2. R Programming

Generating Random Data

- Generating Random Data

  R has several functions for generating random data.

  - sample() function
    - sample(x=*vector*, size=*n*, replace=FALSE, prob=NULL) : takes a sample of size *n* from *vector* either with (replace=TRUE) or without (replace=FALSE) replacement. A vector of probabilities for obtaining the elements of *vector*, prob, can be supplied optionally.

```
> set.seed(123)
> sample(x=1:5, size=3)
[1] 2 4 5
> sample(1:5, 10, replace=TRUE)
 [1] 5 5 1 3 5 3 3 5 3 4
> sample(1:5, 10, replace=TRUE, prob=c(0.6, 0.2, 0.1, 0.05, 0.05))
 [1] 1 1 3 1 1 1 4 3 2 2
```

Generating Random Data

- Built-in statistical distribution functions

  R has several built-in statistical distributions. For each distribution four functions are available,

  - r : random number generator
  - d : density function
  - p : cumulative distribution function
  - q : quantile function

  Each latter can be added as a prefix to any of the R distribution names below.

Distribution	R Name	Additional Arguments
binomial	binom	size, prob
chi-squared	chisq	df
exponential	exp	rate
F	f	df1, df2
normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df
uniform	unif	min, max

Generating Random Data

```
> dnorm(1.96, mean=0, sd=1)
[1] 0.05844094
> dnorm(1.96)
[1] 0.05844094
> pnorm(1.96, mean=0, sd=1)
[1] 0.9750021
> pnorm(1.96, mean=0, sd=1, lower.tail=FALSE)
[1] 0.0249979
> qnorm(0.975, 0, 1)
[1] 1.959964
> rnorm(5, mean=2, sd=1)
[1] 4.5283366 2.5490967 2.2382129 0.9511069 3.2947633
```

Control Statement

- Conditional Execution
  - Syntax for Conditional Execution

```
if (condition) { expression1 } else { expression2 }
```

: evaluates *expression1* if the logical expression *condition* returns TRUE, otherwise it evaluates *expression2*.

- A condition is an expression that evaluates to a *single* logical value.
- *expression1*, *expression2* is a single or a group of expressions.
- The else statement is optional. To avoid a syntax error, you should *not* have a newline between the closing bracket of the if statement and the else statement.

Control Statement

```
> x <- 4
> if ( x == 5 ) { x <- x+1 } else { x <- x*2 }
> x
[1] 8

> exam <- 82
> if( exam > 50 ) { grade <-"Pass" } else { grade <- "Fail" }
> grade
[1] "Pass"

> x<- c( 5, 4, 2, 8, 9, 10)
> n <- length(x)
> if ( n %% 2 == 0 ) {
+     print("even")
+ } else {
+     print("odd")
+ }
[1] "even"
```

Control Statement

- Loops : for-Loop
  - Syntax for for-Loops

```
for( var in seq ) { expression }
```

: sets the value of *var* equal to each element in *seq* in turn, each time do *expression* .

- *var* is the name of the loop variable which increments through the values in the set *seq*.
- *seq* is an expression that must return a vector of values.
- *expression* is a single or a group of expressions which is often written in terms of *var*.

Control Statement

```
> x <- c(10, 20, 30)
> a <- rep( 0, 3 )
> a[1] <- if( x[1]>15 ) x[1]  else x[1]/10
> a[2] <- if( x[2]>15 ) x[2] else x[2]/10
> a[3] <- if( x[3]>15 ) x[3] else x[3]/10
> a
[1]  1 20 30
> a2 <- rep( 0, 3)
> for ( k in 1:3 ) {
+     a2[k] <- if( x[k]>15 ) x[k]  else x[k]/10
+ }
> a2
[1]  1 20 30
> s <- 0
> for ( x in 1:10 ) {
+     if ( x %% 2 == 0 ) { s <- s + x }
+ }
> s
[1] 30
```

Control Statement

- **Loops : while-Loops**
  - Syntax for while-Loops

```
while( condition ) { expression }
```

: as long as *condition* is TRUE do *expressions* .

- *condition* is an expression which must evaluate to a simple logical value, and *expression* is a simple or compound expression.
- *condition* must be TRUE in the 1st iteration.
- You need to have an indicator variable and change its value within each iteration. Otherwise you will have an infinite loop.

Control Statement

```
> threshold <- 100
> n <- s <- 0
> while ( s <= threshold ) {
+     n <- n+1
+     s <- s+n
+ }
> c(n, s)
[1]  14 105
>
> a <- 0
> while( a!=5 ) {
+     a <- sample(1:10, 1)
+     print(a)
+ }
[1] 4
[1] 2
[1] 2
[1] 3
[1] 5
```

Functions

- Dening a R Function

```
function name <- function ( argument list ) { body }
```

  - <*ARGUMENT_LIST*> : a comma separated list of variable names
  - <*BODY*> : A simple or compound expressions. Use return() to return an object that can be assigned
  - Arguments and other objects created inside body are local to the function.

- Call a function by function

<*FTN_NAME*> (<*ARG_NAME*>=<*VALUE*>, …)

  - Arguments do not have to be named if they are entered in the same order as the function's formal argument list.

Functions

```
> square <- function ( x ) { return ( x^2 ) }
> sumsq <- function ( x ) {
+     ssq <- sum ( square ( x-mean ( x ) ) )
+     return ( ssq )
+     }
> sumsq ( x = 1 : 10 )
[1] 82.5

> between <- function ( x, minimum, maximum ) {
+     xsub <- x [ x > minimum & x < maximum ]
+     return ( xsub )
+    }
> pizza <- c( Tata=15, PizzaPizza=8, TwoForOne=16,
+             DoubleDouble=11, Bubba=15, Domino=21,
+             Godfatha=13, Zamaster=15, PizzaHut=22)
> between(minimum=15, maximum=20, x=pizza)
TwoForOne
       16
> between(pizza, 10, 20)
        Tata  TwoForOne DoubleDouble   Bubba  Godfatha  Zamaster
          15         16           11      15        13        15
```