

R Eğitim

Melike Dönertaş

22-03-2021

Contents

Bilgilendirme	5
Yardım	5
1 Çalıştay içeriği	7
2 Giriş	9
2.1 R Kurulumu	9
2.2 RStudio	9
2.3 Başlangıç	9
3 Temel Bilgiler	11
3.1 RStudio projesi	11
3.2 Hesap makinesi olarak R	12
3.3 Fonksiyonlar	13
3.4 Değişkenlerle çalışmak	13
3.5 R'da objeler	14
3.6 Paketler	24
4 İnsan prefrontal korteksinde gen ifadesinin zamansal dinamik- leri	25
4.1 Çalıştay için verinin ön hazırlığı	26
5 Veri ön işleme basamakları	27
5.1 read.csv() ve readRDS() ile veri okuma	27
5.2 Kesifci veri analizi	30

5.3	ProbesetID - Gen eslestirmesi	36
5.4	Quantile Normalization	44
5.5	Ornek bilgisini duzenleme	47
6	Temel bileşenler analizi	51
7	Yasla değişim analizi	53
8	Biyolojik cinsiyete göre farklılık gösteren gen analizi	57
9	Kaynak önerileri	61
9.1	Online ders	61
9.2	Kitaplar	61
9.3	Diğer	62

Bilgilendirme

22 Mart 2021 tarihinde Bioinforange tarafından düzenlenen Bioinfoconference kapsamında verilen R Eğitim çalıştay materyalini içerir. Çalıştay materyali Melike Dönertaş tarafından hazırlanmıştır, öneri ve düzeltmeler için konu başlığında “[Bioinforange R Eğitim]” kullanarak email ile ulaşabilirsiniz (lütfen sadece öneri ve düzeltmeler için arkadaşlar, çalıştay boyutunu göz önünde bulundurursanız hatalarınıza email ile yardımcı olmam imkansız. Yardım için aşağıdaki başlığa bakabilirsiniz).

Yardım

R konusunda karşılaştığımız sorular için öncelikle aldığınız hata ve uyarıları incelemenizi, `?` operatörünü veya `help()` komutunu kullanarak fonksiyon, veriseti ve obje yardım sayfalarına bakmanızı, R sıkça sorulan sorular kitapçıklarına sorularınızın dahil olup olmadığını kontrol etmenizi (Genel SSS, Windows’a özgü SSS, Mac OS’a özgü SSS), bir paket kullanıyorsanız ve sorunuz bununla ilgiliyse paket yardım ve kullanım örnekleri dosyalarını incelemenizi, ve eğer bu aşamalardan sonra hala cevap bulamadıysanız sorunuzu “Nasıl güzel soru sorulur?” yönergesine göz attıktan sonra Stack Overflow R bölümüne sormanızı tavsiye ederim. Alternatif olarak, kodlama konusunda yardımlaşma platformlarına (RSG Türkiye’nin Slack kanalı gibi¹) sorularınızı gönderebilirsiniz.

¹bu şekilde başka platformlar biliyorsanız lütfen bildirin, güncelleyebilirim

Chapter 1

Çalıştay içeriği

R'a giriş niteliğinde birkaç ders içeriği zaten mevcut. Bu çalıştayda, süremizin sadece 2 saat olduğunu ve interaktifliğin kısıtlı olduğunu göz önüne alarak, bir R eğitiminden ziyade R'ın biyolojide uygulamalarına yönelik kısa bir örneklemeye hedefledim. Çalıştay R'daki temel işlemler, operatörler, fonksiyonlar ve görselleştirme araçlarının kullanımına örnekler içerecek. Bunun için bir transkriptom veriseti kullanacağız. Bu veriseti halka açık veritabanı olan GEO'dan indirilecek. Ancak bu çalıştay “transkriptom veri analizi nasıl yapılır?”ı öğretmeyi amaçlamamaktadır. Transkriptom veri analizinde batch etkileri, normalizasyon gereklilikleri vs. bir çok kaygı vardır ve buna göre analiz basamakları gerektirmektedir. Burada yapacağımız örnek çok basit olup, gerekli olabilecek bazı basamakları atlamaktadır, bu sebeple transkriptom veri analizi örneği olarak değil R uygulamasına örnek olarak düşünülmelidir.

Chapter 2

Giriş

2.1 R Kurulumu

R'ı kullandığınız işletim sistemine göre CRAN anasayfasından indirebilirsiniz. Özellikle başlangıç seviyesinde **base** indirmek yeterli olacaktır. Kaynak kodu R bazlı değil de C/C++/Fortran gibi dillerde yazılmış olan paketleri kurmak için **Rtools** indirmeniz gerekebilir ama şimdilik **base** indirerek devam edebiliriz.

2.2 RStudio

RStudio, R için bir IDE yani “entegre geliştirme ortamı”dır ve ücretsiz olarak RStudio websayfasından indirilebilir. Entegre geliştirme ortamları kod yazma, sonuçları ve çizdiğiniz grafikleri vs. görüntüleme, otomasyon, hata ayıklama (debugging) gibi programlama ve yazılım geliştirme bileşenlerini içinde bulunduran araçlardır. RStudio ise bunların ötesinde başka programlama dilleriyle beraber kodlama, raporlama, versiyon kontrolü, komut istemcisi ve zamanlı iş çalıştırma ve daha bir çok özellik ile birlikte geliyor. R kodlama için kesinlikle RStudio ile çalışılmasını öneriyorum.

2.3 Başlangıç

Hem R hem Rstudio'yu yükleyip, RStudio açtığımızda karşımıza 3 ekran çıkar. Bunlar haricinde sol üst köşedeki “yeni” tuşuna basıp R Scriptini seçersek 4. ekran açılır. Hızlıca bunların içeriğine bakalım:

- Files: İçinde bulunduğumuz klasörün içindeki dosyaları listeliyor.

- Console: Burası kodu yazıp sonuçları gördüğümüz kısım. Denemek için `2+2` yazıp enter'a basalım.
- Terminal: Burası da bilgisayarınızın terminali veya komut istemcisine erişim sağlayan bir kısım.
- Environment: Oluşturduğumuz değişkenlerin listesini ve bilgilerini burada görebiliriz.
- Plots: Çizdiğimiz grafikleri görüntüleyebileceğimiz kısım.
- Help: Bu sekme RStudio'daki en büyük dostumuz. Açıp öğrenme linklerini kurcalamak iyi bir başlangıç. Ayrıca `?` operatörü ile burada fonksiyonların yardım sayfalarını görüntüleyebiliriz.
- History: Mevcut R oturumunuzda yazdığımız kodları listeleyen bir kısım. Buradan önceden çalıştırdığımız 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.
- 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 çalıştırılabilin.

Bunlardan yeni açtığımız ekran da script (betik) editör ekranı, yani kodu yazıp kaydedeceğimiz kod editörümüz. Kodu konsola yazıp çalıştırabiliriz ama önerim hep editörde çalışıp çalışan kodları kaydetmeniz. Editörde yazdığımız kodu kopyala yapıştır yapmadan çalıştırmanın kolay bir yolu var: Windows'ta `Ctrl + Enter`, Mac'te `Cmd + Enter`.

Chapter 3

Temel Bilgiler

Kodlari editorde yazip Ctrl+Enter ile calistirabilirsiniz. Alternatif olarak Konsol'da da kodlari yazabilirsiniz ancak editore yazdiklarinizi kaydederek yazdiklarinizdan elinizde bulunmasini istediklerinizi kaydedebilirsiniz. Onerim hep konsolu kullanmaniz, kullanisli olan kodlari tutup digerlerini / calismayanlari silip temiz bir sekilde dosyanizi kaydetmeniz.

Ilk olarak hangi klasorde nerde calisiyoruz buna bakalim. Bu R'da dosya acarken ve yaptiklarimizi kaydederken onemli bir detay. Bunun icin `getwd()` yani 'get directory' komutunu kullanacagiz. Bunu calistirdigimizda aldigimiz sonuc hepimizde farkli olacaktır. Bu bilgisayarinizda nerede calistiginizi gosteriyor:

```
getwd()
```

```
## [1] "/Users/melike/GDrive/githubbooks/20210322_REgitim"
```

Bu varsayilan ortamda degil de belli bir klasorun icinde calismak istiyorsaniz, istediginiz klasore `setwd()` fonksiyonu ile asagidaki sekilde gecebiliriz. Ancak onerim az sonra aciklayacagim sekilde RStudio projesi kullanmaniz.

```
setwd("~/GDrive/projects/R_Egitim")
```

3.1 RStudio projesi

R'da calisirken her zaman proje olusturmanizi tavsiye ederim. Bu sayede hem her projenize ait kod tarihi ayrica saklanir, hem de calistiginiz klasor vs. gibi konularda endiselenmeniz gerekmez. RStudio projelerinin bunlar haricinde versiyon kontrol konusu vs. gibi ileri asamalarda kullanacaginiz diger konularda da

faida saglar. Sag ust kosedeki New Project'e basip, 'yeni klasor'u secerek yeni proje olusturabilirsiniz. Bunu yaptıktan sonra `getwd()` fonksiyonunu tekrar calistiralim:

```
getwd()
```

```
## [1] "/Users/melike/GDrive/githubbooks/20210322_REgitim"
```

Benim sonucumda degisen bir sey yok cunku zaten proje icinde calisiyordum ancak sizin sonucunuz degismis olmalı.

3.2 Hesap makinesi olarak R

```
2+2
```

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

```
5*7
```

```
## [1] 35
```

```
20-4
```

```
## [1] 16
```

```
3^2
```

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

```
100/2
```

```
## [1] 50
```

```
9%%5
```

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

```
9%/%5
```

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

3.3 Fonksiyonlar

R kendi yazacaginiz fonksiyonların yanında kendi içinde çok sayıda fonksiyon barındırır, az önce kullandığımız `getwd()` fonksiyonu gibi. Bunlar fonksiyon adını takiben, parantez içinde belli argümanlar alırlar. Birkac örnek:

```
log10(x = 100)
```

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

```
log10(100)
```

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

```
sqrt(16)
```

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

3.3.1 Yardım almak

R'da çoğu fonksiyon yardım sayfası içerir ve bunlara `?` operatörü ile ulaşılır:

```
?log10  
?getwd
```

3.4 Değişkenlerle çalışmak

Değişkenler veri tutuculardır. Değişken atamak için `<-` ya da `=` operatörleri kullanılır. Tarihsel olarak `<-` kullanılsa da bugün pratik nedenlerle `=` daha yaygındır ve kullanımı yanlış değildir.

```
a <- 3  
print(a)
```

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

```
b = 5
print(b)
```

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

```
c = a + b
print(c)
```

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

```
c
```

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

Rda degisken isimleriyle ilgili kimi kisitlamalar var: - harf ya da nokta ile baslar - ozel anlami olan kelimeler kullanilmaz (`if` ya da `for` gibi). bunlari `?reserved` yazarak ogrenebilirsiniz. - yasak olmasa da Rda yer alan fonksiyon isimleri ile ayni degiskenler yaratmamaya calismak gerekir, soruna yol acabilir. - degiskenler kucuk buyuk harfe duyarlidir, `a` ve `A` birbirinden farkli degiskenlerdir. - simdiye kadar ornekleri `a b x` gibi harflerden olusturmus olsak da, tuttugu veriyle ilgili bir isim secmek iyi bir aliskanliktir.

