Doğa Bilimlerinde R Programlamaya Giriş

Nurbahar Usta, İsmail Bekar

Gerekli Dökümanlar

https://bit.ly/dogaBR2022

Kaynaklar

- R for Data Science ARTIK TÜRKÇE!!!
- Advanced R

Katkı Sağlayanlar

- Ben Weinstein (2014)
- Mike McCann (2015)
- Nicole Kinlock (2016, 2017)
- Bilgecan Şen (2018)

Program

- Bölüm 1: R Programlamaya Giriş
- Bölüm 2: Homojen Veri Yapıları
- Bölüm 3: Loops and Flow İfadeleri
- Bölüm 4: Fonksiyonlar
- Bölüm 5: Heterojen Veri Yapıları
- Bölüm 6: Tidyverse
- Bölüm 7: Temel Görselleştirme
- Bölüm 8: Kapanış

Bölüm 1: R Programlamaya Giriş

R

- Bir programlama dili
- Genellikle kullanıldığı alanlar:
  - veri düzenleme
  - istatistiksel analizler
  - veri görselleştirme
- Ross Ihaka ve Robert Gentlemen tarafından 1996 yılında geliştirildi
- "Interpreted" bir programlama dili, "compiled" değil
- Geliştirilmesi ve dağıtılması bir istatistikçi grubu tarafından üstlenilmiş durumda (R Core Development Team)
- *Comprehensive R Archive Network (CRAN)* tarafından ücretsiz olarak indirilebilir
- MacOS, Windows ve Linux üzerinde kullanılabilir

Neden R ?

- Esneklik
- Tekrar edilebilirlik

RStudio

- Sıradan bir metin editörü R betik dosyaları (script) için yeterli
- RStudio daha rahat bir kullanım tecrübesi sağlıyor
- Ekran dört parçadan oluşuyor:
  - **Workspace** (Sol üst): Betikleri yazdığınız ve kaydetttiğiniz yer
  - **Console** (Sol alt): Komutların çalıştırıldığı yer
  - **Environment, History** (Sağ üst): Objelerin saklandığı yer
  - **Files, Plots, Packages, ve Help** (Sağ alt): Veriyi görüntüleme, yardım

Çalıştırma

- Kodu doğrudan konsola yazın (eğer kodu kaydetmek istemiyorsanız) veya betik dosyasına yazın ve ardırdan çalıştırın (`CTRL + ENTER`)
- **betik** dosyası uzantısı **.R** olan ve içinde R komutları barındıran bir metin dosyası demektir. Yazdığınız kodları saklamanızı sağlar.

R üç temel bileşene sahip

- Operatörler (Operators)
- Objeler (Objects)
- Fonksiyonlar (Functions)

Aritmetik operatörler

- + Toplama
- - Çıkarma
- * Çarpma
- / Bölme
- ^ Katsayı
- %% Modülo (kalanı bulur)
- %/% Tamsayı bölümü (kalanı atar)

Aritmetik operatörler

```
2+2
```

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

```
7-2
```

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

```
4^3
```

```
## [1] 64
```

```
9/2
```

```
## [1] 4.5
```

Aritmetik operatörler

- Klasik matematik işlem öncelikleri geçerlidir.

```
2+5*2/3
```

Mantıksal operatörler

- < Küçüktür
- <= Küçüktür veya eşittir
- > Büyüktür
- >= Büyüktür veya eşittir
- == Eşittir
- != Eşit değildir
- ! NOT
- | veya
- & ve

Mantıksal operatörler

```
2<3
```

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

```
6>=4
```

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

```
(2+2)==5
```

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

```
9<=15 & 10>11
```

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

Objeler

- R'da depolanmış ve bir isme sahip olan "şeyler"
- Değişken, veri, fonksiyon, sonuç veya sayılar "şey" olabilir.
- R'da atama operatörü <-.

Objeler | Atama yapma

```
x <- 1
x
```

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

```
x + 4
```

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

```
x
```

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

Objeler | Atama yapma

```
y <- x + 4
y
```

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

```
x <- x + 4
x
```

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

Objeler | Objeleri kullanarak işlem yapma

```
x <- 6
y <- 38
y-x
```

```
## [1] 32
```

```
x*y
```

```
## [1] 228
```

```
y%%x
```

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

Objeler | Objeleri kullanarak işlem yapma

```
z <- y + x
z
```

```
## [1] 44
```

Objeler | Objeleri kullanarak işlem yapma

```
x <- 10
y <- 3
x==y
```

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

```
x>y
```

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

Alıştırma 1.1

1. Bir sayı seçin ve bunu **z** objesine atayın
2. **z** objesini ikiyle çarpın ve bunu **a** objesine atayın
3. **a** objesine 10 ekleyin ve bunu **b** objesine atayın
4. **b** objesini ikiye bölün ve bunu **c** objesine atayın
5. **z** objesini **c** objesinden çıkarın ve bunu **d** objesine atayın
6. **d** objesi **z** objesine eşit mi?

Objeler | Bir çok elementi tek bir objeye atama c()

```
x <- c(1,2,3,4,5)
x
```

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

```
y <- c(4,2)
y
```

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

Objeler | Bir çok elementle işlem yapma

```
x <- c(1,2,3,4,5)
x + 1
```

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

```
x*2
```

```
## [1]  2  4  6  8 10
```

Objeler | Bir çok elementle işlem yapma

```
y <- c(1,4,3,2,5)
x==y
```

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

```
x>y
```

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

Alıştırma 1.2

1. 45, 3, 9, 99, ve 0'ı barındıran bir obje oluşturun
2. 66, 0, 1, 5, and 0'ı barındıran bir diğer obje oluşturun
3. Bu iki objeyi toplayın
4. Birbiriyle çarpın
5. Birinciyi ikinciye bölün
6. Sonuç kaç çıktı?
7. R tek boyutlu ve birçok elemente sahip objelerle nasıl başa çıkıyor?

Fonksiyonlar

- *fonksiyon* verilen bir girdi (*arguments*) ile belirli bir işlemi gerçekleştiren objelerdir. R halihazırda bir çok fonksiyona sahip ama isterseniz kendi fonksiyonunuzu yazmanız da mümkün.
- Fonksiyonlar şu şekilde kullanılır:
  `name_of_function(inputs)`
- Fonksiyonun sonucu bir objeye saklanabilir:

```
output <- name_of_function(inputs)
```

Fonksiyonlar

- `sum()` fonksiyonunu bir objedeki tüm elementlerin toplamını elde etmek için kullanın:

```
x <- c(45,3,9,99,0)
sum(x)
```

```
## [1] 156
```

```
z <- sum(x)
z
```

```
## [1] 156
```

Fonksiyonlar

- `prod()` fonksiyonunu objedeki tüm elementleri birbiriyle çarpmak için kullanın:

```
prod(x)
```

```
## [1] 0
```

- `mean()` fonksiyonunu objedeki elementlerin ortalamasını almak için kullanın:

```
mean(x)
```

```
## [1] 31.2
```

İpucu: Yardım sistemi

- Yardım dosyaları fonksiyonların neler yaptığı, nasıl çalıştığı gibi bilgilerin yanı sıra çeşitli örnekleri de içinde barındırır.

```
help(mean)
```

- Fonksiyonun adından önce ? yazarak da yardım sayfasını görüntüleyebilirsiniz.

```
?mean   # help(mean) yazmakla aynı
```

- ?? kullanarak "sequence" kelimesini içeren tüm fonksiyonları aratabilirsiniz.

```
??sequence
```

Alıştırma 1.3

1. 34, 16, 105, ve 27'nin medyanını "median" hesaplayın. *Hatırlatma:* fonksiyon isimleri işlevlerine göre verilmiştir.
2. `range()` fonksiyonu ne yaptığını açıklayın ve yardım dosyasındaki örneğin ne olduğuna bakın.
3. `mean(4, 5)`'ün sonucu `mean(c(4, 5))`'den farklı mı?

Bölüm 2: Homojen Veri Yapıları

- R farklı veri tiplerini temsil eden 7 farklı *class*'a sahip
  - vector (atomic vector)
  - factor
  - matrix
  - array
  - data frame
  - list (recursive vector)
  - ts

1. Vektörler | Vektör oluşturma

- Şu ana kadar oluşturduğumuz her obje bir vektör
- Vektörler her uzunlukta olabilir
- Tek boyutlu

```
x <- 1
x <- c(5,6,7,8,9,10)
```

1. Vektörler | Vektör oluşturma

```
x <- 5:10
x
```

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

1. Vektörler | Vektör oluşturma

```
x <- seq(from = 5, to = 10, by = 1)
x
```

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

```
x <- seq(from = 5, to = 6, by = 0.1)
x
```

```
##  [1] 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6.0
```

1. Vektörler | Vektör oluşturma

```
is.vector(x)
```

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

```
y <- c()
is.vector(y)
```

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

```
z <- vector(length = 5, mode = "numeric")
z
```

```
## [1] 0 0 0 0 0
```

1. Vektörler | Vektör birleştirme

- Birden fazla vektörü tek bir vektör olarak birleştirebilirsiniz.

```
x <- c(1,2,3)
y <- c(4,5)
z <- c(x,y)
z
```

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

1. Vektörler | Vektörleri adlandırma

- Vektörleri oluştururken onu oluşturan elementleri adlandırabilirsiniz.

```
z <- c(monday = 1, tuesday = 2, wednesday = 3, thursday = 4
z
```

```
##    monday   tuesday wednesday  thursday    friday
##         1         2         3         4         5
```

```
names(z)
```

```
## [1] "monday"    "tuesday"   "wednesday" "thursday"  "fri
```

1. Vektörler | Vektörleri adlandırma

- veya aynı işlemi vektörü oluşturduktan sonra da yapabilirsiniz.

```
z <- c(1,2,3,4,5)
names(z)
```

```
## NULL
```

```
# Names need to be a `character` vector
names(z) <- c("monday", "tuesday", "wednesday", "thursday",
names(z)
```

```
## [1] "monday"    "tuesday"   "wednesday" "thursday"  "fri
```

1. Vektörler | Vektör elementlerini seçme: pozisyonuna göre

- Bir vektördeki her elemente pozisyonunu kare parentezin [ ] içine koyarak ulaşabilirsiniz.

```
height <- c(76, 72, 74, 74, 78)
height
```

```
## [1] 76 72 74 74 78
```

1. Vektörler | Vektör elementlerini seçme: pozisyonuna göre

- Bir vektördeki her elemente pozisyonunu kare parentezin [ ] içine koyarak ulaşabilirsiniz.

```
height[1] # 1. vektörü seç
```

```
## [1] 76
```

```
height[5] # 5. vektörü seç
```

```
## [1] 78
```

```
height[6] # 6. element yok
```

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

1. Vektörler | Vektör elementlerini seçme: pozisyonuna göre

- Seçilen elementleri yeni objelere atayabiliriz.

```
x <- height[1]
x
```

```
## [1] 76
```

```
y <- height[5]
y
```

```
## [1] 78
```

1. Vektörler | Vektör elementlerini seçme: pozisyonuna göre

- Aynı anda bir çok elementi seçebiliriz.

```
height[c(1,2,3)]
```

```
## [1] 76 72 74
```

```
height[1:3]
```

```
## [1] 76 72 74
```

1. Vektörler | Vektör elementlerini seçme: pozisyonuna göre

- - işaretini belirli elementleri dışarıda bırakarak geri kalanları seçmek için kullanabiliriz.

```
height[-1]
```

```
## [1] 72 74 74 78
```

```
height[c(-1,-5)]
```

```
## [1] 72 74 74
```

```
height_new <- height[-1]
```

1. Vektörler | Vektör elementlerini seçme: isme göre

- Vektör elementlerine isim atayıp daha sonra bu isimlerle seçim yapılabilir.

```
temp <- c(monday = 28.1, tuesday = 28.5, wednesday = 29.0,
temp
```

```
##    monday   tuesday wednesday  thursday    friday
##      28.1      28.5      29.0      30.1      30.2
```

1. Vektörler | Vektör elementlerini seçme: isme göre

- Vektör elementlerine isim atayıp daha sonra bu isimlerle seçim yapılabilir.

```
temp["wednesday"] # İsimle seçim yaparken her zaman ""  ku
```

```
## wednesday
##        29
```

```
temp[3]
```

```
## wednesday
##        29
```

```
# wednesday'i " " kullanmadan seçmeye çalışın.
# Ne oldu?
```

1. Vektörler | Vektör elementlerini seçme: mantıksal

- Çeşitli mantıksal ifadelerle belirli kriterlere ait olan elementler seçilebilir.

```
y <- 5:50
y
```

```
##  [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
## [26] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
```

1. Vektörler | Vektör elementlerini seçme: mantıksal

- Çeşitli mantıksal ifadelerle belirli kriterlere ait olan elementler seçilebilir.

```
# Tüm vektörü mantıksal hale dönüştürür.
y <= 10
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FAI
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAI
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAI
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAI
```

```
# 10'a eşit veya küçük olan tüm elementleri seç
y[y <= 10]
```

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

- Çeşitli mantıksal ifadelerle belirli kriterlere ait olan elementler seçilebilir.

```
# 10'dan küçük olan ve 5'e eşit olmayan tüm elementleri se
```

Alıştırma 2.1

1. `seq(1, 27, 0.5)` vektörünün 9 ve 12.sıradaki elementleri nelerdir?
2. 4 ve 34 arasındaki tüm çift sayıları içeren bir vektör yaratın ve bunu `a` objesine atayın. İpucu: `seq()` fonksiyonunu kullanın.
3. `a` vektöründe 17'den büyük olan tüm elementleri seçerek çıkarın.

