R'a Giriş

H. Melike Dönertaş

14-02-2021

İçindekiler

Bu kitap oluşturulma aşamasında olup, haftalık olarak güncellenecek, yeni bölümler eklenecektir. Önerileriniz ve sorularınız için bana email ile ulaşabilirsiniz.

Bölüm 1

Giriş

1.1 R nedir?

R ile ilgili şimdiye kadar en uygun bulduğum, en çok hoşuma giden benzetme Matthew Keller'a ait. Keller diyor ki: 'R is like a magic, except you have functions instead of spells' - yani diyor ki R büyü gibidir, ancak büyülü sözcükler yerine fonksiyonlarınız vardır. SPSS, SAS kullanıcıları, 'muggle' gibidir. Ortamı değiştirme kabiliyetleri sınırlıdır. Onların analizi için birileri tarafından uygun görülmüş dizayn edilmiş algoritmalarla sınırlıdırlar ve üstüne para ödemek zorundadırlar. Keller, R programcıları ise 'büyücü'lere benzetiyor. R programcılar, alanında uzman olan kişiler tarafından yazılmış fonksiyonlara (yani büyülere) bağlı kalarak devam edebilecekleri gibi, kendi büyülerini de yaratabilirler ('sectumsempra' gibi lanetler de mümkün tabii). Bunları kullanmak / bunlara erişmek için para ödemezler, ve yeterinde deneyim kazandıklarında yapamayacakları bir şey yoktur.

1.2 R Avantajları

- Ücretsiz!
- Açık kaynak kodlu
- Aktif ve dinamik bir komünitesi var. Yardım almak çok kolay.
- Güncel
- Kod yazarken analiz hakkında düşünmeniz gerekiyor. Bu sebeple yaptığınız analizin, deney düzeneğinize, örnekleminize ve en önemlisi hipotezinize uygun olup olmadığını tartabiliyorsunuz. Rastgele tuşlara tıklayıp $p<0.05$ görünce alıp devam etmekten oldukça farklı!
- İstatistiksel testlerin varsayımlarına bağlı kalmadan, simülasyonlar ile empirik dağılımlar yaratıp test yapabilirsiniz.

- R notebook / R markdown gibi dökümanlar oluşturarak analizinizi / deneyinizi takip etmeyi ve yayınlamayı kolaylaştırmak ve tekrar edilebilirliğini sağlamak mümkün.
- Özellikle rutin olarak biyoistatistik / biyoenformatik çalışmıyorsanız, Windows kullanıcısı olma ihtimaliniz yüksek. R işletim sisteminden bağımsız olduğundan, kendi bilgisayarınızda çalışabilir, ve gerektiğinde analizinizi / kodunuzu başka platformlarda çalışanlarla rahatlıkla paylaşabilirsiniz.
- Bilim dili günümüzde İngilizce imiş gibi gözüküyor. Ben buna katılmıyorum. Bilimin dili bence grafikler. Yazdığınız 15 sayfalık bir makaleyi özetleyebilecek 3 grafik yapabilmek çok büyük bir güç (büyük konuştum, tabii her şeyi grafikleştirmek mümkün değil ama zamanla makaleleri okurken farketmeye başlıyor insan Excel, Graph Pad grafiklerinin R'da oluşturulmuş grafikler yanında nasıl kaldığını...)
- Bir de öğrenmesi en kolay dillerden birisi. Ancak bunun yanında öğrenme eğrisi lineer değil. Yani ilk zamanlar çok zor gelebilir (ilk zamanlar çok basit olmasına rağmen kaç defa matrisin satırı yerine sütunu ile işlem yapmaya çalıştığımı anlatamam bile!) ama alıştıktan sonra insanın kendisini geliştirmesi, yeni fonksiyon hatta başkalarının kullanımı için paket yazmak diğer dillere göre çok daha kolay.
- Son olarak, R paket sayısı, paketlerin güncellenme sıklığı, paket yazarlarının ulaşılabilirliği açısından özellikle biyoloji alanında çalışanlar açısından çok avantajlı. `Bioconductor` projesi özellikle biyoloji ile alakalı analizler için -omics data analizi için vs. inanılmaz avantaj sağlıyor.

1.3 R Öğrenmeye Nasıl Başlarım?

Bu sorunun tabii ki kesin net bir cevabı yok, kişiden kişiye, kullanımdan kullanıma farklılık gösterecektir en verimli yol. Ama yine de bir takım basamaklardan geçmemek imkansız (R'ı bilgisayarınıza indirmek ve kurmak gibi!). İlk olarak bu 'mutlaka olması gereken maddeler' ve yararlı bulduğum kimi basamakları sıralayalım.

1.3.1 R kurulumu

Çok basit: CRAN Anasayfasına gidip, kullandığımız işletim sistemi için olan versiyonu indiriyoruz. Bir çok kullanıcı için 'base' sürümü yeterli olacaktır. Sonrasında talimatları takip ederek R'ı kuruyoruz.

1.3.2 RStudio

R'ın kendi arayüzü oldukça sade ve yeterli olsa da, ben herkese RStudio'yu indirmelerini tavsiye ediyorum. RStudio da ücretsiz ve R programlama için bir çeşit arayüz gibi düşünebilirsiniz.

1.3.3 Kaynaklar

1.3.3.1 Online Dersler

- EdX - Statistics and R dersi ve ogrendikten sonra dersin devami
- Coursera - R Programming dersi
- R'ı R içinde öğrenin - swirl paketi

1.3.3.2 Bloglar

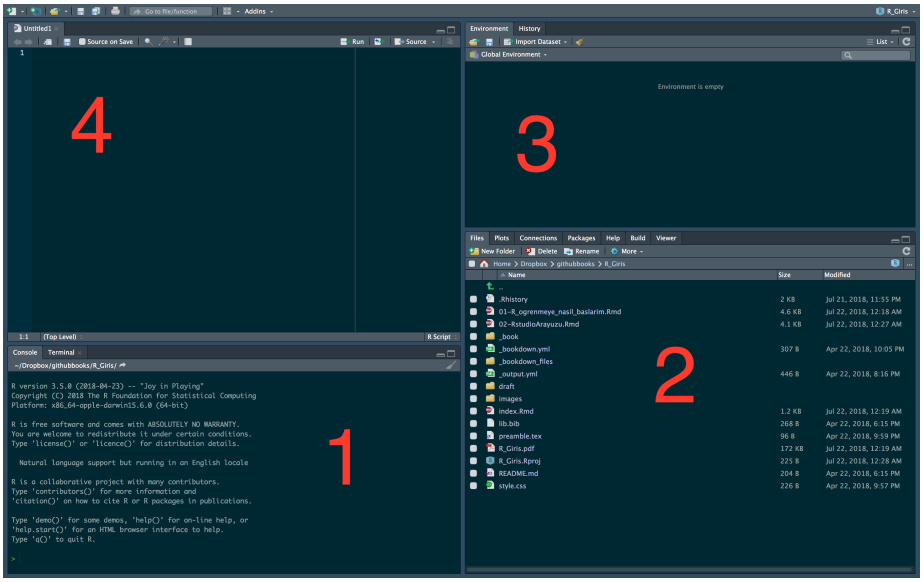
- R Blogger

1.3.3.3 Kitaplar

- R Graphics Cookbook - ggplot and more

Bölüm 2

RStudio Arayüzü



Şekil 2.1: RStudio Arayüzü

İlk olarak çalışacağımız ortamı tanıyalım.

