Presentación

2022-02-12

Contents

Chapter 1

Bienvenidos

Sobre este Proyecto:

Nos propusimos trasncribir al lenguaje R y Python los apéndices del libro "Pobreza y Desigualdad en América Latina" de Gasparini, Cicowiez y Sosa Escudero que originalmente fueron escritos para Stata y que permitían replicar los datos e información presentados por los autores en el texto. Cada capítulo consta de un apéndice con códigos que permiten llevar a la práctica los conceptos desarrollados. Aquí los traducimos a nuevos lenguajes y los presentamos en un formato amigable para permitir que un público más amplio y de diversas disciplinas pueda aprovecharlos.

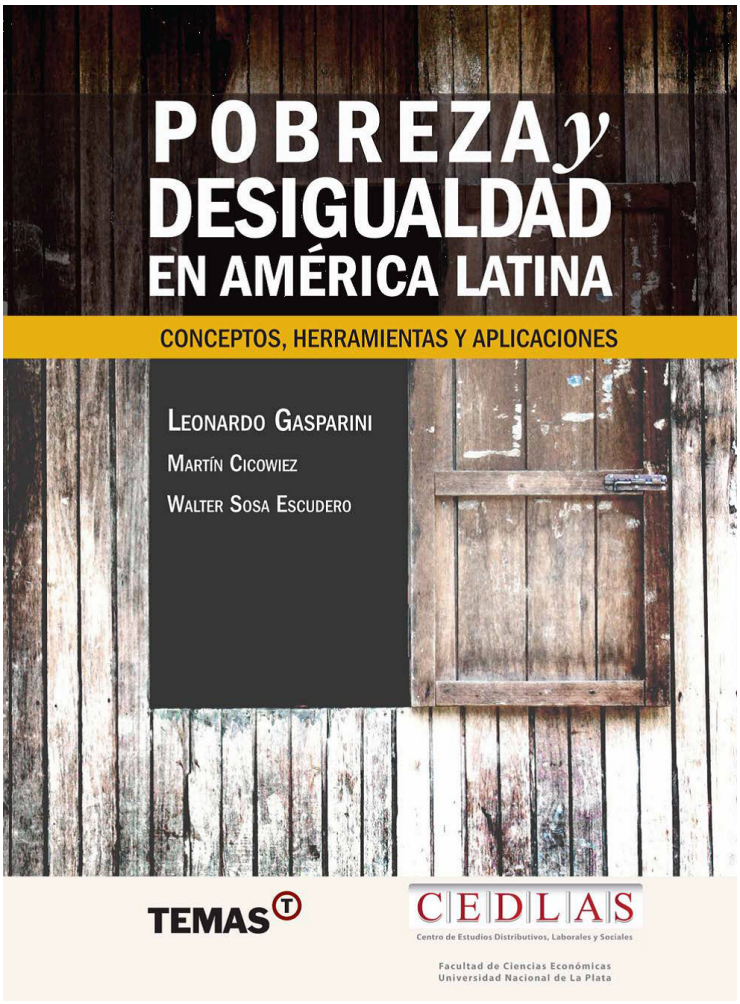
El objetivo de este mini-proyecto no es otro que poner a disposición de un público más amplio estas herramientas útiles y mantener actualizado un material único que permite adentrarse y trabajar sobre la temática de pobreza y desigualdad. Es por eso que este material es de carácter complementario a las explicaciones y detalles conceptuales que se presentan en el libro de texto y los apéndices.

El Libro

Si esta es la primera vez que te encontrás con este libro, antes de empezar con los códigos, dejanos presentartelo.

El propósito del libro es ayudar al lector interesado en América Latina a que recorra el arduo camino entre los datos y el reporte de resultados rigurosos que puedan contribuir al debate sobre la pobreza y la desigualdad en America Latina. El volumen poner al alcance del lector un conjunto de instrumentos que lo motiven hacia la investigación empírica, y que le permitan producir resultados de la manera más rigurosa posible, para así contribuir a los objetivos últimos de

explicar y cambiar la realidad social de la región. Las discusiones conceptuales y analíticas son ilustradas con ejemplos concretos construidos con datos de los países de la región.



Chapter 2

"Apéndices del Libro Pobreza y Desigualdad en R"

2.1 - Capítulo 3

Escrito por: Cristian Bonavida

Last Update: 02/7/2021

Códigos escritos en base a los apéndices del libro "Pobreza y Desigualdad en América Latina" de Gasparini, Cicowiez y Sosa Escudero. El objeto de este material es reproducir la rutina de códigos para STATA presentada en el libro al lenguaje R. Este material es solo de caracter complementario a las explicaciones y detalles conceptuales que se presentan en el libro de texto y los apéndices

2.2 Set Inicial

Cargo las librerias, limpio environment, defino el path y atajo para función paste

```
library(dplyr)
library(tidyverse) # Data wrangling
library(tidygraph)
library(readxl)
library(ggplot2)
library(foreign)
library(TAM)

rm(list=ls())    #empiezo limpiando todo

"%+%" <- function(x,y) paste(x,y,sep = "")      # defino un shorcut parar concat de te
data_dir <- "C:/Users/HP/Desktop/CEDLAS - UNLP/Apendices en R/Material libro/encuestas
```

2.3 Cociente de cuantiles

###- (pág. 151-152)

El siguiente bloque de código puede utilizarse para computar el cociente de quintiles extremos presentado en el cuadro 3.2 del texto del libro. Para iniciar importo la base desde el formato STATA utilizando de la librería `foreing` el comando `read.dta`. y hago una limpieza de la base. Luego ordeno las observaciones según la variable "ipcf", filtro ingresos nulos y por último computo el porcentaje acumulado por población.

```
#cargo base
mex06 <- read.dta(data_dir %+% "Mex/2006/bases/mex06_cedlas.dta")

#elimino observaciones incoherentes o con ingreso missing
mex06 <- mex06 %>% filter(cohh==1, !is.na(ipcf))

#ordenar por ipcf, filtrar ingresos nulos y computar % de población
df <-  mex06  %>% arrange(ipcf) %>% filter(ipcf>0) %>%
       mutate(shrpop=cumsum(pondera)/sum(pondera))
```

A partir de allí genero la variable quintil en el dataframe, que vale 1 para el 20% más pobre de la población, 2 para el 20% siguiente, y así sucesivamente. El comando `ifelse` permite evaluar el porcentaje acumulado de población en cada observación y le otorga el valor del quintil correspondiente en caso afirmativo, o deja la variable inalterada en caso negativo.

```
df$quintil = 0
df$quintil = ifelse(df$shrpop<=0.2, 1, df$quintil)
df$quintil = ifelse(df$shrpop> 0.2 & df$shrpop<= 0.4, 2, df$quintil)
df$quintil = ifelse(df$shrpop> 0.4 & df$shrpop<= 0.6, 3, df$quintil)
df$quintil = ifelse(df$shrpop> 0.6 & df$shrpop<= 0.8, 4, df$quintil)
df$quintil = ifelse(df$shrpop> 0.8 & df$shrpop<= 1,   5, df$quintil)
```

Para terminar computamos el ingreso promedio ponderado para las observaciones del quintil 1 y 5. Aquí la expresión *`[df$quintil==1]`* actúa como filtro. De esta manera el comando le pide a R considerar los valores de la variable ipcf y pondera que corresponden a observaciones que cumplen la condición, en este caso de pertenecer al quintil 1. Por último calculamos el ratio de estos dos valores e imprimimos el resultado. Para mayor claridad siempre se recomienda redondear los valores a imprimir, utilizando el comando `round`.

```
ipcf_q1 = weighted.mean(df$ipcf[df$quintil==1], df$pondera[df$quintil==1], na.rm = TRUE)
ipcf_q5 = weighted.mean(df$ipcf[df$quintil==5], df$pondera[df$quintil==5], na.rm = TRUE)
ratq51 = ipcf_q5/ipcf_q1

paste("ratq51 = ", ratq51)
```

```
## [1] "ratq51 =  13.4341399453849"
```

```
paste("ratq51 = ", round(ratq51, digits = 2))
```

```
## [1] "ratq51 =  13.43"
```

2.4 Replicar programa ratq51

###- (pág. 153-154)

En la siguientes líneas de código proponemos escribir una función que permite computar el cociente de quintiles extremos muy fácilmente; simplemente se lo invoca indicando la base de datos, la variable de la que se quieren computar quintiles, y opcionalmente la variable de ponderación y la condición `if`. Una función en R es similar a lo que en STATA llamaríamos un programa, es decir un conjunto de instrucciones que se guardan en la memoria y que pueden ser fácilmente replicables en distintas bases de datos y para distintas variables.

La función puede resultar un poco extensa por lo que aquí enumeraremos las lineas para mayor claridad. Para iniciar la función, en la primer línea debemos

especificar sus argumentos, que en este caso son: un dataframe (objeto "df"), una variable (objeto "varname") y opcionalmente una variable de ponderación y una condición (objetos "var_pondera" y "condicion"). Para llamar a esta función entonces siempre tendremos que, como mínimo, especificar los dos objetos iniciales.

La siguiente línea se asegura que la base de datos "df" que hemos especificado siempre sea almacenado como un objeto del tipo *data frame* en un objeto llamado "aux" que creamos en el entorno de la función. La noción de entorno es importante para comprender el resto del código. A los fines de nuestro código diremos que un entorno es un espacio determinando en el que se almacenan distintas clases de objetos que R acepta y, en el cual estos tienen un significado, un valor o simplemente existen como objetos. En R pueden "coexistir" environmets distintos, a los cuales se le asigna un nivel o prioridad[1]. Volviendo a nuestro código, cuando a R le especificamos, por ejemplo, el objeto "varname", R buscará si este está definido en el *entorno o environment* general del programa como prioridad, y no en el entorno o espacio de la función, que más acotado que el anterior. Por lo tanto aquí necesitamos evitar este comportamiento y lograr que R no busque evaluar al objeto por fuera del ambito de la función. El comando `substitute` le indica a R que considere al objeto con su nombre literal. Así cuando "varname" sea definida, por ejemplo, como la variable "ipcf", R no buscará a qué es equivalente el objeto "ipcf" en su memoria sino que tomará el nombre dado. Luego la función `eval` le indica a R que ese nombre debe evaluarlo, es decir buscarlo o identificarlo, dentro del dataframe "aux" que hemos definido en nuestra función. De esta forma nos aseguramos de que R considera correctamente las variables dentro de la base de datos que estamos especificando, y por lo tanto que nuestros argumentos sean válidos dentro del ámbito de la función. Es decir que no entren en conflicto con otros objetos similares o definidos previamente en el environment mas amplio del programa.

Utilizando estas dos funciones, en la línea 9 almacenamos todos los valores de ipcf a partir de los cuales luego ordenamos la base en la siguiente línea. Como la condición es un argumento opcional debemos evaluar si esta fue indicada, en términos de R si no es un objeto nulo (línea 15). Si no lo es, luego en la línea 17 verificamos que esté correctamente especificada, escribiendo una sentencia que será VERDADERA cuando al evaluar de forma silenciosa la condición, de como resultado un error *("try-error")*. En ese caso entonces se imprime un mensaje que ayuda a identificar donde se encuentra el problema. En caso de que el resultado no sea del tipo error, filtramos el objeto "aux" de acuerdo a la especificación dada. Para ellos nos valemos de la función `parse` que convierte la condición que era del tipo *"string"* al tipo *"expression"*, lo que la vuelve

[1]Para los usuarios que no esten tan familiarizados con las nociones de *Environment* y *Non-Standard Evaluation* se recomienda revisar las siguientes referencias para un tratamiento mas detallado.