Vektör çeşitleri

- “Storage mode”” veya sadece “mode”” olarak da adlandırılır.
- Altı çeşit vektör vardır:
  - Numeric (Double)
  - Integer
  - Character
  - Logical
  - Complex
  - Raw

Vektör çeşitleri | Numeric ve integer

```
x <- c(1,2,3,4,5)
is.double(x)
```

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

```
y <- c(1.25, 3.755, 9.001)
is.double(y)
```

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

Vektör çeşitleri | Numeric ve integer

```
z1 <- c(1L,2L,3L,4L,5L)
is.integer(z1)
```

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

```
z2 <- as.integer(y)
is.integer(z2)
```

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

Vektör çeşitleri | Character

```
x <- c("İsmail and Nurbahar are the best teaching assistant
x
```

```
## [1] "İsmail and Nurbahar are the best teaching assistant
```

```
length(x)
```

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

Vektör çeşitleri | Character

```
y <- c("İsmail", "and", "Nurbahar", "are", "best", "teachin
y
```

```
## [1] "İsmail"     "and"         "Nurbahar"   "are"
## [6] "teaching"   "assistants"
```

```
y[1]
```

```
## [1] "İsmail"
```

```
y[c(3,5)]
```

```
## [1] "Nurbahar" "best"
```

Vektör çeşitleri | Logical

```
x <- c(1,2,3,4,5)
y <- x<3
y
```

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

```
z <- c(TRUE, FALSE, T, F)
z
```

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

Modların birbirine dönüşümleri | Character'e dönüştürme

```
x
```

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

```
z
```

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

Modların birbirine dönüşümleri | Character'e dönüştürme

```
x <- as.character(x)
x
```

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

```
z <- as.character(z)
z
```

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

Modların birbirine dönüşümleri | numeric'e dönüştürme

```
x <- c("1","-2", "3.25", "A")
x <- as.numeric(x)
```

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

```
x
```

```
## [1]  1.00 -2.00  3.25    NA
```

Modların birbirine dönüşümleri | Numeric'e dönüştürme

```
y <- c(T,F,F,T,T)
y <- as.numeric(y)
y
```

```
## [1] 1 0 0 1 1
```

Modların birbirine dönüşümleri | Logical'a dönüştürme

```
x <- c(0, 0, 5, 79, 3500)
x <- as.logical(x)
x
```

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

```
y <- c("TRUE", "F", "1", "0", "35000")
y <- as.logical(y)
y
```

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

Alıştırma 2.2

1. TRUE ve 50'yi içeren bir vektör oluşturun.
2. "A" ve 1 içeren bir vektör oluşturun.
3. TRUE ve "C" içeren bir vektör oluşturun.
4. Örüntüyü farkettiniz mi? Oluşturduğunuz vektörleri modu nasıl belirlendi?

R ipucu: zorla güzellik olmaz

- Atomic bir vektörün tüm elementleri aynı tipte olmalıdır.
- Yani birbirinden farklı elementleri birleştirmeye çalıştığınızda otomatik olarak en esnek tipte birleşeceklerdir.
- En esnek tipler, en az esnek olandan başlayarak, şu şekilde sıralanır: logical, integer, double, ve character.

2. Faktörler

- Faktör sadece daha önceden belirlenmiş değerleri içerebilen bir vektör tipidir. Genellikle kategorik veri depolamak için kullanılır.
- Faktörler integer vektörlerin üstüne çeşitli bilgilerin eklenmesiyle oluşmuştur.

2. Faktörler | Faktör oluşturma

```
x <- c(1,2,3,4,5)
x <- factor(x)
x
```

```
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5
```

2. Faktörler | Faktör oluşturma

```r
z <- c("a", "b", "c", "d")
z <- as.factor(z)
z
```

```
## [1] a b c d
## Levels: a b c d
```

```r
z <- factor(z, levels = c("c","b","a","d"), ordered = T)
z
```

```
## [1] a b c d
## Levels: c < b < a < d
```

2. Faktörler | Faktör oluşturma

- Faktör "levelı" dışında kalan değerlerin ataması yapılmaz.

```
z[5] <- "a"
z[6] <- "e"
```

```
## Warning in `[<-.factor`(`*tmp*`, 6, value = "e"): inval
## generated
```

```
z
```

```
## [1] a    b    c    d    a    <NA>
## Levels: c < b < a < d
```

2. Faktörler | Faktör oluşturma

- Faktörler birleştirilemez.

```
f <- c(z,x)
f
```

```
##  [1] a    b    c    d    a    <NA> 1    2    3    4    5
## Levels: c b a d 1 2 3 4 5
```

```
is.factor(f)
```

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

```
typeof(f)
```

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

3. Matriksler

- İki boyutlu veri dosyası (tablo)
- Vektörlerle çok benzer davranış gösterirler

3. Matriksler | Matriks oluşturma

```
m <- matrix(nrow = 2, ncol = 2)
m
```

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

3. Matriksler | Matriks oluşturma

```
m <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
m
```

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

3. Matriksler | Matriks oluşturma

```
m <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow = T)
m
```

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

```
is.matrix(m)
```

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

3. Matriksler | Matriks oluşturma

```
v1 <- c(1,2,3)
v2 <- c(4,5,6)

m <- cbind(v1, v2)
m
```

```
##      v1 v2
## [1,]  1  4
## [2,]  2  5
## [3,]  3  6
```

3. Matriksler | Matriks oluşturma

```
m <- rbind(v1, v2)
m
```

```
##    [,1] [,2] [,3]
## v1    1    2    3
## v2    4    5    6
```

```
is.matrix(m)
```

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

3. Matriksler | Satır ve sütun adlandırma

```
names(m)
```

```
## NULL
```

```
colnames(m)
```

```
## NULL
```

```
rownames(m)
```

```
## [1] "v1" "v2"
```

3. Matriksler | Satır ve sütun adlandırma

```
colnames(m) <- c("a", "b", "c")
m
```

```
##    a b c
## v1 1 2 3
## v2 4 5 6
```

3. Matriksler | Aritmetik

```
m <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
m*2
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

3. Matriksler | Aritmetik

```
m2 <- matrix(c(5,6,7,8), nrow = 2, ncol = 2)
m + m2
```

```
##      [,1] [,2]
## [1,]    6   10
## [2,]    8   12
```

3. Matriksler | Matriks elementlerini seçme: pozisyona göre

```
january <- matrix(1:31, nrow = 5, ncol = 7, byrow = T)
```

```
## Warning in matrix(1:31, nrow = 5, ncol = 7, byrow = T):
## a sub-multiple or multiple of the number of rows [5]
```

```
january
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    1    2    3    4    5    6    7
## [2,]    8    9   10   11   12   13   14
## [3,]   15   16   17   18   19   20   21
## [4,]   22   23   24   25   26   27   28
## [5,]   29   30   31    1    2    3    4
```

3. Matriksler | Matriks elementlerini seçme: pozisyona göre

```
january[1,2] # birinci satır, ikinci sütun
```

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

```
january[5,4] # beşinci satır, dördüncü sütun
```

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

3. Matriksler | Matriks elementlerini seçme: pozisyona göre

```
january[1,]  # tüm satırlar
```

```
## [1] 1 2 3 4 5 6 7
```

```
january[,3]  # tüm sütunlar
```

```
## [1]  3 10 17 24 31
```

3. Matriksler | Matriks elementlerini seçme: isme göre

```
colnames(january) <- c("monday", "tuesday", "wednesday",
                       "thursday", "friday", "saturday", "s
rownames(january) <- c("week 1", "week 2", "week 3", "week
january
```

```
##        monday tuesday wednesday thursday friday saturda
## week 1      1       2         3        4      5       6
## week 2      8       9        10       11     12      13
## week 3     15      16        17       18     19      20
## week 4     22      23        24       25     26      27
## week 5     29      30        31        1      2       3
```

3. Matriksler | Matriks elementlerini seçme: isme göre

```
january[,"monday"] # İsimle seçme yaparken "" işaretlerini
```

```
## week 1 week 2 week 3 week 4 week 5
##      1      8     15     22     29
```

```
january[,"sunday"]
```

```
## week 1 week 2 week 3 week 4 week 5
##      7     14     21     28      4
```

3. Matriksler | Matriks elementlerini seçme: isme göre

```
january["week 1",]
```

```
##    monday   tuesday wednesday  thursday    friday  satur
##         1         2         3         4         5
```

```
january["week 3", "thursday"]
```

```
## [1] 18
```

3. Matriksler | Matriks elementlerini seçme: mantıksal

```
january<15 # Tüm matriksi mantıksal hale çevirir.
```

```
##        monday tuesday wednesday thursday friday saturday
## week 1   TRUE    TRUE      TRUE     TRUE   TRUE     TRUE
## week 2   TRUE    TRUE      TRUE     TRUE   TRUE     TRUE
## week 3  FALSE   FALSE     FALSE    FALSE  FALSE    FALSE
## week 4  FALSE   FALSE     FALSE    FALSE  FALSE    FALSE
## week 5  FALSE   FALSE     FALSE     TRUE   TRUE     TRUE
```

3. Matriksler | Matriks elementlerini seçme: mantıksal

```
january[january<15]
```

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

3. Matriksler | Matriks elementlerini seçme: mantıksal

```
january[,"tuesday"]
```

```
## week 1 week 2 week 3 week 4 week 5
##      2      9     16     23     30
```

```
tues <- january[,"tuesday"]
tues[tues<15]
```

```
## week 1 week 2
##      2      9
```

Alıştırma 2.3

1. İki tane 5'e 2lik matriks oluşturun. Bir tanesi 1-20 arasındaki tüm çift sayıları içermeli, diğeri tüm tek sayıları. Matriksleri satırlara göre doldurdun.
2. Bu iki matriksi satır ile birleştirin (rbind) ve yeni bir objeye atayın.
3. Bu iki matriksi sütun ile birleştirin (cbind) ve yeni bir objeye atayın.

Alıştırma 2.4

```
january <- matrix(1:31, nrow = 5, ncol = 7, byrow = T)
colnames(january) <- c("monday", "tuesday", "wednesday",
                       "thursday", "friday", "saturday", "s
rownames(january) <- c("week 1", "week 2", "week 3", "week
```

- Yukarıdaki kodu çalıştırın.

1. January ayından Saturday ve Sunday sütunlarını silin.
2. 9, 10, ve 11. günlerde hasta oldunuz (geçmiş olsun!). Bu günleri "hasta" olarak değiştirin. Matriksteki diğer girdiler bundan nasıl etkilendi?
3. Kalan günleri "sağlıklı" olarak değiştirin.
4. length() fonksiyonunu matriks üzerinde kullanınca ne sonuç alıyorsunuz? Aynı matriks üzerinde nrow() ve ncol() fonksiyonlarını kullanın. İlk sonuçla nasıl ilişkililer?

4. Arrayler

- Arrayler çok boyutludur.
- Vektörler ve matrikslerle benzer davranır.
- Aslında vektörler tek boyutlu array, matriksler iki boyutlu arraydir.

4. Arrayler

```
x <- array(c(1,2,3), dim = 3)
x
```

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

```
is.vector(x)
```

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

```
is.array(x)
```

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

4. Arrayler

```
m <- array(c(1,2,3,4), dim = c(2,2))
m
```

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

```
is.array(m)
```

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

```
is.matrix(m)
```

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

4. Arrayler

```
a <- array(c(1,2,3,4,5,6,7,8), dim = c(2,2,2))
a
```

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

4. Arrayler

```
a[,,1]
```

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

```
a[1,2,]
```

```
## [1] 3 7
```

```
a[1,2,2]
```

```
## [1] 7
```

Bölüm 3: Loops and Flow İfadeleri

Loop

- Looplar önemli! Bugün öğreneceğimiz loopun adı **for loop**.
- For loopları yazdığınız kodu belirlediğiniz sayıda tekrar eder (iteration).
- Temel taslak:

```
for (i in 1:number_of_iterations) {

buraya bakarlar

buraya kod gelecek

buraya bakarlar

}
```

For loop | Neden for loop?

- Aynı işlemi farklı değerler için tekrarlamamız gerektiğinde.
- Yorucu miktarda kod yazmamak için.

For loop | İlk deneme

```
for (i in 1:5){
    print(i)
}
```

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

For loop | İlk deneme

```
for (i in 1:5) {
```

- i 1'den başlar. R, kodları çalıştırır;
- i artar ve 2 olur. R aynı kodu tekrar çalıştırır;
- i artar ve 3 olur. R aynı kodu tekrar çalıştırır;
- . . . ve böyle devam eder, i 5 olduğunda R aynı kodu tekrar ama son kez çalıştırır.

```
}
```

For loop

```
for (i in 1:2){
    print(i)
}
```

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

- For loop bittikten sonra **i**'yi çağırabilirsiniz.

```
i # i şu anda workspace'de bir obje.
```

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

- i, eğer onunla başka bir loop çalıştırırsanız kendini yenileyecektir.

For loop

- Loopun içinde objelere atama yapabiliriz.

```
x <- 2

for (i in 1:4){
    x <- x^2
    print(x)
}
```

```
## [1] 4
## [1] 16
## [1] 256
## [1] 65536
```

Alıştırma 3.1

Bu alıştırma için 2.kısımda oluşturduğunuzda `january` matriksini kullanın.

1. Tüm wednesdaylerin tarihlerini yazdırın.
2. Beşinci haftada yer alan February günlerini `NA` ile değiştirin.

For loop | Loop Environment

- Looplar kendi küçük dünyalarına sahiptir. `print()` fonksiyonunu kullanarak her tekrarın sonucunu görebilirsiniz.
- Eğer `print()` veya atama `<-` kullanmazsanız sonuçlar size dönmez.

```
wizards <- c("Harry", "Hermione", "Ron")

for (i in 1:length(wizards)){
    paste("Hi,", wizards[i], sep=" ")
}
```

Alıştırma 3.2

1. Sizin sıranızda oturan insanların ismiyle bir vektör oluşturup, bir loop ile onlara küçük süprizler yapın (güzel bir şeyler yazın).

For loop | Loop sonuçlarını nasıl kaydederiz?

- Loop sonucunu ekrana çıktı olarak almaktansa (print), sonuçları obje olarak saklamayı tercih etmeli.
- Bunu yapmak için loopa başlamadan önce boş bir obje oluşturmanız lazım.

```
x <- rnorm(n = 10, mean = 1, sd = 0.5)  # Loopda kullanaca

outputs <- c()  # Boş bir vektör oluştur

for (i in 1:length(x)) {
  outputs[i] <- x[i] * 10
}
head(outputs)
```

```
## [1] 11.558330  8.077646 12.353513  5.584462  1.120393  4
```

Alıştırma 3.3

```
x <- rnorm(n = 50, mean = 0, sd = 1)
m <- matrix(x, ncol = 5, nrow = 10)
```

1. `m` matriksindeki her sütunun ortalamasını hesaplayan bir loop yazın ve sonucu a objesine atayarak saklayın.

For loop | Nested for loop

- Eğer manipüle ettiğimiz obje tek boyutluysa, tek bir for loop yeterli.
- Eğer birden fazla boyutlu objelerle uğraşıyorsak iç içe birden fazla for loop yazmamız gerekiyor.

