**R Programlama Dili**

**(Aralık-2016)**

**1. Giriş**

Bu bölümde R Programlama Dili kısaca tanıtılacaktır.

**1.1. R Nasıl Bir Dil ve Ortamdır?**

R istatistiksel ve matematiksel veri analizinde kullanılan domain specific bir programlama dilidir. R özellikle istatistikçiler ve veri madencileri tarafından tercih edilmektedir. R GPL lisansına sahip olan açık kaynak kodlu (open source) bir yazılımdır. Dolayısıyla aynı zamanda bedavadır. R Programlama Dilinin ve paketlerinin gerçekleştirilmesinde ağırlıklı C Programlama Dİli kullanılmıştır. Ancak Fortran’dan da faydalanılmıştır. Bazı paketler ise doğrudan R’ın kendisi ile yazılmış durumdadır.

R yorumlayıcı temelli (interpreted) bir programlama dilidir. Hem komut satırlı karşılıklı etkileşimi (interactive) hem de script tabanlı bir çalışmayı destekler. Çeşitli konulara yönelik onlarca pakete (kütüphaneye) sahiptir. R cross platform’dur. Hem Windows sistemlerinde, hem Linux sistemlerinde hem de Mac OS X sistemlerinde yüklenerek çalıştırılabilmektedir. Güçlü grafik özelliklere sahiptir.

R aslında S isimli programlama dilinin bir türevidir. S’e bazı özellikler eklenerek oluşturulmuştur. Bu anlamda Scheme Dilinden de biraz etkilenmiştir.

R Auckland Üniversitesinde Ros Ihaka ve Roberrt Gentleman tarafından geliştirilmiştir. R’ın geliştirilmesine 1992 yılında başlanmıştır. Stabil ilk versiyonu 1995 yılında oluşturulmuştur.

**1.2. R’ın Kurulumu**

R’ın kurulumu oldukça basittir. Windows sistemlerinde <https://cran.r-project.org/bin/windows/base/> adresinden setup dosyası indirilerek kurulum yapılabilir. Bu kuruluumla birlikte kullanıcıya hem bir yorumlayıcı hem de karşılıklı etkileşimli bir ortam sıunulmaktadır. Linux sistemlerinde R komut satırından paket yöneticileriyle kurulabilir. Örneğin Ubuntu ve türevlerinde komut satırından aşağıdaki komutla doğrudan tban (base) R sistemini kurabiliriz:

sudo apt-get install r-base r-base-dev

Tabii Linux sistemlerinde istenirse yine kurulum paketi indirilerek de kurulum yapılabilir. Mac OS X sistemlerinde kurulum yine çok kolaydır. İlgili dmg dosyası indirilerek kurulum yapılabilir.

Ayrıca R için bazı IDE benzeri ortamlar da oluşturulmuştur. Bunların bazıları üzretlidir. Ancak “RStudio” denilen IDE ücretsizdir ve Windows, Linux, Mac OS X sistemlerinde kullanılabilmektedir. Biz kursumuzda “RStudio” IDE’sini kullanacağız. Bunun indirilip kurulumu da oldukça kolaydır. Bunun dışında R için ismine “RStudio Server” denilen bir “server” program da oluşturulmuştur. Bu server sayesinde biz Web tarayıcılarıyla RStudio ortamını kullanabiliriz.

**2. R Programlma Dilinin ve Ortamının Temel Özellikleri**

R dizisel (vektörel) tabanlı bir dildir. R’da değişkenler n tane değeri tutabilen vektör biçimindedir. Bir değişkenin tek bir değeri tuttuğu durum aslında tek elemanlı bir vektör gibi düşünülebilir. R’da C, Java, C# gibi dillerdeki gibi bir bildirim kavramı yoktur. Bir değişkene ilk kez bir değer atandığında onun bildirimi de adeta kendiliğinden yapılmış kabul edilir. R dinamik türlü (dynamic typed) bir dildir. Yani değişkenklerin türleri sabit değildir. Onlara atanan değerle birlikte onların türleri de değişmektedir.

R’da atama işlemşi için diğer pek çok dildeki gibi ‘=’ operatörü kullanılabilir. Ancak R’da atama için R’a özgü ok operaörü de kullanılabilmektedir. Biz kursumuzda atama işlemlerini bu operatörle yapacağız. Örneğin:

a <- 10

Atama işleminde atamanın yapıldığı değişken sağ tarafta da olabilir. Bu durumda okun yönü de değiştirilmelidir:

10 -> a

R’da vektör oluşturmak için c isimli fonksiyon kullanılır. Vektörün elemanları bu fonksiyona argüman olarak verilmelidir. Örneğin:

> a <- c(10, 20, 30)

> a

[1] 10 20 30

Bu durumda örneğin:

a <- 10

ile

a <- c(10)

eşdeğerdir.

Vektörler operatörlerle işleme sokulduğunda vektörün karşılıklı elemanları işleme sokulmuş olur. Örneğin:

> a <- c(1, 2, 3)

> b <- c(4, 5, 6)

> c <- a + b

> c

[1] 5 7 9

Pekiyi bu işlemlerde vektörler aynı uzunlukta değilse ne olur? İşte bu durumda R’da “döngüye sokma (recycling)” denilen kural uygulanmaktadır. Bu kurla göre uzunluğu kısa olan yinelenerek uzun olanla aynı uzunluğa getirilerek işleme sokulmaktadır. Örneğin:

> a <- c(1, 2, 3, 4, 5, 6)

> b <- c(1, 2)

> c <- a + b

> c

[1] 2 4 4 6 6 8

Burada kısa olan vektördei 1 2 değerleri 1 2 1 2 1 2 durumuna getirilerek işleme sokulmuştur. Ancak “döngüye sokma” işleminde büyük olan vektörün uzunluğunun küçük olanın uzunluğunun katı değilse uyarı (warning) oluşur. gerekir. Örneğin:

> a <- c(1, 2, 3, 4, 5)

> b <- c(1, 2)

> c <- a + b

Warning message:

In a + b : longer object length is not a multiple of shorter object length

Örneğin:

> a <- c(1, 2, 3, 4, 5)

> b <- 10

> c <- a \* b

> c

[1] 10 20 30 40 50

Burda görüldüğü gibi a’nın her elemanı 10 ile çarpılmıştır. Aslında burada yine “döngüye sokma” kuralının işletildiğine dikkat ediniz.