*http://adv-r.had.co.nz/Environments.html#environments

*http://adv-r.had.co.nz/Computing-on-the-language.html

*https://advanced-r-solutions-ed1.netlify.app/non-standard-evaluation.html#non-standard-evaluation-in-subse

compatible de ser evaluada como sentencia, nuevamente con la función `eval` dentro de `filter`.

De forma similar procedemos con el ponderador en la línea 28. Si no fue especificado (objeto nulo), entonces vale 1 para todas las observaciones, a partir de crear una variable que llamamos "w" que pasará a ser nuestra variable ponderador, pero que será inocua a los fines del cálculo. En el objeto "var_pondera_store" almacenamos esta secuencia de valores 1. En cambio, si el ponderador fue especificado entonces ese objeto almacena los valores correspondientes a la variable señalada, sin entrar en conflicto con un posible objeto del mismo nombre definido fuero del espacio de la función.

En la línea 43 actualizamos el objeto que almacena los valores de ipcf (en caso de que se hayan filtrado valores de acuerdo a la condición), y en la siguiente ordenamos el dataframe "aux", para calcular como antes el porcentaje acumulado de población. Por último definimos inicialmente a la variable "quintil" como valores 0. En la línea 50 replicamos la asignación del quintil que se realizó antes con el comando `ifelse` pero ahora de forma iterativa, cambiando en cada iteración los valores del quintil y de población acumulada.

Por último al igual que antes calculamos el ratio entre quintiles extremos filtrando las observaciones del primer y quinto quintil. La única diferencia es que aquí nuevamente debemos asegurarnos de que los valores promedios y los ponderadores se evalúen para estas observaciones y sólo en el contexto o espacio de la función. Para finalizar imprimimos los resultados.

```
1  ratq51 <- function(df, varname, var_pondera=NULL, condicion=NULL) {
2
3
4      aux <- as.data.frame(df)
5
6
7      #substitute() toma literal el nombre sin asignarle ningun valor posible y
8      #eval() evalua ese nombre en el contexto del data frame que es el 1er arguento definido
9      varname_store <- eval(substitute(varname), aux)
10     aux <- aux %>% arrange(varname_store)
11
12
13
14     #la condición es un argumento opcional
15     if(!is.null(substitute(condicion))) {
16
17        if(is(try(eval(substitute(condicion)), silent = TRUE ), "try-error"))
18
19           stop("ERROR: la condicion debe especificarse como character (entre comillas)")
20
21           aux <- aux %>% filter(eval(parse(text=condicion)))   #convierte la condicion de tipo st
```

```
                                                                    #expression y luego la eva
  }



  #set pondera= 1 si no está especificado
  if(is.null(substitute(var_pondera))) {

    aux <- aux %>% mutate(w=1)
    var_pondera=substitute(w)
    var_pondera_store <- eval(substitute(var_pondera), aux)

    } else {

    var_pondera_store <- eval(substitute(var_pondera), aux)

  }



  #ordeno y genero quintiles
  varname_store <- eval(substitute(varname), aux)
  aux <- aux %>% arrange(varname_store) %>%
                 mutate(shrpop=cumsum(var_pondera_store)/sum(var_pondera_store)) %>%
                 mutate(quintil=0)

  for (i in 1:5) {

   aux <- aux %>% mutate(quintil=ifelse((shrpop>(i-1)*0.2 & shrpop<=i*0.2), i, quinti

  }



  #ingreso promedio quintil 1
  aux_1 <- aux %>% filter(quintil==1)
  media_q1 = weighted.mean(eval(substitute(varname),aux_1), eval(substitute(var_ponde
  media_q1 = round(media_q1, digits = 2)
  #ingreso promedio quintil 5
  aux_5 <- aux %>% filter(quintil==5)
  media_q5 = weighted.mean(eval(substitute(varname),aux_5), eval(substitute(var_ponde
  media_q5 = round(media_q5, digits = 2)
  #ratio
  ratq_51 = media_q5/media_q1

```

```
67
68      print(paste("media quintil 5", substitute(varname), "= ", media_q5))
69      print(paste("media quintil 1", substitute(varname), "= ", media_q1))
70      print(paste("ratio = ", round(ratq_51, digits = 2)))
71
72
73  }
```

Al correr el código, R almacenará en la memoria esta función, a la cual podremos llamar simplemente indicando los argumentos relevantes. En las líneas siguientes se detallan algunos ejemplos de cómo replicar el programa bajo distintas especificaciones.

```
#No especifica condicion
ratq51(mex06, ipcf, pondera)
```

```
## [1] "media quintil 5 ipcf =  6655.49"
## [1] "media quintil 1 ipcf =  448.38"
## [1] "ratio =  14.84"
```

```
#Especifica condicion
ratq51(mex06, ipcf, pondera, "ipcf>0")
```

```
## [1] "media quintil 5 ipcf =  6707.64"
## [1] "media quintil 1 ipcf =  499.3"
## [1] "ratio =  13.43"
```

```
#Especifica condición doble
ratq51(mex06, ipcf, pondera, "region==4 | region==2")
```

```
## [1] "media quintil 5 ipcf =  6420.24"
## [1] "media quintil 1 ipcf =  499.01"
## [1] "ratio =  12.87"
```

```
#Especifica incorrectamente la condición
ratq51(df=mex06, ipcf, pondera, region==4 & urbano==1)
```

```
## Error in ratq51(df = mex06, ipcf, pondera, region == 4 & urbano == 1): ERROR: la condicion deb
```

```
#No especifica pondera
ratq51(df=mex06, varname=ipcf, condicion="region==4 & urbano==1")
```

```
## [1] "media quintil 5 ipcf =  6384.76"
## [1] "media quintil 1 ipcf =  594.14"
## [1] "ratio =  10.75"
```

```
#Identifica los argumentos por explicitamente por nombre
ratq51(df=mex06, varname=ipcf, var_pondera=pondera, condicion="region==4 & urbano==1")
```

```
## [1] "media quintil 5 ipcf =  6640.93"
## [1] "media quintil 1 ipcf =  586.57"
## [1] "ratio =  11.32"
```

2.5 Replicar programa gcuan

###- (pág. 154-155)

El bloque de código a continuación permite identificar cuantiles de cualquier variable. En términos del programa "ratq51", nos permite generar variables similares a quintil pero que pueden identificar quintiles, deciles, percentiles, etc.

Por esta razón, esta función tendrá más argumentos, aquí además de los anteriores debemos detallar la cantidad de cuantiles a generar (objeto "num") y la variable que los almacena ("newvar"). Luego el código y la secuencia son idénticos a la de la función anterior, salvo porque aquí definimos de una manera alternativa y más directa a los valores del poderador, pero en esencia replica lo visto antes hasta la línea 43. A partir de aquí el objeto "num" indica cuantos cuantiles deben generarse, hace iterar al bucle "num" cantidad de veces, y define los intervalos de población acumulada de forma equivalente. Por ejemplo, si queremos generar deciles (num=10), necesitamos 10 cuantiles y cada cuantil se asigna de a intervalos de población acumulada iguales a 0.10 (1/10).

Posteriormente en la línea 54 generamos el reporte que imprimiremos como resultado. Este calcula la media ponderada, el desvió standard ponderado y la cantidad de observaciones en base la variable especificada en "varname" y "var_pondera". Los objetos "my_var" y "my_var2" son variables auxiliares que generamos en el data frame solo con el objeto de faciliarnos el cálculo directo. Para terminar cambiamos el nombre de la variable que hasta ahora generamos como "quintil" por el nombre indicado en el argumento. El comando `names(aux)` trae todos los nombres de este dataframe y con la expresión *`[names(aux) == "quintil"]`* elegimos de esos nombres, solo el que coincide con la palabra "quintil". A esta columna específica le asignamos el nombre dado como argumento, tomando la expresión literal con `substitute` pasada al formato string mediante la función `paste`. En resumen la línea 63 sería el equivalente de escribir en la consola *`aux$quintil <- "nombre_asignado"`*,

con la dificultad de que debemos hacerlo dentro del espacio de la función, respetando los argumentos dados. La línea 64 por su parte elimina del output final la variable shrpop.

Para finalizar especificamos que esta función no solo debe imprimirnos los resultados sino que además debe devolver una base de datos nueva. El comando `return(aux)` le dice a R que el resultado será el objeto "aux" en sí mismo, es decir obtendremos la base original con una nueva columna llamada "newvar" que es la que esta función genera.

```
1  gcuan <- function(df, varname, var_pondera=NULL, condicion=NULL, num, newvar) {
2
3
4    aux <- as.data.frame(df)
5
6
7    varname_store <- eval(substitute(varname), aux)
8    aux <- aux %>% arrange(varname_store)
9
10
11   #la condición es un argumento opcional
12   if(!is.null(substitute(condicion))) {
13
14      if(is(try(eval(substitute(condicion)), silent = TRUE ), "try-error"))
15
16         stop("ERROR: la condicion debe especificarse como character (entre comillas)")
17
18         aux <- aux %>% filter(eval(parse(text=condicion)))
19
20    }
21
22
23
24   #set pondera= 1 si no está especificado
25   if(is.null(substitute(var_pondera))) {
26
27     var_pondera_store <- c(rep(1, nrow(aux)))
28
29     } else {
30
31     var_pondera_store <- eval(substitute(var_pondera), aux)
32
33   }
34
35
36
```

```
  #ordeno y genero shrpop
  varname_store <- eval(substitute(varname), aux)
  aux <- aux %>% arrange(varname_store) %>%
    mutate(shrpop=cumsum(var_pondera_store)/sum(var_pondera_store)) %>%
    mutate(quintil=0)

  #genero cuantiles en base a lo indicado
  shareq = 1/num

  for (i in 1:num) {

    aux <- aux %>% mutate(quintil=ifelse((shrpop>(i-1)*shareq & shrpop<=i*shareq), i, c

  }


  #armo la información que se imprime en la consola
  show <- aux %>% mutate(my_var=varname_store,
                         my_var2=var_pondera_store) %>% group_by(quintil) %>%
                  summarise(mean = weighted_mean(my_var, my_var2),
                            std = weighted_sd(my_var, my_var2),
                            obs = sum(my_var2))


  #renombro variable al nombre indicado como argumento y descarto shrpop
  names(show)[names(show) == "quintil"] <- paste(substitute(newvar))
  names(aux)[names(aux) == "quintil"] <- paste(substitute(newvar))
  aux$shrpop <- NULL


  #outputs
  print.data.frame(show)
  return(aux)


}
```

A diferencia entonces de la anterior, esta función devuelve no sólo un resultado impreso sino un objeto, por lo que ahora debemos especificar cómo lo nombramos. Si lo llamamos de la misma manera que el argumento que le pasamos a la función simplemente estamos pisando este objeto y añadiéndole una nueva variable. Otra alternativa es crear un nuevo dataframe con otro nombre.

```
#Especifica todo correctamente y crea nuevo dataframe
mex06_bis <- gcuan(df=mex06, varname=ipcf, var_pondera=pondera, condicion="ipcf>0", nu
```

```
##   quintil      mean       std      obs
## 1       1  499.2978  197.3755 20372798
## 2       2 1011.4695  136.6778 20383133
## 3       3 1539.5536  172.7480 20385227
## 4       4 2382.6530  345.4059 20380795
## 5       5 6707.6362 6532.2051 20381791
```

```
#No especifica correctamente la condición
mex06_bis <- gcuan(df=mex06, varname=ipcf, var_pondera=pondera, condicion=region==4, num=5, newva
```

```
## Error in gcuan(df = mex06, varname = ipcf, var_pondera = pondera, condicion = region == : ERRO
```

```
#No especifica condición
mex06_bis <- gcuan(df=mex06, varname=ipcf, var_pondera=pondera, num=5, newvar=quintil)
```

```
##   quintil      mean       std      obs
## 1       1  448.3757  222.9768 20677642
## 2       2  986.9415  137.2255 20677386
## 3       3 1518.1792  172.0445 20676646
## 4       4 2357.4796  343.4104 20678111
## 5       5 6655.4937 6499.4356 20679061
```

```
#No especifica pondera
mex06_bis <- gcuan(df=mex06, varname=ipcf, num=5, newvar=quintil)
```

```
##   quintil      mean       std   obs
## 1       1  363.0791  202.9644 16400
## 2       2  900.1146  135.1091 16401
## 3       3 1424.6686  177.6839 16401
## 4       4 2273.6985  346.6132 16401
## 5       5 6401.7186 6319.9421 16401
```

```
#Especifica todo correctamente y reemplaza el dataframe dado
mex06 <- gcuan(df=mex06, varname=ipcf, var_pondera=pondera, condicion="ipcf>0", num=5, newvar=qui
```

```
##   quintil      mean       std      obs
## 1       1  499.2978  197.3755 20372798
## 2       2 1011.4695  136.6778 20383133
## 3       3 1539.5536  172.7480 20385227
## 4       4 2382.6530  345.4059 20380795
## 5       5 6707.6362 6532.2051 20381791
```