RStudio'yu açtığınızda 'RStudio Arayüzü' resmine çok benzer bir tablo ile karşılacaksınız - belki 4. kısım hariç, oraya geleceğiz. Arkaplan muhtemelen beyazdır ve belki işletim sistemi farkları söz konusudur. Ama yine de bu ekran aşağı yukarı neresı ne işe yarıyor, kodu nereye yazacağız gibi kısa bir giriş yapmak için yeterli olacaktır.

2.1 Konsol, Terminal (1)

Konsol (Console): Burası kodu yazıp sonuçları gördüğümüz kısım. Denemek için `2+2` yazıp enter'a basalım.

```
2 + 2
```

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

Terminal: Burası da bilgisayarınızın terminali veya komut istemcisine erişim sağlayan bir kısım. RStudio'nun eski versiyonlarında bu özellik yoktu ancak yeni versiyon kullanıyorsanız bu sekme de mevcut olacaktır.

2.2 Dosyalar, Çizimler, Paketler, Yardım (2)

Dosyalar (Files): İçinde bulunduğumuz klasörün içindeki dosyaları listeliyor.

Çizimler (Plots): Çizdiğimiz grafikleri görüntüleyebileceğimiz kısım. Örneğin şu kodu konsola yazarak bu sekmenin işlevini gözlemleyebiliriz:

```
hist(rnorm(1000))
```

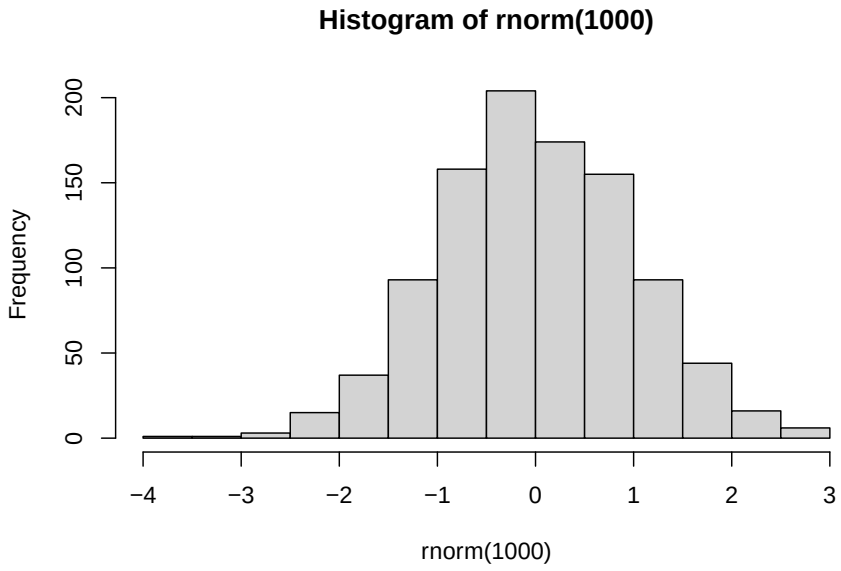
Bu ekranda büyütme, dışarı aktarma (export) gibi seçenekleri de görüyoruz. Ancak grafikleri kaydetmek için bu dışarı aktarma seçeneğini kullanmamayı, kaydetme işlemini de tamamen kod ile yapmayı tavsiye ediyorum.

Paketler (Packages): R'da yüklü olan çeşitli paketleri (kütüphaneleri) görüyoruz. Ancak buradan R ortamımıza yükleme yapmaktansa `library()` komutunu kullanmalıyız ki yazdığımız kodlar daha sonra hem biz hem başkaları tarafından kolaylıkla çalıştırılabilsin. Bu kısma sonra geleceğim.

Yardım (Help): Bu sekme RStudio'daki en büyük dostumuz. Açıp öğrenme linklerini kurcalamak iyi bir başlangıç. Ayrıca `?` operatörü ile burada fonksiyonların yardım sayfalarını görüntüleyebiliriz, örneğin konsolda şu kodu deneyelim:

```
?hist
```

Gördüğünüz gibi, yardım sekmesinde az önce histogram çizmek için kullandığınız `hist()` fonksiyonu için yardım ekranı açıldı. Bu ekranda, fonksiyonun tanımı, argümanları, sonucu hakkında bilgi alabiliyor ve en altta örnekleri görüyorsunuz. Örnekleri çalıştırıp, fonksiyonla ilgili daha fazla bilgi sahibi olabilirsiniz. Burada çıkan bilgiler, R'ı yüklediğimizde gelen 'base R' fonksiyonları için epey detaylı. İndireceğiniz bir çok paket için de detaylı bilgi bulabilirsiniz ancak detay ve örnek miktarı paket yazarlarının oluşturduğu yardım dosyalarına bağlı olan bir şey.



Şekil 2.2: 1000 rastgele değer kullanılarak çizilmiş bir histogram

2.3 Ortam, Geçmiş (3)

Ortam (Environment): Oluşturduğumuz değişkenlerin listesini ve bilgilerini burada görebiliriz. Şunu deneyelim:

```
x <- 3
```

Bu kod ile, x isimli değişkene, 3 değerini vermiş olduk. Artık sadece x yazıp enter'a basarsanız, x'in 3 olduğunu göreceksiniz. Aynı zamanda ortam sekmesinde artık x'in de listelendiğini görebilirsiniz.

Geçmiş (History): Mevcut R oturumunuzda yazdığınız kodları listeleyen bir kısım. Buradan önceden çalıştırdığınız kodları görebilirsiniz. Ancak daha pratik, daha çok kullanılan bir kısa yol, konsolda iken klavyedeki yukarı ok tuşu ile önceki kodları tekrar çağırmak. Geçmiş sadece açık oturumdaki kodları kaydediyor olsa da, .Rhistory dosyası sayesinde ortamlar arası geçmiş bilgisini aktarabilirsiniz. Yine de buna bağlı kalmak yerine daha sonra tekrar kullanacağımız kodları .R scripti olarak kaydetmek çok daha elverişli olacaktır.

2.4 Editör (4)

Editör ekranı, kodlarımızı yazıp çalıştırabileceğimiz ve sonrası için yorumlarımızla beraber kaydedebileceğimiz ekran. Bu bölüm muhtemelen açık olarak başlamamıştır RStudio. Burayı açmak için, sol üst köşedeki tuş ile yeni 'R script'i açabilirsiniz. Burada başka seçenekler de var, bunların ne olduğu avantajşarı hakkında sonrasında kısımlar gelecek. Şimdilik 'R script'ini seçelim. Açtığınız zaman, görüntüdeki gibi boş bir bölüm çıkacak. Önerim kodlarınızı her zaman konsol yerine burada yazmanız. RStudio'nun güzel bir özelliği burada yazdığını kodu konsolda kolay bir şekilde çalıştırmanızı sağlıyor, kopyala yapıştır yapmanıza gerek yok. Şimdi bu ekranda 2+2 yazıp, enter'a basalım. Konsolda hiç bir şey olmadığını göreceksiniz. Şimdi tekrar 2+2 nin olduğu satıra dönüp Ctrl+Enter (Mac kullanıyorsanız Cmd+Enter)'a basın. Konsolda hem kodun çalıştırıldığını hem de sonucun burada görüntülendiğini göreceksiniz (Eğer olmadıysa R Scripti seçeneğini seçmemiş olabilirsiniz). Her çalışma sonunda bu dosyayı kaydederseniz (Ctrl+S ya da Cmd+S), daha sonra yazdığınız kodlara tekrar dönme şansınız olacaktır. Daha önemlisi, buraya yazdığınız kodların içinden hatalı olanları silebilir düzenleyebilirsiniz bu sayede sadece işlevsel kodlarınız kaydolur. Geleneksel olarak scriptlerin uzantıları'.R'dır.

