R ile Populasyon Genetiği Simulasyonuna Giriş

Ders 1

A. Yetkin Alıcı, Elif Bozlak, Evrim Fer, Erinç Yurtman

22 Ocak 2018

Contents

Başlangıç

R, istatistiksel hesaplama yapmak ve grafik çizmek için oluşturulmuş araçlar bütünü ve bu araçları kullanmaya yarayan bir programlama dilidir. Ücretsiz ve özgür bir yazılım olan ve GNU/Linux, Mac ve Windows platformlarında çalışan R ile ilgili bilgi ve kaynaklara https://www.r-project.org/ adresinden ulaşabilirsiniz.

Programı yüklemek için https://cran.ncc.metu.edu.tr/ adresine bağlanarak, kullandığınız işletim sistemine uygun dosyayı indirip aynı adreste anlatıldığı şekilde bilgisayarınıza kurabilirsiniz.

R'ı çalıştırdığınızda karşınıza arayüz olarak bir konsol çıkacaktır. En basit durumda, bu konsola kod girerek R kullanılabilir. R yorumlanan bir programlama dili olduğu için, girdiğiniz kod doğrudan (derleme gibi bir işleme gerek olmaksızın) çalıştırılabilir.

Ders notları boyunca metin içinde aşağıdaki gibi örnek kodlara yer vereceğiz. Arkaplanı gri olan yerlerde kodlar, hemen aşağısında "##" ile başlayan satırlarda ise o kodlar çalıştırılınca R'ın vereceği sonuçlar görünecektir. Örneğin konsola "2+2" yazıp enter'a bastığımızda:

```
2+2
```

```
## [1] 4
```

R'da sonuç satırları hep köşeli parantez içinde sayılarla başlar. R birçok durumda çok elemanlı sonuçlar verdiği için, ilgili satırın sonucun kaçıncı elemanıyla başladığını belirten bu gösterim şekli R sonuç ekranında standart olarak bulunmaktadır. Örneğin 1 ile 10 arasından rastgele 40 sayı çekmek istersek:

```
sample(1:10, 40, replace = T)
```

```
##  [1]  6 10  6  7  2  6  7 10  2  7  2 10 10  3  8  6  8  2  4  2  2  2  5
## [24]  7  8  5 10  5  8 10  8  9 10  7  5  4  6  7  1  5
```

Bu şekilde konsolda etkileşimli olarak kod çalıştırabiliriz. Fakat bu, basit kod denemeleri dışındaki işler için çok kullanışsız bir yöntemdir. Gerçek çalışmalarda kullanılan yöntem, R kodunu örneğin "Analiz1.R" şeklinde .R uzantılı bir dosyaya kaydetmek ve daha sonra bu dosyayı R ile çalıştırmaktır. Bunu yaptığımızda R, dosya içindeki bütün kodları satır satır çalıştıracaktır.

Kod yazma, genellikle küçük kod parçalarının istediğimiz gibi çalıştığını deneme ve bu küçük kod parçalarını diğer kodlarımıza ekleme şeklinde gelişir. Bu süreçte, hem kodun tamamını görebileceğimiz ve çalıştırıp deneyebileceğimiz, hem de küçük parçaları hızlıca deneyebileceğimiz bir ortama ihtiyaç duyarız. Yani R konsolunun ve R kodlarımızı içeren dosyanın aynı anda açık olduğu ve dosyadan seçtiğimiz kod parçalarını kolayca çalıştırabildiğimiz bir ortama ihtiyacımız var. Bu ihtiyaca yönelik olarak çok sayıda görsel arayüz programı var. Biz bu kursta RStudio kullanacağız. Son derece gelişmiş bir arayüz programı olan RStudio, ücretsiz ve GNU/Linux, Mac ve Windows platformlarında çalışmakta. Bilgisayarınıza kurmak için https://www.rstudio.com/products/rstudio/download/ adresine gidebilirsiniz.

