

Yeni Bařlayanlar iin R'a Hızlı Giriř

Hazırlayanlar (soyadı sırasına gre)

İsmail Bařođlu
Mustafa Gke Baydođan
Uzay etin
Berk Orbay

29 Ocak 2016

Akademik Biliřim Konferansı Eđitimleri
30 Ocak - 2 řubat 2016

Eđitmenler

Mustafa Gke Baydođan
Berk Orbay

R Hakkında

R kodlarını çalıştırmak için indirmeniz gereken temel programı <http://cran.r-project.org> adresinden indirebilir veya daha gelişmiş bir geliştirici ortamı için R Studio'yu (<http://www.rstudio.com>) kullanabilirsiniz.

R'a Hızlı Giriş Dökümanı Hakkında

Bu döküman size R hakkında bilmeniz gereken temel unsurları anlatacaktır. R'ın vektörel yapısından başlayarak bazı temel istatistiksel testler yapmanıza yardımcı olacaktır. Önerimiz bu dökümandaki her adımı kendi R ortamınızda da tekrar etmenizdir.

1 R ve Vektörel Çalışma

1.1 Vektör Oluşturma

Bir değişkene bir değer atamak (ör. `x`'e 3 atamak gibi), aşağıdaki şekillerde gerçekleştirilebilir:

Kod 1: Değer atama yöntemleri 1

```
1 x <- 3
```

veya

Kod 2: Değer atama yöntemleri 2

```
1 x = 3
```

Bundan sonraki örneklerde değer atamaları için `<-` operatörünü kullanacağız.¹

Bir değişkene bir değer atadığımızda, R bu değişkeni tek elemanı olan bir vektör olarak algılamaktadır. Değişken adıyla birlikte `[.]` kullanarak aynı değişkende istediğimiz sıraya başka bir eleman atayabiliriz. Son olarak değişkenin durumunu görmek için basitçe değişkenin ismini yazıp Enter'a basabiliriz.

Kod 3: İçerik görüntüleme

```
1 x[4] <- 7.5
2 x # x'in içeriğini görmek için değişken ismini yazıp enter'a
   basmamız yeterli
3 # [1] 3.0 NA NA 7.5
```

Burada, ikinci ve üçüncü elemanlarda görülen `NA`'ın anlamı *not available* yani “geçersiz değer” anlamındadır. Aslında değişkenin ikinci ve üçüncü elemanlarına herhangi bir değer atamadığımız için R, vektörü tamamlamak adına `NA` göstermektedir.

Kodlarımızın içine yorum eklemek için yorumunuzun başına `#` sembolünü koyabilirsiniz. Bir satırın içinde `#` sembolünden sonra gelen her şey yorum olarak algılanacaktır. Yorumlarınızı kapatmak için tekrar bir şey yapmanıza gerek yoktur. Yalnız R'da bir yorum paragrafını belirtmenin kolay bir yolu yoktur. Yorum yazdığınız her satıra `#` sembolünüzü eklemeniz gerekmektedir.

Ardışık gelen bir sıra tam sayıyı çok basit bir komut ile bir vektöre atayabiliriz. Bu ardışık sıranın başlangıç ve bitiş değerlerini yazıp arasına `:` koymamız yeterlidir. Artan veya azalan bir sıra oluşturulabilir.

¹`<-` ve `=` atama operatörleri arasında bazı önemli farklar bulunmaktadır ama bu farklar bu dökümanda bulunan örnekleri etkilemeyecektir. Yine de aradaki farklar hakkında bilgi edinmek isterseniz <http://stat.ethz.ch/R-manual/R-patched/library/base/html/assignOps.html> adresine göz atabilirsiniz.

Kod 4: Ardışık tamsayı vektörü oluşturma

```
1 x <- 1:8 # ardışık tam sayı vektörü oluşturur
2 x
3 # [1] 1 2 3 4 5 6 7 8
4 y <- 15:11 # azalan bir ardışık tam sayı vektörü oluşturur
5 y
6 # [1] 15 14 13 12 11
```

Sıradaki işlem vektördeki her değere 3 ekleyecektir ve yeni oluşturulan vektörü textttty adıyla saklayacaktır.

Kod 5: Vektör elemanlarına değer ekleme

```
1 y <- x+3
2 y
3 # [1] 4 5 6 7 8 9 10 11
4 x
5 # [1] 1 2 3 4 5 6 7 8
```

Önceki komutun aslında yaptığı şey 8 eleman uzunluğunda bir vektörü bir eleman uzunluğunda bir vektörle toplamaktır. R iki farklı uzunlukta olan vektörle işlem yaparken kısa olan vektörü uzun olan vektör ile aynı boyuta erişinceye kadar otomatik olarak tekrarlar.

Kod 6: Farklı uzunluktaki iki vektörün toplamı 1

```
1 x <- 1:8
2 y <- 1:4
3 x
4 # [1] 1 2 3 4 5 6 7 8
5 y
6 # [1] 1 2 3 4
7 x+y # toplamı başka bir değişkene atamadan da görebiliriz
8 # [1] 2 4 6 8 6 8 10 12
```

Bu toplam işleminde y değişkeni, x'in vektör uzunluğu olan 8. elemana kadar tekrarlanır. Diğer bir deyişle x'in 5. elemanı y'nin ilk elemanı ile toplanır, 6. elemanı ikinci elemanla toplanır ve devam eder. Peki iki vektörün uzunlukları oranı tam sayı değilse ne olur beraber deneyip görelim.

Kod 7: Farklı uzunluktaki iki vektörün toplamı 2

```
1 x <- 1:8
2 y <- 1:3
3 x+y
4 # [1] 2 4 6 5 7 9 8 10
5 # Uyarı mesajları:
```

```
6 # In x + y : uzun olan nesne uzunluğu kısa olan nesne uzunluğ  
    onun bir katı değil
```

R bu durumda da y 'yi uzun olan vektörün uzunluğuna eşitleyene kadar tekrarlar ama son tekrar y 'nin bütün uzunluğunu yansıtmayabilir. R bir uyarı mesajı verir ama yine de işlemi yapar.

Ayrıca, çıkarma, bölme, çarma, üslü sayılar ve modüler aritmetik gibi işlemlerde de aynı mantık geçerlidir. Bu tür işlemlere Bölüm 1.4 içerisinde değineceğiz.

Aynı zamanda önceden belirlenmiş değerlerle de kolayca yeni vektörler oluşturabiliriz. Örneğin, 4, 8, 15, 16, 23, 42 değerleriyle 6 elemanlı bir vektör ve 501, 505, 578, 586 değerleriyle de 4 elemanlı farklı bir vektör oluşturabiliriz. Bu vektörleri oluşturmak için `c(.)` (“c”ombine / birleştir) fonksiyonundan yararlanacağız. Bir vektörün içindeki eleman sayısını öğrenmek için de `length(.)` (uzunluk) komutundan yararlanabiliriz. Ayrıca bir vektörün içindeki tekil (unique) değerleri bulmak için de `unique(.)` komutunu kullanabiliriz. Eğer bu tekil değerlerin vektör içinde kaç kere geçtiği ile ilgileniyorsak `table(.)` fonksiyonunu kullanırız.

Kod 8: Birleştirme `c(.)` (“c”ombine) fonksiyonu

```
1 x <- c(4, 8, 15, 16, 23, 42) # c(.) fonksiyonuyla değerleri birleş  
    triyoruz  
2 y <- c(501, 505, 578, 586)  
3 x  
4 # [1]  4  8 15 16 23 42  
5 y  
6 # [1] 501 505 578 586  
7 z <- c(x, y) # ayrıca iki vektörü de c(.) ile birleştirerek yeni  
    bir vektör oluşturabiliriz  
8 z  
9 # [1]  4  8 15 16 23 42 501 505 578 586  
10  
11 length(z) # oluşturulan z vektörünün uzunluğu  
12 # [1] 10  
13 m <- c(1, 5, 1, 4, 7, 4, 1)  
14 unique(m) # oluşturulan m vektöründeki tekil değerler  
15 # [1] 1 5 4 7  
16 table(m) # oluşturulan m vektöründeki tekil değerlerin kaç tane  
    olduğu  
17 # m  
18 # 1 4 5 7  
19 # 3 2 1 1
```

Bir vektörün eleman sırasını `rev()` (“rev”erse / tersine çevir) komutuyla tersine çevirebiliriz.

Kod 9: Tersine çevirme `c(.) rev()` (“rev”erse) fonksiyonu

```
1 z <- rev(z) # aynı vektörü kullanarak eleman sırasını deęiş
  tirebiliriz
2 z
3 # [1] 586 578 505 501 42 23 16 15 8 4
```

Diyelim ki bütün elemanlarının değeri 5 olan 10 eleman uzunluęunda bir vektör oluşturmak istiyoruz. Bunu `rep(.)` (“rep”eat / tekrar et) fonksiyonunu kullanarak kolayca yapabiliriz. Bu yöntemle aynı zamanda vektörleri de tekrar edebiliriz.

Kod 10: Tekrar etme `rep(.)` (“rep”eat) fonksiyonu

```
1 x <- rep(5,10) # 5 deęerini 10 kez tekrar et
2 x
3 # [1] 5 5 5 5 5 5 5 5 5 5
4 y <- c(3,5,7) # y vektörü
5 z <- rep(y,4) # y vektörünü 4 kez tekrar et
6 z
7 # [1] 3 5 7 3 5 7 3 5 7 3 5 7
8 rep(y,c(2,3,5)) # y nin elemanlarını teker teker
9                  # belirlenen deęerlerde tekrar et
10 # [1] 3 3 5 5 5 7 7 7 7 7
```

Önceki örnekte gördüğümüz gibi bir vektörü veya vektörün içindeki elemanları da tekrar etmemiz mümkün. Bu bölümdeki son örnek olarak 2 ve 3 arasındaki eşit aralıkta deęerlerden oluşan 21 eleman uzunluęunda bir vektör yaratacağız. Kullanacağımız fonksiyon `seq(.)` (“seq”uence, seri) olacak.

