Manual de procesamiento del indicador de pobreza multidimensional mediante el uso de estimación de áreas pequeñas para México 2020

Andrés Gutiérrez[1], Stalyn Guerrero[2]

2024-07-31

[1]Experto Regional en Estadísticas Sociales - Comisión Económica para América Latina y el Caribe (CEPAL) - andres.gutierrez@cepal.org

[2]Consultor - Comisión Económica para América Latina y el Caribe (CEPAL), guerrerostalyn@gmail.com

Índice

Índice de figuras

Índice de cuadros

Introducción

Este manual se desarrolla como una herramienta para detallar el paso a paso seguido por la Comisión Económica para América Latina y el Caribe (CEPAL) en la obtención de estimaciones del indicador multidimensional de pobreza en México. El proceso utiliza técnicas de estimación de áreas pequeñas y métodos de Monte Carlo. Aunque el procedimiento se describe con los códigos de 2020, el proceso para 2015 es similar, con pequeñas variaciones. Sin embargo, la esencia y los principios fundamentales son los mismos en ambos periodos.

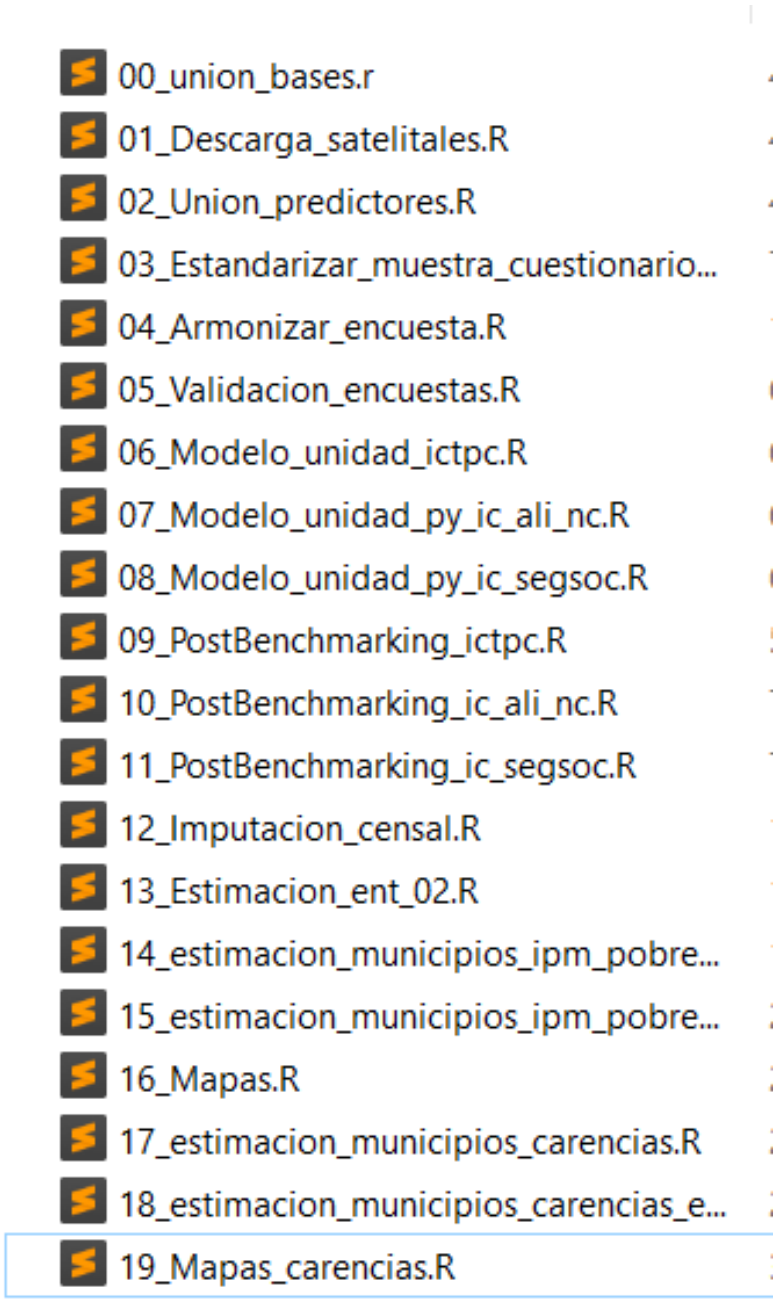
Respositodio de códigos

- En el siguiente enlace encontrará las rutinas de R desarrolladas para la estimación del IPM Descargar

Estructura del Proyecto

Para el desarrollo de los scripts, se estructuró un proyecto en *R* que cuenta con la siguiente organización:

- **`rcodes`**: En esta carpeta se encuentran los diferentes códigos desarrollados para cada año. Dentro de ella, hay subcarpetas específicas para 2020 y 2015, correspondientes a los años en los que se realizaron los cálculos de los indicadores de carencias y del Índice de Pobreza Multidimensional (IPM). Dentro de estas subcarpetas, los scripts están enumerados en orden de ejecución, como se muestra en la imagen a continuación:



Cada subcarpeta contiene una serie de scripts organizados secuencialmente, asegurando que el proceso de cálculo y análisis se realice de manera ordenada y reproducible. Esta estructura facilita la identificación y ejecución de cada paso necesario para obtener los resultados del análisis de pobreza multidimensional.

- `input`: Esta carpeta contiene dos subcarpetas, una para 2015 y otra para 2020. En cada una de estas encontraremos la base de datos de la ENIGH, la encuesta ampliada del censo para el 2020 o la encuesta intercensal para el 2015, además

de las variables predictoras a nivel de municipios. También incluye un archivo en formato CSV que contiene las líneas de pobreza para el año que se esté procesando.

- **`output`**: Esta carpeta sigue la misma estructura, con subcarpetas para 2015 y 2020. En cada una de estas subcarpetas encontraremos la carpeta de iteraciones, donde están disponibles los resultados de las iteraciones realizadas por cada estado para cada uno de los municipios. También dispone de una subcarpeta de modelos que contiene los modelos estimados para cada una de las carencias, así como algunos gráficos de validación. Además, encontraremos la carpeta de intermedias necesarias para el procesamiento de información o para el almacenamiento de algunos resultados.
- **`shapefile`**: Aquí están disponibles los shapefiles para la realización de mapas y descarga de información satelital, que son utilizados como covariables en la implementación del modelo.

Esta organización permite mantener un flujo de trabajo claro y estructurado, donde cada script y conjunto de datos desempeñan un papel específico en el proceso general, desde la preparación de datos hasta la generación de resultados finales.

Librerias de *R* y otros insumos

En el siguiente apartado, se describe el conjunto de librerías utilizadas para el desarrollo de este proyecto, así como una breve descripción de las bases de datos empleadas.

Librerías Utilizadas

Manipulación y Transformación de Datos

- **`tidyverse`**: Conjunto de paquetes (incluyendo `dplyr`, `ggplot2`, `tibble`, `readr`, `purrr`, `tidyr`, y `stringr`) que facilitan la manipulación y visualización de datos de manera coherente y eficiente.
- **`data.table`**: Proporciona herramientas rápidas y eficientes para la manipulación de grandes conjuntos de datos tabulares.
- **`dplyr`**: Parte del `tidyverse`, facilita la manipulación de datos mediante verbos intuitivos como `select`, `filter`, `mutate`, `summarize`, y `arrange`.
- **`magrittr`**: Introduce el operador `%>%`, permitiendo una escritura de código más clara y encadenada.
- **`purrr`**: Extiende las capacidades de programación funcional para trabajar con listas y vectores.
- **`furrr`**: Permite realizar operaciones paralelas utilizando `purrr` y `future`.
- **`stringr`**: Simplifica la manipulación de cadenas de caracteres mediante funciones intuitivas.
- **`labelled`**: Facilita la manipulación de datos etiquetados, comúnmente utilizados en encuestas y datos sociológicos.

Lectura y Escritura de Datos

- **openxlsx**: Permite la creación, lectura y manipulación de archivos Excel sin depender de software adicional.
- **haven**: Permite leer y escribir datos en formatos usados por otros programas estadísticos como SPSS, Stata y SAS.
- **readstata13**: Especializado en la lectura de archivos Stata versión 13, asegurando compatibilidad con datos antiguos.

Análisis de Datos de Encuestas

- **survey**: Proporciona herramientas para el análisis de datos de encuestas complejas, incluyendo ponderaciones y diseños de muestras.
- **srvyr**: Ofrece una interfaz más amigable basada en `dplyr` para trabajar con el paquete `survey`.
- **TeachingSampling**: Incluye métodos y herramientas para realizar muestreo en investigaciones educativas.
- **samplesize4surveys**: Facilita el cálculo del tamaño de muestra necesario para encuestas, asegurando resultados estadísticamente significativos.
- **convey**: Extiende `survey` para analizar medidas de desigualdad y pobreza en datos de encuestas.

Modelado y Análisis Estadístico

- **rstan**: Interfaz de R para Stan, que realiza modelado bayesiano avanzado, permitiendo la creación de modelos complejos.
- **lme4**: Proporciona herramientas para ajustar y analizar modelos lineales y no lineales de efectos mixtos.
- **car**: Incluye diversas herramientas para la regresión aplicada y diagnósticos de modelos.
- **randomForest**: Implementa algoritmos de bosque aleatorio para clasificación y regresión.
- **caret**: Ofrece una amplia gama de herramientas para la creación y validación de modelos de aprendizaje automático.
- **nortest**: Contiene pruebas para evaluar la normalidad de los datos, esencial en muchos análisis estadísticos.

Visualización de Datos

- **ggplot2**: Parte del `tidyverse`, es una potente herramienta para la visualización de datos basada en la gramática de gráficos.
- **DataExplorer**: Simplifica la exploración inicial y la generación de reportes de datos.

- `thematic`: Facilita la personalización de temas gráficos en `ggplot2`, permitiendo una estética consistente.
- `patchwork`: Permite combinar múltiples gráficos de `ggplot2` en una única visualización coherente.
- `tmap`: Especializado en la creación de mapas temáticos, útil para visualizar datos geoespaciales.
- `sf`: Proporciona una estructura eficiente para manipular datos espaciales, facilitando la integración con `ggplot2` y `tmap`.

Informes y Reproducibilidad

- `printr`: Mejora el formato de la impresión de resultados en R Markdown, haciendo los informes más legibles.
- `knitr`: Herramienta clave para la creación de informes dinámicos y reproducibles, integrando código y texto.

Integración con Otros Lenguajes

- `reticulate`: Facilita la interoperabilidad entre R y Python, permitiendo ejecutar código de Python dentro de un entorno de R.

Bases de Datos Utilizadas

Para este proyecto, se emplearon diversas bases de datos que incluyen:

- **Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH)**: Proporciona información detallada sobre los ingresos y gastos de los hogares en México. Esta encuesta es crucial para entender los patrones de consumo y el bienestar económico de la población.
- **Formulario Ampliado del Censo 2020**: Ofrece datos sociodemográficos detallados a nivel de hogar y persona. Esta fuente es esencial para capturar una amplia gama de variables necesarias para el análisis multidimensional de la pobreza.
- **Encuesta Intercensal de 2015**: Complementa la información del censo con datos adicionales recopilados entre periodos censales. Proporciona una actualización intermedia de las condiciones demográficas y socioeconómicas.
- **Imágenes Satelitales**: Proveen datos geoespaciales esenciales para capturar la diversidad de las condiciones de vida en México. Estas imágenes ayudan a incorporar información espacial detallada en los modelos de estimación.

Estas librerías y bases de datos son fundamentales para la implementación de la metodología de estimación del Índice de Pobreza Multidimensional (IPM) y otras carencias en los distintos municipios de México.

Indicadores estimados

En esta sección se presentan los diferentes indicadores utilizados para medir y analizar la situación de pobreza y vulnerabilidad en la población. A continuación, se describen los indicadores estimados: ### Descripción de las Funciones de Estimación en R

1. **Población en situación de pobreza multidimensional (I)**
   - **Scripts utilizados**:
     - *13_Estimacion_ent_02.R*
     - *14_estimacion_municipios_ipm_pobreza.R*
     - *15_estimacion_municipios_ipm_pobreza_error.R*
2. **Población en situación de pobreza moderada**
   - **Scripts utilizados**:
     - *13_Estimacion_ent_02.R*
     - *14_estimacion_municipios_ipm_pobreza.R*
     - *15_estimacion_municipios_ipm_pobreza_error.R*
3. **Población en situación de pobreza extrema**
   - **Scripts utilizados**:
     - *13_Estimacion_ent_02.R*
     - *14_estimacion_municipios_ipm_pobreza.R*
     - *15_estimacion_municipios_ipm_pobreza_error.R*
4. **Población vulnerable por carencias sociales (II)**
   - **Scripts utilizados**:
     - *13_Estimacion_ent_02.R*
     - *14_estimacion_municipios_ipm_pobreza.R*
     - *15_estimacion_municipios_ipm_pobreza_error.R*
5. **Población vulnerable por ingresos (III)**
   - **Scripts utilizados**:
     - *13_Estimacion_ent_02.R*
     - *14_estimacion_municipios_ipm_pobreza.R*
     - *15_estimacion_municipios_ipm_pobreza_error.R*
6. **Población no pobre multidimensional y no vulnerable (IV)**
   - **Scripts utilizados**:
     - *13_Estimacion_ent_02.R*
     - *14_estimacion_municipios_ipm_pobreza.R*
     - *15_estimacion_municipios_ipm_pobreza_error.R*
7. **Población con al menos una carencia social (tol_ic_1)**
   - **Scripts utilizados**:
     - *17_estimacion_municipios_carencias.R*
     - *18_estimacion_municipios_carencias_error.R*
8. **Población con al menos tres carencias sociales (tol_ic_2)**
   - **Scripts utilizados**:
     - *17_estimacion_municipios_carencias.R*

00_union_bases.R

Para la ejecución del presente archivo, debe abrir el archivo **00_union_bases.R** disponible en la ruta *Rcodes/2020/00_union_bases.R*.

