📊 Fonksiyonlar

👨‍🏫 Dr. Kübra Atalay Kabasakal

Bahar 2023

Fonksiyon Nedir-1

- S dilinin kurucusu ve R çekirdek ekibinin üyesi John Chambers, R yazılımında var olan herşeyin nesneler olduğunu; nesnelerin ise bir fonksiyonun sonucunda elde edildiğini belirmiştir.
- R'da her fonksiyonun kendine özgü argümanları bulunur.
- Örneğinin `mean()` fonksiyonunun üç tane argümanı bulunmaktadır.

Fonksiyon Nedir-2

- Argümanları sırası, fonksiyonda yer alan sırada kullandığınızda isimleri belirtmenize gerek kalmaz.

```
vec1 <- sample(1:1000,30)
mean(vec1, 0.1, TRUE)
```

```
## [1] 402.5833
```

- Argümanların adını kullandığınız sürece ise istediğiniz sırada kullanabilirsiniz.

```
mean(na.rm= TRUE, trim=0.1,x=vec1)
```

```
## [1] 402.5833
```

Fonksiyon Nedir-3

- Yazılımının temel paketinde veya özel durumlarda kullanılmak üzere geliştirilen paketlerde çok sayıda fonksiyon bulunmaktadır.
- Örneğin, iki vektör arasındaki korelasyon hesaplaması için temel pakette `cor()` fonksiyonu, Hmisc paketinde ise `rcorr()` fonksiyonu kullanılmaktadır.
- Ancak `rcorr()` fonksiyonu sadece matris ve vektörlerde çalışırken, `cor()` fonksiyonu matris, vektör ve veri setlerinde çalışabilmektedir

Neden Fonksiyon Yazarız-1

- R'da uzmanlaştıkça ve yapılan işler karmaşıklaştıkça fonksiyon yazma ihtiyacı duyulmaktadır.
- Fonksiyon yazma gereksinimi özellikle tekrarlı işlemler yapılması gerektiği durumda ortaya çıkmaktadır.
- Fonksiyon yazmak
  - pratiklik kazandırır (ekonomiktir)
  - Paylaşılmasını koylaştırır.
  - Tekrar kullanılabilirlik sağlar.

Neden Fonksiyon Yazarız-2

- Örneğin farklı ölçeklerde yer alan 6 farklı sınava ait sonuçların bir veri seti olarak elinizde olduğunu düşünün.

- Farklı ölçeklerde yer alan sınav sonuçlarını karşılaştımak amacıyla **aritmetik ortalamayı ve standart sapmayı içerisinde barındıran puan dağılımı hakkında daha çok bilgi veren bağıl değişkenlik katsayısı** kullanılabilir.

Neden Fonksiyon Yazarız-3

- 6 farklı sınav sonucunu içeren veri setinin okunması

```
library(readr)
df <- read_csv("df.csv")
```

- 6 farklı sınav için bağıl değişkenlik katsayılarının hesaplanması

```
BDK1 <- (sd(df$S1)/mean(df$S1)) *100
BDK2 <- (sd(df$S2)/mean(df$S2)) *100
BDK3 <- (sd(df$S3)/mean(df$S2)) *100
BDK4 <- (sd(df$S4)/mean(df$S4)) *100
BDK5 <- (sd(df$S5)/mean(df$S5)) *100
BDK6 <- (sd(df$S6)/mean(df$S6)) *100
```

Neden Fonksiyon Yazarız-4

- Tekrarlı işlemlerde bu tarz hatalardan kurtulmanın yolu fonksiyon kullanmaktır.

- Fonksiyonlar, koşullu önermeler ve döngüler ile kullanılarak çok sayıda komut ile yapılabilecek olan işlemler tek bir komut satırı ile yapılabilir hale gelmektedir.

Fonksiyonun Yapısı

- Fonksiyon, matematiksel tanımı itibariyle bir tanım kümesindeki her bir elamanın değer kümesindeki bir eleman ile eşleyen operatördür.
- Yani girdi elemanları ile çıktı elemanları arasında bir köprü görevi üstlenmektedir. Örneğin,

$$f(x)=x+1$$

```
fonksiyon_adı <- # baslık
function(argüman){ #arguman
return(gövde) #cıktı
}
```

```
F1 <- function(x) {
(x+1)
}
girdi <- c(2,5,6,7,12)
cikti <- F1(girdi)
cikti
```

```
## [1]  3  6  7  8 13
```