2.6 Tamaño de los hogares

###- (pág. 156)

El código siguiente puede utilizarse para computar las estadísticas sobre proporción de hogares unipersonales y multipersonales presentadas en el cuadro 3.4 del texto. Como ya es habitual, en la primera línea indicamos con qué base iremos a trabajar. Luego ordenamos los hogares de forma creciente en base a su identificador "id" y en forma decreciente respecto la variable "jefe". Dado que esta vale 1 solo para el jefe de hogar y 0 para todo el resto, la primera observación por "id" corresponderá siempre a la cabeza del hogar. La función `duplicated` genera una variable que será *FALSE* para la primera observación dentro de cada id, indicando que esta no está duplicada por ser la primera, pero será *TRUE* para todas las que siguen ya que se identificó antes una observación con el mismo id. De esta manera hacemos un "tag" del jefe de hogar y mantenemos todos los demás miembros. En el caso de que nos interesa mantener en nuestra base el resto de las observaciones, estos pasos se simplifican a un simple filtrado por jefe de hogar.

Luego generamos la variable tamaño solo para el jefe de hogar en base la cantidad de miembros que viven con él, utilizando la función `case_when`. Así, por ejemplo, cuando la observación corresponda al jefe y la cantidad de miembros sea igual a 3, en ese caso tamaño valdrá 3. Por último armamos nuestra tabla resultado filtrando a los jefes de hogar, agrupando por tamaño y sumando la cantidad de hogares (porque tenemos una sola observación por hogar) y la frecuencia relativa en cada caso.

```
#indico con qué base de hogares voy a trabajar
df <- mex06

#ordeno por id y jefe (decreciente) e identifico al jefe de hogar
df <- df %>% arrange(id, desc(jefe)) %>%
          mutate(hh=duplicated(id))      #alterantiva para egen hh=tag() en stata

#genero tamaño sólo para los jefes de hogar
df <- df %>% mutate(tamaño=case_when(miembros==1 & hh==FALSE ~ 1,
                                     miembros==2 & hh==FALSE ~ 2,
                                     miembros==3 & hh==FALSE ~ 3,
                                     miembros==4 & hh==FALSE ~ 4,
                                     miembros==5 & hh==FALSE ~ 5,
                                     miembros>=6 & hh==FALSE ~ 6))

#tabla con resultados
table <- df %>% filter(hh==FALSE) %>% group_by(tamaño) %>% summarise(N = sum(pondera))
                                                           mutate(freq = 100*N/sum(N))
table
```

```
## # A tibble: 6 x 3
##   tamaño       N  freq
##    <dbl>   <int> <dbl>
## 1      1 2070859  8.15
## 2      2 3717891 14.6
## 3      3 4815417 18.9
## 4      4 5875572 23.1
## 5      5 4422391 17.4
## 6      6 4516967 17.8
```

El objeto table almacena los resultados que se visualizan en la consola. A veces al imprimir los resultados estos no se visualizan con un formato muy amigable. A modo de extensión presentamos dos alternativas para refinar este aspecto. El paquete `formattable` nos permite, entre otros cosas, especificar el tipo de datos y el formato para distintas variables al generar el dataframe. Por su parte `print.data.frame` nos da una visualización más limpia. Incorporando este aspecto el código para nuestra tabla sería:

```
#install.packages("formattable")
library(formattable)

table <- df %>% filter(hh==FALSE) %>% group_by(tamaño) %>% summarise(N = accounting(sum(pondera),
                                                                    mutate(freq = percent(N/sum(N)))
print.data.frame(table)
```

```
##   tamaño         N   freq
## 1      1 2,070,859  8.15%
## 2      2 3,717,891 14.63%
## 3      3 4,815,417 18.94%
## 4      4 5,875,572 23.11%
## 5      5 4,422,391 17.40%
## 6      6 4,516,967 17.77%
```

Además de agregarle esta customización, para los usuarios que utilizan "Rmarkdown" habitualmente, es posible mostrar los resultados con una visualización amigable de forma sencilla.

```
rmarkdown::paged_table(table)
```

2.7 La distribución intrahogar

###- (pág. 157)

El fragmento de código siguiente puede utilizarse para generar resultados similares a los presentados en el cuadro 3.7 del texto, que muestra cómo se modifica la desigualdad calculada a través del cociente de deciles extremos cuando cambia la distribución del ingreso hacia el interior del hogar. Cabe recordar que la distribución del ingreso intrahogar se modifica mediante un impuesto proporcional al ingreso per cápita familiar combinado con un subsidio que solo recibe el jefe de hogar. En la implementación, utilizamos quintiles en lugar de deciles ingreso.

En la primer línea cargamos la base, en este caso trabajaremos con la base de Venezuela del año 2006, limpiamos y ordenamos los hogares. Mas abajo creamos el objeto "ty" que toma el valor de la tasa del impuesto aplicada sobre el ipcf, el cual calculamos en línea siguiente generando en el dataframe una nueva columna "impuesto". Posteriormente calculamos el valor del subsidio a otorgar que surge de sumar el valor de los impuestos para los integrantes de un mismo hogar. En la line siguiente modificamos la variable subsidio, haciendo que valga 0 para todo miembro distinto al jefe, de esta forma redistribuimos ingreso al interior del hogar: a todos se le hemos quitado una porción "ty" que ahora la recibe solamente el jefe de hogar.

En la anteúltima línea creamos una variable para el nuevo valor de ingreso per cápita familiar, restando el impuesto y sumando el subsidio. Por último hacemos uso de nuestra función del ratio de quintiles para computar el cociente del ingreso promedio de los quintiles 5 y 1 como indicador de desigualdad, a partir del ingreso modificado.

```
ven06 <- read.dta(data_dir %+% "Ven/2006/bases/ven06_cedlas.dta")

#elimino observaciones incoherentes o con ingreso missing y ordeno
df <- ven06 %>% filter(cohh==1, !is.na(ipcf)) %>% arrange(id)

#tasa del impuesto
ty=0.1

#impuesto al ipcf
df$impuesto = df$ipcf*ty    #alternativa a escribir: df <- df %>% mutate(impuesto = ip

#recaudación impuesto total por hogar
df <- df %>% group_by(id) %>% mutate(subsidio = sum(impuesto))

#subsidio solo lo recibe el jefe de hogar
df$subsidio <- ifelse(df$jefe!=1, 0, df$subsidio)

#nuevo ipcf
df$ipcf_star = df$ipcf - df$impuesto + df$subsidio

ratq51(df, ipcf_star, pondera, "ipcf>0")
```

```
## [1] "media quintil 5 ipcf_star =  842520.38"
## [1] "media quintil 1 ipcf_star =  79335.26"
## [1] "ratio =  10.62"
```

A modo de comentario, un usuario mas familiarizado con el lenguaje podrá notar que las lineas que computan y asignan el subsidio puden resumirse en una sola, escribiendo.

```
df <- df %>% group_by(id) %>% mutate(subsidio=ifelse(jefe!=1, 0, sum(impuesto)))
```

2.8 Empleo de ponderadores

###- (pág. 157)

El bloque de código que sigue puede utilizarse para construir un cuadro como el 3.9 del texto, que muestra la relación entre el ingreso per cápita familiar y el valor de la variable de ponderación.

La línea 7 genera la variable shrobs, que contiene el porcentaje acumulado de observaciones, a diferencia de shrpop que contiene el acumulado de población usando el ponderador. Aquí la expresión *1:n()* enumera secuencialmente las observaciones en orden de aparición mientras que *'n()"* computa el total.

```
#indico con que base de hogares voy a trabajar
df <- mex06

#ordenar por ipcf y computo porcentaje acumulado de observaciones
df <- df %>% arrange(ipcf) %>%  mutate(shrobs = 1:n()/n())

#identificar quintiles de ipcf
df$quintil = 0
df$quintil = ifelse(df$shrobs<= 0.2, 1, df$quintil)
df$quintil = ifelse(df$shrobs>0.2 & df$shrobs<= 0.4, 2, df$quintil)
df$quintil = ifelse(df$shrobs>0.4 & df$shrobs<= 0.6, 3, df$quintil)
df$quintil = ifelse(df$shrobs>0.6 & df$shrobs<= 0.8, 4, df$quintil)
df$quintil = ifelse(df$shrobs>0.8 & df$shrobs<= 1,   5, df$quintil)

show <- df %>% group_by(quintil) %>% summarise(mean = accounting(mean(ipcf), digits = 0),
                                               means_w = accounting(mean(pondera), digits = 0))
print.data.frame(show)
```

```
##   quintil  mean means_w
## 1       1   413   1,038
## 2       2   926   1,269
## 3       3 1,449   1,345
## 4       4 2,302   1,344
## 5       5 6,457   1,319
```

El bloque de código siguiente calcula las tasas de pobreza con y sin ponderadores para el total del país y para cada una de las regiones de México en 2006 correspondientes al cuadro 3.10 del texto. En la primer línea el objeto "lp" almacena el valor de la línea de pobreza, en base a la cual se genera la variable binaria "pobre", vale 1 para los individuos pobres (es decir, $ipcf < lp$) y 0 para el resto. La suma de esta variable arroja el número total de personas pobres en la muestra, al multiplicarla por el ponderador obtenemos el número de personas pobres en la población y dividiendo por la población total del país nos devuelve la tasa de incidencia de la pobreza. En este caso el valor es de 13.57%

```
###linea de pobreza us$2.5 Mexico 2006
lp= 633.90918

#identificar individuos pobres
df$pobre = ifelse(df$ipcf<lp, 1, 0)

#total pais
sum(df$pobre*df$pondera)*100/sum(df$pondera)
```

```
## [1] 13.57454
```

Las siguientes líneas calculan el mismo valor de una manera alternativa, almacenándolo en un dataframe, a partir de contabilizar la frecuencia absoluta y relativa de personas pobres y no pobres.

```
pobreza <- df %>% group_by(pobre) %>% summarise(n = accounting(sum(pondera), digits = 0
                                    mutate(tasa_pobreza = percent(n/sum(n)))
print.data.frame(pobreza)
```

```
##   pobre          n tasa_pobreza
## 1     0 88,070,783       86.43%
## 2     1 13,832,961       13.57%
```

