# Yeni Başlayanlar için R'a Hızlı Giriş

February 4, 2017

**Hazırlayanlar (soyadı sırasına göre)**

İsmail Başoğlu
Mustafa Gökçe Baydoğan
Uzay Çetin
Berk Orbay

**Kullanıldığı Eğitim**

Akademik Bilişim Konferansı Eğitimleri
4-7 Şubat 2017

## 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

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

```
x <- 3
```

veya

Kod 2: Değer atama yöntemleri 2

```
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

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

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

Kodlarınızın 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.

Kod 4: Ardışık tamsayı vektörü oluşturma

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

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

Kod 5: Vektör elemanlarına değer ekleme

```
y <- x+3
```

[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.

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

Önceki komutun aslında yaptığı şey 8 eleman uzunluğunda bir vektörü bir eleman uzunluğunda bir vektörle toplamaktı. 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

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

```
x <- 1:8
y <- 1:3
x+y
# [1]  2  4  6  5  7  9  8 10
# Uyarı mesajları:
# In x + y : uzun olan nesne uzunluğu kısa olan nesne uzunluğunun 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

```
x <- c(4,8,15,16,23,42) # c(.) fonksiyonuyla değerleri birleştriyoruz
y <- c(501,505,578,586)
x
# [1]  4  8 15 16 23 42
y
# [1] 501 505 578 586
z <- c(x,y) # ayrıca iki vektörü de c(.) ile birleştirerek yeni bir vektör oluş
    turabiliriz
z
```

```
# [1]    4    8   15   16   23   42  501  505  578  586

length(z) # oluşturulan z vektörünün uzunluğu
# [1] 10
m<- c(1,5,1,4,7,4,1)
unique(m) # oluşturulan m vektöründeki tekil değerler
# [1] 1 5 4 7
table(m) # oluşturulan m vektöründeki tekil değerlerin kaç tane olduğu
# m
# 1 4 5 7
# 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

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

Diyelim ki bütün elemanlarının değeri 5 olan 10 eleman uzunluğunda 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

```
x <- rep(5,10) # 5 değerini 10 kez tekrar et
x
# [1] 5 5 5 5 5 5 5 5 5 5
y <- c(3,5,7) # y vektörü
z <- rep(y,4) # y vektörünü 4 kez tekrar et
z
# [1] 3 5 7 3 5 7 3 5 7 3 5 7
rep(y,c(2,3,5)) # y nin elemanlarını teker teker
              # belirlenen değerlerde tekrar et
# [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 değerlerden oluşan 21 eleman uzunluğunda bir vektör yaratacağız. Kullanacağımız fonksiyon `seq(.)` ("seq"uence, seri) olacak.

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

```
x <- seq(2,3,length.out=21) # length.out vektorun uzunlugunu belirleyen parametre adidir

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

Eğer vektör uzunluğu 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

```
x <- seq(2,3,by=0.05)
x
#  [1] 2.00 2.05 2.10 2.15 2.20 2.25 2.30 2.35 2.40 2.45 2.50
```

```
# [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 $90, 91, \ldots, 100$ sayıları gözlemlenir. Bu sayıları bulmak için `which(.)` fonksiyonu ve `%in%` operatörünü kullanabiliriz.

Kod 13: Mantıksal ifadeler 1

```
x <- 10:20
x
# [1] 10 11 12 13 14 15 16 17 18 19 20
x<17 # x'in hangi değerleri 17den küçüktür
# [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
x<=17 # x'in hangi değerleri 17den küçük eşittir
# [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
x>14 # x'in hangi değerleri 14ten büyüktür
# [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
x>=14 # x'in hangi değerleri 14ten büyük eşittir
# [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
x==16 # x'in hangi değerleri 16ya eşittir
# [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
x!=16 # x'in hangi değerleri 16ya eşit değildir
# [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE

(x<=16) & (x>=12) # x'in hangi değerleri 16dan küçük eşit ve 12den büyük eşittir
# [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
(x<=11) | (x>=18) # x'in hangi değerleri 11den küçük eşit veya 18den büyük eşittir
# [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE

```

```
ind=which(x<17)  # x'in 17den küçük değerlerinin indeksleri nedir
ind
# [1] 1 2 3 4 5 6 7
ind2=which(x==16) #x'in hangi indeksteki değeri 16ya eşittir
ind2
# [1] 7
v1=90:120   #v1'in hangi elemanları v2 içinde geçer
v2=10:100
ind3=which(v1 %in% v2)
ind3
# [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 başka işlemler yapılabilir? Basit bir ilk örnek olarak 1den 20ye kadar değerler içeren bir `x` vektörü oluşturup 8den küçük her elemanını 0 haline getirebiliriz. Böylece 8den küçük elemanlar 0 değeri alırken diğer elemanlar değerlerini koruyacaklardır.

Kod 14: Mantıksal ifadeler 2