Bu açıkladığım panellerin yeri, sırası ve açık olup olmayacağı seçeneklerden değiştirilebilir.

Son olarak, RStudio'yu kapatırken size 'workspace'i kaydetmek isteyip istemediğinizi soracaktır. Buna benim önerim her zaman hayır demeniz, hatta seçeneklerden bunu varsayılan olarak ayarlayıp asla kaydetmemeniz. Objeleri ayrıca istediğimiz şekilde nasıl kaydedeceğimizi ilerde göreceğiz. Yine aynı şekilde history kaydetmek yerine scriptinizi kaydetmenizi ya da history kaydetseniz bile ayrıca mutlaka scriptinizi kaydetmenizi öneririm.

Bölüm 3

İlk Adımlar

Her zaman kod yazmaya başlamadan önce, en önemli basamak çalıştığımız klasörün ne olduğunu bilmek, gerekirse değiştirmek.

Hangi klasörde çalıştığımızı öğrenmek için `getwd()` fonksiyonunu kullanabiliriz (get working directory). Çalıştığımız klasörü belirlemek içinse `setwd()` fonksiyonu kullanılıyor:

```
getwd()
```

```
## [1] "/Users/melike/GDrive/githubbooks/R_Giris"
```

```
setwd('~/Desktop')
getwd()
```

```
## [1] "/Users/melike/Desktop"
```

Önerim, bilgisayarınızda R öğrenmek için bir klasör oluşturmanız ve kodunuzu, kullandığınız dosyaları vs. hep burada tutmanız. Aynı şekilde, belirli bir amaçla R kullandığınız zaman da masaüstüne rastgele isimlerle scriptleriniz kaydetmek yerine, düzenli bir şekilde uygun klasörleri içeriğe dair fikir veren isimlendirmelerle ya da belirli bir sistemle kaydetmeniz hayatınızı kolaylaştıracaktır.

3.1 Hesap Makinesi olarak R

En temekde hesap makinesi işlemleri yapmak için kullanabilirsiniz:

```
2+2
```

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

```
2*3
```

```
## [1] 6
```

```
10-5
```

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

```
3^2
```

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

```
450/3
```

```
## [1] 150
```

```
200/3
```

```
## [1] 66.66667
```

```
200%%3
```

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

```
200%/%3
```

```
## [1] 66
```

Çok açık olmayan operatörler:

- %%– belirli bir tabanda kalan işlemi, modüler aritmetik. Örneğimizde 200ün 3e bölümünden kalan 2
- %/% – tam sayı bölmesi, kalan olsa bile tam sayı olarak bölme işleminin sonucu

3.2 Fonksiyonlar

R kendi içinde fonksiyonlar barındırır. Örneğin, başlangıçta kullandığımız `getwd()` gibi. Farkettiysenz, fonksiyonlardan bahsederken hep parantez kullanıyorum. Fonksiyonları değişkenlerden ayırabileceğiniz en basit şekil bu. Çok basit bir kaç fonksiyona bakalım:

```
#10 tabaninda log
log10(100)
```

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

```
#2 tabaninda log
log2(100)
```

```
## [1] 6.643856
```

```
#4 tabaninda 10un log'u
log(10,4)
```

```
## [1] 1.660964
```

```
#e uzeri 1
exp(1)
```

```
## [1] 2.718282
```

```
# e uzeri 2
exp(2)
```

```
## [1] 7.389056
```

```
 # e uzeri 2 nin ln i - log taban belirtilmediginde e tabaninda islem yapar
log(exp(2))
```

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

```
# 16 nin koku
16^(1/2)
```

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

```
# ayni islem fonksiyon ile
sqrt(16)
```

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

3.3 R'da değişkenler

Değişkenler, veri tutucular olarak düşünülebilir. R'da değişkene değer atamak icin `=` veya `<-` operatörleri kullanılabilir. Örneğin x'e 3 değerini atamak için, aşağıdaki iki kod da geçerlidir.

```
x = 3
x
```

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

```
a <- 4
a
```

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

```
3 * 4
```

```
## [1] 12
```

```
x * a
```

```
## [1] 12
```

`<-` tarihsel olarak R camiasında çokça kullanılsa da, pratik nedenlerle `=` kullanımı da oldukça yaygın ve yanlış değildir.

3.3.1 Değişken isimleri

R'da değişkenlere verebileceğimiz isimler için bazı sınırlayıcı kurallar vardır. Değişkenler, bir harf ile ya da harfin takip ettiği nokta ile başlar. Öğrneğin, `benimdegiskenim` geçerli bir değişken isim iken `2değisken` geçerli değildir, çünkü 2 bir harf değildir. `.degisken` geçerli bir değişken ismidir ancak `.2degisken` değildir. Ayrıca R'da kimi özel anlam içeren keilemerin değişken

olarak kullanılması mümkün değildir, `if` ve `for` gibi. Bunları `?reserved` yazarak öğrenebilirsiniz.

Yasak olmasa da diğer bir sorun R içinde varolan fonksiyonların isimleri ile değişken yaratmak. Çoğu zaman sorun olmadan çalışsa bile, karışıklığa sebep olduğu durumlar olabilir.

R'da değişken isimleri küçük / büyük harfe duyarlıdır.

Değişken isimlerini tuttuğu veriyle alakalı seçmek kolaylık sağlar.

Bölüm 4

Temel obje türleri

R'da 5 çeşit temel obje türü var:

4.1 Karakter (character)

```
x = 'a'
x
```

```
## [1] "a"
```

```
a
```

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

Karakter değişkeni yaratmak için, tırnak işaretlerini (`''`) kullanıyoruz. Yani, `x` bir obje iken `'a'` bir karakterdir. Bu sebeple, `a` yazdığınızda, R `a` isminde bir obje arıyor, bizim a isminde bir objemiz olmadığından `obje bulunamadı` hatası veriyor. Bu arada R'da en iyi dostumuz hata mesajları. Genelde hata mesajı sorunun nerede olduğu hakkında çok iyi fikir verecektir.

```
class(x)
```

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

`class()` fonksiyonu obje sınıfını öğrenmemizi sağlıyor.

```
x = 'melike'
x
```

```
## [1] "melike"
```

```
class(x)
```

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

Karakter sınıfındaki objeler, tek bir karakter içermek zorunda değil, `'a'` da `'aaa'` da karakter objeleridir.