Şimdi, açmış olduğumuz R konsolunu kapatalım, ve RStudio'yu açalım. Çoğu programda görülen şekilde bir üst menü ve çeşitli pencerelere bölünmüş bir ekranla karşılaşacaksınız. Menüde "File" sekmesine tıklayıp "New File -> R Script"e tıkladığınızda, kodlarımızı yazacağımız yeni bir dosya açılacak. 4 pencereye bölünmüş olan RStudio ekranında; sol üst köşede kodlarımızı yazacağımız dosya, sol alt köşede R konsolu, sağ üst köşede tanımlayacağımız değişkenleri ve fonksiyonları gösterecek pencere ve sağ alt köşede yardım sayfalarını okumamıza ve çizdiğimiz grafikleri görmemize yarayan pencere bulunuyor olacak.

Konsola kod yazarak çalıştırmayı gördük. RStudio'nun sol alt penceresi, az önce açıp iki deneme kodu yazıp kapattığımız R konsolunun tam kendisi. Aynı kodları burada da deneyerek aşinalık kazanabilirsiniz. Şimdi de, dosyadan kod çalıştırmaktan bahsedelim. Sol üst pencerede, kod dosyamız bir metin editöründe açılmış durumda. Kodlarımız genellikle, buradaki örneklerin aksine tekrar kullanmak isteyeceğimiz kodlardan oluşacağı için, onları bir dosyada tutacağız. Dosyada yazılı ola bir kod öbeğini çalıştırmak için, ilgili satırların tamamını fare ile seçerek pencerenin üst tarafında bulunan "Run" tuşuna basın. Eğer bir satır seçimi yapmazsanız, "Run" tuşu sadece aktif olan satırı (imlecin yanıp söndüğü satırı) çalıştıracaktır. "Run" tuşu yerine, klavyeden "CTRL"+"ENTER" kısayolunu da kullanabilirsiniz. Dosyanızdaki tüm kodu baştan sona çalıştırmak için ise, menüden "Code -> Run Region -> Run All" tuşuna basabilir ya da klavyeden "CTRL"+"ALT"+"R" kısayolunu kullanabilirsiniz. İleride birden çok satır içeren örnek kodlar yazmaya başlayacağız. Bunları kendiniz de RStudio'da çalıştırarak aşinalık kazanabilirsiniz.

RStudio'da, kod dosyasını diske kaydetmeden de içindeki kodları çalıştırabiliyoruz. Fakat çoğu durumda kaydetmek isteyeceğiz.

Dosyayı kaydetmeden önce değinmek istediğimiz bir konu, çalışma klasörü konusu. Menüde, "Session -> Set Working Directory -> Choose Directory" tuşuna basarak, bilgisayarımızda bir klasörü çalışma klasörü olarak seçebiliriz. RStudio, kaydedilen dosyaları varsayılan olarak bu klasöre kaydeder. Ayrıca şu an bahsetmeyeceğimiz ama ileride önemli olacak olan, bilgisayardaki veri dosyalarımızı R oturumumuzda açma işlemi için de, çalışma klasörümüzün hangisi olduğunu biliyor olmak önemli olacak. Bu nedenle, bilgisayarınızda kolay ulaşabileceğiniz ve unutmayacağınız bir klasörü çalışma klasörü olarak seçin.

Örnekleri kendi dosyanıza kopyalayıp denedikten sonra, pencerenin üstündeki disket işaretine basarak ya da menüden "File -> Save" seçerek dosyanızı kaydedin. Yeni açılan bir dosya ilk defa kaydedildiğinde dosyaya vermek istediğiniz ad ve nereye kaydetmek istediğiniz sorulacak. İstediğiniz adı yazın ve çalışma klasörünü zaten belirlemiş olduğunuz için kaydedilecek varsayılan yeri değiştirmeden onaylayın. Eğer üzerinde çalıştığınız bir dosyanın eski halini bozmadan yeni çalışmanızı da kaydetmek istiyorsanız "File -> Save As" seçeneğini kullanarak farklı bir adla kaydedin. Kaydettiğiniz bir dosyayı açmak için de, menüden "File -> Open File" tuşunu kullanın.

Anlattıklarımız, kod dosyalarımızı kaydetmek ve tekrar açmak için geçerliydi. Veri dosyaları için işlemler biraz daha farklı, onlardan daha sonra bahsedeceğiz. Sağdaki iki pencerenin de nasıl bilgiler sunduğunu ileride göreceğiz.