3.5 R’da objeler

3.5.1 Obje türleri

5 temel obje turu var; karakter, numerik, tam sayi (integer), kompleks, mantiksal (logical/boolean).

3.5.1.1 Karakter

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

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

```
class(x)
```

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

Karakter objeleri tek karakter icermek zorunda degildir

```
y = "bu da karakter"  
y
```

```
## [1] "bu da karakter"
```

```
class(y)
```

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

Degiskenler `a` gibi tirnak isareti olmadan yazilir, karakterler ise `"a"` ya da `'a'` seklinde yazilir. R'da `'` ile `"` kullanimi karakterlerde farketmez. Ayrica tirnak icindeki karakter sadece harf olmak zorunda degil, ne olursa olsun bir karakter objesidir:

```
a = '5'  
a
```

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

```
class(a)
```

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

3.5.1.2 Nümerik ve tam sayılar

```
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"
```

Özellikle tam sayı oluşturmak istiyorsak sonda L ekini kullanmalıyız.

```
x = 3L  
x
```

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

```
class(x)
```

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

3.5.1.3 Kompleks

```
x = 5+2i  
x
```

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



```
class(x)
```

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

3.5.1.4 Mantıksal (Logical, boolean)

```
x = TRUE  
x
```

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

```
class(x)
```

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

```
y = T  
y
```

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

```
class(y)
```

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

```
z = FALSE  
z
```

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

```
class(z)
```

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

```
a = F  
a
```

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

```
class(a)
```

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

```
a = 'F'  
a
```

```
## [1] "F"
```

```
class(a)
```

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

Bunlar temel obje turleri olsa da, ileride class fonksiyonunu kullandiginizda baska sonuclar alabilirsiniz. Bu temelde R'in nesne tabali dogasindan kaynakli.

Ayrice objeniz bir matris ise, numerik degerlere mi karakterlere mi sahip, bunu ogrenmek icin `mode()` fonksiyonunu kullanmaniz gerekecek.

3.5.2 Vektör

Ayni temel obje turunden olusan elementler iceren dizilerdir:

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

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

```
class(x)
```

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

burada `c` vektor olusturmak icin kullanilmis bir fonksiyondur, combine ya da concatenate kelimelerinin kisaltilmisi olarak dusunulebilir.

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

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

```
class(x)
```

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

```
x = c('a', 'b', 'cc')
x
```

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

```
class(x)
```

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

Aynı tipte olmayan değerleri birleştirirseniz, beklenmeyen davranışlarla karşılaşabilirsiniz.

```
x = c(3.14, 'karakter', 3i+2, TRUE, F, 5L)
x
```

```
## [1] "3.14"      "karakter" "2+3i"      "TRUE"      "FALSE"     "5"
```

```
class(x)
```

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

Hata almadık ama objeler birleştirme sırasında kendi modlarını koruyamadılar. Bu sebeple eğer numerik vektör karakter vektörüyle birleştirirseniz, numerik değerler karaktere dönüşeceğinden artık numerik işlemler yapamazsınız.

Vektörün belli bir elemanını almak için `[]` operatörü kullanılır.

```
x[2]
```

```
## [1] "karakter"
```

3.5.3 Liste

Listeler, birden fazla obje türünü içerebilir:

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

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

Listenin belli bir elemanini almak icinse `[[]]` operatoru kullanilmalidir:

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

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

```
mylist[[3]]
```

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

3.5.4 Matris

Matrisler iki boyutlu verilerdir, yanyana dizilmis ayni uzunluktaki vektorler gibi de dusunebilirsiniz:

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

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

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

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

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

```
dim(x)
```

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

```
x[1,1]
```

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

```
x[1,2]
```

```
## b
## 3
```

```
x[1,]
```

```
## a b c
## 1 3 5
```

```
x[,2]
```

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

```
colnames(x)
```

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

```
rownames(x)
```

```
## NULL
```

```
rownames(x) = c('satir1','satir2')
x
```

```
##          a b c
## satir1 1 3 5
## satir2 2 4 6
```

```
dimnames(x)
```

```
## [[1]]
## [1] "satir1" "satir2"
##
## [[2]]
## [1] "a" "b" "c"
```

3.5.5 Faktör

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

```
a = c("elma", "muz", "elma", "armut", "muz")
a
```

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

```
class(a)
```

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

```
a = factor(c("elma", "muz", "elma", "armut", "muz"))
a
```

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

```
class(a)
```

```
## [1] "factor"
```

```
levels(a)
```

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

3.5.6 Data frame

Data frame görüntü itibariyle matrisleri andirsa da farklı veri tiplerini barındırabilmesinden dolayı, liste gibidir.

```
x = data.frame(id = c(1,2,3,4),
               isim = c('ali','veli','ayse','fatma'),
               ogrenci = c(T,T,F,T))
x
```

```
##   id  isim ogrenci
## 1  1   ali     TRUE
## 2  2   veli     TRUE
## 3  3   ayse    FALSE
## 4  4 fatma     TRUE
```

```
class(x)
```

```
## [1] "data.frame"
```

data.frame içinde bir sütunu almak için matrislerde olduğu gibi [] ya da \$ operatorunu kullanabilirsiniz:

```
x[,2]
```

```
## [1] "ali"  "veli" "ayse" "fatma"
```

```
x$isim
```

```
## [1] "ali"  "veli" "ayse" "fatma"
```

```
x[3,2]
```

```
## [1] "ayse"
```

```
x$isim[3]
```

```
## [1] "ayse"
```

3.6 Paketler

Son olarak R’da ozellesmis bazi fonksiyonlari ve veri tiplerini bulabileceginiz fonksiyonlar vardır. Bunlardan bazilari R’i indirdiginiz CRAN üzerinden, bazilari (ozellikle biyolojik veri analizi ile ilgili olanlari) Bioconductor üzerinden, bazilari da paketi olusturan kisilerin github repolari üzerinden indirilir. Indirme komutlari farkli olsa da, hepsi `library(PAKETADI)` komutu ile R’a yuklenir. Kullanisli bir kac paketi yukleyelim:

CRAN’dan indirmek icin:

```
install.packages('ggplot2')
```

Bioconductor’dan paket indirmeden once, CRAN’dan BiocManager paketini indirmeniz gerekir:

```
install.packages("BiocManager")
```

Bioconductor’dan paket indirmek icin, BiocManager’i `library()` fonksiyonu ile Ra yuklemeden `BiocManager::` ile bu paket icindeki bir fonksiyonu kullanabilirsiniz:

```
BiocManager::install("preprocessCore")
```

GitHub’dan bir paket indirmek icinse CRANDan indireceginiz `devtools` paketine ihtiyaciniz vardır. Paketlerin Github sayfalarindan indirme komutuna ulasabilirsiniz. Biz calistayda bu sekilde bir paket kullanmayacagiz.

Chapter 4

Insan prefrontal korteksinde gen ifadesinin zamansal dinamikleri

Bu calismada 2011 yilinda yayinlanmis bir makalede uretilmis olan bir microarray (mikrodizin?) veriseti kullanacagiz. Gorece eski bir teknolojiyle uretilmis olsa da verinin ve veri isleme basamaklari acik bir sekilde makalede paylasilmis durumda. Bu calistay mikrodizin / transkriptom veri analizi calistayi olmadigindan bu basamaklardan bazilarini atlayacagiz (orn. kismen islenmis veri ile baslayacak ve veride batch etkisi olmadigini varsayacagiz) ve kendi verinizi analiz edecek olursaniz bu calistayi birebir takip etmeniz yeterli olmayacaktır. Ancak makalede basamaklar anlatildigindan, kendiniz onlari tekrar edebilir daha detayli ogrenebilirsiniz.

Kisaca, bu verisetinde cok sayida, farkli yaslarda bireylerin olum sonrasi beyin, prefrontal korteks bolgesinden alinan orneklerde gen ifade verisi var. Biz de bu verisetini indirip, yasa bagli anlatimi degisen genler neler vs. gibi kimi sorulara cevap arayacagiz. Veriseti GEO’da herkese acik bir sekilde bulunuyor, ancak veriyi indirme asamasinda kullanilacak paket ve fonksiyonlar calistayin odagini degistirebileceginden o kismi atlayip, ilk indirme islemi tamamlandiktan sonraki kisma odaklanacagiz.

Veriyi indirmek icin GEO baglantisi: <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE30272>

Makale: Colantuoni, C., Lipska, B. K., Ye, T., Hyde, T. M., Tao, R., Leek, J. T., Colantuoni, E. A., Elkahouloun, A. G., Herman, M. M., Weinberger, D. R., & Kleinman, J. E. (2011). Temporal dynamics and genetic control of transcription in the human prefrontal cortex. *Nature*, 478(7370), 519–523. <https://doi.org/10.1038/nature10524>

4.1 Calistay icin verinin on hazirligi

Bu kısmi calistay sirasinda yapmayacagim ve sizlere on hazirligi yapilmis bir veriseti sunacagim. Ancak kendiniz veriyi GEO'dan indirip ayni basamaklari tekrar etmek isterseniz, basamaklari burada paylasiyorum:

Kismi islenmis veriyi GEOquery paketi ile GEOdan indirebiliriz:

```
BiocManager::install("GEOquery")

library(GEOquery)
gse <- getGEO('GSE30272', GSEMatrix=TRUE)
```

Bu bize bir ExpressionSet objesi veriyor:

```
gse
```

Buradan bireylere ait metadata, gen ekspresyon matrisi, ve gen-probeset eslesmesini bulabilecegimiz bir feature verisi olusturalim:

```
metadata = pData(gse$GSE30272_series_matrix.txt.gz)
exprmat = exprs(gse$GSE30272_series_matrix.txt.gz)
featdata = fData(gse$GSE30272_series_matrix.txt.gz)
```

Bazi bilgileri kullanmayacagiz, bu yuzden verisetinin sadece belli sutunlarini aliyorum:

```
metadata = metadata[,c(1,2,10:17)]
featdata = featdata[,c(1,6,7,8)]
```

.rds objesi olarak kaydetmek icin:

```
saveRDS(exprmat, './data/expressionmatrix.rds')
saveRDS(featdata, './data/featuredat.rds')
saveRDS(metadata, './data/metadata.rds')
```

.csv dosyasi olarak kaydetmek icin:

```
write.csv(exprmat, './data/expressionmatrix.csv')
write.csv(featdata, './data/featuredat.csv')
write.csv(metadata, './data/metadata.csv')
```

Chapter 5

Veri on isleme basamaklari

Veriyi GEO'dan ifade matrisi, metadata yani bireylere dair bilginin bulunduğu bir tablo, ve feature data yani probeset-gen eşleşmelerinin verildiği bir tablo olarak kaydettim. Şimdi bunları okuyarak başlayacağız. İlk olarak gen ifade verisiyle başlayalım.

