three 3s, two 4s, and one 5
vector in R

```r
#original vector
> weight <- c(60, 72, 57, 90, 95, 72)
> weight
[1] 60 72 57 90 95 72

#find the specific values
> weight[2]
[1] 72

> weight[3:5] #the same as weight[c(3, 4, 5)]
[1] 57 90 95

#remove an entry
> weight[-2]
[1] 60 57 90 95 72

> weight
[1] 60 72 57 90 95 72

> weight <- weight[-2] #very important!
> weight
[1] 60 57 90 95 72
```

vector in R

# original vector
> weight <- c(60, 72, 57, 90, 95, 72)

> weight
[1] 60 72 57 90 95 72

# to add an entry at the end
> weight <- c(weight, 80)

> weight
[1] 60 72 57 90 95 72 80

> weight <- append(weight, 100)

> weight
[1] 60 72 57 90 95 72 80 100

# to add an entry in the middle
> weight <- c(weight[1:3], 300, weight[4:8])

> weight
[1] 60 72 57 300 90 95 72 80 100
Common arithmetic operations and mathematical functions work element-wise on vectors, and produce another vector:

```r
> weight <- c(60, 72, 57, 90, 95, 72)

> weight
[1] 60 72 57 90 95 72

> height <- c(1.75, 1.8, 1.65, 1.9, 1.74, 1.91)

> height^2

> bmi <- weight/height^2

> bmi

> log(bmi)
```
Note: If the lengths of the two vectors are not the same, the shorter one is recycled.

```r
> 1:20 + c(0, 4)
[1]  1  6  3  8  5 10  7 12  9 14 11 16 13 18 15 20 17 22 19 24
```

#The command adds 0 to all the odd elements and 4 to all the even elements of 1:20

Exercise: How to get a subset of vector?

For example, 1, 3, 5 of the weight vector
Remember that vector elements must all have the same mode!!!

What if we want to create a more complex data structure, e.g. with numbers and characters?

```r
> test<-c(c(60, 72), c("c", "d"))
> test
[1] "60" "72" "c" "d"
> class(test)
[1] "character"

> test<-list(c(60, 72), c("c", "d"))
> test
[[1]]
[1] 60 72

[[2]]
[1] "c" "d"

> class(test)
[1] "list"
```
vector vs. list

Difference between `vector` and `list`

```r
> xx <- list(analystI = c(22.3, 23, 51), names = c("Tom", "Tommy", "Tony"))
> xx
$analystI
[1] 22.3 23.0 51.0
$names
[1] "Tom" "Tommy" "Tony"

> xx$names
[1] "Tom" "Tommy" "Tony"

> xx[[1]][2]
[1] 23

> xx[[1]]
[1] 22.3 23.0 51.0

> xx <- c(analystI = c(22.3, 23, 51), names = c("Tom", "Tommy", "Tony"))
> xx
analystI1 analystI2 analystI3 names1 names2 names3
 "22.3"   "23"    "51"    "Tom"  "Tommy"  "Tony"
> class(xx)
[1] "character"
```
Construct a data matrix

```r
> mydata <- matrix(1:6, nrow=2, ncol=3)
> mydata
     [,1] [,2] [,3]
[1,]   1   3   5
[2,]   2   4   6

> mydata2 <- matrix(1:6, nrow=2, byrow=T)
> mydata2
     [,1] [,2] [,3]
[1,]   1   2   3
[2,]   4   5   6
```

Default: by column
Data Matrices in R

```r
> sam_matrix <- matrix(1:12, nrow=3, byrow=T)
> sam_matrix
[1,]  1  2  3  4
[2,]  5  6  7  8
[3,]  9 10 11 12

# add column and row names
> rownames(sam_matrix) <- c("A", "B", "C")
> colnames(sam_matrix) <- c("W", "X", "Y", "Z")

> sam_matrix
   W X Y Z
A 1 2 3 4
B 5 6 7 8
C 9 10 11 12

# How to get the data in the matrix?
> sam_matrix[1,]
> sam_matrix[,3]
> sam_matrix[2,3]
```
Data Matrices in R

Given the table below, you want to do an association assay to test if the color and the shape are independent, how to get the data into R?