4.2 Nümerik (numeric)

```
x = 3
x
```

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

```
class(x)
```

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

```
x = 3.14
x
```

```
## [1] 3.14
```

```
class(x)
```

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

```
x= 1/0
x
```

```
## [1] Inf
```

```
class(x)
```

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

```
x = 0 / 0
x
```

```
## [1] NaN
```

```
class(x)
```

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

Burada ilginç olan kısımlar:

1. R tam sayıları da özellikle belirtmediğimiz sürece nümerik olarak alıyor
2. `Sonsuz`un (`Inf`) veri tipi nümerik
3. `NaN` ("not a number", "sayı değil") değeri de nümerik sınıfında.

4.3 Tam sayılar (integer)

```
x = 3L
x
```

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

```
class(x)
```

```
## [1] "integer"
```

Özellikle tam sayı oluşturmak istiyorsak sondaki L ekini kullanmalıyız.

4.4 Kompleks (complex)

```
x = 1+4i
x
```

```
## [1] 1+4i
```

```
class(x)
```

```
## [1] "complex"
```

4.5 Mantıksal (logical, boolean)

```
x = TRUE
x
```

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

```
class(x)
```

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

```
x = FALSE
x
```

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

```
class(x)
```

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

```
x = T
x
```

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

TRUE ve FALSE iki temel mantıksal değişken (True=Doğru, False=Yanlış). Sadece baş harfleri kullanılarak da ifade edilebilirler.

```
x = f
```

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

Ancak bu harfler büyük harf olmali yoksa `f` ismindeki objeye eşitlemeye çalışmış oluyoruz x'i.

Alıştırma olarak şu objelerin sınıflarını tahmin etmeye çalışıp, yanılıp yanılmadığınızı kontrol edebilirsiniz:

```
a='5'
b='T'
y=10.5L
x='y'
d=x
```

Tırnak işaretleri içinde veriler girildiğinden `a`, `b`, `x`, ve dolayısıyla `d` objeleri karakter tipindedir. `y` ise ondalıklı değer girdiğimizden `L` uzantısını kullansak da tam sayı değil nümerik tipindedir.

İlerledikçe `class()` fonksiyonu ile objelere baktığınızda farklı cevaplar alacağız. Mesela objelerden oluşan objelerimiz olduğunda - 1den 100e kadar olan sayıları içeren 10x10luk bir matrisiniz varsa mesela (`class(matrix(1:100,10))`). Veya R'ın nesne tabalı programlama (object oriented programming) dili olmasının getirisi olarak, yeni obje sınıfları ile çalıştığınızda.

Eğer ki obje açık bir obje sınıfına sahipse (örn. matrix gibi), nümerik değerlerden oluşan bir obje mi, karakter objesi mi, bunu görmek yerine sınıfını göreceksiniz. Bu durumda `mode()` fonksiyonunu kullanarak ne tip objelerden oluştuğunu görebiliriz, örn `mode(matrix(1:100,10))`.

Bölüm 5

Vektörler

R kimi veri yapıları üzerinde işlem yapar. En basit veri yapılarından biri olan vektörler, aynı temel obje türünden öğeler içeren dizilerdir.

```
x = c(1,2,3)
x
```

```
## [1] 1 2 3
```

Burada 'x'; 1, 2, ve 3 öğelerine sahip bir vektördür. Vektör oluşturmak için bu örnekte `c()` fonksiyonunu kullandık. cyi, `concatenate` (bağlamak) ya da `combine` (birleştirmek) kelimelerinden birinin kısaltılmış hali olarak aklınızda tutabilirsiniz.

Önceki örnekte nümerik tipte öğelerden oluşan bir vektör yarattık. Ancak vektörler hepsi aynı olduğu sürece diğer tiplerden de oluşabilir.

```
x = c(T,F,F) # logical
x
```

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

```
x = c('x','y','aa','oiasjfioasjf') # character
x
```

```
## [1] "x"           "y"           "aa"          "oiasjfioasjf"
```

```
x = c(1+4i,5+2i) # complex
x
```

```
## [1] 1+4i 5+2i
```

```
x = c(1L,5L) #integer
x
```

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

Şimdi de aynı tipte olmayan öğelerle vektör oluşturmaya çalıştığımızda ne oluyor, bunu görelim. İlk deneme olarak bütün tipleri girdi olarak verip bir vektör yaratmaya çalışalım.

```
x = c(1.5, 'karakter', 3i+2, TRUE, F)
```

Bu örnekte; nümerik, karakter, kompleks ve boolean tipinde öğelerle ile bir vektor yaratmak istedik. Ve hata almadık Peki 'x' objesinin modu ne ve nasıl bir vektor elde ettik?

```
x
```

```
## [1] "1.5"      "karakter" "2+3i"     "TRUE"     "FALSE"
```

Hata almadık ama vektör oluşturma sırasında öğeler kendi modlarını koruyamadılar. Data tipleri arasında bir hiyerarşi vardır ve R farklı tipte öğeleri birleştirirken bu objelerde hiyerarşide en yüksek olanı seçerek vektör yaratır. Buradan yapacağımız çıkarım, karakter tipi hiyerarşide en yüksek olan. Şimdi sırasıyla en yüksekte olanı çıkartarak hiyerarşiyi çözmeye çalışalım.

```
# karakteri cikarttigimizda
x = c(1.5, 3i+2, TRUE, F)
mode(x)
```

```
## [1] "complex"
```

```
x
```

```
## [1] 1.5+0i 2.0+3i 1.0+0i 0.0+0i
```

```
# kompleksi cikarttigimizda
x = c(1.5, TRUE, F)
mode(x)
```

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

```
x
```

```
## [1] 1.5 1.0 0.0
```

```
# numerigi cikarttigimizda
x = c(TRUE, F)
mode(x)
```

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

```
x
```

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

Bu durumda boolean < nümerik < kompleks < karakter çıkarımını yapabiliriz.

5.1 Vektorleri birleştirmek

Vektör yaratmak için kullandığımız c() fonksiyonu, aynı zamanda vektörleri birleştirmek için de kullanılabilir.

```
x = c(1,2,3)
x
```

```
## [1] 1 2 3
```

```
y = c(4,5,6)
y
```

```
## [1] 4 5 6
```

```
z = c(x,y)
z
```

```
## [1] 1 2 3 4 5 6
```

```
a = c(z,9,8)
a
```

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

Ve önceden bahsettiğimiz, data tipinin hiyerarşide en yüksek olana dönüştürülmesi bu durumda da geçerlidir.

```
x = c(1,2,3)
x
```

```
## [1] 1 2 3
```

```
y = c('a','b','c')
y
```

```
## [1] "a" "b" "c"
```

```
z = c(x,y)
z
```

```
## [1] "1" "2" "3" "a" "b" "c"
```

5.2 Vektor aritmetiği

Daha önce R'ı hesap makinesi gibi kullanabileceğimizi görmüştük. Vektörlerle de aynı işlemleri yapabilirsiniz.

```
x = c(1,2,3)
x * 3
```

```
## [1] 3 6 9
```

```
x + 1
```

```
## [1] 2 3 4
```

```
(x * 5) -2
```

```
## [1]  3  8 13
```

