Chapter 1: R Essentials

Jixiang Wu, Associate Professor of Quantitative Genetics/Biostatistics

This R document and many others are developed and available for students who take the special topics course: Data Analysis with R. Transfering the course materials beyond students' course work and/or research work may not be allowed.

1. An overgrown calculator

```
2+2

## [1] 4

exp(-2)

## [1] 0.1353353

rnorm(15)

##  [1] -0.89794353 -0.10450145  0.14829913  2.68685028 -0.73044872
##  [6] -0.44580328 -0.12526030  2.01371623  0.45779102  1.85954804
## [11]  1.05530878  1.35607121 -0.05180772 -1.60210453 -0.45091087
```

2. Assignments

```
x=2
x

## [1] 2

x+x

## [1] 4

x^2

## [1] 4

y=x+5
y

## [1] 7
```

3. Vectorized arithmetic

```r
weight=c(60,72,57,90,95,72)
weight

## [1] 60 72 57 90 95 72

height=c(1.75,1.80,1.65,1.90,1.74,1.91)
height

## [1] 1.75 1.80 1.65 1.90 1.74 1.91

bmi=weight/height^2
bmi

## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630

sum(weight)

## [1] 446

sum(weight)/length(weight)

## [1] 74.33333

xbar=sum(weight)/length(weight)
xbar

## [1] 74.33333

weight-xbar

## [1] -14.333333  -2.333333 -17.333333  15.666667  20.666667  -2.333333

(weight-xbar)^2

## [1] 205.444444   5.444444 300.444444 245.444444 427.111111   5.444444

sum((weight-xbar)^2)

## [1] 1189.333

sum((weight-xbar)^2)/(length(weight)-1)

## [1] 237.8667

mean(weight)

## [1] 74.33333

sd(weight)

## [1] 15.42293
```

4. Standard procedures

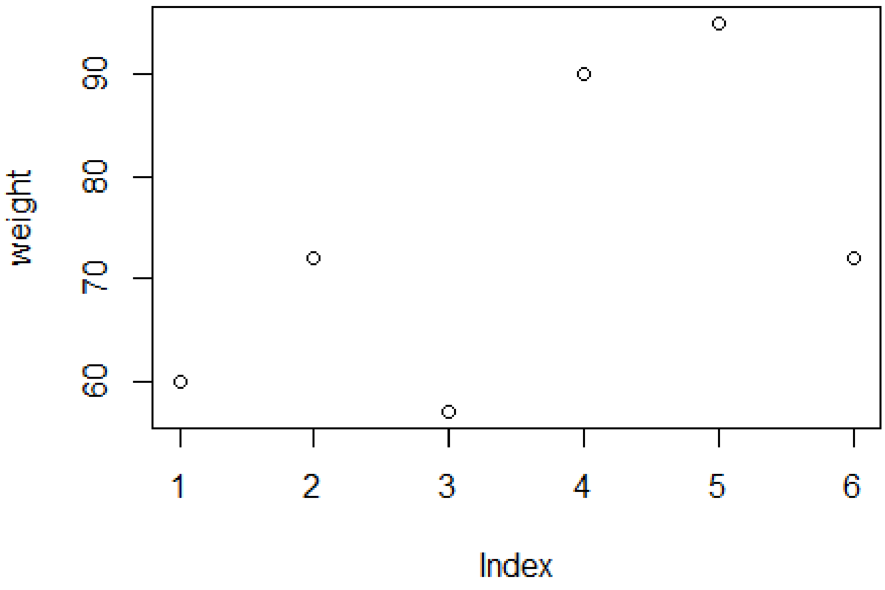
```
t.test(weight,mu=2.25)
```

```
##
##  One Sample t-test
##
## data:  weight
## t = 11.4484, df = 5, p-value = 8.907e-05
## alternative hypothesis: true mean is not equal to 2.25
## 95 percent confidence interval:
##  58.14796 90.51870
## sample estimates:
## mean of x
##  74.33333
```