```
x <- 1:20
y <- (x>=8)*(x)
y
# [1]  0  0  0  0  0  0  0  8  9 10 11 12 13 14 15 16 17
#[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

```
siparisMiktari <- 30:50
birimMaliyet <- 7*siparisMiktari*(siparisMiktari<40)+6.5*siparisMiktari*(units
    >=40)
birimMaliyet
# [1] 210.0 217.0 224.0 231.0 238.0 245.0 252.0 259.0
# [9] 266.0 273.0 260.0 266.5 273.0 279.5 286.0 292.5
#[17] 299.0 305.5 312.0 318.5 325.0

sabitMaliyet <- 50*(siparisMiktari<=45)+15*(siparisMiktari>45)
sabitMaliyet
# [1] 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 15
#[18] 15 15 15 15

toplamMaliyet <- sabitMaliyet + birimMaliyet
toplamMaliyet
# [1] 260.0 267.0 274.0 281.0 288.0 295.0 302.0 309.0
# [9] 316.0 323.0 310.0 316.5 323.0 329.5 336.0 342.5
#[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

```
siparisMiktari[toplamMaliyet<=318]
#318TL'den daha dusuk maliyete sahip olan siparis miktarlarini gosterir

# [1] 30 31 32 33 34 35 36 37 38 40 41 46
toplamMaliyet[toplamMaliyet<=318]
#318TL'den daha dusuk maliyete sahip olan siparis maliyetlerini gosterir
# [1] 260.0 267.0 274.0 281.0 288.0 295.0 302.0 309.0
# [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. Diğeri 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

```
x <- seq(5,8,by=0.3) # 11 elemanlı bir vektör oluşturuyoruz
x
# [1] 5.0 5.3 5.6 5.9 6.2 6.5 6.8 7.1 7.4 7.7 8.0
length(x)
# [1] 11

y1 <- x[3:7] # 3üncü elemanından 7inci elemana kadar olan kısmını alıyoruz
y1
# [1] 5.6 5.9 6.2 6.5 6.8

y2 <- x[2*(1:5)] # çift sayı sırasındaki elemanları alıyoruz 2inci, 4üncü gibi
y2
# [1] 5.3 5.9 6.5 7.1 7.7

y3 <- x[-1] # ilk elemanı çıkarıp geri kalanı alıyoruz
y3
# [1] 5.3 5.6 5.9 6.2 6.5 6.8 7.1 7.4 7.7 8.0

y4 <- x[-length(x)] # son elemanı çıkarıp geri kalanı alıyoruz
y4
# [1] 5.0 5.3 5.6 5.9 6.2 6.5 6.8 7.1 7.4 7.7

y5 <- x[-seq(1,11,3)] # belirtilen elemanları çıkarıp geri kalanı alıyoruz
y5
# [1] 5.3 5.6 6.2 6.5 7.1 7.4 8.0

y6 <- x[c(1,3,7)] # sadece birinci, üçüncü ve yedinci elemanları alıyoruz
y6
# [1] 5.0 5.6 6.8

y7 <- x[seq(1,11,3)] # sadece belirtilen elemanları alıyoruz
y7
# [1] 5.0 5.9 6.8 7.7
```

## 1.3 Matrisler

Bölümler 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.

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

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

```
t(y) #veya sadece t(t(x)) de yazabilirdik
#     [,1]
# [1,]   1
# [2,]   2
# [3,]   3
# [4,]   4
# [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).

```
vec <- 1:12
x <- matrix(vec,nrow=3,ncol=4)

x #rakamların sırasına dikkat
#     [,1] [,2] [,3] [,4]
# [1,]   1   4   7   10
# [2,]   2   5   8   11
# [3,]   3   6   9   12

t(x) #döndürülmüşü
#     [,1] [,2] [,3]
# [1,]   1   2   3
# [2,]   4   5   6
# [3,]   7   8   9
# [4,]   10  11  12
```

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

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

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

```
x <- matrix(c(1,2,-1,1,2,1,2,-2,-1),nrow=3,ncol=3)
x
  [,1] [,2] [,3]
[1,]  1  1  2
[2,]  2  2   -2
[3,]   -1  1   -1

xinv <- solve(x)
xinv
#          [,1]        [,2] [,3]
# [1,] 0.0000000  0.25000000 -0.5
# [2,] 0.3333333  0.08333333  0.5
# [3,] 0.3333333 -0.16666667  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.

```
x <- matrix(0,nrow=4,ncol=4)
x
#    [,1] [,2] [,3] [,4]
# [1,]  0  0  0  0
# [2,]  0  0  0  0
# [3,]  0  0  0  0
# [4,]  0  0  0  0

diag(x) <- 1 # matris köşegenin bütün değerlerini 1 yapar
x
#    [,1] [,2] [,3] [,4]
# [1,]  1  0  0  0
# [2,]  0  1  0  0
# [3,]  0  0  1  0
# [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.

```
x <- matrix(0,ncol=5,nrow=4)
ncol(x)
# [1] 5
nrow(x)
# [1] 4
length(x)
# [1] 20
dim(x)
# [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 aritmetik olarak bahsedilen kalan işlemi %% sembolüyle yapılmaktadır) işlemlerini yapabileceğimizden bahsetmiştik.

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

```r
y <- 3:7
y
# [1] 3 4 5 6 7

x%%y # iki vektörde de her eleman için kalan işlemi
# [1] 2 0 1 2 3
x%/%y # bölme işleminin tam sayı kısmı
# [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.