5.1 read.csv() ve readRDS() ile veri okuma

`read.csv` fonksiyonu hem URL'den okuma hem de bilgisayarımızdaki bir dosyadan okuma yapmamızı sağlar. Dosyayı github repomdan indirip, fonksiyona `file =` değişkeni olarak bilgisayarınızda indirdiğiniz yeri vererek okuyabilirsiniz. Şimdi kolaylık olması açısından direkt olarak github repomdan okuyacağız. Bu fonksiyon ayrıca başka argümanlar da alıyor. `header` dosyada sütun isimlerinin yer alıp almadığını belirtmek için kullanılıyor ve mantıksal bir argüman yani `TRUE` ya da `FALSE` değerlerini alabilir. Kullanacağımız bir diğer argüman ise `row.names =`, bu da satır isimlerinin hangi sütunda yer aldığını belirtmemiz için gerekli. Eğer bu argümanı kullanmazsak, matrisimizin satır isimleri de bir sütun olarak okunacaktır.

```
expr = read.csv(file = 'https://github.com/mdonertas/20210322_REgitim/raw/master/data/expressionm
                header = T,
                row.names = 1)
```

İlk 6 satır ve 4 sütuna bakalım:

```
expr[1:6,1:4]
```

```
##                GSM749899  GSM749900  GSM749901  GSM749902
```

```
## HEEBO-002-CTRL2I1 -1.455124 -2.6244200 -1.904436 -1.5793507
## HEEBO-002-CTRL2I10 1.044670 0.4490985 0.619243 0.1930628
## HEEBO-002-CTRL2I11 1.164344 1.5833201 1.150024 1.5764080
## HEEBO-002-CTRL2I14 1.597227 1.4105770 1.469449 2.6433802
## HEEBO-002-CTRL2I15 -1.454337 -1.9784923 -1.485409 -1.5038039
## HEEBO-002-CTRL2I16 1.396667 0.7461117 1.212514 0.6586667
```

Her sey yolunda gozukuyor ama bir de bu objenin daha onceden ogrendigimiz gibi `class`ina bakalim:

```
class(expr)
```

```
## [1] "data.frame"
```

Genellikle `read.csv` fonksiyonu bir `data.frame` olusturur. Biz bu objede kimi matris operasyonlari yapmak istiyoruz ve verimiz de sadece numerik degerlerden oluyor. O yuzden bunu matrise donusturebiliriz. Bunun icin `as.matrix` yani ‘matrix olarak’ fonksiyonunu kullanabiliriz.

```
expr = as.matrix(expr)
```

Tekrar ilk birkac satir sutun ve class kontrolumuzu yapalim:

```
expr[1:6, 1:4]
```

```
##                GSM749899  GSM749900  GSM749901  GSM749902
## HEEBO-002-CTRL2I1 -1.455124 -2.6244200 -1.904436 -1.5793507
## HEEBO-002-CTRL2I10 1.044670 0.4490985 0.619243 0.1930628
## HEEBO-002-CTRL2I11 1.164344 1.5833201 1.150024 1.5764080
## HEEBO-002-CTRL2I14 1.597227 1.4105770 1.469449 2.6433802
## HEEBO-002-CTRL2I15 -1.454337 -1.9784923 -1.485409 -1.5038039
## HEEBO-002-CTRL2I16 1.396667 0.7461117 1.212514 0.6586667
```

```
class(expr)
```

```
## [1] "matrix" "array"
```

Arraylerden baslangicta bahsetmemistim, kullanimi cok asiri yaygin degil ama bunlar da 1 ya da daha fazla boyutlu vektorler olarak dusunulebilir, yani iki boyutlusu matris ama daha fazla boyutlu array olusturmak mumkun.

Simdi bir de matrisimizin boyutlarına bakalim. Bunu ya satir (`nrow()`) ve sutun sayisini (`ncol()`) yazdirarak ya da `dim()` fonksiyonu ile yapabiliriz.

```
nrow(expr)
```

```
## [1] 30176
```

```
ncol(expr)
```

```
## [1] 269
```

```
dim(expr)
```

```
## [1] 30176 269
```

Matrisimizde 30176 satir, 269 sutun var - yani 30176 probeset ID, 269 ornek. Daha fazla ilerlemeden once bir veri okuma sekli daha gostermek istiyorum.

Alternatif olarak ayni veriyi bir `.rds` - R objesi olarak da kaydetmistim. Onu okumak icin de `readRDS()` fonksiyonunu kullanabiliriz. Bunun avantajı verinizin bir csv dosyasina kaydedilerek olasi bozulmalarini engellemenin yani sıra, obje turunu vs. de koruyabilirsiniz.

```
expr = readRDS(file = './data/expressionmatrix.rds')
class(expr)
```

```
## [1] "matrix" "array"
```

```
expr[1:6,1:4]
```

```
##          GSM749899 GSM749900 GSM749901 GSM749902
## HEEBO-002-CTRL2I1 -1.455124 -2.6244200 -1.904436 -1.5793507
## HEEBO-002-CTRL2I10 1.044670 0.4490985 0.619243 0.1930628
## HEEBO-002-CTRL2I11 1.164344 1.5833201 1.150024 1.5764080
## HEEBO-002-CTRL2I14 1.597227 1.4105770 1.469449 2.6433802
## HEEBO-002-CTRL2I15 -1.454337 -1.9784923 -1.485409 -1.5038039
## HEEBO-002-CTRL2I16 1.396667 0.7461117 1.212514 0.6586667
```

```
dim(expr)
```

```
## [1] 30176 269
```

Gordugunuz gibi bu dosyayi okudugumuzda veriyi direkt olarak matrix olarak okuyabildik ve `data.frame`'den donusturmek zorunda kalmadik.

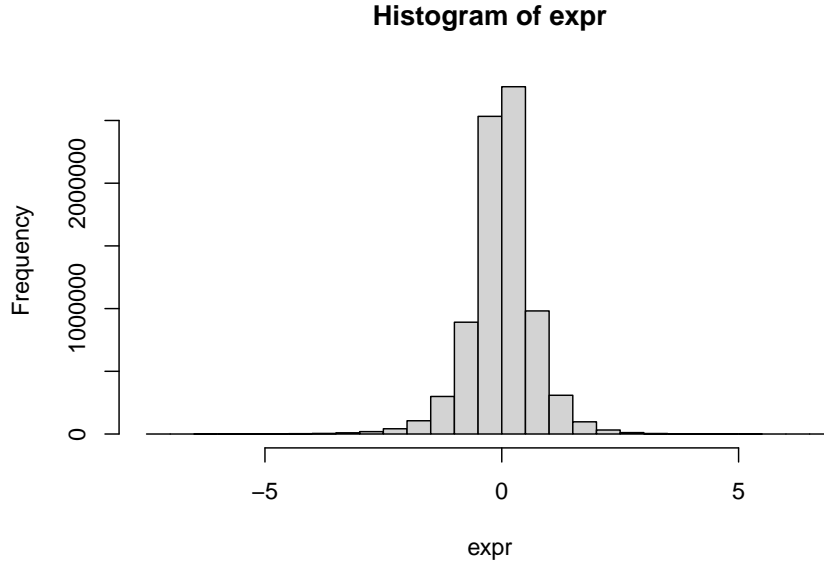
5.2 Kesifci veri analizi

Daha ileri gecmeden hemen bu asamada cok kisaca verimiz neye benziyor bakabiliriz. Bunun icin temel bir kac gorsellestirme fonksiyonu kullanacagiz.

5.2.1 Histogram

Ilk olarak verimizin dagilimi nedir, ne gibi degerler var bunlara bakalim. Bunun icin histogram kullanabiliriz, ve ilk olarak temel Rda histogram cizmeyi gorelim.

```
hist(expr)
```

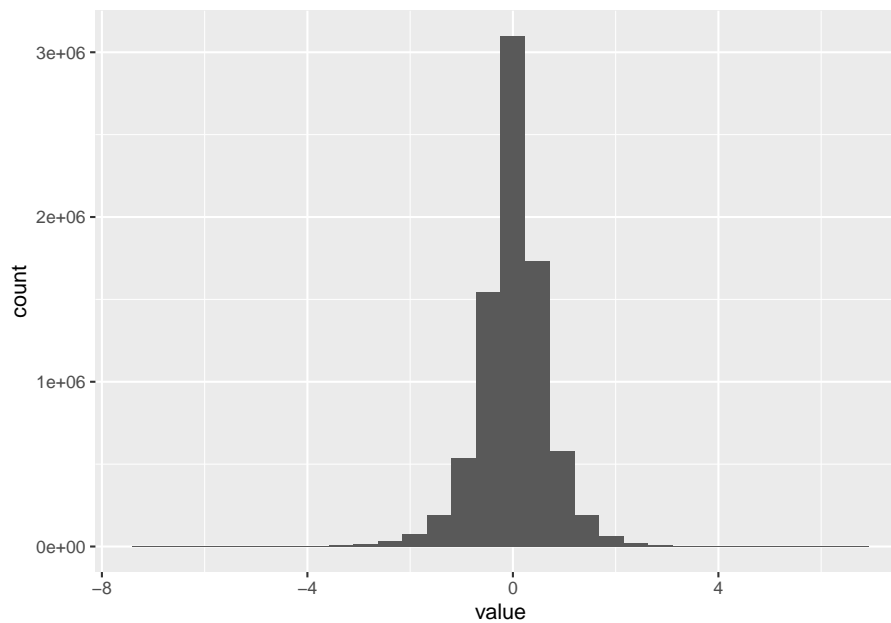


Bu sekilde, hangi bireyin veya genin verisi olduguna bakmaksizin tum degerlerin bir histogramini aliriz. Bu histogram cogu gen ekspresyon verisinde alacaginiz ham veri grafiginden cok farkli, bunun sebebi bunun two channel microarray olmasi ve loess normalizasyonu gibi kimi normalizasyon asamalarindan da gecirilmis olmasi.