Para replicar el mismo cálculo pero para las regiones del país, generamos un bucle que itera 8 veces, una vez por cada región. En cada vuelta del bucle se estima, de la misma forma que arriba, el valor de pobreza tanto ponderado como no ponderado. Para este segundo caso el único paso adicional consiste en

fijar la variable "pondera" como igual a 1. Como vimos en la salida anterior de la tabla, en la 2da fila de la 3er columna se encuentra la tasa de incidencia de pobreza, con la expresión *pobreza_pondera[2,3]* llamamos a este valor para guardarlo en el objeto "share_p", luego de redondearlo a dos dígitos con el comando `round`. Lo mismo hacemos para las estimaciones sin ponderar. Por último, imprimimos los resultados acompañados por una leyenda indicativa.

```
#por region
for (i in 1:8){

  pobreza_pondera <- df %>% filter(region==i) %>% group_by(pobre) %>% summarise(n=sum(pondera)) %
                                                                      mutate(tasa_pobreza=n/sum(n))

  pobreza_s_pondera <- df %>% filter(region==i) %>% mutate(pondera=1) %>% group_by(pobre) %>% sum
                                                                          mutate(tasa

  #recupero el valor llamando a la fila y columna y redondeo
  share_p = round(pobreza_pondera[2,3]*100, digits = 2)
  share_sp = round(pobreza_s_pondera[2,3]*100, digits = 2)

  print(paste("H_ponderado =", share_p, "/ H_sin_ponderar =", share_sp))

}
```

```
## [1] "H_ponderado = 7.73 / H_sin_ponderar = 8.21"
## [1] "H_ponderado = 12.23 / H_sin_ponderar = 17.43"
## [1] "H_ponderado = 4.73 / H_sin_ponderar = 10.03"
## [1] "H_ponderado = 11.94 / H_sin_ponderar = 11.18"
## [1] "H_ponderado = 9.55 / H_sin_ponderar = 13.4"
## [1] "H_ponderado = 35.13 / H_sin_ponderar = 44.68"
## [1] "H_ponderado = 20.64 / H_sin_ponderar = 21.14"
## [1] "H_ponderado = 14.51 / H_sin_ponderar = 18.91"
```

Este bloque de códigos imprime los resultados en la consola. Si quisiéramos generar un cuadro u objeto que los almacene (dataframe) para luego por ejemplo exportarlo en Excel u otro formato, solo debemos modificar ligeramente el código de arriba. Antes de correr el bucle generamos un dataframe que contiene todos valores 0, pero con 3 columnas (region, pondera, sin_pondera). Al final del bucle guardamos los datos de la región, de pobreza con ponderador y sin ponderador en la columna 1, 2 y 3 respectivamente, pero en cada iteración lo hacemos en una fila distinta, que corresponde a cada región. Por ejemplo en la tercera iteración el código filtra la región a la cual le corresponde el valor 3, y almacena las estimaciones de línea de pobreza en la tercera fila de la tabla "results".

```
#creo data frame solo con ceros, de 8 filas y 3 columnas
results <- data.frame(region=c(rep(0,8)),
                      pondera=c(rep(0,8)),
                      sin_pondera=c(rep(0,8)))

#por region
for (i in 1:8){

  pobreza_pondera <- df %>% filter(region==i) %>% group_by(pobre) %>% summarise(n=sum(
                                                                    mutate(tasa_pobreza=

  pobreza_s_pondera <- df %>% filter(region==i) %>% mutate(pondera=1) %>% group_by(pobr


  #recupero el valor llamando a la fila y columma y redondeo
  results[i,1] = paste("region", i)
  results[i,2] = round(pobreza_pondera[2,3]*100, digits = 2)
  results[i,3] = round(pobreza_s_pondera[2,3]*100, digits = 2)

}
```

De esta manera al finalizar el bucle queda armado un objeto de 3 columnas con 8 filas, una para cada región

2.9 Diseño muestraL

###- (pág. 160)

En este material no se cubre con ejemplos este apartado pero para los usuarios interesados en adentrarse en el manejo de encuestas de hogares contemplando el diseño muestral, se recomienda explorar el uso del paquete "SURVEY", similar al paquete "svy" en STATA.

```
#install.packages("survey")
library(survey)
?survey
```

2.10 Fuentes de ingreso

###- (pág. 161)

El bloque de código a continuación muestra cómo computar la importancia que tiene cada fuente de ingresos identificada en las encuestas de hogares (cuadro 3.13). Dento de las fuentes de ingreso consideramos: laboral (variable ila), jubilaciones (ijubi), capital (icap), transferencias (itran) y otros (ionl).

Para comenzar declaramos la base y la limpiamos. Luego sumamos generamos la variable ingreso total ("itot"), como la suma de las columnas para cada ingreso. Hasta ahora todas las veces que aplicamos el comando `sum` lo hicimos sumando una misma columna, por ejemplo para obtener el total de población, sumamos la variable "pondera". En este caso queremos sumar por fila distintas columnas, para eso debemos anteponer el comando `rowwise`.

Como resultado final necesitamos mostrar la participación en el ingreso total de cada fuente de ingreso. Para ello debemos considerar cada fuente por separado y divirla por el ingreso total. La forma de hacerlo nuevamente es utilizando un bucle. Entonces el código a escribir debe lograr en cada iteración tomar todos los valores de cada una de las variables que corresponde a la fuente de ingresos, en otras palabras debemos iterar entre columnas distintas. La forma que proponemos aquí consiste en valernos de las listas. Las listas son un objeto que en cada elemento puede almacenar otro objeto de cualquier tipo. Una lista de n elementos puede contener n dataframes distinto en cada uno de ellos, o n columnas de un dataframe, o n vector. Aquí entonces nos valdremos de esta flexibilidad para almacenar en un mismo objeto (lista) múltiples objetos distintos (columnas)

La cuarta línea de códigos declara a "ingresos" como una lista que como elementos contiene a las variables de ingreso de nuestra base de datos. En la línea siguiente creamos un simple vector con los nombres de cada una de estas. Con esto ya podemos generar nuestro bucle, haciéndolo iterar desde 1 hasta n, siendo n la cantidad de fuentes de ingresos distintas que consideramos. Para evitarnos contar manualmente cuantas son, directamente calculamos n con el comando `length` que nos devuelve la cantidad de elementos dentro de la lista "ingresos". De esta forma si agregamos o quitamos una variable de ingreso no debemos preocuparnos por fijar este valor cada vez.

En la primer línea del código definimos la variable "y" que, en cada iteración, será un elemento distinto del objeto ingreso, es decir una fuente de ingreso distinta, comenzando por "ila" y terminando en "ionl". A esta variable la expandimos multiplicándola por el ponderador, de la misma manera que expandimos la variable de ingreso total, que ya calculamos mas arriba. Solo nos resta hacer el cociente entre la suma de los ingresos de todas las personas correspondiente a cada fuente, sobre el total de todos los ingresos. El objeto "value" almacena este cálculo, que luego redondeamos a dos dígitos. Nótese la importancia de la opción *`na.rm=TRUE`* que evita que un valor missing convierta en missing a toda la suma, es decir que indica ignorar los valores missings y preservar el cálculo sobre el resto de las observaciones no missings. Para terminar, "concepto" almacena el nombre de la fuente correspondiente a cada vuelta del bucle, para imprimirlo en la línea final junto con el valor calculado.

```
arg06 <- read.dta(data_dir %+% "Arg/2006/s2/bases/arg06_cedlas.dta")

df <- arg06 %>% filter(cohh==1, !is.na(ipcf))

#sumo los ingresos por fila (rowwise)
df <- df %>% rowwise %>% mutate(itot = sum(ila, ijubi, icap, itran, ionl, na.rm = TRUE)

#creo una lista en la que cada elemento es un vector distinto de ingreso
ingresos <- list(df$ila, df$ijubi, df$icap, df$itran, df$ionl)
names <- c("laboral", "jubilación", "capital", "transferencias", "otros")


#itero sobre cada uno de esos vectores
for (i in 1:length(ingresos)){

  y = ingresos[[i]]
  y_expand = y * df$pondera
  itot_expand = df$itot * df$pondera

  value <- sum(y_expand, na.rm=TRUE) / sum(itot_expand, na.rm=TRUE) * 100
  value <- round(value, digits = 2)

  concepto = names[i]
  print(paste("shr %", concepto, "= ", value))


  }
```

```
## [1] "shr % laboral =  80.4"
## [1] "shr % jubilación =  12.24"
## [1] "shr % capital =  1.5"
## [1] "shr % transferencias =  4.77"
## [1] "shr % otros =  1.09"
```

Chapter 3

"Apéndices del Libro Pobreza y Desigualdad en R"

3.1 - Capítulo 4

Escrito por: Cristian Bonavida

Last Update: 28/8/2021

Códigos escritos en base a los apéndices del libro "Pobreza y Desigualdad en América Latina" de Gasparini, Cicowiez y Sosa Escudero. El objeto de este material es reproducir la rutina de códigos para STATA presentada en el libro al lenguaje R. Este material es solo de caracter complementario a las explicaciones y detalles conceptuales que se presentan en el libro de texto y los apéndices

3.2 Set Inicial

Cargo las librerias, limpio environment, defino el path y atajo para función paste

```
library(dplyr)
library(tidyverse) # Data wrangling
library(tidygraph)
library(readxl)
library(ggplot2)
library(foreign)
library(TAM)

rm(list=ls())    #empiezo limpiando todo

"%+%" <- function(x,y) paste(x,y,sep = "")      # defino un shorcut parar concat de te
data_dir <- "C:/Users/HP/Desktop/CEDLAS - UNLP/Apendices en R/Material libro/encuestas
```

3.3 Indicador FGT

###- (PÁG. 249-250)

En este apartado se presenta cómo calcular la familia de indicadores FGT. En primer lugar, se muestra cómo puede computarse el indicador FGT de manera relativamente sencilla. Luego, al igual que con el ratio de quintiles y el cálculo de quintiles, se se introduce una función para el indicador FGT. Como ejemplo, computamos la pobreza de 2.5 dólares para Ecuador en 2006, utilizando microdatos que provienen de la Encuesta de Condiciones de Vida (ENCOVI).

Luego de cargar y limpiar las bases fijamos los valores de pobreza así como el parámtero *"alfa"* del indicador fgt de aversión a la desigualdad entre los pobres. Luego se computa, para cada individuo pobre, su brecha de pobreza elevada al valor de alfa, que se asigna a una nueva variable que llamamos "each". La distinción entre individuo pobre y no pobre se operativiza en el comando `ifelse`: en caso de cumplirse la condición de que el ingreso esté por debajo de la linea de pobreza, se computa la brecha, en caso de que sea falso se otorga un valor de cero. Por último se obtiene el valor de FGT como el promedio ponderado de la brecha en toda la muestra.

```
#cargo base y elimino observaciones incoherentes o con ingreso missing
ecu06 <- read.dta(data_dir %+% "Ecu/2006/bases/ecu06ecv_cedlas.dta") %>%
         filter(cohh==1, !is.na(ipcf))

df <- ecu06

#linea de pobreza
lp=39.74
```

```
#parametro alfa indicador fgt
alfa=0

#computar fgt
df <- df %>% mutate(each = ifelse(ipcf<lp, (1 - ipcf/lp)^alfa, 0 ))
fgt = weighted.mean(df$each, df$pondera, na.rm = TRUE)*100

print("fgt = " %+% round(fgt, d=2))
```

```
## [1] "fgt = 19.02"
```

Otra opción posible es computar los valores directamente como un vector, sin alojarlo como una nueva columna del data frame. Esta opción, desde el punto de vista de la escritura de los códigos, es más eficiente ya que se evita agregar una nueva columna que solo se emplea para el cálculo del indicador. El detalle a considerar es que para poder calcularlo como medida ponderada la cantidad de elementos del vector "each" debe ser exactamente igual la cantidad de observaciones de la columna pondera del dataframe.

```
#alternativa
each <- ifelse(df$ipcf<lp, (1 - df$ipcf/lp)^alfa, 0 )
fgt = weighted.mean(each, df$pondera, na.rm = TRUE)*100

print("fgt = " %+% round(fgt, d=2))
```

```
## [1] "fgt = 19.02"
```