```r

x <- c(3,1,6,5,8,10,9,12,3)

min(x)
# [1] 1

max(x)
# [1] 12

sum(x)
```

```
# [1] 57

prod(x)
# [1] 2332800
```

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

```
x <- 1:10
y <- 10:1
z <- c(3,2,1,6,5,4,10,9,8,7)

a <- pmax(x,y,z) # istediğimiz sayıda vektör yazabiliriz
a
# [1] 10  9  8  7  6  6 10  9  9 10

b <- pmin(x,y,z)
b
# [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.

```

 veri<-c(5,32,6,11,43,11,4,3,2,8)

#Sort bize güzel bir sıralı liste verir
sort(veri)
# [1]  2  3  4  5  6  8 11 11 32 43

#Order bize sıralamanın hücre numaralarını (indeks) verir
order(veri)
# [1]  9  8  7  1  3 10  4  6  2  5

#Yani en küçük rakam 9uncu hücre olan 2, diğeri 8. hücre olan 3 ve devam eder

#Veriyi sıraya sokmak için yapılacak iş basit.

 veri[order(veri)]
# [1]  2  3  4  5  6  8 11 11 32 43

#Büyükten küçüğe sıralamak için decreasing parametresini kullanıyoruz
order(veri, decreasing=TRUE)
#[1]  5  2  4  6 10  3  1  7  8  9

#Rank komutu ise o hücrenin kaçıncı sırada olduğunu söyler.
rank(veri)
#[1]  4.0  9.0  5.0  7.5 10.0  7.5  3.0  2.0  1.0  6.0

#Farkettiyseniz iki tane 7.5 var. Bunun sebebi iki tane 11 olması.
#7. ve 8. sıradaki yerleri işgal ettiğinden otomatik olarak
#ortalama değeri alıyorlar. Bunu değiştirmenin yolu ise söyle

```

```
#First metodu hücre sırası önce gelen değeri daha üst sırada gösterir. ilk 11
#7inci sırada ikinci 11 8inci sırada.
rank(veri,ties.method="first")
#[1]  4  9  5  7 10  8  3  2  1  6

#Random bu işi rastgele yapar.
rank(veri,ties.method="random")
# [1]  4  9  5  7 10  8  3  2  1  6
rank(veri,ties.method="random")
# [1]  4  9  5  8 10  7  3  2  1  6

#Max en yüksek sıralamayı verir.
rank(veri,ties.method="max")
# [1]  4  9  5  8 10  8  3  2  1  6

#Min en dusuk siralamayı verir.
rank(veri,ties.method="min")
# [1]  4  9  5  7 10  7  3  2  1  6

#Average ise ilk baştaki 7.5lu vektörün aynısını verir.
rank(veri,ties.method="average")
# [1]  4.0  9.0  5.0  7.5 10.0  7.5  3.0  2.0  1.0  6.0

#rankte orderda olan büyükten küçüğe sıralama parametresi yoktur.
#Onu da ufak bir numara ile yapabiliyoruz :)
rank(-veri)
# [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 `%*%` kullanmalısınız. Matris çarpım kuralları burada da işlemektedir ama matris boyutları uyumlu olmadığında R bazen otomatik düzeltmeler yapıp bir sonuç verebilir.

```
x <- matrix(1:6,ncol=2,nrow=3)
x
#    [,1] [,2]
# [1,]   1  4
# [2,]   2  5
# [3,]   3  6

y <- matrix(1:4,ncol=2,nrow=2)
y
#    [,1] [,2]
# [1,]   1  3
# [2,]   2  4

x%*%y
#    [,1] [,2]
# [1,]   9   19
# [2,]    12   26
# [3,]    15   33

y%*%x
# Hata oluştu y %*% x : uygun olmayan argümanlar

y%*%t(x) # x'i döndürmek sonuç verecektir
   [,1] [,2] [,3]
[1,]    13   17   21
```

```
[2,]   18   24   30
```

Şimdi iki dikey vektörün matris çarpımı ile ilgili bir örnek düşünelim. R birinci vektörü yatay bir vektör olarak düzeltip sonucu tek bir değer olarak döndürür. Eğer iki yatay vektör için matris çarpımı yapacak olsaydık R hata verecekti. Eğer vektörlerden ilkini dikey ($m \times 1$) diğerini yatay ($1 \times n$) olarak alacak olursak $m \times n$ bir matris elde etmiş oluruz.

```
x <- 1:3
y <- 3:1

x%*%y # R bir düzeltme yaparak ilk vektörü yatay hale getiriyor
#     [,1]
# [1,]   10

t(x)%*%y # yukarıdaki örnek ile aynı sonucu veriyor
#     [,1]
# [1,]   10


t(x)%*%t(y)
# Hata oluştu t(x) %*% t(y) : uygun olmayan argümanlar

x%*%t(y) # bu alternatif de bir mxn matrisi oluşturur
#     [,1] [,2] [,3]
# [1,]   3  2  1
# [2,]   6  4  2
# [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.

```
x <- c(1,4,5,6,2,12)
y <- cumsum(x)
y
# [1]  1  5 10 16 18 30
# cumsum ile her hücre kendisinden önceki hücrelerin toplamını kendi değerine
    ekler

z <- cumprod(x)
z
# [1] 1 4   20   120   240  2880
# cumprod ile her hücre kendisinden önceki hücrelerin çarpımını kendi değeriyle ç
    arpar

diff(x)
# [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()` tams ayıya 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.