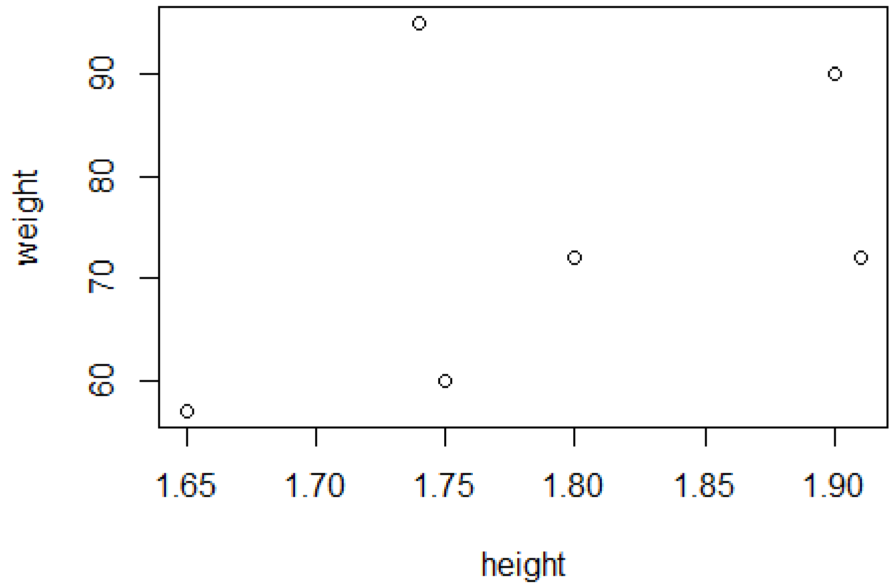
5. Graphics

A little bit of taste of data visualization is mentioned here. More graphics details will be available later in this semester.

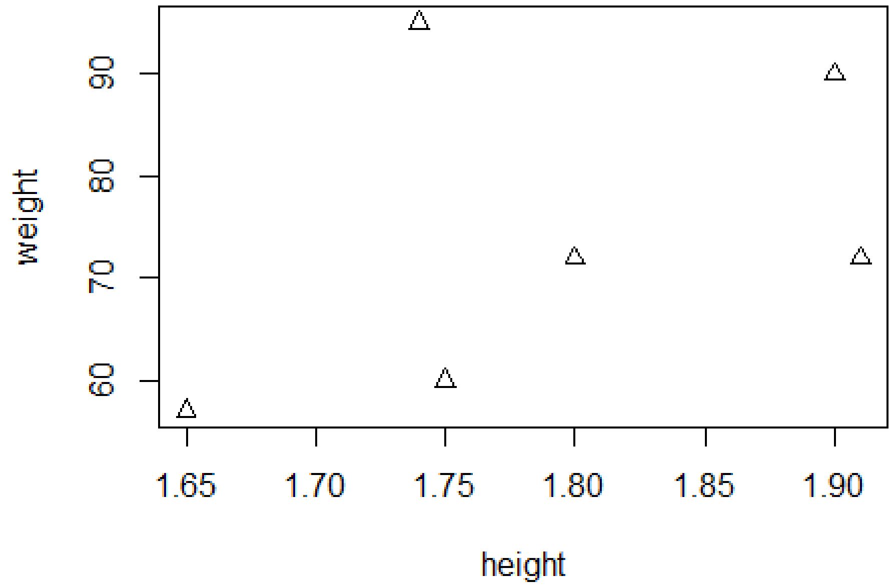
```
plot(weight)
```



```
plot(height,weight)
```



```r
plot(height,weight,pch=2)
```



6. Functions and arguments

For example plot(height,weight) and mean()

7. Vectors

Read pages 12-13. Vectors can be numbers, factors, strings, and/or logicals values.

```
c("Student1","Student2","Student3")

## [1] "Student1" "Student2" "Student3"

c('Student1','Student2','Student3')

## [1] "Student1" "Student2" "Student3"
```

Above two expressions generate the same vector.

```
c(1,1,2,1.3)

## [1] 1.0 1.0 2.0 1.3

c(T,T,F,F)

## [1]  TRUE  TRUE FALSE FALSE

bmi>25

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

Functions that create vectors

Use "c", which is "concatenate".

```
c(12,56,46)

## [1] 12 56 46

x=c(1,2,3)

y=c(10,14)

c(x,y,18)

## [1]  1  2  3 10 14 18
```

Name a vector

```
student=c(jack="Student1",tim="Student2",saha="Student3")
student

##       jack        tim       saha 
## "Student1" "Student2" "Student3"
```

```
names(student)*

## [1] "jack" "tim"  "saha"
```

Generating seqences

```
x=0:10
x

##  [1]  0  1  2  3  4  5  6  7  8  9 10

x=15:5
x

##  [1] 15 14 13 12 11 10  9  8  7  6  5

x=seq(1,10,0.1)
x

##  [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3
## [15]  2.4  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7
## [29]  3.8  3.9  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1
## [43]  5.2  5.3  5.4  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5
## [57]  6.6  6.7  6.8  6.9  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9
## [71]  8.0  8.1  8.2  8.3  8.4  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3
## [85]  9.4  9.5  9.6  9.7  9.8  9.9 10.0

x=seq(6,4,-0.2)
x

##  [1] 6.0 5.8 5.6 5.4 5.2 5.0 4.8 4.6 4.4 4.2 4.0
```

Generating repeats

Sometimes we need to generate repeats of numbers or characters such as used in design of experiments. By doing so, we can use the function rep. For example, if we want to generate five 4s, we can use the following R codes.