El código comienza limpiando el entorno de R y cargando varias bibliotecas esenciales para la manipulación y análisis de datos. Posteriormente, define un conjunto de variables a validar y obtiene listas de archivos de datos en formato `.dta` correspondientes a diferentes conjuntos de datos del censo 2020. A continuación, el código itera sobre estos archivos para leer, combinar y almacenar los datos de cada estado en archivos `.rds` individuales. Estos datos se combinan en un único dataframe que se guarda para su uso posterior.

En la sección final, el código se centra en los datos de la ENIGH 2020. Lee los archivos de hogares y pobreza, y los combina utilizando un `inner_join`. Se seleccionan y renombran variables clave para asegurar la consistencia en el análisis. Finalmente, el dataframe combinado se guarda en un archivo `.rds` para facilitar el acceso y análisis futuros.

Limpieza del Entorno y Carga de Bibliotecas

Se limpia el entorno de R eliminando todos los objetos y se ejecuta el recolector de basura para liberar memoria.

```
rm(list = ls())
gc()
```

Se cargan varias bibliotecas esenciales para la manipulación y análisis de datos, incluyendo `tidyverse`, `data.table`, `haven` y otras.

```
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(DataExplorer)
library(haven)
library(purrr)
library(furrr)
```

```
library(labelled)
cat("\f")
```

Configuración de la Memoria

Se define un límite para la memoria RAM a utilizar, en este caso 250 GB.

```
memory.limit(250000000)
```

Definición de Variables y Obtención de Archivos

Para llevar a cabo la validación y procesamiento de datos del censo 2020, se define un conjunto de variables que serán validadas. Estas variables incluyen indicadores clave como rezago educativo (`ic_rezedu`), acceso a servicios de salud (`ic_asalud`), seguridad social (`ic_segsoc`), calidad de la vivienda (`ic_cv`), servicios básicos en la vivienda (`ic_sbv`), acceso a alimentación nutritiva y de calidad (`ic_ali_nc`), y el índice de pobreza multidimensional (`ictpc`).

```
validar_var <- c(
  "ic_rezedu",
  "ic_asalud",
  "ic_segsoc",
  "ic_cv",
  "ic_sbv",
  "ic_ali_nc",
  "ictpc"
)
```

Posteriormente, se obtienen listas de archivos con extensión `.dta`, que corresponden a diferentes conjuntos de datos del censo 2020. Estos archivos se organizan en tres categorías principales:

1. **Archivos de muestra del censo 2020 por estado**: Se buscan archivos en la ruta `../input/2020/muestra_ampliada/SegSocial/SegSoc/` que contengan información de seguridad social.

```
file_muestra_censo_2020_estado <- list.files(
  "../input/2020/muestra_ampliada/SegSocial/SegSoc/",
  full.names = TRUE,
  pattern = "dta$"
)
```

2. **Archivos complementarios de muestra del censo 2020 por estado**: Se buscan archivos adicionales en la ruta `../input/2020/muestra_ampliada/SegSocial/Complemen` que contienen datos complementarios de seguridad social.

```
file_muestra_censo_2020_estado_complemento <- list.files(
  "../input/2020/muestra_ampliada/SegSocial/Complemento_SegSoc/",
  full.names = TRUE,
  pattern = "dta$"
)
```

3. **Archivos de cuestionario ampliado del censo 2020 por estado**: Se buscan archivos en la ruta `../input/2020/muestra_ampliada/IndicadoresCenso/` que incluyen indicadores generales del censo.

```
muestra_cuestionario_ampliado_censo_2020_estado <- list.files(
  "../input/2020/muestra_ampliada/IndicadoresCenso/",
  full.names = TRUE,
  pattern = "dta$"
)
```

Estas listas de archivos permiten acceder y gestionar de manera eficiente los datos necesarios para el análisis y validación de los indicadores de pobreza y carencias sociales del censo 2020.

Lectura y Combinación de Datos del Censo

Este bloque de código está diseñado para leer, combinar y guardar datos provenientes de múltiples archivos del censo. El objetivo es consolidar la información de diferentes fuentes en un único dataframe y almacenar los resultados intermedios y finales en archivos `.rds`.

```
df <- data.frame()

for (ii in 1:32) {
  muestra_censo_2020_estado_ii <-
    read_dta(file_muestra_censo_2020_estado[ii])
  muestra_censo_2020_estado_complemento_ii <-
    read_dta(file_muestra_censo_2020_estado_complemento[ii]) %>%
    mutate(id_per = id_persona)
  muestra_cuestionario_ampliado_censo_2020_estado_ii <-
    read_dta(muestra_cuestionario_ampliado_censo_2020_estado[ii])

  muestra_censo <-
    inner_join(muestra_censo_2020_estado_ii,
               muestra_censo_2020_estado_complemento_ii) %>%
    select(-tamloc) %>%
    inner_join(muestra_cuestionario_ampliado_censo_2020_estado_ii)
```

```
  saveRDS(muestra_censo,
          paste0("../output/2020/muestra_censo/depto_", ii, ".rds"))

  df <- bind_rows(df, muestra_censo)
  cat(file_muestra_censo_2020_estado[ii], "\n")
}
```

Se guarda el dataframe combinado en un archivo `.rds`.

```
saveRDS(df, file = "../output/2020/muestra_cuestionario_ampliado.rds")
```

Lectura y Combinación de Datos de la ENIGH

Este bloque de código está diseñado para leer, verificar y combinar datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH) 2020. La finalidad es consolidar la información de diferentes archivos de datos en un único dataframe para su posterior análisis.

```
enigh_hogares <- read_dta("../input/2020/enigh/base_hogares20.dta")
enigh_pobreza <- read_dta("../input/2020/enigh/pobreza_20.dta") %>%
  select(-ent)
```

Se leen los datos de hogares y pobreza de la ENIGH 2020.

```
n_distinct(enigh_pobreza$folioviv)
n_distinct(enigh_hogares$folioviv)
```

Se verifica la unicidad de los identificadores de viviendas.

```
enigh <- inner_join(
  enigh_pobreza,
  enigh_hogares,
  by = join_by(
    folioviv, foliohog, est_dis, upm,
    factor, rururb, ubica_geo
  ),
  suffix = c("_pers", "_hog")
)
```

Se combinan los datos de pobreza y hogares utilizando un `inner_join`.

Selección y Renombramiento de Variables

En esta sección, se seleccionan y renombran algunas variables clave del dataframe combinado para asegurar la consistencia en el análisis y facilitar su manipulación posterior.

```
enigh$ic_ali_nc
enigh$ictpc_pers
enigh$ictpc <- enigh$ictpc_pers
enigh$ic_segsoc
```

- `enigh$ic_ali_nc`: Selección de la variable que indica carencia por acceso a la alimentación nutritiva y de calidad.
- `enigh$ictpc_pers`: Selección de la variable que representa el ingreso corriente total por persona.
- `enigh$ictpc <- enigh$ictpc_pers`: Renombramiento de `ictpc_pers` a `ictpc` para simplificar su uso en análisis posteriores.
- `enigh$ic_segsoc`: Selección de la variable que indica carencia por acceso a la seguridad social.

Guardado del DataFrame Combinado

Finalmente, se guarda el dataframe combinado en un archivo `.rds` para facilitar el acceso y análisis futuros.

```
saveRDS(enigh, file = "../output/2020/enigh.rds")
```

Se guarda el dataframe `enigh` en un archivo `.rds` en la ruta especificada `../output/2020/enigh.rds`.

01_Descarga_satelitales.R

Para la ejecución del presente archivo, debe abrir el archivo **01_Descarga_satelitales.R** disponible en la ruta *Rcodes/2020/01_Descarga_satelitales.R*.

Este código en R se enfoca en el procesamiento y análisis de datos geoespaciales utilizando una combinación de bibliotecas y herramientas de Google Earth Engine. La primera parte del script configura el entorno de trabajo, carga las librerías necesarias, y establece la conexión con Python mediante la biblioteca `reticulate`, esencial para trabajar con `rgee`, que es la interfaz de R para Google Earth Engine.

En la segunda parte, se realiza la limpieza y transformación de datos espaciales. Se cargan y transforman archivos shapefiles de México a un sistema de referencia de coordenadas adecuado y se convierten a un formato `MULTIPOLYGON`. Luego, se extraen y procesan datos de imágenes satelitales para diversas variables como luminosidad, urbanización, y distancia a servicios de salud utilizando colecciones de imágenes de Google Earth Engine. Estos datos se agrupan por región y se guardan en archivos RDS para su análisis posterior. Finalmente, se combinan todos los datos procesados en un único dataframe y se escalan para prepararlos para su uso en modelos o análisis adicionales.

Lectura de Librerías

Para comenzar, se realiza una limpieza del entorno de trabajo en R mediante la eliminación de todos los objetos existentes usando `rm(list = ls())`. Esto asegura que no haya interferencias de sesiones anteriores y que el entorno esté limpio. Luego, se establece un límite de memoria de 500,000 MB con `memory.limit(500000)`, lo que permite manejar grandes volúmenes de datos sin que R se quede sin memoria, preparando el entorno para análisis intensivos.

A continuación, se cargan varias librerías esenciales para el análisis de datos. `tidyverse` se utiliza para la manipulación y visualización de datos de manera eficiente, mientras que `sampling` proporciona herramientas para realizar muestreo estadístico. `rgee` permite la conexión con Google Earth Engine para realizar análisis geoespaciales avanzados. `sf` es fundamental para manejar datos geográficos, y `concaveman` ayuda a generar polígonos cóncavos a partir de conjuntos de puntos. Además, `geojsonio` facilita la conversión

entre diversos formatos de datos geoespaciales, `magrittr` mejora la legibilidad del código con el operador `%>%`, `furrr` habilita la programación paralela utilizando `purrr`, y `readr` se emplea para la lectura eficiente de archivos de datos en formatos como CSV.

```
rm(list = ls())

memory.limit(500000)

library(tidyverse)
library(sampling)
library(rgee) # Conexión con Google Earth Engine
library(sf) # Paquete para manejar datos geográficos
library(concaveman)
library(geojsonio)
library(magrittr)
library(furrr)
library(readr)
```

configuración inicial de Python

Para comenzar, se obtiene la lista de entornos conda disponibles y se selecciona el entorno llamado "rgee_py", que es específico para la conexión con Google Earth Engine mediante `rgee`. Esto asegura que se utilice el entorno de Python correcto para las operaciones geoespaciales.

Luego, se configura `reticulate` para usar el entorno de Python especificado. La función `py_config()` verifica la configuración de Python y se carga la librería `reticulate` para habilitar la conexión entre R y Python, lo que permite ejecutar código Python directamente desde R.

Finalmente, se establece el entorno de Python para `rgee` mediante `ee_install_set_pyenv()`, y se configuran las variables de entorno para asegurar que `reticulate` y `rgee` usen el entorno de Python correcto. La función `ee_Initialize(drive = T)` inicializa la conexión con Google Earth Engine, permitiendo el acceso y manipulación de datos geoespaciales desde R.

```
rgee_environment_dir <- reticulate::conda_list()
rgee_environment_dir <-
  rgee_environment_dir[rgee_environment_dir$name == "rgee_py", 2]
reticulate::use_python(rgee_environment_dir, required = T)
reticulate::py_config()
library(reticulate) # Conexión con Python

rgee::ee_install_set_pyenv(py_path = rgee_environment_dir, py_env = "rgee_py")
Sys.setenv(RETICULATE_PYTHON = rgee_environment_dir)
```

```
Sys.setenv(EARTHENGINE_PYTHON = rgee_environment_dir)
rgee::ee_Initialize(drive = T)
```

Arreglar la shapefile

Este bloque de código está diseñado para procesar y transformar archivos shapefile (archivos de formato geoespacial) de México para asegurarse de que estén en el formato correcto y preparados para su análisis. Incluye la lectura de archivos, la transformación del sistema de referencia de coordenadas (CRS), la combinación de los archivos shapefile y la corrección de la codificación de caracteres.

Primero, se descomenta la configuración para ejecutar en paralelo con dos trabajadores usando `plan(multisession, workers = 2)`, lo que mejora la eficiencia del procesamiento. Luego, se listan los archivos shapefile en el directorio "shapefile/2020/" y se leen en una lista `MEX` utilizando `furrr::future_map()`, que permite el procesamiento en paralelo. Cada archivo se transforma para seleccionar columnas específicas (CVEGEO, NOM_ENT, NOMGEO, y geometry). La función `st_transform()` se utiliza para cambiar el sistema de referencia de coordenadas a WGS84, y `st_cast()` convierte los objetos geométricos a "MULTIPOLYGON".