Kod 11: Eşit aralıklı elemanlardan oluşan bir dizi `seq(.)` (“seq”uence) yaratma fonksiyonu 1

```
1 x <- seq(2,3,length.out=21) # length.out vektorun uzunlugunu
  belirleyen parametre adidir
2
3 # [1] 2.00 2.05 2.10 2.15 2.20 2.25 2.30 2.35 2.40 2.45 2.50
4 # [12] 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 2.95 3.00
```

Eğer vektör uzunluęu yerine adım büyüklüğünü vermek istersek `length.out` yerine `by` parametresini kullanmamız gerekir.

Kod 12: Eşit aralıklı elemanlardan oluşan bir dizi `seq(.)` (“seq”uence) yaratma fonksiyonu 2

```
1 x <- seq(2,3,by=0.05)
```

```

2 x
3 # [1] 2.00 2.05 2.10 2.15 2.20 2.25 2.30 2.35 2.40 2.45 2.50
4 # [12] 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 2.95 3.00

```

1.2 Mantıksal İfadeler

Mantıksal ifadeleri oluşturmak için aşağıdaki mantıksal işlemleri kullanabiliriz. Bu işlemlerin sonucunda TRUE (doğru) ve FALSE (yanlış) içeren vektörler göreceğiz.

- `<` : küçüktür
- `<=`: küçük eşittir
- `>` : büyüktür
- `>=`: büyük eşittir
- `==`: eşittir (Uyarı: Tek = işareti atama işlemi için kullanılmaktadır. Bu hata sıkça yapılır.)
- `!=`: eşit değildir

Sıradaki örneklerimizde, önce bir vektör yaratacağız ve değişik mantıksal ifadelerde kullanacağız. Eğer bir vektör elemanı ifadeyi sağlıyorsa TRUE değerini, sağlamıyorsa da FALSE değerini dönecektir. Bunları 1 ve 0 değerlerinden oluşan bir vektör olarak da düşünebiliriz.

Ayrıca birden fazla koşullu bir mantıksal ifadeye ihtiyacımız olduğunda `&` sembolünü “ve” anlamında ve `|` sembolünü de “veya” anlamında kullanabiliriz.

Bunların dışında belli bir koşulu sağlayan vektör değerlerinin hangi indekslerde (hücre numaralarında) geçtiğini öğrenmek için `which(.)` fonksiyonundan faydalanırız.

Bazı durumlarda iki ayrı vektörde tutulan elemanların birbirine benzeyip benzemediğiyle ilgilenebiliriz. Bu durumlarda `%in%` operatörünü kullanırız. Örneğin 90dan 120ye kadar olan sayıları içeren bir vektörün elemanları 10dan 100e kadar sayıları içeren bir vektörün içinde ortak olarak 90, 91, ..., 100 sayıları gözlemlenir. Bu sayıları bulmak için `which(.)` fonksiyonu ve `%in%` operatörünü kullanabiliriz.

Kod 13: Mantıksal ifadeler 1

```

1 x <- 10:20
2 x
3 # [1] 10 11 12 13 14 15 16 17 18 19 20
4 x<17 # x'in hangi değerleri 17den küçüktür
5 # [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE
   FALSE FALSE

```

```

6 x<=17 # x'in hangi deęerleri 17den kk eřittir
7 # [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE
  FALSE FALSE
8 x>14 # x'in hangi deęerleri 14ten bktr
9 # [1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE
  TRUE TRUE
10 x>=14 # x'in hangi deęerleri 14ten bk eřittir
11 # [1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
  TRUE TRUE
12 x==16 # x'in hangi deęerleri 16ya eřittir
13 # [1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
  FALSE FALSE
14 x!=16 # x'in hangi deęerleri 16ya eřit deęildir
15 # [1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE
  TRUE TRUE
16
17 (x<=16) & (x>=12) # x'in hangi deęerleri 16dan kk eřit ve 12
  den bk eřittir
18 # [1] FALSE FALSE TRUE TRUE TRUE TRUE TRUE FALSE FALSE
  FALSE FALSE
19 (x<=11) | (x>=18) # x'in hangi deęerleri 11den kk eřit veya
  18den bk eřittir
20 # [1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE
  TRUE TRUE
21
22 ind=which(x<17) # x'in 17den kk deęerlerinin indeksleri
  nedir
23 ind
24 # [1] 1 2 3 4 5 6 7
25 ind2=which(x==16) #x'in hangi indeksteki deęeri 16ya eřittir
26 ind2
27 # [1] 7
28 v1=90:120 #v1'in hangi elemanları v2 iinde geer
29 v2=10:100
30 ind3=which(v1 %in% v2)
31 ind3
32 # [1] 1 2 3 4 5 6 7 8 9 10 11 # v1'in ilk 11 elemanı
  v2 ile ortak

```

Peki bu mantıksal ifadeler ile ne gibi bařka iřlemler yapılabilir? Basit bir ilk rnek olarak 1den 20ye kadar deęerler ieren bir x vektr oluřturup 8den kk her elemanını 0 haline getirebiliriz. Bylece 8den kk elemanlar 0 deęeri alırken dięer elemanlar deęerlerini koruyacaklardır.

Kod 14: Mantıksal ifadeler 2

```

1 x <- 1:20
2 y <- (x>=8) * (x)
3 y
4 # [1] 0 0 0 0 0 0 0 0 8 9 10 11 12 13 14 15 16 17
5 # [18] 18 19 20

```

İkinci ve daha gerçek hayattan bir örnek olarak, bazı ürünlerin sipariş maliyetlerini değerlendirebiliriz. Diyelim ki tedarikçimizden tek bir siparişte en az 30 adet en fazla da 50 adet ürün isteyebiliriz.

Eğer 45 adet veya daha az ürün sipariş edersek sabit maliyetimiz 50TL, daha fazla sipariş edersek de 15TL olsun.

Ürünün birim maliyeti; eğer 40 adetten daha az sipariş edersek 7TL, daha fazla sipariş edersek 6.5TL olsun.

Her sipariş alternatifi için maliyet hesabını yapalım.

Kod 15: Mantıksal ifadeleri kullanan maliyet hesabı örneği

```

1 siparisMiktari <- 30:50
2 birimMaliyet <- 7*siparisMiktari*(siparisMiktari<40)+6.5*
   siparisMiktari*(units>=40)
3 birimMaliyet
4 # [1] 210.0 217.0 224.0 231.0 238.0 245.0 252.0 259.0
5 # [9] 266.0 273.0 260.0 266.5 273.0 279.5 286.0 292.5
6 # [17] 299.0 305.5 312.0 318.5 325.0
7
8 sabitMaliyet <- 50*(siparisMiktari<=45)+15*(siparisMiktari>45)
9 sabitMaliyet
10 # [1] 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 15
11 # [18] 15 15 15 15
12
13 toplamMaliyet <- sabitMaliyet + birimMaliyet
14 toplamMaliyet
15 # [1] 260.0 267.0 274.0 281.0 288.0 295.0 302.0 309.0
16 # [9] 316.0 323.0 310.0 316.5 323.0 329.5 336.0 342.5
17 # [17] 314.0 320.5 327.0 333.5 340.0

```

Önceki örneğe bakarak diyelim ki 318TL'nin üzerindeki maliyetler bizi aşıyor. Bu koşullar altında sadece sipariş verebileceğimiz miktarları ve o siparişlere denk gelen maliyetleri görmek istiyoruz.

Kod 16: Mantıksal ifadeleri kullanan maliyet hesabı örneğine devam ediyoruz

```

1 siparisMiktari[toplamMaliyet<=318]
2 #318TL'den daha dusuk maliyete sahip olan siparis miktarlarini
   gosterir
3
4 # [1] 30 31 32 33 34 35 36 37 38 40 41 46
5 toplamMaliyet[toplamMaliyet<=318]
6 #318TL'den daha dusuk maliyete sahip olan siparis maliyetlerini
   gosterir
7 # [1] 260.0 267.0 274.0 281.0 288.0 295.0 302.0 309.0
8 # [9] 316.0 310.0 316.5 314.0

```

Yukarıdaki örnekteki iki komuttan ilki siparisMiktari vektörünün sadece 318TL'ye eşit veya daha düşük maliyete sahip elemanlarını getirdi. Diğer ise toplamMaliyet vektörünün sadece 318TL'ye eşit veya daha düşük maliyete sahip elemanlarını getirdi.

Önceki örnekte yaptığımız gibi bir vektörün bir parçasını değişik şekillerde çıkarabiliriz. Aşağıda bulunan örnekleri inceleyelim:

Kod 17: İndeksleme ve vektör elemanlarına erişim

```

1 x <- seq(5,8,by=0.3) # 11 elemanlı bir vektör oluşturunuz
2 x
3 # [1] 5.0 5.3 5.6 5.9 6.2 6.5 6.8 7.1 7.4 7.7 8.0
4 length(x)
5 # [1] 11
6
7 y1 <- x[3:7] # 3üncü elemanından 7inci elemana kadar olan kısmı
   nı alıyoruz
8 y1
9 # [1] 5.6 5.9 6.2 6.5 6.8
10
11 y2 <- x[2*(1:5)] # çift sayı sırasındaki elemanları alıyoruz 2
   inci, 4üncü gibi
12 y2
13 # [1] 5.3 5.9 6.5 7.1 7.7
14
15 y3 <- x[-1] # ilk elemanı çıkarıp geri kalanı alıyoruz
16 y3
17 # [1] 5.3 5.6 5.9 6.2 6.5 6.8 7.1 7.4 7.7 8.0
18
19 y4 <- x[-length(x)] # son elemanı çıkarıp geri kalanı alıyoruz
20 y4
21 # [1] 5.0 5.3 5.6 5.9 6.2 6.5 6.8 7.1 7.4 7.7
22
23 y5 <- x[-seq(1,11,3)] # belirtilen elemanları çıkarıp geri
   kalanı alıyoruz

```

```

24 y5
25 # [1] 5.3 5.6 6.2 6.5 7.1 7.4 8.0
26
27 y6 <- x[c(1,3,7)] # sadece birinci, üçüncü ve yedinci elemanlar
    1 alıyoruz
28 y6
29 # [1] 5.0 5.6 6.8
30
31 y7 <- x[seq(1,11,3)] # sadece belirtilen elemanları alıyoruz
32 y7
33 # [1] 5.0 5.9 6.8 7.7

```

1.3 Matrisler

Bölmeler 1.1 ve 1.2 içerisinde verdiğimiz örneklerde kullandığımız her vektör aslında varsayılan olarak dikey vektör olarak tanımlanmıştır. Vektörün yatay olarak gösterilmesinden dolayı şu anda kafanız karışmış olabilir. Yatay bir vektör oluşturmak için `t()` ("t"ranspose / döndürme) komutundan yararlanabiliriz.