3.4 Computar FGT

###- (pág 250-251)

Como ya vimos en el capitulo 3, las funciones nos permiten replicar un indicador o un cálculo en cualquier base, sobre cualquier variable, imponiendo una condición especifica o ponderando por algún factor de expansión. En este caso además agregamos como argumentos los parámetros asociados al FGT, *"alfa"*, para la aversión a la desigualdad, y *"zeta"* para la linea de pobreza. De esta forma podremos replicar y comparar rapidamente el índice para distintos valores que podemos pasarle a estos argumentos. La estructura de la función replica lo visto en las funciones del capítulo 3 hasta la linea 39 donde se computa el FGT con las mismas lineas que empleamos arriba. Adicionalmente se añade la opción de correr la función sin de forma silenciosa, sin imprimir el resultado.

```
FGT <- function(df, varname, var_pondera=NULL, condicion=NULL, alfa, zeta, quiet=FALSE)

  aux <- as.data.frame(df)

  varname_store <- eval(substitute(varname), aux)
  aux <- aux %>% arrange(varname_store)


  #la condición es un argumento opcional
  if(!is.null(substitute(condicion))) {

     if(is(try(eval(substitute(condicion)), silent = TRUE ), "try-error"))

        stop("ERROR: la condicion debe especificarse como character (entre comillas)")

        aux <- aux %>% filter(eval(parse(text=condicion)))

   }


  #set pondera igual a 1 si no está especificado
  if(is.null(substitute(var_pondera))) {

    var_pondera_store <- c(rep(1, nrow(aux)))

  } else {

    var_pondera_store <- eval(substitute(var_pondera), aux)

  }


  #Cómputo de brecha y valor del indicador
  varname_store <- eval(substitute(varname), aux)
  aux <- aux %>% mutate(each = ifelse( varname_store < zeta,
                                       ( 1 - varname_store/zeta)^alfa,
                                         0 ))

  fgt = weighted.mean(aux$each, var_pondera_store, na.rm = TRUE)*100
  fgt = round(fgt, digits = 2)

  #output
  if(substitute(quiet)==TRUE){
    a=fgt

```

```
  } else {

    print(paste("FGT(alfa=", alfa, ",Z=", zeta, ") = ", fgt, sep=""))
    a=fgt

  }

}
```

```
#No especifica condicion
FGT(df=df, varname=ipcf, var_pondera=pondera, alfa=0, zeta=39.740)
```

```
## [1] "FGT(alfa=0,Z=39.74) = 19.02"
```

```
#Especifica condicion
FGT(df=df, varname=ipcf, var_pondera=pondera, condicion="urbano==1", alfa=0, zeta=39.740)
```

```
## [1] "FGT(alfa=0,Z=39.74) = 11.82"
```

```
#Especifica incorrectamente la condición
FGT(df=df, varname=ipcf, var_pondera=pondera, condicion=urbano==1, alfa=0, zeta=39.740)
```

```
## Error in FGT(df = df, varname = ipcf, var_pondera = pondera, condicion = urbano == : ERROR: la
```

```
#Especifica opcion "quiet"
FGT(df=df, varname=ipcf, var_pondera=pondera, alfa=0, zeta=39.740, quiet=TRUE)
```

3.5 Pobreza relativa

###- (pág. 251)

La estimación de la pobreza relativa implica, como primer paso, el cálculo de una línea de pobreza relativa. A modo de ejemplo, se computa una línea de pobreza igual al 50% del ingreso mediano de Ecuador. El comando `weightedMedian` de la libreria `matrixStats` arroja el valor de la mediana. Luego, el cálculo de la pobreza se realiza empleando la función FGT.

```
#linea de pobreza del 50% de la mediana del ingreso
lp = matrixStats::weightedMedian(df$ipcf, df$pondera) * 0.50
FGT(df=df, varname=ipcf, var_pondera=pondera, condicion="urbano==1", alfa=0, zeta=lp)
```

```
## [1] "FGT(alfa=0,Z=47.484118603645) = 16.13"
```

3.6 Descompoisición regional de la pobreza

###- (pág. 252-253)

El código a continuación realiza una descomposición por regiones de la tasa de incidencia de la pobreza (cuadro 4.7 del libro de texto). Para este caso utilizamos la Encuesta Nacional de Ingresos y Gastos de los Hogares de México para el año 2006 con la línea de pobreza de 2.5 dólares, equivalentes a 608.245 pesos mensuales. Luego de cargar, limpiar e indicar la base de datos con la que se trabajará, se fija la linea de pobreza y se genera, a partir de la función de FGT, la tasa de incidencia para el total del país que se almacena en el objeto "p0". En la linea siguiente, se toma una unica observación de cada una de las 8 regiones diferentes de México y se las ordena en orden creciente para almacernas como un vector en el objeto "list_rgn". Sobre este objeto haremos iterar un bucle, para que en cada vuelta los calculos indicados se hagan para cada una de estas regiones.

La primer sentencia dentro de ese bucle, calcula la participación de la región sobre la población y en la siguiente linea se estima su indicador FGT para un valor de alfa=0 y con la linea de pobreza de 2.5 dolares (notar como se instrumenta la condición en la función). Por último la contribución se calcula como el producto entre la participación de cada región en la población total y el cociente entre la tasa de pobreza regional y la tasa de pobreza nacional, se redondea y se indica que se imprima dicho valor.

```
#cargo y limpio base
mex06 <- read.dta(data_dir %+% "Mex/2006/bases/mex06_cedlas.dta") %>%
         filter(cohh==1, !is.na(ipcf))

df <- mex06

lp=608.245
p0 = FGT(df=df, varname=ipcf, var_pondera=pondera, alfa=0, zeta=lp, quiet=TRUE)

list_rgn = sort(unique(df$region))
```

```
for (i in list_rgn){

  #participación region
  shr_rgn = sum(df$pondera[df$region==i]) / sum(df$pondera)
  #fgt region
  p_r <- FGT(df=df, varname=ipcf, var_pondera=pondera, condicion=paste("region==", i, sep = ""),

  #contribución
  contribut = round( shr_rgn*(p_r/p0)*100 , digits = 2 )
  print(paste("contribución (%) region", i, "=", contribut))


}
```

```
## [1] "contribución (%) region 1 = 6.09"
## [1] "contribución (%) region 2 = 9.52"
## [1] "contribución (%) region 3 = 3.07"
## [1] "contribución (%) region 4 = 13.99"
## [1] "contribución (%) region 5 = 24.22"
## [1] "contribución (%) region 6 = 26.12"
## [1] "contribución (%) region 7 = 13.07"
## [1] "contribución (%) region 8 = 3.95"
```

3.7 Pobreza según consumo e ingreso

###- (pág. 253-254)

El código que sigue puede utilizarse para replicar los resultados sobre pobreza por consumo e ingreso presentados en el cuadro 4.9 del texto para el caso de Nicaragua en 2005. Luego de fijar la tasa de pobreza relevante, el bucle hace que el objeto i tome valores desde 0.5 hasta 1.5 a intervalos de 0.1. Estos valores expanden la linea de pobreza que se utilizaapra calcular el FGT sobre consumo (variable “cpcf”) e ingreso (variable “ipcf”)

```
nic05 <- read.dta(data_dir %+% "Nic/2005/bases/nic05_cedlas.dta") %>%
         filter(coh_oficial==1)

df <- nic05

#linea de pobreza oficial
lp0=576.5028
```

```
for (i in seq(0.5,1.5, by=0.1)) {

  #linea de pobreza
  lp=lp0*i; lp=round(lp, d=2)

  #consumo
  print(paste(i, "*lp = ", lp, "- Consumo", sep=""))
  FGT(df=df, varname=cpcf, var_pondera=pondera, alfa=0, zeta=lp)

  #ingreso
  print(paste(i, "*lp = ", lp, "- Ingreso", sep=""))
  FGT(df=df, varname=ipcf, var_pondera=pondera, alfa=0, zeta=lp)

}
```

```
## [1] "0.5*lp = 288.25- Consumo"
## [1] "FGT(alfa=0,Z=288.25) = 12.21"
## [1] "0.5*lp = 288.25- Ingreso"
## [1] "FGT(alfa=0,Z=288.25) = 16.59"
## [1] "0.6*lp = 345.9- Consumo"
## [1] "FGT(alfa=0,Z=345.9) = 19.03"
## [1] "0.6*lp = 345.9- Ingreso"
## [1] "FGT(alfa=0,Z=345.9) = 22.91"
## [1] "0.7*lp = 403.55- Consumo"
## [1] "FGT(alfa=0,Z=403.55) = 26.23"
## [1] "0.7*lp = 403.55- Ingreso"
## [1] "FGT(alfa=0,Z=403.55) = 28.51"
## [1] "0.8*lp = 461.2- Consumo"
## [1] "FGT(alfa=0,Z=461.2) = 33.63"
## [1] "0.8*lp = 461.2- Ingreso"
## [1] "FGT(alfa=0,Z=461.2) = 33.77"
## [1] "0.9*lp = 518.85- Consumo"
## [1] "FGT(alfa=0,Z=518.85) = 40.06"
## [1] "0.9*lp = 518.85- Ingreso"
## [1] "FGT(alfa=0,Z=518.85) = 38.76"
## [1] "1*lp = 576.5- Consumo"
## [1] "FGT(alfa=0,Z=576.5) = 46.02"
## [1] "1*lp = 576.5- Ingreso"
## [1] "FGT(alfa=0,Z=576.5) = 43.62"
## [1] "1.1*lp = 634.15- Consumo"
## [1] "FGT(alfa=0,Z=634.15) = 52.18"
## [1] "1.1*lp = 634.15- Ingreso"
## [1] "FGT(alfa=0,Z=634.15) = 47.75"
## [1] "1.2*lp = 691.8- Consumo"
```

```
## [1] "FGT(alfa=0,Z=691.8) = 56.67"
## [1] "1.2*lp = 691.8- Ingreso"
## [1] "FGT(alfa=0,Z=691.8) = 51.33"
## [1] "1.3*lp = 749.45- Consumo"
## [1] "FGT(alfa=0,Z=749.45) = 60.71"
## [1] "1.3*lp = 749.45- Ingreso"
## [1] "FGT(alfa=0,Z=749.45) = 54.91"
## [1] "1.4*lp = 807.1- Consumo"
## [1] "FGT(alfa=0,Z=807.1) = 64.76"
## [1] "1.4*lp = 807.1- Ingreso"
## [1] "FGT(alfa=0,Z=807.1) = 58.29"
## [1] "1.5*lp = 864.75- Consumo"
## [1] "FGT(alfa=0,Z=864.75) = 68.21"
## [1] "1.5*lp = 864.75- Ingreso"
## [1] "FGT(alfa=0,Z=864.75) = 61.55"
```

En este caso el código imprime una larga lista de resultados en la consola. Con un cambio menor en el bucle, con las mismas sentencias, podemos generar directamente la tabla a replicar y almacenarla como un dataframe que luego es exportable facilmente a otros formatos para su presentación. Para ello generamos inicialmente 3 vectores vacíos para cada una de las columnas que tendrá la tabla. Utilizamos un contador auxiliar que irá incrementandose de a uno en cadad itereación del bucle, en las cuales iremos guardando los datos que antes se imprimian, ahora como elementos de estos objetos. Así, por ejemplo en la tercer iteración el calculo de FGT se guardará como el tercer elemento del vector consumo y el vector ingreso. Al finalizar creamos un dataframe de 3 columnas a partir de estos 3 vectores, y por ultimo agregamos la cuarta columna calculando la diferencia.

```
consumo <- c()
ingreso <- c()
linea_pobreza <- c()

j = 1

for (i in seq(0.5,1.5, by=0.1)) {

  #linea de pobreza
  lp=lp0*i; lp=round(lp, d=2)

  linea_pobreza[j] <- paste(i, "*lp", sep="")

  #consumo
  consumo[j] <- FGT(df=df, varname=cpcf, var_pondera=pondera, alfa=0, zeta=lp, quiet = TRUE)
  #ingreso
```

```
  ingreso[j] <- FGT(df=df, varname=ipcf, var_pondera=pondera, alfa=0, zeta=lp, quiet =

  j=j+1

}

tabla <- data.frame(linea_pobreza, consumo, ingreso) %>% mutate(diferencia = ingreso -
```