R kodları nelerden oluşur?

Bir R kod öbeği, aşağıdaki şekilde sınıflandırabileceğimiz unsurlardan oluşmaktadır.

- Sayı, karakter dizileri ve mantıksal değer olabilen değerler:

5 1024 0.2 gibi nümerik, "x" "Aa" "ATCG" gibi karakter, TRUE FALSE gibi mantıksal değerler, R'da işlenen esas malzemelerdir. Aslında diğer bütün öğeler, bu değerleri/verileri organize etmek ve işlemek için oluşturulmuştur.

- Dilin temel özellikleri için ayrılmış özel kelime ve semboller ve R ile birlikte gelen temel fonksiyonlar:

, () " " {} [ ] = gibi semboller ile "for" "if" "function" gibi özel kelimeler, değişkenlerin ve fonksiyonların çalışmasında kullanılır ve kod akışını düzenler. Bunların nasıl kullanıldığından ilgili yerlerde bahsedeceğiz. Daha sonra değineceğimiz + - * / ^ %% gibi aritmetik operatörler ve == > < & | gibi mantıksal operatörler de çeşitli değişkenler veya değerler arasında basit işemler yaparlar.

Başlangıç kısmında, konsolun nasıl çalıştığını göstermek için bir örnek fonksiyon kullanmıştık. sample() gibi fonksiyonlar da, parantez içinde belirtilen parametrelere göre bir çıktı verirler. Örneklerimizde yeri geldikçe çeşitli temel fonksiyonlar kullanacağız. Bölümün sonunda da, ileride kullanacağımız bazı fonksiyonları tanıtacağız.

- Kullanıcı tarafından tanımlanan değişken ve fonksiyon isimleri:

Verilerimizi bellekte tutmak için değişkenler, analizlerimizi organize etmek ve tekrarlayabilmek için de fonksiyonlar tanımlarız. Bu değişken ve fonksiyonlara erişimimizi sağlayan isimler de R kod öbeklerini oluşturan diğer unsurlardır.

Veri tipleri

Bu çalışmamız boyunca, 3 tipte veri kullanacağız. Verilerin hangi tipte olduğunu görmek için class() fonksiyonunu kullanabiliriz.

```
class(1.1)
```

```
## [1] "numeric"
```

```
class(256)
```

```
## [1] "numeric"
```

```
class("atcg")
```

```
## [1] "character"
```

```
class(TRUE)
```

```
## [1] "logical"
```

Burada, dikkatinizi karakter tipindeki verilerin tırnak işaretleri içinde ifade edildiğine çekmek istiyoruz. Tırnak içerisinde gösterilen semboller, karakter tipinde algılanır.

```r
class("HLA-DPA1")
```

```
## [1] "character"
```

```r
class("256")
```

```
## [1] "character"
```

```r
class("TRUE")
```

```
## [1] "character"
```

Yine dikkat edilmesi gereken bir durum da, bir harf/sembol dizisinin tırnak içerisinde gösterilmezse karakter verisi olarak algılanmayacağıdır.

```r
class(atcg)
```

```
## Error in eval(expr, envir, enclos): object 'atcg' not found
```

Burada R, tırnak işareti görmediği için, atcg olarak bir değişken/fonksiyon ismi ya da tanımlı bir özel kelime aradı ve bulamayınca hata verdi.

Değişkenler

Verilerimizi tekrar ulaşabileceğimiz şekilde kullanabilmek için onları bilgisayarın belleğinde tutmak gerekir. Bunu değişkenler tanımlayarak yaparız.

```r
dizi = "acgtcagcatgcat"
```

Yukarıdaki kodda, "=" operatörü ile karakter tipinde bir veriyi, kendi tanımladığımız (istediğimiz bir isim ile oluşturduğumuz) bir değişkene değer olarak atadık. R'da (ve başka bir çok programlama dilinde) "=" işareti bir eşitliği göstermiyor. Bir değer atama operasyonunu ifade ediyor. Bu işlemle, işaretin sağındaki değer, solundaki değişkene atanıyor. Az sonra göreceğimiz üzere, işaretin sağındaki ifadenin bir değer olması gerekmiyor. Orada bir değişken olması durumunda değişkenin içindeki değer soldaki değişkene atanıyor. Bir fonksiyon olması durumunda fonksiyonun çıkardığı değer atanıyor.