Fonksiyona İsim Verme

- Dikkatinizi çekmiştir, fonksiyon adları genelde bir eylemi yansıtır.
  - `select`, `filter`, `draw`
- Bunun haricinde kısaltmaların kullanıldığı fonksiyonlar da bulunmaktadır.
  - `lm()`
- Kullanışlı bir fonksiyon yazmak için mümkün olduğunca kısa isimler kullanılmalıdır; bununla birlikte bu isimler kullanıcıya yapılacak işlemi anlaşılır kılmalıdır.
- Bunun yanında R'da özel anlamı olan **c,C,D,F,I,q,t,T** gibi tek harflik fonksiyon isimleri kullanmaktan ve R'da hazır olan fonksiyon isimlerini kişisel tanımlı fonksiyonlara vermekten kaçınılmalıdır.

Argümanlar

- R yazılımında hazır olan bir çok fonksiyonda hem zorunlu hem da ek argümanlar bulunmaktadır.
- Örneğin ortalama hesaplayan `mean()` fonskiyonun bir adet zorunlu iki adet ek argümanı bulunmaktadır. Kullanımı

```
mean(x,trim=0,na.rm=FALSE)
```

- şeklinde olan fonksiyonda `trim` argümanının olağan değeri 0, `na.rm` argümanının ise olağan değeri *FALSE*'dur.

```
args(read.csv)
```

```
## function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
##     fill = TRUE, comment.char = "", ...)
## NULL
```

Argümanlar

- Fonksiyona eklenen argümanların hem argüman kısmında tanımlanması hem de gövde kısmında işlevininin belirtilmesi gerekmektedir. Aksi durumda fonksiyonda beklenmedik çıktılar elde edilebilir.

- üs almak amacıyla fonksiyon yazılması, `us` argümanın değerinin 2 olarak belirlenmesi

```
us_alma <- function(x,us=2){
x^us
}
# Fonksiyonun olağan değeri ile çalıstırılması
us_alma(2) # ikinci dereceden üs alınır.
```

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

```
us_alma(2,5)
```

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

Gövde

- Fonksiyondaki kodların/işlemlerin olduğu kısımdır.
- Fonksiyonun gövde kısmı başlıktan sonra gelen ve **{ }** arasında bulunan kısmıdır. Gövde fonksiyonun yapması gereken işlemin tanımlandığı alandır. Bu işlemler bir veya birden fazla da olabilmektedir.
- Fonksiyonun çıktısı çalışan **en son komutun sonucu** ya da `return` fonksiyonundaki değerdir.

Gövde

- Fonksiyonun gövdesi `body(fonksiyonadı)`
- Fonksiyonun argümanlari `formals(fonksiyonadı)`
-

```
#us_alma() fonksiyonun gövdesinin incelenmesi
body(us_alma)
```

```
## {
##     x^us
## }
```

return

- Fonksiyondaki kodların/işlemlerin olduğu kısımdır.
- Fonksiyonlar bir ve birden fazla argümanla tanımlabileceği gibi, çıktıları da bir ya da birden fazla olabilir. Örneğin küpün taban alanı, tüm alanı ve hacmini veren bir fonksiyon yazılabilir.
- çıktıyı vektör, veri seti ya da liste olarak belirlemek ve değişkenlere isim vermek kolaylıkla yapılabilmektedir.*
- küpün taban alanı, tüm alanı ve hacmini veren bir fonksiyon

```
kup_ozellik <- function(){
}
```

Environment

- f(x,y,z) şeklinde yazılan bir fonksiyon üç argümandan oluşmaktadır.
- Bir fonksiyonun son kısmı olan çevre ise nesnelerin (fonksiyon, değişken vb) toplandığı bir alan olarak düşünülebilir.
- Yapılan tüm işlemler bu çevre üzerine kaydedilir. `environment()` komutu hangi çevre üzerinde çalışıldığının bilgisini verir. Halihazırda bulunan çevre ise R_GlobalEnv global çevresine kayıt edilmektedir.