Vektör elde etmenin diğer bir yolu ‘:’ operatörünü kullanmaktır. Bu operatör iki operandlı araek bir operatördür. Sol tarafındaki değerden başlayarak sağ tarafındaki değere kadar (kapalı aralık) birer artırılmış değerlerden oluşan bir vektör elde etmemizi sağlar. Örneğin:

> a <- 1:10

> a

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

> b <- 1:10

> a \* b

[1] 1 4 9 16 25 36 49 64 81 100

Aslında soldaki değer sağdaki değerden küçük olmak zorunda değildir. Yani vektör ters sırada da elde edilebilir:

> a <- 10:1

> a

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

‘:’ operatörünün operandlarının tamsayı olması gerekmez. Fakat her zaman 1 artırım yapılmaktadır:

> a <- 1.5:10

> a

[1] 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5

‘:’ operatöründe artırım miktarı her zaman birdir bunu değiştiremeyiz. Ancak seq isimli fonksiyon bize istediğimiz artımı vermeye olanak sağlar. Örneğin:

> a <-seq(1, 10, 2)

> a

[1] 1 3 5 7 9

R’da default argüman kullanımı vardır. Yani bazı fonksiyonlarda biz bazı değerleri belirtmediğimizde onlar için default birtakınm değerler alınmaktadır. Örneğin seq fonksiyonunda biz artırımı belirtmek zorunda değiliz. Bu durumda artırım default olarak 1 alınmaktadır. Örneğin:

> a <-seq(1, 10)

> a

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

Bir vektörün uzunluğu (yani eleman sayısı) length fonksiyonuyla elde edilebilir. Örneğin:

> x <- seq(-3.14,+3.14, 0.01)

> length(x)

[1] 629

R’da pek çok temel matematiksel ve istatistiksel fonksiyon bulunmaktadır. Bu fonksiyonların pek çoğu bizden argüman olarak aldıkları vektörü işleme sokarak aynı uzunlukta bir vektör ya da duruma göre farklı uzunluklarda vektörler verebilmektedir.

- sum fonksiyonu vektördeki değerlerin toplamına geri dönmektedir. Örneğin:

> a <- 1:10

> b <- sum(a)

> b

[1] 55

Aynı işlem şöyle de yapılabilirdi:

> sum(1:10)

[1] 55

- mean fonksiyonu aritmetik ortalama bulmak için kullanılır. Örneğin:

> a <- c(10, 20, 30, 20, 20)

> mean(a)

[1] 20

Aynı işlem şöyle yapılabilirdi:

> mean(c(10, 20, 30, 20, 20))

[1] 20

- var fonksiyonu varyans hesaplamakta kullanılır. (Varyans standart sapmanın karesidir. Ve istatistikte değişkenliğin bir ölçüsü olarak kullanılmaktadır.) Örneğin:

> x <- c(4, 7, 4, 3, 4, 7, 8, 4, 3, 5)

> length(x)

[1] 10

> var(x)

[1] 3.211111

**Anahtar Notlar:** R Studio’da konsol ekranını silmek için Ctrl+L tuş kombinasyonu kullanılmaktadır. Yardım almak için ise ?<sözcük> sentaksı kullanılır.

varyans işlemini bir ifade olarak aşağıdaki gibi de yapabiliriz:

> sum((x - mean(x))^2) / (length(x) - 1)

[1] 3.211111

Burada var varyans için n değil n - 1 bölümünün uygulandığına dikkat ediniz.

- Standart sapma varyansın kareköküdür. sd fonksiyonu standart sapma için kullanılır. Örneğin:

> sqrt(var(x))

[1] 1.791957

> sd(x)

[1] 1.791957

- min fonksiyonu vektörün en küçük elemanını, max fonksiyonu en büyük elemanını bulmak için kullanılır. range fonksiyonu ise vektörün en küçük ve en büyük elemanını bize iki elemanlı bir vektör olarak verir. Örneğin:

> x <- c(1, 4, 6, 3, 2, 10, -4)

> min(x)

[1] -4

> max(x)

[1] 10

> range(x)

[1] -4 10

- runif (random uniform) fonksiyonı belli bir aralıkta belli sayıda düzgün dağılmış rastgele gerçek sayılar üretir. Eğer fonksiyonu tek bir argümanla çağırırsak o sayıda fakat [0, 1] aralığında gerçek sayılar üretecektir. Örneğin:

> runif(10)

[1] 0.06231093 0.35488917 0.02725558 0.06025808 0.06001106 0.71441643

[7] 0.05443122 0.97073749 0.67106107 0.65793174

Biz belli bir aralıkta belli sayıda rastgele değerler de üretebiliriz. Bunun için runif fonksiyonun diğer iki argümanını (min ve max) girmek gerekir. Örneğin:

> x <- runif(10, 0, 10)

> x

[1] 2.676859 9.194332 8.272974 2.266594 4.355374 1.833164 5.258495

[8] 1.806366 3.781353 5.384136

Düzgün dağılmış rastgele tamsayı değerler üretmek için sample fonksiyonu kullanılmaktadır. Fonksiyonun birinci parametresi bir vektör almaktadır. İkinci parametre de bu vektörden kaç tane rastgele eleman çekileceğini belirtir. Örneğin:

> x <- c(10, 20, 30, 40, 50)

> sample(x, 3)

[1] 40 30 10

Default durumda “iadesiz” çekim yapılmaktadır. Yani vektörden çekilen rastgele değer sonraki çekilirken yeniden vektöre yerleştirilmez. Böylece aynı elemandan birden fazla kez çıkma olasılığı yoktur. Tabii bu default durumda vektörün elemanından daha fazla iadesiz çekim yapamayız. Örneğin:

> sample(x, 7)

Error in sample.int(length(x), size, replace, prob) :

cannot take a sample larger than the population when 'replace = FALSE'