Kod 18: Matris `t()` ("t"ranspose) transpozu 1

```

1 x <- 1:5
2 y <- t(x)
3 y
4 #      [,1] [,2] [,3] [,4] [,5]
5 # [1,]    1  2  3  4  5

```

Gördüğümüz gibi R yatay vektörü tamamen değişik bir şekilde gösteriyor. Eğer yataya döndürdüğümüz `y` vektörünü tekrar döndürürsek dikey vektörün gerçek gösterimini görebiliriz.

Kod 19: Matris `t()` ("t"ranspose) transpozu 2

```

1 t(y) #veya sadece t(t(x)) de yazabilirdik
2 #      [,1]
3 # [1,]    1
4 # [2,]    2
5 # [3,]    3
6 # [4,]    4
7 # [5,]    5

```

R'da $m \times n$ bir matris yaratmak için önce bir vektör yaratmamız gerekiyor (diyelim ismi `vec` olsun). Bu vektör ilk sütunun tepesinden başlayarak son sütunun aşağısına kadar matrisin değerlerini oluşturur. Matris yaratmak için son derece açık olan `matrix(vec, nrow=m, ncol=n)` fonksiyonunu kullanacağız (`nrow` satır sayısı, `ncol` sütun sayısını belirten parametrelerdir).

Kod 20: Satır ve sütun sayısını belirterek matris yaratma

```

1 vec <- 1:12
2 x <- matrix(vec,nrow=3,ncol=4)
3
4 x #rakamların sırasına dikkat
5 #      [,1] [,2] [,3] [,4]
6 # [1,]  1  4  7   10
7 # [2,]  2  5  8   11
8 # [3,]  3  6  9   12
9
10 t(x) #döndürülmüşü
11 #      [,1] [,2] [,3]
12 # [1,]  1  2  3
13 # [2,]  4  5  6
14 # [3,]  7  8  9
15 # [4,] 10 11 12

```

Eğer sizin için yukarıdan aşağıya sütunları doldurmaktansa, soldan sağa satırları doldurmak daha önemliyse byrow parametresini TRUE olarak belirleyebilirsiniz.

Kod 21: Satır satır matris verisi oluşturma

```

1 vec <- 1:12
2 x <- matrix(vec,nrow=3,ncol=4,byrow=TRUE)
3 x
4 #      [,1] [,2] [,3] [,4]
5 # [1,]  1  2  3  4
6 # [2,]  5  6  7  8
7 # [3,]  9 10 11 12

```

$n \times n$ bir matrisin tersini solve() fonksiyonuyla alıyoruz.

Kod 22: Matris tersi

```

1 x <- matrix(c(1,2,-1,1,2,1,2,-2,-1),nrow=3,ncol=3)
2 x
3      [,1] [,2] [,3]
4 [1,]  1  1  2
5 [2,]  2  2 -2
6 [3,] -1  1 -1
7
8 xinv <- solve(x)
9 xinv
10 #      [,1]      [,2] [,3]
11 # [1,] 0.0000000 0.2500000 -0.5
12 # [2,] 0.3333333 0.0833333 0.5

```

```
13 # [3,] 0.3333333 -0.1666667 0.0
```

Bütün elemanları aynı olan bir matris yaratmak için `matrix()` fonksiyonuna tek bir değer atamak yeterlidir.

Ayrıca matrisin köşegenine değer atamak için de `diag()` fonksiyonunu kullanıyoruz.

Kod 23: Matris köşegen değerlerine erişim için `diag()` fonksiyonu

```
1 x <- matrix(0,nrow=4,ncol=4)
2 x
3 #      [,1] [,2] [,3] [,4]
4 # [1,]    0  0  0  0
5 # [2,]    0  0  0  0
6 # [3,]    0  0  0  0
7 # [4,]    0  0  0  0
8
9 diag(x) <- 1 # matris köşegenin bütün değerlerini 1 yapar
10 x
11 #      [,1] [,2] [,3] [,4]
12 # [1,]    1  0  0  0
13 # [2,]    0  1  0  0
14 # [3,]    0  0  1  0
15 # [4,]    0  0  0  1
```

Bir matrisin içindeki eleman sayısı `length()` ile öğrenilebilir, eğer sadece sütun sayısı gerekiyorsa `ncol()`, sadece satır sayısı gerekiyorsa `nrow()` veya ikisini birden öğrenmek için `dim()` kullanılabilir.

Kod 24: Matris boyutunu döndüren fonksiyonlar

```
1 x <- matrix(0,ncol=5,nrow=4)
2 ncol(x)
3 # [1] 5
4 nrow(x)
5 # [1] 4
6 length(x)
7 # [1] 20
8 dim(x)
9 # [1] 4 5
```

1.4 R'da Aritmetik İşlemler

Bölüm 1.1 içerisinde aritmetik işlemlere ufak bir giriş yapmıştık. İki vektörü çarpıp toplayabilir ve aynı zamanda çıkarma, bölme ve modüler aritmetik (burada modüler arit-

metik olarak bahsedilen kalan işlemi %% sembolüyle yapılmaktadır) işlemlerini yapabileceğimizden bahsetmiştik.

Kod 25: Aritmetik İşlemler

```
1 x <- 2*(1:5)
2 x
3 # [1] 2 4 6 8 10
4 y <- 1:5
5 y
6 # [1] 1 2 3 4 5
7 x+y
8 # [1] 3 6 9 12 15
9 x*y
10 # [1] 2 8 18 32 50
11 x/y
12 # [1] 2 2 2 2 2
13 x-y
14 # [1] 1 2 3 4 5
15 x^2 # üslü sayı işlemi
16 # [1] 4 16 36 64 100
17 x^y
18 # [1] 2 16 216 4096 100000
19 x%%3 # mod(3) ile kalan hesabı
20 # [1] 2 1 0 2 1
```

Kod 26: Kalan işlemi

```
1 y <- 3:7
2 y
3 # [1] 3 4 5 6 7
4
5 x%%y # iki vektörde de her eleman için kalan işlemi
6 # [1] 2 0 1 2 3
7 x%/y # bölme işleminin tam sayı kısmı
8 # [1] 0 1 1 1 1
```

Önceki örnekte x ve y dikey vektörlerdir. Eğer bir tanesi bile yatay vektör olarak tanımlanmış olsaydı, R bu işlemleri gene yapardı ancak bu sefer sonuçlar yatay vektör olarak çıkardı.

Bir vektörün maksimum değeri `max()` ile minimum değerini ise `min()` ile bulunabilir. Bir vektörün bütün elemanlarını toplamak istersek `sum()`, çarpımını bulmak istersek de `prod()` fonksiyonlarını kullanabiliriz.

Kod 27: Vektör elemanlarını karşılaştırma işlemleri

```

1 x <- c(3,1,6,5,8,10,9,12,3)
2
3 min(x)
4 # [1] 1
5
6 max(x)
7 # [1] 12
8
9 sum(x)
10 # [1] 57
11
12 prod(x)
13 # [1] 2332800

```

Eğer iki vektörün karşılıklı elemanlarından hangileri en büyük ve hangileri en küçük öğrenmek istersek `pmax()` ve `pmin()` fonksiyonlarını kullanabiliriz.

Kod 28: Farklı vektörleri eleman eleman karşılaştırma

```

1 x <- 1:10
2 y <- 10:1
3 z <- c(3,2,1,6,5,4,10,9,8,7)
4
5 a <- pmax(x,y,z) # istediğimiz sayıda vektör yazabiliriz
6 a
7 # [1] 10 9 8 7 6 6 10 9 9 10
8
9 b <- pmin(x,y,z)
10 b
11 # [1] 1 2 1 4 5 4 4 3 2 1

```

Eğer bir vektörün içindeki değerleri sıralamak istersek `sort()`, `rank()` ve `order()` fonksiyonlarını kullanabiliriz. `sort()` ve `order()` varsayılan olarak küçükten büyüğe sıralar. Eğer büyükten küçüğe sıralamak istersek fonksiyonların içine `decreasing = TRUE` parametresini eklemek sorunumuzu çözer.

Kod 29: Verileri sıralama işlemleri

```

1
2 veri<-c(5,32,6,11,43,11,4,3,2,8)
3
4 #Sort bize güzel bir sıralı liste verir
5 sort(veri)
6 # [1] 2 3 4 5 6 8 11 11 32 43
7
8 #Order bize sıralamanın hücre numaralarını (indeks) verir

```

```

9 order(veri)
10 # [1]  9  8  7  1  3 10  4  6  2  5
11
12 #Yani en küçük rakam 9uncu hücre olan 2, diğeri 8. hücre olan 3
    ve devam eder
13
14 #Veriyi sıraya sokmak için yapılacak iş basit.
15
16 veri[order(veri)]
17 # [1]  2  3  4  5  6  8 11 11 32 43
18
19 #Büyükten küçüğe sıralamak için decreasing parametresini kullan
    ıyoruz
20 order(veri, decreasing=TRUE)
21 #[1]  5  2  4  6 10  3  1  7  8  9
22
23 #Rank komutu ise o hücrenin kaçınıcı sırada olduğunu söyler.
24 rank(veri)
25 #[1]  4.0  9.0  5.0  7.5 10.0  7.5  3.0  2.0  1.0  6.0
26
27 #Farkettiyseniz iki tane 7.5 var. Bunun sebebi iki tane 11
    olması.
28 #7. ve 8. sıradaki yerleri işgal ettiğinden otomatik olarak
29 #ortalama değeri alıyorlar. Bunu değiştirmenin yolu ise söyle
30
31 #First metodu hücre sırası önce gelen değeri daha üst sırada gö
    sterir. ilk 11
32 #7inci sırada ikinci 11 8inci sırada.
33 rank(veri,ties.method="first")
34 #[1]  4  9  5  7 10  8  3  2  1  6
35
36 #Random bu işi rastgele yapar.
37 rank(veri,ties.method="random")
38 # [1]  4  9  5  7 10  8  3  2  1  6
39 rank(veri,ties.method="random")
40 # [1]  4  9  5  8 10  7  3  2  1  6
41
42 #Max en yüksek sıralamayı verir.
43 rank(veri,ties.method="max")
44 # [1]  4  9  5  8 10  8  3  2  1  6
45
46 #Min en düşük sıralamayı verir.
47 rank(veri,ties.method="min")
48 # [1]  4  9  5  7 10  7  3  2  1  6
49

```



```

50 #Average ise ilk baştaki 7.5lu vektörün aynısını verir.
51 rank(veri,ties.method="average")
52 # [1]  4.0  9.0  5.0  7.5 10.0  7.5  3.0  2.0  1.0  6.0
53
54 #rankte orderda olan büyükten küçüğe sıralama parametresi
   yoktur.
55 #Onu da ufak bir numara ile yapabiliyoruz :)
56 rank(-veri)
57 # [1]  7.0  2.0  6.0  3.5  1.0  3.5  8.0  9.0 10.0  5.0

```

R'da matris çarpımı yapmak için düz çarpım işlemi sembolü `*` yerine özel bir sembol olan `%*%` kullanmalıyız. Matris çarpım kuralları burada da işlemektedir ama matris boyutları uyumlu olmadığında R bazen otomatik düzeltmeler yapıp bir sonuç verebilir.