<table>
<thead>
<tr>
<th></th>
<th>Yellow</th>
<th>Green</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rounded</td>
<td>30</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Wrinkled</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

1. You can use the method from `matrix()` function

2. Use `cbind()` or `rbind()` function:

```r
> plant_seed <- rbind(c(30, 20, 50), c(10, 20, 30))

> dimnames(plant_seed) <- list(c("Rounded", "Wrinkled"), c("Yellow", "Green", "Red"))

> plant_seed
     Yellow Green Red
Rounded    30    20    50
Wrinkled   10    20    30
```
Matrix Transposition in R

```r
> plant_seed
   Yellow  Green  Red
Rounded  30     20   50
Wrinkled 10     20   30

> t(plant_seed)
   Rounded  Wrinkled
   Yellow   30     10
  Green     20     20
    Red     50     30

function t() flips the matrix

--the rows become columns
--the columns become rows
```
Data frames:
The most common data-storage format
--each column stores a variable, usually each row for one subject in paired data
--Columns have names for easy reference

```r
> aa <- c(31.4, 37.0, 44.0, 28.8, 59.9, 37.6)
> bb <- c(28.1, 37.1, 40.6, 27.3, 58.4, 38.9)
> cc <- data.frame(analystI=aa, analystII=bb)

> cc
  analystI analystII
 1     31.4      28.1
 2     37.0      37.1
 3     44.0      40.6
 4     28.8      27.3
 5     59.9      58.4
 6     37.6      38.9
```
Variables are accessible using the $notation:

\begin{verbatim}
> cc$analystI
[1] 31.4 37.0 44.0 28.8 59.9 37.6

> cc$analystII
[1] 28.1 37.1 40.6 27.3 58.4 38.9
\end{verbatim}

A convenient way to access variables is to use \texttt{attach()} function:

\begin{verbatim}
> attach(cc)
> analystI
[1] 31.4 37.0 44.0 28.8 59.9 37.6

> analystII
[1] 28.1 37.1 40.6 27.3 58.4 38.9
\end{verbatim}
The data frame “cc” is placed in the system’s search path

To remove it from the path use

> detach(cc)
Q: 1. How to add a 7\textsuperscript{th} sample (22.3, 24.2) to the data frame?

2. How to add data from analystIII (30.2, 36.8, 42.5, 28.0, 58.9, 38.2)?
Methods for Data Input-data frame

```r
> cc
   analystI analystII
1     31.4      28.1
2     37.0      37.1
3     44.0      40.6
4     28.8      27.3
5     59.9      58.4
6     37.6      38.9

#How to add a 7th sample (22.3, 24.2) to the data frame?
> rbind(cc, data.frame(analystI=22.3, analystII=24.2))

> rbind(cc, list(analystI=22.3, analystII=24.2))

#How to add data from analystIII (30.2, 36.8, 42.5, 28.0, 58.9, 38.2) ?
> cbind(cc, list(analystIII=c(30.2, 36.8, 42.5, 28.0, 58.9, 38.2)))

> cc

> cc <- rbind(cc, data.frame(analystI=22.3, analystII=24.2))
```
### Getting Data From a File