```
factorial(3)
# [1] 6
factorial(1:6)
# [1]   1   2   6  24 120 720

abs(-4)
# [1] 4
abs(c(-3:3))
# [1] 3 2 1 0 1 2 3

sqrt(4)
# [1] 2
sqrt(1:9)
# [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
# [9] 3.000000

log(100) # doğal logaritma
# [1] 4.60517
log10(100) # tabanı 10 olan logaritma
# [1] 2
log2(100) # tabanı 2 olan logaritma
# [1] 6.643856
log(100,5) # tabanı 5 olan logaritma
# [1] 2.861353
log(c(10,20,30,40))
# [1] 2.302585 2.995732 3.401197 3.688879

exp(4.60517) # yaklaşık olarak 100 değerini vermeli
# [1] 99.99998
exp(log(100)) # yuvarlama hataları olmadan
# [1] 100
exp(seq(-2,2,0.4))
# [1] 0.1353353 0.2018965 0.3011942 0.4493290 0.6703200 1.0000000 1.4918247
# [8] 2.2255409 3.3201169 4.9530324 7.3890561

gamma(5) # factorial(4) ile aynı olmalı
# [1] 24
gamma(5.5) # factorial(4.5) ile aynı olmalı
# [1] 52.34278

x <- c(-3,-3.5,4,4.2)
```

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

```
floor(x)
# [1] -3 -4  4  4
ceiling(x)
# [1] -3 -3  4  5
as.integer(x)
# [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 örneklere bakacak olursak:

```r
dnorm(0.5) # ortalama ve standart sapma parametreleri tanımlanmazsa
        # R standart normal dağılım kullanır
# [1] 0.3520653
dnorm(0,2,1)
# [1] 0.05399097
dnorm(3,3,5)
# [1] 0.07978846

pnorm(0) # eğrinin altında kalan alan
      # standart normal dağılımda "0" solundaki kalan alan
# [1] 0.5
pnorm(2)
# [1] 0.9772499
pnorm(5,3,1)
# [1] 0.9772499

# Önceki "pnorm()" fonksiyonlarının tersi işlemi yapma (quantile)
qnorm(0.5)
# [1] 0
qnorm(0.9772499)
# [1] 2.000001
qnorm(0.9772499,3,1)
# [1] 5.000001

rnorm(20,2,1) # ortalaması 2 standart sapması 1 olan
            # normal dağılımdan 20 tane rastgele sayı
#  [1]  2.31502453  0.37445729  2.04994863  1.89381118  0.63099383  1.50837615
#  [7]  0.57363369  2.84601422  2.54003868  3.43652548  0.88941281  3.36373629
# [13]  0.58945290  2.44678124 -0.05360271  2.73920472  2.73643684  1.79465998
# [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`z` 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

```
x <- rnorm(1000000,5,2) # x ortalaması 5, standart sapması 2 olan normal dağılımdan
                        # gelen 1000000 rastgele sayıdan oluşan bir dizidir.

mean(x)
# [1] 4.997776
sd(x)
# [1] 2.000817
```

```
var(x)
# [1] 4.003268
median(x)
# [1] 4.997408
summary(x)
#    Min.  1st Qu.   Median    Mean  3rd Qu.    Max.
# -4.904    3.650    4.997    4.998    6.346   14.420
summary(x, digits=6)
#    Min.   1st Qu.    Median    Mean   3rd Qu.     Max.
# -4.90360   3.65020   4.99741   4.99778   6.34564  14.42310
quantile(x) # bu fonksiyon çeyreklik dilimler hakkında da bilgi verir
#        0%       25%        50%        75%     100%
# -4.903599   3.650201   4.997408   6.345639  14.423129

# çeyreklik dilimleri aşağıdaki gibi de elde edebiliriz.
sort(x)[1000000*0.25]
# [1] 3.650189
sort(x)[1000000*0.5]
# [1] 4.997408
sort(x)[1000000*0.75]
# [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.