R’da Boolean değerler T (True) ve F (False) harfleriyle temsil edilmektedir. sample fonksiyonun üçüncü parametresi default olarak F (False) biçimdedir. Biz bu parametre için açıkça T (True) girersek “iadeli çekim” anlaşılır. Örneğin:

> x <- c(10, 20, 30, 40, 50)

> sample(x, 10, T)

[1] 30 20 20 10 40 30 40 10 50 40

Örneğin:

> sample(1:100, 10, T)

[1] 79 25 9 1 93 11 4 35 10 31

Sayısal lotoda bir kolon oynamak için “iadesiz çekim” yapmak daha uygundur. Örneğin:

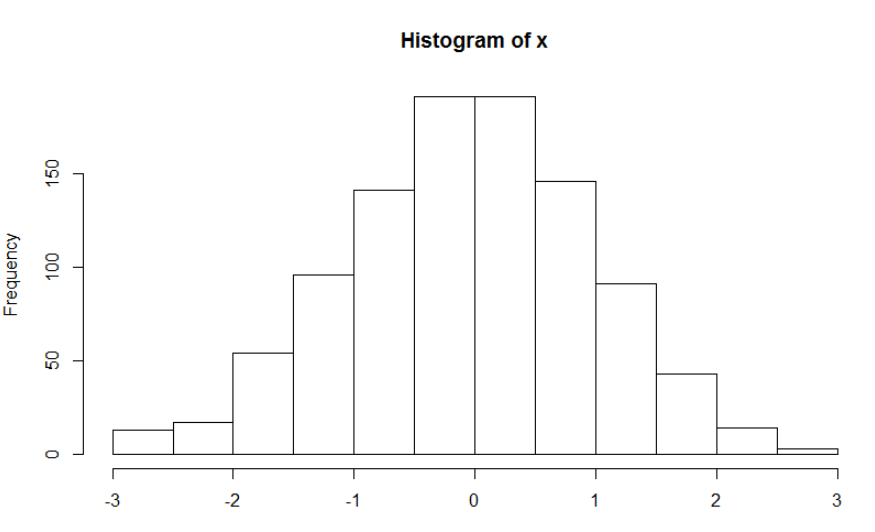
> sample(1:49, 6)

[1] 4 27 22 1 49 5

- rnorm fonksiyonu (“random normal” sözcüklerinden geliyor. rastgele normal dağılmış (Gauss dağılmış) n tane değer üretmek için kullanılır. Default durumda bu değerler ortalaması 0, standart sapması 1 olan bir sistemden elde edilmektedir (standart normal dağılım). Örneğin:

> x <- rnorm(1000)

> hist(x)



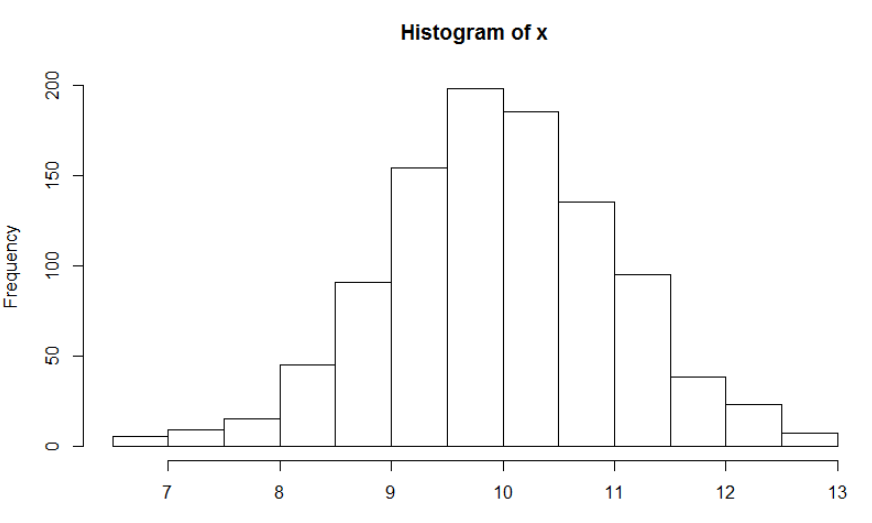
Fonksiyonun parametrik yapısı şöyledir:

rnorm(n, mean = 0, sd = 1)

Biz istersek ortalama ve standart sapması tam istediğimiz gibi olan bir sistemden rastgele normal dağılmış değerler de elde edebiliriz. Örneğin:

> x <- rnorm(1000, 10, 1)

> hist(x)



Bilindiği gibi istatistikte sürekli rassal değişkenlerin olasılık yoğunluk fonksiyonlarının (probablity density functions) eğri altında kalan alanları 1’dir. İki nokta arasında eğri altında kalan alan ilgili rassal değişkenin o iki nokta arasına düşme olasılığını verir. Standart normal dağılımda (ortalaması 0, standart sapması 1 olan normal dağılım) kümülatif olasılıklara Z değeri denilmektedir. Biz pnorm fonksiyonu ile eksi sonsuzdan belli bir değere kadarki kümülatif olasılığı elde edebiliriz. Örneğin:

> pnorm(2)

[1] 0.9772499

Buradan eksi sonsuzdan 2’ye kadar standart normal dağılım eğrisi altında kalan alanın 0.9772999 olduğu görülmektedir. Bu aynı zamanda P{X < 2} durumunu belirtmektedir. Başka bir deyişle pnorm(n) ile P{X < n} aynı anlamdadır.

pnorm fonksiyonu da default olarak ortalaması 0, standart sapması 1 olan standart normal dağılımı dikkate almaktadır. Tabii biz iki argüman daha girerek bunu değiştirebiliriz:

> pnorm(10, 10, 1)

[1] 0.5

qnorm fonksiyonu pnorm fonksiyonun tersini yapmaktadır. Yani bu fonksiyon bizden eğri altında kalan alanı (kümülatif olasılığı) alır ona karşı gelen X (Z değerini) değerini verir. Örneğin:

> p <- pnorm(2)

> p

[1] 0.9772499

> x <- qnorm(p)

> x

[1] 2

qnorm fonksiyonunun da ortalamayı ve standart sapmayı girebileceğimiz default argümanları vardır. Örneğin:

> p <- pnorm(11, 10, 1)

> qnorm(p, 10, 1)