Después de la transformación y combinación, se unen todos los shapefiles en un solo objeto usando `bind_rows()`. Para asegurar que los nombres geográficos estén en la codificación correcta, se usa `iconv()` para convertir `NOMGEO` a UTF-8 desde ISO-8859-1. El archivo shapefile transformado y combinado se guarda con `st_write()` en la ubicación "shapefile/2020/MEX_2020.shp".

Finalmente, se limpia el entorno con `rm(list = ls())`, y el archivo shapefile resultante se lee nuevamente con `read_sf()`. Se agrega una nueva columna `ent` mediante `mutate()` para extraer los dos primeros caracteres de `CVEGEO`, que representan la entidad federativa.

```
# plan(multisession, workers = 2)
#
# MEX <- list.files("shapefile/2020/", pattern = "shp$",
#             full.names = TRUE) %>%
#   furrr::future_map(~read_sf(.x) %>%
#                             select(CVEGEO, NOM_ENT, NOMGEO,geometry),
#                     .progress = TRUE)
#
# mi_crs <- "+proj=longlat +datum=WGS84"
#
# # Transforma el polígono al nuevo CRS
# MEX_trans <- map(MEX,  ~st_transform(.x, crs = mi_crs))
# MEX_trans <- map(MEX_trans,  ~st_cast(.x, "MULTIPOLYGON"))
```

```
#
# MEX_trans <- bind_rows(MEX_trans)
#
#
# # stringi::stri_enc_detect(MEX_trans$NOMGEO)
#
# MEX_trans$NOMGEO <- iconv(MEX_trans$NOMGEO,
#                           to = "UTF-8", from = "ISO-8859-1")
#
#
# st_write(MEX_trans,
#          "shapefile/2020/MEX_2020.shp",
#          append = TRUE)
#
# rm(list = ls())

MEX <- read_sf("../shapefile/2020/MEX_2020.shp")

MEX %<>% mutate(ent = substr(CVEGEO,1,2))
```

Descargar las imegenes satelitales

Este bloque de código se centra en la descarga y procesamiento de imágenes satelitales utilizando Google Earth Engine (GEE) en R. Se recopilan datos de diferentes fuentes satelitales para diversas características geoespaciales de México, incluyendo luces nocturnas, cobertura urbana y agrícola, accesibilidad a servicios de salud y modificación humana global. Cada conjunto de datos se extrae, procesa y guarda para su uso posterior en análisis.

Luces Nocturnas Se utiliza la colección de imágenes "NOAA/VIIRS/DNB/ANNUAL_V21" para obtener datos de luces nocturnas del año 2020. Las imágenes se filtran por fecha y se selecciona la banda "average". Posteriormente, se extraen los datos de luces nocturnas promedio para cada entidad federativa en México, y se combinan los resultados en un solo dataframe. Finalmente, se guardan los datos en un archivo `.rds`.

```
#https://developers.google.com/earth-engine/datasets/catalog/NOAA_VIIRS_DNB_ANNUAL_V

luces = ee$ImageCollection("NOAA/VIIRS/DNB/ANNUAL_V21") %>%
  ee$ImageCollection$filterDate("2020-01-01", "2021-01-01") %>%
  ee$ImageCollection$map(function(x) x$select("average")) %>%
  ee$ImageCollection$toBands()
ee_print(luces)
```

```
MEX_luces <- map(unique(MEX$ent),
                 ~tryCatch(ee_extract(
                   x = luces,
                   y = MEX[c("ent","CVEGEO")] %>%
                     filter(ent == .x),
                   ee$Reducer$mean(),
                   sf = FALSE
                 ) ,
                 error = function(e)data.frame(ent = .x))
)


MEX_luces %<>% bind_rows()

MEX_luces %>% filter(is.na(X20200101_average)) %>% select(CVEGEO)


saveRDS(MEX_luces, "../output/2020/Satelital/MEX_luces.rds")
```

Urbanismo y Cultivos Se extraen datos de cobertura urbana y de cultivos de la colección "COPERNICUS/Landcover/100m/Proba-V-C3/Global" para el año 2019. Se seleccionan las bandas "urban-coverfraction" y "crops-coverfraction", y se calculan los promedios de estas coberturas para cada entidad federativa. Los resultados se combinan y se guardan en un archivo `.rds`.

```
tiposuelo = ee$ImageCollection("COPERNICUS/Landcover/100m/Proba-V-C3/Global") %>%
  ee$ImageCollection$filterDate("2019-01-01", "2019-12-31") %>%
  ee$ImageCollection$map(function(x)
    x$select("urban-coverfraction", "crops-coverfraction")) %>%
  ee$ImageCollection$toBands()
ee_print(tiposuelo)

MEX_urbano_cultivo <- map(unique(MEX$CVEGEO),
                          ~tryCatch(ee_extract(
                            x = tiposuelo,
                            y = MEX[c("ent","CVEGEO")] %>%
                              filter(CVEGEO == .x),
                            ee$Reducer$mean(),
                            sf = FALSE
                          ) ,
                          error = function(e)data.frame(CVEGEO = .x))
)
```

```
MEX_urbano_cultivo %<>% bind_rows()

MEX_urbano_cultivo %>% filter(is.na(X2019_crops.coverfraction)) %>% select(CVEGEO)
MEX_urbano_cultivo %>% filter(is.na(X2019_urban.coverfraction)) %>% select(CVEGEO)

saveRDS(MEX_urbano_cultivo, "../output/2020/Satelital/MEX_urbano_cultivo.rds")
```

Distancia a Hospitales Se utilizan datos de la imagen "Oxford/MAP/accessibility_to_healthcare_2 para calcular la accesibilidad a servicios de salud. Se extraen los datos de accesibilidad para cada entidad federativa y se combinan en un solo dataframe. Los resultados se guardan en un archivo `.rds`.

```
dist_salud = ee$Image('Oxford/MAP/accessibility_to_healthcare_2019')
ee_print(dist_salud)


MEX_dist_salud <- map(unique(MEX$CVEGEO),
                          ~tryCatch(ee_extract(
                            x = dist_salud,
                            y = MEX[c("ent","CVEGEO")] %>%
                              filter(CVEGEO == .x),
                            ee$Reducer$mean(),
                            sf = FALSE
                          ) ,
                          error = function(e)data.frame(CVEGEO = .x))
)


MEX_dist_salud %<>% bind_rows()

MEX_dist_salud %>% filter(is.na(accessibility )) %>% select(CVEGEO)
MEX_dist_salud %>% filter(is.na(accessibility_walking_only)) %>% select(CVEGEO)

saveRDS(MEX_dist_salud, "../output/2020/Satelital/MEX_dist_salud.rds")
```

Modificación Humana Global Se recopilan datos de la colección "CSP/HM/GlobalHumanModificat para medir la modificación humana global en el año 2016. Se extraen los datos para cada entidad federativa y se calculan los promedios, combinándolos en un dataframe. Finalmente, se guardan los resultados en un archivo `.rds`.

```
CSP_gHM = ee$ImageCollection('CSP/HM/GlobalHumanModification')
ee_print(CSP_gHM)

MEX_GHM <- map(unique(MEX$CVEGEO),
                       ~tryCatch(ee_extract(
                         x = CSP_gHM,
                         y = MEX[c("ent","CVEGEO")] %>%
                           filter(CVEGEO == .x),
                         ee$Reducer$mean(),
                         sf = FALSE
                       ) ,
                       error = function(e)data.frame(CVEGEO = .x))
)


MEX_GHM %<>% bind_rows()

MEX_GHM %>% filter(is.na(X2016_gHM )) %>% select(CVEGEO)

saveRDS(MEX_GHM, "../output/2020/Satelital/MEX_GHM.rds")
```

Unir y guardar los resultados de las descargas realizadas

En este bloque de código, se realiza la integración y almacenamiento de los datos satelitales descargados en archivos `.rds`. El objetivo es combinar los datos de diferentes tipos de imágenes satelitales en un único dataframe para facilitar el análisis posterior.

Primero, se generan una lista de archivos `.rds` ubicados en el directorio `"../output/2020/Satelital/"`, que contienen los datos procesados de las imágenes satelitales. Utilizando `map()` y `readRDS()`, se leen todos los archivos y se combinan en un solo dataframe mediante la función `reduce()` con `full_join`, que une los datos basándose en todas las claves presentes.

Luego, se estandarizan las columnas numéricas en el dataframe usando `scale()` para normalizar los valores y se renombran las columnas para asegurar que los nombres sean descriptivos y consistentes. Los nuevos nombres de columna incluyen información sobre modificación humana, acceso a servicios de salud, cobertura urbana y de cultivos, y luces nocturnas.

Se identifican y visualizan las filas que contienen valores faltantes (`NA`) en cualquier columna para realizar una revisión o limpieza adicional. Finalmente, se guarda el dataframe combinado en un archivo `.rds` con el nombre `"statelevel_predictors_satelite.rds"` en el directorio `"../input/2020/predictores/"`, lo que facilita el acceso y análisis de los datos integrados en futuras etapas del proyecto.

```
statelevel_predictors_df <-
    list.files("../output/2020/Satelital/", full.names = TRUE) %>%
  map( ~ readRDS(file = .x)) %>%
  reduce(., full_join) %>%
  mutate_if(is.numeric, function(x)
    as.numeric(scale(x))) %>%
  rename(
    cve_mun = CVEGEO,
    "modifica_humana" = X2016_gHM,
    "acceso_hosp" = accessibility,
    "acceso_hosp_caminando" = accessibility_walking_only,
    cubrimiento_urbano = X2019_urban.coverfraction,
    cubrimiento_cultivo = X2019_crops.coverfraction,
    luces_nocturnas = X20200101_average
  )

statelevel_predictors_df[apply(statelevel_predictors_df, 1, function(x)
  any(is.na(x))), ] %>% view()

saveRDS(
  statelevel_predictors_df,
  "../input/2020/predictores/statelevel_predictors_satelite.rds"
)
```

02_Union_predictores.R

Para la ejecución del presente archivo, debe abrir el archivo **02_Union_predictores.R** disponible en la ruta *Rcodes/2020/02_Union_predictores.R*.

Este script en R está orientado a la integración y limpieza de bases de datos a nivel municipal para el año 2020, combinando datos satelitales y de contexto. Primero, el entorno de trabajo se limpia y se cargan las bibliotecas necesarias para la manipulación y análisis de datos, incluyendo tidyverse, data.table, y openxlsx.

En la primera parte, se carga una base de datos satelital en formato RDS, se renombra una columna para alinear los nombres y se eliminan columnas no necesarias. Luego, se obtienen los códigos municipales únicos de esta base de datos. A continuación, se carga una base de datos adicional en formato Stata (.dta), se convierten algunas variables a factores y se escalan las variables numéricas. Se realiza un análisis para identificar códigos municipales presentes en la base de datos satelital pero no en la de contexto, y se comprueba si hay columnas sin valores faltantes en la base de datos de contexto. Finalmente, se realiza un inner_join para combinar ambas bases de datos utilizando solo las columnas sin valores faltantes y se guarda el resultado en un archivo RDS para futuros análisis.

Configuración Inicial y Carga de Librerías

Este bloque de código está dedicado a la limpieza del entorno de trabajo y a la carga de las librerías necesarias para el análisis de datos en R.

Primero, `rm(list = ls())` se utiliza para eliminar todos los objetos en el entorno de trabajo de R, asegurando que no haya datos o variables residuales que puedan interferir con el análisis actual.

A continuación, se cargan varias librerías esenciales:

- **`tidyverse`**: Un conjunto de paquetes que incluye `ggplot2`, `dplyr`, `tidyr`, entre otros, para la manipulación y visualización de datos.
- **`data.table`**: Ofrece una manera rápida y eficiente de manejar y manipular datos en grandes conjuntos.
- **`openxlsx`**: Utilizado para leer y escribir archivos Excel.

- **magrittr**: Proporciona operadores para la manipulación fluida de datos, como %>%.
- **DataExplorer**: Facilita la exploración y visualización rápida de datos.
- **haven**: Permite la importación y exportación de datos en formatos utilizados por otros paquetes estadísticos, como Stata y SPSS.
- **purrr**: Ofrece herramientas para la programación funcional, como la aplicación de funciones a listas o vectores.
- **labelled**: Maneja y procesa datos con etiquetas, comúnmente utilizados en encuestas.

Finalmente, `cat("\f")` limpia la consola para asegurar que el entorno esté ordenado antes de comenzar el análisis.

```
### Cleaning R environment ###
rm(list = ls())

#################
### Libraries ###
#################
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(DataExplorer)
library(haven)
library(purrr)
library(labelled)
cat("\f")
```

Lectura de bases de contexto 2020

Este bloque de código se encarga de cargar y procesar datos adicionales relacionados con el contexto municipal para el año 2020, que serán utilizados para enriquecer el análisis de datos.