Değişkeni istediğimiz isimle oluşturduğumuzu söyledik, ama şunlara dikkat etmeliyiz:

- Aynı harfin büyük ve küçük halleri farklı karakterler olarak algılanır.
- İsimler bir harfle başlamak zorundadır.
- İsimler boşluk içeremez.
- İsim içinde rakam ve bazı semboller geçebilir, fakat alttan çizgi (_) dışındaki sembollerin kullanılmasını tavsye etmiyoruz.

Bir değişkenin taşıdığı veriye ulaşmak için, değişken adını kullanıyoruz. Örneğin bir satıra sadece değişkenin adını yazarak, değişkenin taşıdığı değeri konsolda görebiliriz.

```
dizi
```

```
## [1] "acgtcagcatgcat"
```

Ya da, bu değişkeni bir fonksiyonda parametre olarak kullanarak, fonksiyonun ilgili veriyi işlemesini sağlayabiliriz. nchar() fonksiyonu, bir karakter verisinin kaç sembolden oluştuğu bilgisini çıktı olarak verir.

```
nchar(dizi)
```

```
## [1] 14
```

Burada, değişken isimlerinin tırnak içinde olmaması gerektiğini söylemek istiyoruz. Örneğin:

```
nchar("dizi")
```

```
## [1] 4
```

Görüldüğü üzere, dizi isimli değişken çağırıldığında taşıdığı "acgtcagcatgcat" verisini iletirken, "dizi" yazdığımızda 4 sembolden oluşan bir karakter verisini fonksiyona girmiş oluyoruz.

class() fonksiyonuna bir değer yerine bir değişken girdiğimizde, o değişkenin taşıdığı değerin tipini verecektir. Değişkenler, herhangi tipte bir değer taşıyabilir.

```
class(dizi)
```

```
## [1] "character"
```

İstersek, fonksiyonun çıktısını konsola yazdırmak yerine, başka bir değişkende kaydedebiliriz.

```
uzunluk = nchar(dizi)
uzunluk
```

```
## [1] 14
```

Halihazırda bir değer taşımakta olan değişkene, yeni bir değer atayabiliriz. Yukarıda gördüğümüz gibi, bu durumda eski değerin yerini son atanan değer alır.

```
uzunluk = 5
uzunluk
```

```
## [1] 5
```

Atama fonksiyonunu açıklarken, "=" işaretinin iki tarafında da değişken ismi yazabileceğini söylemiştik. Bu durumda, soldaki değişkenin içeriğine bakılmaksızın, sağdaki değişkenin içeriği soldakine kopyalanır. İşaretin sağındaki değişken ise hiçbir değişikliğe uğramaz.

```
a = 2
b = 5
a
```

```
## [1] 2
```

```
b
```

```
## [1] 5
```

```
a = b
a
```

```
## [1] 5
```

```
b
```

```
## [1] 5
```

Not: Belli bir tipte değer taşıyan değişken, isteyerek ya da farkında olmadan farklı bir tipe dönüştürülebilir. Belli bir tipte değer kabul eden fonksiyonlar/operatörler, farklı tip değer gelmesi durumunda o değeri tipini değiştirerek kullanabiliyor. Bu durum beklenmedik sonuçlara yol açabiliyor. Bu nedenle operatörlere ve fonksiyonlara varsayılan tipte değerler girmeye dikkat edilmelidir.

Operatörler

Operatörleri, fonksiyonların daha yalın halleri olarak düşünebiliriz. Yukarıdaki örneklerden de gördüğümüz üzere, fonksiyonlar parantez içindeki değerleri bir takım işlemlerden geçirdikten sonra bir değer verirler. Operatörler de, çevrelerindeki iki değeri belli bir işlemden geçirip sonuç olarak bir değer veren sembollerdir. (Tek değer üstünde çalışan operatörler de var, fakat onlara değinmeyeceğiz.)