**file name:**
AffinityCorrelation.txt

**file location:**
http://bioinfozen.uncc.edu/BINF6200_13

<table>
<thead>
<tr>
<th>experiment</th>
<th>new_potential</th>
<th>old_potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8.9</td>
<td>-183.62</td>
<td>-215.32</td>
</tr>
<tr>
<td>-7.82</td>
<td>-109.52</td>
<td>-33.46</td>
</tr>
<tr>
<td>-12.9</td>
<td>-180.87</td>
<td>-86.29</td>
</tr>
<tr>
<td>-6.8</td>
<td>-8.55</td>
<td>-8.2</td>
</tr>
<tr>
<td>-7.9</td>
<td>-87.57</td>
<td>-42.17</td>
</tr>
<tr>
<td>-9.11</td>
<td>-111.99</td>
<td>68.64</td>
</tr>
<tr>
<td>-8.15</td>
<td>-104.24</td>
<td>17.83</td>
</tr>
<tr>
<td>-6</td>
<td>-25.78</td>
<td>3.98</td>
</tr>
<tr>
<td>-9.32</td>
<td>-226.38</td>
<td>-274.52</td>
</tr>
<tr>
<td>-5.5</td>
<td>-90.75</td>
<td>-29.78</td>
</tr>
<tr>
<td>-9.1</td>
<td>0.02</td>
<td>-62.98</td>
</tr>
<tr>
<td>-10.7</td>
<td>-239.76</td>
<td>-143.15</td>
</tr>
<tr>
<td>-10.8</td>
<td>-176.88</td>
<td>-134.4</td>
</tr>
<tr>
<td>-6</td>
<td>-41.25</td>
<td>-43.53</td>
</tr>
<tr>
<td>-8</td>
<td>-83.21</td>
<td>114.57</td>
</tr>
<tr>
<td>-9.3</td>
<td>-234.75</td>
<td>-74.93</td>
</tr>
<tr>
<td>-9.15</td>
<td>-157.59</td>
<td>-43.9</td>
</tr>
<tr>
<td>-8.73</td>
<td>-74.28</td>
<td>-163.67</td>
</tr>
<tr>
<td>-8.7</td>
<td>-167.74</td>
<td>-48.8</td>
</tr>
<tr>
<td>-10.8</td>
<td>-234.8</td>
<td>-147.97</td>
</tr>
<tr>
<td>-7.8</td>
<td>-220.85</td>
<td>-177.56</td>
</tr>
<tr>
<td>-10.9</td>
<td>-184.53</td>
<td>-269.12</td>
</tr>
<tr>
<td>-7.64</td>
<td>-122.14</td>
<td>-22.58</td>
</tr>
<tr>
<td>-8.2</td>
<td>-89.93</td>
<td>-60.44</td>
</tr>
<tr>
<td>-6.8</td>
<td>-86.16</td>
<td>-85.73</td>
</tr>
<tr>
<td>-8.6</td>
<td>-161.65</td>
<td>19.57</td>
</tr>
<tr>
<td>-8.15</td>
<td>-86.11</td>
<td>-90.01</td>
</tr>
<tr>
<td>-7.5</td>
<td>-186.64</td>
<td>-15.58</td>
</tr>
<tr>
<td>-8.8</td>
<td>-290.86</td>
<td>-253.43</td>
</tr>
<tr>
<td>-8.7</td>
<td>-155.31</td>
<td>-186.41</td>
</tr>
</tbody>
</table>
Getting Data From a File

> mydata <- read.table("http://bioinfozen.uncc.edu/BINF6200_13/AffinityCorrelation.txt", header=TRUE)

> mydata

1. Do you have a row of variable names (headers) in the first row? If yes, set `header=TRUE`. If not, try file `AffinityCorrelation1.txt`.

2. What character do you use to separate columns of data? Is it a tab, a space, a comma? e.g. `sep = "","`.

3. Are there any missing data? Try file `AffinityCorrelation2.txt`.

4. Where is the data located? Local or on a server?
## Getting Data From a File

**no header**

<table>
<thead>
<tr>
<th>AffinityCorrelation1.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8.9</td>
</tr>
<tr>
<td>-7.82</td>
</tr>
<tr>
<td>-12.9</td>
</tr>
<tr>
<td>-6.8</td>
</tr>
<tr>
<td>-7.9</td>
</tr>
<tr>
<td>-9.11</td>
</tr>
<tr>
<td>-8.15</td>
</tr>
<tr>
<td>-6</td>
</tr>
<tr>
<td>-9.32</td>
</tr>
<tr>
<td>-5.5</td>
</tr>
<tr>
<td>-9.1</td>
</tr>
<tr>
<td>-10.7</td>
</tr>
<tr>
<td>-10.8</td>
</tr>
<tr>
<td>-6</td>
</tr>
<tr>
<td>-8</td>
</tr>
<tr>
<td>-9.3</td>
</tr>
<tr>
<td>-9.15</td>
</tr>
<tr>
<td>-8.73</td>
</tr>
<tr>
<td>-8.7</td>
</tr>
<tr>
<td>-10.8</td>
</tr>
<tr>
<td>-7.8</td>
</tr>
<tr>
<td>-10.9</td>
</tr>
<tr>
<td>-7.64</td>
</tr>
<tr>
<td>-8.2</td>
</tr>
<tr>
<td>-6.8</td>
</tr>
<tr>
<td>-8.6</td>
</tr>
<tr>
<td>-8.15</td>
</tr>
<tr>
<td>-7.5</td>
</tr>
<tr>
<td>-8.8</td>
</tr>
<tr>
<td>-8.7</td>
</tr>
</tbody>
</table>