Burada önemli olan, işlemlerin her bir öğe üzerinde ayrı ayrı gerçekleştiriliyor oluşu. Eğer iki vektörümüz olsaydı,ve bu iki vektörü toplamak isteseysdik, benzer şekilde birinci öğenin ikinci vektördeki birinci öğe ile, ikinci öğenin ikinci vektördeki ikinci öğe ile vb. şekilde toplandığını görürdük.

```
x = c(1,2,3)
y = c(4,5,6)
x + y
```

```
## [1] 5 7 9
```

```
x * y
```

```
## [1]  4 10 18
```

Peki ya vektörlerimi eşit uzunlukta değilse? Bu durumda kısa olan vektor, uzun vektordeki öğeler bitene kadar tekrar tekrar kullanılır.

```
x = c(1,2,3,4)
y = c(2,3)
x * y
```

```
## [1]  2  6  6 12
```

- İlk vektördeki ilk eleman olan 1, ikinci vektördeki ilk eleman olan 2 ile çarpıldı ve sonucun ilk elemanı 2 oldu.
- İlk vektördeki ikinci eleman olan 2, ikinci vektördeki ikinci eleman olan 3 ile çarğıldı ve sonucun ikinci elemanı 6 oldu.
- İlk vektördeki üçüncü eleman olan 3, ikinci vektörde üçüncü bir eleman olmamasından dolayı ilk eleman olan 2 ile çarpıldı ve sonucun üçüncü elemanı 6 oldu.
- İlk vektördeki dördüncü eleman olan 4, ikinci vektörde ikinci eleman olan 3 ile çarpıldı ve sonucun üçüncü elemanı 12 oldu.

Yani bu tarz işlemleri yaparken dikkat etmek gerekiyor. Bu işlemin sonucunda R'ın nasıl davranacağını bilmeyen birisi 8 elemanlı bir vektör almayı, yani ilk vektör önce 2 sonra 3 ile çarpıp birleştirilir diye bekleyebilir örneğin. Ancak R böyle davranmıyor.

Peki ya vektörlerimizin uzunluğu birbirinin katı olmasaydı? Burada 4 öğeli bir vektörü 2 öğeli bir vektör ile çarptık. 4 öğeli bir vektörü, 3 öğeli bir vektör ile çarpmaya çalışsaydık farklı bir davranış bekler miyiz?

```
x = c(1,2,3,4)
y = c(1,2,3)
x * y
```

```
## Warning in x * y: longer object length is not a multiple of shorter object
## length
```

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

Hayır. Tam olarak önceki gibi bir davranış görüyoruz, ancak bu sefer R aynı zamanda bir uyarı mesajı veriyor ve vektörlerin birbirinin katı uzunlukta olmadığını söylüyor.

5.3 Vektör indisleri

Bir vektörün içindeki elemanlara ulaşmak için köşeli parantez `[]` operatorunu kullanabiliriz. Köşeli parantez içine çağırmak - ulaşmak istediğimiz öğenin indeksini, yani vektör içindeki sırasını yazarak istediğimiz elemana ulaşabiliriz. Burada önemli olan bir konu, R diğer kimi programlama dillerinin aksine 1-tabanlı indeksleme kullanıyor, yani ilk elemanın indeksi 1 (Örneğin python'da ilk elemanın indeksi 0dır).

```
x = c('a','b','c','d')
x[1] # Birinci eleman
```

```
## [1] "a"
```

```
x[1:2] # Birinciden ikinciye kadar (dahil) elemanlar
```

```
## [1] "a" "b"
```

```
x[2:4] # Ikinciden dorduncuye kadar (dahil) elemanlar
```

```
## [1] "b" "c" "d"
```

```
x[-3] # Ucuncu haric tum elemanlar
```

```
## [1] "a" "b" "d"
```

```
x[c(1,3)] # Bir ve ucuncu elemanlar
```

```
## [1] "a" "c"
```

Gördüğünüz gibi, bu sistemle sadece tek bir eleman değil, istediğiniz alt kümeye de ulaşabilirsiniz.

Burada daha önce görmediğimiz başka bir operatör (`:`) daha kullandık. Bu operatörü, bir dizi üretmek istediğimizde kullanıyoruz.

```
1:4
```

```
## [1] 1 2 3 4
```

```
5:10
```

```
## [1]  5  6  7  8  9 10
```

Tekrar indekslere dönecek olursak, açıkta bıraktığımız kimi sorular var.

```
x = c('a','b','c','d')
# olmayan bir elemani istemek NA verir
x[5]
```

```
## [1] NA
```

```
# ayni indeksi tekrar tekrar isteyerek elemanin tekrar etmesini saglayabiliriz
x[c(1,2,2,2,3)]
```

```
## [1] "a" "b" "b" "b" "c"
```

```
# elemanlari orijinal siralarina sadik kalarak cagirmak zorunda degiliz
x[c(3,1,4)]
```

```
## [1] "c" "a" "d"
```

Ayrıca boolean vektör kullanarak vektör altkümesi almak da mümkün. Eğer ki boolean `TRUE` ise, o indekse karşılık gelen eleman elde edilen altkümede yer alır, `FALSE` ise yer almaz.

```
x = c('a','b','c','d')
x[c(T,F,T,T)]
```

```
## [1] "a" "c" "d"
```

5.4 İsimlendirilmiş vektör

Şimdiye kadar vektör içindeki elemanları çağırmak için hangi sırada olduklarını kullanıyorduk. Oysa eğer ki vektör içindeki elemanların isimleri olsaydı bu isimleri kullanabilirdik.

```
myvec = c(birinci='x',ikinci='y')
myvec
```

```
## birinci  ikinci
##     "x"     "y"
```

```
myvec[1]
```

```
## birinci
##     "x"
```

```
myvec['ikinci']
```

```
## ikinci
##    "y"
```

Vektöre isim atamak, örnekteki gibi `c()` fonksiyonu içinde yapılabileceği gibi, sonradan `names()` fonksiyonu kullanılarak da yapılabilir.

```
a = c('x','y')
a
```

```
## [1] "x" "y"
```

```
isimvektorum = c('birinci','ikinci')
isimvektorum
```

```
## [1] "birinci" "ikinci"
```

```
names(a)=isimvektorum
a
```

```
## birinci  ikinci
##     "x"     "y"
```

Bölüm 6

Listeler

Bir önceki konuda vektörlerin tek bir temel obje tipinden oluştuğunu söyledik. Listeler bu açıdan vektörlerden farklıdır ve birden fazla obje tipini içerebilir.