Çalışma Alanı

Fonksiyonlar, kendi çalışma alanını oluştururlar!!!

```
# İfadenin bir nesneye tanımlanarak fonksiyon oluşturulması
# Tek argümanlı fonksiyon örneği
kup_hacim <- function(x){
hacim <- x^3 #hacimin tanımlanması
return(hacim)
}
# Kenar uzunluğu 3cm olan kupün hacmi
kup_hacim(3)
```

```
## [1] 27
```

```
#İfadenin bir nesneye tanımlamadan fonksiyon oluşturulması
kup_hacim <- function(x){
return(x^3)   #hacimin tanımlanması
}
```

Çalışma Alanı

```
kup_hacim <- function(x){
hacim <- x^3 #hacimin tanımlanması
return(hacim)
}
body(kup_hacim) # Fonsiyonun gövdesi
```

```
## {
##     hacim <- x^3
##     return(hacim)
## }
```

```
formals(kup_hacim) # Fonsiyonun argümanlarının listesi
```

```
## $x
```

```
environment(kup_hacim) # Fonsiyonun çevresinin belirlenmesi
```

```
## <environment: 0x0000028099aaefe8>
## attr(,"handlers")
## attr(,"handlers")[[1]]
## attr(,"handlers")[[1]]$expr
```

Yazım Aşamaları

Fonksiyon yazmak kadar iyi bir fonksiyon yazmak da önemlidir.

- İyi bir fonksiyonun ilk özelliği doğru sonucu veriyor olmasıdır.
- Bunu sağlayabilmek için fonksiyon yazmadan önce problemi iyi tanımlamak ve problemin çözümünü komut satırları ile yazmak daha sonra bunu fonksiyona dönüştürmek gereklidir.
- Bir fonksiyonun doğru sonucu vermesi kadar diğer kullanıcılar tarafından anlaşılır olması da önemlidir.
  - Önce bir taslak oluşturun.
  - Taslağınızı içine komut satırlarınıza yapıştırın
  - Fonksiyonun argümanları belirleyin
  - Argüman isimlerinizi kullanacağınız değişkenlerle değiştirin.

Fonksiyonlara Mesaj Ekleme

Bazı fonksiyonlar bazı durumlarda `stop()` fonksiyonu ile durdurabilir, bazı durumlarda `message()`, `print()`, `cat()` fonksiyonu ile kullanıcıya mesaj verilebilir. Ayrıca **assertive** paketi ile de uyarı mesajları sağlanabilir.

```
library(assertive)
 bolme <- function(x){
  assert_is_numeric(x)
  1/x }
  bolme(3)
```

```
## [1] 0.3333333
```

Örnek_Adım_1

Öğrencileri sözlüye kaldırmak için random seçen fonksiyon yazma

```
ogrenci <- c("ARIF","ASLI","ATA",
             "AYSE","BURCAK",
              "CAGATAY","EMRE","FEYZI",
              "FURKAN", "HARUN","KORKUT",
              "MEHMET","RAMAZAN",
              "SEMIH","SINEM")
sample(ogrenci,1)
```

```
## [1] "MEHMET"
```

Örnek_Adım_2

Önce bir taslak oluşturun.

```
# taslak hazırlama
random_secici <- function() {
### burası fonksiyon kodlarının yazılacagı alan
}
```

Örnek_Adım_3

Daha önce yaptığınız işlemleri taslağa yapıştırın.

```
random_secici <- function() {
ogrenci <- c("ARIF","ASLI","ATA",
             "AYSE","BURCAK",
              "CAGATAY","EMRE","FEYZI",
              "FURKAN", "HARUN","KORKUT",
              "MEHMET","RAMAZAN",
              "SEMIH","SINEM")
sample(ogrenci,1)
}
```

Örnek_Adım_3

Fonksiyonu çalıştırın.

```
random_secici()
```

```
## [1] "ARIF"
```

Örnek_Adım_4

Bu fonksiyonun hangi argümanlara ihtiyacı var onu düşünün.

```
random_secici <- function() {
ogrenci <- c("ARIF","ASLI","ATA",
             "AYSE","BURCAK",
              "CAGATAY","EMRE","FEYZI",
              "FURKAN", "HARUN","KORKUT",
              "MEHMET","RAMAZAN",
              "SEMIH","SINEM")
sample(ogrenci,1)
}
```

Örnek_Adım_5

Bu fonksiyonun hangi argümanlara ihtiyacı var onu düşünün.

```
random_secici <- function(x,n) {
sample(x,n)
  }
```

Fonksiyonu ilk yazdığınız özel durum için çalıştırma.

```
random_secici <- function(x,n) {
sample(x,n)
  }
random_secici(ogrenci,1)
```

```
## [1] "SINEM"
```

