Handout 1 R Basics

1. Installation of **R** and packages

The way to obtain R is to download it from one of the CRAN (Comprehensive R Archive Network) sites. The main site is

http://cran.r-project.org/.

R is available to be freely downloaded to your computer:
1. Go to Google and type: R
2. You will get the following website: https://www.r-project.org/
3. Go to download R and choose a mirror, for example, UCLA.
4. Go ahead and download R!

Note: I highly recommend the Quick_R website: http://www.statmethods.net/

Package installation: To work through the examples and exercises in this book, you should install the ISwR package, which contains the data sets. If you are connected to the Internet, you can start R and from the Windows and Macintosh versions use their convenient menu interfaces.

On other platforms, you can type

```
> install.packages("ISwR")
```

This will give off a harmless warning and install the package in the default location. If your R machine is not connected to the Internet, you can also download the package as a file via a different computer.

Then you may need to load the package you need for your work at the command prompt, e.g.,

```
> library(ISwR)
```

2. First steps

Starting R is straightforward, but the method will depend on your computing platform.

2.1 Calculating an arithmetic expression

One of the simplest possible tasks in R is to enter an arithmetic expression and receive a result.

```
> exp(-2)
[1] 0.1353353
```

Exercises: Compute

(1) log(3.14), log10(3.14), log(3.14, 20);

(2) sin(2.1);
(3) 3^2.81
(4) sqrt(34.3).

The [1] in front of the result is part of R's way of printing numbers and vectors. It is not useful here, but it becomes so when the result is a longer vector. The number in brackets is the index of the first number on that line. Consider the case of generating 20 random numbers from a normal distribution:

```
>  rnorm(20)
```

```
?rnorm
??random
??normal
```

2.2 Assignments:

To assign the value 2 to the variable x, you can enter

```
> x <- 2
> x
[1] 2
> x + x
[1] 4
> x=2                    ### another two ways of assignments
> 3-> x
```

2.3 Vectorized arithmetic

The construct c(…) is used to define vectors.

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

You can do calculations with vectors just like ordinary numbers, as long as they are of the same length.

```
> height <- c(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
```

It is in fact possible to perform arithmetic operations and calculate some basic statistics on vectors of different length.

```
> bmi^2
> length(bmi)
> sum(bmi)
> mean(bmi)
```

```
> sd(bmi)
```

Exercise: (1) Compute the (A) median and (B) variance of 'weight'
(2) Compute the (A) covariance and (B) correlation of 'weight' and 'height'
(3) Implement the following operations:

```
> xbar <- sum(weight)/length(weight)
> weight – xbar
> sqrt(sum((weight - xbar)^2)/(length(weight) - 1))

cor(weight,height)
cov(weight,height)/sd(weight)/sd(height)
```

2.4 Standard statistical procedures

You could run standard T-test to assess whether the six persons' BMI can be assumed to have mean 22.5 given that they come from a normal distribution.

```
> t.test(bmi, mu=22.5)
One Sample t-test
data: bmi
t = 0.3449, df = 5, p-value = 0.7442
alternative hypothesis: true mean is not equal to 22.5
95 percent confidence interval:
18.41734 27.84791
sample estimates:
mean of x
23.13262
```

2.5 Graphics

```
> plot(height, weight)
> plot(height, weight, pch=2)
 > hh <- c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)
> lines(hh, 22.5 * hh^2)

plot(height, weight, pch=2,type="l")
```

3. R language essentials

3.1 Expressions and objects

The basic interaction mode in R is one of expression evaluation. The user enters an expression; the system evaluates it and prints the result. All R expressions return a value (possibly NULL), but sometimes it is "invisible" and not printed.

```
>  x<-NULL
```

3.2 Functions and arguments

Many things in R are done using function calls, commands that look like an application of a mathematical function of one or several variables; for example, log(x).