```
m <-  matrix(1:6, nrow = 3, ncol = 2)
for (i in 1:nrow(m)) {
  for(h in 1:ncol(m)) {
    print(m[i,h])
  }
}
```

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

For loop | Nested for loop

- Tekrar indekseleri (i, h vb.) objenin doğru kısmıyla eşleşmeli.

```
for (i in 1:nrow(m)) {
  for(h in 1:ncol(m)) {
    print(m[h,i])
  }
}
```

For loop | Nested for loop

- Tekrar indekslerini dikkatlice takip edin. İç içe geçmiş looplarda tekrar tekrar kullanmayın.

```
for (i in 1:nrow(m)) {
  for(i in 1:ncol(m)) {
    print(m[i,i])
  }
}
```

Flow ifadeleri | If ifadeleri

```
if (3 > 2) {
  print("Yes")
}
```

```
## [1] "Yes"
```

Flow ifadeleri | If ifadeleri

- Temel taslak:

```
if (mantıksal ifade) {

buraya kod gelecek

}
```

- Eğer mantıksal önerme doğru ise, **TRUE**, kod çalışacak. Eğer yanlış ise, **FALSE**, çalışmayacak.

Flow ifadeleri | If ifadeleri

- Genellikle loopların farklı koşullar altında farklı şeyler yapmasını isteriz. Mesela: değişkenleri, seçenekleri ya da mantıksal önermeleri dikkate almasını isteriz.

Flow ifadeleri | If ifadeleri

```
x <- 1:5

for (i in 1:length(x)) {
  if(x[i] > 3) {
    print(paste(x[i],"3'ten büyüktür"))
  }
  if(x[i] <= 3) {
    print(paste(x[i],"3'ten küçük ya da 3'e eşittir"))
  }
}
```

```
## [1] "1 3'ten küçük ya da 3'e eşittir"
## [1] "2 3'ten küçük ya da 3'e eşittir"
## [1] "3 3'ten küçük ya da 3'e eşittir"
## [1] "4 3'ten büyüktür"
## [1] "5 3'ten büyüktür"
```

Alıştırma 3.4

1. `x` isimli 1 ve 100 arasındaki tüm sayıları içeren bir vektör oluşturun.
2. Vektör için `x[i] * 2` işlemini 100 kez (vektörün uzunluğu) gerçekleştirin. Fakat eğer `x[i]` 32'den büyük ve 50'den küçükse `x[i] * 3`'ı hesaplayın. Çıktıyı bir vektöre atayın.

Flow ifadeleri | Else ifadeleri

- `if` kullanırken eğer mantıksal önerme yanlış ise, `FALSE`, hiç bir şey çalışmaz.
- `else` kullanarak mantıksal önerme yanlış, `FALSE`, olduğunda da bir başka kod çalıştırabiliriz.

```
if (1 > 2) {
  print("Yes")
} else print("No")
```

```
## [1] "No"
```

Flow ifadeleri | Else ifadeleri

- `if` kullanırken eğer mantıksal önerme yanlış ise, `FALSE`, hiç bir şey çalışmaz.
- `else` kullanarak mantıksal önerme yanlış, `FALSE`, olduğunda da bir başka kod çalıştırabiliriz.

```
if (1 > 2) {
  print("Yes")
} else {
  print("No") # Süslü parantezleri else ifadeleriyle de ku
}
```

```
## [1] "No"
```

Alıştırma 3.5

- Kene istilasına uğramış beş farklı coğrafi alan var. Bilim insanları her bölgeden 10 geyik için örneklem alarak sayım yapıyor. Bunun gibi basit bir senaryoyu her bölgedeki beklenen ortalama kene sayısını bildiğimizi ve sayımı yapılan kenelerin poisson dağılımına uyduğunu varsayarak simüle edebiliriz.

```
tick_mean <- c(4,32,17,10,12)
tick_count <- matrix(nrow = 10, ncol = 5)
for (i in 1:5) {
  tick_count[,i] <- rpois(10, tick_mean[i])
}
```

- Yukarıdaki kodu çalıştırın.

1. Yeni bir veriseti oluşturun ve bu verisetinde 10 veya 10'dan daha az kenesi olan hayvanlara 0, 10'dan daha fazla kenesi olanlara 1 atayın. Bunu loop kullanarak yapın. Daha sonra loop kullanmadan yapın.

Bölüm 4: Fonksiyonlar

Temel fonksiyonlar

- Fonksiyonlar sık sık gerçekleştirdiğimiz işlemleri gerçekleştiren komutlara sahiptir.
- For loopları gibi gereksiz yere kod yazmayı önlerler.
- R ile yüklü olarak gelen bir çok temel fonksiyonu şimdiden gördük.

```
sum(seq(1, 100, 1))
abs(-100 + 50)
dim(iris)
str(iris)
colnames(iris)
```

Paket Fonksiyonlar

- Bir çok başka fonksiyon paketler (packages) aracılığıyla yüklenebilir.

```
# Bu çalışmayacak çünkü paket yüklü (install) ve açık (loa
tree(formula = Species ~ . -Species, data = iris)

install.packages("tree") # Paketi yükler
library(tree) # Paketi açar

# Şimdi çalışmalı.
tree(formula = Species ~ . -Species, data = iris)
```

- R kullanan onlarca insan kendi amaçları için yazdıkları paketleri paylaşıyor. Bu sayede büyük bir çeşitlilik oluşuyor.

Kendi fonksiyonuzu yazın

- R'da *Kendi fonksiyonunuzu yazmak* mümkün. Bu eğer aynı kodu tekrar tekrar yazacaksanız çok önemli ve gerekli bir durum.
- R fonksiyonları diğer her şey gibi birer obje.

Fonksiyon yazma

- Temel taslak:

```
function_name <- function(arguments) {   body }
```

```
# fonksiyon tanımlayın, f
f <- function(x, y) {
  x + y
}

# f fonksiyonunu çağırın
f(x = 1, y = 3)
```

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

Fonksiyon yazma

- Temel taslak:

```
function_name <- function(arguments) {   body }
```

```
f
```

```
## function(x, y) {
##   x + y
## }
```

```
# Fonksiyonu f() olarak çağırmayı deneyin. Ne hata aldınız
```

Fonksiyonun temel bileşenleri

- Body
- Formals
- Environment

```
body(f)
```

```
## {
##     x + y
## }
```

Fonksiyonun temel bileşenleri

- Body
- Formals
- Environment

```
formals(f)
```

```
## $x
##
##
## $y
```

Fonksiyonun temel bileşenleri

- Body
- Formals
- Environment

```
environment(f)
```

```
## <environment: R_GlobalEnv>
```

```
# Sum fonksiyonunda bu üç bileşene bakmaya çalışın. Ne gör
```

Fonksiyon ve environment

- Fonksiyon içinde tanımlanmış değişkenler genel environment'a dahil değildir, bunun yerine farklı bir environment'ta yer alırlar. Yani fonksiyonun dışında varlıkları geçerli değildir.
- Fakat eğer değişken fonksiyon içinde tanımlanmış değilse fonksiyon bir üst seviyeye bakacaktır.
- Eğer bu fonksiyonu çalıştırırsanız göreceksiniz ki environment panelinde yobjesi oluşmayacak.

Fonksiyon ve environment

```
nurbahar <- 2  # fonksiyon dışında oluşturulan obje

g <- function() {
     bilgecan <- 1 # fonksiyon içinde oluşturulan obje
    c(nurbahar, bilgecan)
}

g()
```

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

- Environemnt kımsında nurbahar adlı bir objeniz var mı? Peki bilgecan?

Fonksiyonlar

- `return()` fonksiyonunu kullanmadığınız ya da fonksiyonun sonuna çıktı yazmadığınız müddetçe fonksiyonun sonucuna ulaşamazsınız.

```
f1 <- function(a, b) {
    x <- a + b
    y <- (a + b)^2
    z <- a/b
}

f1(1, 2)  # sonuç yok.
```

Fonksiyonlar

```r
f2 <- function(a, b) {
    x <- a + b
    y <- (a + b)^2
    z <- a/b
    return(c(x, y, z))
}

f3 <- function(a, b) {
    x <- a + b
    y <- (a + b)^2
    z <- a/b
    c(x, y, z) # return(c(x, y, z)) yazmakla aynı
}

# x, y, ve z'yi veriyor.
f2(1, 2)
```

```
## [1] 3.0 9.0 0.5
```

Alıştırma 4.1

1. İki girdi (arguments) alan bir fonksiyon yazın. Girdiler `x` ve `y` olsun ve fonksiyon `x * 2 * y`'yi hesaplasın.
2. Üç girdi alan bir fonksiyon yaratın ve sonuçtan vektör yaratın.

Fonksiyonlar

- Fonksiyon girdileri bir vektör ya da liste olarak verilebilir.

```
params <- c(5, 25)
params
```

```
## [1]  5 25
```

```
f3 <- function(p){
    alpha <- p[1]
    beta <- p[2]
    alpha * beta
}

f3(params)
```

```
## [1] 125
```

Fonksiyonlar ve önceden atanmış değerler

```
subtract <- function(a = 5, b = 2){
    return(a - b)
}

subtract()
```

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

```
subtract(5, 6)
```

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

Alıştırma 4.2

1. Bir vektörü girdi olarak alan, daha sonra vekötürün toplamını 10 ile çarpıp sonucun 1000'den küçük olduğunu mantıksal önerme ile dönen bir fonksiyon yazın.

Alıştırma 4.3 (Zor)

1. Verilen bir numarayı bölenlerine ayıran bir fonksiyon yazın.

Bölüm 5: Heterojen Veri Yapıları

5. Data frameler

- R'la çalışırken kullanacağınız veriler genellikle tablo formatında olacak fakat farklı değişken tipleri içerecek. Yani matriksler ve vektörler bu ihtiyacı karşılamayacak.
- Data frameler farklı veri tiplerinin bir araya geldiği objelerdir.
- R'la yüklü gelen bir çok veri seti var. Biz bugün ünlü **iris** (R.A. Fisher, 1936) verisetiyle çalışacağız.

5. Data frameler

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
```

5. Data frameler

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Spec
## 1          5.1         3.5          1.4         0.2  set
## 2          4.9         3.0          1.4         0.2  set
## 3          4.7         3.2          1.3         0.2  set
## 4          4.6         3.1          1.5         0.2  set
## 5          5.0         3.6          1.4         0.2  set
## 6          5.4         3.9          1.7         0.4  set
```

```
is.data.frame(iris)
```

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

5. Data frameler | Dataframeler ve matriksler için faydalı fonksiyonlar

- `head()` - ilk 6 satırı gör
- `tail()` - son 6 satırı gör
- `dim()` - boyutları gör (# rows, # columns)
- `nrow()` - satır sayısı
- `ncol()` - sütun sayısı
- `str()` - objenin yapısı
- `rownames()` - satır isimleri
- `colnames()` - sütun isimleri

Alıştırma 5.1

trees ve *mtcars* R'da yüklü olan diğer iki veri setidir.

1. Bu iki veri setinin kaç satıra sahip olduğuna bakın.
2. Kaç sütuna sahipler? Sütünların isimleri neler?
3. `str()` fonksiyonunu kullanarak *iris* veri setinin kaç türe sahip olduğunu bulun.
4. İki veri setinde hangi class'lar mevcut?

5. Data frameler | Veri setinden seçim yapma (subsetting)

- Matrikslerle çok benzer.

```
iris[1,3]
```

```
## [1] 1.4
```

```
head(iris[,4])
```

```
## [1] 0.2 0.2 0.2 0.2 0.2 0.4
```

```
tail(iris[,"Species"])
```

```
## [1] virginica virginica virginica virginica virginica vi
## Levels: setosa versicolor virginica
```

5. Data frameler | Veri setinden seçim yapma (subsetting)

- Belirli sütunları seçmek için dolar işaretini kullanabiliriz.