Fonksiyon Okuma

`harf_not()` fonksiyonunun işlevini açıklayabilir misiniz?

```
harf_not <- function(x, n, na.rm, labels, interval_type) {
      probs <- seq(0, 1, length.out = n + 1)
      qtiles <- quantile(x, probs, na.rm = na.rm, names = FALSE)
      right <- switch(interval_type, "(lo, hi]" = TRUE, "[lo, hi)" = FALSE)
cut(x, qtiles, labels = labels, right = right, include.lowest = TRUE)
}
```

Default argumanlar-1

`harf_not()` fonksiyonunun n argümanını default olarak tanımlayınız. Fonksiyonu buna göre yeniden düzenleyip çalıştırınız.

```
harf_not <- function(x, n, na.rm, labels, interval_type) {
      probs <- seq(0, 1, length.out = n + 1)
      qtiles <- quantile(x, probs, na.rm = na.rm, names = FALSE)
      right <- switch(interval_type, "(lo, hi]" = TRUE, "[lo, hi)" = FALSE)
cut(x, qtiles, labels = labels, right = right, include.lowest = TRUE)
}
```

Default argumanlar-1

```
harf_not(
  x,
  n,
  na.rm,
  labels = c("very low", "low", "medium", "high", "very high"),
  interval_type = "(lo, hi]"
)
```

```
## Error in n + 1: non-numeric argument to binary operator
```

Default argumanlar-2

`harf_not()` fonksiyonunun `na.rm` argümanını default olarak tanımlayınız. Fonksiyonu buna göre yeniden düzenleyip çalıştırınız.

```
harf_not <- function(x, n, na.rm, labels, interval_type) {
      probs <- seq(0, 1, length.out = n + 1)
      qtiles <- quantile(x, probs, na.rm = na.rm, names = FALSE)
      right <- switch(interval_type, "(lo, hi]" = TRUE, "[lo, hi)" = FALSE)
cut(x, qtiles, labels = labels, right = right, include.lowest = TRUE)
}
```

Default argumanlar-3

`harf_not()` fonksiyonunun labels argumanını NULL olarak tanımlayınız. Fonksiyonu buna göre yeniden düzenleyip çalıştırınız.

```
x <- sample(0:100,30)

harf_not <- function(x, n = 5, na.rm = FALSE, labels, interval_type) {
      probs <- seq(0, 1, length.out = n + 1)
      qtiles <- quantile(x, probs, na.rm = na.rm, names = FALSE)
      right <- switch(interval_type, "(lo, hi]" = TRUE, "[lo, hi)" = FALSE)
      cut(x, qtiles, labels = labels, right = right, include.lowest = TRUE)
}

harf_not(
  x,
  labels = c("F", "D", "C", "B", "A"),
  interval_type = "(lo, hi]"
)
```

```
##  [1] F C F C A D C F A B B C D F B C F D D C A A D D F A
## [27] B B B A
## Levels: F D C B A
```

Default argumanlar-4

`harf_not()` fonksiyonunun interval type değerleri de argümanın içinde tanımlanırsa daha kullanışlı olur.

```
x <- sample(0:100,30)
harf_not <- function(x, n = 5, na.rm = FALSE, labels = NULL,
                             interval_type = c("(lo, hi]", "[lo, hi)")) {
  interval_type <- match.arg(interval_type)
  probs <- seq(0, 1, length.out = n + 1)
  qtiles <- quantile(x, probs, na.rm = na.rm, names = FALSE)
  right <- switch(interval_type, "(lo, hi]" = TRUE, "[lo, hi)" = FALSE)
  cut(x, qtiles, labels = labels, right = right, include.lowest = TRUE)
}

harf_not(x)
```

```
##  [1] (62.2,83.8] (19.6,34.4] (19.6,34.4] [5,19.6]
##  [5] (34.4,62.2] (83.8,100]  (83.8,100]  (83.8,100]
##  [9] (19.6,34.4] (62.2,83.8] (83.8,100]  (62.2,83.8]
## [13] (19.6,34.4] [5,19.6]    [5,19.6]    (34.4,62.2]
## [17] (62.2,83.8] (34.4,62.2] (62.2,83.8] (19.6,34.4]
## [21] (19.6,34.4] (83.8,100]  (34.4,62.2] (34.4,62.2]
## [25] (34.4,62.2] [5,19.6]    [5,19.6]    (83.8,100]
## [29] [5,19.6]    (62.2,83.8]
```