[1] 11

Ayrıca bir de dnorm isimli fonksiyonn vardır. Bu fonksiyon bizden X değerini alır ve Gauss dağılım fonksiyonuna sokarak bize Y değerini verir. Örneğin:

> dnorm(1)

[1] 0.2419707

> dnorm(11, 10, 1)

[1] 0.2419707

- sort fonksiyonu vektörün elemanlarını sıraya dizerek bize sıraya dizilmiş yeni vektör verir. Örneğin:

> x <- sample(1:100, 10)

> x

[1] 43 36 58 13 41 49 39 63 100 64

> sort(x)

[1] 13 36 39 41 43 49 58 63 64 100

sort fonksiyonun ikinci parametresi isteğe bağlıdır ve default durumu F (False) biçimdedir. Bu parametre girilmezse (default durum) sıraya dizme küçükten büyüğe yapılır. Büyükten küçüğe sıraya dizme için bu ikinci parametrenin T (True) olarak girilmesi gerekir. Örneğin:

> sort(x)

[1] 13 36 39 41 43 49 58 63 64 100

> sort(x, F)

[1] 13 36 39 41 43 49 58 63 64 100

> sort(x, T)

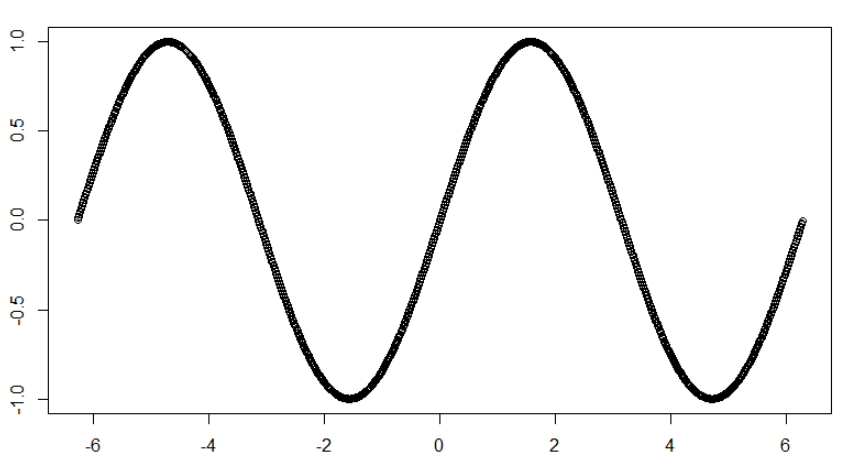
[1] 100 64 63 58 49 43 41 39 36 13

- sin, cos, tan, asin, ascos, atan fonksiyonları temel trigonometrik işlemleri yapmaktadır. Bu fonksiyonlarda açılar radyan cinsinden belirtilirler. Örneğin:

> x <- seq(-3.14\*2, +3.14\*2, 0.01)

> y <- sin(x)

> plot(x, y)



**Stringler ve Stringler Türünden Türünden Vektörler**

R’da string oluşturmak için tek tırnak ya da çift tırnak kullanılır. Tek tırnak ya da çift tırnak arasında bu bağlamda hiçbir fark yoktur. Örneğin:

> "Kaan Aslan"

[1] "Kaan Aslan"

> 'Kaan Aslan'

[1] "Kaan Aslan"

String türünden vektörler de söz konusu olabilir. Örneğin:

> cities = c("Ankara", "İzmir", "Adana", "Bursa", "Eskişehir")

> cities

[1] "Ankara" "İzmir" "Adana" "Bursa" "Eskişehir"

Boş string söz konusu olabilir. Örneğin:

> str = c("Ali", "", "Veli")

> str

[1] "Ali" "" "Veli"

İki string + operatörü ile toplanamaz. (Bazı dillerde bu yapılabilmektedir) paste isimli, fonksiyon iki yazıyı birleştirmekte kullanılmaktadır. Örneğin:

> a <- "Ali"

> b <- "Veli"

> c <- paste(a, b)

> c

[1] "Ali Veli"

paste değişken sayıda argüman alan bir fonksiyondur. paste fonksiyonun default olarak ayıraç olarak araya bir tane boşluk karakteri yerleştirdiğine dikkat ediniz. Eğer istenirse sep parametresiyle biz yıraç karakterini değiştirebiliriz. Örneğin:

> a <- "ali"

> b <- "veli"

> c <- "selami"

> d <- paste(a, b, c)

> d

[1] "ali veli selami"

> d <- paste(a, b, c, sep="")

> d

[1] "aliveliselami"

**Vektör Elemanlarına Erişim**

R’da bir vektörün elemanlarına erişmek için [ ] operatörü kullanılmaktadır. Vektör indeksleri 0’dan değil 1’den başlar. Örneğin:

> a <- c(10, 20, 30, 40, 50)

> a[1]

[1] 10

> a[5]

[1] 50

> a[100]

[1] NA

Tabii vektörün belli bir elemanına yeni bir değer de atayabiliriz:

> a <- c(10, 20, 30, 40, 50)

> a[1] <- 100

> a

[1] 100 20 30 40 50

Köşeli parantez içerisine genel olarak bir indeks vektörü yerleştirilebilir. Bu durumda o indekslerdeki elemanlardan oluşan bir vektör elde edilecektir. Örneğin:

> a <- 1:10

> b <- a[1:5]

> b

[1] 1 2 3 4 5

Ancak köşeli parantez içerisinde tek bir değer bulunmak zorundadır. O değer genel olarak bir vektör olabilir. Index vektörü aynı elemanları içerebilir. Örneğin:

Örneğin:

> a <- c(1, 2, 3, 4, 5)

> a[c(1, 3, 3, 5)]

[1] 1 3 3 5

Örneğin:

> a[1, 3]

Error in a[1, 3] : incorrect number of dimensions

Köşeli parentezler içerisindeki vektör (tek eleman da bir vektördür) negatif değerler içerebilir. Negatif bir değer “bu indeksin dışındaki tüm değerler” anlamına gelir. Örneğin:

> a <- c(10, 20, 30, 40, 50)

> a[-2]

[1] 10 30 40 50

Tabii indeks vektör birden fazla negatif değer de içerebilir. Örneğin:

> a <- c(10, 20, 30, 40, 50)

> a[c(-2, -2, -4)]

[1] 10 30 50

Ancak köşeli parantez içerisindeki indeks vektöründe pozitif ve negatif elemanlar bir arada bulunamazlar:

> a[c(-2, 4)]

Error in a[c(-2, 4)] : only 0's may be mixed with negative subscripts

Köşeli parentez içerisinde Bool türden (yani T (True) ve F (False) değerlerden oluşan) bir vektör yerleştirilebilir. Bu durumda T (True) olan elemanlara karşı gelen vektör değerleri elde edilir. Örneğin:

> a <- c(10, 20, 30, 40, 50)

> b <- c(T, F, F, T, T)

> a[b]

[1] 10 40 50

Bool türden vektör daha az elemana sahip olabilir. Bu durumda döngüye sokma işlemi (recycling) gerçekleştirilecektir. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> a[c(T, F)]

[1] 10 30 50

R’da >, <, >=, <= == ve != biçimindeki karşılaştırma operatörleri iki vektörün karşılıklı elemanları üzerinde işlem yapıp ürün olarak Bool türden bir vektörü bize verir. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> a > 35

[1] FALSE FALSE FALSE TRUE TRUE TRUE

Bu durumda biz bir vektörde belli koşulu sağlayan elemanları elde edebiliriz. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> a[a > 35]

[1] 40 50 60

Burada önce a > 35 işleminden Bool bir vektör elde edilmektedir. Sonra o vektör indeks vektörü olarak kullanılarak T (True) olan vektör elemanları seçilmiştir.

Köşeli parantez içerisindeki indeks vektöründe büyükj indeksler olabilir. Bu durumda o indeks değerleri için NA (Not Available) özel değeri elde edilir. Ancak sıfır değeri ya da negatif büyük değerler varsa onlar tamamen görmemezlikten gelinmektedir. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> a[c(1, 3, 100, 5, 200)]

[1] 10 30 NA 50 NA

> a[c(-2, -3, -100)]

[1] 10 40 50 60

Köşeli parantez içerisine sıfır indeksi yerleştirilirse bu durumda R bize sıfır uzunlukta bir vektör verir. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> b <- a[0]

> length(b)

[1] 0

Vektörün birden fazla elemanına tek hamlede atama yapılabilir. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> a[c(1, 3, 5)] <- 1000

> a

[1] 1000 20 1000 40 1000 60

Bir vektörün uzunluğunun ötesindeki bir elemanına da atama yapılabilir. Bu durumda vektör büyütülmüş olur. Fakat aradaki elemanlar NA değerini alır. Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> a[20] = 11111111

> a

[1] 10 20 30 40 50 60 NA

[8] NA NA NA NA NA NA NA

[15] NA NA NA NA NA 11111111

> length(a)

[1] 20

Vektörün negatif büyük bir elemanına değer atamaya çalışırsak bundan tüm elemanlar etkilenir. Örneğin:

> a <- 1:10

> a[-20] <- 100

> a

[1] 100 100 100 100 100 100 100 100 100 100

Bir vektörün elemanlarına isim de verilebilir. Bu durumda ilgili elemana isim belirtilerek de erişilebilmektedir. İsim verme işlemi iki biçimde yapılabilir. Birincisinde c fonksiyonunda elemanları girerken “isim = değer” biçiminde girişi yapmaktır. Örneğin:

> a <- c(ali=100, veli=200, selami=300)

> a

ali veli selami

100 200 300

Elemanlara isim vermenin diğer bir yolu da names isimli property fonksiyonunu aşağıdaki gibi kullanmaktır:

names(vektör\_ismi) <- string\_vektörü

Örneğin:

> a <- c(10, 20, 30, 40, 50, 60)

> names(a) <- c("Ali", "Veli", "Selami", "Ayşe", "Fatma", "Tuğrul")

> a

Ali Veli Selami Ayşe Fatma Tuğrul

10 20 30 40 50 60

Vektörün ilgili elemanına isim verilmişse artık biz o elemana köşeli parantez içerisinde o ismi geçirerek de referans edebiliriz. Örneğin:

> a["Ali"]

Ali

10

Örneğin:

> a[c("Ali", "Selami", "Fatma")]

Ali Selami Fatma

10 30 50

Eğer vektörde o isimli bir eleman yoksa NA değeri elde edilir. Örneğin:

> a[c("Ali", "Selma", "Fatma")]

Ali <NA> Fatma

10 NA 50

İsimlerde büyük harf küçük harf önemlidir (case sensetive). Dolayısıyla biz burada yanlış giriş yaparsak yine NA değeri elde ederiz. Örneğin:

> a[c("Ali", "selami", "Fatma")]

Ali <NA> Fatma

10 NA 50

**R’da Fonksiyon Çağırma ve Argüman Girme İşlemi**

R’ın temel kütüphanesinde başkaları tarafından yazılmış pek çok fonksiyon vardır.Bu fonksiyonlar çeşitli parametreler almaktadır. Biz de fonksiyonları çağırırken bu parametreler için argümanlar gireriz. Argüman girmenin bazı kuralları vardır. R’da bir fonksiyonun parametrik yapısı ve bunların anlamları “yardım” alınarak görülebilir. Komut satırında ?<fonksiyon ismi> ifadesiyle o fonksiyon hakkında bilgi hızlı bir biçimde elde edilebilir. Örneğin seq fonksiyonunun parametrik yapısı şöyledir:

seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),

length.out = NULL, along.with = NULL, ...)

R’da fonksiyonlar default argüman alabilmektedir. Default argüman ilgili parametreye bir değer girilmezse sanki o argüman girilmiş gibi etki yapar. Örneğin biz bıurada from parametresi için argüman girmezsek sanki 1 girmişiz gibi ele alınır.