```
head(iris$Petal.Width)
```

```
## [1] 0.2 0.2 0.2 0.2 0.2 0.4
```

```
head(iris[,4])
```

```
## [1] 0.2 0.2 0.2 0.2 0.2 0.4
```

Alıştırma 5.2

1. `iris` veri setinde `Sepal.Width` sütunundaki 9. elementi bulun?
2. `iris` veri setindeki 17.satırı seçin.
3. `iris` veri setinin 1, 4 ve 7. satırları ile bir obje oluşturun.
4. `iris` veri setindeki tüm tek sayılı satırları seçmek için `seq()` fonksiyonunu kullanın.
5. `iris` vei setindeki `Sepal.Width` sütununu başka bir objeye atayın ve sonrasında silin.
6. `iris` veri setinde petal uzunluğu 3'den büyük olanları seçin.
7. Bir veri setinde `length()` fonksiyonunu kullandığınızda ne sonuç alıyorsunuz? Matriksten farkı ne?

5. Data frameler | Veri seti oluşturma

```r
d <- data.frame(x = c(5.6, 2.45, 7.09), y = c("a","b","c"),
d
```

```
##      x y z
## 1 5.60 a 1
## 2 2.45 b 2
## 3 7.09 c 3
```

```r
str(d)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ x: num  5.6 2.45 7.09
##  $ y: chr  "a" "b" "c"
##  $ z: num  1 2 3
```

5. Data frameler | Veri seti oluşturma

```
d <- data.frame(x = c(5.6, 2.45, 7.09), y = c("a","b","c"),
                stringsAsFactors = F)
str(d)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ x: num  5.6 2.45 7.09
##  $ y: chr  "a" "b" "c"
##  $ z: num  1 2 3
```

5. Data frameler | Veri seti oluşturma

```
d$t <- as.factor(c(10,20,30))
d
```

```
##      x y z  t
## 1 5.60 a 1 10
## 2 2.45 b 2 20
## 3 7.09 c 3 30
```

```
str(d)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ x: num  5.6 2.45 7.09
##  $ y: chr  "a" "b" "c"
##  $ z: num  1 2 3
##  $ t: Factor w/ 3 levels "10","20","30": 1 2 3
```

5. Data frameler | Veri setlerini birleştirme

```
d1 <- data.frame(x = c(5.6, 2.45, 7.09), y = c("a","b","c")
                 stringsAsFactors = F)
d2 <- data.frame(z = c(1,2,3), t = as.factor(c(10,20,30)))
d3 <- cbind(d1,d2)
str(d3)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ x: num  5.6 2.45 7.09
##  $ y: chr  "a" "b" "c"
##  $ z: num  1 2 3
##  $ t: Factor w/ 3 levels "10","20","30": 1 2 3
```

5. Data frameler | Veri setlerini birleştirme

```
# İki veri setini satırlarla birleştirirken sütun isimleri
# Bir önceki slayttan d1 ve d2'yi birleştirmeyi deneyin. N

d2 <- data.frame(x = c(1,2,3), y = as.factor(c(10,20,30)))
d4 <- rbind(d1,d2)

# Zorla güzellik olmayacağını hatırlayın.
str(d4)
```

```
## 'data.frame':    6 obs. of  2 variables:
##  $ x: num  5.6 2.45 7.09 1 2 3
##  $ y: chr  "a" "b" "c" "10" ...
```

Alıştırma 5.3

1. c(1,2,3) ve c(4,5,6,7) ile iki sütunlu bir veri seti oluşturun. Sonuç ne?
2. c("a","b","c") ve c(1,2,3) ile `dataframe` kulanarak bir veri seti oluşturun. Bir diğer veri setini önce `cbind`, sonra `data.frame` kullanarak oluşturun. Aradaki fark ne?

Kendi verinizi R'da açma

- R bir spreadsheet programı değil. Bu yüzden doğrudan veri girişi için güzel değil. O yüzden veri girişi ve saklamak için spreadsheetsleri kullanmak ve bunları görselleştirme ve analizler için R'a aktarmak en mantıklısı.
- Genellikle **.csv** (comma separated values) formatı tercih edilir.
- Buna geçmeden önce **working directory** (çalışma dizini) kavramına bakalım.

Kendi verinizi R'da açma | Working directory

- Şu anda kullandığınız çalışma dizinini bulun

```
getwd()
```

```
## [1] "/Users/ibekar/Github/DogaBilimleriR/Sunumlar"
```

- Bu klasör, bilgisayarınızın R'da dosya açmak ya da yazmak için kullanacağı klasördür.

Kendi verinizi R'da açma | Working directory

- Çalışma dizininiz ayarlayın.

```
setwd("/Users/bilgecan/Desktop/")
```

Kendi verinizi R'da açma

```
blue.hill <- read.csv("BlueHill.csv")
head(blue.hill)

# Çalışma dizininde olmayan dosyayı açma
blue.hill <- read.csv("/Users/bilgecan/Desktop/BlueHill.csv
```

İpucu: değişken isimleri

- Bir veriyi R'da açmak bazen çok sıkıntılı hale gelebilir. Genellikle sorun değişken isimlerinden gelir.
- Değişken isimlerinde boşluk kullanmayın.
- Küçük harfler kullanın
- Büyük harfleri sadece gerekli olduğunda kullanın.

```
Average Height # KÖTÜ - hata verecek
Average.Height # EH - çalışır fakat daha iyi olabilir.
average.height # DAHA İYİ - tekrar tekrar yazarken yavaşla
avg.height     # ÇOK İYİ!
```

Alıştırma 5.4

1. Şu adresteki veriyi BlueHill.csv R'da açın.
2. Tüm istasyonlar ve yıllardaki günlük ortalama sıcaklık nedir? (**MNTM**)?
3. Dördüncü sütunu seçin ve bir objeye atayın. Class'ı ve modu nedir? İpucu: `class` ve `mode` fonksiyonlarını kullanın.
4. İlk satırı seçin ve bir objeye atayın. Class'ı nedir? Tek bir moda mı sahip?
5. Üçüncü ve dördüncü alıştırmaları düşündüğünüzde dataframedeki seçim ile matrikslerdeki seçim arasında nasıl farklar var?

Veri setlerini dışarı aktarma

- Değişiklk yaptıktan sonra veri setlerinizi dışarı aktarmak isteyebilirsiniz.

```
# Bu dosyayı yazar
write.csv(iris, file = "iris.csv", row.names = FALSE)

# Dosya çalışma dizininizde mi diye kontrol eder
list.files()
```

6. Listeler (lists, recursive vectors)

- Listeler aynı anda herhangi bir tipte elementi ve class'ı barındırabilir.
- Veri seti vektörlerden oluşan bir dizi listeler barındırır.
- Özellikle belirtmediğiniz sürece listelerdeki veriler tablo formatında saklanmaz.

6. Listeler

```
d <- data.frame(x = c(5.6, 2.45, 7.09), y = c("a","b","c")
l <- list(x = c(5.6, 2.45, 7.09), y = c("a","b","c"), z = a
l
```

```
## $x
## [1] 5.60 2.45 7.09
##
## $y
## [1] "a" "b" "c"
##
## $z
## [1] 1 2 3
## Levels: 1 2 3
```

```
typeof(l)
```

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

```
typeof(d)
```

6. Listeler

- Listelerdeki elementler başka listeler ve veri setleri içerebilir.

```
# Bir veri setine sütun ekleme ile benzer
l$d <- d
l[2:4]
```

```
## $y
## [1] "a" "b" "c"
##
## $z
## [1] 1 2 3
## Levels: 1 2 3
##
## $d
##      x y z
## 1 5.60 a 1
## 2 2.45 b 2
## 3 7.09 c 3
```

6. Listeler

```
# Listception
l$l <- l
l[4:5]
```

```
## $d
##      x y z
## 1 5.60 a 1
## 2 2.45 b 2
## 3 7.09 c 3
##
## $l
## $l$x
## [1] 5.60 2.45 7.09
##
## $l$y
## [1] "a" "b" "c"
##
## $l$z
```

6. Listeler | Listelerde seçim yapma (subsetting)

- Kare parantezler seçilen elementi liste olarak geri döndürür.

```
l[1]
```

```
## $x
## [1] 5.60 2.45 7.09
```

```
str(l[1])
```

```
## List of 1
##  $ x: num [1:3] 5.6 2.45 7.09
```

6. Listeler | List subsetting

- Çift kare parantez elementi orjinal tipi ve classı ile döndürür.

```
l[[1]]
```

```
## [1] 5.60 2.45 7.09
```

```
str(l[[1]])
```

```
##  num [1:3] 5.6 2.45 7.09
```

6. Listeler | Liste isimlendirme

```
names(l)
```

```
## [1] "x" "y" "z" "d" "l"
```

6. Listeler | Liste isimlendirme

```
names(l)[1] <- "new name"
names(l)[1]
```

```
## [1] "new name"
```

```
names(l[1])
```

```
## [1] "new name"
```

6. Listeler | Liste isimlendirme

```
names(l[[1]])
```

```
## NULL
```

```
names(l[[1]])[1] <- "a"
l[1]
```

```
## $`new name`
##    a <NA> <NA>
## 5.60 2.45 7.09
```

6. Listeler | Liste isimlendirme

- Element seçmek için $ işaretini de kullanabilirsiniz. Bu, iki kare [[]] parentez kullanımı ile aynı.

```
l$x
```

```
## NULL
```

```
l$y
```

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

Alıştırma 5.5

- Aşağıdaki objeyi oluşturun.

```
d <- data.frame(x = c(5.6, 2.45, 7.09),
                y = c("a","b","c"),
                z = as.factor(c(1,2,3)))
l <- list(x = c(5.6, 2.45, 7.09),
          y = c("a","b","c"),
          z = as.factor(c(1,2,3)))
l2 <- list(a = c(5:10), b = c(500:600))
l$d <- d
l$l2 <- l2
```

Alıştırma 5.5

1. `z` listesindeki ilk elementi seçin.
2. `d` listesindeki ikinci sütunu seçin.
3. `l` listesinden b'den 50. elementi seçin
4. `x`'in içindeki `1`'yi `[]` ve `[[]]` ile çıkarak 2 ile çarpın. Neden biri çalışırken diğeri çalışmıyor?

Genel seçim yapma önerileri

- Vektörleri ve faktörleri `[]` kullanarak seçin.
- Matriksleri, arrayleri ve veri setlerini `[,]` ile seçin.
- Listelerden seçim yaparken, eğer sonucun bir liste olmasını istiyorsanız veya birden fazla element seçmek istiyorsanız `[]` kullanın.
- Listelerden seçim yaparken gerçek veriyi seçip kullanmak istiyorsanız `[[]]` kullanın.

Bölüm 6: Tidyverse

Tidyverse nedir?

- Birden fazla R paketinden oluşan bir paket topluluğu
- ggplot2, dplyr, tibble, readr

Tibble

- Tibble = Steroidli dataframe

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
```

Tibble

```
library(tibble)
iris <- as_tibble(iris)
iris
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Spe
##           <dbl>       <dbl>        <dbl>       <dbl> <fc
##  1          5.1         3.5          1.4         0.2 set
##  2          4.9         3            1.4         0.2 set
##  3          4.7         3.2          1.3         0.2 set
##  4          4.6         3.1          1.5         0.2 set
##  5          5           3.6          1.4         0.2 set
##  6          5.4         3.9          1.7         0.4 set
##  7          4.6         3.4          1.4         0.3 set
##  8          5           3.4          1.5         0.2 set
##  9          4.4         2.9          1.4         0.2 set
## 10          4.9         3.1          1.5         0.1 set
## # ... with 140 more rows
```

Tibble

```
library(tibble)
df_tibble <- tibble(`İsim` = "İsmail",
                    `000` = "numara",)
df_tibble
```

```
## # A tibble: 1 x 2
##   İsim   `000`
##   <chr>  <chr>
## 1 İsmail numara
```

```
df_normal <- data.frame(`İsim` = "İsmail",
                        `000` = "numara")
df_normal
```

```
##     İsim   X000
## 1 İsmail numara
```

Readr

- Dosyaları daha hızlı ve hatasız bir şekilde okumak için geliştirildi

Readr

```
blue.hill <- read.csv("BlueHill.csv")
head(blue.hill)
```

```
##                STATION                                    STATION_N
## 1 GHCND:USC00190736 EAST MILTON BLUE HILL OBSERVATORY MA
## 2 GHCND:USC00190736 EAST MILTON BLUE HILL OBSERVATORY MA
## 3 GHCND:USC00190736 EAST MILTON BLUE HILL OBSERVATORY MA
## 4 GHCND:USC00190736 EAST MILTON BLUE HILL OBSERVATORY MA
## 5 GHCND:USC00190736 EAST MILTON BLUE HILL OBSERVATORY MA
## 6 GHCND:USC00190736 EAST MILTON BLUE HILL OBSERVATORY MA
```

Readr

```
library(readr)
blue.hill <- read_csv("BlueHill.csv")
```

```
## Rows: 1210 Columns: 4
## -- Column specification --------------------------------
## Delimiter: ","
## chr (2): STATION, STATION_NAME
## dbl (2): DATE, MNTM
##
## i Use `spec()` to retrieve the full column specification
## i Specify the column types or set `show_col_types = FALS
```

Readr

```
blue.hill <- read.csv("BlueHill.csv")
```

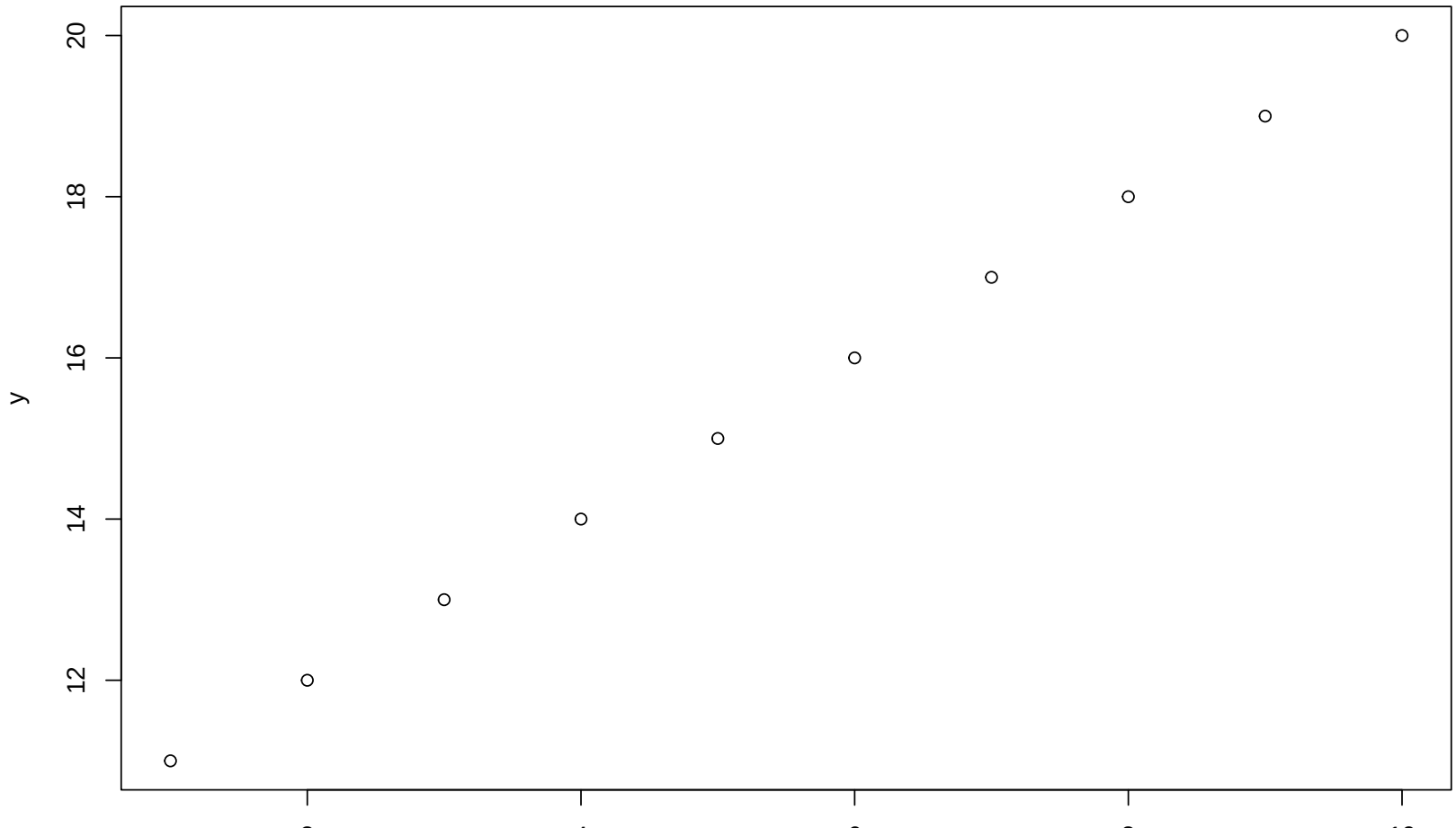
Alıştırma 6.1

1. Temel R kullanarak bir data frame oluşturun.
2. Temel R kullanarak bu data frame'i csv dosyası olarak yazdırın.
3. Temel R kullanarak bu csv dosyasını R'a aktarın.
4. 1, 2 ve 3 numaralı alıştırmaları tidyverse fonksiyonları kullanarak yapın ve aradaki farkı bulun.