Fonskiyon Yazma -Harmonik ortalama

- Harmonik ortalama, gözlem sonuçlarının (birim değerlerinin) terslerinin aritmetik ortalamasının tersidir. Özetle oranların ortalamasıdır. .xsmall[- Birim değerleri x_1, x_2, ... , x_n gibi gösterilirse harmonik ortalama aşağıdaki gibi yazılı

$$\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\ldots+\frac{1}{x_n}}$$

formülden de anlaşılacağı üzere, işlemler birbirine bağlı gerçekleşmektedir.

```
y <- 1:5
length(y)/sum(1/y) #ortalama işleminin tersi
```

```
## [1] 2.189781
```

```
harmonik_ort <- function() {

}
```

Fonskiyon Yazma -Harmonik ortalama

```
tersal <- function(x) {
    1/x
}
y <- 1:5
# harmonik ortalama hesaplama

y %>% tersal() %>% mean() %>% tersal
```

```
## [1] 2.189781
```

```
harmonik_ort <- function(x) {
    x%>%
    tersal() %>%
    mean() %>%
    tersal

}
harmonik_ort(y)
```

```
## [1] 2.189781
```

Fonskiyon Yazma -Harmonik ortalama

S&P 500 borsa endeksi veri kullanılarak fiyat/kazanç oranı (pe_ratio) değişkenin her bir sektor için ayrı ayrı harmonik ortalamasını hesaplayalım.

```
sp500 <- readRDS("sp500.rds")

sp500 %>%
  group_by(sector) %>%
  summarise(hmean_pe_ratio = harmonik_ort(pe_ratio))
```

```
## # A tibble: 11 × 2
##    sector                 hmean_pe_ratio
##    <chr>                           <dbl>
##  1 Communication Services           NA
##  2 Consumer Discretionary           NA
##  3 Consumer Staples                 NA
##  4 Energy                           NA
##  5 Financials                       NA
##  6 Health Care                      NA
##  7 Industrials                      NA
##  8 Information Technology           NA
##  9 Materials                        NA
## 10 Real Estate                      32.5
## 11 Utilities                        NA
```

Fonskiyon Yazma -Harmonik ortalama

Eksik verileri çıkararak ortalama almak için, fonksiyona default değeri ile eksik veri silme argümanını ekleyelim.

```
harmonik_ort <- function(x,na.rm=FALSE) {
    x%>%
    tersal() %>%
    mean(na.rm=na.rm) %>%
    tersal()

}
sp500 %>%
  group_by(sector) %>%
  summarise(hmean_pe_ratio = harmonik_ort(pe_ratio,na.rm=TRUE))
```

```
## # A tibble: 11 × 2
##    sector                 hmean_pe_ratio
##    <chr>                           <dbl>
##  1 Communication Services           17.5
##  2 Consumer Discretionary           15.2
##  3 Consumer Staples                 19.8
##  4 Energy                           13.7
##  5 Financials                       12.9
##  6 Health Care                      26.6
##  7 Industrials                      18.2
```

Fonskiyon Yazma -Harmonik ortalama

Argüman atlama ... argüman kullanmada esneklik sağlamak için eklebilenecek argümanlar yerine **...** (three dots ellipsis (...)) kullanılabilir.

```
harmonik_ort <- function(x,...) {
    x%>%
    tersal() %>%
    mean(...) %>%
    tersal()

}

sp500 %>%
  group_by(sector) %>%
  summarise(hmean_pe_ratio = harmonik_ort(pe_ratio,na.rm=TRUE))
```

```
## # A tibble: 11 × 2
##    sector                 hmean_pe_ratio
##    <chr>                           <dbl>
##  1 Communication Services           17.5
##  2 Consumer Discretionary           15.2
##  3 Consumer Staples                 19.8
##  4 Energy                           13.7
##  5 Financials                       12.9
##  6 Health Care                      26.6
```

Fonskiyon Yazma - Harmonik ortalama

Kullanıcıların argüman değerlerini yanlış girmesi durumunda,hata mesajları ile uyarı sağlanabilir.

```
library(assertive)

harmonik_ort <- function(x,...) {
    assert_is_numeric(x)
    x%>%
    tersal() %>%
    mean(...) %>%
    tersal()

}

# karakter deger girildiğinde
harmonik_ort(sp500$sector)
```

```
## Error in harmonik_ort(sp500$sector): is_numeric : x is not of class 'numeric'; it has class 'character'
```