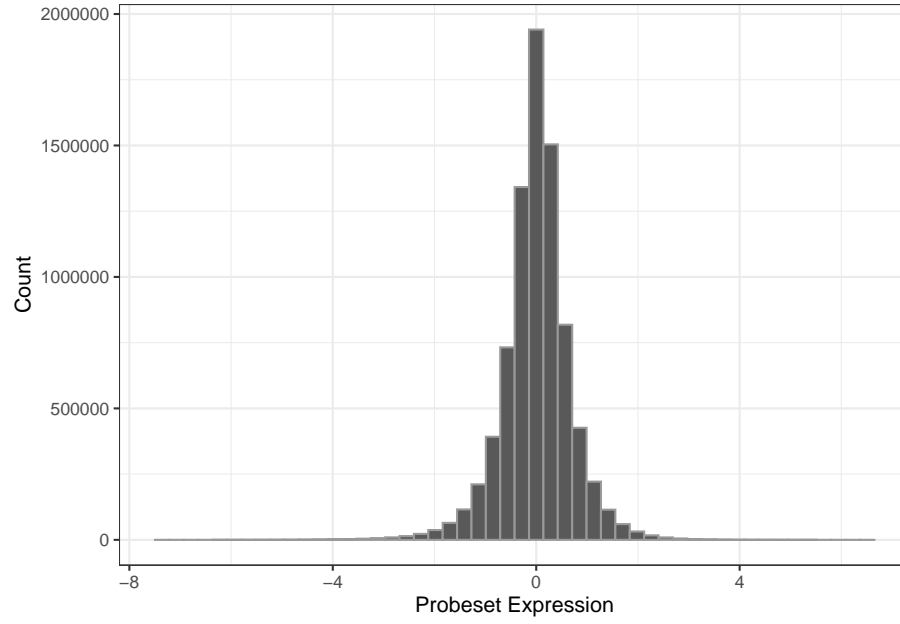
Buna bakmak oldukca kolaydi, ama ben genel olarak `ggplot2` paketinin gorsellestirmelerini kullaniyorum ve herkese en bastan itibaren `ggplot` ogrenmesini tavsiye ediyorum. Basit grafiklerde biraz zahmetli gelse de kompleks yayina hazir figur olusturmak icin cok ciddi kolaylik sagliyor. `ggplot2` `data.frame` objesiyle calisir, oysa bu bir matris. Bu yuzden ufak bir degisiklik yapmamiz lazim:

```
library(ggplot2)
mydat = reshape2::melt(expr)
ggplot(mydat, aes(x = value)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



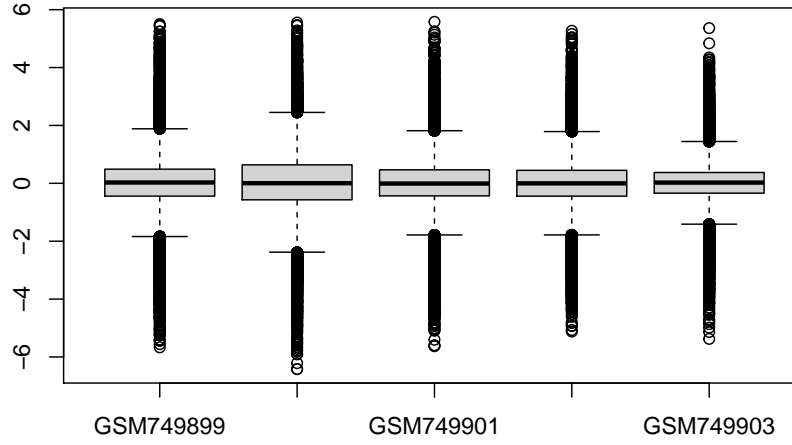
```
ggplot(mydat, aes(x = value)) +
  geom_histogram(color = 'gray60', bins = 50) +
  xlab('Probeset Expression') +
  ylab('Count') +
  theme_bw()
```



5.2.2 Boxplot

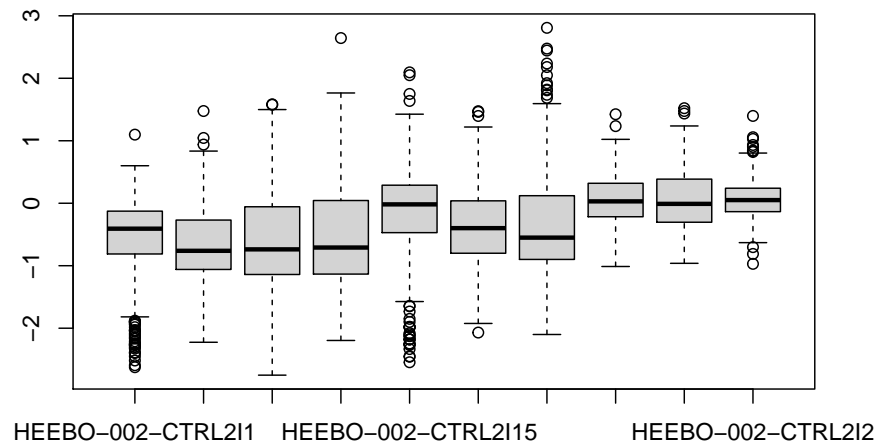
Simdi de ilk birkac bireyin gen anlatim profillerine bakalim. Ilk once base R gorsellestirme fonksiyonu ile:

```
boxplot(expr[,1:5])
```

Aynı grafiği ilk bir kaç gen için çizmek isteyecek olsak, hem matrisi subset etmemiz hem de transpoze etmemiz gerekir. Bunu `t()` fonksiyonu ile kolaylıkla yapabiliriz:

```
boxplot(t(expr[1:10,]))
```

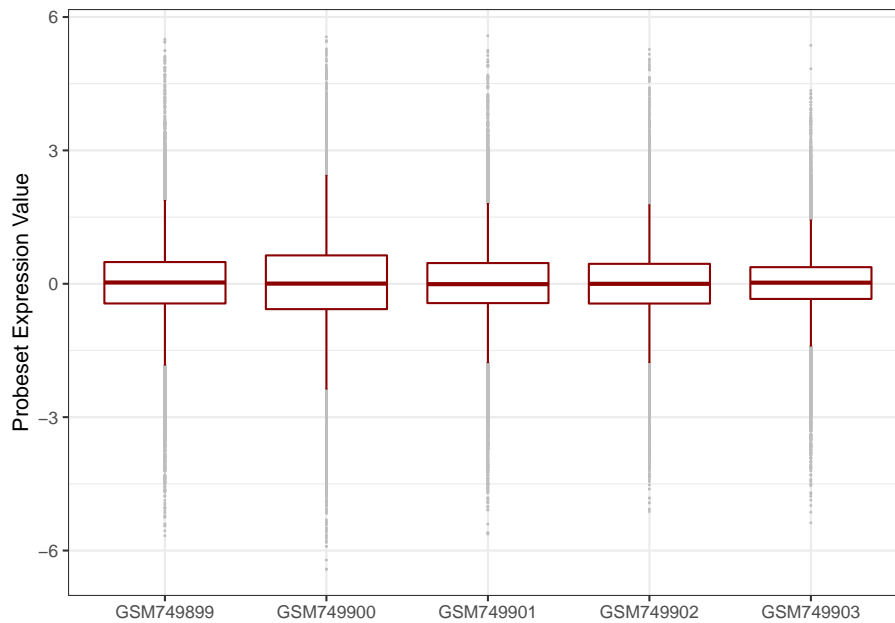


Aynisini ggplot ile yapalım:

```
mydat = reshape2::melt(expr[,1:5])
head(mydat)
```

```
##           Var1      Var2    value
## 1 HEEDO-002-CTRL2I1 GSM749899 -1.455124
## 2 HEEDO-002-CTRL2I10 GSM749899  1.044670
## 3 HEEDO-002-CTRL2I11 GSM749899  1.164344
## 4 HEEDO-002-CTRL2I14 GSM749899  1.597227
## 5 HEEDO-002-CTRL2I15 GSM749899 -1.454337
## 6 HEEDO-002-CTRL2I16 GSM749899  1.396667
```

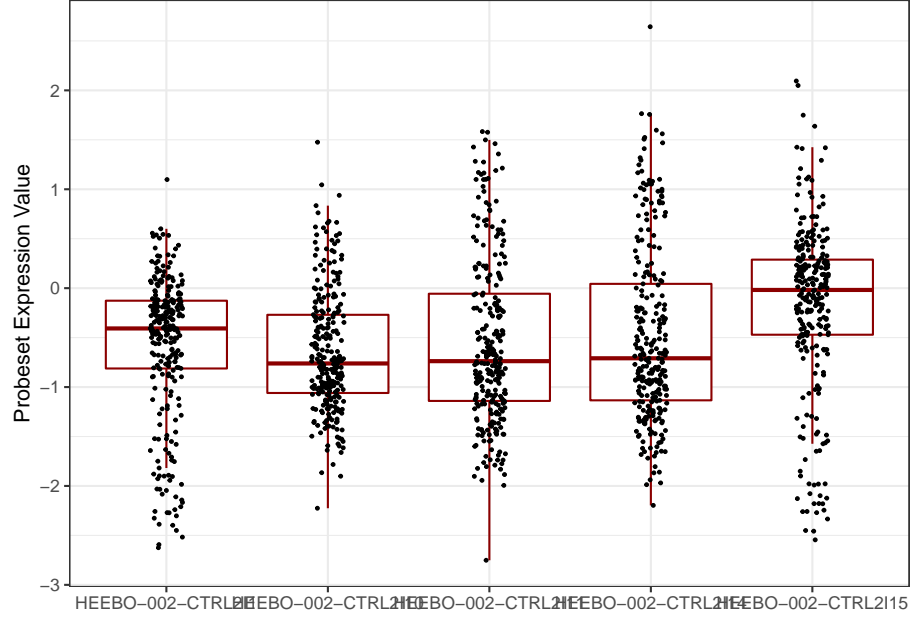
```
ggplot(mydat, aes(x = Var2, y = value)) +
  geom_boxplot(outlier.size = 0.01, outlier.color = 'gray', color = 'darkred') +
  theme_bw() +
  xlab(NULL) + ylab('Probeset Expression Value')
```



```
mydat = reshape2::melt(expr[1:5,])
head(mydat)
```

```
##           Var1      Var2    value
## 1 HEEBO-002-CTRL2I1 GSM749899 -1.455124
## 2 HEEBO-002-CTRL2I10 GSM749899  1.044670
## 3 HEEBO-002-CTRL2I11 GSM749899  1.164344
## 4 HEEBO-002-CTRL2I14 GSM749899  1.597227
## 5 HEEBO-002-CTRL2I15 GSM749899 -1.454337
## 6 HEEBO-002-CTRL2I1  GSM749900 -2.624420
```

```
ggplot(mydat,aes(x = Var1, y = value)) +
  geom_boxplot(outlier.shape=NA, color = 'darkred') +
  geom_jitter(size = 0.5, width = 0.1) +
  theme_bw() +
  xlab(NULL) + ylab('Probeset Expression Value')
```



5.3 ProbesetID - Gen eslestirmesi

Aynı gen ifade verisi gibi probesetID - gen eşlesmesi bilgisinin kayıtlı olduğu bir tablo daha kaydetmistim. Şimdi onu okuyalım:

```
genedata = read.csv('https://github.com/mdonertas/20210322_REgitim/raw/master/data/feat')
genedata[1:3,]
```

```
##           X           ID   OligoID
## 1 HEEBO-002-CTRL2I1 HEEBO-002-CTRL2I1 hCT000577
## 2 HEEBO-002-CTRL2I10 HEEBO-002-CTRL2I10 hCT000586
## 3 HEEBO-002-CTRL2I11 HEEBO-002-CTRL2I11 hCT000587
##
##                                     SEQUENCE
## 1 ACTCTGGATCCTGGGCGACGTCTTCATCGGCCGCTACTACACTGTGTTTGACCGTGACAACAACAGGGTG
## 2 CATCCTCACCGACATCACCAAGGGTGTGCAGTACCTCAACGAGATCAAGGACAGTGTGGTGGCCGGCTTC
## 3 GTCCCTGGAACGCCAGATGCGTGAAATGGAAGAGAACTTTGCCGTTGAAGCTGCTAACTACCAAGACACT
##
##      GB_ACC Strand Gene_Symbol      Gene_Title
## 1 NM_001909.4      +         CTSD      cathepsin D
## 2 NM_001961.3      +       EEF2 eukaryotic translation elongation factor 2
## 3 NM_003380.3      +         VIM      vimentin
##
##      Entrez_Gene_ID UniGene_Cluster_ID Feb2012_Blast_Cutoffs SPOT_ID
## 1          1509          Hs.654447 100/100_Unique_RefSeq
## 2          1938          Hs.515070 100/100_Unique_RefSeq
```

```
## 3          7431          Hs.455493 100/100_Unique_RefSeq
```

```
class(genedata)
```

```
## [1] "data.frame"
```

İlk olarak gen bilgisini düzenleyelim. Bunun için bu data.frame'den iki sütunu kullanacağız.