R’da argüman girerken parametre değişkeninin ismi de istenirse belirtilebilir. Bu işlem isim=değer biçiminde yapılır. Böylece biz eğer argümanı isimlendirirsek onları farklı sıralarda girebiliriz. Örneğin:

> a <- seq(1, 10)

Burada “from” parametresi için 1, “to” parametresi için 10 girilmiştir. Eğer argümanda isimler belirtilmezse girişin parametre sırasına göre yapıldığı anlaşılır. Ancak argümanda isim belirtilirse biz girişi istediğimiz sırada yapabiliriz. Örneğin:

> a <- seq(to = 10, from = 1)

R’da fonksiyonlar çok parametreli olma eğilimindedir. Bu nedenle tüm parametreler için sırasıyla tek tek argüman girmek yerine yalnızca bazıları için onların isimlerini belirterek argüman girmek geri kalanları için default değerleri kullanmak çok uygulanan bir yöntemdir. Örneğin normal dağılmış rastgele üretmek için kullanılan rnorm fonksiyonun parametrik yapısı şöyle belirtilmiştir:

rnorm(n, mean = 0, sd = 1)

Bu parametrik yapıya göre rnorm fonksiyonunu biz en az bir parametre çağırmak zorundayız. Örneğin:

> a <- rnorm(20)

> a

[1] -1.50279473 -0.05871795 -1.00874670 0.43146554 0.46016773 2.23006472

[7] 0.06181493 0.02350081 -0.79207504 1.86465588 0.65671625 -1.40508083

[13] -0.25499096 -0.60345576 -1.26534176 0.78253359 -0.85565088 -1.26187570

[19] -0.94883431 0.51986000

rnorm fonksiyonunu tek parametreyle çağırdığımızda aslında biz mean parametresi için 0, sd parametresi için 1 girmiş gibi oluruz. Yani yukarıdaki çağrıyla biz ortalaması 10, standart sapması 1 olan 20 tane normal dağılmış rastgele sayı elde ederiz. Örneğin:

> a <- rnorm(20, 10)

> a

[1] 11.348835 10.063667 8.407719 9.760907 10.809608 9.782837 10.083690

[8] 9.150398 9.431458 11.650230 9.378288 10.947187 9.322524 9.156081

[15] 9.426711 10.640213 8.802042 10.202101 9.087402 11.807196

Burada 10 değeri mean parametresi için girilmiştir. Ancak sd yine 1 alınacaktır. Örneğin:

> a <- rnorm(20, 10, 2)

> a

[1] 8.915381 11.850115 11.110305 13.714220 8.347249 7.517320 9.424634

[8] 11.560255 9.476678 7.752482 8.347498 11.690092 5.954294 8.779689

[15] 11.814129 9.414470 13.332150 11.348432 9.072280 7.853918

Burada artık standart sapma değeri 2 olarak girilmiştir. Yukarıdaki çağrılarda parametre isimlerini belirtilmediğine dikkat ediniz. Bu durumda argümanlar sırasıyla parametrelerle eşleşmektedir. Ancak biz argümanalada parametre isimlerini belirterek onları farklı sıralarda da girebiliriz. Örneğin:

> a <- rnorm(sd = 2, mean = 10, n = 20)

> a

[1] 10.686163 8.914080 9.633009 12.001096 6.794499 9.258110 12.156021

[8] 12.350864 9.008265 7.961633 13.951242 10.728708 11.143667 13.370082

[15] 9.116364 11.758134 8.802849 10.870438 8.677136 11.507828

Burada argümanlar farklı sıralarda girilmiştir. Pekiyi bazı arümanlar için parametre isimleri verip bazıları için vermeyebilir miyiz? Örneğin:

a <- rnorm(sd = 2, 10, me=20)

Bu geçerli midir? Geçerliyse hangi parametreler için hangi değerler girilmiştir?

İşte R’da argüman parametre eşleştirmesi için şu kurallar sırasıyla işletilmektedir:

1) Önce çağırma ifadesindeki isimli argümanların isimleri ile tam eşleşen parametreler varsa bu argümanların o parametreler için girildiği kabul edilir ve bu parametreler listeden çıkartılır.

2) Sonra çağırma ifadesindeki isimli argümanların isimleri ile kısmi eşleşen parametreler belirlenir. Bu argümanların o parametreler için girildiği kabul edilir. Onlar da listeden çıkartılır. (Argümanda parametre isminin tamamı değil onun yalnızca ilk karakterleri belirtilebilir. Buna kısmi eşleşme (partial matching) denilmektedir. Tabii kısmi eşleşen birden fazla parametre bulunması durumu error oluşturur.)

3) Geri kalanlar sırasıyla isim verilmemiş argümanlarla eşleştirilir.

4) Eşleşmeyen parametreler varsa bunların default değer almış olması gerekir. Aksi taktirde bu durum da error oluşturur.

Şimdi yukarıdaki çağrıya bakalım:

a <- rnorm(sd = 2, 10, me=20)

rnorm fonksiyonun parametreik yapısı da şöyleydi:

rnorm(n, mean = 0, sd = 1)

Burada birinci aşamada tam uyum sağlayan sd argümanı vardır. Bu parametre listesinde çıkartılır. Sonra kısmi uyum sağlayan me argümanı vardır. Bu da mean parametresi ile eşleşir. Bu da listeden çıkartılır. Listede yalnızca n kalmıştır. İsimsiz olan 10 parametresi bununla eşleşir. Sonuç olarak bu çağrı ile biz 10 tane standart sapması 2 olan, ortalaması 20 olan normal dağılmış rastgele sayı elde etmiş oluruz.