Fonskiyon Yazma - Harmonik ortalama

`assert_*()` fonksiyonlari isteilen uyarıyı sağlamadığında, koşulara bağlı olarak geliştirici hata mesajı ekleyebilir.

```
harmonik_ort <- function(x,...) {
    assert_is_numeric(x)
    if(any(is_non_positive(x), na.rm = TRUE)) {
    # Throw an error
    stop("x negatif degerler icermektedir..")
    }
    x%>%
    tersal() %>%
    mean(...) %>%
    tersal()

}

# karakter deger girildiğinde
harmonik_ort(sp500$pe_ratio -50)
```

```
## Error in harmonik_ort(sp500$pe_ratio - 50): x negatif degerler icermektedir..
```

Çoklu çıktılarda düzenleme

```
mod <- lm(mpg ~ wt + qsec, data = mtcars)

str(mod)
```

```
## List of 12
##  $ coefficients : Named num [1:3] 19.746 -5.048 0.929
##   ..- attr(*, "names")= chr [1:3] "(Intercept)" "wt" "qsec"
##  $ residuals    : Named num [1:32] -0.8151 -0.0482 -2.5273 -0.1806 0.5039 ...
##   ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
##  $ effects      : Named num [1:32] -113.65 -29.116 -9.103 0.357 0.503 ...
##   ..- attr(*, "names")= chr [1:32] "(Intercept)" "wt" "qsec" "" ...
##  $ rank         : int 3
##  $ fitted.values: Named num [1:32] 21.8 21 25.3 21.6 18.2 ...
##   ..- attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
##  $ assign       : int [1:3] 0 1 2
##  $ qr           :List of 5
##   ..$ qr   : num [1:32, 1:3] -5.657 0.177 0.177 0.177 0.177 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
##   .. .. ..$ : chr [1:3] "(Intercept)" "wt" "qsec"
##   .. ..- attr(*, "assign")= int [1:3] 0 1 2
##   ..$ qraux: num [1:3] 1.18 1.05 1.08
##   ..$ pivot: int [1:3] 1 2 3
##   ..$ tol  : num 1e-07
```

Çoklu çıktılarda düzenleme

```
library(broom)

list(
  # Get model-level values
  model = glance(mod),
  # Get coefficient-level values
  coefficients = tidy(mod),
  # Get observation-level values
  observations = augment(mod)
)
```

```
## $model
## # A tibble: 1 × 12
##   r.squared adj.r.…¹ sigma stati…²  p.value    df logLik
##       <dbl>    <dbl> <dbl>   <dbl>    <dbl> <dbl>  <dbl>
## 1     0.826    0.814  2.60    69.0 9.39e-12     2  -74.4
## # … with 5 more variables: AIC <dbl>, BIC <dbl>,
## #   deviance <dbl>, df.residual <int>, nobs <int>, and
## #   abbreviated variable names ¹adj.r.squared,
## #   ²statistic
##
## $coefficients
## # A tibble: 3 × 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
```

SORU - 1

- öğrenci vektorunu ilk sütun olduğu bir veri seti olusturunuz. veri setinizin ikinci sütunu ise öğrencilerin ara sinav puanları olsun. Bu değişkeni sample fonkisyonu ile 0-100 arasında olacak şekilde oluşturabilirsiniz.

- Oluşturduğunuz veri setinden öğrencilerin, ara sinav puanlarina göre ağırlandırarak dörder kişilik gruplar seçecek bir fonkisyon yazınız. Fonksiyonunuz kullanıncının veri seti haricinde bir tür girmesi durumunda ve girilen veri setinin ikiden fazla sütun içermesi durumunda çalışmayı durdursun.

SORU - 2

- Geometrik ortalamanın farklı hesaplama yolları bulunmaktadır.
- Logaritma değerlerine dayalı olarak hesaplandığında, geometrik ortalama, gözlem değerlerinin logaritmalarının aritmetik ortalamasıdır.
- Bir x vektorunun geometrik ortalamaasını logartimalara dayalı olarak hesaplayan bir fonsiyon yazıp, x <- 1:100 için çalıştırınız.

Kaynaklar

Atar, B., Atalay Kabasakal, K, Ünsal Özberk, E. B., Özberk, E. H. Ve Kıbrıslıoğlu Uysal, N. (2020).R ile Veri Analizi ve Psikometri Uygulamaları, Editör, Pegem Akademi, Ankara.

Garcia, S. (2012). Introduction to Creating Functions in R [html]. Erişim adresi https://rpubs.com/Sergio_Garcia/introduction_functions_R