```
genedata$ID[1:10]
```

```
## [1] "HEEB0-002-CTRL2I1" "HEEB0-002-CTRL2I10" "HEEB0-002-CTRL2I11"
## [4] "HEEB0-002-CTRL2I14" "HEEB0-002-CTRL2I15" "HEEB0-002-CTRL2I16"
## [7] "HEEB0-002-CTRL2I17" "HEEB0-002-CTRL2I18" "HEEB0-002-CTRL2I19"
## [10] "HEEB0-002-CTRL2I2"
```

```
genedata$Gene_Symbol[1:10]
```

```
## [1] "CTSD" "EEF2" "VIM" "VIM" "GSN" "EEF2" "VIM" "CTSD" "RPN2" "RPN2"
```

Şimdi isimlendirilmiş bir vektör oluşturacağım ki, belli bir probe IDye karşılık gelen geni bulabileyim. Bunu yapmak için `setNames()` fonksiyonunu kullanacağım. İsimler, yani ikinci argüman, probeID olacak, vektörün kendisi ise genler, bu sayede vektörü belli bir probe ID kullanarak subset ettiğimde hangi gen olduğunu bulabilirim. Buna geçmeden bir örnek:

```
x = setNames(c(1:3),c('bir','iki','uc'))
x
```

```
## bir iki uc
## 1 2 3
```

ikinci elemanı almak için:

```
x[2]
```

```
## iki
## 2
```

```
x['iki']
```

```
## iki
## 2
```

Bunların ikisini de kullanabiliyorum. ID'leri kullanarak genleri bulmam da aynı şekilde olacak:

```
genemap = setNames(genedata$Gene_Symbol, genedata$ID)
genemap[1:5]
```

```
## HEEBO-002-CTRL2I1 HEEBO-002-CTRL2I10 HEEBO-002-CTRL2I11 HEEBO-002-CTRL2I14
##          "CTSD"          "EEF2"          "VIM"          "VIM"
## HEEBO-002-CTRL2I15
##          "GSN"
```

```
head(genemap)
```

```
## HEEBO-002-CTRL2I1 HEEBO-002-CTRL2I10 HEEBO-002-CTRL2I11 HEEBO-002-CTRL2I14
##          "CTSD"          "EEF2"          "VIM"          "VIM"
## HEEBO-002-CTRL2I15 HEEBO-002-CTRL2I16
##          "GSN"          "EEF2"
```

Matrisin satır isimleriyle genemap'i subset ettğimiz zaman, doğru sırayla hangi genlere karşılık geliyorsa bize bu bilgiyi verecek. Şimdi önce bir yanlış yapma ihtimalimize karşı matrisimizi kopyalayalım:

```
expr_yedek = expr
```

Şimdi ilk olarak bir probeset ID birden fazla genle eşlesiyor mu buna bakmalıyız, bunlar veri duplikasyonu yaratacağından istatistiksel test yapmamızı zorlaştırır bunları veriden çıkararak başlayacağız:

```
any(duplicated(names(genemap)))
```

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

Hic tekrar eden bir probesetID'miz yok - yani bu veri seti için problem değil. Bir de çok fazla sayıda probesetID'ye map olmuş olan genlere bakalım:

```
sort(table(genemap),dec=T)[1:10]
```

```
## genemap
## ##noname##          LRP1      OCRL      TLN1      TTN      CDH23
##      903      825      88      62      55      55      41
##      ANXA7      RB1      DNAH9
##      40      40      37
```

Bunlardan '##noname##' ve '' tabii ki gen ismi değil. Bunları veriden çıkartmak gerekir, ama bunu matrice isimleri geçirdikten sonra yapacağım. Şimdi probeset ID'leri gen isimleriyle değiştirerek devam edelim:

```
rownames(expr) = genemap[rownames(expr)]
```

```
expr[1:5,1:5]
```

```
##      GSM749899  GSM749900  GSM749901  GSM749902  GSM749903
## CTSD -1.455124 -2.6244200 -1.904436 -1.5793507 -2.1666262
## EEF2  1.044670  0.4490985  0.619243  0.1930628 -0.4370544
## VIM   1.164344  1.5833201  1.150024  1.5764080  1.3569308
## VIM   1.597227  1.4105770  1.469449  2.6433802  1.7567335
## GSN   -1.454337 -1.9784923 -1.485409 -1.5038039 -1.2828650
```

```
sort(table(rownames(expr)), dec = T)[1:10]
```

```
##
## ##noname##          LRP1      OCRL      TLN1      TTN      CDH23
##      903      825      88      62      55      55      41
##      ANXA7      RB1      DNAH9
##      40      40      37
```

Burada gördüğünüz gibi bazı gen isimleri birden fazla kez tekrar ediliyor. Bu mikrodizin dizaynından kaynaklı bir durum, her gen bir ya da daha fazla probeset ile temsil ediliyor. Genelde her gen bu probesetlerin ortalaması ya da maksimumu alınarak özetlenir - bu sayede her gen matriste bir defa yer alır. Bunu yapmak için matrisi gen isimleriyle subset etmeyi öğreneceğiz. Zaten belli bir isimle subset yapmayı biliyoruz, ancak bu sadece tekrarlı olmayan durumlarda geçerli. Suna bakalım:

```
x = setNames(c(1:5), c('bir','iki','iki','dört','bes'))
x['iki']
```

```
## iki
## 2
```

oysa 3 degerine sahip olan elemanın adı da ‘iki’. Bu tarz tekrarlı durumlarda otomatik olarak ilk eleman verilir. İkisini birden istiyorsak subset işlemini `%in%` operatörü ile yapmamız gerekir.

5.3.1 `%in%` operatörü

```
'elma' %in% c('elma','armut')
```

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

```
'muz' %in% c('elma','armut')
```

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

```
c('elma', 'muz') %in% 'elma'
```

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

```
c('elma', 'muz') %in% 'muz'
```

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

Bilmemiz gereken bir konu da bir vektörü ya da matrisi mantıksal değişkenlerle de subset edebildiğimiz:

```
c(1:5)[c(T,T,F,T,F)]
```

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

İlk olarak gen ismi olarak `'##noname##'` ve `' '` görünen satırları veriden çıkartarak baslamalıyız:

```
expr = expr[!rownames(expr) %in% c('##noname##',' '),]
```



```
sort(table(rownames(expr)),dec=T)[1:10]
```

```
##
## LRP1 OCRL TLN1 TTN CDH23 ANXA7 RB1 DNAH9 WDFY3 SYNE2
## 88 62 55 55 41 40 40 37 37 36
```

Simdi tekrar eden genleri ozetlemeye geri donelim.

```
expr[rownames(expr) %in% 'VIM', 1:5]
```

```
## GSM749899 GSM749900 GSM749901 GSM749902 GSM749903
## VIM 1.164344 1.583320 1.150024 1.576408 1.356931
## VIM 1.597227 1.410577 1.469449 2.643380 1.756734
## VIM 2.475041 2.236114 2.179526 2.807048 2.443889
## VIM 2.414010 2.242931 2.379348 1.909755 2.596801
## VIM 2.913906 1.343048 2.536056 2.271887 2.118884
## VIM 1.743049 1.492585 1.621053 1.394594 2.030154
## VIM 1.171174 1.165681 1.099818 2.274270 1.541334
## VIM 1.446696 1.451936 1.782307 2.605349 1.957581
## VIM 1.131674 1.172329 1.053107 1.871871 1.780982
```

Simdi yeni bir fonksiyon ogrenelim `colMeans()` bu da adi ustunde sutun ortalamasini aliyor. Yani matrisi tum VIM genine karsilik gelen probesetler icin subset edip, `colMeans()` dersek tek boyutlu, satirlarin ortalamasina sahip bir vektor verecek:

```
vimmeans = colMeans(expr[rownames(expr) %in% 'VIM', ])
head(vimmeans)
```

```
## GSM749899 GSM749900 GSM749901 GSM749902 GSM749903 GSM749904
## 1.784125 1.566502 1.696743 2.150507 1.953699 1.409357
```

Ama bu sadece tek bir gen icin. Oysa biz tek genle ilgilenmiyoruz, butun genler icin bunu yapmak istiyoruz. Eger programlama dillerine asinaysaniz `for` loop larini duymussunuzdur. Bunlar ayni islemin defalarca ayni ya da farkli inputlar icin tekrarlanmasini saglar. R'da da `for` loop yazabiliriz bunun icin kullanılacak fonksiyon `for`. Bizse `for` yerine R'a ozgu `*apply` fonksiyonlari olarak gecen fonksiyonlardan biri olan `sapply()` fonksiyonu ile bunu yapacagiz. Ilk olarak butun gen isimlerini almaliyiz. Bunu sadece `rownames()` ile alamiyoruz cunku isimler tekrar ediyor. Onun yerine her degerin sadece bir kere tekrar edilmesini `unique()` fonksiyonu ile saglayacagiz.

```
genisimleri = unique(rownames(expr))
nrow(expr)
```

```
## [1] 28448
```

```
length(genisimleri)
```

```
## [1] 17160
```

Sadece 17bin gen var. Yani her gen çok defa tekrar ediliyor olmak zorunda değil. Eğer tekrar edilmeyen bir geni subset edersek, ‘sütun ortalamalarını ver’ demek için kullanacağımız `colMeans` fonksiyonu çalışmaz, çünkü bir matris değil bir vektör söz konusu olacak. Bunun için bir de `if` fonksiyonunu öğrenmeliyiz. `if` bir kodu ancak belli koşullar altında çalıştıracaktır. Bu sayede eğer bir defa geçiyorsa bunu yap, birden fazla kez tekrar ediliyorsa bunu yap şeklinde bir kod yazmamız mümkün olacak. `If`’e çok kısa bir örnek:

```
vektorum = c('elma','muz','armut')
if('cilek'%in%vektorum){
  print('cilek var!')
} else if('elma' %in% vektorum){
  print('elma var!')
} else{
  print('ne cilek ne elma var :(')
}
```

```
## [1] "elma var!"
```

daha iyi anlayabilmemiz için bir örnek daha:

```
vektorum = c('elma','muz','armut')
if('muz'%in%vektorum){
  print('muz var!')
} else if('elma' %in% vektorum){
  print('elma var!')
} else{
  print('ne cilek ne elma var :(')
}
```

```
## [1] "muz var!"
```

hem elma hem muz olmasına rağmen sadece birinci komut çalıştı. Bunun sebebi `else` komutu ancak ilk kodumuz yanlışsa çalışacak olması.

Şimdi de çok kısa `sapply()` nasıl çalışıyor buna bakalım:

```
sapply(c(1:10),function(i){
  i * 2
})
```

```
## [1] 2 4 6 8 10 12 14 16 18 20
```

```
sapply(c(1:10),function(i){
  i * (i-1)
})
```

```
## [1] 0 2 6 12 20 30 42 56 72 90
```

Şimdi biraz ileri düzey gibi görünmesi muhtemel bir kod yazacağız ama basamak basamak inceleyince anlaşılması kolay olacaktır:

```
genexpr = sapply(genisimleri, function(gen) {
  i = rownames(expr)%in%gen
  if(sum(i)>1){
    colMeans(expr[i,])
  } else if(sum(i)==1){
    expr[i,]
  }
})
```

```
genexpr[1:5,1:5]
```

```
##           CTSD      EE2      VIM      GSN      RPN2
## GSM749899 -0.8124619 1.5620237 1.784125 -0.6578933 1.658038
## GSM749900 -1.3310489 1.4339156 1.566502 -1.1007844 1.528385
## GSM749901 -1.2447029 0.9832866 1.696743 -1.0306526 1.286318
## GSM749902 -1.2017866 0.7393814 2.150507 -0.8413292 1.153392
## GSM749903 -1.1041998 0.2957918 1.953699 -0.7242716 0.789019
```

satir-sutun yer degistirmis durumda. bunu yine transpoze ederek duzeltebiliriz:

```
genexpr = t(genexpr)
```

```
genexpr[1:5,1:5]
```

```
##          GSM749899 GSM749900 GSM749901 GSM749902 GSM749903
## CTSD -0.8124619 -1.331049 -1.2447029 -1.2017866 -1.1041998
## EEF2  1.5620237  1.433916  0.9832866  0.7393814  0.2957918
## VIM   1.7841245  1.566502  1.6967431  2.1505070  1.9536987
## GSN  -0.6578933 -1.100784 -1.0306526 -0.8413292 -0.7242716
## RPN2  1.6580375  1.528385  1.2863177  1.1533917  0.7890190
```

```
dim(genexpr)
```

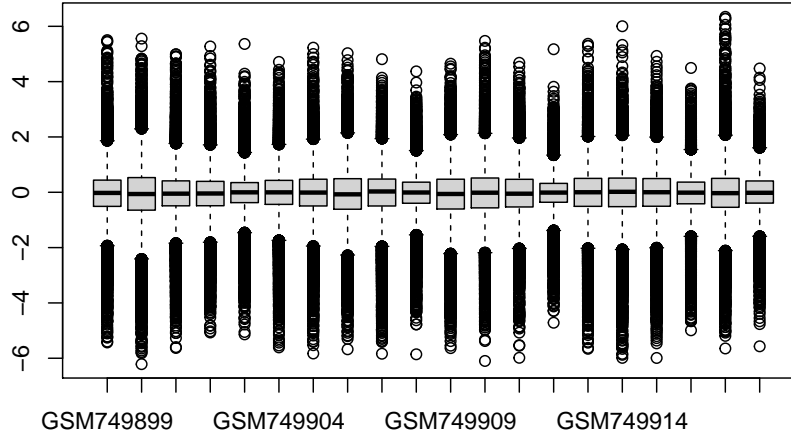
```
## [1] 17160 269
```

Artık her gen için bir satirimiz var!

5.4 Quantile Normalization

Son olarak bu gen ifade matrixinde bir normalizasyon metodu uygulayalım. Burada amacımız farklı örnekler arasındaki olası teknik varyasyonu ortadan kaldırmak. Uygulayacağımız normalizasyonun adı ‘quantile normalization’ - bunun sonunda tüm örneklerin dağılımları aynı olacak - genler farklı değerler alacak ama genlerin dağılımı aynı olacak. Normalizasyon öncesi ilk bir kaç bireyin dağılımına bakalım.

```
boxplot(genexpr[,1:20])
```



Gordugunuz gibi ortalamalari hemen hemen ayni olsa da (bunun sebebi verinin zaten baska bir yontemle kısmen normalize edilmiş olması), Dağılımlar farklı.

Normalizasyon için Rda olmayan bir fonksiyon kullanacağız, bunun için de `preprocessCore` paketini yüklememiz gerekiyor:

```
BiocManager::install("preprocessCore")
```

```
library(preprocessCore)
```

Fonksiyonumuzun adı `normalize.quantiles()`

```
genexpr_qn = normalize.quantiles(genexpr)
dim(genexpr_qn)
```

```
## [1] 17160 269
```

```
genexpr_qn[1:5,1:5]
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -0.4993920 -0.6776740 -0.7796459 -0.8040486 -0.8587960
## [2,]  1.0141904  0.8355595  0.6704056  0.5563232  0.2924776
## [3,]  1.1413791  0.9140343  1.1026208  1.4685350  1.5103426
## [4,] -0.4112961 -0.5602207 -0.6620208 -0.5704519 -0.6015157
## [5,]  1.0659865  0.8908564  0.8555507  0.8361596  0.6749602
```

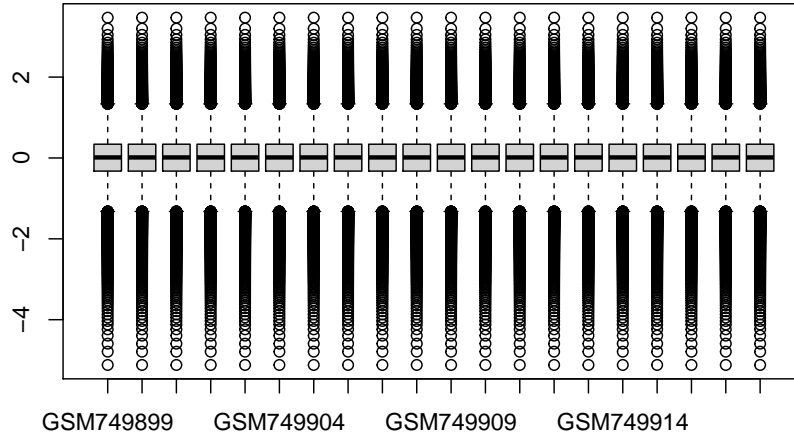
Gordugunuz gibi bu fonksiyon ne yazik ki satir sutun isimlerimizi sildi. Ama bunlarin ilk matrisimizle ayni oldugunu biliyoruz cunku bir subset islemi yapmadik. Eski satir-sutun isimlerimizi yeni matrisimize aktaralim:

```
dimnames(genexpr_qn) = dimnames(genexpr)
genexpr_qn[1:5,1:5]
```

```
##          GSM749899  GSM749900  GSM749901  GSM749902  GSM749903
## CTSD -0.4993920 -0.6776740 -0.7796459 -0.8040486 -0.8587960
## EEF2  1.0141904  0.8355595  0.6704056  0.5563232  0.2924776
## VIM   1.1413791  0.9140343  1.1026208  1.4685350  1.5103426
## GSN   -0.4112961 -0.5602207 -0.6620208 -0.5704519 -0.6015157
## RPN2  1.0659865  0.8908564  0.8555507  0.8361596  0.6749602
```

Tekrar dagilimlara bakacak olursak:

```
boxplot(genexpr_qn[,1:20])
```



Gordugunuz gibi dagilimlar birebir esit oldu. Bundan sonraki asamalarda bu veriyi kullanacagiz.

5.5 Ornek bilgisini duzenleme

Son olarak, bir de orneklerle ilgili bilgilerin yer aldigi (yas, biyolojik cinsiyet) bir tablomuz var. Ama bu verinin de duzenlenmesi gerekiyor.

```
metadata = read.csv('https://github.com/mdonertas/20210322_REgitim/raw/master/data/metadata.csv')
head(metadata)
```

```
##           X      title geo_accession characteristics_ch1 characteristics_ch1.1
## 1 GSM749899 HB_18_34    GSM749899    array batch: 10    age: -0.498630137
## 2 GSM749900 HB_22_35    GSM749900    array batch: 18    age: -0.498630137
## 3 GSM749901 HB_16_29    GSM749901    array batch: 1     age: -0.498630137
## 4 GSM749902 HB_17_91    GSM749902    array batch: 8     age: -0.498630137
## 5 GSM749903 HB_18_16    GSM749903    array batch: 9     age: -0.479452055
## 6 GSM749904 HB_16_62    GSM749904    array batch: 3     age: -0.479452055
## characteristics_ch1.2 characteristics_ch1.3 characteristics_ch1.4
## 1                      Sex: F                race: AA postmortem interval (pmi): 3
## 2                      Sex: F                race: AA postmortem interval (pmi): 1
## 3                      Sex: M                race: AA postmortem interval (pmi): 1
## 4                      Sex: M                race: AA postmortem interval (pmi): 4
## 5                      Sex: M                race: AA postmortem interval (pmi): 2
## 6                      Sex: M                race: AA postmortem interval (pmi): 4
## characteristics_ch1.5 characteristics_ch1.6 characteristics_ch1.7
## 1                      ph: NULL rna integrity number (rin): 9.7    smoke at death: No
## 2                      ph: 6.21 rna integrity number (rin): 9.8    smoke at death: No
## 3                      ph: NULL rna integrity number (rin): 9.6    smoke at death: No
## 4                      ph: NULL rna integrity number (rin): 9.8    smoke at death: No
## 5                      ph: NULL   rna integrity number (rin): 9    smoke at death: Unknown
## 6                      ph: NULL rna integrity number (rin): 9.8    smoke at death: No
```

Bu veriyi kullanarak, ayni probeset-gen eslestirmesi icin yaptigimiz gibi yas ve biyolojik cinsiyet icin birey bilgisi vektörü olusturalim:

```
yas = setNames(metadata$characteristics_ch1.1, metadata$geo_accession)
head(yas)
```

```
##           GSM749899          GSM749900          GSM749901          GSM749902
## "age: -0.498630137" "age: -0.498630137" "age: -0.498630137" "age: -0.498630137"
##           GSM749903          GSM749904
## "age: -0.479452055" "age: -0.479452055"
```

```
class(yas)
```

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

```
cinsiyet = setNames(metadata$characteristics_ch1.2, metadata$geo_accession)
head(cinsiyet)
```

```
## GSM749899 GSM749900 GSM749901 GSM749902 GSM749903 GSM749904
## "Sex: F" "Sex: F" "Sex: M" "Sex: M" "Sex: M" "Sex: M"
```

```
class(cinsiyet)
```

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

Bu veriler gordugunuz gibi karakter verileri ve formatlari pek iyi degil. Ornegin bu yas verisiyle yapabileceklerimiz sinirli cunku numerik degil. Ilk olarak basta yazan ‘age:’ kisimindan kurtulmamiz lazim ki bu yas verisini kullanabilelim. Ayni sekilde biyolojik cinsiyeti de ‘F’ veya ‘M’ haline, hatta daha iyisi ‘kadin’ ve ‘erkek’ haline getirelim.

5.5.1 Yas verisinin duzenlenmesi

Basta yazan ‘age:’ yazisinin ortadan kaldirilmasi icin bir kac secenek var. Bunlardan en basiti gsub fonksiyonu. Bu fonksiyon, bir karakter vektorunde belli karakterlerin degistirilmesini saglar:

```
gsub('range', '', 'Bioinforange')
```

```
## [1] "Bioinfo"
```

```
gsub('range', 'conference', 'Bioinforange')
```

```
## [1] "Bioinfoconference"
```

Bu sekilde ‘age:’ yazisini silebiliriz:

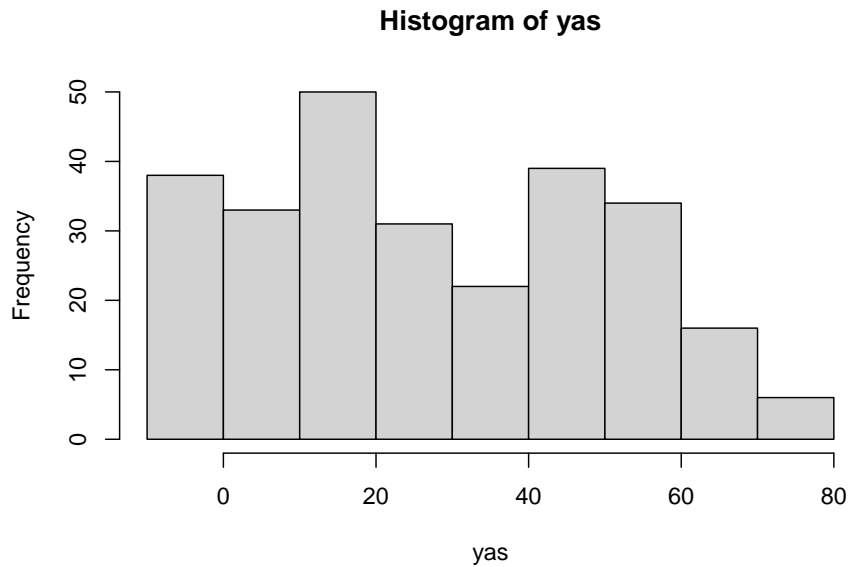
```
yas = gsub('age: ', '', yas)
head(yas)
```

```
##      GSM749899      GSM749900      GSM749901      GSM749902      GSM749903
## "-0.498630137" "-0.498630137" "-0.498630137" "-0.498630137" "-0.479452055"
##      GSM749904
## "-0.479452055"
```

Ancak gordugunuz gibi, veri hala karakter. Bunu numerik yapmak icin as.numeric() kullaniriz:


```
yas = setNames(as.numeric(yas),names(yas))
```

```
hist(yas)
```



Daha net bir sekilde verinin ozetini gormek istersek `summary()` fonksiyonunu kullanabiliriz:

```
summary(yas)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4986  5.3452 24.2658 27.7920 46.9671 78.2274
```

Dogum oncesi zamandan 78 yasina kadar bireyler var ve ortalamamız 27.7.

5.5.2 Biyolojik cinsiyet verisinin duzenlenmesi

Bu sefer sadece 2 biyolojik cinsiyet mumkun oldugundan daha farkli bir yol izleyebiliriz. Ama oncelikle, sadece iki secenek oldugundan emin olalim.