Kod 30: Matris çarpımı 1

```

1 x <- matrix(1:6,ncol=2,nrow=3)
2 x
3 #      [,1] [,2]
4 # [1,]  1  4
5 # [2,]  2  5
6 # [3,]  3  6
7
8 y <- matrix(1:4,ncol=2,nrow=2)
9 y
10 #      [,1] [,2]
11 # [1,]  1  3
12 # [2,]  2  4
13
14 x%*%y
15 #      [,1] [,2]
16 # [1,]   9  19
17 # [2,]  12  26
18 # [3,]  15  33
19
20 y%*%x
21 # Hata oluştu y %*% x : uygun olmayan argümanlar
22
23 y%*%t(x) # x'i döndürmek sonuç verecektir
24 #      [,1] [,2] [,3]
25 # [1,]  13  17  21
26 # [2,]  18  24  30

```

Şimdi iki dikey vektörün matris çarpımı ile ilgili bir örnek düşünelim. R birinci vektörü yatay bir vektör olarak düzelterip sonucu tek bir değer olarak döndürür. Eğer iki yatay vektör için matris çarpımı yapacak olsaydık R hata verecekti. Eğer vektörlerden ilkinin

dikey ($m \times 1$) diğerini yatay ($1 \times n$) olarak alacak olursak $m \times n$ bir matris elde etmiş oluruz.

Kod 31: Matris çarpımı 2

```
1 x <- 1:3
2 y <- 3:1
3
4 x%*%y # R bir düzeltme yaparak ilk vektörü yatay hale getiriyor
5 #      [,1]
6 # [1,]    10
7
8 t(x)%*%y # yukarıdaki örnek ile aynı sonucu veriyor
9 #      [,1]
10 # [1,]    10
11
12
13 t(x)%*%t(y)
14 # Hata oluştu t(x) %*% t(y) : uygun olmayan argümanlar
15
16 x%*%t(y) # bu alternatif de bir mxn matrisi oluşturur
17 #      [,1] [,2] [,3]
18 # [1,]    3  2  1
19 # [2,]    6  4  2
20 # [3,]    9  6  3
```

Gerçek sayılardan oluşan bir matriste biriken (kümülatif) bir toplam ve çarpım elde etmek için `cumsum()` ve `cumprod()` fonksiyonlarını kullanabiliriz. Ayrıca `diff()` fonksiyonu vektörün ardışık elemanları arasındaki farkı verir.

Kod 32: Matrislerde biriken (kümülatif) toplam ve çarpım işlemleri

```
1 x <- c(1,4,5,6,2,12)
2 y <- cumsum(x)
3 Y
4 # [1]  1  5 10 16 18 30
5 # cumsum ile her hücre kendisinden önceki hücrelerin toplamını
6   kendi değerine ekler
7
8 Z <- cumprod(x)
9 Z
10 # [1] 1 4 20 120 240 2880
11 # cumprod ile her hücre kendisinden önceki hücrelerin çarpımını
12   kendi değeriyle çarpar
13
14 diff(x)
15 # [1]  3  1  1 -4 10
```

Sayılarla ilgili diğer önemli fonksiyonlar

- `factorial()` faktöriyel ($x!$)
- `abs()` mutlak değer ($|x|$)
- `sqrt()` karekök (\sqrt{x})
- `log()` logaritma (eğer taban belirtilmezse varsayılan değer doğal logaritmadır) ($\log x$)
- `exp()` euler sayısı ile üssel işlem (e^x)
- `gamma()` gamma fonksiyonu ($\Gamma(x)$)
- `round()` tam sayıya yuvarlama ²
- `floor()` tamsayıya aşağı yuvarlama
- `ceiling()` tam sayıya yukarı yuvarlama
- `as.integer()` tam sayıya çevirir (sayılar için `floor` ile hemen hemen aynı işleve sahiptir)

Bu fonksiyonları hem tek değer hem de birden çok değer içeren vektörlerde uygulayabilirsiniz.

Kod 33: Matematiksel işlemler

```
1 factorial(3)
2 # [1] 6
3 factorial(1:6)
4 # [1] 1 2 6 24 120 720
5
6 abs(-4)
7 # [1] 4
8 abs(c(-3:3))
9 # [1] 3 2 1 0 1 2 3
10
11 sqrt(4)
12 # [1] 2
13 sqrt(1:9)
14 # [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
15 # [1] 2.645751 2.828427
16 # [9] 3.000000
```

²R'ın 0.5 değerleri için tam sayıya yuvarlama metodu farklıdır. Mesela R 1.5 ve 2.5'ü 2'ye yuvarlar ama 0.5 0'a yuvarlanır. Excel her buçuğu yukarı yuvarlar

```

17 log(100) # doğal logaritma
18 # [1] 4.60517
19 log10(100) # tabanı 10 olan logaritma
20 # [1] 2
21 log2(100) # tabanı 2 olan logaritma
22 # [1] 6.643856
23 log(100,5) # tabanı 5 olan logaritma
24 # [1] 2.861353
25 log(c(10,20,30,40))
26 # [1] 2.302585 2.995732 3.401197 3.688879
27
28 exp(4.60517) # yaklaşık olarak 100 değerini vermeli
29 # [1] 99.99998
30 exp(log(100)) # yuvarlama hataları olmadan
31 # [1] 100
32 exp(seq(-2,2,0.4))
33 # [1] 0.1353353 0.2018965 0.3011942 0.4493290 0.6703200
34 # [8] 2.2255409 3.3201169 4.9530324 7.3890561
35
36 gamma(5) # factorial(4) ile aynı olmalı
37 # [1] 24
38 gamma(5.5) # factorial(4.5) ile aynı olmalı
39 # [1] 52.34278
40
41 x <- c(-3,-3.5,4,4.2)
42 floor(x)
43 # [1] -3 -4 4 4
44 ceiling(x)
45 # [1] -3 -3 4 5
46 as.integer(x)
47 # [1] -3 -3 4 4

```

2 R’da Olasılık ve İstatistiksel İşlemler

2.1 Olasılık fonksiyonları

Temel (base) R paketinde neredeyse tüm temel olasılık dağılımları tanımlıdır. Aşağıda olasılık teorisi ve istatistikte sıkça kullanılan dört tane temel R fonksiyonu anlatılır. Bu fonksiyonların tanımları normal dağılım üzerinden açıklandıktan sonra R’da bulunan diğer olasılık dağılımları hakkında da bilgi vereceğiz.

- `dnorm(x, y, z)`: Ortalaması y and standart sapması z olan bir normal dağılımın x sayısındaki olasılık dağılım fonksiyon değerini döner.
- `pnorm(x, y, z)`: Ortalaması y and standart sapması z olan bir normal dağılımın x sayısındaki kümülatif dağılım fonksiyon değerini döner.
- `qnorm(x, y, z)`: Ortalaması y and standart sapması z olan bir normal dağılımın x olasılığındaki kümülatif dağılım fonksiyon değerinin tersini (*quantile*) döner. x bir olasılık olduğu için 0 ve 1 değerleri arasında olmalıdır. ($x \in [0, 1]$).
- `rnorm(x, y, z)`: Ortalaması y and standart sapması z olan bir normal dağılımdan x tane rastgele sayı üretir. Sonuç olarak x uzunluğunda bir dizi yaratır.

Normal dağılım ile ilgili aşağıdaki örneklerle bakacak olursak:

Kod 34: Olasılık fonksiyonları

```
1 dnorm(0.5) # ortalama ve standart sapma parametreleri tanı
   mlanmazsa
2           # R standart normal dağılım kullanır
3 # [1] 0.3520653
4 dnorm(0,2,1)
5 # [1] 0.05399097
6 dnorm(3,3,5)
7 # [1] 0.07978846
8
9 pnorm(0) # eğrinin altında kalan alan
10         # standart normal dağılımda "0" solundaki kalan alan
11 # [1] 0.5
12 pnorm(2)
13 # [1] 0.9772499
14 pnorm(5,3,1)
15 # [1] 0.9772499
16
17 # Önceki "pnorm()" fonksiyonlarının tersi işlemi yapma (
   quantile)
18 qnorm(0.5)
19 # [1] 0
20 qnorm(0.9772499)
```

```

21 # [1] 2.000001
22 qnorm(0.9772499, 3, 1)
23 # [1] 5.000001
24
25 rnorm(20, 2, 1) # ortalaması 2 standart sapması 1 olan
26                # normal dağılımdan 20 tane rastgele sayı
27 # [1] 2.31502453 0.37445729 2.04994863 1.89381118
28    0.63099383 1.50837615
29 # [7] 0.57363369 2.84601422 2.54003868 3.43652548
30    0.88941281 3.36373629
31 # [13] 0.58945290 2.44678124 -0.05360271 2.73920472
32    2.73643684 1.79465998
33 # [19] 1.30906099 2.18648566

```

R ile olasılık dağılım hesapları yapmak için faydalı olabilecek çeşitli olasılık dağılımlarının listesini aşağıda bulabilirsiniz. Aşağıda yazmayan fakat R’da tanımlı başka olasılık dağılımları da mevcuttur. Ayrıca aşağıdaki her olasılık dağılımının kümülatif yoğunluk fonksiyonu için d yerine p , kümülatif yoğunluk fonksiyonun tersi için q ve rastgele sayı üretmek için r kullanabiliriz.

