Programación científica en R

Programación

Marcos Ehekatzin García Guzmán

Octubre de 2024

- Estructura básica:

```
> Función <- function(x,y,z){
    if(condition){
      #Código a ejecutar conado la condición es TRUE
    } else {
      #Código a ejecutar cuando la condición es FALSE
    }
    return()
  }
```

- También podemos combinar las condiciones con la función `stop()` para que la función se detenga en ciertos casos.

Funciones: Ejercicio

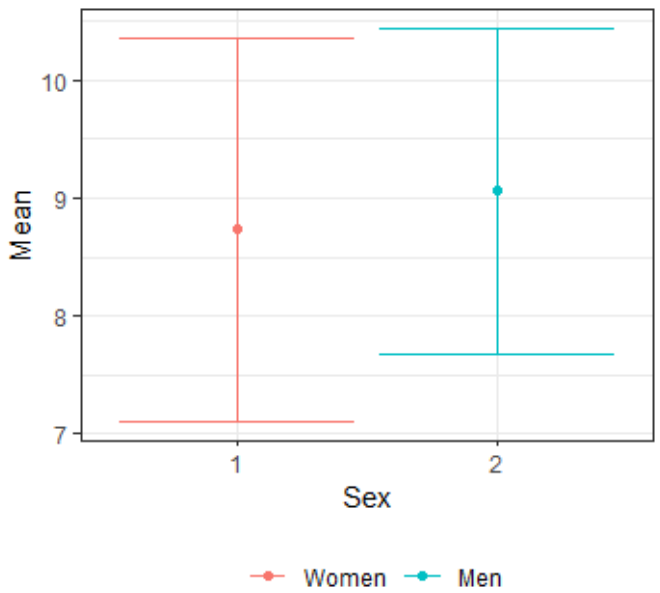
Haga una función con las siguientes características:

1. La función debe calcular la media (ponderada) el error estándar (ponderado) y el intervalo de confianza de la media al 95 %.
2. Como resultado la función nos debe de dar un dataframe de un 1×4, donde las columnas sean: i) La media, ii) El error estándar, iii) El valor máximo del CI y iv) El valor mínimo del CI.

- Para la media ponderada use la función weighted.mean
- Para el error estándar, utilice la fórmula $se = \frac{\sum_{i=1}^{n} w_i(x_i - \bar{X})^2}{(\sum_{i=1}^{n} w_i) - 1}$
- Para el CI utilice la fórmula $\bar{X} \pm 1{,}96 \times se$
- w_i es el ponderador de la observación i del vector x y $\bar{X}$ es la media ponderada de las observaciones del vector x.

Funciones: Repaso

- Cargue la ENOE y aplique la función sobre el logaritmo del ingreso de los hombres y las mujeres por separado.
  - Solo quédese con las observaciones en donde se cumpla que i) tipo==1, ii) ingocup >0, iii) eda>14 & eda <66.
  - Recuerde cambiar los valores del ingreso cuando éste es igual a 999998.
- Replique la siguiente gráfica:



Iteraciones

- En el caso anterior es fácil aplicar la función porque solo lo hacemos dos veces (hombres y mujeres).
- Pero ¿qué pasaría si necesitáramos aplicar la función a hombres y mujeres por separado y para cada decil?
- Eso rompe la regla de dedo: No copiar y pegar más de dos veces el mismo código.
- Para solucionarlo podemos utilizar iteraciones (loops)

Iteraciones

- En R tenemos dos tipos de iteraciones: i) Iteraciones `for` e iteraciones `while`
- Las iteraciones del tipo `for` las ocuparemos cuando sepamos exactamente el número de veces que queremos repetir un proceso.
- Las iteraciones del tipo `while`las usaremos cuando no sepamos exactamente el número de veces que necesitamos repetir un proceso, pero sí sepamos cuál es la condición lógica en donde la iteración tiene que terminar.

Iteraciones: `for`

- El ejemplo más básico de una iteración `for` es la siguiente:

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

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

- Lo que estamos haciendo es equivalente a hacer el siguiente proceso:

```
i<-1
print(i+1)
```

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

```
i<-2
print(i+1)
```

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

```
i<-3
print(i+1)
```

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

- Este proceso sigue hasta que la iteración alcanza el valor

Iteraciones: `for`

- Podemos utilizar estas iteraciones para hacer operaciones en distintos elementos de una lista o diferentes variables de una matriz.
- Por ejemplo, podríamos hacer un `for loop` para obtener la media de las variables ingreso, edad y horas trabajadas en la semana.

```
vars <- list(enoe$lwage, enoe$eda, enoe$hrsocup)

for (i in 1:length(vars)) {
  print(weighted.mean(vars[[i]],enoe$fac_tri))
  }
```

```
## [1] 8.92222
## [1] 38.6263
## [1] 41.1525
```