Carga y Procesamiento de Datos Satelitales:

Se carga una base de datos previamente guardada en formato `.rds` que contiene predictores satelitales a nivel municipal. La columna "CVEGEO" se renombra a "cve_mun" para estandarizar los nombres de las variables, y se elimina la columna "ent" que no es necesaria para el análisis actual. Luego, se extraen los códigos municipales únicos de esta base de datos para su posterior uso.

```
## Covariables
# Contexto_munXX : la base de datos contexto_20XX.dta contiene información de
```

```r
# variables a nivel municipal.

# Cargar la base de datos satelital y renombrar la columna "CVEGEO" a "cve_mun"
satelital <-
  readRDS("../input/2020/predictores/statelevel_predictors_satelite.rds") %>%
  rename(cve_mun  = CVEGEO) %>% mutate(ent = NULL)

# Obtener los códigos municipales únicos de la base de datos satelital
cod_mun <- satelital %>% distinct(cve_mun)
```

Carga y Exploración de Datos Contextuales:

Se carga una base de datos en formato Stata (`contexto_2020.dta`) que contiene variables contextuales a nivel municipal. La función `as_factor()` se usa para asegurar que las variables sean tratadas como factores, y `apply(., 2, n_distinct) %>% sort()` se emplea para obtener y ordenar el número de valores únicos por variable. Esto ayuda a entender la estructura de los datos y la cantidad de información contenida en cada variable.

```r
# Cargar la base de datos "Contexto_mun20" en formato Stata y renombrar la columna "
Contexto_mun <- read_dta("../input/2020/predictores/contexto_2020.dta")
Contexto_mun %>% as_factor() %>%
  apply(., 2, n_distinct) %>% sort()

Contexto_mun$elec_mun19
Contexto_mun$elec_mun20
Contexto_mun$smg1
Contexto_mun$ql_porc_cpa_urb
Contexto_mun$ql_porc_cpa_rur
Contexto_mun$dec_geologicas
Contexto_mun$dec_hidrometeo
```

Procesamiento y Unión de Datos Contextuales

Este bloque de código se centra en la preparación final y la combinación de datos contextuales a nivel municipal, asegurando que se utilicen solo las variables relevantes y sin valores faltantes.

1. **Estandarización y Escalado de Variables Contextuales:** La base de datos `Contexto_mun` se transforma para asegurar que las variables categóricas sean tratadas como factores, mientras que las variables numéricas se escalan. Se utilizan las funciones `mutate_at` y `mutate_if` para realizar estas transformaciones en las variables especificadas y en todas las variables numéricas, respectivamente.

2. **Identificación de Códigos Municipales Inexistentes:** Se realiza una

operación de `anti_join` entre los códigos municipales de la base de datos satelital (`cod_mun`) y la base de datos de contexto (`Contexto_mun`). Esta operación identifica los códigos municipales que están presentes en la base de datos satelital pero que no se encuentran en la base de datos de contexto.

3. **Detección de Columnas sin Valores Faltantes:** Se analiza cada columna de `Contexto_mun` para verificar si tiene valores faltantes utilizando `apply` y `any(is.na(x))`. El resultado es una lista que indica si cada columna contiene valores faltantes o no. La función `table(paso)` se utiliza para mostrar la frecuencia de columnas sin valores faltantes.

4. **Unión de Bases de Datos:** Se realiza una `inner_join` entre las bases de datos `satelital` y `Contexto_mun`, pero solo utilizando las columnas sin valores faltantes, como se determinó en el paso anterior. Esto garantiza que la base de datos resultante solo incluya variables completas y coherentes.

5. **Guardado del Conjunto de Datos Combinado:** Finalmente, la base de datos combinada `statelevel_predictors` se guarda en un archivo `.rds` para su uso futuro.

```
Contexto_mun %<>% as_factor() %>%
  mutate_at(
    .vars = c(
      "elec_mun19",
      "elec_mun20",
      "smg1",
      "ql_porc_cpa_urb",
      "ql_porc_cpa_rur",
      "dec_geologicas",
      "dec_hidrometeo"
    ),
    as.factor
  ) %>%
  mutate_if(is.numeric, function(x)
    as.numeric(scale(x)))

# Realizar una anti_join para identificar códigos municipales presentes en "cod_mun"
anti_join(cod_mun, Contexto_mun)

# Realizar un análisis para determinar si alguna columna de "Contexto_mun" no tiene
paso <- apply(Contexto_mun, 2, function(x) !any(is.na(x)))

# Mostrar la frecuencia de columnas sin valores faltantes
table(paso)
```

```r
# Realizar una inner_join entre las bases de datos "satelital" y "Contexto_mun" util
statelevel_predictors <- inner_join(satelital, Contexto_mun[, paso])

# Obtener las dimensiones de la base de datos resultante
dim(statelevel_predictors)

## Guardar
saveRDS(statelevel_predictors,file = "../input/2020/predictores/statelevel_predictors_
```

03_Estandarizar_muestra cuestionario_ampliado.R

Para la ejecución del presente archivo, debe abrir el archivo **02_Union_predictores.R** disponible en la ruta *Rcodes/2020/02_Union_predictores.R*. Este script está diseñado para llevar a cabo un análisis detallado de datos mediante un proceso estructurado en varias fases.

En la etapa inicial, el código limpia el entorno de R eliminando todos los objetos existentes para comenzar con un espacio de trabajo limpio. A continuación, carga las bibliotecas necesarias, como `tidyverse` y `data.table`, que son fundamentales para la manipulación y análisis de datos, así como para la importación y exportación de archivos. Luego, el código lee las bases de datos correspondientes al censo de personas 2020 y a la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). Estas bases se utilizan para identificar indicadores clave relacionados con carencia en salud, calidad de vivienda, servicios básicos y rezago educativo.

Seguidamente, el código valida y armoniza las variables entre ambas bases de datos para asegurar la consistencia en códigos y etiquetas. Esto incluye la comparación de variables como códigos de entidad y municipio, áreas urbanas y rurales, sexo, edad, nivel educativo, discapacidad y lengua indígena. Posteriormente, estandariza las variables del censo para alinearlas con las de la encuesta y filtra los datos para eliminar cualquier inconsistencia. Finalmente, resume el conjunto de datos estandarizado por grupo y guarda los datos procesados en dos versiones: una con los datos estandarizados y otra con los datos resumidos.

Lectura y Preparación de Datos

En este bloque de código, se inicia el entorno de trabajo en R, cargando las bibliotecas necesarias y preparando los datos para el análisis.

Primero, se limpia el entorno de R con `rm(list = ls())` para asegurar que no haya variables o datos residuales que puedan interferir con el análisis. Luego, se cargan diversas bibliotecas que proporcionan herramientas para la manipulación de datos, la lectura de archivos, la exploración de datos y el análisis estadístico.

Entre estas bibliotecas se encuentran `tidyverse`, `data.table`, `openxlsx`, `magrittr`, `DataExplorer`, `haven`, `purrr`, `labelled` y `sampling`.

A continuación, se lee el archivo `muestra_cuestionario_ampliado.rds` usando `readRDS()`, que contiene un dataframe con los datos ampliados del censo. También se lee el archivo `enigh.rds`, que contiene datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH) de 2020. Finalmente, se muestra la dimensión del dataframe `muestra_cuestionario_ampliado` con `dim()` para obtener una visión general de su tamaño y estructura.

```
### Cleaning R environment ###
rm(list = ls())

#################
### Libraries ###
#################
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(DataExplorer)
library(haven)
library(purrr)
library(labelled)
library(sampling)
cat("\f")

#############################################################################
# Lectura de bases censo persona 2020
#############################################################################
muestra_cuestionario_ampliado <-
  readRDS(
    "../output/2020/muestra_cuestionario_ampliado.rds"
  )
enigh <- readRDS("../output/2020/enigh.rds")
muestra_cuestionario_ampliado %>% dim()
```

Identificando indicadores

En este bloque de código, se realiza una revisión inicial de los indicadores clave presentes en el dataframe `muestra_cuestionario_ampliado` para comprender su contenido y estructura.

Se comienza revisando el indicador de carencia por acceso a los servicios de salud

(ic_asalud) mediante la función head(), que muestra las primeras entradas de este indicador en el dataframe. Esto ayuda a obtener una visión general de los datos disponibles para este indicador específico.

Luego, se exploran las etiquetas asociadas a los atributos de los indicadores de carencia por calidad y espacios de la vivienda (ic_cv) y carencia por servicios básicos en la vivienda (ic_sbv) utilizando la función attributes() y $label. Esto proporciona información sobre las descripciones o categorías asociadas con estos indicadores, facilitando la comprensión de los datos.

Finalmente, se revisa el indicador de carencia por rezago educativo (ic_rezedu) de la misma manera, explorando sus etiquetas para entender las descripciones asociadas con este indicador.

```
# Indicador de carencia por acceso a los servicios de salud (CENSO)
head(muestra_cuestionario_ampliado$ic_asalud)

# Carencia por la calidad y espacios de la vivienda. (CENSO)
attributes(muestra_cuestionario_ampliado$ic_cv)$label

# Carencia por servicios básicos en la vivienda. (CENSO)
attributes(muestra_cuestionario_ampliado$ic_sbv)$label

# Carencia por rezago educativo.
attributes( muestra_cuestionario_ampliado$ic_rezedu)$label
```

Validaciones para la armonización de variables

En este bloque de código, se realizan una serie de validaciones para asegurar la armonización de variables entre los datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH) y el censo ampliado.

Primero, se revisan los códigos de entidad (ent) y municipio (cve_mun) en ambos conjuntos de datos. Se utiliza n_distinct() para contar el número de valores únicos en cada variable, comparando así los códigos de entidad y municipio entre la encuesta y el censo.

A continuación, se analiza la variable rururb, que indica si un área es urbana o rural. Se exploran los atributos de esta variable en la encuesta y se utiliza table() para mostrar la distribución de esta variable en el censo, verificando los valores únicos y su correspondencia.

Luego, se validan las variables relacionadas con el sexo (sexo). Se revisan los atributos de esta variable en la encuesta y se utiliza table() para mostrar la distribución en el censo. Esto asegura que las categorías de sexo sean consistentes entre ambos conjuntos de datos.

Se revisa la variable de edad (`edad`). Se exploran los atributos de esta variable en la encuesta y se muestran las primeras entradas de la variable `edad_cat` en el censo para comparar y validar las categorías de edad.

Para la variable de nivel educativo (`niv_ed`), se comparan los atributos de esta variable en la encuesta y el censo. Se asegura que las categorías de nivel educativo sean consistentes en ambos conjuntos de datos.

Se valida la variable de discapacidad (`discap`). Se revisan los atributos de esta variable en ambos conjuntos de datos para determinar si se necesita realizar alguna conversión o armonización.

Finalmente, se revisa la variable que indica si la persona habla un dialecto o lengua indígena (`hli`). Se comparan los atributos de esta variable en la encuesta y el censo para asegurar que los datos sean coherentes.

```
## codigos de ent y municipio

## Encuesta
n_distinct(enigh$ent) # cod_dam
n_distinct(enigh$cve_mun) # cod_mun

## censo
n_distinct(muestra_cuestionario_ampliado$ent)
n_distinct(muestra_cuestionario_ampliado$cve_mun)

#########################################################################

## area

## Encuesta
attributes(enigh$rururb)

## censo
table(muestra_cuestionario_ampliado$rururb, useNA = "a")
# 0 Urbano
# 1 Rural


#########################################################################

## Sexo

## Encuesta
attributes(enigh$sexo)
```

```
# 2 Mujer
# 1 Hombre

## censo
table(muestra_cuestionario_ampliado$sexo, useNA = "a")
# 0 Mujer
# 1 Hombre

#########################################################################
## edad

## Encuesta
attributes(enigh$edad)

## censo
head(muestra_cuestionario_ampliado$edad_cat)

#########################################################################
## nivel_edu #

## Encuesta
attributes(enigh$niv_ed)

# 0 [Con primaria incompleta o menos]
# 1 [Primaria completa o secundaria incompleta]
# 2 [Secundaria completa o media superior incompleta]
# 3 [Media superior completa o mayor nivel educativo]
# NA  Niños de 0 a 3 años

## censo
attributes(muestra_cuestionario_ampliado$niv_ed)

# 0 [Con primaria incompleta o menos]
# 1 [Primaria completa o secundaria incompleta]
# 2 [Secundaria completa o media superior incompleta]
# 3 [Media superior completa o mayor nivel educativo]
# NA  Niños de 0 a 3 años


#########################################################################
## Discapacidad #
```

```
## Encuesta
attributes(enigh$discap)

## censo
attributes(muestra_cuestionario_ampliado$discap)


#########################################################################
## Habla dialecto o lengua indígena

## Encuesta
attributes(enigh$hli)


## censo
attributes(muestra_cuestionario_ampliado$hli)
```

Estandarizando el censo

Este bloque de código se centra en la estandarización y limpieza del conjunto de datos del censo para prepararlo para el análisis.