- `dpois(x, y)` : y ortalamaya sahip Poisson dağılan x sayısının okf (olasılık kütle fonksiyonu / *probability mass function (pmf)*) değeri.
- `dbinom(x, y, z)` : deney sayısı y ve başarı olasılığı z olan Binom dağılımın x sayısı için okf değeri.
- `dgeom(x, y)` : başarı olasılığı z olan Geometrik dağılımın x sayısı için okf değeri.
- `dunif(x, y, z)` : alt sınırı y ve üst sınırı z olan Uniform dağılımın x sayısı için odf (olasılık dağılım fonksiyonu) değeri.
- `dexp(x, y)` : oran (rate) parametresi y olan Üstel dağılımın x sayısı için odf değeri.
- `dgamma(x, y, scale=z)` : şekil (shape) parametresi y and ölçek (scale) parametresi z olan Gamma dağılımın x sayısı için odf değeri.
- `dchisq(x, y, z)` : serbestlik derecesi (degrees of freedom) y ve merkezsizlik (non-centrality) parametresi z olan Ki-kare dağılımın x sayısı için odf değeri.
- `dt(x, y, z)` : serbestlik derecesi y ve merkezsizlik (non-centrality) parametresi z olan T dağılımın x sayısı için odf değeri.
- `df(x, y, z, a)` : ilk serbestlik derecesi y , ikinci serbestlik derecesi z ve merkezsizlik parametresi a olan F dağılımın x sayısı için odf değeri.
- `dcauchy(x, y, z)` : yer (location) parametresi y and ölçek parametresi z olan Cauchy dağılımın x sayısı için odf değeri.

- `dnbinom(x, y, z)` : dağılım (dispersion) parametresi y and başarı olasılığı olan Negatif Binom dağılımın x sayısı için okf değeri.
- `dhyper(x, y, z, a)` : Toplam beyaz top sayısı y ve siyah top sayısı z olan keseden, a sayıda top çekimini tanımlayan Hiper geometrik dağılımın x (beyaz top sayısı) sayısı için okf değeri.
- `dlnorm(x, y, z)` : log-ortalaması y and log-standart dağılımı z olan Log-normal dağılımın returns x sayısı için odf değeri.
- `dbeta(x, y, z)` : birinci şekil parametresi y and ikinci şekil parametresi z olan Beta dağılımın x sayısı için odf değeri.
- `dlogis(x, y, z)` : yer (location) parametresi y and ölçek parametresi z olan Lojistik dağılımın x sayısı için odf değeri.
- `dweibull(x, y, z)` : şekil (shape) parametresi y and ölçek (scale) parametresi z olan Weibull dağılımın x sayısı için odf değeri.

2.2 R ile İstatistiksel Fonksiyonlar

Bir dizideki sayıların ortalamasını `mean()`, standart sapmasını `sd()`, varyansını `var()` ve medyan değerini `median()` fonksiyonlarını kullanarak bulabiliriz. Ayrıca `summary()` fonksiyonu ile çeşitli yüzdelik dilimlere (percentiles) düşen değerler hakkında hakkında bilgi edinebiliriz (Örneğin

Kod 35: İstatistiksel fonksiyonlar

```

1 x <- rnorm(1000000, 5, 2) # x ortalaması 5, standart sapması 2
   olan normal dağılımdan
2                               # gelen 1000000 rastgele sayıdan oluşan
                               bir dizidir.
3
4 mean(x)
5 # [1] 4.997776
6 sd(x)
7 # [1] 2.000817
8 var(x)
9 # [1] 4.003268
10 median(x)
11 # [1] 4.997408
12 summary(x)
13 #   Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
14 # -4.904   3.650   4.997   4.998   6.346  14.420
15 summary(x, digits=6)
16 #   Min. 1st Qu.  Median    Mean 3rd Qu.   Max.
17 # -4.90360  3.65020  4.99741  4.99778  6.34564 14.42310

```

```

18 quantile(x) # bu fonksiyon çeyreklik dilimler hakkında da bilgi
    verir
19 #      0%      25%      50%      75%     100%
20 # -4.903599  3.650201  4.997408  6.345639 14.423129
21
22 # çeyreklik dilimleri aşağıdaki gibi de elde edebiliriz.
23 sort(x)[1000000*0.25]
24 # [1] 3.650189
25 sort(x)[1000000*0.5]
26 # [1] 4.997408
27 sort(x)[1000000*0.75]
28 # [1] 6.345639

```

Yukarıdaki komutları çalıştırdığınızda aynı sonuçlara ulaşamıyor olmanız beklenir. Bunun sebebi rastgele değişkenler üretilirken kullanılan başlangıç değeri (seed) ile alakalıdır. Örneğin her `rnorm()` fonksiyonun çalıştırdığınızda üretilen rassal sayılar birbirinden farklıdır. Eğer sonuçlarınızın tekrarlanabilmesi konusunda endişemiz olursa (Örneğin bilimsel bir makale için sonuç sunuyorsanız.) `set.seed()` fonksiyonunun içine seçeceğimiz bir sayıyı belirterek, yaratılacak rassal değişkenlerin sırasını sabitleyebiliriz. Bununla ilgili detaylı bilgi için <https://stat.ethz.ch/pipermail/r-help/2006-June/107399.html> bağlantısını ziyaret edebilirsiniz.

3 R ile Fonksiyon ve Döngü Tanımları

3.1 R’da fonksiyon tanımlamak

Yapmak istediğimiz hesapları yerine getiren halihazırda bir R fonksiyonu olmadığında kendi fonksiyonlarımızı tanımlarız. Fonksiyonların R’da tanımlanması için aşağıdaki yapıyı izlememiz gerekir.

Kod 36: R ile fonksiyon tanımı

```
1 # f <- function(p1,p2,...) # fonksiyon ismini ve girdilerini
2                             # (argüman veya parametre de denir)
3                             belirle
4 # {
5 #   girdileri (argümanları) ve diğer araçları kullanarak
6   hesapları yap
7 #   gerekliyse değerleri ekrana bas
8 #   gerekliyse görseller oluştur
9 #   sonuç değişkenini son satıra yaz ve fonksiyon bu değişkeni
10  dönsün
11 # }
```

Özetle yukarıdaki komutlar `f` fonksiyonunun girdilerini `(p1,p2,...)` olarak tanımlar. `f` fonksiyonu `{ }` içerisinde tanımlanan işleri yapar.

Aşağıdaki basit fonksiyon örnekleri ile R’da fonksiyonları nasıl yazılır anlamaya çalışacağız:

Kod 37: R ile fonksiyon örneği 1

```
1 # ÖRNEK 01
2 # Yarıçapı r olan bir çemberin çevresini ve dairenin alanını
3   hesaplayan bir fonksiyon
4 cevre_alan <- function(r) # yarıçap
5 {
6   cf <- 2*pi*r # çevreyi hesaplar, pi R’da tanımlı bir sabittir
7   a <- pi*r^2 # alanı hesaplar
8   res <- c(cf,a) # sonuçları birleştirir
9   names(res) <- c("çevre", "alan")
10  res
11 }
12 circle(3)
13 #      çevre      alan
14 # 18.84956    28.27433
15 circle(1)
16 #      çevre      alan
17 #  6.283185     3.141593
```

Kod 38: R ile fonksiyon örneği 2

```

1 # ÖRNEK 02
2 # Köşe koordinatları iki boyutlu uzayda belirlenmiş üçgenin
3 # çevresini ve alanını hesaplayan bir fonksiyon
4 ucgen <- function(
5   a, # birinci köşenin koordinatı (2 uzunluğunda bir dizi olmal
      1 (x,y))
6   b, # ikinci köşenin koordinatı (2 uzunluğunda bir dizi olmal
      1 (x,y))
7   c # üçüncü köşenin koordinatı (2 uzunluğunda bir dizi olmalı
      (x,y))
8 ){
9   if(length(a)!=2 || length(b)!=2 || length(c)!=2){
10     print("hata, en az bir koordinat hatalı ya da eksik girilmi
        5")
11   }
12 # çevre hesaplanır
13 ab <- sqrt((a[1]-b[1])^2+(a[2]-b[2])^2)
14 bc <- sqrt((c[1]-b[1])^2+(c[2]-b[2])^2)
15 ac <- sqrt((a[1]-c[1])^2+(a[2]-c[2])^2)
16 pm <- ab+bc+ac
17 # alan hesaplanır
18 trab <- abs((a[1]-b[1])*(a[2]-b[2]))/2
19 trbc <- abs((c[1]-b[1])*(c[2]-b[2]))/2
20 trac <- abs((a[1]-c[1])*(a[2]-c[2]))/2
21
22 maxx <- pmax(a,b,c)
23 minx <- pmin(a,b,c)
24
25 sqa <- min(max((a[1]-minx[1])*(a[2]-minx[2]),0),max((maxx
      [1]-a[1])*(maxx[2]-a[2]),0))
26 sqb <- min(max((b[1]-minx[1])*(b[2]-minx[2]),0),max((maxx
      [1]-b[1])*(maxx[2]-b[2]),0))
27 sqc <- min(max((c[1]-minx[1])*(c[2]-minx[2]),0),max((maxx
      [1]-c[1])*(maxx[2]-c[2]),0))
28 area <- (maxx[1]-minx[1])*(maxx[2]-minx[2])-trab-trbc-
      trac-sqa-sqb-sqc
29
30 pm <- (area!=0)*pm # if area=0, then there is no triangle
31
32 res <- c(pm,area)
33 names(res) <- c("çevre", "alan")
34 res
35 }
36

```

```

37 coora <- c(23,18)
38 coorb <- c(13,34)
39 coorc <- c(50,5)
40 ucgen(coora,coorb,coorc)
41 #  çevre      alan
42 #  95.84525 151.00000
43
44 coora <- c(10,18)
45 coorb <- c(13,34)
46 coorc <- c(50,5)
47 ucgen(coora,coorb,coorc)
48 #  çevre      alan
49 #  105.3489  339.5000

```

```

1 coora <- c(3,5)
2 coorb <- c(9,15)
3 coorc <- c(6,10)
4 ucgen(coora,coorb,coorc)
5 #  çevre      alan
6 #    0         0

```

Bölüm 1.2 içerisindeki sipariş maliyet problemini hatırlayalım. We will create a function that yields the output in case of a change in unit costs and ordering costs. In this function we will also assign default values to input parameters. So, whenever a parameter is undefined in the function call, R will assume the default value for this parameter.