Bölüm 7: Temel Görselleştirme

Scatter Plot

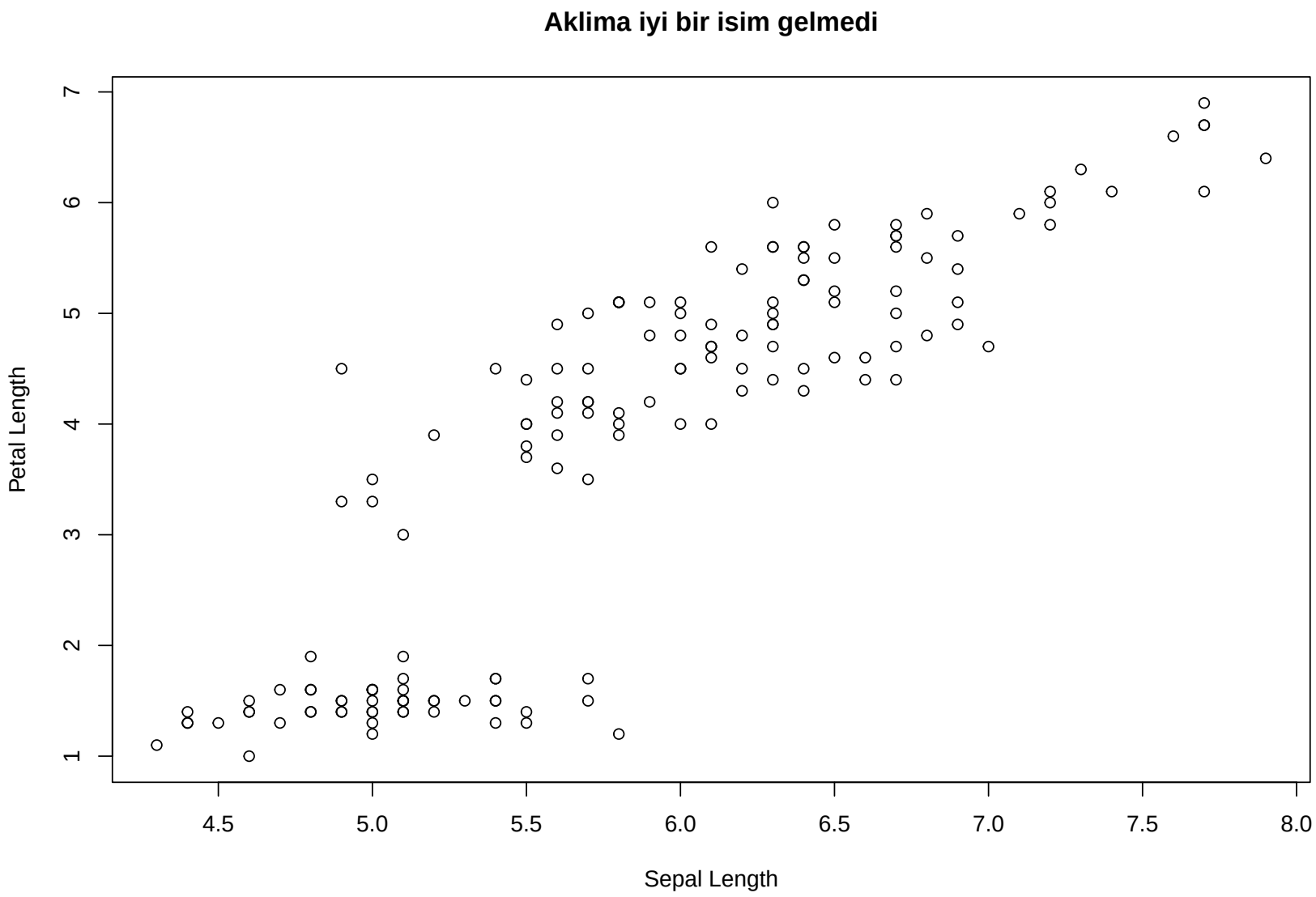
```
x <- 1:10
y <- 11:20
plot(x, y)
```



Scatter Plot

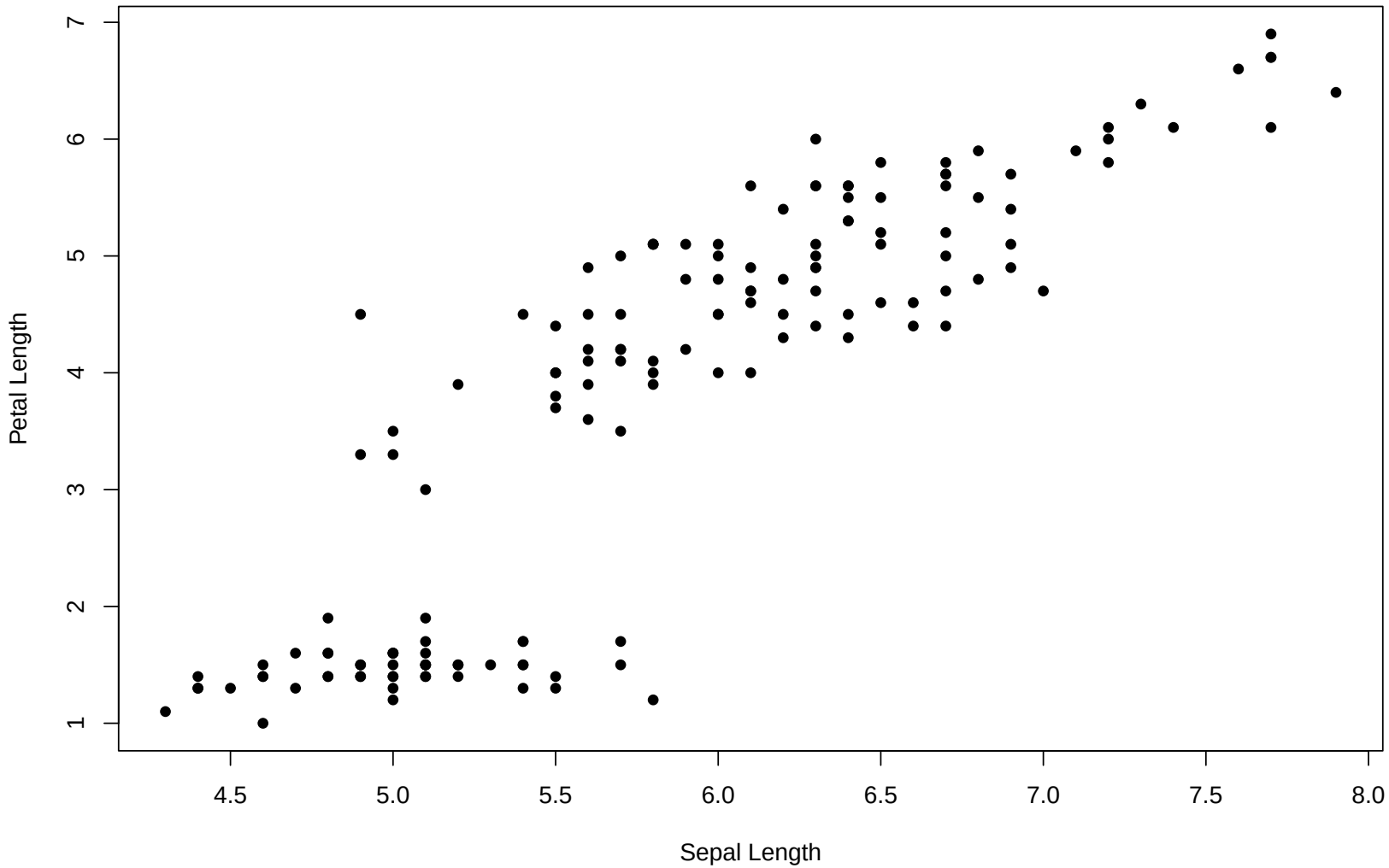
```
iris <- iris[order(iris$Sepal.Length),]
x <- iris$Sepal.Length
y <- iris$Petal.Length
plot(x, y,
     xlab = "Sepal Length",
     ylab = "Petal Length",
     main = "Aklima iyi bir isim gelmedi")
```

Scatter Plot



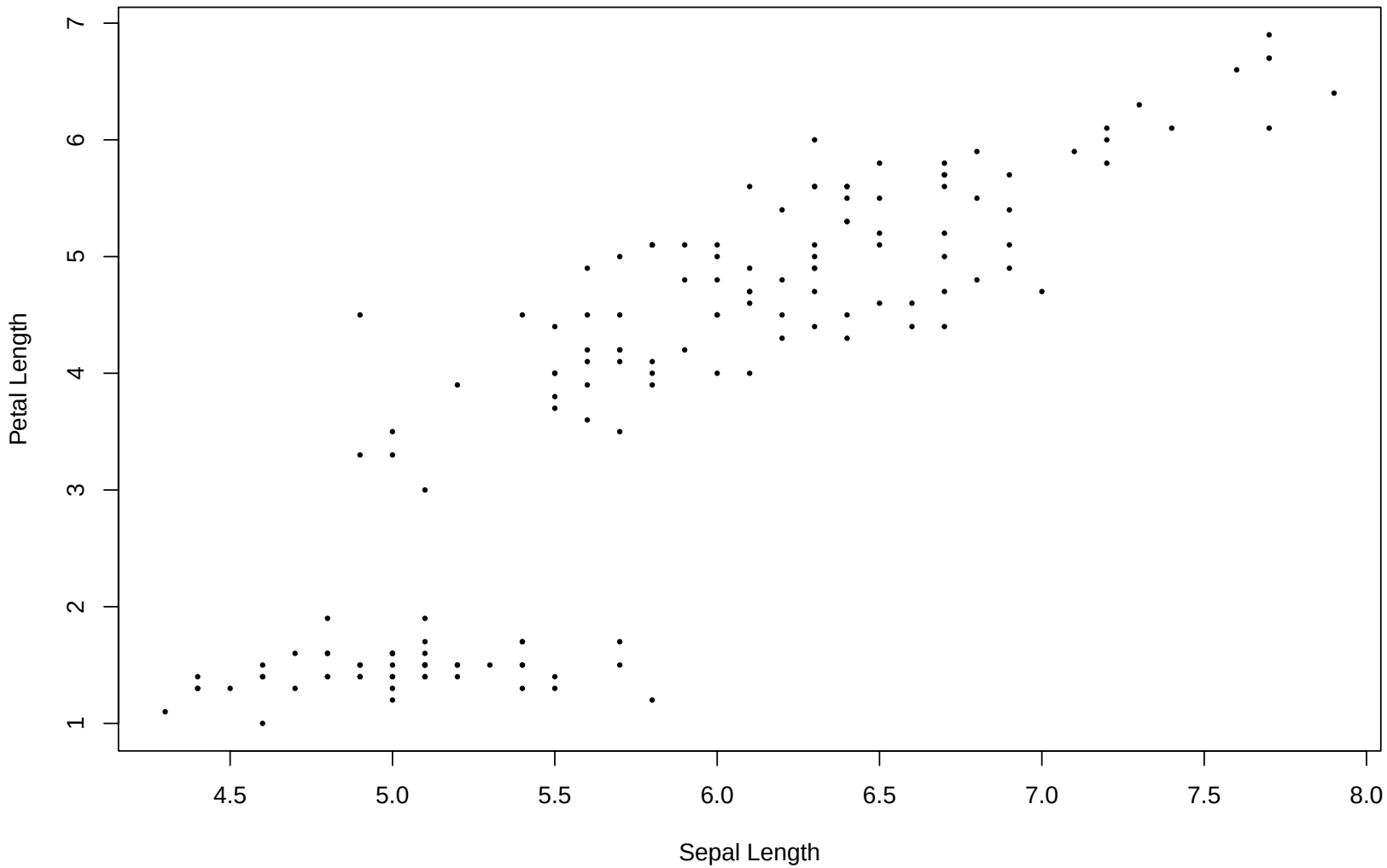
Scatter Plot

```
plot(x, y, xlab = "Sepal Length", ylab = "Petal Length", po
```