Primero, se crea un nuevo dataframe `censo_sta` a partir del dataframe `muestra_cuestionario_ampliad` En este proceso, se realiza una transformación y estandarización de las variables. Se convierten las variables categóricas a tipo `character` o `factor`, y se manejan los valores faltantes. Específicamente:

- La variable `area` se convierte a tipo `character`.
- La variable `sexo` se estandariza para que los valores sean "1" para hombre y "2" para mujer.
- Las variables de edad, nivel educativo y discapacidad se transforman a tipo `character`, con los valores de discapacidad y lengua indígena establecidos en "0" si son `NA`.
- Las variables `ic_asalud`, `ic_cv`, `ic_sbv`, e `ic_rezedu` se convierten a tipo `factor` utilizando `haven::as_factor()`.

Después, se filtran ciertos municipios excluidos mediante `filter()`, eliminando aquellos con códigos específicos (04012, 07125, 29048).

A continuación, se crea un segundo dataframe `censo_sta2` a partir de `censo_sta`, filtrando los registros con valores faltantes en las variables de interés (`nivel_edu`, `edad`, `ic_sbv`, `ic_cv`, `ic_rezedu`, `ic_asalud`).

Finalmente, se calcula la proporción del total del `factor` en `censo_sta2` respecto a `censo_sta`, lo que permite verificar la representatividad del subconjunto filtrado. El

dataframe estandarizado `censo_sta2` se guarda en un archivo `.rds` para su uso en análisis futuros.

```
censo_sta <- muestra_cuestionario_ampliado %>%
  transmute(
    ent,
    cve_mun,
    upm,
    estrato,
    area = as.character(rururb),
    sexo = if_else(sexo == 1, "1", "2"),
    edad = as.character(edad_cat),
    nivel_edu = haven::as_factor(niv_ed, levels  = "values"),
    discapacidad = as.character(discap),
    discapacidad = if_else(is.na(discapacidad), "0", discapacidad),
    hlengua = as.character(hli),
    hlengua = if_else(is.na(hlengua), "0", hlengua),

    ic_asalud= haven::as_factor(ic_asalud, levels  = "values"),
    ic_cv = haven::as_factor(ic_cv, levels  = "values"),
    ic_sbv = haven::as_factor(ic_sbv, levels  = "values"),
    ic_rezedu = haven::as_factor(ic_rezedu, levels  = "values"),
    factor
  ) %>%
  filter( !cve_mun %in% c("04012", "07125", "29048"))




censo_sta2 <- censo_sta %>%
  filter(
    !is.na(nivel_edu),
    !is.na(edad),
    !is.na(ic_sbv),
    !is.na(ic_cv),
    !is.na(ic_rezedu),
    !is.na(ic_asalud),
  )

sum(censo_sta2$factor) / sum(censo_sta$factor)

#Se guarda el conjunto de datos estandarizado:
```

```
saveRDS(censo_sta2,"../output/2020/encuesta_ampliada.rds")
```

Resumen y Guardado de Datos Agrupados

Este bloque de código realiza un resumen de los datos estandarizados del censo y guarda el conjunto de datos resultante.

Primero, el dataframe `censo_sta2` se agrupa utilizando `group_by()` en base a múltiples variables: `ent`, `cve_mun`, `area`, `hlengua`, `sexo`, `edad`, `nivel_edu`, `discapacidad`, `ic_asalud`, `ic_cv`, `ic_sbv`, e `ic_rezedu`. Para cada grupo definido por estas variables, se calcula la suma de la variable `factor` utilizando `summarise()`. El resultado es un dataframe que contiene el total de observaciones (`n`) para cada combinación única de los grupos especificados.

Finalmente, el dataframe resumido se guarda en un archivo `.rds` en la ruta `"../input/2020/muestra_ampliada/muestra_cuestionario_ampliado.rds"`. Este archivo puede ser utilizado para análisis posteriores, proporcionando una versión compacta y organizada del conjunto de datos.

```
#Se resumen los datos por grupo:

censo_sta2 %<>% group_by(
  ent,
  cve_mun,
  area,
  hlengua,
  sexo,
  edad,
  nivel_edu,
  discapacidad,
  ic_asalud,
  ic_cv,
  ic_sbv,
  ic_rezedu
) %>%
  summarise(n = sum(factor), .groups = "drop")

#Se guarda el conjunto de datos resumido:

saveRDS(censo_sta2, "../input/2020/muestra_ampliada/muestra_cuestionario_ampliado.rds"
```

04_Armonizar_encuesta.R

Para la ejecución del presente archivo, debe abrir el archivo **04_Armonizar_encuesta.R** disponible en la ruta *Rcodes/2020/04_Armonizar_encuesta.R*. Este script está diseñado para llevar a cabo un análisis exhaustivo de datos a través de un proceso metódico en varias fases.

En la primera fase, el código elimina todos los objetos del entorno de R para asegurar que el análisis se inicie con un entorno limpio y sin datos residuales. A continuación, se cargan las bibliotecas necesarias para el análisis, que proporcionan herramientas esenciales para la manipulación de datos y la importación y exportación de archivos. Después, se leen las bases de datos del censo de personas 2020 y de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). Estas bases se utilizan para identificar y verificar indicadores relacionados con carencias sociales.

Posteriormente, se definen y validan las variables clave de la encuesta, como códigos de entidad, municipio, área, sexo, edad, nivel educativo, discapacidad y lengua indígena. Luego, se estandarizan las variables de la encuesta para alinear los datos con los del censo, se filtran los datos para eliminar inconsistencias y se guarda el conjunto de datos estandarizado. Además, se actualiza la tabla censal utilizando la técnica de calibración IPFP (Iterative Proportional Fitting Procedure) para asegurar que las distribuciones marginales de las variables estandarizadas coincidan con las del censo. Finalmente, se generan histogramas y gráficos de dispersión para visualizar los resultados de la calibración y se guarda el conjunto de datos ampliado.

Limpieza del Entorno y Carga de Bibliotecas

El código presentado realiza dos tareas principales: limpiar el entorno de trabajo en R y cargar las bibliotecas necesarias para el análisis de datos.

La primera parte del código se encarga de eliminar todos los objetos en el entorno de trabajo de R y ejecutar el recolector de basura para liberar memoria. Esta operación es esencial antes de comenzar un nuevo análisis para asegurarse de que el entorno esté limpio y libre de datos o objetos antiguos que puedan interferir con el nuevo análisis.

```
### Cleaning R environment ###
rm(list = ls())
```

```
gc()
```

La segunda parte del código se enfoca en cargar una serie de bibliotecas cruciales para el análisis de datos. Estas bibliotecas proporcionan diversas funcionalidades que facilitan la manipulación, visualización y análisis de datos.

- `tidyverse`: Un conjunto de paquetes para la manipulación y visualización de datos.
- `data.table`: Paquete para la manipulación eficiente de grandes conjuntos de datos.
- `openxlsx`: Herramienta para leer y escribir archivos de Excel.
- `magrittr`: Proporciona el operador de tubería (`%>%`) para encadenar operaciones.
- `DataExplorer`: Paquete para la exploración rápida de datos.
- `haven`: Permite leer y escribir datos en formatos de Stata, SPSS y SAS.
- `purrr`: Facilita la programación funcional con listas y vectores.
- `labelled`: Trabaja con datos etiquetados, especialmente útil para datos de encuestas.
- `sampling`: Proporciona métodos para la toma de muestras estadísticas.

```
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(DataExplorer)
library(haven)
library(purrr)
library(labelled)
library(sampling)
cat("\f")
```

La función `cat("\f")` se utiliza para limpiar la pantalla de la consola en R, proporcionando una interfaz más ordenada para el usuario.

Lectura de Datos

En esta sección, se realiza la lectura de dos conjuntos de datos cruciales para el análisis y se lleva a cabo una verificación para asegurar la unicidad de los identificadores.

Primero, se cargan las bases de datos `muestra_cuestionario_ampliado.rds` y `enigh.rds` desde los archivos correspondientes. La base `muestra_cuestionario_ampliado.rds` contiene información ampliada del cuestionario, mientras que `enigh.rds` incluye datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares.

```
encuesta_ampliada <- readRDS("../input/2020/muestra_ampliada/muestra_cuestionario_ampl
enigh <- readRDS("output//2020/enigh.rds")
```

A continuación, se verifica la unicidad de los identificadores de entidad y municipio en la base de datos `enigh`. Esta verificación es importante para asegurar que los datos estén correctamente identificados y evitar errores en análisis posteriores que puedan surgir de duplicados o inconsistencias en los identificadores.

```
n_distinct(enigh$ent) # cod_dam
n_distinct(enigh$cve_mun) # cod_mun
```

La función `n_distinct()` se utiliza para contar el número de identificadores únicos en las columnas `ent` y `cve_mun`. Este paso garantiza que cada entidad y municipio tenga un identificador único en los datos, lo cual es crucial para la integridad y precisión del análisis.

Definición de Variables para la Encuesta

Se definen y transforman las variables relacionadas con el área urbana o rural en la base de datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH). Primero, se revisa la distribución de la variable `rururb` para identificar el número de registros en cada categoría de área urbana o rural. Posteriormente, se convierte `rururb` en una variable de factor con niveles definidos por la variable "values" usando la función `as_factor()` del paquete `haven`, y luego se transforma a formato de carácter para simplificar su uso en análisis posteriores. Finalmente, se verifica que la conversión se haya realizado correctamente, asegurando que los valores de `rururb` y `area` sean consistentes.

```
# codigo municipio

# area urabo o rural
enigh %>% group_by(rururb) %>% summarise(n = sum(factor))

enigh %<>% mutate(
  area = haven::as_factor(rururb, levels  = "values"),
  area = as.character(area)
)

enigh %>%   distinct(rururb,area)
```

Transformación de Variables Demográficas y Socioeconómicas

Se realizan transformaciones en las variables demográficas y socioeconómicas de la base de datos `enigh` para prepararlas para el análisis. Primero, se revisa la distribución de la variable de `sexo`, que tiene dos categorías: "2" para mujeres y "1" para hombres. Luego, se examina la variable de `edad`, transformando las edades en grupos de edad específicos: "1" para menores de 14 años, "2" para 15 a 29 años, "3" para 30 a 44 años, "4" para 45

a 64 años, y "5" para 65 años o más. La transformación se realiza utilizando la función `case_when()`.

En cuanto al `nivel educativo`, se revisa la distribución actual y se transforma en un factor con niveles específicos definidos por la variable "values", asegurando que los valores sean consistentes y descriptivos. Posteriormente, se realiza la misma transformación para la variable `discapacidad`, convirtiéndola en un factor con niveles apropiados y luego a formato de carácter para su uso en el análisis.

Finalmente, para la variable `hli`, que indica si se habla algún dialecto o lengua indígena, se transforma en un factor con niveles definidos, asegurando la consistencia en la representación de esta variable. Las transformaciones aseguran que todas las variables sean adecuadas para el análisis, con valores claros y uniformes.

```
# Sexo
enigh %>% group_by(sexo) %>% summarise(n = sum(factor))
# 2 Mujer
# 1 Hombre

# Edad
enigh %>% group_by(edad) %>% summarise(n = sum(factor))
encuesta_ampliada %>% distinct(edad)
# 1 Menor de 14 años
# 2 15 a 29 años
# 3 30 a 44 años
# 4 45 a 64 años
# 5 65 años o más

enigh %<>% mutate(g_edad = case_when(edad <= 14 ~ "1", # 0 a 14
                                     edad <= 29 ~ "2", # 15 a 29
                                     edad <= 44 ~ "3", # 30 a 44
                                     edad <= 64 ~ "4", # 45 a 64
                                     edad >  64 ~ "5", # 65 o mas
                                     TRUE ~ NA_character_))
enigh %>% group_by(g_edad) %>% summarise(n = sum(factor))

# nivel educativo

enigh %>% group_by(niv_ed) %>%
  summarise(n = sum(factor), .groups = "drop") %>%
  mutate(prop = n / sum(n))

enigh %>%
  mutate(nivel_edu = haven::as_factor(niv_ed, levels  = "values")) %>%
```

```r
  group_by(nivel_edu, g_edad) %>% summarise(n = sum(factor)) %>%
  data.frame()

enigh %<>% mutate(
  nivel_edu = haven::as_factor(niv_ed, levels  = "values"),
  nivel_edu = as.character(nivel_edu)
)

enigh %>%   distinct(niv_ed,nivel_edu)

# Discapacidad

enigh %>% group_by(discap) %>% summarise(n = sum(factor))

enigh %<>% mutate(
  discapacidad = haven::as_factor(discap, levels  = "values"),
  discapacidad = as.character(discapacidad)
)

enigh %>%   distinct(discap,discapacidad)

# Habla dialecto o lengua indígena
attributes(enigh$hli)

enigh %>% group_by(hli) %>% summarise(n = sum(factor))

enigh %<>% mutate(
  hlengua = haven::as_factor(hli, levels  = "values"),
  hlengua = as.character(hlengua)
)

enigh %>%   distinct(hlengua,hli)
```

Análisis de Variables de Carencia

Para el análisis de variables de carencia, se realiza lo siguiente:

1. **Distribución Logarítmica del Ingreso Per Cápita**: Se visualiza la distribución del ingreso per cápita (`ictpc`) mediante un histograma en escala logarítmica. Esta transformación es útil para observar la distribución de datos con alta variabilidad y para identificar patrones o anomalías en la distribución del ingreso.