```
x=rep(5,4)
x

## [1] 5 5 5 5
```

We can also generate two 1 to 4 sequence using the following R codes.

```
x=rep(1:4,2)
x

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

The following R code generates two repeated numbers from 1 to 4.

```
x=rep(1:4,each=2)
x
```

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

A little complicated cases like the following:

```
x=rep(1:4,each=2,times=3)
x

##  [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4

x=rep(1:4,c(4,1,2,3))
x

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

Generating factor levels

The function gl (generate levels) is useful when you want to encode vectors of factor levels.

```
x=gl(4,3)
x

##  [1] 1 1 1 2 2 2 3 3 3 4 4 4
## Levels: 1 2 3 4
```

Here is a code when we want the whole pattern repeated twice.

```
x=gl(4,3,24)
x

##  [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3 3 4 4 4
## Levels: 1 2 3 4
```

If we want test for the factor levels, rather than numbers, the following codes are very helpful.

```
Temp=gl(2,2,24,labels=c("Low","High"))
Soft=gl(3,8,24,labels=c("Hard","Medium","Soft"))
User=gl(2,4,24,labels=c("N","Y"))
Brand=gl(2,1,24,labels=c("X","M"))

dat=data.frame(Temp,Soft,User,Brand)
dat

##    Temp   Soft User Brand
## 1   Low   Hard    N     X
## 2   Low   Hard    N     M
## 3  High   Hard    N     X
## 4  High   Hard    N     M
## 5   Low   Hard    Y     X
## 6   Low   Hard    Y     M
## 7  High   Hard    Y     X
## 8  High   Hard    Y     M
## 9   Low Medium    N     X
## 10  Low Medium    N     M
## 11 High Medium    N     X
## 12 High Medium    N     M
## 13  Low Medium    Y     X
## 14  Low Medium    Y     M
## 15 High Medium    Y     X
## 16 High Medium    Y     M
## 17  Low   Soft    N     X
## 18  Low   Soft    N     M
## 19 High   Soft    N     X
## 20 High   Soft    N     M
## 21  Low   Soft    Y     X
## 22  Low   Soft    Y     M
## 23 High   Soft    Y     X
## 24 High   Soft    Y     M
```

Infinity

```
3/0
```

```
## [1] Inf
```

```
is.finite(20)
```

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

```
is.infinite(20)
```

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

```
is.infinite(Inf)
```

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

Missing values: NA

```
y=c(1:4,NA,7)
y
```

```
## [1]  1  2  3  4 NA  7
```

```
is.na(y)
```

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

```
y[is.na(y)]
```

```
## [1] NA
```

```
y[!is.na(y)]
```

```
## [1] 1 2 3 4 7
```

Calculating mean with missing value

```
mean(y)

## [1] NA

mean(y,na.rm=TRUE)

## [1] 3.4
```

Finding the missing values

The which function is very commonly used.

```
miss.id=which(is.na(y))
miss.id

## [1] 5
```

Vectors and subscripts

A vector is a variable with one or more than one values of the same type. For example:

```
y=1:10
y

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

We can look at the type of y by using the function class:

```
class(y)

## [1] "integer"
```

The length of the vector y by using the function length:

```
length(y)

## [1] 10
```

The mean of the vetor y:

```
mean(y)

## [1] 5.5
```

The variance of the vector y:

```
var(y)

## [1] 9.166667
```

The minimum value of the vector y:

```
min(y)
```

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

The position of the minimum value of the vector y:

```
which(y==min(y))

## [1] 1
```

The maximum value of the vector y:

```
max(y)

## [1] 10

which(y==max(y))

## [1] 10
```

Quantiles of the vector y:

```
quantile(y)

##    0%   25%   50%   75%  100% 
##  1.00  3.25  5.50  7.75 10.00
```

Extracting elements of a vector using subscripts

```
y=c(4,5,6,4,5,6,7,8)
y

## [1] 4 5 6 4 5 6 7 8
```

The forth element:

```
y[4]

## [1] 4
```

The elements for position 1,3,4:

```
id=c(1,3,4)
y[id]

## [1] 4 6 4
```

Drop the second element from the vector y:

```
y[-2]

## [1] 4 6 4 5 6 7 8
```

Drop the last element from the vector y:

```
y[-length(y)]

## [1] 4 5 6 4 5 6 7
```

Obtain the first three element from the vector y:

```
y[1:3]

## [1] 4 5 6
```

Naming elelments with a vector

```
names=paste("y",1:length(y),sep="")
names(y)=names
y

## y1 y2 y3 y4 y5 y6 y7 y8
##  4  5  6  4  5  6  7  8

x=rpois(200,1.2)
table(x)

## x
##  0  1  2  3  4  5
## 57 79 49  4  9  2

#Removing the element names by using as.vector
as.vector(table(x))

## [1] 57 79 49  4  9  2
```