İlk olarak bir liste oluşturalım:

```
mylist = list(1, 2, TRUE, 'a')
mylist
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] "a"
```

Bu sefer vektörlerde karşılaştığımız tür dönüştürme davranışı işe karşılaşmadık ve her eleman ilk girdiğimiz şekilde kaldı. Şimdi listenin 1. elemanına ulaşalım:

```
mylist[1]
```

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

Ne yazık ki eğer listemizin birinci elemanı ile işlem yapmak istiyorsak, bu doğru bir yöntem değil. Çünkü şu anda hala 1. elemana ulaşmadık. Listelerde her eleman bir tutucu olup, bir eleman içindeki değere ulaşmak istiyorsak çift köşeli parantez kullanmamız gerekir. Farkı daha iyi görebilmek için:

```
mylist[1]*5
```

```
## Error in mylist[1] * 5: non-numeric argument to binary operator
```

```
class(mylist[1])
```

```
## [1] "list"
```

```
mylist[[1]]
```

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

```
mylist[[1]]*5
```

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

İlk olarak listenin birinci elemanına tek köşeli parantez ile ulaşabileceğimizi varsayıp, 5 ile çarpmaya çalıştık. Ancak hata aldık. `class()` fonksiyonunu kullanıp gördük ki, tek köşeli parantez kullandığımda elde ettiğim obje hala bir liste. Listenin elemanının içinde depolanan bilgiye erişmek içinse iki tane köşeli parantez kullandım ve bu durumda 5 ile çarpabildim.

6.1 Listele indisleri

Listelerin diğer çok bariz olmayan özelliklerine bakalım. Örneğin, listemizin 4.elemanından kurtulmak isteyelim:

```
mylist[-4]
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] TRUE
```

Ya da

```
mylist[1:3]
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] TRUE
```

Bunlar tam olarak beklediğimiz gibi davranıyor ve 4. elemanı eksik liste elde ediyoruz. Peki çift köşeli parantez kullanarak, ilk üç elemandan oluşan bir vektör elde edebilir miyiz?

```
mylist[[1:3]]
```

```
## Error in mylist[[1:3]]: recursive indexing failed at level 2
```

6.2 Listeyi vektöre dönüştürmek

Hayır. İşe yaramadı. Peki bu ne demek, listeden asla vektör oluşturamaz mıyız? Hep liste olarak mı kalacak? Hayır, bunun için de `as.vector()` fonksiyonunu kullanabiliriz.

```
as.vector(mylist, mode = 'numeric')
```

```
## Warning in as.vector(mylist, mode = "numeric"): NAs introduced by coercion
```

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

```
as.vector(mylist, mode = 'character')
```

```
## [1] "1"    "2"    "TRUE" "a"
```

Bu fonksiyonlar varsa da çok kullanıldıklarını söyleyemem. Her veri türünün kendine özgü avantaj ve dezavantajları var ve farklı tür veri için belli veri tipleri daha uygun olacaktır. Genelde dönüştürmeler kimi fonksiyonlar sadece vektör, ya da sadece liste tipinde objelerle çalışıyorsa gerekli olabilir. Onun dışında listeden vektöre dönüştürme işlemleri kafa karıştırıcı olabilir. Bunun bir sebebi de listelerin elemanlarının tek bir elemandan oluşma zorunluluğunun olmamasıdır.

```
mylist2 = list(0, c(1, 2), T, 'a')
mylist2
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 1 2
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] "a"
```

Bu örnekte, listemizin ikinci elemanında iki elemanlı bir vektör sakladık. Şimdi az önceki yöntemi kullanmaya çalışırsak:

```
as.vector(mylist2, mode = 'numeric')
```

```
## Error in as.vector(mylist2, mode = "numeric"): 'list' object cannot be coerced to ty
```

```
as.vector(mylist2, mode = 'character')
```

```
## [1] "0"       "c(1, 2)" "TRUE"    "a"
```

İkisi de beklediğimiz gibi bir sonuç değil. Eğer ki elde etmek istediğimiz vektör `c('0', '1', '2', 'T', 'a')` vektörü ise, `unlist()` fonksiyonunu kullanabiliriz.

```
unlist(mylist2)
```

```
## [1] "0"    "1"    "2"    "TRUE" "a"
```

6.3 Listelerde aritmetik işlemler

Şimdi bir de vektörler üzerinde yaptığımız işlemler listeler üzerinde de geçerli mi ona kısaca göz atalım. İlk olarak eşlenik olarak değerlendirebileceğimiz bir vektör objesi (myvec) ve liste objesi (mylist) yaratalım:

```
myvec = 1:3
myvec
```

```
## [1] 1 2 3
```

```
mylist = list(1, 2, 3)
mylist
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

Vektörler konusunda gördüğümüz kimi işlemlerin üzerinden geçip, listeler üzerinde de geçerli olup olmadıklarına bakalım. İlk olarak birleştirme işlemi:

```
c(myvec, myvec)
```

```
## [1] 1 2 3 1 2 3
```

```
c(mylist, mylist)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 2
##
## [[6]]
## [1] 3
```

Yani, evet! Listeleri de birleştirebiliyoruz. Peki aritmetik işlemler?

```
myvec * 3
```

```
## [1] 3 6 9
```

```
mylist * 3
```

```
## Error in mylist * 3: non-numeric argument to binary operator
```

Hayır, aritmetik işlemleri yapabilmek için listenin elemanlarına tek tek erişmeliyiz.

6.4 İsimlendirilmiş liste

```
names(myvec) = c('a', 'b', 'c')
myvec
```

```
## a b c
## 1 2 3
```

```
names(mylist) = c('a', 'b', 'c')
mylist
```

```
## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3
```

Evet, isimlendirme de çalışıyor! Peki isimli listede alt küme alma işlemleri nasıl oluyor?

```
myvec['a']
```

```
## a
## 1
```

```
mylist['a']
```

```
## $a
## [1] 1
```

```
mylist[['a']]
```

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

Aynı indismiş gibi isimleri kullanarak da liste elemanlarına ulaşabiliyoruz. Ancak listelerde aynı amaçla kullanabileceğimiz başka bir operatör daha var: `$`

```
mylist$a
```

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

Ve bu operatör çift köşeli paranteze denk işliyor, görüdüğünüz gibi liste tipinde bir obje değil, direkt olarak ilk eleman içine kaydedilen bilgiye ulaştık. Peki bu vektörlerde de çalışır mı?

```
myvec$a
```

```
## Error in myvec$a: $ operator is invalid for atomic vectors
```

Hayır!

Bölüm 7

Matrisler

Matrisler, iki boyutta düzenlenmiş verilerdir, yanyana ya da altalta dizilenmis vektör gibi düşünülebilirler. İkişer elemanlı 3 vektörümüz oldugunu düşünelim:

```
a <- 1:2
b <- 3:4
c <- 5:6
```

Şimdi bunları yanyana birleştirip, iki satır üç sütunlu bir matris oluşturalım. Bunun için `cbind()` yani, column-bind (sütun bağlama) fonksiyonunu kullanacağız.

```
cbind(a, b, c)
```

```
##      a b c
## [1,] 1 3 5
## [2,] 2 4 6
```

Aynı şekilde `rbind()` yani row-bind (satır bağlama) fonksiyonu ile bu vektörleri satır satır birleştirip, 3 satır iki sutunlu bir matris de elde edebiliriz:

```
rbind(a, b, c)
```

```
##   [,1] [,2]
## a    1    2
## b    3    4
## c    5    6
```

Her satır ve sütunu ayrı ayrı vektör şeklinde birleştirerek matris oluşturmak yerine `matrix()` fonksiyonunu da kullanabiliriz.