2. **Indicadores de Carencia**:
    - **Acceso a Servicios de Salud**: Se revisa el atributo de la variable `ic_segsoc`, que representa el indicador de carencia por acceso a servicios de salud.
    - **Acceso a Alimentación Nutritiva y de Calidad**: Se revisa el atributo de la variable `ic_ali_nc`, que mide la carencia en el acceso a alimentación nutritiva y de calidad.
3. **Distribución de Carencia por Acceso a Servicios de Salud**: Se agrupan los datos por la variable `ic_segsoc` (carencia por acceso a servicios de salud) y se calcula la suma del factor en cada grupo para analizar la distribución de esta carencia.

```r
hist(log(enigh$ictpc))

# Indicador de carencia por acceso a servicios de salud
attributes(enigh$ic_segsoc)

# Indicador de carencia por acceso a la alimentación nutritiva y de calidad
attributes(enigh$ic_ali_nc)

enigh %>% group_by(ic_segsoc) %>% summarise(n = sum(factor))
```

Preparación y Guardado del Conjunto de Datos

El código realiza un proceso exhaustivo para preparar y guardar un conjunto de datos final denominado `encuesta_sta`. Este conjunto de datos se crea a partir de la base de datos `enigh`, aplicando transformaciones y ajustes necesarios para garantizar que los datos estén en el formato adecuado para el análisis posterior.

En primer lugar, el código transforma varias variables clave. La variable `ent`, que representa la entidad, se ajusta para asegurar que siempre tenga dos caracteres, añadiendo ceros a la izquierda si es necesario. Las variables relacionadas con la edad, el nivel educativo, la discapacidad y la lengua indígena se procesan para reemplazar los valores faltantes con valores predeterminados específicos, como “99” para las edades no especificadas y “0” para la discapacidad y la lengua indígena no declarada. Además, las variables relacionadas con las carencias (`ictpc`, `ic_segsoc`, `ic_ali_nc`, `ic_rezedu`, `ic_asalud`, `ic_sbv`, `ic_cv`) se convierten a tipo numérico para facilitar el análisis cuantitativo.

El conjunto de datos también incluye variables de diseño, como `estrato`, `upm`, y `fep`, que son esenciales para el análisis de diseño y ponderación. Estas variables ayudan a asegurar que los resultados del análisis reflejen correctamente la estructura de la muestra y las características del estudio.

Posteriormente, se realiza un análisis de frecuencias para cada una de las variables clave. Esto incluye calcular la frecuencia absoluta y la proporción relativa para variables como el código de municipio, área (urbano o rural), sexo, edad, nivel educativo, discapacidad, y lengua indígena, así como para los indicadores de carencia. Este análisis proporciona una visión general de la distribución de los datos y permite verificar la representatividad y la calidad del conjunto de datos.

Finalmente, el conjunto de datos transformado y analizado se guarda en un archivo con formato `.rds`. Este archivo es ahora accesible para su uso en análisis posteriores, garantizando que los datos están bien preparados y que todas las transformaciones necesarias se han aplicado correctamente.

```
##############################################################################
encuesta_sta <- enigh %>%
  transmute(
    ent = str_pad(
      string = ent,
      width = 2,
      pad = "0"
    ),
    cve_mun,
    area,
    sexo,
    edad = ifelse(is.na(g_edad), "99", g_edad),
    nivel_edu = ifelse(is.na(nivel_edu), "99", nivel_edu),
    discapacidad = ifelse(is.na(discapacidad), "0", discapacidad),
    hlengua = ifelse(is.na(hlengua), "0", hlengua),

    ## Variable de estudio
    ictpc = as.numeric(ictpc),
    ic_segsoc = as.numeric(ic_segsoc),
    ic_ali_nc = as.numeric(ic_ali_nc),
    ic_rezedu = as.numeric(ic_segsoc),
    ic_asalud = as.numeric(ic_segsoc),
    ic_sbv = as.numeric(ic_sbv_hog),
    ic_cv = as.numeric(ic_cv_hog),

    ## Variables diseño
    estrato = est_dis,
    upm = upm,
    fep = factor
  )

map(c(
```

```
  "ent",
  "cve_mun",
  "area",
  "sexo",
  "edad",
  "nivel_edu",
  "hlengua",
  "discapacidad",
  "ic_segsoc",
  "ic_ali_nc"
),
function(x) {
  encuesta_sta %>% group_by_at(x ) %>%
    summarise(Nd = sum(fep)) %>%
    mutate(N = sum(Nd),
           prop = Nd / N)
})

# Se guarda el conjunto de datos
saveRDS(encuesta_sta, file = "../input/2020/enigh/encuesta_sta.rds")
```

Actualización de la Tabla Censal

El proceso de actualización de la tabla censal mediante la calibración de pesos es crucial para asegurar que los datos reflejen de manera precisa la estructura de la población y las características de la muestra. A continuación, se describe el proceso detallado llevado a cabo en el código:

En primer lugar, se identifican y seleccionan las covariables que están presentes tanto en la base de datos `encuesta_sta` como en la base de datos `encuesta_ampliada`. Esto se realiza mediante la comparación de los nombres de las covariables en ambas bases de datos y asegurando que solo se consideren aquellas con niveles completos en ambas tablas. Esta selección es fundamental para evitar problemas derivados de niveles incompletos en las covariables, lo que podría afectar la precisión de la calibración.

Luego, se utiliza una función auxiliar (`auxSuma`) para calcular las sumas ponderadas de las covariables. Esta función convierte las variables categóricas en variables dummy, multiplica cada variable dummy por su peso correspondiente, y luego calcula la suma total para cada categoría. Este cálculo se realiza tanto para la muestra (`encuesta_sta`) como para el censo (`encuesta_ampliada`), permitiendo una comparación directa entre los valores estimados en la muestra y los valores esperados en la población censal.

La comparación de las sumas ponderadas permite calibrar los pesos de la muestra para que se ajusten a las características observadas en el censo. Los valores obtenidos para

cada categoría en la muestra (N.g) se comparan con los valores correspondientes en el censo (N_censo.g). Esta calibración asegura que la muestra sea representativa de la población en términos de las covariables seleccionadas, mejorando la precisión y la fiabilidad de los análisis posteriores basados en esta tabla censal actualizada.

Finalmente, se guarda el conjunto de datos calibrado, lo cual es esencial para que los análisis posteriores se realicen utilizando datos ajustados y representativos. Este proceso de calibración es un paso crucial en el manejo de datos de encuestas y censos, ya que permite corregir desviaciones y mejorar la precisión de las estimaciones derivadas de la muestra.

```
# Actualización de tabla censal- IPFP ------------------------------------
names_cov <- c("ent", "cve_mun", "area", "sexo", "edad", "discapacidad", "hlengua")
names_cov <- names_cov[names_cov %in% names(encuesta_sta)]

num_cat_censo <- apply(encuesta_ampliada[names_cov], MARGIN  = 2, function(x)
  length(unique(x)))

num_cat_sample <- apply(encuesta_sta[names_cov], MARGIN  = 2, function(x)
  length(unique(x)))

names_cov <- names_cov[num_cat_censo == num_cat_sample]

# MatrizCalibrada creada únicamente para los niveles completos
# IMPORTANTE: Excluir las covariables que tengan niveles incompletos

auxSuma <- function(dat, col, ni) {
  dat %>% ungroup() %>% select(all_of(col))  %>%
    fastDummies::dummy_cols(remove_selected_columns = TRUE) %>%
    mutate_all(~.*ni) %>% colSums()
}

N.g <- map(names_cov,
           ~ auxSuma(encuesta_sta, col = .x, ni = encuesta_sta$fep)) %>%
  unlist()

N_censo.g <- map(names_cov,
                 ~ auxSuma(encuesta_ampliada, col = .x, ni = encuesta_ampliada$n)) %>%
  unlist()

names_xk <- intersect(names(N.g), names(N_censo.g))

N.g <- N.g[names_xk]
N_censo.g <- N_censo.g[names_xk]
```

El proceso de calibración y ajuste de pesos en la tabla censal es una etapa crítica para asegurar que los datos reflejen fielmente la estructura de la población. A continuación, se detalla el flujo de trabajo descrito en el código:

1. **Generación de Variables Dummy:** Primero, se crea un conjunto de datos con variables dummy para las covariables seleccionadas (`names_cov`). Esto se realiza usando la función `fastDummies::dummy_cols`, que transforma las variables categóricas en variables dummy, facilitando así la calibración posterior. Se seleccionan únicamente las variables de interés (`names_xk`) que coinciden entre la muestra y el censo.

2. **Calibración de Pesos:** Utilizando la función `calib`, se calibra la muestra (`Xk`) para que los totales ponderados coincidan con los valores esperados en la población censal (`N.g`). La calibración se realiza mediante el método lineal, que ajusta los pesos de manera que las sumas ponderadas en la muestra se alineen con los totales esperados en la población.

3. **Verificación de Calibración:** Se verifica la calidad de la calibración mediante la función `checkcalibration`, que compara los totales ajustados con los totales esperados. Esta verificación es crucial para asegurarse de que el proceso de calibración ha sido efectivo.

4. **Análisis de Pesos Calibrados:** Se realiza un análisis de la distribución de los pesos calibrados (`gk`) mediante un histograma y un resumen estadístico. Esto permite evaluar la variabilidad y la distribución de los pesos ajustados.

5. **Ajuste de Conteos Censales:** Los pesos calibrados se aplican a los conteos censales (`n1`) para obtener los conteos ajustados. Se compara gráficamente el conteo censal original con el ajustado mediante un gráfico de dispersión, donde la línea roja discontinua representa una relación 1:1. Esto ayuda a visualizar cómo los conteos censales han cambiado después de la calibración.

6. **Actualización de Datos:** Finalmente, se actualizan los conteos en la base de datos `encuesta_ampliada` con los valores ajustados y se guarda el conjunto de datos calibrado en un archivo `.rds`. Este archivo contiene la tabla censal actualizada, lista para ser utilizada en análisis posteriores.

Este proceso asegura que los datos censales reflejen de manera precisa la estructura poblacional y corrige cualquier desviación en los conteos, mejorando la calidad y la representatividad de la información.

```
Xk <- encuesta_ampliada %>% ungroup() %>% select(all_of(names_cov)) %>%
  fastDummies::dummy_cols(remove_selected_columns = TRUE) %

>%
  select(all_of(names_xk))
```

```r
gk <- calib(Xs = Xk,
            d = encuesta_ampliada$n,
            total = N.g,
            method = "linear") # linear primera opcion

checkcalibration(Xs = Xk,
                 d = encuesta_ampliada$n,
                 total = N.g,
                 g = gk)

hist(gk)
summary(gk)
ggplot(data.frame(x = gk), aes(x = x)) +
  geom_histogram(binwidth = diff(range(gk)) / 20,
                 color = "black", alpha = 0.7) +
  labs(title = "", x = "", y = "") +
  theme_minimal() +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank())

n1 <- encuesta_ampliada$n * gk

ggplot(data = data.frame(x = encuesta_ampliada$n, y = n1), aes(x = x, y = y)) +
  geom_point() +  # Agregar puntos
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(title = "", x = "Conteo censales antes", y = "Conteos censales después") +
  theme_minimal(20)

encuesta_ampliada$n <- encuesta_ampliada$n * gk

# Se guarda el conjunto de datos ampliado:
saveRDS(encuesta_ampliada, "../output/2020/encuesta_ampliada.rds")
```

05_Validacion_encuestas.R

Para ejecutar este archivo, es necesario abrir el archivo **05_Validacion_encuestas.R** ubicado en la ruta *Rcodes/2020/05_Validacion_encuestas.R*. Este script está diseñado para llevar a cabo un análisis integral de datos con varias fases que aseguran una evaluación rigurosa de la información disponible.

En la primera etapa del código, se eliminan todos los objetos del entorno de R para garantizar que el análisis se realice en un entorno limpio y sin interferencias. Luego, se cargan las librerías necesarias para llevar a cabo el análisis y se establece el tema de visualización predeterminado usando `bayesplot::theme_default()`. También se carga un archivo de script adicional que incluye funciones específicas para la creación de gráficos. Posteriormente, se leen las bases de datos `encuesta_sta` y `muestra_ampliada`, que contienen los datos de la encuesta y del censo, respectivamente.

El análisis procede realizando comparaciones entre los datos censales y los datos de la encuesta en diferentes variables categóricas, tales como edad, sexo, nivel educativo y estado. Para cada una de estas variables, se generan gráficos que permiten visualizar las diferencias. Utilizando la librería `patchwork`, se combinan estos gráficos en un solo diseño para facilitar la comparación. Además, se llevan a cabo análisis adicionales en variables como lengua indígena y discapacidad, y se exploran las interacciones entre variables como sexo y edad, nivel educativo y sexo, y estado y sexo. Finalmente, los gráficos que ilustran estas interacciones se integran en un diseño consolidado para una visualización comprensiva de los efectos.