Kod 39: R ile fonksiyon örneği 3

```

1 # ÖRNEK 03
2 siparisMaliyetListesi <- function(
3   huc=7,    # yüksek birim maliyet
4   luc=6.5,  # düşük birim maliyet
5   ucc=40,   # düşük birim maliyetten en az sipariş miktarı
6   hfc=50,   # yüksek sabit maliyet
7   lfc=15,   # düşük sabit maliyet
8   fcc=45,   # yüksek birim maliyetten en çok sipariş miktarı
9   tcub=318 # toplam maliyet için en üst (sınırlayıcı) seviye
10 ){
11   units <- 30:50
12   birimMaliyet <- huc*units*(units<ucc)+luc*units*(units>=ucc)
13   sabitMaliyet <- hfc*(units<=fcc)+lfc*(units>fcc)
14   toplamMaliyet <- sabitMaliyet+birimMaliyet
15   res <- toplamMaliyet[toplamMaliyet<=tcub]
16   names(res) <- units[toplamMaliyet<=tcub]
17   res

```

```

18 }
19
20 siparisMaliyetListesi() # önceki değerler ile aynı sonucu verir
21 # 30 31 32 33 34 35 36 37 38 40 41 46
22 # 260.0 267.0 274.0 281.0 288.0 295.0 302.0 309.0 316.0 310.0
   316.5 314.0
23
24 siparisMaliyetListesi(hfc=55,luc=6.3) # iki argümanın değerini
   değıştirelim
25 # 30 31 32 33 34 35 36 37 40 41 46 47 48
26 # 265.0 272.0 279.0 286.0 293.0 300.0 307.0 314.0 307.0 313.3
   304.8 311.1 317.4

```

Son örnek olarak if-else ifadelerinin R’da kullanımını göstermek için aşağıdaki örnek ile devam ediyoruz.

$$f(x) = \begin{cases} x^2 & x < -2 \\ x + 6 & -2 \leq x < 0 \\ -x + 6 & 0 \leq x < 4 \\ \sqrt{x} & x \geq 4 \end{cases}$$

Kod 40: R ile fonksiyon örneği 4

```

1 # ÖRNEK 04
2 f <- function(x) {
3   if(x < (-2)) {
4     x^2
5   } else if(x < 0) {
6     x+6
7   } else if(x < 4) {
8     -x+6
9   }
10  else {
11    sqrt(x)
12  }
13 }
14
15 c(f(-4), f(-1), f(3), f(9))
16 # [1] 16 5 3 3

```

Bunların dışında tanımlı R fonksiyonlarını yarattığınız fonksiyonların içinde argüman olarak kullanabilirsiniz. Bölüm 3.2 içerisinde bununla ilgili bir örnek yapacağız.

3.2 R’da Döngüler

Döngü içinde belirli sayıda iterasyon yapmak için aşağıdaki yapıyı kullanınız:

Kod 41: R'da döngüler

```
1 # for(i in x){ # i x dizisi içindeki değerleri sırasıyla alır
2 #   gerekli işlemler her i için yapılır
3 # }
```

Tüm vektörel işlemler (örneğin iki dizi çarpımı) for döngüsü ile yapılabilir. Fakat R'daki döngüler daha alt seviye bir programlama diline göre (örneğin C) çok yavaştır. Dolayısıyla bu tarz işlemleri mümkün olduğu kadarıyla vektörel olarak gerçekleştirmeyi öneririz.

Aşağıdaki örnekte π sayısının değerini Monte Carlo simülasyon yöntemi ile bulmaya çalışan bir fonksiyon yazacağız. Bunun için (-1,1) aralığındaki Uniform bir dağılımdan n tane rastgele sayı üreterek kenar uzunluğu 2 olan bir karenin alanını simüle etmeye çalışalım.

Kod 42: R'da döngü kullanmaya örnek

```
1 simPI <- function(n) {
2   y <- array(0,n)
3
4   # n uzunluğunda bir sıfırlarla dolu bir dizi yaratıyoruz
5   # Orijine Öklit (Euclidean) uzaklığın birden küçük ve eşit oldu
6   #   ğu noktalar daireyi temsil eder
7   nDaire <- 0 # daire içine düşen notka sayısını tutar
8   for(i in 1:n){ # i will take integer values from 1 to n
9     u1 <- runif(1,-1,1)
10    u2 <- runif(1,-1,1)
11    y[i] <- sqrt((u1-0)^2 + (u2-0)^2) # orijine uzaklığı
12    #   tutuyoruz
13    nDaire <- nDaire + (y[i]<1) # mantıksal operatörleri
14    #   uyguladığımızda 0 ve 1 değerleri döner
15  }
16
17  # simüle edilmiş karenin dairenin alanına oranı gerçek
18  #   karenin alanın gerçek dairenin alanına oranının
19  #   tahminiDaireAlan/tahminiKareAlan=daireAlan/KareAlan
20  #   dolayısıyla tahminiPi=(KareAlan*(tahminiDaireAlan/
21  #   tahminiKareAlan))/r^2
22  yaklasikPi=4*(nDaire/n)/1^2
23  names(yaklasikPi) <- c("tahmini")
24  return(yaklasikPi) # sonuç dönmek için return() fonksiyonu da
25  #   kullanılabilir
26 }
```

Kod 43: R'da döngü kullanan örneğe devam ediyoruz

```
1 simPI(1000)
2 # tahmini
```

```

3 # 3.196
4 simPI(10000)
5 # tahmini
6 # 3.1344
7 simPI(100000)
8 # tahmini
9 # 3.14988
10
11 # n yani simüle edilen nokta sayısı arttıkçe gerçek pi değerine
    yakınsamamız beklenir
12 # fakat rassal işlemler yaptığımız için bunu her zaman gö
    zlemleyemeyebiliriz
13 # bu durumlarda deneyimizi (kodumuzu) tekrarlamamız (replicate)
    ve tekrarlardan alınan
14 # sonuç ortalamalarını kullanmamız daha güvenilir sonuçlar
    verir
15
16 system.time(x <- simPI(100000)) # saniye cinsinden koşma süresi
17 # user system elapsed
18 # 2.07 0.00 2.09
19
20 # Aynı işlemi apply adlı bir fonksiyon ile yapan bir kod yazalı
    m
21 simPI_apply <- function(n){
22   #rassal sayıları tek seferde n x 2 lik bir matriste tutalım
23   rnd <- matrix(runif(2*n,-1,1),ncol=2) # 2*n kadar rastgele
        sayı üretip bunu sütun
24                                     # ("col"umn) sayısı 2
                                                olan bir matrise dağıt
                                                ır
25   # değerleri görselleştirmek isterseniz plot(rnd[,1],rnd[,2])
        komutunu kullanabilirsiniz.
26   # görselleştirme ile ilgili detaylar dökümanın ilerleyen kısm
        ında anlatılacaktır
27   y <- sqrt(apply(rnd^2,1,sum)) # apply fonksiyonu kullanarak
        satırlar ya da sütunlar
28                                     # üzerinden çeşitli fonksiyonlar
                                                çalıştırılabilir
29   # apply fonksiyonu içinde önce matris değerlerinin karesini
        alıp sonra satırlardaki
30   # değerlerin (1 argümanı satır olduğunu ifade eder) toplamını
        alıyoruz
31   nDaire <- sum(y<=1) # daire içine düşen nokta sayısını
        buluyoruz

```

```

32 # mantıksal operatörleri dizilere uyguladığımızda operatör
    her elemana uygulanır
33 # 0 ve 1 değerlerin toplamını aldığımızda daire içine düşen
    nokta sayısını buluruz
34 # daire içine düşen değerleri önceki grafik üzerinde görselle
    ştırmek isterseniz
35 # points(rnd[y<=1,1],rnd[y<=1,2],col=2) komutunu
    kullanabilirsiniz.
36 yaklasikPi=4*(nDaire/n)/1^2
37 names(yaklasikPi) <- c("tahmini")
38 return(yaklasikPi) # sonuç dönmek için return() fonksiyonu da
    kullanılabilir
39 }
40
41 simPI_apply(100000)
42 # tahmini
43 # 3.14808
44 system.time(x <- simPI_apply(100000)) # saniye cinsinden koşma
    süresi
45 # user system elapsed
46 # 0.66 0.00 0.65
47
48 # Aynı işlemi vektörel olarak yapan bir kod yazalım ve sonuç
49 # artı bir takım ekstra bilgileri liste halinde dönelim
50 simPI_vektor <- function(n){
51     #rassal sayıları tek seferde n x 2 lik bir matriste tutalım
52     rnd <- matrix(runif(2*n,-1,1),ncol=2) # 2*n kadar rastgele
        sayı üretip bunu sütun ("col"umn)
53                                     # sayısı 2 olan bir
                                                matrise dağıtır
54     y <- sqrt(rnd[,1]^2+rnd[,2]^2) # apply yerine toplamı vektö
        rel olarak yapalım (iki vektör toplamı)
55     nDaire <- sum(y<=1)
56     yaklasikPi=4*(nDaire/n)/1^2
57     return(list(tahminiPi=yaklasikPi,gercekPi=pi,DaireNoktaSayisi
        =nDaire,ToplamNoktaSayisi=n))
58     # sonucu bir liste halinde dönebilirsiniz
59 }
60
61 # liste halindeki çıktıya gözatalım
62 simPI_vektor(100000)
63 # $tahminiPi
64 # [1] 3.14228
65
66 # $gercekPi

```

```

67 # [1] 3.141593
68
69 # $DaireNoktaSayisi
70 # [1] 78557
71
72 # $ToplamNoktaSayisi
73 # [1] 1e+05
74
75 snc=simPI_vektor(100000)
76 # sadece tahmini pi değerine bakalım
77 snc$tahminiPi
78 # [1] 3.13708 #sonuç öncekinden farklı (rassallık sebebiyle)
79
80 system.time(x <- simPI_vektor(100000)) # saniye cinsinden koşma
    süresi
81 # user  system elapsed
82 # 0.05  0.00  0.055

```

Görüldüğü üzere, döngü kullanımı yerine vektörel olarak yapılan hesaplamalar daha az koşma zamanı gerektirmekte. Fakat algoritmalarınızı kodlarken for döngüleri tek opsiyonunuz olabilir. Daha önce de bahsedildiği üzere bu durumlarda döngüleri daha alt seviye bir programlama dili aracılığıyla yapmak çalışma süreleri açısından büyük faydalar sağlayacaktır.