Aritmetik operatörler, nümerik tipte değerler üzerinde 4 işlem yapmamızı sağlar.

```
4+4
```

```
## [1] 8
```

```
5-8
```

```
## [1] -3
```

```
6*9
```

```
## [1] 54
```

```
13/5
```

```
## [1] 2.6
```

```
2^3
```

```
## [1] 8
```

```
7%%5
```

```
## [1] 2
```

Mantıksal operatörler, her üç tip değer üzerinde işlem yapıp mantıksal tipte bir değer verirler. “Eşit mi?” (==) operatörü, iki değerin eşitliğini sınamaya yarar.

```
(1+4) == (2+3)
```

```
## [1] TRUE
```

```
"abc" == "Abc"
```

```
## [1] FALSE
```

```
TRUE == FALSE
```

```
## [1] FALSE
```

“Eşit değil mi?” (!=) operatörü, aynı karşılaştırmayı yukarıdakinin tersi sonuçları verecek şekilde yapar.

```
(1+4) != (2+3)
```

```
## [1] FALSE
```

```
"abc" != "Abc"
```

```
## [1] TRUE
```

```
TRUE != FALSE
```

```
## [1] TRUE
```

Büyük, küçük, büyük-eşit, küçük-eşit (>,<,>=,<=) operatörleri de yukarıdakiler gibi, ifadenin doğruluk/yanlışlık değerini sonuç olarak veren operatörlerdir. Tanımlama operatörleri değildir. Bekleneceği üzere, bu operatörlerin esas kullanım alanları, değişkenleri taşıdıkları değerler açısından karşılaştırmaktır.

```
a = 1
b = 1
a == b
```

```
## [1] TRUE
```

```
a < b
```

```
## [1] FALSE
```

```
a <= b
```

```
## [1] TRUE
```

Kullanacağımız diğer iki mantıksal operatör de, “ve” ve “veya” (&, |) operatörleri. Ve (&), aldığı iki değerin de doğru olması durumunda doğru, diğer durumlarda yanlış değeri iletirken; veya (|), değerlerin en az birisinin doğru olduğu durumda doğru, diğer durumda yanlış değeri iletiyor.

```
TRUE & FALSE
```

```
## [1] FALSE
```

```
TRUE | FALSE
```

```
## [1] TRUE
```

```
a = 5; b = 10; c = "Aa"; d = "aa"
(a < b) & (c == d)
```

```
## [1] FALSE
```

```
(a < b) | (c == d)
```

```
## [1] TRUE
```

Bu örnekle birlikte, noktalı virgül kullanımını da görmüş olduk. Ayrı satırlarda yazılması gereken kodlar çok kısalarsa, okunabilirlik açısından bu şekilde “;” ile ayrılarak aynı satırda yazılabilirler. Mantıksal değerleri, verilerimizden istediğimiz şartlara uyan alt kümeleri seçmek için ve kod akışını kontrol etmek için sıklıkla kullanacağız.

Veri yapıları

Şimdiye kadar örnek verilerimizi tek elemandan oluşan değişkenlerde tuttuk. Şimdi birden çok elemandan oluşan veri yapıları olan vektör ve matrisi tanıyacağız.

Vektör, her elemanı aynı tip değerden oluşan bir dizi elemandan oluşan bir değişkendir. c() fonksiyonu kullanılarak oluşturulur. Değişkenin taşıdığı değerlere ulaşmak, ve değiştrmek tek elemanlı değişkenlerde olduğu gibidir.

```
a = c(10,20,30,40)
b = c("AA", "aa", "Aa", "aa")
c = c(FALSE, TRUE, TRUE)
b
```

```
## [1] "AA" "aa" "Aa" "aa"
```

```
b = c("bb", "Bb")
b
```

```
## [1] "bb" "Bb"
```

Vektörlerin önemli bir özelliği, taşıdığı elemanlara ayrı ayrı da ulaşılabilmesi ve birbirlerinden bağımsız olarak değiştirilebilmeleridir. Bu işlemler için köşeli parantezler ([ ]) kullanılır.

```
a[2]
```

```
## [1] 20
```

```
a[3:4]
```

```
## [1] 30 40
```

```
a[1:3]
```

```
## [1] 10 20 30
```

```
a[1] = 75
a
```

```
## [1] 75 20 30 40
```

```
a[2:4] = c(11,12,13)
a
```

```
## [1] 75 11 12 13
```