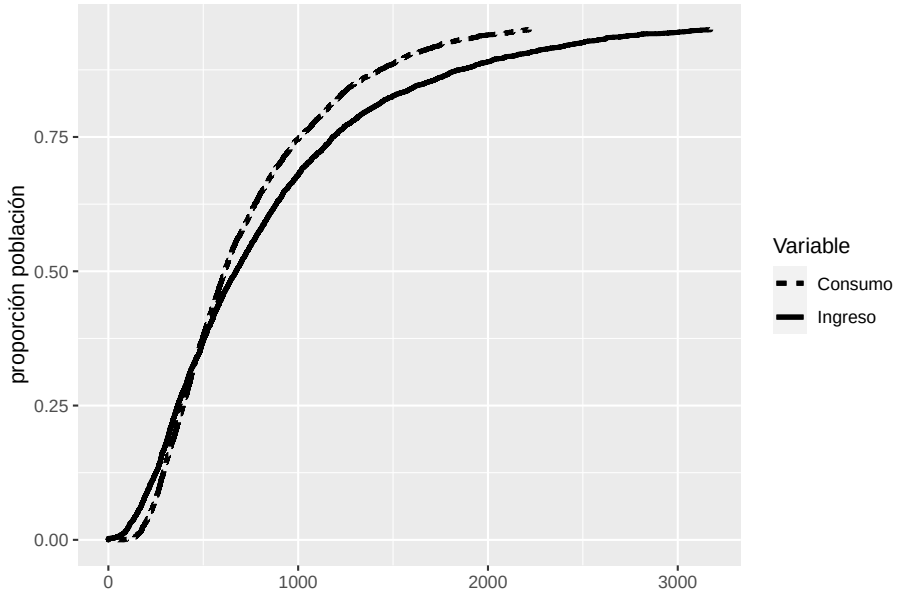
El código siguiente permite replicar la figura 4.13 del texto, que compara las funciones de distribución del ingreso y el consumo per cápita. Para ello se ordena de forma creciente por las variables de ingreso y luego de consumo, calculando en cada caso el share de población. El *"cutoff"* se utiliza para indicar qué porcentaje de las observaciones se mostrará en el gráfico.

```
##FUNCIÓN DE DISTRIBUCIÓN ACUMULADA

#ordenar según ipcf y calcular shrpop
df <- df %>% arrange(ipcf) %>% mutate(shrpop_i = cumsum(pondera)/sum(pondera))
#ordenar según cpcf y calcular shrpop
df <- df %>% arrange(cpcf) %>% mutate(shrpop_c = cumsum(pondera)/sum(pondera))

cutoff=0.95

ggplot(df %>% filter(shrpop_i < cutoff), aes(x=ipcf, y=shrpop_i, linetype="Ingreso"))+
    geom_line(size=1.2) +
    #como la condición es sobre otra variable tengo que volver a indicar dataframe y a
    geom_line(data = df %>% filter(shrpop_c < cutoff), aes(x=cpcf, y=shrpop_c, linetyp
    scale_linetype_manual(name = "Variable", values=c(Ingreso="solid", Consumo="twodash
    labs(y="proporción población", x="")
```



3.8 Pobreza por edad

###- (pág. 254-255)

El bloque de código siguiente muestra cómo puede graficarse la relación entre pobreza y edad (ver figura 4.15 del texto). Para ello luego de cargar, limpiar la base y definir la linea de pobreza, generamos dos objetos "x" e "y" vacíos donde se almacenará los valores del eje x y el eje y. Estos valores se generaran de forma iterativa mediante un bucle que incremente secuencialmente la edad en 5 años, comenzando en 0 y terminando en 80 años. En cada vuelta estaremos generando el indicador FGT condicionando a las observaciones que caigan dentro de distintos intervalos de edad. Esto equivale a dividir a la población en grupos de edad y para cada uno de ellos calcular el indicador. Al igual que en el código anterior, el contador nos permite almacenar los valores de edad y FGT como elementos de los vectores creados incialmente.

```
#indico con qué base de hogares voy a trabajar
mex06 <- read.dta(data_dir %+% "Mex/2006/bases/mex06_cedlas.dta") %>%
         filter(cohh==1, !is.na(ipcf))
df <- mex06

#linea de pobreza oficial
```

```
lp0=608.245

x <- c()
y <- c()

j=1

for (i in seq(0,80,by=5)) {

  print(paste("rango = [", i, ",", i+4, "]", sep=""))
  fgt_edad = FGT(df=df, varname=ipcf, var_pondera=pondera, condicion=paste("edad>=", i
                 alfa=0, zeta=lp, quiet = TRUE)

  x[j]=i
  y[j]=fgt_edad

  j=j+1

}
```

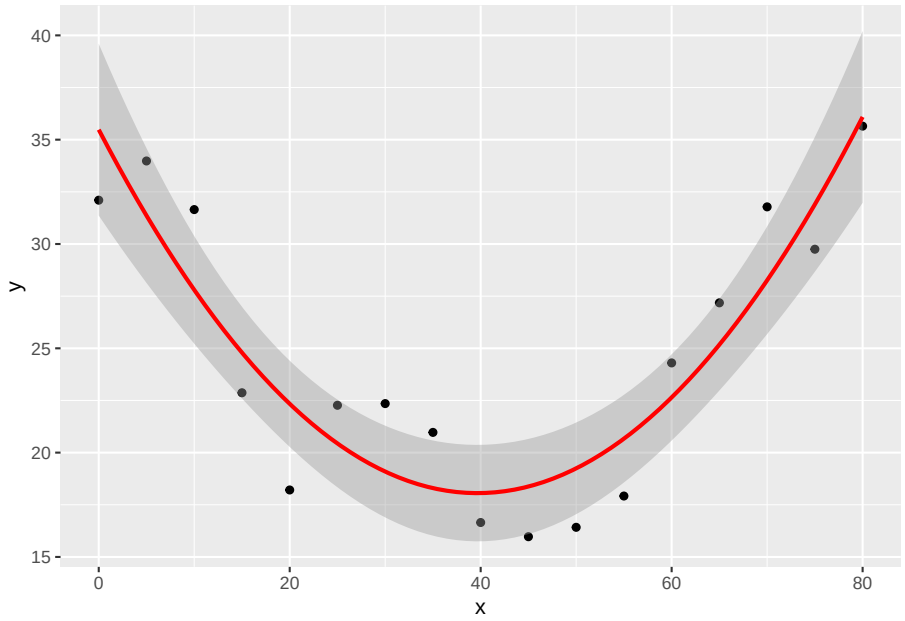
```
## [1] "rango = [0,4]"
## [1] "rango = [5,9]"
## [1] "rango = [10,14]"
## [1] "rango = [15,19]"
## [1] "rango = [20,24]"
## [1] "rango = [25,29]"
## [1] "rango = [30,34]"
## [1] "rango = [35,39]"
## [1] "rango = [40,44]"
## [1] "rango = [45,49]"
## [1] "rango = [50,54]"
## [1] "rango = [55,59]"
## [1] "rango = [60,64]"
## [1] "rango = [65,69]"
## [1] "rango = [70,74]"
## [1] "rango = [75,79]"
## [1] "rango = [80,84]"
```

Las líneas finales grafican los resultados, superponiendo a las estimaciones de pobreza una línea de regresión polinomial de orden dos. .

```
xst=x^2
aux <- data.frame(x, y, xst)

ggplot(aux, aes(x = x, y = y)) +
```

```
  geom_point() +
geom_smooth(method=lm, formula = y ~ x + I(x^2) , colour="red")
```



3.9 Significatividad estadistica

###- (pág. 255-256)

En esta última sección recreamos la técnica del *boostrap* o reseampleo para obtener errores estándares e intervalos de confianza para las estimaciones del FGT. La versión más simple del bootstrap requiere (i) tomar una muestra de tamaño N (el tamaño muestral) de la muestra original con reemplazo, (ii) computar el índice de pobreza deseado y (iii) repetir el procedimiento B veces, con B grande. Esto es lo que haremos mediante un bucle, fijaremos una cantidad de repeticiones, en las que en cada una estaremos tomando un resampleo de la muestra original de igual tamaño. De esta forma la muestrá irá cambiando en su composición aleatoriamente y por tanto permitirá generar distintos valores del FGT en cada iteración, a partir de una misma base. El comando en R `sample_n` realiza esta tarea de resampleo, seteando el dataframe, el tamaño de la nueva muestra y la opción con reposición. La expresión `nrow(df)` indica que la nueva muestra tendrá el mismo tamaño que la base original. Cada valor del indicador se almacena en el objeto “store”, sobre el que posteriormente se calcula el desvío,

la media y el tamaño. Estos son inputs necesarios para la formula que estima los intervalos de confianza de nuestras estimaciones. En este caso estimamos un intervalo de confianza del 95%.

```
per06 <- read.dta(data_dir %+% "Per/2006/bases/per06_cedlas.dta") %>%
        filter(cohh==1, !is.na(ipcf))

df <- per06

#genero un resampleo del data frame en cada iteración y para ese data frame obtengo el
store <- c()
rep=50

lp = 128.136

for (i in 1:rep) {

   df_sample <- sample_n(df, size=nrow(df), replace=T)

   fgt = FGT(df=df_sample, varname=ipcf, var_pondera=pondera, alfa=0, zeta=lp, quiet =
   store[i] = fgt

}
```

```
store
```

```
##  [1] 25.73 25.12 24.98 25.39 25.16 25.10 25.05 24.98 25.29 25.03 25.35 25.20
## [13] 25.28 25.33 25.23 25.37 25.05 24.89 25.12 24.88 25.30 25.36 24.93 25.52
## [25] 25.08 25.33 25.23 25.40 24.98 24.96 24.96 24.92 25.12 25.07 25.01 25.23
## [37] 25.33 25.13 25.12 25.41 24.92 25.33 25.54 25.18 25.25 25.11 25.45 25.07
## [49] 25.15 25.33
```

```
sd=sd(store)
mean=mean(store)
n=length(store)

#con intervalo de confianza del 95%
error <- qt(0.975,df=n-1)*sd/sqrt(n)
left <- mean - error; left
```

```
## [1] 25.13112
```

```
right <- mean + error; right
```

```
## [1] 25.23888
```

A modo de extensión es facil escribir una función que, tomando los resultados alojados en un vector y el intervalo de confianza deseado, nos devuelve directamente el cálculo del intervalo.

```
#también es posible hacer un función para calcular los intervalos de confianza
ci <- function(vector, intervalo){

  sd=sd(vector)
  mean=mean(vector)
  n= length(vector)

  error <- qt((intervalo+1)/2, df=n-1) * sd/sqrt(n)
  result <- c("lower" = mean - error, "upper" = mean + error)
  return(result)
}
```

```
ci(store, 0.90)
```

```
##    lower    upper
## 25.14005 25.22995
```

```
ci(store, 0.95)
```

```
##    lower    upper
## 25.13112 25.23888
```

```
ci(store, 0.99)
```

```
##    lower    upper
## 25.11314 25.25686
```

Chapter 4

"Apéndices del Libro Pobreza y Desigualdad en R"