> a <- rnorm(sd = 2, 10, me=20)

> a

[1] 16.84862 24.46545 22.58462 19.21947 19.51264 18.46788 19.87926 20.00262

[9] 20.64131 18.08158

Örneğin:

> a <- rnorm(s = 2, 10, m=20)

> a

[1] 22.48644 17.96020 19.53078 22.33452 17.10982 19.31221 20.64080 19.56527

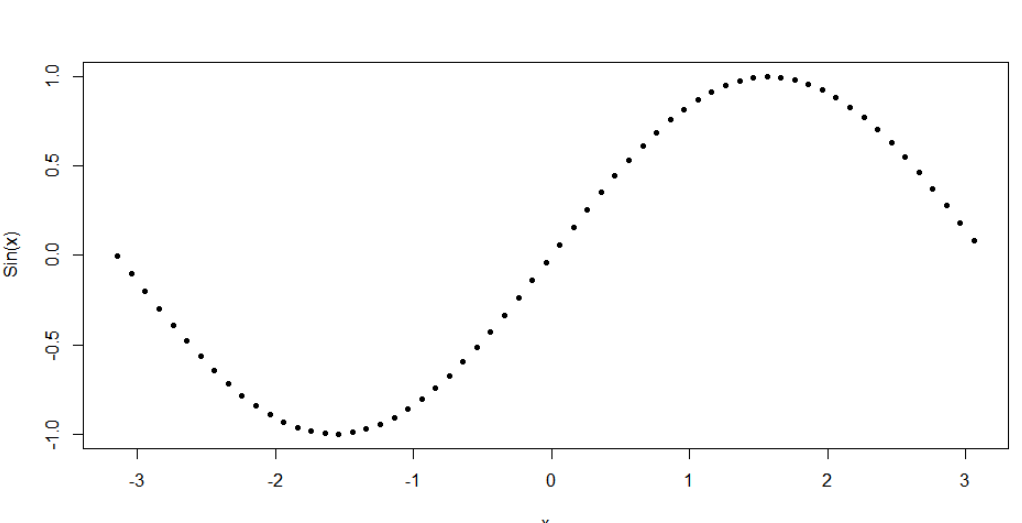
[9] 18.73219 18.25137

Ayrıca R’da değişken sayıda argüman alan fonksiyonlar da yazılabilmektedir. Bunun için “...” (ellipsis) karakteri kullanılır. “...” (ellipsis) parametresi için fonksiyonu çağıran o fonksiyonun kabul edeceği belli sayıda isimli argüman girebilir. Örneğin plot fonksiyonun parametrik yapısı R dokümanlarında şöyle belirtilmiştir:

plot(x, y, ...)

Burada biz en azından x ve y parametreleri için argüman girmeliyiz. Ancak bunun dışında çok sayıda daha fazla argüman girebiliriz. Tabii onların isimlerini vermemiz gerekir. Örneğin:

plot(x, y, pch=20, xlab="x", ylab="Sin(x)")



**R’da Listeler (Lists)**

Biz şimdiye kadar yualnızca vektör kullandık. Vektör elemanları aynı türden olan (aynı moddan olan) bir dizi gibi düşünülebilir. Örneğin bir vektörün bazı elemanları tamsayı bazı elemanları string olamaz:

> a <- c(10, 20, "ali")

> a

[1] "10" "20" "ali"

Burada da görüldüğü gibi görünüşte c fonksiyonu ile biz farklı türlerden vektör elemanları girdiğimizi sanabiliriz. Halbuki c fonksiyonu bu girilen elemanları ortak bir türe (örnekte string’e) dönüştürmektedir.

Listeler list isimli fonksiyonla oluşturulmaktadır. Örneğin:

> a <- list("Ali Serçe", 123)

> a

[[1]]

[1] "Ali Serçe"

[[2]]

[1] 123

Bir listenin elemanlarına isimler verebiliriz. Bunun için list fonksiyonunda argümanlar isim=değer biçiminde girilmelidir. Örneğin:

> person <- list(name = "Ali Serçe", no = 123)

> person

$name

[1] "Ali Serçe"

$no

[1] 123

Genellikle liste elemanlarına isim verilmektedir. Listeler bileşik nesnelerdir. Bunların da parçaları (elemanları) vardır. Listenin belli bir elemanına erişmek için $ operatörü kullanılır. Bu operatörün solunda liste değişkenin ismi sağında listenin eleman ismi bulunur. Tabii bu operatörün kullanılabilmesi için liste elemanlarına isim verilmiş olmalıdır. Örneğin:

> person <- list(name = "Ali Serçe", no = 123)

> person$name

[1] "Ali Serçe"

> person$no

[1] 123

Listenin elemanları birer vektörüdür. Yani yukarıdaki örnekte person$name aslına bir elemanlı bir string vektörü, person$no da bir elemanlı numeric bir vektördür.

Listenin elemanlarına $ operatörü ile erişilirken isim kısmi eşleşme olarak verilebilir. Örneğin:

> person <- list(name = "Ali Serçe", no = 123)

> person$na

[1] "Ali Serçe"

> person$n

NULL

> person$no

[1] 123

Birden fazla elemanın kısmi eşleşmede eşleşmesinin elemanın elde edilememesine (NULL elde edilmesine) yol açtığına dikkat ediniz.

Bir listenin elemanlarına tek köşeli parantez ile erişebiliriz. Tek köşeli parantezin içerisine bir indeks vektörü getirilebilir. Tabii bu vektör tek elemanlı da olabilir. Ancak tek köşeli parantez bize listenin bazı elemanlarını vektör olarak değil yeni bir liste olarak vermektedir. Halbuki $ operatörü bize listenin elemanını vektör olarak verir. Örneğin:

> person <- list(name = "Ali Serçe", no = 123, bdate="10/10/1970")

> subperson <- person[c(1, 2)]

> subperson

$name

[1] "Ali Serçe"

$no

[1] 123

Burada görüldüğü gibi person[c(1, 2)] ifadesi ile biz person listesinin 1’inci ve 2’inci indekslerindeki bilgileri alarak bunu yeni bir liste (alt liste de diyebiliriz) biçiminde elde ettik. $ operatörünün sağına bir indeks getiremeyiz. Yalnızca tam eşleşen ya da kısmi eşleşen eleman ismi getirebiliriz.