**missing data**

<table>
<thead>
<tr>
<th>AffinityCorrelation2.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>experiment</td>
</tr>
<tr>
<td>-8.9</td>
</tr>
<tr>
<td>-7.82</td>
</tr>
<tr>
<td>-12.9</td>
</tr>
<tr>
<td>-6.8</td>
</tr>
<tr>
<td>-7.9</td>
</tr>
<tr>
<td>-9.11</td>
</tr>
<tr>
<td>-8.15</td>
</tr>
<tr>
<td>-6</td>
</tr>
<tr>
<td>-9.32</td>
</tr>
<tr>
<td>-5.5</td>
</tr>
<tr>
<td>-9.1</td>
</tr>
<tr>
<td>-10.7</td>
</tr>
<tr>
<td>-10.8</td>
</tr>
<tr>
<td>-6</td>
</tr>
<tr>
<td>-8</td>
</tr>
<tr>
<td>-9.3</td>
</tr>
<tr>
<td>-9.15</td>
</tr>
<tr>
<td>-8.73</td>
</tr>
<tr>
<td>-8.7</td>
</tr>
<tr>
<td>-10.8</td>
</tr>
<tr>
<td>-7.8</td>
</tr>
<tr>
<td>-10.9</td>
</tr>
<tr>
<td>-7.64</td>
</tr>
<tr>
<td>-8.2</td>
</tr>
<tr>
<td>-6.8</td>
</tr>
<tr>
<td>-8.6</td>
</tr>
<tr>
<td>-8.15</td>
</tr>
<tr>
<td>-7.5</td>
</tr>
<tr>
<td>-8.8</td>
</tr>
<tr>
<td>-8.7</td>
</tr>
</tbody>
</table>
matrix vs. data.frame

Difference between `matrix` and `data.frame`

- Matrices are for data of the *same type* (remember the difference between `vector` and `list`?)
- Use data frames if *columns (variables) have different types* (numeric/character/logical etc.).

<table>
<thead>
<tr>
<th></th>
<th>Yellow</th>
<th>Green</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rounded</td>
<td>30</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Wrinkled</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Names</th>
<th>Height</th>
<th>glasses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>6</td>
<td>T</td>
</tr>
<tr>
<td>Tommy</td>
<td>5.6</td>
<td>F</td>
</tr>
<tr>
<td>Tony</td>
<td>5.8</td>
<td>T</td>
</tr>
</tbody>
</table>
Golub Data in Bioconductor

```r
> library(multtest)  # load the package
> data(golub)        # make the data available for usage

golub.cl            # a numeric vector for tumor class, 0 for ALL, 1 for AML

golub.gnames        # a matrix of gene index, name and ID

golub               # a matrix of gene expression values

> golub.cl
> golub.gnames[1030:1050,]
> nrow(golub)
> ncol(golub)
> dim(golub)
```
Getting Golub Data in Bioconductor

To get the expression values of gene CCND3 Cyclin D3 in ALL and AML patients. They are in row 1042

> gene1042ALL <- golub[1042, 1:27]

> gene1042AML <- golab[1042, 28:38]

If we are interested in one gene (for example CD33), how can we find the row number?

> grep ("CD33", golub.gnames[, 2])
Where am I?

Save it as a plain text file test1.R

```r
pdf("binf.pdf")  # set graphical output file
hist(rnorm(100))  # generate 100 random numbers and plot histogram
dev.off()         # close the graphical file
```

Where to find the file?

```r
> getwd()
[1] "/Users/jguo4"
> setwd("/Users/jguo4/Documents/R_test")
> getwd()
```

There are two ways to run the code:

1. **Interactive mode in R**
   > `source("test1.R")`

2. **Batch mode in terminal**
   `$R CMD BATCH test1.R`

In both cases, a histogram file will be generated.