Limpieza del Entorno y Carga de Bibliotecas

Se realiza una limpieza del entorno en R para eliminar todos los objetos y liberar memoria, asegurando que no haya interferencias de variables o datos previos en el análisis actual. Posteriormente, se cargan las bibliotecas necesarias para la manipulación de datos y la creación de gráficos. Entre las bibliotecas cargadas se encuentran `tidyverse` para la manipulación y visualización de datos, `reshape2` para la reorganización de datos, `stringr` para el manejo de cadenas de texto, `ggalt` para gráficos avanzados, `gridExtra` para la disposición de múltiples gráficos, `scales` para la adaptación de escalas en los gráficos, `formatR` para el formateo de código, y `patchwork` para combinar múltiples gráficos.

Se establece un tema predeterminado para los gráficos mediante `bayesplot::theme_default()` para mantener una presentación visual consistente. Además, se carga un archivo de script adicional, `Plot_validacion.R`, que contiene funciones personalizadas para la validación de gráficos, permitiendo así una integración eficiente y consistente de las herramientas gráficas necesarias para el análisis.

```
#### Cleaning R environment ###
rm(list = ls())

#################
#### Libraries ###
#################
library(tidyverse)
library(reshape2)
library(stringr)
library(ggalt)
library(gridExtra)
library(scales)
library(formatR)
library(patchwork)

theme_set(bayesplot::theme_default())

source(file = "../source/Plot_validacion.R", encoding = "UTF-8")
```

Lectura de Datos

Se leen las bases de datos necesarias para el análisis. La primera base de datos, `encuesta_sta`, se carga desde el archivo `../input/2020/enigh/encuesta_sta.rds`, y la segunda, `muestra_ampliada`, se carga desde el archivo `../input/2020/muestra_ampliada/muestra_`. Estos conjuntos de datos proporcionan la información requerida para las siguientes etapas del análisis.

```
encuesta_sta <- readRDS("../input/2020/enigh/encuesta_sta.rds")
muestra_ampliada <- readRDS("../input/2020/muestra_ampliada/muestra_cuestionario_ampli
```

Comparación de Variables

Se comparan las distribuciones de diferentes variables entre los datos censales y los de la encuesta mediante la función `Plot_Compare`.

Para analizar la variable **edad**, se crea el gráfico `age_plot`, que compara las distribuciones de edad en los datos censales y en los de la encuesta.

Se genera el gráfico `sex_plot` para comparar las distribuciones de **sexo** entre ambas

fuentes de datos.

El gráfico `escolar_plot` se utiliza para comparar el **nivel educativo** en los datos censales con los datos de la encuesta.

Finalmente, el gráfico `depto_plot` compara las distribuciones de **entidad** entre los datos censales y de la encuesta. Estos gráficos permiten evaluar las diferencias en las características de la población entre las dos fuentes de datos.

```
#### AGE ###
age_plot <-
  Plot_Compare(dat_censo = muestra_ampliada,
               dat_encuesta = encuesta_sta,
               by = "edad")
#### Sex ###
sex_plot <-
  Plot_Compare(dat_censo = muestra_ampliada,
               dat_encuesta = encuesta_sta,
               by = "sexo")
#### Level of schooling (LoS) ###
escolar_plot <-
  Plot_Compare(dat_censo = muestra_ampliada,
               dat_encuesta = encuesta_sta,
               by = "nivel_edu")

#### States ###
depto_plot <-
  Plot_Compare(dat_censo = muestra_ampliada,
               dat_encuesta = encuesta_sta,
               by = "ent")
```

Se crean gráficos comparativos para edad, sexo, nivel educativo y estados. Estos gráficos se combinan utilizando `patchwork`.

```
#--- Patchwork en acción ---#
(age_plot | sex_plot | escolar_plot) / (depto_plot)
```

Comparación de Lengua Indígena y Discapacidad

Para comparar las variables de **lengua indígena** y **discapacidad** entre los datos censales y los datos de la encuesta, se utilizan dos gráficos generados mediante la función `Plot_Compare`.

El gráfico `hlengua` muestra la comparación de la distribución de la variable **lengua indígena** en los datos censales frente a los datos de la encuesta.

Por otro lado, el gráfico `plot_discapacidad` compara la distribución de la variable **discapacidad** entre ambos conjuntos de datos.

Ambos gráficos se visualizan conjuntamente usando el operador `|`, permitiendo una comparación lado a lado de estas dos variables.

```
hlengua <- Plot_Compare(dat_censo = muestra_ampliada,
                dat_encuesta = encuesta_sta,
                by = "hlengua")

plot_discapacidad <- Plot_Compare(dat_censo = muestra_ampliada,
              dat_encuesta = encuesta_sta,
              by = "discapacidad")

(plot_discapacidad | hlengua )
```

Efectos de Interacción

Para analizar los efectos de interacción entre diferentes variables en los datos de la encuesta, se utiliza la función plot_interaction. A continuación, se presentan dos análisis específicos:

Interacción Edad x Sexo

Se examina cómo varía el porcentaje de personas en situación de pobreza en función de la combinación de las variables **edad** y **sexo**. El gráfico `p_sex_age` ilustra esta interacción, mostrando cómo la pobreza se distribuye entre diferentes grupos etarios y de sexo.

```
#### AGE x SEX ###
encuesta_sta$pobreza <- encuesta_sta$ictpc
#--- Percentage of people in poverty by AGE x SEX ---#
p_sex_age <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "edad")
```

Interacción Nivel Educativo x Sexo

Se investiga la relación entre **nivel educativo** y **sexo** en cuanto a su influencia en la pobreza. El gráfico `p_sex_escolar` representa esta interacción, proporcionando una visión de cómo el nivel educativo combinado con el sexo afecta las tasas de pobreza.

```
#### Level of schooling (LoS) x SEX ###
p_sex_escolar <-
```

```
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "nivel_edu")
```

Interacción Estado x Sexo

Este análisis explora cómo varía la pobreza en función del **estado** y el **sexo**. El gráfico `p_sex_depto` muestra la distribución de la pobreza entre diferentes estados y géneros, proporcionando una visión sobre cómo estas variables combinadas influyen en la pobreza.

```
#### State x SEX ###
p_sex_depto <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "ent")
```

Se combinan los gráficos de interacción de **Edad x Sexo** y **Nivel Educativo x Sexo** utilizando la funcionalidad `patchwork`. Esto permite una visualización conjunta para comparar fácilmente las interacciones entre las diferentes combinaciones de variables.

```
#--- Patchwork in action ---#
(p_sex_age + p_sex_escolar) / p_sex_depto
```

Interacción Nivel Educativo x Edad

Se analiza cómo el **nivel educativo** y la **edad** interactúan en relación con la pobreza. El gráfico `p_escolar_edad` muestra cómo varía el nivel educativo entre diferentes grupos etarios y se ajusta el tema para colocar la leyenda en la parte inferior.

```
#### Level of schooling (LoS) x AGE ###
p_escolar_edad <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "nivel_edu",
                   by2 = "edad") +
  theme(legend.position = "bottom") + labs(colour = "nivel_edu")
```

Interacción Estado x Edad

Se investiga la interacción entre el **estado** y la **edad** en cuanto a la pobreza. El gráfico `p_depto_edad` muestra la relación entre estos factores y se ajusta el tema para que la leyenda esté en la parte inferior.

```
#### State x AGE ###
p_depto_edad <-
```

```
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "edad",
                   by2 = "ent") +
  theme(legend.position = "bottom") + labs(colour = "Edad")

p_escolar_edad / p_depto_edad
```

Interacción Nivel Educativo x Estado

Se examina la relación entre el **nivel educativo** y el **estado** en términos de pobreza. El gráfico `p_depto_escolar` proporciona una visión de cómo el nivel educativo varía entre diferentes estados.

```
#### Level of schooling (LoS) x State ###
p_depto_escolar <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "nivel_edu",
                   by2 = "ent") +
  theme(legend.position = "bottom") + labs(colour = "nivel_edu")

p_depto_escolar
```

Repetición del Análisis para ic_segsoc e ic_ali_nc

Para realizar un análisis detallado de diferentes carencias, se repite el procedimiento de interacción para cada indicador específico. Este enfoque permite comparar cómo las interacciones entre variables afectan distintos aspectos de la pobreza. A continuación se presenta la descripción del análisis:

Interacción Edad x Sexo: Se analiza la interacción entre **edad** y **sexo** en relación con el indicador de pobreza `ic_segsoc`. El gráfico `p_sex_age` muestra cómo varía la pobreza, medida por `ic_segsoc`, en función de la combinación de edad y sexo.

Interacción Nivel Educativo x Sexo: Se examina cómo el **nivel educativo** interactúa con el **sexo** en relación con el indicador `ic_segsoc`. El gráfico `p_sex_escolar` proporciona una visión de la pobreza según el nivel educativo entre diferentes géneros.

Interacción Estado x Sexo: El gráfico `p_sex_depto` muestra la pobreza medida por `ic_segsoc` en función del **estado** y el **sexo**, ayudando a entender cómo varía la pobreza entre diferentes estados y géneros.

Combinación de Gráficos de Interacción: Se combinan los gráficos de interacción de **Edad x Sexo**, **Nivel Educativo x Sexo**, y **Estado x Sexo** usando `patchwork`,

para una visualización consolidada de cómo estas combinaciones afectan el indicador `ic_segsoc`.

Interacción Nivel Educativo x Edad: El gráfico `p_escolar_edad` muestra cómo el **nivel educativo** y la **edad** interactúan en relación con el indicador `ic_segsoc`. Se ajusta la leyenda para que esté en la parte inferior, facilitando la interpretación.

Interacción Estado x Edad: Se explora la relación entre **edad** y **estado** en función del indicador `ic_segsoc`. El gráfico `p_depto_edad` ilustra cómo varía la pobreza entre diferentes edades y estados.

Interacción Nivel Educativo x Estado: Finalmente, el gráfico `p_depto_escolar` muestra cómo el **nivel educativo** y el **estado** interactúan en relación con `ic_segsoc`, proporcionando una visión de la pobreza según el nivel educativo en distintos estados.

Resultados para ic_segsoc

```
#### AGE x SEX ###
encuesta_sta$pobreza <- encuesta_sta$ic_segsoc
#--- Percentage of people in poverty by AGE x SEX ---#
p_sex_age <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "edad")

#### Level of schooling (LoS) x SEX ###
p_sex_escolar <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "nivel_edu")

#### State x SEX ###
p_sex_depto <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "ent")

#--- Patchwork in action ---#
(p_sex_age + p_sex_escolar) / p_sex_depto

#### Level of schooling (LoS) x AGE ###
p_escolar_edad <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "nivel_edu",
```

```
                    by2 = "edad") +
  theme(legend.position = "bottom") + labs(colour = "nivel_edu")

#### State x AGE ###
p_depto_edad <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "edad",
                   by2 = "ent") +
  theme(legend.position = "bottom") + labs(colour = "Edad")

p_escolar_edad / p_depto_edad

#### Level of schooling (LoS) x State ###
p_depto_escolar <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "nivel_edu",
                   by2 = "ent") +
  theme(legend.position = "bottom") + labs(colour = "nivel_edu")

p_depto_escolar
```

Resultados para ic_ali_nc

```
#### AGE x SEX ###
encuesta_sta$pobreza <- encuesta_sta$ic_ali_nc
#--- Percentage of people in poverty by AGE x SEX ---#
p_sex_age <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "edad")

#### Level of schooling (LoS) x SEX ###
p_sex_escolar <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "nivel_edu")

#### State x SEX ###
p_sex_depto <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "sexo",
                   by2 = "ent")
```

```
#--- Patchwork in action ---#
(p_sex_age + p_sex_escolar) / p_sex_depto

#### Level of schooling (LoS) x AGE ###
p_escolar_edad <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "nivel_edu",
                   by2 = "edad") +
  theme(legend.position = "bottom") + labs(colour = "nivel_edu")

#### State x AGE ###
p_depto_edad <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "edad",
                   by2 = "ent") +
  theme(legend.position = "bottom") + labs(colour = "Edad")

p_escolar_edad / p_depto_edad

#### Level of schooling (LoS) x State ###
p_depto_escolar <-
  plot_interaction(dat_encuesta = encuesta_sta,
                   by = "nivel_edu",
                   by2 = "ent") +
  theme(legend.position = "bottom") + labs(colour = "nivel_edu")

p_depto_escolar
```

06_Modelo_unidad_ictpc.R

Para la ejecución del presente archivo, debe abrir el archivo **06_Modelo_unidad_ictpc.R** disponible en la ruta *Rcodes/2020/06_Modelo_unidad_ictpc.R*.

Este script en R se enfoca en la creación de un modelo multinivel utilizando datos de encuestas y censos para predecir ingresos. El proceso comienza con la limpieza del entorno de trabajo mediante `rm(list = ls())` y la carga de librerías esenciales para el análisis, tales como `patchwork`, `nortest`, `lme4`, `tidyverse`, `magrittr`, `caret`, `car`, y una fuente externa de modelos (`source("source/modelos_freq.R")`). Posteriormente, se definen las variables de agregación (`byAgrega`), se incrementa el límite de memoria disponible, y se cargan las bases de datos necesarias (`encuesta_sta`, `censo_sta`, `statelevel_predictors_df`). Se seleccionan las variables covariantes excluyendo aquellas que comienzan con `hog_` o `cve_mun`.