4.1 - Capítulo 5

Escrito por: Cristian Bonavida

Last Update: 02/7/2021

Códigos escritos en base a los apéndices del libro "Pobreza y Desigualdad en América Latina" de Gasparini, Cicowiez y Sosa Escudero. El objeto de este material es reproducir la rutina de códigos para STATA presentada en el libro al lenguaje R. Este material es solo de caracter complementario a las explicaciones y detalles conceptuales que se presentan en el libro de texto y los apéndices

4.2 Set Inicial

Cargo las librerias, limpio enviroment, defino el path y atajo para funcion paste

```
library(dplyr)
library(tidyverse) # Data wrangling
library(tidygraph)
library(readxl)
library(ggplot2)
library(foreign)
library(TAM)
library(margins)

rm(list=ls())    #empiezo limpiando todo

"%+%" <- function(x,y) paste(x,y,sep = "")      # defino un shorcut parar concat de te
data_dir <- "C:/Users/HP/Desktop/CEDLAS - UNLP/Apendices en R/Material libro/encuestas
```

4.3 Pobreza Multidimensional

###- (pág. 334-335)

En este primer apartado se muestra cómo puede replicarse el cuadro 5.1 del libro, sobre tasas de pobreza multidimensional en Nicaragua, Perú y Uruguay. Comenzamos cargando y defiendo la base a utilizar

```
nic05 <- read.dta(data_dir %+% "Nic/2005/bases/nic05_cedlas.dta")
df <- nic05
```

Nos asegurarnos de convertir los missings que puede contener la variable pondera a 0. De esta manera al calcular estimadores ponderados los valores para estas observaciones no tienen ningun peso y evitamos que los missings afecten el nuestros cálculos.

```
df$pondera <- ifelse(is.na(df$pondera), 0, df$pondera)
```

El bloque de código siguiente asigna a todos los miembros del hogar las variables que solo están definidas para el jefe de hogar. Como se trata de variables relacionadas con las características de la vivienda, típicamente se encuentran en las bases de datos de hogares y no de personas.

En STATA la solución se propone con un bucle. En este caso aplicamos la misma lógica pero nos valemos de la función `across` que nos permite realizar un mismo cálculo para un conjunto de columnas especificadas. Previamente debemos ordenar y agrupar las observaciones por id.

```
df <- df %>% arrange(id) %>% group_by(id) %>%
      mutate(
        across(
         .cols  = c(habita, matpreca, agua, banio), #sobre qué columnas aplicar operación
         .fns   = mean,   #que operación/función queremos realizar
          na.rm = TRUE,
         .names = "{col}" #como deben llamarse las nuevas variables
                )
                  )
```

La expresión `names = "{col}"` indica que el nombre de las nuevas columnas sea el nombre de las variables orginales, por lo que las estamos sobreescribiendo.

En las siguiente bloque de código calculamos los indicadores de pobreza para cada una de las dimensiones relevantes, creando un nueva variable en el data frame para cada caso. Empleamos el comando `ifelse` cuando el cálculo es directo y `mutate` cuando se requiere de una variable auxiliar previa.

```
# (1) ipcf < 2.5 USD
df$indic1 <- ifelse(df$ipcf < 564.12, 1 ,0)

# (2) mas de 3 miembros por cuarto
df <- df %>% mutate(
              rat_miembros_cuartos = miembros/habita,
              aux = case_when(
                    (rat_miembros_cuartos>3 & !is.na(rat_miembros_cuartos)) ~ 1,
                    (rat_miembros_cuartos<=3 & !is.na(rat_miembros_cuartos)) ~ 0)) %>%
            group_by(id) %>% mutate(indic2=max(aux)) %>% select(-aux)

# (3) vivienda construida con material precario
df$indic3 <- ifelse(df$matpreca==1, 1 ,0)

# (4) vivienda sin acceso a agua potable
df$indic4 <- ifelse(df$agua==0, 1 ,0)

# (5) vivienda sin acceso a baño sanitario
df$indic5 <- ifelse(df$banio==0, 1 ,0)

# (6) educación promedio menor a 7 años solo para el jefe y conyuge
df <- df %>% mutate(aedu_avg = ifelse(jefe==1 | conyuge==1, mean(aedu, na.rm=TRUE), NA),
                    aux = ifelse(aedu_avg<7 & jefe==1, 1 ,0)) %>% group_by(id) %>%
            mutate(indic6 = max(aux))  %>% select(-aux)
```

Seguidamente obtenemos el porcentaje de personas con privaciones para cada indicador, utilizando la media ponderada.

```
weighted.mean(df$indic1, df$pondera, na.rm=TRUE)*100
```

```
## [1] 42.66527
```

```
weighted.mean(df$indic2, df$pondera, na.rm=TRUE)*100
```

```
## [1] 35.15812
```

```
weighted.mean(df$indic3, df$pondera, na.rm=TRUE)*100
```

```
## [1] 14.43805
```

```
weighted.mean(df$indic4, df$pondera, na.rm=TRUE)*100
```

```
## [1] 37.31393
```

```
weighted.mean(df$indic5, df$pondera, na.rm=TRUE)*100
```

```
## [1] 73.09805
```

```
weighted.mean(df$indic6, df$pondera, na.rm=TRUE)*100
```

```
## [1] 73.3585
```

La variable "npriv" contiene el número de privaciones de cada individuo. Para crearla utilizamos una operación a nivel de fila con el comando `rowSums` que, combinado con `across`, nos permite sumar todas las columnas especificadas. En este caso especificamos todas las columnas que comienzan con el patron "indic". Notese que a diferencia del uso anterior aqui no se realiza una misma operación repetida para cada columna sino que se especifican las columnas que se incluyen como argumento de la operación suma.

```
#contar condiciones:por fila sumo todas las columnas que comienzan con indic
df <- df %>% mutate(npriv = rowSums(across(starts_with("indic"))))
```

Una forma alternativa más intuitiva sería especificar manualmente las columnas a sumar, pero se vuelve poco efeciente en el caso de que estas sean numerosas, por lo que la posibilidad de identificar columnas por patrones se vuelve particularmente atractiva.

```
#manera alternativa
df$npriv = df$indic1 + df$indic2 + df$indic3 + df$indic4 + df$indic5 + df$indic6
```

A partir de la variable "npriv" se generan las variables "pobre1" a "pobre6" que valen 1 de acuerdo con la cantidad de privaciones que sufre cada individuo. Por ejemplo, la variable "pobre4" vale 1 para los individuos que tienen 4 o más privaciones, y 0 en caso contrario.

En cada iteración se concatena el prefijo *pobre* con el contador `i`, dandole nombre a cada nueva columna del data frame. Luego se calcula el porcentaje como la media ponderada de esta columna, se redondea e imprime el resultado.

```
#condición de pobreza segun cantidad de privaciones
for (i in 1:6){

  df[paste("pobre",i,sep="")] <- ifelse(df$npriv>=i, 1, 0)

  p = weighted.mean(df[paste("pobre",i,sep="")], df["pondera"], na.rm=TRUE)*100
  print(paste(i, " privaciones = ", round(p, d=2), "%", sep = ""))

}
```

```
## [1] "1 privaciones = 86.75%"
## [1] "2 privaciones = 73.02%"
## [1] "3 privaciones = 56.6%"
## [1] "4 privaciones = 37.67%"
## [1] "5 privaciones = 19.2%"
## [1] "6 privaciones = 4.51%"
```

4.4 Indice Bourguignon y Chakravarty (BC) - Pobreza Multidimensional

###- (pág. 335-336)

El código a continuación permite reproducir el cuadro 5.2 del texto sobre pobreza multidimensional computada con el índice de Bourguignon y Chakravarty (BC). El cómputo de dicho índice se realiza empleando solo las observaciones que tienen información para las tres dimensiones consideradas en el texto; por lo que se eliminan las observaciones con missing en al menos una de esas dimensiones. Se eligen los valores para los parametros relevantes y se fija el número de dimensiones a considerar

```
df <- df %>% filter(!is.na(ipcf), !is.na(aedu_avg), !is.na(rat_miembros_cuartos))

theta=1
alpha=2

dim_t=3    #total dimensiones
```

Posteriormente almacenamos los valores de las observacion en una lista, donde cada elemento contiene todos los valores de cada una de las 3 variables a considerar. En vectores separados almacenamos los umbrales y los pesos

```
dimension <- list( df$ipcf,                    # (1) ipcf
             1/df$rat_miembros_cuartos,        # (2) ratio de miembros por cuarto
             df$aedu_avg                       # (3) educación promedio de jefe y conyug
              )

umbral <- c(564.119195, 1/3, 7)   #valores para los umbrales de cada dimensiones
wt <- c(1, 1, 1)                  #wt correspondiente
```

El objeto brechas se define como vacío y cada uno de sus elementos se genera en las iteraciones sucesivas del bucle al comparar cada valor de la variable contra los umbrales fijados. El objeto "suma_brechas" se crea como un vector único con valores 0, y luego se reemplaza iterativamente para computar la suma de brechas. El bucle itera n veces en total, siendo n la cantidad de dimensiones relevadas. En cada iteracion replica la formula de BC para cada dimensión.

```
#defino la lista brechas como vacia para generar cada uno de sus elementos en el bucle
brecha <- list()

suma_brechas <- c(rep(0, nrow(df)))

for (i in 1:dim_t) {

  #generar brechas a partir de valores de las dimensiones vs umbrales
  brecha[[i]] <- ifelse(dimension[[i]]<umbral[i], 1-dimension[[i]]/umbral[i], 0)

  #construir brechas ponderadas
  brecha[[i]] <- wt[i]/dim_t * (brecha[[i]]^theta)

  #computar suma de las brechas. Suma_brechas será = 0 solo si todas las brechas son 0
  suma_brechas = suma_brechas + brecha[[i]]

}
```

Finalmente se calcula, para cada individuo, la suma de las brechas ponderadas elevadas a la potencia *theta*, siempre que la suma de las brechas sea distinta

de cero. Por último, se computa el índice BC como el cociente entre la suma ponderada de las brechas individuales almacenadas en la variable suma_brechas y la población de referencia.

```
suma_brechas = ifelse(suma_brechas!=0, suma_brechas^(alpha/theta), suma_brechas)

BC = round(sum(suma_brechas*df$pondera)/sum(df$pondera), d=3)

print(paste("BC =", BC))
```

```
## [1] "BC = 0.08"
```

4.5 Indice Alkire y Foster (AF) - Pobreza Multidimensional

###- (pág. 337-338)

A continuación se replica la formula de Alkire y Foster que permite replicar el cuadro 5.3 del texto. Las primeras lineas son identicas al caso anterior, cambiando los parametros de interés y agregando la lista "pobre" como objeto vacío.

```
k=2
alpha=2

dim_t=3

dim <- list( df$ipcf,
             1/df$rat_miembros_cuartos,
             df$aedu_avg
)

umbral <- c(564.119195, 1/3, 7)
wt <- c(1, 1, 1)

#defino la lista "brechas" Y pobre como vacia para generar cada uno de sus elementos en el bucle
brecha <- list()
pobre <- list()
```

Nuevamente el bucle itera sobre las n dimensiones fijada generando los valores de brecha para cada observación y ahora también completando el objeto binario

"pobre" según el valor que toma la brecha. Al finalizar el bucle se construye el objeto "npriv" que contiene el número de dimensiones en que cada individuo fue identificado como pobre (vale cero para los individuos no pobres). El objeto "pobre_k" vale 1 para los individuos que son pobres en, al menos, k dimensiones.