While döngüleri yakınsama algoritmalarında sıkça kullanılır. Döngü sayısı belli olmayan durumlarda, while döngüsü kullanılır ve döngü aşağıdaki gibi tanımlanır:

Kod 44: While döngüsü

```

1 # while(koşul){ # koşul sağlandığı sürece döngüye devam et
2 #   gerekli işlemleri gerçekleştir
3 # }

```

While döngüsü kullanan bir kök bulma fonksiyonu

Kod 45: While döngüsü kullanan bir kök bulma fonksiyonu

```

1 # kök bulma fonksiyonu
2 # belirli bir aralıktaki sürekli bir fonksiyonun kökünü bulur
3 # sürekli fonksiyon x eksenini kesmelidir fakat dik
    kesmemelidir
4 kokbul <- function(
5   f,                # sıfır değeri için çözülecek sürekli fonksiyon
6   interval,        # tek çözümün aranacağı aralık (2 elemanlı bir
    dizi)
7   errbound=1e-12,  # izin verilen maksimum hata
8   trace=FALSE      # trace doğru yapılırsa, yakınsanan sayı
    dizileri ekrana yazılır
9 ) {

```



```

10 a <- interval[1]
11 b <- interval[2]
12 if(f(a)*f(b)>0){
13   print("hata - çözüm yok ya da birden fazla çözüm var")
14 }else{
15   counter <- 0
16   res <- 0
17   err <- abs(a-b)
18   while(err>errbound){
19     c <- (a+b)/2
20     fc <- f(c)
21     if(f(a)*fc>0){
22       a <- c
23     }else{
24       b <- c
25     }
26     err <- abs(a-b)
27     counter <- counter+1
28     res[counter] <- a
29   }
30   print(c(a,counter))
31   if(trace){
32     print(res)
33   }
34 }
35 }
36
37 func <- function(x){x^2-2}
38 int <- c(1,2)
39 kokbul(func,int)
40 # [1] 1.414214 40.000000
41 kokbul(func,int,trace=TRUE)
42 # [1] 1.414214 40.000000
43 # [1] 1.000000 1.250000 1.375000 1.375000 1.406250 1.406250
44 # [9] 1.414062 1.414062 1.414062 1.414062 1.414185 1.414185
45 # [17] 1.414207 1.414211 1.414213 1.414213 1.414213 1.414213
46 # [25] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
47 # [33] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214

```

4 R ile Grafik Çizme

Örneğin standart normal dağılımın olasılık dağılım fonksiyonunun değerini (-4,4) aralığındaki grafiğini çizelim. Bunun için öncelikle x eksenindeki değerleri temsil edecek uzun bir dizi yaratacağız (böylelikle fonksiyonun gerçek şekline iyi bir yakınsama elde ederiz.) ve fonksiyonu sonucunu ikinci bir veride tutup y-eksenindeki değerler olarak tanımlayacağız.

Kod 46: R ile grafik çizme

```
1 x <- seq(-4,4,length.out=51) # yeteri kadar uzun değil
2 y <- dnorm(x)
3 plot(x,y) # boş noktalar içeren bir grafik (şekil 1)
4
5 windows() # şekli yeni bir pencerede göstermek isterseniz bu
   komutu kullanabilirsiniz
6 plot(x,y,type="l") # noktaları birleştirir "l" line anlamına
   geliyor (şekil 2)
7
8 x <- seq(-4,4,length.out=10001) # yeteri kadar uzun (yoğun)
9 y <- dnorm(x)
10 windows()
11 plot(x,y,type="l") # daha çok noktayı birleştirir (şekil 3)
```

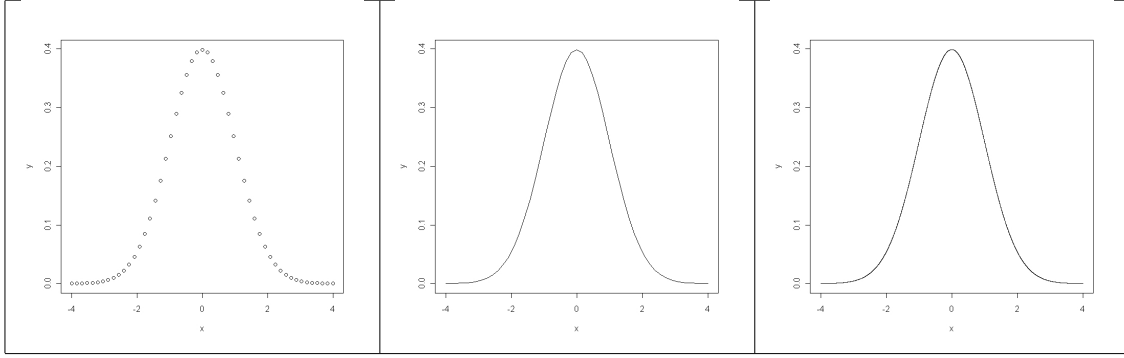
Tablo 1 içerisinde bu grafikleri görebiliriz. Şimdi bir dizi içindeki değerlerin histogramı `hist()` fonksiyonunu kullanarak çizelim. Verinizin dağılımı iyi bir şekilde görselleştirmek için histogramları kullanabiliriz. `hist()` fonksiyonunun `break` argümanını değiştirerek daha iyi (`break` değerine göre kötü) görünen histogramlar elde edilebilir.

Kod 47: R ile histogram çizme

```
1 x <- rnorm(1000000,3,1.5)
2 # ortalaması 3 and std. sapması 1.5 olan Normal dağılımdan
3 # yaratılmış bir 1000000 sayılıklı bir dizi
4
5 hist(x)
6
7 windows()
8 hist(x,breaks=50)
9
10 windows()
11 hist(x,breaks=100)
```

Tablo 2 çizilen histogramları göstermektedir. Bir grafiğe ve histograma çeşitli komutlar³ kullanarak yeni çizgiler veya noktalar ekleyebiliriz. Aşağıdaki örneklere bakalım.

³`points()` ve `abline()` gibi fonksiyonlar kullanarak grafiklere ek bilgiler koyulabilir. Detaylar için <http://stat.ethz.ch/R-manual/R-patched/library/graphics/html/points.html> bağlantısına bakabilirsiniz.



Şekil 1: Standart normal dağılımın yoğunluk grafikleri

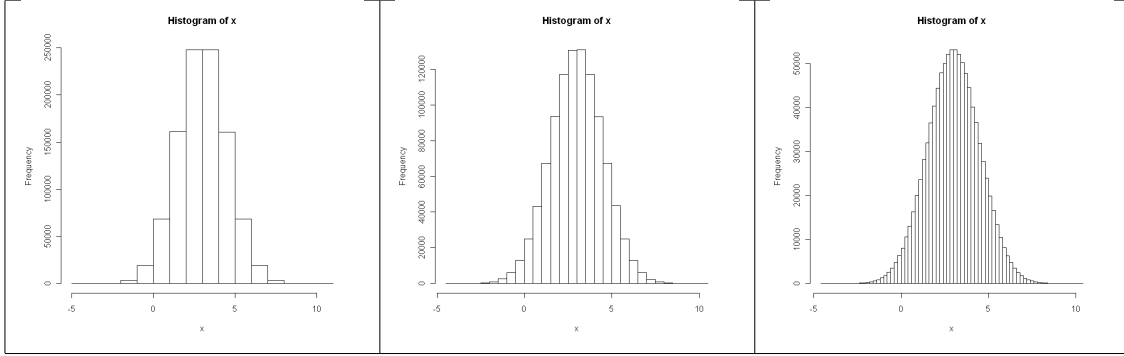
Kod 48: Grafiğe doğru ekleme

```

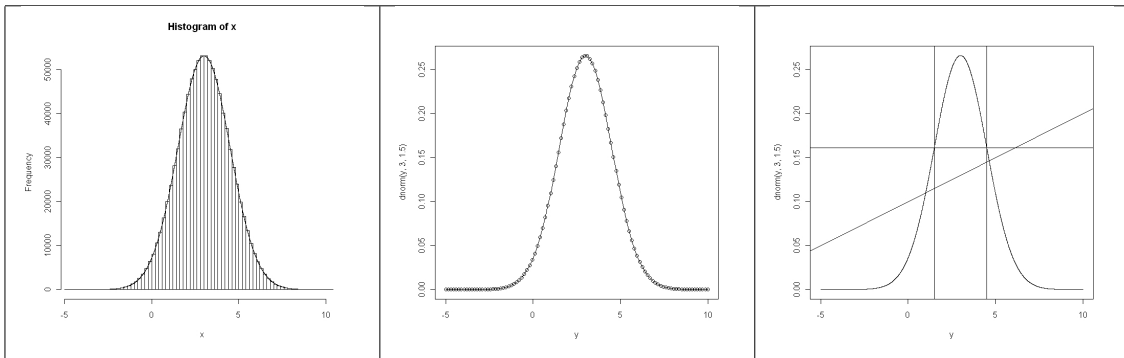
1 hist(x,breaks=100)
2 y <- seq(-5,10,length.out=100001)
3 lines(y,dnorm(y,3,1.5)*200000)
4
5 y <- seq(-5,10,length.out=101)
6 windows()
7 plot(y,dnorm(y,3,1.5))
8 lines(y,dnorm(y,3,1.5))
9
10 windows()
11 plot(y,dnorm(y,3,1.5),type="l")
12 abline(v=4.5) # x=4.5 noktasından geçen dikey ("v"ertical) bir
    doğru ekleyelim
13 abline(v=1.5) # x=1.5 noktasından geçen dikey ("v"ertical) bir
    doğru ekleyelim
14 abline(h=dnorm(1.5,3,1.5)) # y=dnorm(1.5,3,1.5) noktasından geç
    en
15                                     # yatay ("h"orizontal) bir doğru
    ekleyelim
16 abline(a=0.10,b=0.01) # 0.01 and x=0.10 noktasından geçen bir
    doğru ekleyelim

```

Tablo 3 bu çizilen grafikleri içermektedir.