Working with logical subscripts

```
mean(x)

## [1] 1.175

var(x)

## [1] 1.150126

sum(x)

## [1] 235

mean(x<1)

## [1] 0.285

var(x<1)

## [1] 0.204799

sum(x<1)

## [1] 57
```

Sorting a vector

From the smallest to the largest:

```
y=rnorm(10)
y

##  [1]  1.5545998 -0.3617998 -0.5398638  1.0276314  0.8973074  0.5160712
##  [7]  0.3978116 -0.5786195  1.5303154 -0.7585115

sort(y)

##  [1] -0.7585115 -0.5786195 -0.5398638 -0.3617998  0.3978116  0.5160712
##  [7]  0.8973074  1.0276314  1.5303154  1.5545998
```

From the largeste to the smallest:

```
rev(sort(y))

##  [1]  1.5545998  1.5303154  1.0276314  0.8973074  0.5160712  0.3978116
##  [7] -0.3617998 -0.5398638 -0.5786195 -0.7585115
```

The largest three elements in vector y:

```
rev(sort(y))[1:3]

## [1] 1.554600 1.530315 1.027631
```

The positions for minimum and maximum values:

```
which.max(y)

## [1] 1

which.min(y)

## [1] 10
```

8. Matrix

```
x=1:12
dim(x)=c(3,4)
x

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

matrix(1:12,nrow=3)

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
matrix(1:12,nrow=3,byrow=T)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```
x=matrix(1:12,nrow=3,byrow=T)
rownames(x)=LETTERS[1:3]
x
```

```
##   [,1] [,2] [,3] [,4]
## A    1    2    3    4
## B    5    6    7    8
## C    9   10   11   12
```

```
t(x)
```

```
##      A B  C
## [1,] 1 5  9
## [2,] 2 6 10
## [3,] 3 7 11
## [4,] 4 8 12
```

```
cbind(A=1:4,B=5:8,C=9:12)
```

```
##      A B  C
## [1,] 1 5  9
## [2,] 2 6 10
## [3,] 3 7 11
## [4,] 4 8 12
```

```
rbind(A=1:4,B=5:8,C=9:12)
```

```
##   [,1] [,2] [,3] [,4]
## A    1    2    3    4
## B    5    6    7    8
## C    9   10   11   12
```

9. Factors

```
pain=c(0,3,2,1)
pain
```

```
## [1] 0 3 2 1
```

```
fpain=factor(pain)
fpain
```

```
## [1] 0 3 2 1
## Levels: 0 1 2 3
```

```
fpain=factor(pain,levels=0:3)
fpain
```

```
## [1] 0 3 2 1
## Levels: 0 1 2 3
```

```
levels(fpain)=c("none","mild","medium","severe")
fpain
```

```
## [1] none   severe medium mild
## Levels: none mild medium severe
```

```
as.numeric(fpain)
```

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

10.Lists

```
x=factor(1:10)
y=rnorm(10)
z=list(x,y)
z
```

```
## [[1]]
##  [1] 1  2  3  4  5  6  7  8  9  10
## Levels: 1 2 3 4 5 6 7 8 9 10
##
## [[2]]
##  [1]  0.14809912  0.69724767  0.17490751 -0.81362388 -0.49302883
##  [6]  0.42329354  0.28446853 -0.19503670 -0.04130296 -1.95530230
```

```
z=list(x=x,y=y)
z
```

```
## $x
##  [1] 1  2  3  4  5  6  7  8  9  10
## Levels: 1 2 3 4 5 6 7 8 9 10
##
## $y
##  [1]  0.14809912  0.69724767  0.17490751 -0.81362388 -0.49302883
##  [6]  0.42329354  0.28446853 -0.19503670 -0.04130296 -1.95530230
```

```
z$x
```

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

```
z$y
```

```
##  [1]  0.14809912  0.69724767  0.17490751 -0.81362388 -0.49302883
##  [6]  0.42329354  0.28446853 -0.19503670 -0.04130296 -1.95530230
```

11.Dataframe and index

This will be detailed in Chapter 2

12. Workspace

```
ls()
```

```
##  [1] "bmi"      "Brand"    "dat"      "fpain"    "height"  "id"       "miss.id"
##  [8] "names"    "oops"     "pain"     "Soft"     "student" "Temp"     "User"
## [15] "weight"   "x"        "xbar"     "y"        "z"
```

Remove or delete objects using the function rm(remove)

```
#rm(x)
#rm(list=ls())
```

12. Getting help

```
#help(t.test)
```