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

    #generar brechas a partir de valores de las dimensiones vs umbrales
    brecha[[i]] <- ifelse(dim[[i]]<umbral[i], (1-dim[[i]]/umbral[i])^alpha, 0)

    #identificar si es pobre en dimensión i
    pobre[[i]] <- ifelse(brecha[[i]]!=0, 1, 0)

  }


#identificar pobres en al menos k dimensiones
npriv = pobre[[1]] + pobre[[2]] + pobre[[3]]
pobre_k = ifelse(npriv>=k, 1, 0)
```

La línea siguiente genera la variable "suma_brechas" que, como antes, se emplea luego para almacenar la suma de las brechas en cada una de las dimensiones consideradas. El bucle constuye para cada dimensión los objetos necesarios para replicar la formula de AK

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

    #brechas positivas solo si el número de privaciones mayor a k
    brecha[[i]] <- ifelse(pobre_k!=1, 0, brecha[[i]])

    #construir brechas ponderadas
    brecha[[i]] <- wt[i] * brecha[[i]]

    #computar suma de las brechas. Suma_brechas será = 0 solo si todas las brechas son
    suma_brechas = suma_brechas + brecha[[i]]

}
```

Por último se computa y se redondean los valores del índice de AF

```
AK = round(sum(suma_brechas*df$pondera)/(dim_t*sum(df$pondera)), d=5)

print(paste("AK =", AK))
```

```
## [1] "AK = 0.12319"
```

4.6 Perfiles de Pobreza

###- (pág. 338-339)

El bloque de código a continuación puede emplearse para computar el perfil de pobreza monetaria para vivienda y servicios que se muestra en el cuadro 5.8. El código del ejemplo se aplica a la EPH (Encuesta Permanente de Hogares) de Paraguay para el año 2007. Luego de cargar y definir la base, la tercer linea de código genera la variable "hh" que vale 1 para una única observación de cada hogar, a partir de indentificar las observaciones duplicadas. El comando `duplicated` asigna valor `FALSE` a la primera observación del por hogar, y `TRUE` a todo el resto.

```
pry07 <- read.dta(data_dir %+% "Par/2007/bases/par07_cedlas.dta")
df <- pry07

df$hh <- ifelse(duplicated(df$id)==FALSE, 1,0)
```

Luego generamos en el data frame la variable indicativa de pobreza monetaria y a partir de ella computamos para el grupo de pobres y no pobres, el promedio de las variables "habita", "matpreca", "agua", "banio" y "elect", indicado la proporción de personas que cuenta con estos servicios.

```
df$pobre <- ifelse(df$ipcf<205970.366, 1, 0)

df %>% filter(hh==1 & !is.na(pobre)) %>% group_by(pobre) %>%
       summarise( mean_habita = weighted.mean(habita, pondera, na.rm = TRUE),
                  mean_matpreca = weighted.mean(matpreca,pondera, na.rm = TRUE),
                  mean_banio = weighted.mean(banio, pondera, na.rm = TRUE),
                  mean_agua = weighted.mean(agua, pondera, na.rm = TRUE),
                  mean_elect = weighted.mean(elect, pondera, na.rm = TRUE))
```

```
## # A tibble: 2 x 6
##   pobre mean_habita mean_matpreca mean_banio mean_agua mean_elect
##   <dbl>       <dbl>         <dbl>      <dbl>     <dbl>      <dbl>
## 1     0        3.36        0.0108      0.761     0.975      0.978
## 2     1        2.53        0.0414      0.364     0.905      0.898
```

Con el comando `ttest` buscamos evaluar la significatividad estadística de estas diferencias de medias entre pobres y no pobres para las variables incluidas. Para esto elegimos el nivel de confianza y definimos un bucle que itera sobre

cada variable respectiva del data frame. Dentro de él construimos un objeto "x" que contiene los valores para estas variables sólo para una única observación por hogar y para el grupo de pobres y otro objeto "y" con los mismos datos para el grupo de no pobres. A partir de estos objetos se evaluan las significatividad de la diferencia de medias entre ambos grupos y se reporta si su p-valor es mayor al nivel de confianza fijado.

```
set_confidence = 95
confidence = 1 - set_confidence/100

dim <- c("habita", "matpreca", "banio", "agua", "elect")
for (i in dim){

  print(i)
  x <- df[df$pobre==1 & df$hh==1, colnames(df)==i]
  y <- df[df$pobre==0 & df$hh==1, colnames(df)==i]

  test = t.test(x,y)
  print(paste("No significative mean diff:", test$p.value > confidence))

  }
```

```
## [1] "habita"
## [1] "No significative mean diff: FALSE"
## [1] "matpreca"
## [1] "No significative mean diff: FALSE"
## [1] "banio"
## [1] "No significative mean diff: FALSE"
## [1] "agua"
## [1] "No significative mean diff: FALSE"
## [1] "elect"
## [1] "No significative mean diff: FALSE"
```

4.7 Perfiles de Pobreza Condicionados

###- (pág. 340-341)

El bloque de código siguiente permite replicar el cuadro 5.12, que muestra perfiles condicionados de pobreza. En el ejemplo se emplea la encuesta de México para el año 2006. Luego de cargar la base eliminamos las observaciones incohrentes y al igual que antes generamos la variable indicativa de pobreza monetaria línea de 2.5 dólares. Las lineas siguientes agregan nuevas variables al data

frame que suman la cantidad de individuos que pertenecen a distintos grupos etarios dentro de cada hogar, calculan el ratio de miembros por habitaciones y los valores de educación y edad al cuadrado

```
#cargo base
mex06 <- read.dta(data_dir %+% "Mex/2006/bases/mex06_cedlas.dta")
df <- mex06

df <- df %>% filter(cohh==1)
df$pobre <- ifelse(df$ipcf<608.24533, 1, 0)

#número de miembros en cada grupo
df <- df %>% arrange(id) %>% group_by(id) %>%
             mutate( miembros_edad_0015 = sum(ifelse(edad<=15, 1, 0)),
                     miembros_edad_1625 = sum(ifelse(edad %in% (16:25), 1, 0)),
                     miembros_edad_2640 = sum(ifelse(edad %in% (26:40), 1, 0)),
                     miembros_edad_4160 = sum(ifelse(edad %in% (41:64), 1, 0)),
                     miembros_edad_65mas= sum(ifelse(edad>=65, 1, 0)),

                     rat_miembros_cuartos = miembros/habita,

                     aedu2=aedu^2,
                     edad2=edad^2 )
```

Las líneas siguientes contienen la sentencia que estima, para los jefes de hogar, el modelo probit para la probabilidad de ser pobre. Para ello empleamos el comando `glm` en el cual definimos la variable independiente y todo el conjunto de regresores, indicamos el data frame referido y la familia de modelos que buscamos estimar. Esta estimación la guardamos en el objeto "probit" que luego visualizamos con un `summary`

Para computar los efectos marginales para el rango 0 a 22 años de educación del jefe de hogar, empleamos el comando `margins`. Para ello indicamos donde almacenamos nuestra estimación (objeto probit), para qué variables deseamos calcular los efectos (aedu) y sobre qué valores evaluarlos (0:22).

```
store <- summary(margins(probit, variables = "aedu", at = list(aedu = 0:22)))
store
```

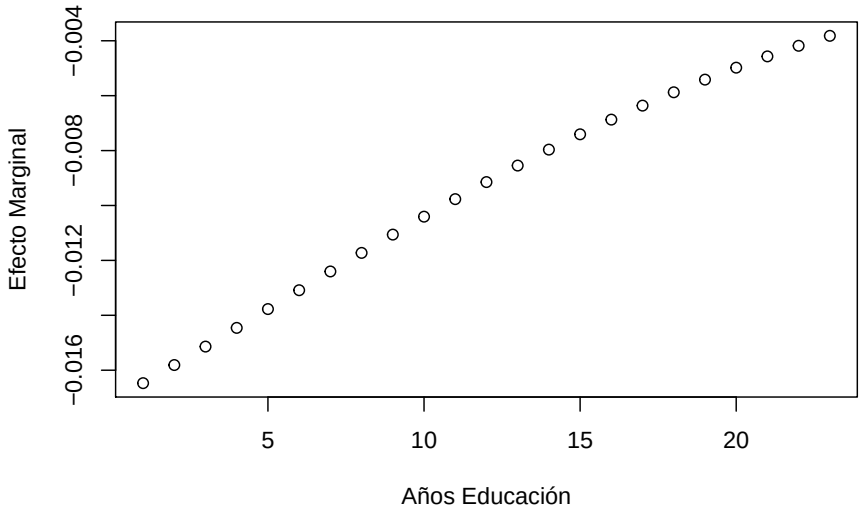
```
##  factor    aedu     AME     SE        z      p   lower   upper
##    aedu  0.0000 -0.0165 0.0029  -5.6272 0.0000 -0.0222 -0.0107
##    aedu  1.0000 -0.0158 0.0028  -5.7352 0.0000 -0.0212 -0.0104
##    aedu  2.0000 -0.0151 0.0026  -5.8741 0.0000 -0.0202 -0.0101
##    aedu  3.0000 -0.0145 0.0024  -6.0478 0.0000 -0.0191 -0.0098
##    aedu  4.0000 -0.0138 0.0022  -6.2616 0.0000 -0.0181 -0.0095
##    aedu  5.0000 -0.0131 0.0020  -6.5223 0.0000 -0.0170 -0.0092
```

```
##    aedu  6.0000 -0.0124 0.0018  -6.8395 0.0000 -0.0160 -0.0088
##    aedu  7.0000 -0.0117 0.0016  -7.2263 0.0000 -0.0149 -0.0085
##    aedu  8.0000 -0.0111 0.0014  -7.7006 0.0000 -0.0139 -0.0082
##    aedu  9.0000 -0.0104 0.0013  -8.2882 0.0000 -0.0129 -0.0079
##    aedu 10.0000 -0.0098 0.0011  -9.0270 0.0000 -0.0119 -0.0076
##    aedu 11.0000 -0.0091 0.0009  -9.9743 0.0000 -0.0109 -0.0074
##    aedu 12.0000 -0.0085 0.0008 -11.2219 0.0000 -0.0100 -0.0071
##    aedu 13.0000 -0.0080 0.0006 -12.9242 0.0000 -0.0092 -0.0068
##    aedu 14.0000 -0.0074 0.0005 -15.3614 0.0000 -0.0084 -0.0065
##    aedu 15.0000 -0.0069 0.0004 -19.0911 0.0000 -0.0076 -0.0062
##    aedu 16.0000 -0.0064 0.0003 -25.3519 0.0000 -0.0069 -0.0059
##    aedu 17.0000 -0.0059 0.0002 -37.0835 0.0000 -0.0062 -0.0056
##    aedu 18.0000 -0.0054 0.0001 -56.6339 0.0000 -0.0056 -0.0052
##    aedu 19.0000 -0.0050 0.0001 -52.3155 0.0000 -0.0052 -0.0048
##    aedu 20.0000 -0.0046 0.0001 -32.5281 0.0000 -0.0048 -0.0043
##    aedu 21.0000 -0.0042 0.0002 -21.7719 0.0000 -0.0046 -0.0038
##    aedu 22.0000 -0.0038 0.0002 -15.9736 0.0000 -0.0043 -0.0034
```

Esta información la guardamos en un objeto llamado "store", del cual nos interesa recuperar el valor de los coficientes almacenados bajo el nombre AME (Average Mean Effect). Con ellos generamos un vector "y" que denota los efectos para cada valor de años de educación, los cuale guardamos en el vector "x". Finalmente graficamos la relación.

```
y = store$AME
x = seq(0:22)

plot(x, y,
     ylab = "Efecto Marginal",
     xlab = "Años Educación")
```



Una forma más directa de graficar los efectos marginales es a partir del comando `cplot` de la familia margins, que estima automaticamente estos mismos valores a partir de la estimación del modelo probit

```
#cplot(probit, "aedu", what = "effect", main = "Average Marginal Effect of Weight")
```