Vektörlere eleman eklenebilir, vektörler birleştirilerek yeni vektörler oluşturulabilir. Fakat her elemanın aynı tip değer taşıması gerektiği unutulmamalıdır.

```
a[5] = 5
b = c(1.1, 1.2, 1.3)
c(a, b)
```

```
## [1] 75.0 11.0 12.0 13.0  5.0  1.1  1.2  1.3
```

Vektörlerle aritmetik ve mantıksal işlemler yapılırken, işleme giren diğer değişken/değer tek elemanlı ise işlem vektörün her elemanına ayrı ayrı uygulanır. İşlem iki vektör arasında ise, vektörlerin aynı sıradaki elemanları arasında uygulanır.

```
a = c(1,2,3); b = c(10,20,30)
a + b
```

```
## [1] 11 22 33
```

```
b ^ a
```

```
## [1]    10   400 27000
```

```
a ^ 2
```

```
## [1] 1 4 9
```

```
a == b
```

```
## [1] FALSE FALSE FALSE
```

```
a < 1.5
```

```
## [1]  TRUE FALSE FALSE
```

Vektörün bazı elemanlarını çağırmak için köşeli parantez içinde elemanların sıra numaralarını kullanmıştık. Bu numaralara indeks deniyor. Sadece tek bir elemanı ya da sıralı şekilde bir grup elemanı çağırmakla sınırlı değiliz. Vektörün istediğimiz elemanlarını çağırmanın 3 yolu daha var:

- Nümerik vektörle çağırmak:

```
a = c(10, 20, 30, 40, 50, 60, 70, 80); b = c(5, 2, 1, 7)
a[b]
```

```
## [1] 50 20 10 70
```

- Mantıksal vektörle çağırmak:

```
b = c(T, F, T, F, T, T, F, T)
a[b]
```

```
## [1] 10 30 50 60 80
```

```
b = c(T, F)
a[b]
```

```
## [1] 10 30 50 70
```

Eğer indeks olarak kullanılacak vektörün boyu, veriyi taşıyan vektörle aynıysa, TRUE olan değerler çağırılır. Indeks vektörü kısaysa, kendisini tekrar edecek şekilde kullanılır. TRUE yerine T, FALSE yerine F de örnekte görüldüğü gibi kullanılabilir.

Bu şekilde mantıksal vektörü oluşturmanın bir anlamı yok, istediğimiz elemanların indekslerini bir önceki örnekteki gibi yazabilirdik. Bunu, aşağıdaki örneğe giriş amacıyla gösterdik.

```
b = a < 45
b
```

```
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
a[b]
```

```
## [1] 10 20 30 40
```

```
b = (a > 25) & (a < 65)
a[b]
```

```
## [1] 30 40 50 60
```

- Eleman isimleriyle çağırmak:

Vektör elemanlarını ayrıca isimlendirmek de faydalı olabiliyor. İsimlendirilmiş vektörlerin elemanları, indeks yerine isimleriyle de çağırılabiliyor. names() fonksiyonu, verilen vektörün isimlerini gösterebildiği gibi, bu isimlerin değiştirilmesini de "=" operatörü aracılığıyla yapabiliyor.

```
names(a)
```

```
## NULL
```

```
names(a) = c("A1","A2","A3","A4","B1","B2","B3","B4")
names(a)
```

```
## [1] "A1" "A2" "A3" "A4" "B1" "B2" "B3" "B4"
```

```
a
```

```
## A1 A2 A3 A4 B1 B2 B3 B4
## 10 20 30 40 50 60 70 80
```

```
a["B2"]
```

```
## B2
## 60
```

```
b = c("A1","B1","A3","B3")
a[b]
```

```
## A1 B1 A3 B3
## 10 50 30 70
```