```
unique(cinsiyet)
```

```
## [1] "Sex: F" "Sex: M" "Sex: 5"
```

Bir de '5' var, bu muhtemelen veri girişinde bir hata demek ama bu bireyi en azından cinsiyet için inceleyemeyeceğimiz anlamına geliyor. İlk olarak bu veriye sahip olan bireyi hem bu veriden hem expression verisinden hem de yas verisinden çıkaralım.

```
cikart = names(cinsiyet[(cinsiyet %in% 'Sex: 5')])
cinsiyet = cinsiyet[!names(cinsiyet) %in% cikart]
yas = yas[!names(yas) %in% cikart]
genexpr_qn = genexpr_qn[,!colnames(genexpr_qn) %in% cikart]
```

Aynı probeset-gen eşleşmesi yapmak için kullandığımız gibi bir vektor oluşturalım:

```
cinsiyetMap = setNames(c('Kadin','Erkek'),c('Sex: F','Sex: M'))
cinsiyet = setNames(cinsiyetMap[cinsiyet],names(cinsiyet))
head(cinsiyet)
```

```
## GSM749899 GSM749900 GSM749901 GSM749902 GSM749903 GSM749904
## "Kadin" "Kadin" "Erkek" "Erkek" "Erkek" "Erkek"
```

Kac kadin kac erkek bireyin olduguna bakmak için `table()` fonksiyonunu kullanabiliriz:

```
table(cinsiyet)
```

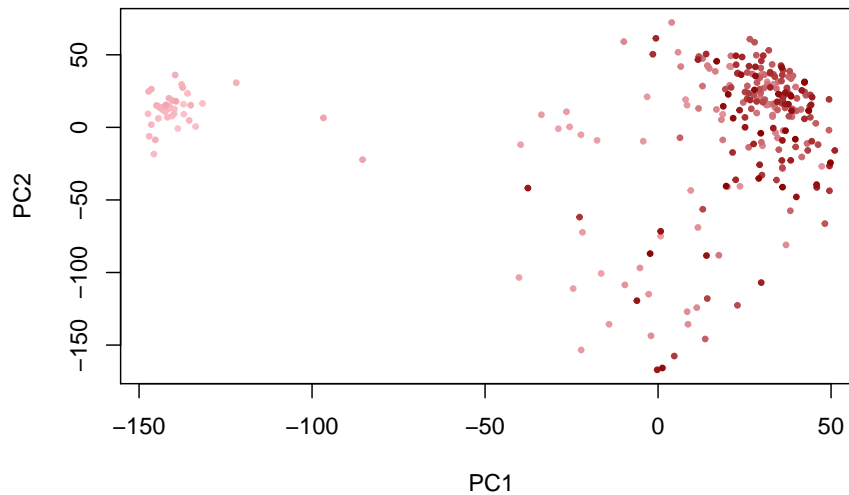
```
## cinsiyet
## Erkek Kadin
## 177 91
```

Chapter 6

Temel bileşenler analizi

```
tba = prcomp(t(genexpr_qn), scale. = T)

renkler = colorRampPalette(c('pink', 'darkred'))(length(yas))
renkler_sirali = renkler[rank(yas[rownames(tba$x)], ties.method = 'min')]
plot(x = tba$x[,1], y = tba$x[,2], pch = 19, cex = 0.5, col = renkler_sirali,
     xlab = 'PC1', ylab = 'PC2')
```

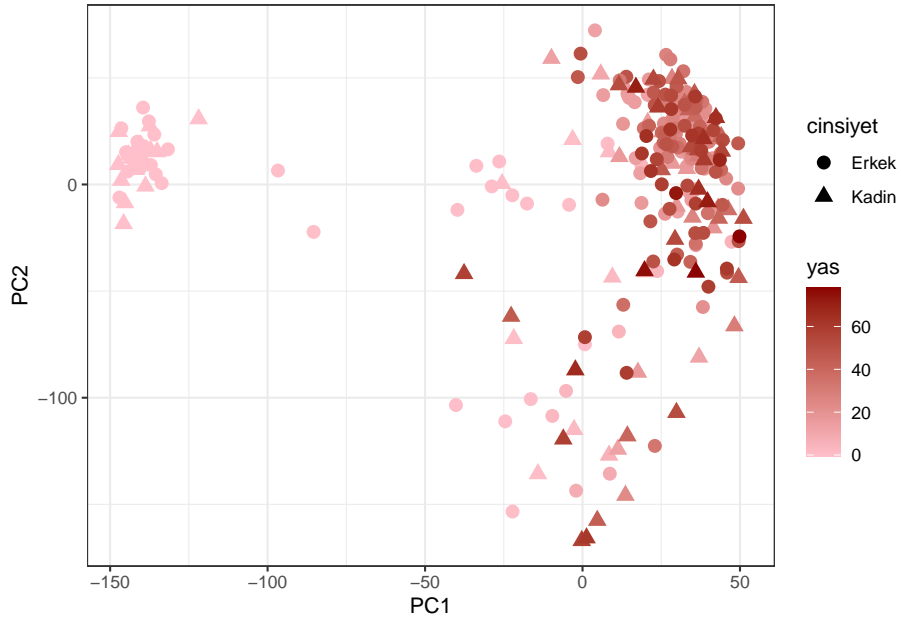


Aynı figuru ggplot ile yapabiliriz. Ancak bunun için bir kaç `tidyverse`de yer alan paketlerden bir kaç fonksiyon kullanarak veriyi şekillendirmem gerekiyor:

```
library(tidyverse)
pcadat = as.data.frame(tba$x[,1:2]) %>%
  mutate(sample = rownames(tba$x)) %>%
  mutate(yas = yas[sample],
         cinsiyet = cinsiyet[sample])
head(pcadat)
```

```
##          PC1          PC2   sample      yas cinsiyet
## 1 -138.7854 -0.7978871 GSM749899 -0.4986301   Kadın
## 2 -137.1751 15.9124596 GSM749900 -0.4986301   Kadın
## 3 -142.7689 13.8844269 GSM749901 -0.4986301   Erkek
## 4 -137.0419  9.0980053 GSM749902 -0.4986301   Erkek
## 5 -133.6389  0.6007799 GSM749903 -0.4794521   Erkek
## 6 -131.6583 16.4774424 GSM749904 -0.4794521   Erkek
```

```
ggplot(pcadat, aes(x = PC1, y = PC2, color = yas, shape = cinsiyet)) +
  geom_point(size = 3) +
  scale_color_gradient(low = 'pink', high = 'darkred') +
  theme_bw()
```



Chapter 7

Yasla degisim analizi

Diyelim ki sadece dogum sonrasi donemde 20 sonrasi yasta yasla degisen genlerle ilgileniyoruz. Bunun icin ilk olarak veriyi subset etmeliyiz. Yas'in 20den büyük olduğu bireyleri almaliyiz:

```
subsamp = names(which(yas>=20))
subexp = genexpr_qn[,subsamp]
subyas = yas[subsamp]
```

```
yasladegisim = apply(subexp, 1, function(x){
  co = cor.test(x, subyas[colnames(subexp)], method = 'spearman')
  c(co$est, co$p.val)
})
```

```
dim(yasladegisim)
```

```
## [1]      2 17160
```

```
yasladegisim = t(yasladegisim)
head(yasladegisim)
```

```
##           rho
## CTSD  -0.03253336 0.6956624723
## EE2F2  -0.12309924 0.1374343793
## VIM    0.28300932 0.0005395356
## GSN    0.11697104 0.1581010511
## RPN2   -0.22977129 0.0052073498
## POLR2A 0.01173037 0.8878589579
```

```
colnames(yasladegisim) = c('rho','p')
yasladegisim = cbind(yasladegisim, p.adjust(yasladegisim[, 'p'], method= 'fdr'))
```

Hangi genler yaslâ istatistiksel olarak anlamlı deęisim gösteriyor?

```
anlamli = names(which(yasladegisim[,3]<=0.05))
table(yasladegisim[rownames(yasladegisim)%in%anlamli,1]>=0)
```

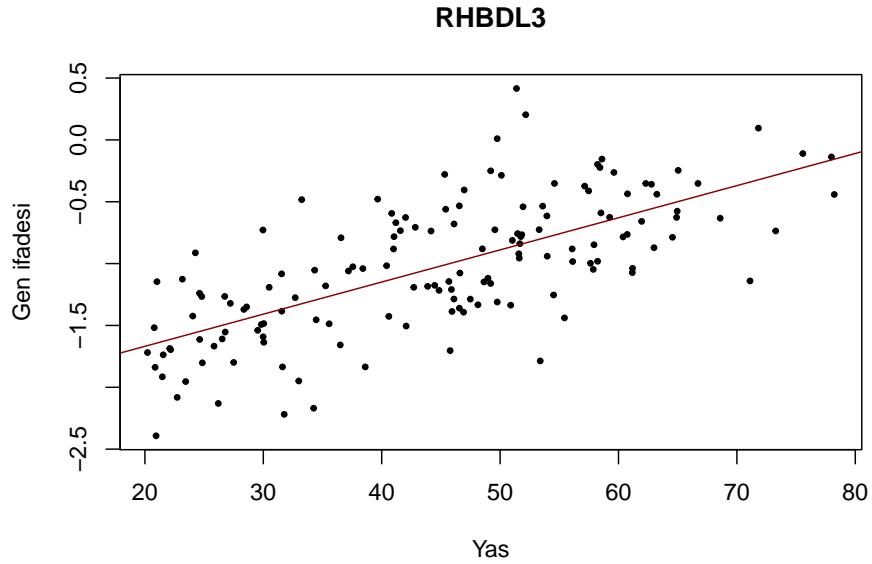
```
##
## FALSE TRUE
## 1538 1418
```

Bunlardan 1419 tanesi artis yonunde anlamlı deęisim gösteriyor, 1538 tanesi ise dusus. Bunlardan en cok artis gostereni secip gorsellestirelim:

```
maxartis = names(which.max(yasladegisim[,1]))
yasladegisim[maxartis,]
```

```
##          rho          p
## 6.980232e-01 8.761466e-23 1.002312e-19
```

```
plot(x = subyas, y = subexp[maxartis,], pch = 19, cex = 0.5, xlab='Yas',
      ylab = 'Gen ifadesi', main = maxartis)
abline(lm(subexp[maxartis,]~subyas), col = 'darkred', cex = 2)
```



```
pdf('./yasladegisim.pdf')
plot(x = subyas, y = subexp[maxartis,], pch = 19, cex = 0.5, xlab='Yas',
     ylab = 'Gen ifadesi', main = maxartis)
abline(lm(subexp[maxartis,]~subyas), col = 'darkred', cex = 2)
dev.off()
```

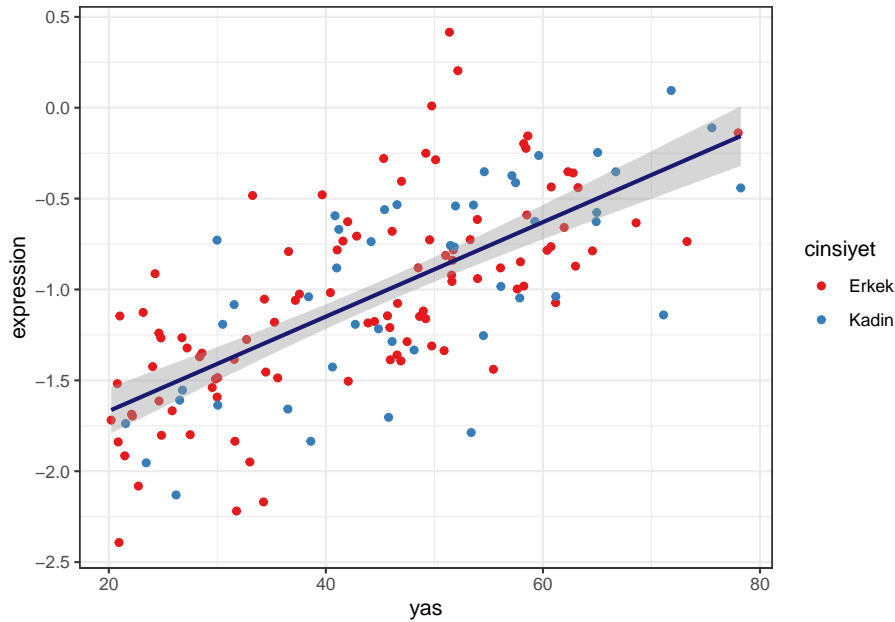
```
## pdf
## 2
```

```
write.csv(x = yasladegisim, file = './yasladegisim.csv', quote = F,
         row.names = T)
```

Aynı figuru ggplot ile çizelim:

```
mydat = data.frame(expression = subexp[maxartis,], sample = colnames(subexp),
                   yas = subyas[colnames(subexp)], cinsiyet = cinsiyet[colnames(subexp)])
ggplot(mydat, aes(x = yas, y = expression)) +
  geom_point(aes(color = cinsiyet)) +
  geom_smooth(method = 'lm', color = 'midnightblue') +
  scale_color_brewer(type = 'qual', palette = 6) +
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Chapter 8

Biyolojik cinsiyete gore farklilik gosteren gen analizi

Diyelim ki sadece dogum sonrasi donemde 20 sonrasi ve 40 yas oncesi yasta kadin ve erkekler arasinda farkli gen ifadesi gosteren genlerle ilgileniyoruz.

```
subsamp = names(which(yas>=20 & yas<40))
subexp = genexpr_qn[,subsamp]
subyas = yas[subsamp]
subcins = cinsiyet[subsamp]
table(subcins)
```

```
## subcins
## Erkek Kadin
##      41     12
```

Kadin ve erkek sayisindaki fark cok fazla. Hem erkek hem de kadinlari icinden rastgele 10ar birey secip analiz yapalim (- rastgele secim yapmayi gostermek icin!).

```
erkek = sample(names(which(subcins == 'Erkek')),10,replace = F)
kadin = sample(names(which(subcins == 'Kadin')),10,replace = F)
subsamp = c(erkek,kadin)
subexp = genexpr_qn[,subsamp]
subyas = yas[subsamp]
subcins = cinsiyet[subsamp]
table(subcins)
```

```
## subcins
```

```
## Erkek Kadın
##      10      10
```

```
cinsiyetfark = apply(subexp, 1, function(x){
  tres = t.test(x~subcins)
  c(tres$statistic, tres$p.value)
})
```

```
dim(cinsiyetfark)
```

```
## [1]      2 17160
```

```
cinsiyetfark = t(cinsiyetfark)
head(cinsiyetfark)
```

```
##              t
## CTSD   -0.66266752 0.5224955
## EEF2    0.07559578 0.9406862
## VIM     0.51997912 0.6098727
## GSN     0.20729499 0.8381238
## RPN2    1.38597402 0.1828271
## POLR2A  0.73682091 0.4712232
```

```
colnames(cinsiyetfark) = c('t', 'p')
cinsiyetfark = cbind(cinsiyetfark, p.adjust(cinsiyetfark[, 'p'], method= 'fdr'))
```

Hangi genler yaşla istatistiksel olarak anlamlı değişim gösteriyor?

```
anlamli = names(which(cinsiyetfark[,3]<=0.05))
length(anlamli)
```

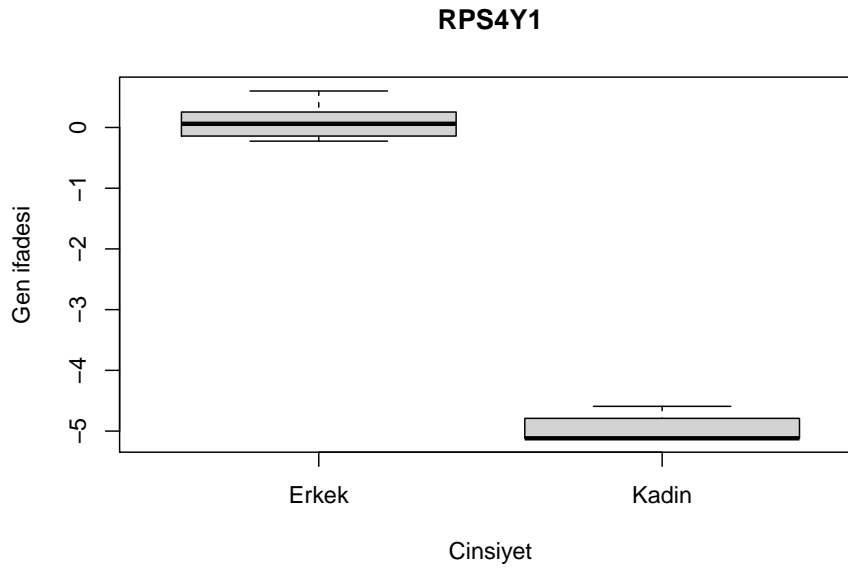
```
## [1] 12
```

Sadece 12 gen anlamlı fark göstermiş.

```
maxfark = names(which.max(abs(cinsiyetfark[,1])))
cinsiyetfark[maxfark,]
```

```
##              t              p
## 4.570803e+01 9.547559e-20 1.638361e-15
```

```
boxplot(subexp[maxfark,]~subcins, ylab = 'Gen ifadesi', main = maxfark,
        xlab = 'Cinsiyet')
```



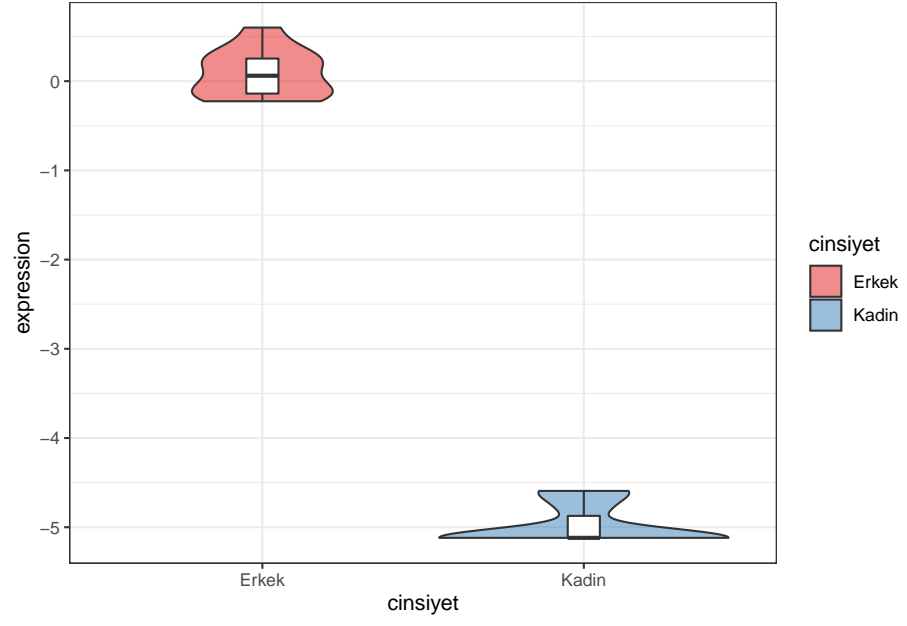
Figuru kaydetmek icin:

```
pdf('./cinsiyetfark.pdf')
boxplot(subexp[maxfark,]~subcins, ylab = 'Gen ifadesi', main = maxfark,
        xlab = 'Cinsiyet')
dev.off()
```

```
## pdf
## 2
```

Aynı figuru ggplotta cizelim:

```
mydat = data.frame(expression = subexp[maxfark,],
                   cinsiyet = subcins[colnames(subexp)],
                   yas = yas[colnames(subexp)])
ggplot(mydat, aes(x = cinsiyet, y = expression)) +
  geom_violin(aes(fill = cinsiyet), alpha = 0.5) +
  scale_fill_brewer(type='qual', palette = 6)+
  geom_boxplot(outlier.shape = NA, width = 0.1) +
  theme_bw()
```



Chapter 9

Kaynak önerileri

9.1 Online ders

- R ve veri bilimine giriş: <https://www.edx.org/professional-certificate/harvardx-data-science>
- R ve yaşam bilimlerinde veri analizi: <https://www.edx.org/professional-certificate/harvardx-data-analysis-for-life-sciences>
- Rda genomik analiz: <https://www.edx.org/professional-certificate/harvardx-data-analysis-for-genomics>
- R programlama: <https://www.coursera.org/learn/r-programming>
- Coursera Data Science Specialization içinde ~10 ders var: <https://www.coursera.org/specializations/jhu-data-science>

9.2 Kitaplar

- R Programming for Data Science: <https://leanpub.com/rprogramming>
- Data Analysis for the Life Sciences: <https://leanpub.com/dataanalysisforthelifesciences>
- Tidyverse öğrenmek için: <https://r4ds.had.co.nz/>
- ggplot'ta grafikler: <https://ggplot2-book.org/>

9.3 Diger

Zaman zaman R'da hem giris duzeyinde Turkce ders notlari hem de biyolojide kullanilan grafiklere ornekler paylastigim bir blogum var: <https://compbioscratchpad.com/bilgi-deposu/>

Kaynak onerileriniz olursa bana iletirseniz bu sayfayi guncelleye-bilirim.