Tabii listenin elemanlarına isim vermemişsek $ operatörünü kullanamayız ancak tek köşeli parantez ile yine onun bazı elemanlarını alabiliriz. Örneğin:

> person <- list("Ali Serçe", 123, "10/10/1970")

> person[1]

[[1]]

[1] "Ali Serçe"

> typeof(person[1])

[1] "list"

Listelerin elemanlarına çift köşeli parantez operatöryle de erişilebilir. Ancak çift köşeli parantezin içerisindeki indeks vektörünün tek elemanlı olması zorunludur. Çift köşeli parantez bize listenin ilgili elemanını vektör olarak verir. (Halbuki tek köşeli parantez operatörünün ilgili elemanı liste olarak verdiğini anımsayınız.) Örneğin:

> person <- list("Ali Serçe", 123, "10/10/1970")

> person[1]

[[1]]

[1] "Ali Serçe"

> typeof(person[1])

[1] "list"

> person[[c(1)]]

[1] "Ali Serçe"

> person[[c(1, 2)]]

Error in person[[c(1, 2)]] : subscript out of bounds

Elemanlara isim verilmişse tek köşeli parantezde de çift köşeli parantezde de elemanlara erişmek için string vektör kullanılabilir. Örneğin:

> person <- list(name = "Ali Serçe", no = 123, bdate="10/10/1970")

> person[c("name", "bdate")]

$name

[1] "Ali Serçe"

$bdate

[1] "10/10/1970"

> person[["no"]]

[1] 123

O halde liste elemanlrına erişme işleminin kuralları şöyle özetlenebilir:

1) Elemana $ operatöryle erişilecekse bu operatörün sağında indeks bulunamaz, yalnızca isim bulunabilir. Ancak isim tam eşleşen ya da kısmi eşleşen biçimde olabilir. Bu biçimde eleman vektör biçiminde elde edilir.

2) Elemanlara tek köşeli parantez ile erişilecekse tek köşeli parantezin içerisinde bir indeks vektörü ya da isim vektörü bulunabilir. Bu durumda vektörün eleman sayısı birden fazla olabilir. Ancak bu operatör bize listenin elemanlarını bir liste biçiminde vermektedir. İsim vektöründe kısmi eşleşme uygulanmaz.

3) Elemana çift köşeli parantez ile erişilecekse çift köşeli parantezin içerisinde indeks vektörü ya da isim vektörü bulunabilir. Ancak bu vektörün tek elemanı olmak zorundadır. Yine isimde kısmi eşleşme uygulanmaz. Çift köşeli parantez ile erişimde eleman bize vektör olarak verilir.

Listelerin elemanlarının birer vektör olduğuna dolayısıyla bunların eleman sayılarının da birden fazla olabileceğine dikkat ediniz. Örneğin:

> names <- c("Ali Serçe", "Kaan Aslan", "Necati Ergin")

> numbers <- c(123, 234, 567)

> persons <- list(names=names, numbers=numbers)

> persons

$names

[1] "Ali Serçe" "Kaan Aslan" "Necati Ergin"

$numbers

[1] 123 234 567

> persons$names[1]

[1] "Ali Serçe"

> persons$numbers[1]

[1] 123

O halde R’da SPS gibi programlardaki “data set”ler aslında listeler biçiminde oluşturulabilir. Aslında R’da ileride de görüleceği gibi aynı uzunlukta vektörlerlerden oluşan listelere “faktör (factor)” denilmektedir.

Bir listenin elemanlarına liste yaratıldktan sonra da names isimli öznitelik (attribute) fonksiyonuyla da isim verilebilir. Örneğin:

> names <- c("Ali Serçe", "Kaan Aslan", "Necati Ergin")

> numbers <- c(123, 234, 567)

> persons <- list(names, numbers)

> persons

[[1]]

[1] "Ali Serçe" "Kaan Aslan" "Necati Ergin"

[[2]]

[1] 123 234 567

> names(persons) <- c("names", "numbers")

> persons

$names

[1] "Ali Serçe" "Kaan Aslan" "Necati Ergin"

$numbers

[1] 123 234 567

Dosyadan Bilgi Okumak İçin Kullanılan scan Fonksiyonu

R’da çeşitli dosya formatlarından bilgi okumak için kullanılan pek çok hazır fonksiyon vardır. Böyelce bu sayede biz bir text dosyadan, bir excel dosyasında, bir veritabanı dosyasından, bir SPSS dosyasından verileri alıp R ortamına aktararak işleyebiliriz.

Text bir dosyadan veri okumak için kullanılan en genel fonksiyonlardan biri scan fonksiyonudur. scan fonksiyonun parametrik yapısı şöyledir:

scan(file = "", what = double(), nmax = -1, n = -1, sep = "",

quote = if(identical(sep, "\n")) "" else "'\"", dec = ".",

skip = 0, nlines = 0, na.strings = "NA",

flush = FALSE, fill = FALSE, strip.white = FALSE,

quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,

comment.char = "", allowEscapes = FALSE,

fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

scan fonksiyonu dosyadan veri elemanlarını okur, default durumda onu bize bir vektör olarak verir. Fonksiyonun ilk parametresi olan “file” okunacak dosyasnın yol ifadesini belirtmektedir. Bu parametre için boş string girilirse bu durum klavyeden (stdin dosyasından) okuma yapılacağı anlamına gelir. Örneğin:

> a <- scan("")

1: 123

2: 234

3: 567

4: 23.4

5: 45.6

6:

Read 5 items

> a

[1] 123.0 234.0 567.0 23.4 45.6

scan fonksiyonu normal olarak boşluk karakterlerini dikkate alarak dosya içerisindeki karakter öbeklerini ayrıştırır ve onları birer sayı olarak nümerik bir vektöre yerleştirir, onu da bize verir. scan fonksiyonuyla yol ifadesi vererek bir text dosyadan da okuma yapabiliriz. Örneğin:

> a <- scan("C:\\Dropbox\\Shared\\Kurslar\\Arcelik-BigData\\Example Files\\data.txt")

Read 7 items

> a

[1] 10.0 12.4 45.6 23.0 34.0 45.0 49.0

R’da default çalışma dizini (current working directory) getwd fonksiyonuyla elde edilebilir. Örneğin:

> getwd()

[1] "C:/Dropbox/Study/R"

setwd fonksiyonuyla bu dizin değiştrilebilir. Örneğin:

> setwd("C:\\Dropbox\\Shared\\Kurslar\\Arcelik-BigData\\Example Files")

> getwd()

[1] "C:/Dropbox/Shared/Kurslar/Arcelik-BigData/Example Files"

> a <- scan("data.txt")

Read 7 items

> a

[1] 10.0 12.4 45.6 23.0 34.0 45.0 49.0

RStudio açıldığında default çalışma dizininin istediğimiz bir dizin olmasını sağlamak için Tools/Global Options/Default working directory menüsü kullanılır.