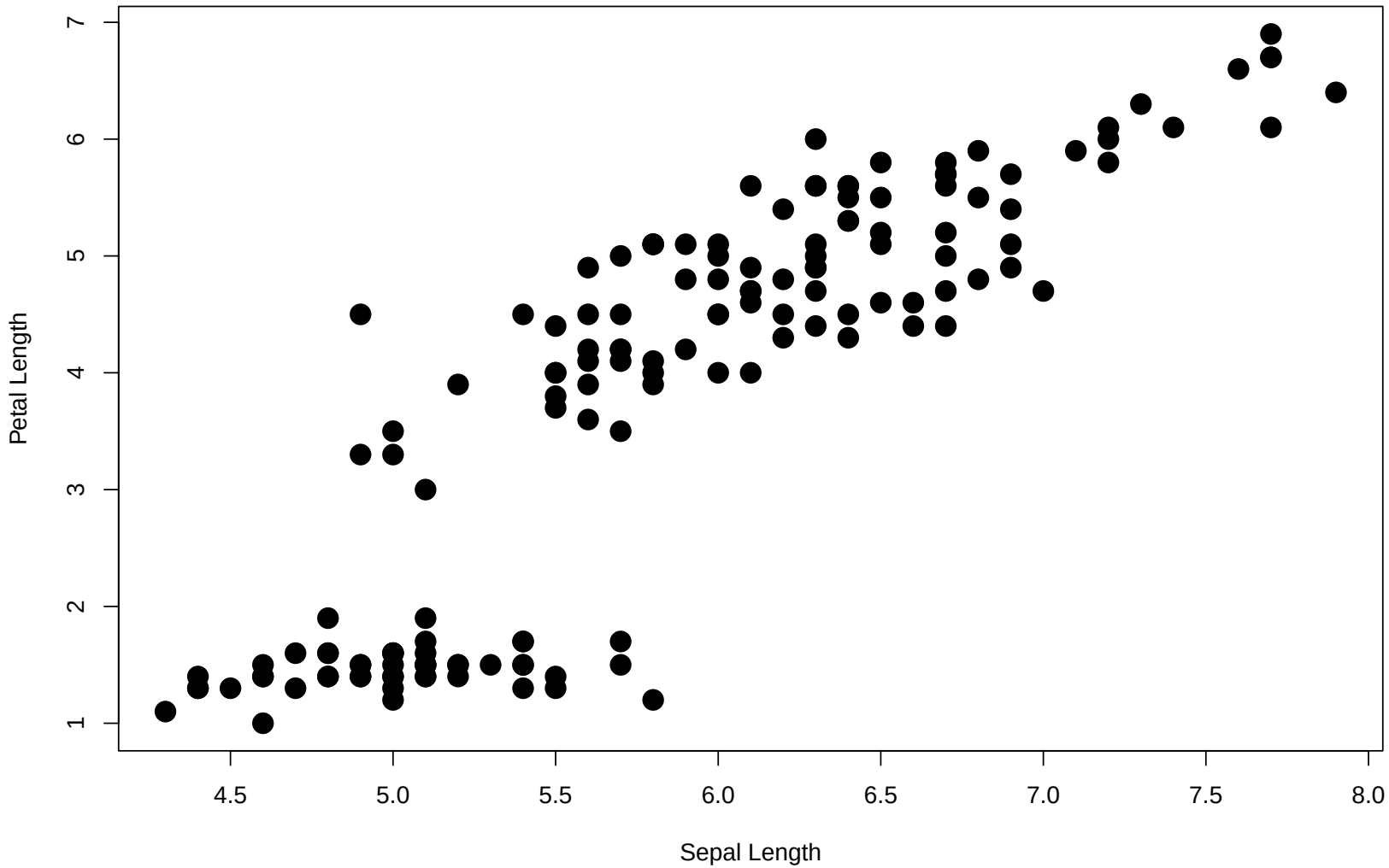
Scatter Plot

```
plot(x, y, xlab = "Sepal Length", ylab = "Petal Length", po
```



Scatter Plot

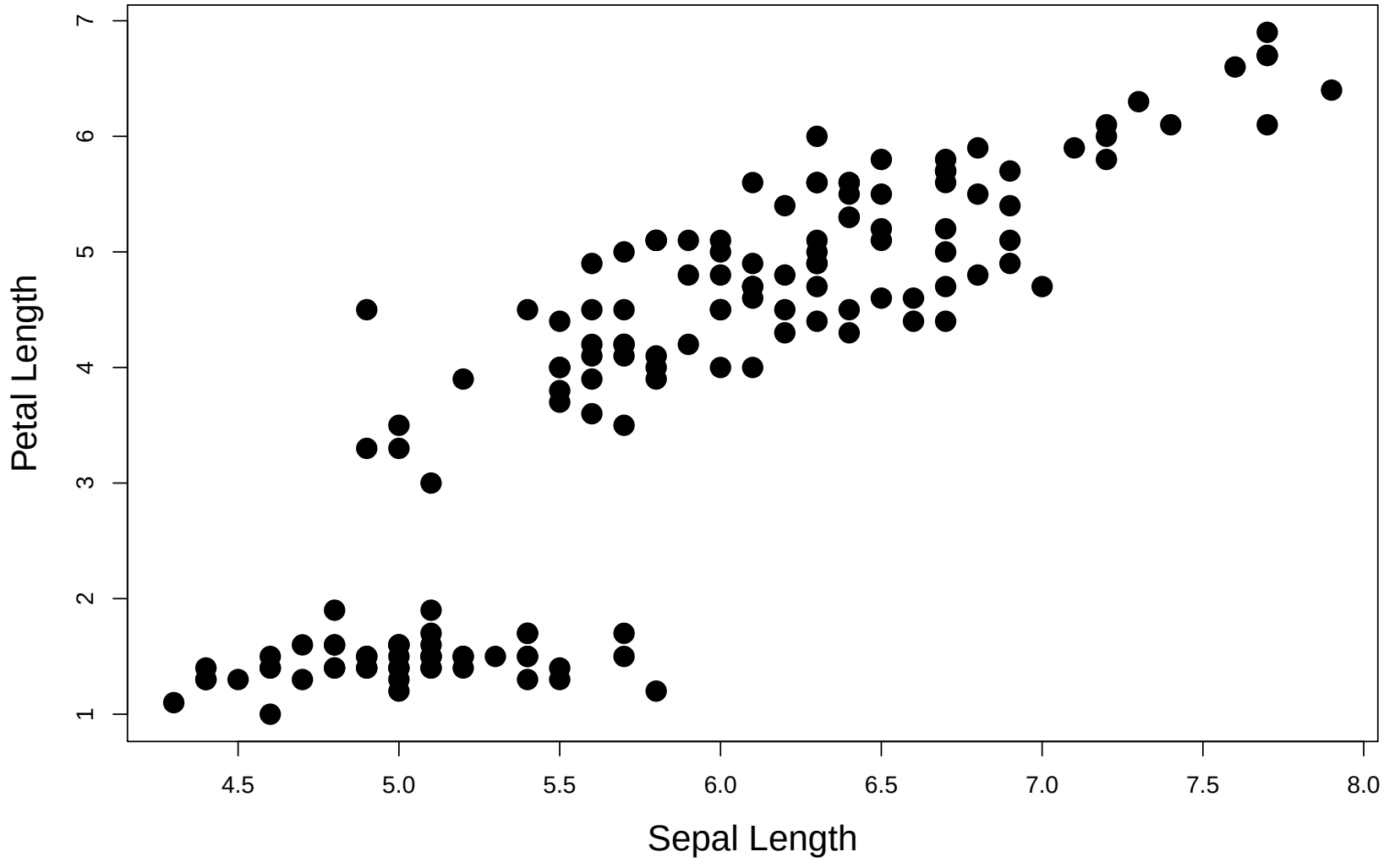
```
plot(x, y, xlab = "Sepal Length", ylab = "Petal Length", po
```



Scatter Plot

```
plot(x,
     y,
     xlab = "Sepal Length",
     ylab = "Petal Length",
     pch = 16,
     cex = 2,
     cex.lab = 1.5)
```

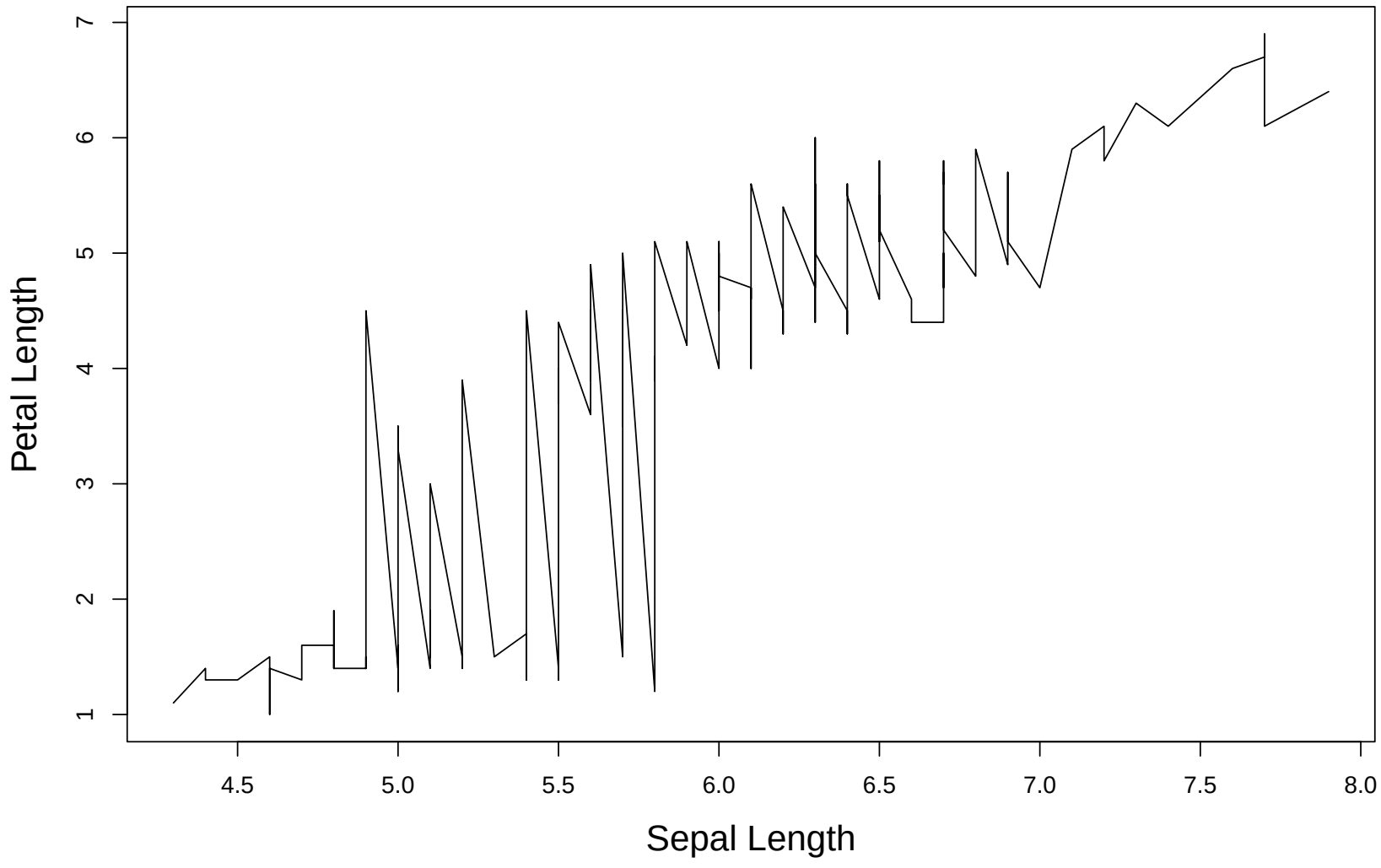
Scatter Plot



Line Graph

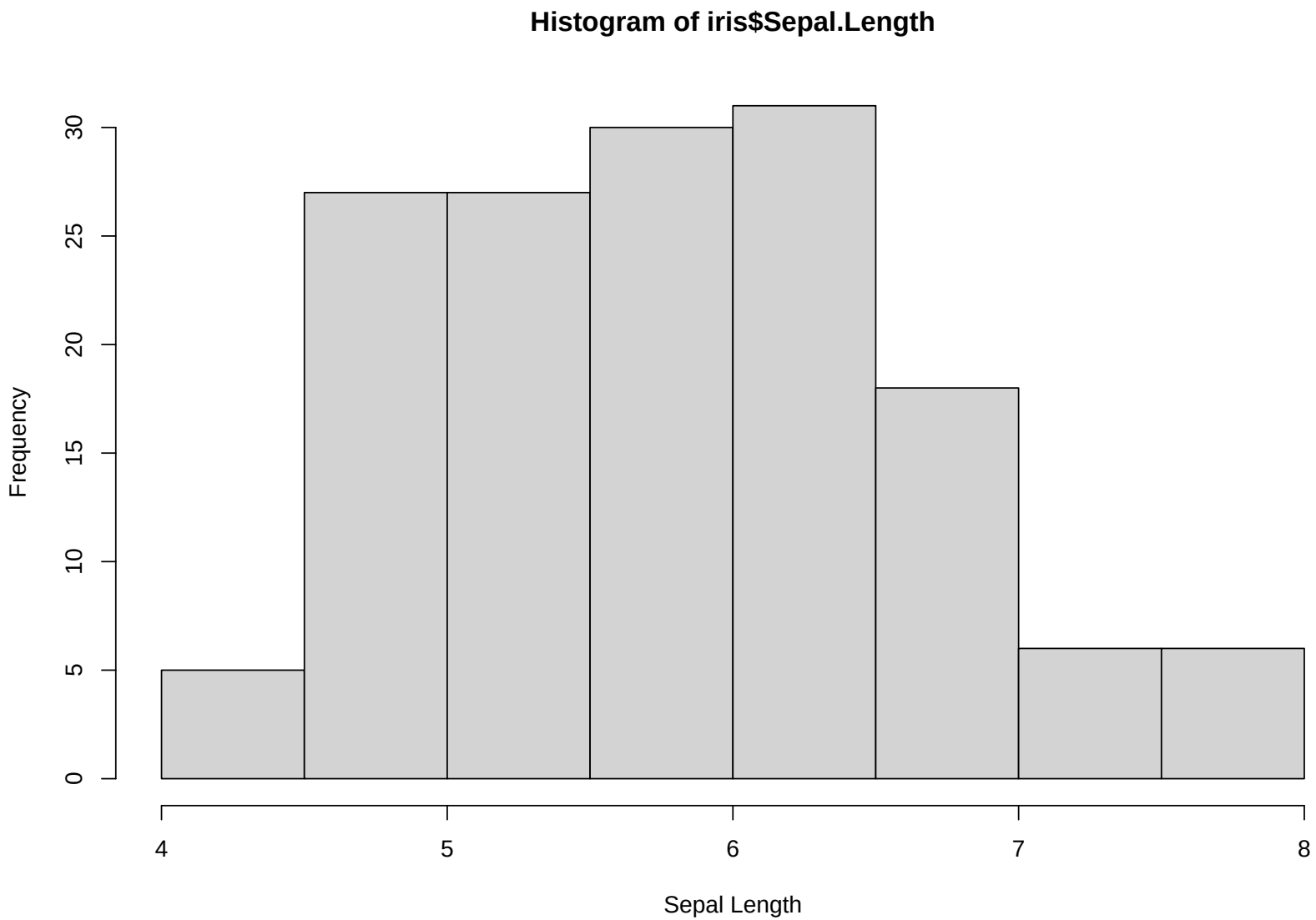
```
plot(x,
     y,
     xlab = "Sepal Length",
     ylab = "Petal Length",
     pch = 16,
     cex = 2,
     cex.lab = 1.5,
     type = "l")
```