Şekil 2: Ortalaması 3 and std. sapması 1.5 olan Normal dağılımdan yaratılmış bir 1000000 sayılık bir dizinin histogramları



Şekil 3: Varolan grafiklere `lines()` (1-2) and `abline()` (3) fonksiyonları ile doğrular ekleme

5 Temel Kullanıcı Bilgileri

5.1 Veri Okuma ve Yazma

Örneğin elimizde aşağıdaki biçimde metin dosyasına yazılmış bir veri⁴ olsun.

Kod 49: Veri

```
1 3 25 94.9 12
2 547 32556 56
3 89 567
4 435 342.1
5 76.5 983.2
6 0 343
7 # There are 15 real values
```

Bu veriyi okumak için `scan()` komutunu da kullanabiliriz. Bu durumda boşluklar ve yeni satıra geçmeler yeni bir değer girildiğini işaret eder.

Kod 50: `scan()` komutu

```
1 x <- scan()
2 # enter tuşuna bastıktan sonra, komut satırında "1:" görünür
3 # CTRL+V yazarak kopyalanmış veriyi yapıştırabiliriz, 15 sayı x
  isimli dizide tutulur
4 # sonrasında komut satırında "16:" görünür
5 # enter tuşuna basarak veri girişi sonlandırılır ve 15 sayı
  okunmuş olur
6
7 # 1: 3 25 94.9 12
8 # 5: 547 32556 56
9 # 8: 89 567
10 # 10: 435 342.1
11 # 12: 76.5 983.2
12 # 14: 0 343
13 # 16:
14 # Read 15 items
15
16 x
17 # [1] 3.0 25.0 94.9 12.0 547.0 32556.0 56.0 89.0
   567.0
18 # [10] 435.0 342.1 76.5 983.2 0.0 343.0
```

Excel dosyasında bulunan sütunları R'dan okuyabiliriz (maalesef satırları değil). Bunu yaparken tek dikkat edilmesi gereken konu R'da ondalık ayrımının (.) ile yapmasıdır. Dolayısıyla farklı bir biçim kullanıyorsanız (Excel'de ondalık ayrımı virgül ile belirleniyorsa), bu problem yaratabilir.

⁴Bu tür verinin hep aynı tipten verileri barındırması gerekmektedir.

Tablo değerlerini metin dosyasından da okuyabiliriz. Verinizi aşağıdaki gibi bir yapıda tutan bir metin dosyamız olduğunu varsayalım:

Kod 51: Metin dosyası halinde veri

```
1 boy    kilo yas
2 1.72   72.3   25
3 1.69   85.3   23
4 1.80   75.0   26
5 1.61   66     23
6 1.73   69     24
7 # her satırda 3 değer
```

Masaüstümüzdeki R kısayoluna sağ tıklayarak özelliklere girerek R'ın çalıştığı başlama klasörünü öğrenelim⁵. Oluşturduğumuz metin dosyasını bu klasöre `data.txt` ismi ile kopyalayalım ve aşağıdaki komutları yazalım.

Kod 52: `read.table` ile veri okuma

```
1 x <- read.table(file="data.txt",header=TRUE)
2 # eğer verinizde sütun başlıkları yoksa header=FALSE yapmak
  gerekir
3 x # x tablosunu görmek için <enter>a basın
4 #   boy    kilo yas
5 # 1   1.72   72.3  25
6 # 2   1.69   85.3  23
7 # 3   1.80   75.0  26
8 # 4   1.61   66.0  23
9 # 5   1.73   69.0  24
10 x$boy
11 # [1] 1.72 1.69 1.80 1.61 1.73
12 x$kilo
13 # [1] 72.3 85.3 75.0 66.0 69.0
14 x$yas
15 # [1] 25 23 26 23 24
```

Veri dosyasını kopyaladığımız klasöre R'ın çalışma klasörü (working directory) denir. Bu klasörün ne olduğunu R oturumunda `getwd()` komutunu çalıştırarak da öğrenebiliriz. Ayrıca `setwd()` kullanılarak da o an çalıştığımız R oturumu için çalışma klasörünü değiştirebiliriz. `setwd()` komutu argüman olarak klasörün tam yolunu (full path) ister. Burada önemli bir nokta bu komut Unix tipi yol belirtmenizi ister. Klasörler arası ayırım için `/` kullanılması önemlidir. Ayrıca ayırımın ile yapılması da mümkündür.

Kod 53: Çalışma klasörü ile ilgili ayarlamalar

```
1 calisma <- getwd() # şu anki çalışma klasörünü calisma değiş
  kenine eşitle
```

⁵Bunu R oturumunda da değiştirebilirsiniz

```

2 print(calisma) # ekrana bas
3 # "C:/Users/baydogan/Documents"
4 setwd("C:/") # çalışma klasörünü C klasörü yap^
5 getwd() # ekranda çalışma klasörünü bas
6 setwd("C:\\Programlar") # çalışma klasörünü C altında
   Programlar klasörü yap
7 getwd() # ekranda çalışma klasörünü bas

```

Eğer Excel tablolarından veri okumak istersek, tablo bilgisini bir metin dosyasına yapıştırıp, yukarıdaki gibi okuyabiliriz. Bunun dışında Excel ya da MINITAB gibi hesaplama tabloları (spreadsheet) içeren dosya tiplerinden veri okumak için özelleşmiş R paketleri mevcuttur. Paketler ile ilgili daha detaylı bilgi eğitim sırasında sağlanacaktır. `print()` fonksiyonu ekrana yorum, obje⁶ ya da bilgi yazmaya yarar. Eğer ekrana yorum yazacaksanız, komut içinde tek veya çift tırnak arasında yazmamız gerekir.

Kod 54: `print()` fonksiyonu

```

1 print("hata")
2 # [1] "hata"
3 x <- 1:5
4 print(x)
5 # [1] 1 2 3 4 5

```

5.2 Oturum Yönetimi

R ile birlikte sağlanan fonksiyonlar ile ilgili detaylı bilgilere R'ın yardımını kullanarak ulaşabiliriz. Fonksiyonun detaylı olarak ne yaptığına, parametrelerine (argümanlarına) ve donksiyon kullanımı ile ilgili çeşitli örnekler R ortamında sağlanmaktadır. Bir fonksiyon hakkında tüm bilgilere `?` ardından boşluk bırakmadan fonksiyon ismini yazarak ulaşabiliriz. Örneğin aşağıdaki fonksiyonlar için detaylı açıklamalara bakalım:

Kod 55: Fonksiyon işlemleri ile ilgili yardım alma

```

1 ?det
2 ?sample
3 ?sin
4 ?cbind

```

Ayrıca `apropos(". ")` fonksiyon adında kullanarak belli bir kelimeyi içeren fonksiyonları da listeleyebiliriz. Bunlar varsayılan paketlerden gelen fonksiyonlar olabilmekle birlikte sizin tanımladığınız fonksiyonlar da olabilir.

Kod 56: `apropos(". ")` fonksiyonu ile belirli bir kelimeyi içeren fonksiyon ve verileri sorgulama

```

1 apropos("norm")
2 # [1] "dlnorm"      "dnorm"      "normalizePath" "plnorm"

```

⁶Objeler vektör, matris, dizi, fonksiyon, liste (listeler C dilindeki yapılara benzer), tablo vs. olabilir.

```

3 # [5] "pnorm"          "qlnorm"          "qnorm"          "qqnorm"
4 # [9] "qqnorm.default" "rlnorm"          "rnorm"

```

Kod 57: `apropos (".")` fonksiyonu örneği

```

1 apropos ("exp")
2 # [1] ".__C__expression"      ".expand_R_libs_env_var" ".Export"
3 # [4] ".mergeExportMethods"  ".standard_regexps"     "as."
4 # [7] "as.expression.default" "char.expand"           "dexp"
5 # [10] "exp"                  "expand.grid"           "expand."
6 # [13] "expm1"                "expression"            "getExportedValue"
7 # [16] "getNamespaceExports"  "gregexpr"              "is."
8 # [19] "namespaceExport"      "path.expand"           "pexp"
9 # [22] "qexp"                 "regexpr"               "rexp"
10 # [25] "SSbiexp"              "USPersonalExpenditure"

```

Bir çalışma oturumundaki tüm objeleri görmek istediğinizde `objects ()` komutunu kullanabiliriz.

Kod 58: Çalışma oturumundaki objeleri görüntüleme

```

1 objects ()
2 # [1] "a"          "b"          "circle"     "coora"
3 # [5] "coorb"      "coorc"      "error"      "f"
4 # [9] "findroot"   "fixedcost"  "func"       "int"
5 # [13] "lbound"     "marginalcost" "n"          "orderingcostlist"
6 # [17] "res"        "simmax2unif" "simmax2unif_2" "totalcost"
7 # [21] "triangle"   "ubound"     "units"      "vec"
8 # [25] "x"          "xest"       "xinv"       "y"
9 # [29] "y1"         "y2"         "y3"         "y4"
10 # [33] "y5"         "y6"         "z"

```

R oturumunda yaptığımız tüm işlemleri ve objeleri hızlı erişim alanındaki *Dosya* altında *Çalışma alanı kaydet* seçeneğine tıklayarak kaydedebilirsiniz. Kaydedilmiş R oturum dosyalarımıza çift tıklayarak o zamanki R oturumuna dönebilirsiniz.