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

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:

```
# ÖRNEK 01
# Yarıçapı r olan bir çemberin çevresini ve dairenin alanını hesaplayan bir fonksiyon
cevre_alan <- function(r) # yarıçap
{
  cf <- 2*pi*r # çevreyi hesaplar, pi R'da tanımlı bir sabittir.
  a <- pi*r^2  # alanı hesaplar
  res <- c(cf,a) # sonuçları birleştirir
  names(res) <- c("çevre","alan")
  res
}

circle(3)
#       çevre     alan
#   18.84956   28.27433
circle(1)
#       çevre     alan
#   6.283185   3.141593
```

```
# ÖRNEK 02
# Köşe koordinatları iki boyutlu uzayda belirlenmiş üçgenin
# çevresini ve alanını hesaplayan bir fonksiyon
ucgen <- function(
  a, # birinci köşenin koordinatı (2 uzunluğunda bir dizi olmalı (x,y))
  b, # ikinci köşenin koordinatı  (2 uzunluğunda bir dizi olmalı (x,y))
  c  # üçüncü köşenin koordinatı (2 uzunluğunda bir dizi olmalı (x,y))
){
  if(length(a)!=2 || length(b)!=2 || length(c)!=2){
    print("hata, en az bir koordinat hatalı ya da eksik girilmiş")
  }
# çevre hesaplanır
  ab <- sqrt((a[1]-b[1])^2+(a[2]-b[2])^2)
  bc <- sqrt((c[1]-b[1])^2+(c[2]-b[2])^2)
  ac <- sqrt((a[1]-c[1])^2+(a[2]-c[2])^2)
```

```
  pm <- ab+bc+ac
# alan hesaplanır
  trab <- abs((a[1]-b[1])*(a[2]-b[2]))/2
  trbc <- abs((c[1]-b[1])*(c[2]-b[2]))/2
  trac <- abs((a[1]-c[1])*(a[2]-c[2]))/2

  maxxy <- pmax(a,b,c)
  minxy <- pmin(a,b,c)

  sqa <- min(max((a[1]-minxy[1])*(a[2]-minxy[2]),0),max((maxxy[1]-a[1])*(maxxy[2]-
      a[2]),0))
  sqb <- min(max((b[1]-minxy[1])*(b[2]-minxy[2]),0),max((maxxy[1]-b[1])*(maxxy[2]-
      b[2]),0))
  sqc <- min(max((c[1]-minxy[1])*(c[2]-minxy[2]),0),max((maxxy[1]-c[1])*(maxxy[2]-
      c[2]),0))
  area <- (maxxy[1]-minxy[1])*(maxxy[2]-minxy[2])-trab-trbc-trac-sqa-sqb-sqc

  pm <- (area!=0)*pm # if area=0, then there is no triangle

  res <- c(pm,area)
  names(res) <- c("çevre","alan")
  res
}

coora <- c(23,18)
coorb <- c(13,34)
coorc <- c(50,5)
ucgen(coora,coorb,coorc)
#  çevre     alan
#  95.84525 151.00000

coora <- c(10,18)
coorb <- c(13,34)
coorc <- c(50,5)
ucgen(coora,coorb,coorc)
#  çevre     alan
#  105.3489  339.5000
```

```
coora <- c(3,5)
coorb <- c(9,15)
coorc <- c(6,10)
ucgen(coora,coorb,coorc)
#  çevre     alan
#    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.

```
# ÖRNEK 03
siparisMaliyetListesi <- function(
  huc=7,    # yüksek birim maliyet
  luc=6.5,  # düşük birim maliyet
  ucc=40,   # düşük birim maliyetten en az sipariş miktarı
  hfc=50,   # yüksek sabit maliyet
```

```
  lfc=15,  # düşük sabit maliyet
  fcc=45,  # yüksek birim maliyetten en çok sipariş miktarı
  tcub=318 # toplam maliyet için en üst (sınırlayıcı) seviye
){
  units <- 30:50
  birimMaliyet <- huc*units*(units<ucc)+luc*units*(units>=ucc)
  sabitMaliyet <- hfc*(units<=fcc)+lfc*(units>fcc)
  toplamMaliyet <- sabitMaliyet+birimMaliyet
  res <- toplamMaliyet[toplamMaliyet<=tcub]
  names(res) <- units[toplamMaliyet<=tcub]
  res
}

siparisMaliyetListesi() # önceki değerler ile aynı sonucu verir
# 30  31  32  33  34  35  36  37  38  40  41  46
# 260.0 267.0 274.0 281.0 288.0 295.0 302.0 309.0 316.0 310.0 316.5 314.0

siparisMaliyetListesi(hfc=55,luc=6.3) # iki argümanın değerini değiştirelim
# 30  31  32  33  34  35  36  37  40  41  46  47  48
# 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.

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

```
# ÖRNEK 04
f <- function(x){
  if(x<(-2)){
  x^2
  }else if(x<0){
  x+6
  }else if(x<4){
  -x+6
  }
  else{
  sqrt(x)
  }
}

c(f(-4),f(-1),f(3),f(9))
# [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 kullanırız:

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

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 $\pi$ 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.

```
simPI <- function(n){
  y <- array(0,n)

# n uzunluğunda bir sıfırlarla dolu bir dizi yaratıyoruz
# Orijine Öklit (Euclidean) uzaklığın birden küçük ve eşit olduğu noktalar daireyi temsil eder
  nDaire <- 0 # daire içine düşen notka sayısını tutar
  for(i in 1:n){ # i will take integer values from 1 to n
    u1 <- runif(1,-1,1)
    u2 <- runif(1,-1,1)
    y[i] <- sqrt((u1-0)^2 + (u2-0)^2) # orijine uzaklığı tutuyoruz
    nDaire <- nDaire + (y[i]<1)  # mantıksal operatörleri uyguladığımızda 0 ve 1 değerleri döner
  }

  # simüle edilmiş karenin dairenin alanına oranı gerçek karenin alanın gerçek dairenin alanına oranın
  # kareAlan=2*2, daireAlan=pi*(r^2) r=1  dolayısıyla tahminiDaireAlan/tahminiKareAlan=daireAlan/KareAlan
  # dolayısıyla tahminiPi=(KareAlan*(tahminiDaireAlan/tahminiKareAlan))/r^2
  yaklasikPi=4*(nDaire/n)/1^2
  names(yaklasikPi) <- c("tahmini")
  return(yaklasikPi) # sonuç dönmek için return() fonksiyonu da kullanılabilir
}
```

```
simPI(1000)
#  tahmini
#  3.196
simPI(10000)
# tahmini
#  3.1344
simPI(100000)
# tahmini
# 3.14988

