CHAPTER 22

If-Else, Logic OR, and Logic AND

Here is the simplest control using the **if()** function.

```
> x<-10
> if(x>0) print("x>0")
```

Assume that we have the following simple data set with just two columns.

```
> year<-c(1991,1992,1993,1994)
> ret<-c(0.01,0.02,0.03,0.034)
> data<-cbind(year,ret)
```

```
> data
year ret
[1,] 1991 0.010
[2,] 1992 0.020
[3,] 1993 0.030
[4,] 1994 0.034
```

We want to get a subset of data, such as years after 1992.

```
> x<-subset(data,year>1992)
> x
 year ret
[1,] 1993 0.030
[2,] 1994 0.034
```

22.1. Introduction

Conditions—such as **if()** and **if()-else()** functions—are commonly used to manipulate data and/or redirect the flows of your programs. Assume that we have an R data set with daily return data for a thousand stocks over multiple years. We might be interested in just a few stocks for one year. To choose IBM for 2010, we can issue commands similar to the following codes: **if(ticker == "IBM" & year==2010)**, where & is logic AND.

22.2. The if() Function

In the following codes, we have a default value for the variable called **decision: reject**. When the NPV of our project is positive, we accept the project. From chapter 2, this is the NPV rule: if NPV>0, we accept the project; if NPV<0, we reject the project.

```
>decision<-"reject the project"
>if(npv>0) decision<-"accept the project"
>print(decision)
```

22.3. If-Else Function

If-else is another widely used structure (see the following example).

```
if(x>0){
 print("x>0")
 # your codes here
} else {
 print("x<=0")
 # your codes here
}
```

In the above codes, we classify **x** values into two groups.

22.4. If-Else–If-Else Function

For a multigroup classification, we can have multiple **if()** functions combined together. Below, we generate a function to convert a percentage grade into its corresponding letter grade.

```
letter_grade<-function(grade){
 if(grade>=90){
     final<-"A"
 } else if(grade>=80){
     final<-"B"
 } else if (grade>=70){
     final<-"C"
 } else if (grade>=60){
     final<-"D"
 } else{
     final<-"Fail"
}
 return(final)
}
```

To call the function is easy.

```
> letter_grade(90)
[1] "A"
> letter_grade(89)
[1] "B"
> letter_grade(79)
[1] "C"
> letter_grade(50)
[1] "Fail"
```

22.5. The if() and stop() Pair

In our future-value function, an interest rate cannot be negative. In this case, we use the **if()** and **stop()** pair.

```
fv_f<-function(pv,r,n){
 if(r<0)stop("interest is negative")
 return(pv*(1+r)^n)
}
```

We can test this by entering a negative interest rate.

```
> fv_f(100,0.1,1)
[1] 110
> fv_f(100,-0.1,1)
Error in fv_f(100, -0.1, 1) : interest is negative
```

22.6. Logic OR

The logic OR means that if any of the given condition is met, we go to the next step (i.e., we do something).

```
> x=1
> y=-3
> if(x>0 | y>0) cat("x=",x,"y=",y,"\n")
x= 1 y= 3
```

Assume that we are trying to choose a sample data from a pool of all American stocks with publicly trading data. According to our research topic, we are interested in NYSE-listed or American-listed stocks. If we have a variable called **EXCHAGE** code—which takes the value of 1 for NYSE, 2 for AMEX, or 3 or NASDAQ—then we can have the following codes:

```
>x2<-subset(x,EXCHAGE==1 | EXCHAGE==2)
```

22.7. Logic AND

In R, we use an ampersand (**&**) to represent a logic AND.

```
> x<-1
> y <- -3
> if(x>0 & y>0) print("both positive")
```

22.8. Going to the Next Line After Every Ten Numbers

First, let's look at the modulus function **%%**. It gives us the remainder **n %% m** (e.g., **11 %% 10** will be 1, and **23 %% 10** will be 3). The following one-line codes would print 100 values on one line with a blank space between each adjacent numbers.

```
for(i in 1:100) cat(" ",i)
```

Now, we add a return after every ten numbers. The output looks much better now (see the right panel below).

```
for(i in 1:100){
 cat(" ",i)
 if(i%%10==0) cat("\n")
}
```

```
 1  2  3  4  5  6  7  8  9 10
 11 12 13 14 15 16 17 18 19 20
 21 22 23 24 25 26 27 28 29 30
 31 32 33 34 35 36 37 38 39 40
 41 42 43 44 45 46 47 48 49 50
 51 52 53 54 55 56 57 58 59 60
 61 62 63 64 65 66 67 68 69 70
 71 72 73 74 75 76 77 78 79 80
 81 82 83 84 85 86 87 88 89 90
 91 92 93 94 95 96 97 98 99 100
```

22.9. Combination of Various Conditions

When we have different types of combinations, we should test our codes. We have two variables (x, y). Our condition is that both should be positive and that x be an even number. If this is met, then we print pass.

```
> x<-7
> y<-9
> if((x>0 | y>0) & x%%2==1) print("pass")
```

How about the following codes?

```
> if(x>0 | y>0 & x%%2==1) print("pass")
```

Exercises

22.1. Assume that we have five hundred stocks in a data set. How do we randomly choose fifty stocks from it?

22.2. What is wrong with the following codes?

```
If(x=0) print("x is zero")
```