```
matrix(1:9, ncol = 3)
```

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

Burada 1den 9a kadar sayıları 3 sütuna (ncol= 3 ile belirttik) yerleştirdik. Aynı şekilde satır sayısını da girebilirdik (nrow). `matrix()` fonksiyonu ile kullanabileceğimiz bir diğer argüman ise `byrow` argümanı. Önceki örnekte 1den 9a kadar sayıları içeren bir vektör oluşturup 3 sütuna bölmüştük. Bu fonksiyon varsayılan argümanlarla çalıştırıldığında önce birinci sütunu doldurup sonra ikinciye geçiyor. Eğer bunun yerine önce satırları doldurmasını istersek:

```
matrix(1:9, ncol = 3, byrow = T)
```

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

Şimdi 1:3 birinci satırı, 4:6 ikinci satırı oluşturdu.

7.1 Matris boyutları

Bunlar dışında matrislerin vektörlerden bir farkı da boyut özelliğinin olması. Oysa vektörlerde uzunluk özelliğimiz vardır. Şu örneklere bakalım:

```
mymat <- cbind(a, b, c)
length(a)
```

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

```
length(mymat)
```

```
## [1] 6
```

'mymat' objesinin boyuna bakmak istediğimizde toplam eleman sayısı olan 6yı veriyor. Oysa bu kaç satır ve sütundan oluştuğu bilgisini vermediğinden, matrisin yapısını anlamamızı zorlaştırıyor. Bu bilgiyi öğrenmek için `dim()` fonksiyonunu kullanacağız.

```
dim(mymat)
```

```
## [1] 2 3
```

Burada önemli olan bir bilgi, dim() fonksiyonu ilk olarak satır sayısını sonra sütun sayısını yazar. Son olarak `dim()` fonksiyonunun vektörler üzerinde işe yaramayacağını görelim:

```
dim(a)
```

```
## NULL
```

7.2 Matris indisleri

Matris içindeki verilere ulaşmak için yine köşeli parantez kullanılabilir. Ancak bu sefer birden fazla boyut olduğundan satır ve sütunu virgül ile ayırarak yazacağız. Örneğin:

```
mymat
```

```
##      a b c
## [1,] 1 3 5
## [2,] 2 4 6
```

```
mymat[1, 2:3]
```

```
## b c
## 3 5
```

`mymat[1, 2:3]` kodu ile, mymat matrisinin 1. satırının 2den 3e kadar olan elemanlarını istemiş oluyoruz.

7.3 Aritmetik işlemler

Aynı vektörlerde olduğu gibi aritmetik işlemler yapmak da mümkün.

```
mymat * 3
```

```
##      a  b  c
## [1,] 3  9 15
## [2,] 6 12 18
```

```
mymat + 2
```

```
##      a b c
## [1,] 3 5 7
## [2,] 4 6 8
```

Peki ya uzunluğu birden büyük bir vektör ile işlem yapmak istersek?

```
mymat
```

```
##      a b c
## [1,] 1 3 5
## [2,] 2 4 6
```

```
mymat * c(1, 2)
```

```
##      a b  c
## [1,] 1 3  5
## [2,] 4 8 12
```

Bunu anlamak için matrisin ilk üç elemanına bakalım:

```
mymat[1:3]
```

```
## [1] 1 2 3
```

Virgül ile satır ve sütun ayırmadığımızdan elde ettiğimiz obje de matrisin ilk 3 elemanından oluşan bir vektör oldu. Buradan gördüğümüz gibi matrisin birinci elemanı ilk satır ilk sütunda, ikinci elemanı ilk sütün ikinci satırda - ve sonra diğer sütunlara geçiyoruz. Bu matrisi bir vektör ile işlem yapmak istediğimizde de aynı vektörlerde olduğu gibi önce birinci eleman vektördeki birinci elemanla, ikinci eleman vektördeki ikinci elemanla işleme giriyor. Vektördeki elemanlar bittiğinden tekrar başa dönüyoruz ve matrisin üçüncü elemanı vektörün ilk elemanı ile işleme giriyor ve böyle devam ediyor. Bu işlemin satır ve sütun sayısıyla alakalı olmadığını göstermek adına bir başka örnek:

```
mymat * 1:3
```

```
##      a b  c
## [1,] 1 9 10
## [2,] 4 4 18
```

Matris transpozu, satır ve sütunların yer değiştirmesi işlemidir ve `t()` fonksiyonu ile gerçekleştirilir.

```
t(mymat)
```

```
##   [,1] [,2]
## a    1    2
## b    3    4
## c    5    6
```

Önceki örnekler matris çarpımı değil, matris içindeki değerlerin başka bir skalar ya da vektörle çarpımına örnekti. Rda matris çarpımı, ya da matris nokta çarpımı, yapmak da mümkün ve bu **%*%** operatörü ile yapılabilir. İki matrisin nokta çarpımı işlemine girebilmesi için birinci matrisin sütün sayısının, ikinci matrisin satır sayısına eşit olması gerekir.

```
mymat2 = matrix(c(1,2,3), nrow = 3, ncol = 1)
mymat2
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```
mymat %*% mymat2
```

```
##      [,1]
## [1,]   22
## [2,]   28
```

Bir matrisin tersi, kendisi ile çarpıldığında birim matrisi verir ve Rda `solve()` fonksiyonu ile hesaplanabilir. Bu işlem ancak kare matrisler üzerinde gerçekleştirilebilir.

```
mymat = matrix(sample(1:9), 3, 3)
mymat
```

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

```
solve(mymat)
```

```
##             [,1]        [,2]       [,3]
## [1,] -0.50000000  0.05000000  0.5500000
## [2,]  0.08333333  0.10833333 -0.1416667
## [3,]  0.41666667 -0.05833333 -0.3083333
```

```
mymat %*% solve(mymat)
```

```
##               [,1]          [,2]         [,3]
## [1,]  1.000000e+00 -5.551115e-17 4.440892e-16
## [2,] -4.440892e-16  1.000000e+00 5.551115e-16
## [3,]  0.000000e+00  0.000000e+00 1.000000e+00
```

Bunlar dışında sık kullanılan matris işlemleri için `crossprod()`, `det()`, `eigen()`, `svd()` gibi fonksiyonlar da temel Rda mevcuttur.

Son olarak, matrislerle çalışırken kolaylaştırıcı `rowMeans()`, `colMeans()`, `rowSums()`, `colSums()` gibi satır / sütun ortalamalarına ve toplamlarına erişmek için fonksiyonlar mevcut.

```
mymat
```

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

```
rowSums(mymat)
```

```
## [1] 15 16 14
```

```
colSums(mymat)
```

```
## [1] 15 12 18
```

```
rowMeans(mymat)
```

```
## [1] 5.000000 5.333333 4.666667
```

```
colMeans(mymat)
```

```
## [1] 5 4 6
```

Bölüm 8

Faktorler

Faktörler kategorik verilerle uğraşırken kullandığımız verilerdir. Yani değişkenimiz sadece belli bir set içinden değer alabilir.

Faktör oluşturmak için `factor()` fonksiyonunu kullanabiliriz. Diyelim ki sadece elma, armut, ya da muz olabilen bir verimiz olsun.