Line Graph



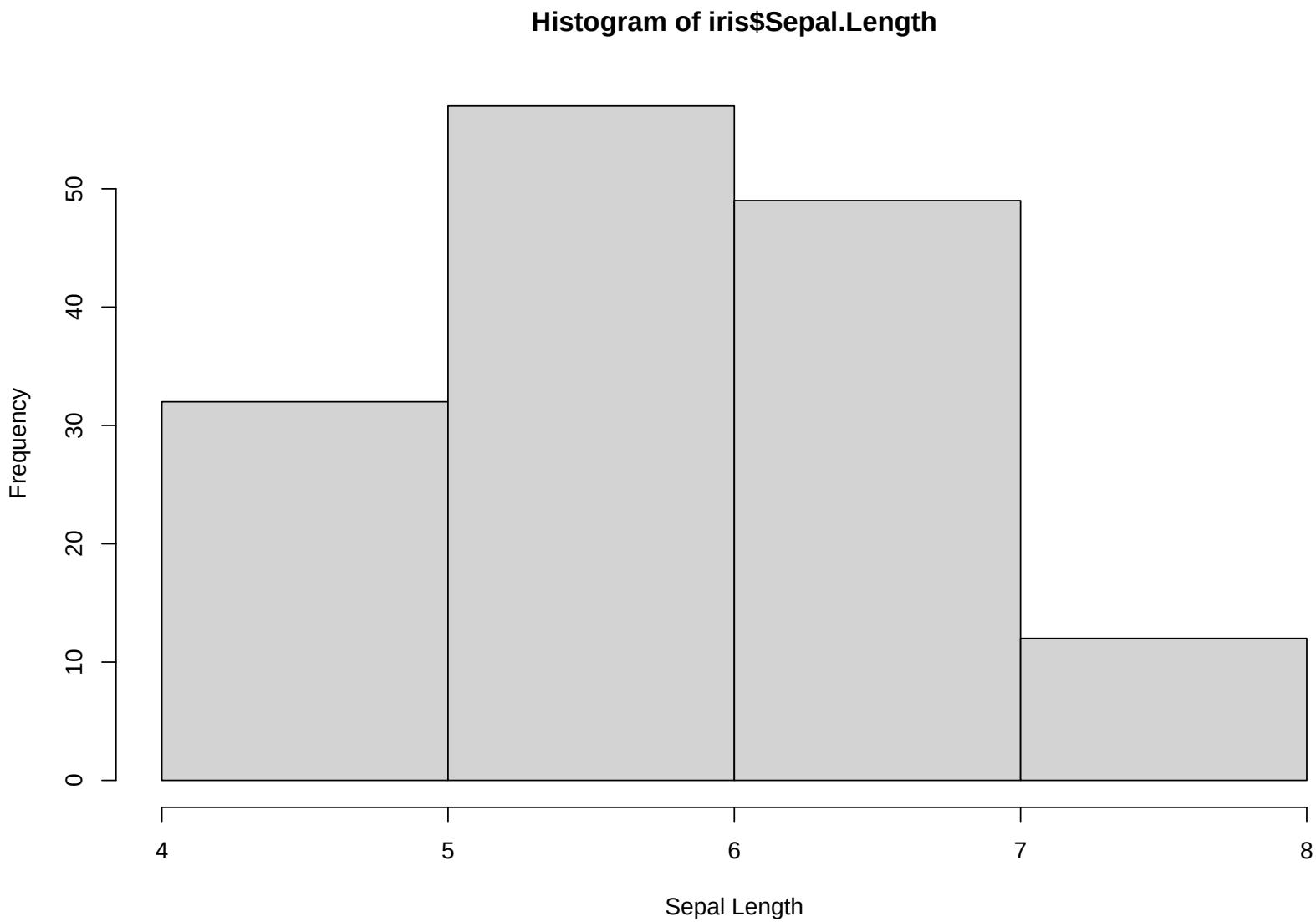
Histogram

```
hist(iris$Sepal.Length, xlab = "Sepal Length")
```



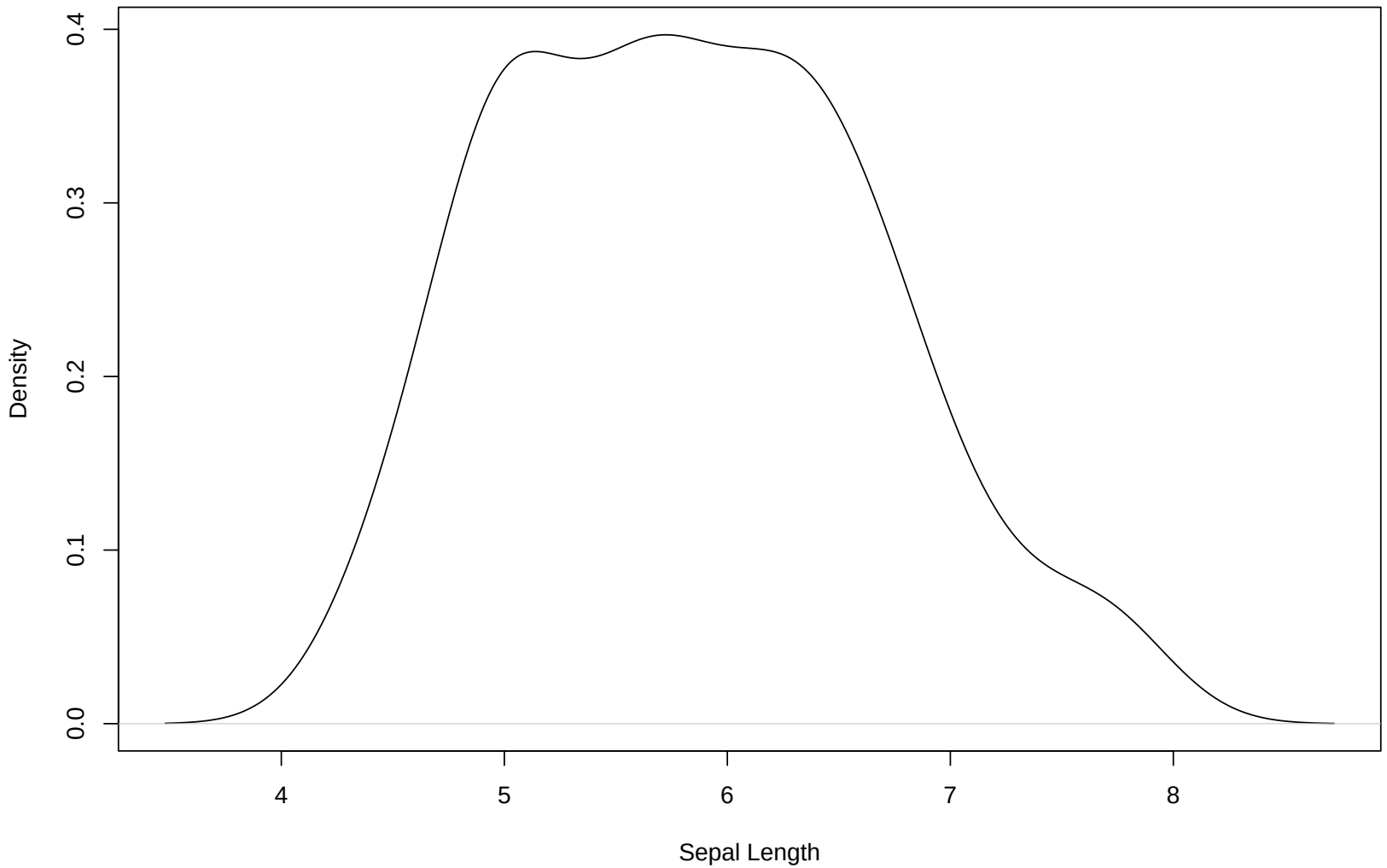
Histogram

```
hist(iris$Sepal.Length, xlab = "Sepal Length", breaks = 5)
```



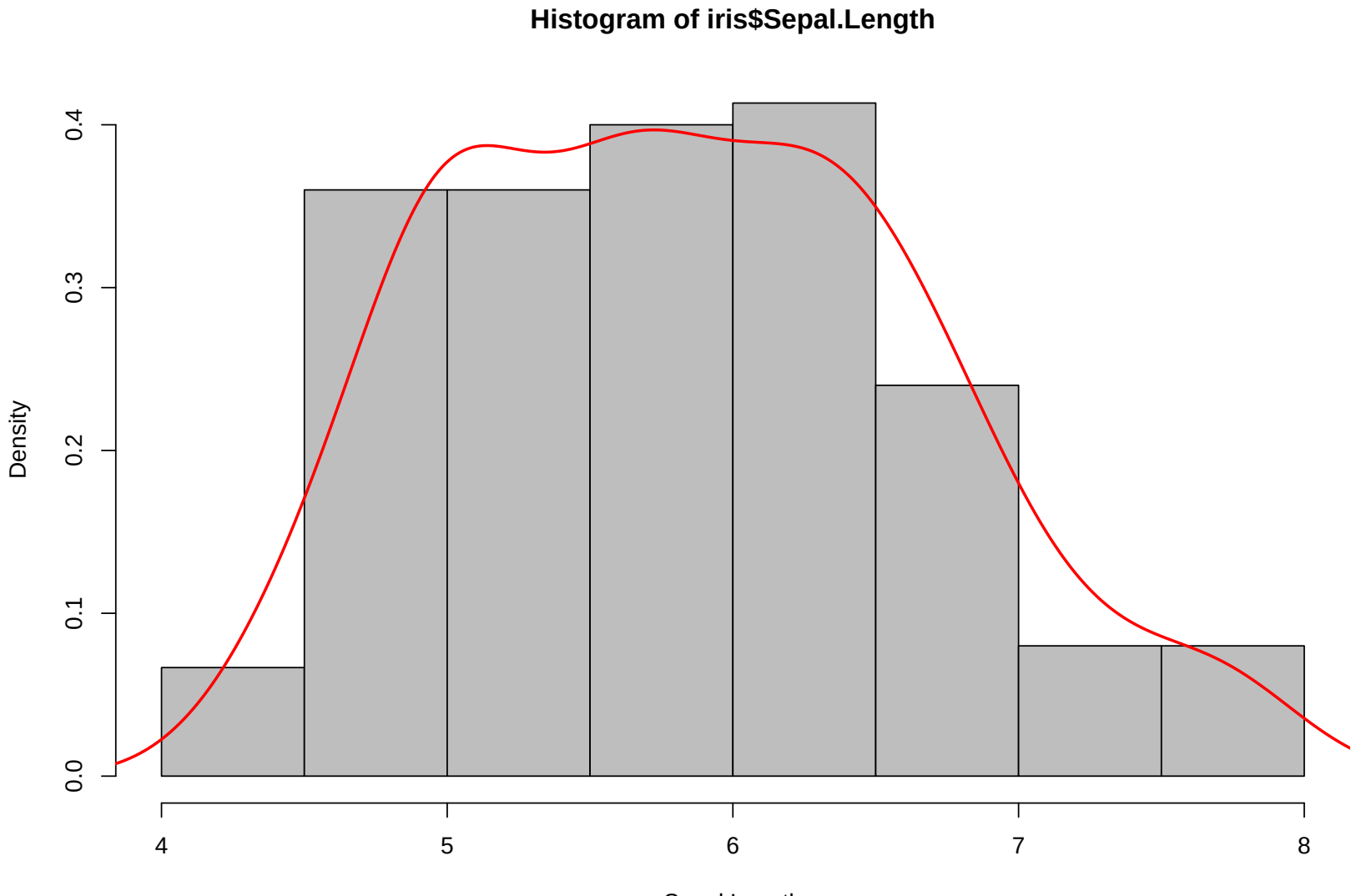
Histogram

```
plot(density(iris$Sepal.Length), xlab = "Sepal Length", ma
```