3.3 Vectors

We have already seen numeric vectors. There are two further types, character vectors and logical vectors. A character vector is a vector of text strings, whose elements are specified and printed in quotes:

```
> c("Huey","Dewey","Louie")
[1] "Huey" "Dewey" "Louie"
```

It does not matter whether you use single- or double-quote symbols, as long as the left quote is the same as the right quote:

```
> c('Huey','Dewey','Louie')
[1] "Huey" "Dewey" "Louie"
```

Logical vectors are constructed using the c function just like the other vector types:

```
> c(T,T,F,T)
[1] TRUE TRUE FALSE TRUE
```

Exercise: (1) Try the following R script:

```
> a<-c(2, 3, exp(3.2), sin(8))
> a>3
```

3.4 Quoting and escape sequences

```
> cat("Huey","Dewey","Louie", "\n")
Huey Dewey Louie
> cat("What is \"R\"?\n")
What is "R"?
```

```
cat("how are \"you\"?\n");cat("m")
```

3.5 Missing values

R allows vectors to contain a special NA value as missing values.

```
> a<-"NA"
```

3.6 Functions that create vectors

We introduce three functions, c, seq, and rep, that are used to create vectors in various situations.

```
> c(42,57,12,39,1,3,4)
[1] 42 57 12 39 1 3 4
```

```
> x <- c(1, 2, 3)
```

```
> y <- c(10, 20)
> c(x, y, 5)
[1] 1 2 3 10 20 5

> x <- c(red="Huey", blue="Dewey", green="Louie")
> x
red blue green
"Huey" "Dewey" "Louie"
> names(x)
[1] "red" "blue" "green"

> c(FALSE, 3)
[1] 0 3
> c(pi, "abc")
[1] "3.14159265358979" "abc"
> c(FALSE, "abc")
[1] "FALSE" "abc"

> seq(4,9)
[1] 4 5 6 7 8 9
> seq(4,10,2)
[1] 4 6 8 10
> 4:9
[1] 4 5 6 7 8 9

> oops <- c(7,9,13)
> rep(oops,3)
[1] 7 9 13 7 9 13 7 9 13
> rep(oops,1:3)
[1] 7 9 9 13 13 13
```

3.7 Matrices and arrays

```
> 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,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
?t
??transpose
colnames(x)<-letters[1:4]
colnames(x)<-c("j","a","b","c")

> 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
```

3.8 Lists

It is sometimes useful to combine a collection of objects into a larger composite object. This can be done using lists.

```
> intake.pre <- c(5260,5470,5640,6180,6390,6515,6805,7515,7515,8230,8770)
> intake.post <- c(3910,4220,3885,5160,5645,4680,5265,5975,6790,6900,7335)
> mylist <- list(before=intake.pre,after=intake.post)

> mylist
> mylist$before

rbind(before=intake.pre,after=intake.post)[1,]
```

3.9 Data frame

A data frame corresponds to what other statistical packages call a "data matrix" or a "data set". You can create data frames from preexisting variables:

```
d <- data.frame(intake.pre,intake.post)
d
      intake.pre    intake.post
1     5260          3910
2     5470          4220
3     5640          3885
4     6180          5160
5     6390          5645
6     6515          4680
7     6805          5265
8     7515          5975
9     7515          6790
10     8230         6900
11    8770          7335
d$intake.pre
```

3.10 Indexing

If you need a particular element in a vector, for instance the premenstrual energy intake for woman no. 5, you can do

```
intake.pre[5]
intake.pre[c(3,5,7)]
v <- c(3,5,7);  intake.pre[v]
 intake.pre[1:5]
intake.pre[-c(3,5,7)]
```

3.11 Conditional selection

```
intake.post[intake.pre > 7000]
intake.post[intake.pre > 7000 & intake.pre <= 8000]
intake.pre > 7000 & intake.pre <= 8000

d <- data.frame(intake.pre,intake.post)
d[5,1]
d[5,]
d[d$intake.pre>7000,]
sel <- d$intake.pre>7000
```