```
a <-  c("elma", "muz", "elma", "armut", "muz")
a
```

```
## [1] "elma"  "muz"   "elma"  "armut" "muz"
```

```
a_f <- factor(a)
a_f
```

```
## [1] elma  muz   elma  armut muz
## Levels: armut elma muz
```

Burada a ve a_f objeleri bir kaç açıcan birbirinden farklı. Değerler aynı gibi görülebilir ancak a objemiz bir karakter objesidir. a_f ise bir faktör objesidir ve karşılık aldığı değerler karakter değil, alabileceği kategorik değişkenlerin sayısıdır. Bunu daha iyi anlamak açısından:

```
as.numeric(a)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA NA NA
```

Bu karakterleri nümerik tipe dönüştürmek mümkün olmadığından NA'lardan oluşan bir vektör elde ediyoruz. Aynısını a_f'ye yapalım:

```
a_f
```

```
## [1] elma  muz   elma  armut muz
## Levels: armut elma muz
```

```
as.numeric(a_f)
```

```
## [1] 2 3 2 1 3
```

Şimdi NA yerine sayılar aldık. Bu sayılar da değerlerin faktörün alabileceği değerler arasından kaçıncı olduğuna denk geliyor. a_f'nin "levels" yani alabileceği değerler sırasıyla armut, elma, muz olduğundan, armut 1, elma 2, muz ise 3 oluyor. Burada değerlerin sıralaması varsayılan olarak alfabetik olarak yapılıyor. Ancak bunu değiştirmemiz mümkün:

```
a_f <- factor(a, levels = c("elma","armut", "muz"))
a_f
```

```
## [1] elma  muz   elma  armut muz
## Levels: elma armut muz
```

```
as.numeric(a_f)
```

```
## [1] 1 3 1 2 3
```

Karakterlerle çalışırken faktörleri anlamak görece daha kolay olabilir. Ancak özellikle nümerik kategorilerimiz varsa, objenin faktör olduğunu farketmemek analizimizi çok kötü bir çıkmaza götürebilir ve en sık yapılan hatalardan biridir.

```
a <- c(1, 2, 5, 10)
a_f <- factor(a)
```

Şimdi 30u a ve a_f vektörüne böldüğümüzde karşımıza ne çıkıyor bakalım:

```
30/as.numeric(a)
```

```
## [1] 30 15  6  3
```

```
30/as.numeric(a_f)
```

```
## [1] 30.0 15.0 10.0  7.5
```

Gördüğünüz gibi a_f'deki değerler nümerik hale dönüştürüldüğünde kendi nümerik değerlerini değil 1,2,3 ve 4 değerlerini aldığından işlem yanlış oldu. Bu özellikle R'a csv ya da excel dosyasından veri okunduğunda karşılaşılan bir problem.

Peki diyelim ki yanlışlıkla nümerik verimizi a_f gibi, faktör olarak okuduk. Bundan geri dönüş yok mu? Veriyi önce karakter sonra nümerik olarak alırsak sorun ortadan kalkıyor:

```
a_f
```

```
## [1] 1  2  5  10
## Levels: 1 2 5 10
```

```
as.character(a_f)
```

```
## [1] "1"  "2"  "5"  "10"
```

```
as.numeric(as.character(a_f))
```

```
## [1]  1  2  5 10
```

```
30 / a
```

```
## [1] 30 15  6  3
```

```
30 / as.numeric(as.character(a_f))
```

```
## [1] 30 15  6  3
```

Gördüğünüz gibi aynı sonucu aldık. Diğer bir çözüm de değerleri alabilecekleri seviyeleri kullanarak gerçek değerleri haline getirerek olabilir:

```
levels(a_f)
```

```
## [1] "1"  "2"  "5"  "10"
```

```
30 / as.numeric(levels(a_f)[a_f])
```

```
## [1] 30 15  6  3
```

Birebir bize çok yansımasa da faktörün en büyük yararı verinin depolanmasıyla ilgili. Diyelim ki elimizde 1 milyon kişinin doğduğu ay bilgisi var. Bunu karakter olarak kaydetmek yerine faktör olarak kaydedersek sadece her birey için 1den 12ye kadar birer sayı kaydedilir ve bir de bu sayıların karşılık geldiği değerler. Bu bütün ayların bütün karakterleriyle tek tek yazılmasından oldukça verimli bir çözümdür. read.table'ın değişkenleri faktör olarak okuma eğilimi de bundan kaynaklıdır.

Diğer önemli bir detay da faktör'e `c()` fonskiyonu ile yeni bir değer eklemekle ilgili. Bunu yapamayız:

```
a_f
```

```
## [1] 1  2  5  10
## Levels: 1 2 5 10
```

```
c(a_f, 8)
```

```
## [1] 1 2 3 4 8
```

```
c(a_f, "armut")
```

```
## [1] "1"     "2"     "3"     "4"     "armut"
```

Gördüğünüz gibi aynı `as.numeric()` fonksiyonunda karşılaştığımız gibi tüm değerler sıralamaya yani 1, 2, 3, ve 4e dönüştürülüp sonradan ekleme yapılıyor. Peki faktör vektörüne yeni değer ekleyemez miyiz?

```
factor(c(as.numeric(as.character(a_f)),8))
```

```
## [1] 1  2  5  10 8
## Levels: 1 2 5 8 10
```

Genel olarak yapmak istediğinizi yapacak hale getirmeniz mümkün ama özellikle tidyverse'te `forcats` paketini kullanmıyorsanız, çok da gerekli olmadıkça, özellikle başlangıç seviyesindeyken faktörlerden uzak durup, karakter ve nümerik verilerle çalışmanızı tavsiye ederim. Bu faktörler yararlı olmadığından değil,

alışkın olana kadar çalışması zor olduğundan. Faktörlerin özellikle yararlı olduğu bir kaç kullanım:

1) Modelleme: Faktörlerin en yaygın kullanımı modelleme yaparken karşımıza çıkıyor. Değişkenleri sürekli değişkenler olarak değil de faktör olarak kaydettiğimizde, modelde doğru şekilde yer almasını sağlayabiliyoruz. Ayrıca çoğu model fonksiyonları karakter vektörü aldıklarında çalışmayabiliyor.

2)`table` fonksiyonu ile kullanım: Diyelim ki 5 değer alabilen bir değişkeniniz var. Ama elinizdeki örneklem bu 5 değerin sadece 3ünü içeriyor. `table()` fonksiyonunu kullanarak her değerin kaç kez geçtiğini özetleyebilirsiniz ancak eğer faktörse bu özet 5 değerin tümünü içerirken, karakter ya da nümerik olduğunda 3 değer içerir:

```
a <-  c("elma", "muz", "elma", "armut", "muz")
table(a)
```

```
## a
## armut  elma   muz
##     1     2     2
```

```
a_f <- factor(a, levels = c("elma","muz","armut","karpuz","çilek"))
table(a_f)
```

```
## a_f
##   elma    muz  armut karpuz  çilek
##      2      2      1      0      0
```

Bu özellik kimi zaman kullanışlı olabilir.

3) Büyük veri ile çalışırken: Daha önce de bahsettiğim gibi faktörler özellikle büyük veri ile çalışırken tekrarlı veri söz konusu ile ciddi yer kazancı sağlamanız anlamına gelebilir.