Matrisler, iki boyutlu eleman dizilerinden oluşan değişkenlerdir. Her bir satır indeksi - sütun indeksi kombinasyonuna denk gelen bir elemanı vardır. Vektörlerle bir çok özellikleri ortaktır. Tek tip veri taşıyabilirler, elemanlarına yine vektörlerin elemanlarına ulaşıldığı gibi ulaşılır. matrix() fonksiyonuyla oluşturulurlar.

```
matrix(data = 1:16, nrow = 4, ncol = 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

":" operatörü, etrafındaki değerlerle sınırlı olacak şekilde birer artan bir sayılardan oluşan bir nümerik vektör oluşturur. matrix() fonksiyonu da, görüldüğü şekilde, sütunları sırayla doldurarak, parantez içinde ilk parametre olarak girdiğimiz veriyi istediğimiz sayıda satır (nrow) ve sütuna (ncol) yerleştiriyor. Parametrelerin kullanımını ileride açıklayacağız.

```
b = c("A","T","C","G")
a = matrix(data = sample(b, 20, replace = T), nrow = 5, ncol = 4)
a
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "G"  "A"  "C"  "C"
## [2,] "A"  "T"  "T"  "A"
## [3,] "T"  "G"  "T"  "C"
## [4,] "A"  "C"  "C"  "G"
## [5,] "T"  "C"  "G"  "G"
```

```
a[2,4]
```

```
## [1] "A"
```

Yukarıdaki örnekte, "A","T","C" ve "G" karakterleri arasından rastgele 20 tane seçip 5 satır ve 4 sütunlu bir matris oluşturuyoruz ve ona "a" adını veriyoruz. Matrisimizi, adını kullarak konsola yazdırıyoruz. Sonra, a matrisinin 2. satır ve 4. sütunundaki elemanı yazdırıyoruz. Köşeli parantez içinde virgülle ayrılmış olarak önce satır, sonra sütun indeksi girilerek ilgili elemanlara ulaşılır.

```
a[2,4] = "T"
a
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "G"  "A"  "C"  "C"
## [2,] "A"  "T"  "T"  "T"
## [3,] "T"  "G"  "T"  "C"
## [4,] "A"  "C"  "C"  "G"
## [5,] "T"  "C"  "G"  "G"
```

Matrisin istediğimiz elemanına değer atayabiliriz. Fakat yeni eleman eklemek istediğimizde, vektöre göre farklı bir durumla karşılaşıyoruz. En az bir ya da daha fazla satır veya sütun ekleyebiliyoruz. Yani tek eleman eklemek mümkün değil.

```
a[1,5] = "G"
```

```
## Error in `[<-`(`*tmp*`, 1, 5, value = "G"): subscript out of bounds
```

Satır ya da sütun eklemek için, rbind() (row bind) ve cbind() (column bind) fonksiyonları kullanılıyor. Birleştirilmek istenen matris ya da vektörler, parantez içinde parametre olarak girilir. Satır ekleyerek birleştirmek için:

```
a = matrix(1:6, nrow = 2, ncol = 3)
b = matrix(21:29, nrow = 3, ncol = 3)
rbind(a,b)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## [3,]   21   24   27
## [4,]   22   25   28
## [5,]   23   26   29
```

Sütun ekleyerek birleştirmek için:

```
a = matrix(1:6, nrow = 2, ncol = 3)
b = matrix(21:26, nrow = 2, ncol = 3)
cbind(a,b)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5   21   23   25
## [2,]    2    4    6   22   24   26
```

Elemanları seçmek için, nümerik veya mantıksal vektörler de kullanılabilir.

```
a = matrix(sample(c("AA","Aa","aa"), 16, replace = T), nrow = 4, ncol = 4)
a
```

```
##      [,1] [,2] [,3] [,4]
## [1,] "aa" "aa" "AA" "Aa"
## [2,] "aa" "AA" "aa" "AA"
## [3,] "AA" "Aa" "aa" "Aa"
## [4,] "aa" "Aa" "AA" "AA"
```

```
satirlar = c(2,3)
sutunlar = c(T, F, F, T)
a[satirlar, sutunlar]
```

```
##      [,1] [,2]
## [1,] "aa" "AA"
## [2,] "AA" "Aa"
```

Matrisin satır ve sütunlarına isim atamak için rownames() ve colnames() fonksiyonları kullanılır.