# n yani simüle edilen nokta sayısı arttıkça gerçek pi değerine yakınsamamız beklenir
# fakat rassal işlemler yaptığımız için bunu her zaman gözlemleyemeyebiliriz
# bu durumlarda deneyimizi (kodumuzu) tekrarlamamız (replicate) ve tekrarlardan alınan
# sonuç ortalamalarını kullanmamız daha güvenilir sonuçlar verir

system.time(x <- simPI(100000)) # saniye cinsinden koşma süresi
#  user  system elapsed
#  2.07  0.00   2.09

# Aynı işlemi apply adlı bir fonksiyon ile yapan bir kod yazalım
simPI_apply <- function(n){
  #rassal sayıları tek seferde n x 2 lik bir matriste tutalım
```

```r
  rnd <- matrix(runif(2*n,-1,1),ncol=2) # 2*n kadar rastgele sayı üretip bunu sütun
                                        # ("col"umn) sayısı 2 olan bir matrise dağıtır
  # değerleri görselleştirmek isterseniz plot(rnd[,1],rnd[,2]) komutunu kullanabilirsiniz.
  # görselleştirme ile ilgili detaylar dökümanın ilerleyen kısmında anlatılacaktır
  y <- sqrt(apply(rnd^2,1,sum)) # apply fonksiyonu kullanarak satırlar ya da sütünlar
                                # üzerinden çeşitli fonksiyonlar çalıştırılabilir
  # apply fonksiyonu içinde önce matris değerlerinin karesini alıp sonra satırlardaki
  # değerlerin (1 argümanı satır olduğunu ifade eder) toplamını alıyoruz
  nDaire <- sum(y<=1) # daire içine düşen nokta sayısını buluyoruz
  # mantıksal operatörleri dizilere uyguladığımızda operatör her elemana uygulanır
  # 0 ve 1 değerlerin toplamını aldığımızda daire içine düşen nokta sayısını buluruz
  # daire içine düşen değerleri önceki graifk üzerinde görselleştirmek isterseniz
  # points(rnd[y<=1,1],rnd[y<=1,2],col=2) komutunu kullanabilirsiniz.
  yaklasikPi=4*(nDaire/n)/1^2
  names(yaklasikPi) <- c("tahmini")
  return(yaklasikPi) # sonuç dönmek için return() fonksiyonu da kullanılabilir
}

simPI_apply(100000)
# tahmini
# 3.14808
system.time(x <- simPI_apply(100000)) # saniye cinsinden koşma süresi
#  user  system elapsed
#  0.66  0.00   0.65

# Aynı işlemi vektörel olarak yapan bir kod yazalım ve sonuç
# artı bir takım ekstra bilgileri liste halinde dönelim
simPI_vektor <- function(n){
  #rassal sayıları tek seferde n x 2 lik bir matriste tutalım
  rnd <- matrix(runif(2*n,-1,1),ncol=2) # 2*n kadar rastgele sayı üretip bunu sütun ("col"umn)
                                        # sayısı 2 olan bir matrise dağıtır
  y <- sqrt(rnd[,1]^2+rnd[,2]^2) # apply yerine toplamı vektörel olarak yapalım (iki vektör toplamı)
  nDaire <- sum(y<=1)
  yaklasikPi=4*(nDaire/n)/1^2
  return(list(tahminiPi=yaklasikPi,gercekPi=pi,DaireNoktaSayisi=nDaire, ToplamNoktaSayisi=n))
  # sonucu bir liste halinde dönebilirsiniz
}

# liste halindeki çıktıya gözatalım
simPI_vektor(100000)
# $tahminiPi
# [1] 3.14228

# $gercekPi
# [1] 3.141593

# $DaireNoktaSayisi
# [1] 78557

# $ToplamNoktaSayisi
```

```
# [1] 1e+05

snc=simPI_vektor(100000)
# sadece tahmini pi değerine bakalım
snc$tahminiPi
# [1] 3.13708 #sonuç öncekinden farklı (rassallık sebebiyle)

system.time(x <- simPI_vektor(100000)) # saniye cinsinden koşma süresi
# user  system elapsed
# 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 kullanır. Döngü sayısı belli olmayan durumlarda, while döngüsü kullanır ve döngü aşağıdaki gibi tanımlanır:

```
# while(koşul){ # koşul sağlandığı sürece döngüye devam et
#   gerekli işlemleri gerçekleştir
# }
```

While döngüsü kullanan bir kök bulma fonksiyonu

```
# kök bulma fonksiyonu
# belirli bir aralıktaki sürekli bir fonksiyonun kökünü bulur
# sürekli fonksiyon x eksenini kesmelidir fakat dik kesmemelidir
kokbul <- function(
  f,             # sıfır değeri için çözülecek sürekli fonksiyon
  interval,      # tek çözümün aranacağı aralık (2 elemanlı bir dizi)
  errbound=1e-12, # izin verilen maksimum hata
  trace=FALSE    # trace doğru yapılırsa, yakınsanan sayı dizileri ekrana yazılır
){
  a <- interval[1]
  b <- interval[2]
  if(f(a)*f(b)>0){
  print("hata - çözüm yok ya da birden fazla çözüm var")
  }else{
  counter <- 0
  res <- 0
  err <- abs(a-b)
  while(err>errbound){
    c <- (a+b)/2
    fc <- f(c)
    if(f(a)*fc>0){
      a <- c
    }else{
      b <- c
    }
    err <- abs(a-b)
    counter <- counter+1
    res[counter] <- a
  }
  print(c(a,counter))
  if(trace){
    print(res)
```

```
  }
  }
}

func <- function(x){x^2-2}
int <- c(1,2)
kokbul(func,int)
# [1]  1.414214 40.000000
kokbul(func,int,trace=TRUE)
# [1]  1.414214 40.000000
#  [1] 1.000000 1.250000 1.375000 1.375000 1.406250 1.406250 1.414062 1.414062
#  [9] 1.414062 1.414062 1.414062 1.414062 1.414185 1.414185 1.414185 1.414200
# [17] 1.414207 1.414211 1.414213 1.414213 1.414213 1.414213 1.414214 1.414214
# [25] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
# [33] 1.414214 1.414214 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.