- Los valores anteriores no se han guardado en ningún lugar.
- Para guardarlos, podemos crear un vector que iremos rellenando.

```
medias <- vector("double", length(vars))
names(medias)<-c("Ingreso", "Edad", "Horas")
for (i in 1:length(vars)) {
  medias[i] <-weighted.mean(vars[[i]],enoe$fac_tri)
  }
medias
```

```
##   Ingreso     Edad    Horas
##   8.92222 38.62630 41.15250
```

- Para hacer más sencilla la secuencia, podemos utilizar la función `sec_along()`, de la siguiente manera:

```
medias <- vector("double", length(vars))
names(medias)<-c("Ingreso", "Edad", "Horas")
for (i in seq_along(vars)) {
  medias[i] <-weighted.mean(vars[[i]],enoe$fac_tri)
  }
```

Iteraciones: `for`

- Al igual que con las funciones, podemos añadir condicionales en una iteración:

```
enoe$entidad <- as.character(enoe$ent)
vars <- list(enoe$lwage, enoe$eda, enoe$hrsocup,
             enoe$entidad)

medias <- vector("double", length(vars))
names(medias) <- c("Ingreso", "Edad","Horas","Entidad")

for (i in seq_along(vars)) {
  if(class(vars[[i]]) == "character"){
    warning("Variable no numérica")
    medias[i] <- NA
  } else{
      medias[i] <-weighted.mean(vars[[i]],enoe$fac_tri)
  }
}
```

Iteraciones: `for`

- Hay veces que no conocemos el largo del output.
- Imaginen que queremos simular algunos vectores con un largo aleatorio y juntar todos los elementos en un solo vector.
- Una posible solución es la siguiente:

```
means <- c(0,1,2)
output <- double()
for (i in seq_along(means)) {
  n<- sample(100,1)
  output <- c(output, rnorm(n,means[i]))
}
length(output)
```

```
## [1] 218
```

- La solución anterior no es eficiente, ya que estamos copiando los datos resultantes de cada iteración.
- Una mejor solución es guardar los resultados en una lista y y después combinarlos en un solo vector cuando terminen las iteraciones.

```
out <- vector("list", length(means))
for (i in seq_along(means)) {
  n<- sample(100,1)
  out[[i]] <- rnorm(n,means[i])
}
out2<-unlist(out)
length(out2)
```

```
## [1] 39
```

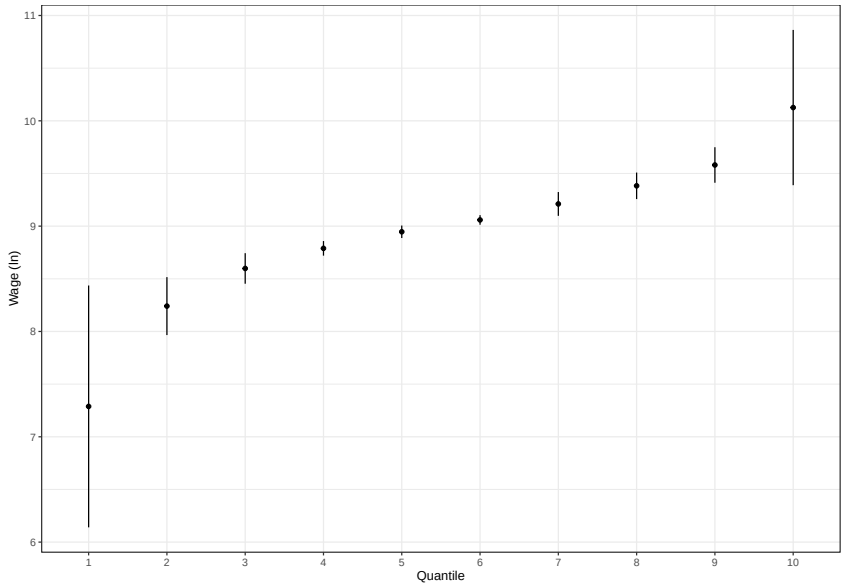
Iteraciones: `for`

- También podemos utilizar iteraciones para guardar nuevos dataframes.
- Por ejemplo, si queremos aplicar nuestra función a distintas variables de la ENOE y guardar los resultados en dataframes distintos.

```
vars<-list(enoe$lwage,enoe$eda,enoe$hrsocup)
for (i in seq_along(vars)) {
  dfName <- paste0("m",i)
  assign(dfName, wtd_mean_se(vars[[i]],enoe$fac_tri))
}
```

Iteraciones: `for`

- Ejercicio: Utilizando una iteración `for` calcule, por decil, el promedio, el error estándar y el CI al 95 % del ingreso en logaritmos.
- Guarde todos los resultados en un solo dataframe.
  - Hint: Utilice la función `bind_rows(lista)`
- Grafique.



Iteraciones: `for`