```
rownames(a)
```

```
## NULL
```

```
colnames(a)
```

```
## NULL
```

```
rownames(a) = c("Jenerasyon1", "Jenerasyon2", "Jenerasyon3", "Jenerasyon4")
colnames(a) = c("Birey1", "Birey2", "Birey3", "Birey4")
a
```

```
##             Birey1 Birey2 Birey3 Birey4
## Jenerasyon1 "aa"   "aa"   "AA"   "Aa"
## Jenerasyon2 "aa"   "AA"   "aa"   "AA"
## Jenerasyon3 "AA"   "Aa"   "aa"   "Aa"
## Jenerasyon4 "aa"   "Aa"   "AA"   "AA"
```

```
a["Jenerasyon4",]
```

```
## Birey1 Birey2 Birey3 Birey4
##   "aa"   "Aa"   "AA"   "AA"
```

Matrisin tüm satırlarını, ya da tüm sütunlarını çağırmak istiyorsak, ilgili indeksin yerini boş bırakabiliriz. Yukarıdaki örnekte, "Jenerasyon4" satırını ve bütün sütunları çağırmış olduk.

Temel fonksiyonlar

Notlarda şimdiye kadar bazı fonksiyonların kullanımlarını gördük. Kendisini hemen parantezlerin takip ettiği kelimeler fonksiyonlardır. Aşağıdaki yapıda olurlar:

fonksiyon_adi(parametre_adi1 = "parametre1", numerik_parametre = 2, mantiksal_parametre = TRUE, "isim_kullanilmadan_girilen_parametre")

Parantez içinde virgülle ayrılmış şekilde parametreler verilir. Parametreler, isim = "parametre" şeklinde yazılabileceği gibi, isimsiz olarak sadece değer şeklinde de girilebilir. İsim kullanılmadan girilecekse, sıraları önemlidir. Opsiyonel parametreler yerine değer girilmediğinde, varsayılan değer kullanılır, mecburi parametrelerden girilmeyen olursa fonksiyon hata verir.

Her fonksiyonun bir yardım sayfası vardır. "?fonksiyon_adi" komutu yazıldığında, RStudio sağ alt pencerede yardım sayfasını açar. Bu sayfada fonksiyonun kullanım şekli ve parametrelerin açıklamaları örneklerle birlikte yer alır.

Fonksiyon kullanılan örneklerde görülen, fakat bazı okuyucuların gözünden kaçmış olabilecek bir durumu göstererek devam etmek istiyoruz. Kod içinde, kullanılan fonksiyonların yerine, o fonksiyonun üreteceği değer veya değerleri düşünmek gerekir. Yani,

```
a = sample(1:10, 5, replace = T)
```

örneğindeki gibi, sample() fonksiyonunun sonucunu bir değişkene atayabileceğimiz gibi, o fonksiyonun sonucunu doğrudan bir işlem içinde kullanabiliriz:

```
sample(1:10, 5, replace = T) + 100
```

```
## [1] 103 109 110 104 106
```

Bir fonksiyonun değerini başka bir fonksiyonun içinde parametre olarak kullanabiliriz:

```
matrix(data = sample(1:20, 20, replace = T), nrow = 5, ncol = 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    5   11    8   10
## [2,]   12    6    2   10
## [3,]   15    9    5   13
## [4,]   14    2   16    5
## [5,]    7    2   11    6
```

Bir fonksiyonun değerini, bir vektör ya da matrisin elemanlarını seçmek için de kullanabiliriz. Örneğin 10 eleman uzunluğunda bir vektörün rastgele 5 elemanını seçmek için:

```
a = c("a","b","c","d","e","f","g","h","i","j")
a[sample(1:10, 5, replace = FALSE)]
```

```
## [1] "e" "a" "j" "g" "b"
```

Şimdiye kadar örneklerde sample(), class(), nchar(), c(), names(), matrix(), rbind(), cbind(), rownames(), colnames() fonksiyonlarını kullandık. İleride yeri geldikçe başka fonksiyonlar da kullanacağız.

R projesinin internet sayfasında, R'a giriş notları bulunmaktadır: https://cran.r-project.org/doc/manuals/r-release/R-intro.html. Bu notların "Appendix D" bölümünde, R'da tanımlı olarak gelen fonksiyonların bir listesine ulaşılabilir.