```
x <- seq(-4,4,length.out=51) # yeteri kadar uzun değil
y <- dnorm(x)
plot(x,y) # boş noktalar içeren bir grafik (şekil 1)

windows() # şekli yeni bir pencerede göstermek isterseniz bu komutu kullanabilirsiniz
plot(x,y,type="l") # noktaları birleştirir "l" line anlamına geliyor (şekil 2)

x <- seq(-4,4,length.out=10001) # yeteri kadar uzun (yoğun)
y <- dnorm(x)
windows()
plot(x,y,type="l") # daha çok noktayı birleştirir (şekil 3)
```

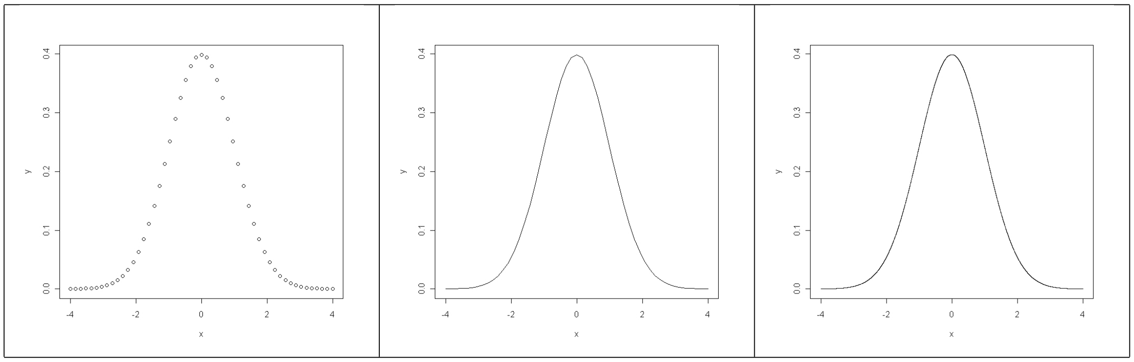
Tablo 1 içerisinde bu grafikleri görebiliriz. Şimdi bir dizi içindeki değerlerin histogramı `hist()` fonksiyonunu kullanarak çizelim. Verinizin dağılımı iyi bir şekilde görselleştirmek için histogramları kullanabiliriz. `hist()` fonksiyonunun `break` argümanını değiştirerek daha iyi (`break` değerine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

```
x <- rnorm(1000000,3,1.5)
# ortalaması 3 and std. sapması 1.5 olan Normal dağılımdan
# yaratılmış bir 1000000 sayılık bir dizi

hist(x)

windows()
hist(x,breaks=50)

windows()
hist(x,breaks=100)
```

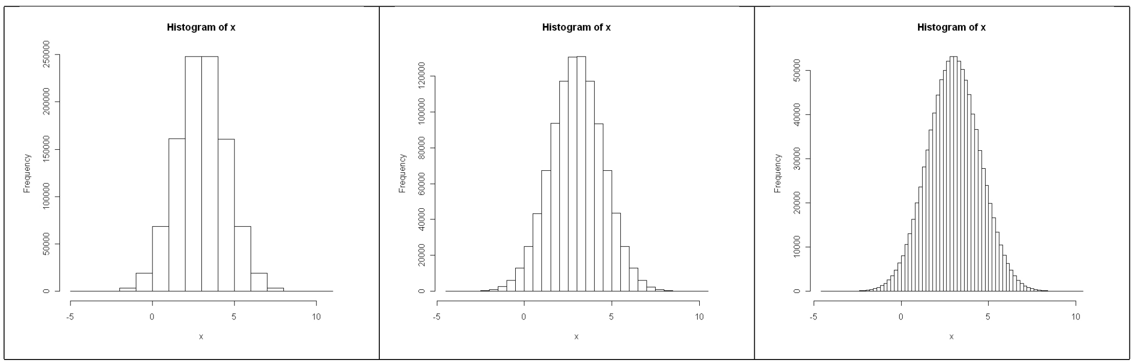
Tablo 2 çizilen histogramları göstermektedir. Bir grafiğe ve histograma çeşitli komutlar[3] kullanarak yeni çizgiler veya noktalar ekleyebiliriz. Aşağıdaki örneklere bakalım.