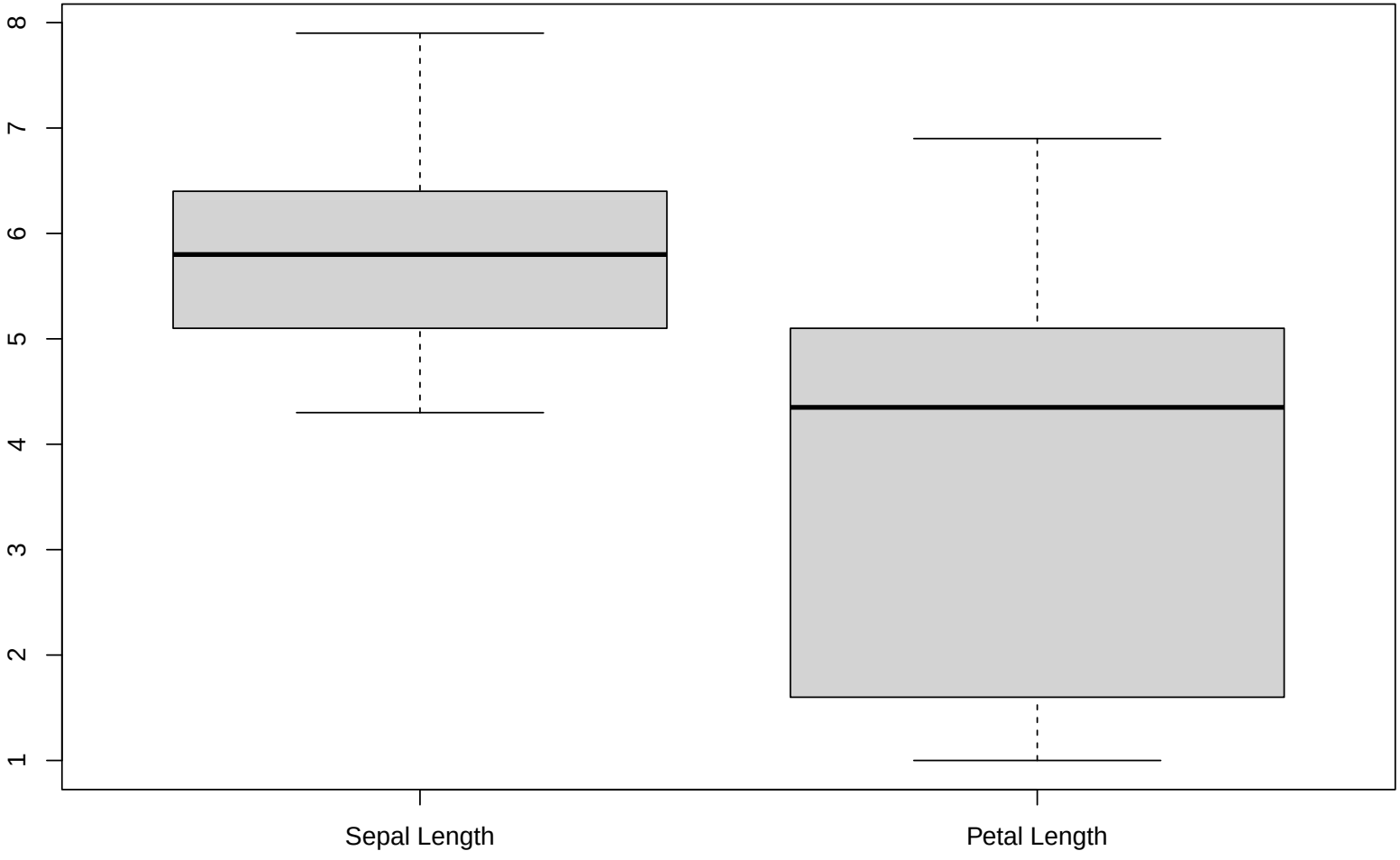
Histogram

```
hist(iris$Sepal.Length, xlab = "Sepal Length", probability
lines(density(iris$Sepal.Length), col =  "red", lwd = 2)
```



Box Plot

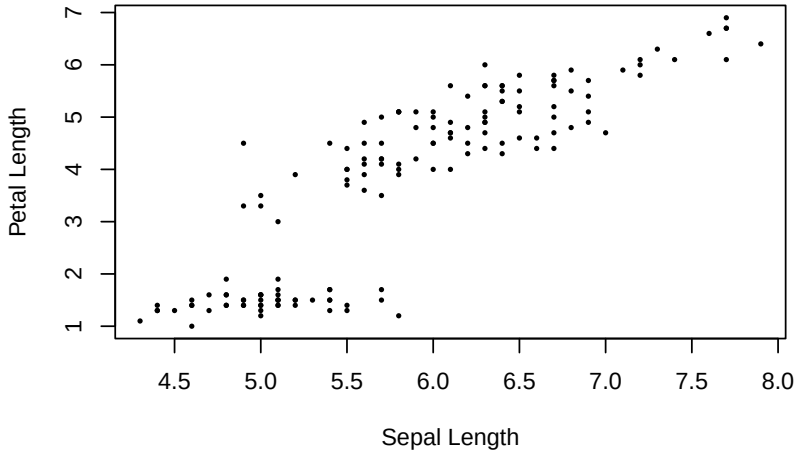
```
boxplot(iris$Sepal.Length, iris$Petal.Length, names = c("Se
```

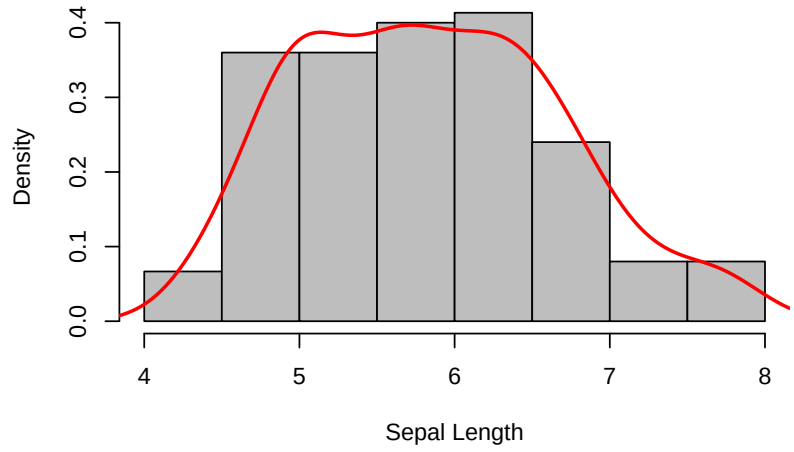


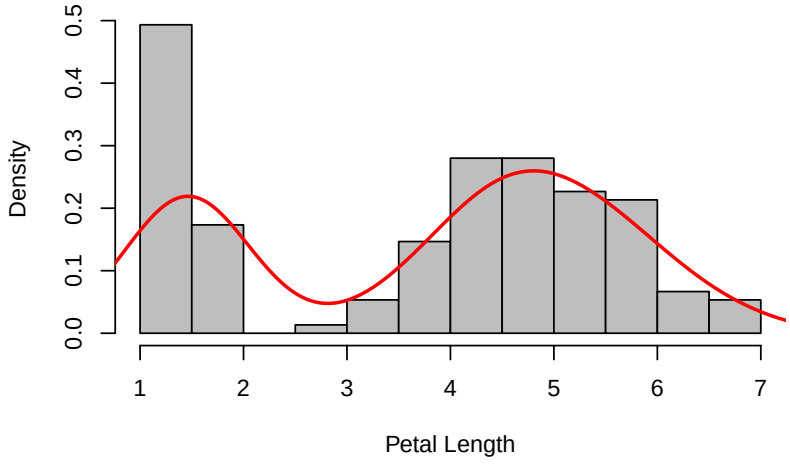
Birden çok grafiği birleştirmek

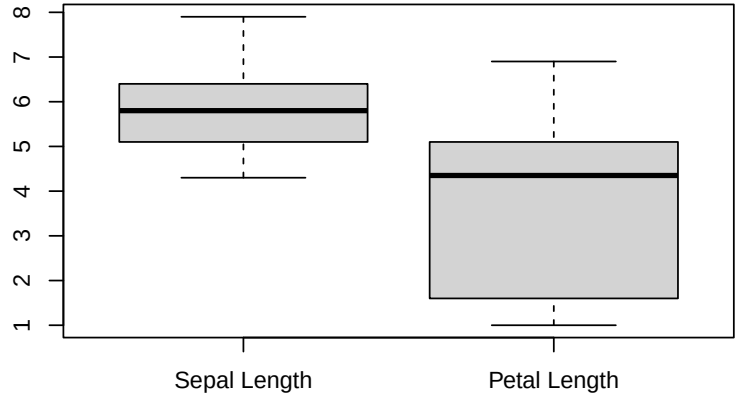
```
par(mfrow = c(2,2)) # mfcol() da kullanabilirsiniz.

plot(x,
     y,
     xlab = "Sepal Length",
     ylab = "Petal Length",
     pch = 16,
     cex = 0.5)

hist(x, xlab = "Sepal Length", probability = T, col = "grey
lines(density(x), col =  "red", lwd = 2)

hist(y, xlab = "Petal Length", probability = T, col = "grey
lines(density(y), col =  "red", lwd = 2)

boxplot(x, y, names = c("Sepal Length", "Petal Length"))
```

Birden çok grafiği birleştirmek





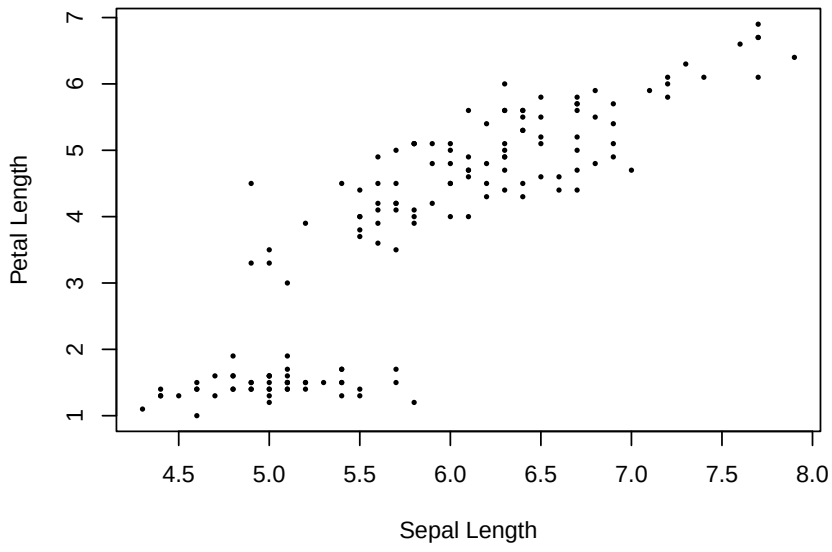


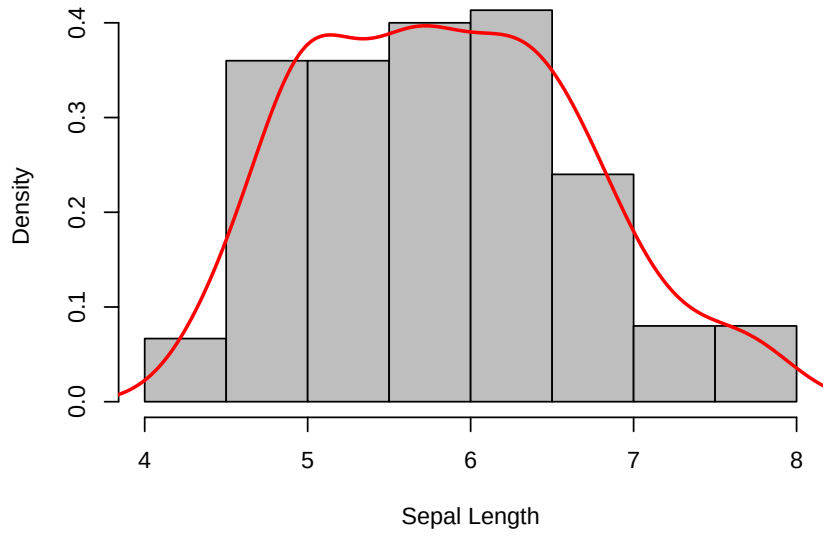


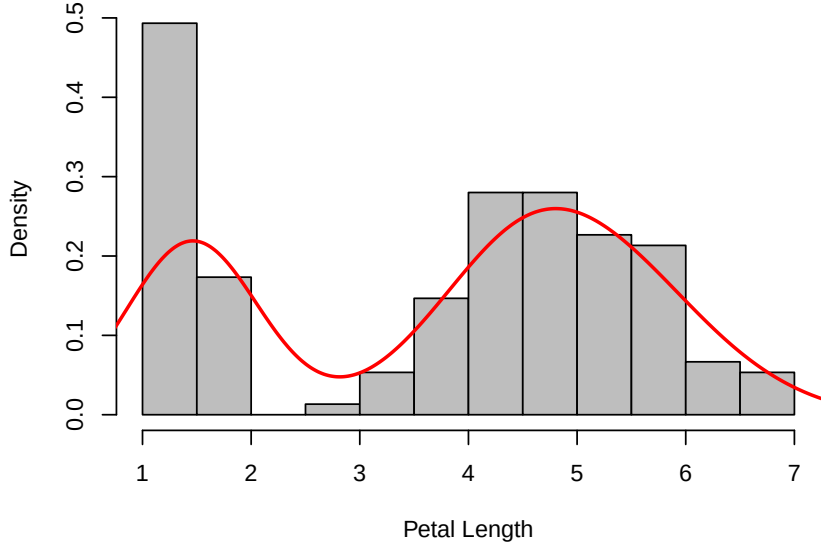
Birden çok grafiği birleştirmek

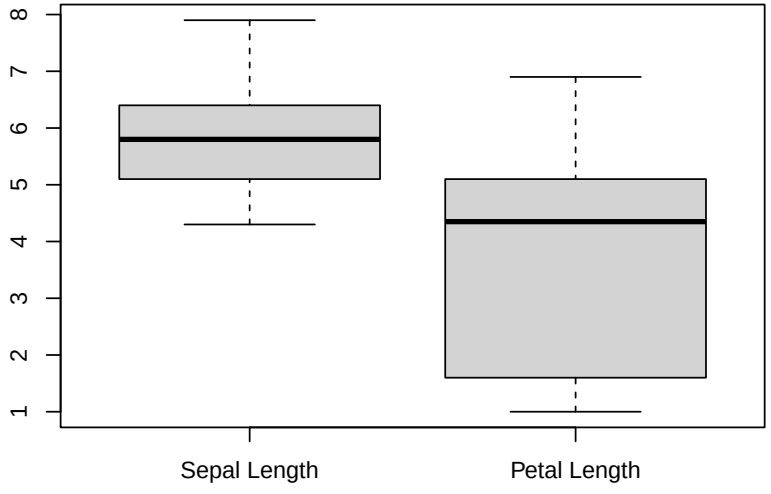
```
par(mfrow = c(2,2)) # mfcol() da kullanabilirsiniz.

par(mar = c(5,4,1,1)) # varsayılan değerler c(5, 4, 4, 2)
plot(x, y, xlab = "Sepal Length", ylab = "Petal Length",
     pch = 16,
     cex = 0.5)

par(mar = c(5,4,1,1))
hist(x, xlab = "Sepal Length", probability = T,
     col = "grey",
     main = "")
lines(density(x), col =  "red", lwd = 2)

hist(y, xlab = "Petal Length", probability = T, col = "grey
lines(density(y), col =  "red", lwd = 2)

boxplot(x, y, names = c("Sepal Length", "Petal Length"))
```

Birden çok grafiği birleştirmek









Sonsöz

"To understand computations in R, two slogans are helpful:

Everything that exists is an object.

Everything that happens is a function call."
— John Chambers

Objepction!

- Operatörler bile bir fonksiyon ve dolayısıyla obje!

```
`+`(2,3)
```

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

```
`*`(10,10)
```

```
## [1] 100
```

```
`[`(x,1)
```

```
## [1] 4.3
```

```
`[`
```

```
## .Primitive("[")
```