```
hist(x,breaks=100)
y <- seq(-5,10,length.out=100001)
lines(y,dnorm(y,3,1.5)*200000)

y <- seq(-5,10,length.out=101)
windows()
plot(y,dnorm(y,3,1.5))
lines(y,dnorm(y,3,1.5))

windows()
plot(y,dnorm(y,3,1.5),type="l")
abline(v=4.5) # x=4.5 noktasından geçen dikey ("v"ertical) bir doğru ekleyelim
```

[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.



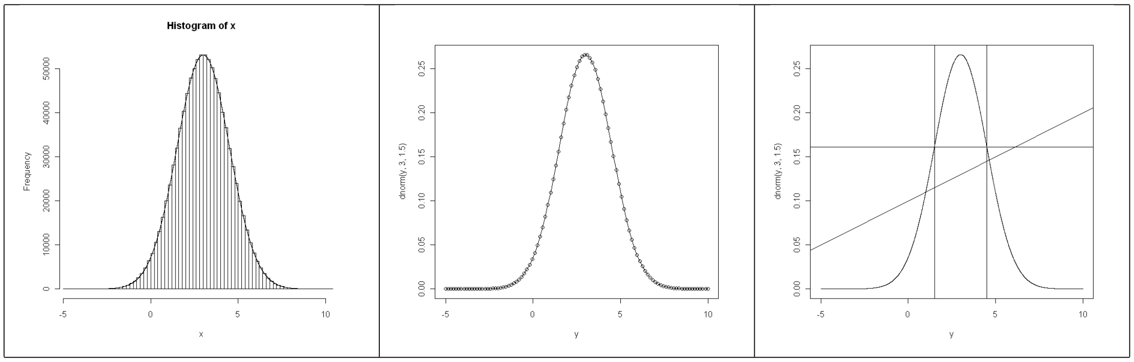

Şekil 1: Standart normal dağılımın yoğunluk grafikleri




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

```
abline(v=1.5) # x=1.5 noktasından geçen dikey ("v"ertical) bir doğru ekleyelim
abline(h=dnorm(1.5,3,1.5)) # y=dnorm(1.5,3,1.5) noktasından geçen
                            # yatay ("h"orizontal) bir doğru ekleyelim
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 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 veri[4] olsun.

```
3 25 94.9 12
547 32556 56
89 567
435 342.1
76.5 983.2
0 343
# 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.

```
x <- scan()
# enter tuşuna bastıktan sonra, komut satırında "1:" görünür
# CTRL+V yazarak kopyalanmış veriyi yapıştırabiliriz, 15 sayı x isimli dizide tutulur
# sonrasında komut satırında "16:" görünür
# enter tuşuna basarak veri girişi sonlandırılır ve 15 sayı okunmuş olur

# 1: 3 25 94.9 12
# 5: 547 32556 56
# 8: 89 567
# 10: 435 342.1
# 12: 76.5 983.2
# 14: 0 343
# 16:
# Read 15 items

x
#  [1]   3.0  25.0   94.9   12.0   547.0 32556.0   56.0   89.0   567.0
# [10]   435.0   342.1   76.5   983.2   0.0   343.0
```

Excel dosyasında bulunan sütünları 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.

[4] Bu 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:

```
boy    kilo  yas
1.72   72.3    25
 1.69    85.3    23
1.80  75.0  26
  1.61  66   23
1.73  69   24
# her satırda 3 değer
```

Masaüstümüzdeki R kısayoluna sağ tıklayarak özelliklere girerek R'ın çalıştığı başlama klasörünü öğrenelim[5]. Oluşturduğumuz metin dosyasını bu klasöre `data.txt` ismi ile kopyalayalım ve aşağıdaki komutları yazalım.

```
x <- read.table(file="data.txt",header=TRUE)
# eğer verinizde sütun başlıkları yoksa header=FALSE yapmak gerekir
x # x tablosunu görmek için <enter>a basın
#    boy    kilo  yas
# 1    1.72    72.3   25
# 2    1.69    85.3   23
# 3    1.80    75.0   26
# 4    1.61    66.0   23
# 5    1.73    69.0   24
x$boy
# [1] 1.72 1.69 1.80 1.61 1.73
x$kilo
# [1] 72.3 85.3 75.0 66.0 69.0
x$yas
# [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 ayrım için / kullanılması önemlidir. Ayrıca ayrımın
ile yapılması da mümkündür.

```
calisma <- getwd() # şu anki çalışma klasörünü calisma değişkenine eşitler
print(calisma) # ekrana bas
# "C:/Users/baydogan/Documents"
setwd("C:/") # çalışma klasörünü C klasörü yap^
getwd() # ekranda çalışma klasörünü bas
setwd("C:\\Programlar") # çalışma klasörünü C altında Programlar klasörü yap
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, obje[6] ya da bilgi yazmaya yarar. Eğer ekrana yorum yazacaksanız, komut içinde tek veya çift tırnak arasında yazmamız gerekir.

[5] Bunu R oturumunda da değiştirebilirsiniz
[6] Objeler vektör, matris, dizi, fonksiyon, liste (listeler C dilindeki yapılara benzer), tablo vs. olabilir.

```r
print("hata")
# [1] "hata"
x <- 1:5
print(x)
# [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:

```r
?det
?sample
?sin
?cbind
```

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

```r
apropos("norm")
# [1] "dlnorm"         "dnorm"          "normalizePath"  "plnorm"
# [5] "pnorm"          "qlnorm"         "qnorm"          "qqnorm"
# [9] "qqnorm.default" "rlnorm"         "rnorm"
```

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

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

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

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