En la selección de variables relevantes, se mencionan algunos pasos comentados que realizan la selección de características utilizando el método `rfe` de `caret`. Las variables seleccionadas se listan explícitamente en `variables_seleccionadas` y se combinan con otras covariantes para formar `cov_names`. Luego, se construye la fórmula del modelo (`formula_model`), que incluye efectos aleatorios para `cve_mun`, `hlengua` y `discapacidad`, así como efectos fijos para las demás variables. Esto asegura que el modelo considere tanto la variabilidad dentro de cada grupo como las características específicas de cada variable.

Finalmente, se realiza una transformación la variable de ingreso (`ingreso`) a formato numérico y se ajusta el modelo utilizando la función `modelo_ingreso`, que está definida en el archivo fuente cargado previamente. Los datos de entrada incluyen `encuesta_sta`, `statelevel_predictors_df`, `censo_sta` y la fórmula del modelo. Los resultados del modelo se guardan en un archivo RDS (`fit_mrp_ictpc.rds`). Se generan y guardan gráficos de densidad y histogramas de las distribuciones posteriores utilizando `ggsave`, proporcionando una visualización clara de los resultados obtenidos.

Limpieza del Entorno y Carga de Bibliotecas

Para preparar el entorno de análisis, se inicia con la limpieza del espacio de trabajo en R, eliminando todos los objetos existentes y ejecutando el recolector de basura para liberar

memoria. Esta acción asegura que no haya conflictos o residuos de análisis anteriores que puedan afectar el trabajo actual. A continuación, se cargan las bibliotecas necesarias para el análisis. Entre estas se encuentran `patchwork` para la combinación de gráficos, `nortest` para pruebas de normalidad, `lme4` para modelos lineales mixtos, `tidyverse` para manipulación y visualización de datos, `magrittr` para la sintaxis de tuberías, `caret` para la creación de modelos predictivos, y `car` para técnicas de regresión y diagnóstico. Además, se incluye un archivo de funciones adicionales `modelos_freq.R`, que contiene scripts personalizados para el análisis de modelos.

Este entorno preparado permite realizar un análisis riguroso y detallado de los datos, asegurando que todas las herramientas y funciones necesarias estén disponibles y actualizadas.

```
rm(list =ls())

# Loading required libraries ---------------------------------------------

library(patchwork)
library(nortest)
library(lme4)
library(tidyverse)
library(magrittr)
library(caret)
library(car)
source("../source/modelos_freq.R")
```

Variables de Agregación

Se define un vector de variables llamado `byAgrega`, que incluye los campos `ent`, `cve_mun`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`, y `nivel_edu`. Estas variables se utilizarán para la agregación de datos en el análisis, permitiendo estructurar y resumir la información según diferentes dimensiones y características.

```
byAgrega <-
  c("ent",
    "cve_mun",
    "area",
    "sexo",
    "edad",
    "discapacidad",
    "hlengua",
    "nivel_edu" )
```

Carga de Datos

Se cargan los datos necesarios para el análisis, incluyendo la encuesta (`encuesta_sta`), el censo (`censo_sta`), y los predictores a nivel estatal (`statelevel_predictors_df`).

La memoria se ajusta a un límite de 10 millones para manejar estos datos. Tras la carga de los archivos RDS, se extraen los nombres de las covariables de `statelevel_predictors_df`, excluyendo aquellos que comienzan con "hog_" y el identificador de municipio (`cve_mun`). Esto permite enfocarse en las variables relevantes para el análisis.

```
# Loading data ------------------------------------------------------------
memory.limit(10000000)
encuesta_sta <- readRDS("../input/2020/enigh/encuesta_sta.rds")
censo_sta <- readRDS("../input/2020/muestra_ampliada/muestra_cuestionario_ampliado.rds
statelevel_predictors_df <- readRDS("../input/2020/predictores/statelevel_predictors_d

cov_names <- names(statelevel_predictors_df)
cov_names <- cov_names[!grepl(x = cov_names,pattern = "^hog_|cve_mun")]
```

Selección de Variables

Se procede a la selección de las variables predictoras que se utilizarán en el modelo. Se inicia por definir un conjunto inicial de variables y luego se elige un subconjunto relevante para el análisis. En esta fase, se han identificado y seleccionado variables predictoras clave, entre las que se encuentran indicadores relacionados con educación, ingreso, y características geográficas y económicas.

Primero, se propone una opción para seleccionar las variables usando una técnica de selección de características denominada `rfe` (recursive feature elimination), que evalúa el impacto de diferentes variables en la variable objetivo (`ingreso`). Sin embargo, esta opción está comentada en el código.

Luego, se definen explícitamente las variables seleccionadas para el análisis. Estas incluyen indicadores como `prom_esc_rel_urb`, `smg1`, `gini15m`, `ictpc15`, `altitud1000`, entre otros. Estas variables se consideran relevantes para el modelo y son seleccionadas para su uso en el análisis.

Finalmente, se filtran las variables que se registrarán para la inclusión en el análisis, excluyendo ciertas variables específicas que no serán consideradas. Este paso asegura que solo las covariables pertinentes sean utilizadas en la modelización, basándose en las necesidades específicas del análisis y los objetivos del estudio.

```
# Selección de variables

# encuesta_sta2 <- encuesta_sta %>% group_by_at((byAgrega)) %>%
```

```
#   summarise(ingreso = mean(ictpc), .groups = "drop",
#             n = n()) %>%
#   inner_join(statelevel_predictors_df)
#
#
# resultado_rfe <-
#   rfe(
#     encuesta_sta2[, cov_names],
#     encuesta_sta2$ingreso,
#     sizes = c(1:10),
#     rfeControl = rfeControl(functions = lmFuncs, method = "LOOCV")
#   )

variables_seleccionadas <-
  c(
    "prom_esc_rel_urb",
    "smg1",
    "gini15m" ,
    "ictpc15" ,
    "altitud1000",
    "prom_esc_rel_rur" ,
    "porc_patnoagrnocal_urb",
    "acc_medio"  ,
    "derhab_pea_15_20" ,
    "porc_norep_ing_urb"
  )
cov_names <- c(
  "modifica_humana",  "acceso_hosp",
  "acceso_hosp_caminando",  "cubrimiento_cultivo",
  "cubrimiento_urbano",     "luces_nocturnas" ,
  variables_seleccionadas
)

cov_registros <- setdiff(
  cov_names,
  c(
    "elec_mun20",
    "elec_mun19",
    "transf_gobpc_15_20",
    "derhab_pea_15_20",
    "vabpc_15_19" ,
    "itlpis_15_20"  ,
```

```
    "remespc_15_20",
    "desem_15_20",
    "smg1",
    "ql_porc_cpa_urb",
    "ql_porc_cpa_rur"
  )
)
```

Fórmula del Modelo

Se define la fórmula del modelo para el análisis, la cual especifica cómo se relacionan las variables predictoras con la variable dependiente `ingreso`. Esta fórmula incluye tanto efectos aleatorios como efectos fijos para capturar diferentes niveles de variabilidad en los datos.

En la fórmula del modelo, se incorporan efectos aleatorios para los códigos municipales (`cve_mun`), el habla de lenguas indígenas (`hlengua`), y la discapacidad (`discapacidad`). Estos efectos aleatorios permiten capturar la variabilidad específica entre estos grupos y ajustar el modelo en consecuencia.

Además, se incluyen varios efectos fijos que representan variables independientes clave en el análisis: entidad (`ent`), nivel educativo (`nivel_edu`), edad (`edad`), área (`area`), y sexo (`sexo`). Estas variables fijas proporcionan información sobre cómo estos factores afectan la variable dependiente.

Finalmente, se incorporan las variables predictoras seleccionadas, que se concatenan en la fórmula como términos adicionales. Estas variables predictoras se han seleccionado previamente como relevantes para el modelo y se añaden para evaluar su impacto en el ingreso. La fórmula resultante combina estos elementos para construir un modelo completo que considera tanto efectos aleatorios como fijos en el análisis de los datos.

```
cov_registros <- paste0(cov_registros, collapse = " + ")

formula_model <-
  paste0("ingreso ~ (1 | cve_mun) + (1 | hlengua) + (1 | discapacidad) + ent + nivel_e
      " + ", cov_registros)
```

Ajuste del Modelo

Se ajusta el modelo de ingreso utilizando la función `modelo_ingreso`, la cual se aplica a los datos de la encuesta (`encuesta_sta`) junto con los predictores a nivel estatal (`statelevel_predictors_df`) y la tabla censal (`censo_sta`). La fórmula del modelo, previamente definida (`formula_model`), se emplea para especificar las relaciones entre las variables y los efectos aleatorios y fijos en el análisis.

Una vez ajustado el modelo, se guardan los resultados en un archivo `.rds` en la ruta especificada, lo que permite almacenar y compartir el modelo ajustado para su posterior análisis o uso. Además, se generan y guardan visualizaciones de los resultados del modelo, incluyendo un gráfico de densidad y un histograma de la posteriori, que se almacenan como archivos de imagen (`.png`). Estos gráficos permiten examinar la distribución de los resultados y la calidad del ajuste del modelo, proporcionando una representación visual de los resultados obtenidos.

```
encuesta_sta$ingreso <- as.numeric(encuesta_sta$ictpc)

fit <- modelo_ingreso(
  encuesta_sta = encuesta_sta ,
  predictors = statelevel_predictors_df,
  censo_sta = censo_sta,
  formula_mod = formula_model,
  byAgrega = byAgrega
)

#--- Exporting Bayesian Multilevel Model Results ---#

saveRDS(fit,
        file = "../output/2020/modelos/fit_mrp_ictpc.rds")

ggsave(plot = fit$plot_densy,
       "../output/2020/plots/01_densidad_ictpc.png",scale = 3)
fit$plot_hist_post
```

07_Modelo_unidad_py_ic_ali_nc.R

Para la ejecución del presente archivo, debe abrir el archivo **07_Modelo_unidad_py_ic_ali_nc.R** disponible en la ruta *Rcodes/2020/07_Modelo_unidad_py_ic_ali_nc.R*.

Este script en R se centra en la creación de un modelo multinivel para predecir la carencia en alimentos nutritivos y de calidad , utilizando datos de encuestas y censos. Comienza limpiando el entorno de trabajo (`rm(list = ls())`) y cargando las librerías necesarias para la integración con Python. Además, se importan los módulos `pandas` y las bibliotecas `sklearn.feature_selection` y `sklearn.ensemble` de Python. También se carga una fuente externa de modelos (`source("source/modelos_freq.R")`). Luego, se definen las variables para la agregación (`byAgrega`) y se aumenta el límite de memoria disponible. Las bases de datos necesarias (`encuesta_sta`, `censo_sta` y `statelevel_predictors_df`) se cargan, y se seleccionan las variables covariantes relevantes excluyendo aquellas que comienzan con `hog_` o `cve_mun`.

El script incluye un paso comentado para la selección de características utilizando el método RFE de `caret` y Python, pero en esta versión, se listan explícitamente las variables seleccionadas (`variables_seleccionadas`) y se combinan con otras covariantes para formar `cov_names`. Luego, se construye la fórmula del modelo (`formula_model`), que incluye efectos aleatorios para `cve_mun`, `hlengua` y `discapacidad`, así como efectos fijos para las demás variables. Se ajusta el modelo utilizando la función `modelo_dummy`, que está definida en el archivo fuente cargado previamente. Los datos de entrada incluyen `encuesta_sta` (con una nueva variable `yk`), `statelevel_predictors_df`, `censo_sta` y la fórmula del modelo. Este enfoque permite una modelización robusta y ajustada a las especificaciones de los datos disponibles.

Finalmente, los resultados del modelo se guardan en un archivo RDS (`fit_mrp_ic_ali_nc.rds`). Este paso asegura que los resultados del análisis estén disponibles para futuras referencias y análisis adicionales. Guardar los resultados en un archivo RDS facilita su recuperación y análisis posterior, permitiendo a los investigadores y analistas continuar trabajando con los resultados sin necesidad de recalcular el modelo cada vez. Además, se generan gráficos de densidad y histogramas de las distribuciones posteriores utilizando ggsave, lo que proporciona una visualización clara y comprensible de los resultados del modelo.

Limpieza del Entorno y Carga de Bibliotecas

En esta etapa, se realiza la limpieza del entorno de R para eliminar objetos previamente cargados y garantizar un inicio limpio para el análisis. Esto se hace utilizando el comando `rm(list = ls())`, que elimina todos los objetos del espacio de trabajo actual.

A continuación, se cargan las bibliotecas necesarias para el análisis. Estas bibliotecas incluyen:

- **`patchwork`**: Para la combinación de múltiples gráficos en una sola visualización.
- **`nortest`**: Proporciona pruebas de normalidad para analizar la distribución de los datos.
- **`lme4`**: Utilizada para ajustar modelos lineales mixtos, que permiten manejar datos con estructura jerárquica.
- **`tidyverse`**: Un conjunto de paquetes para la manipulación y visualización de datos.
- **`magrittr`**: Proporciona el operador `%>%` para facilitar el encadenamiento de operaciones en R.
- **`caret`**: Para la creación de modelos de machine learning y la selección de variables.
- **`car`**: Ofrece herramientas para el análisis de regresión y diagnóstico de modelos.
- **`randomForest`**: Implementa el algoritmo de bosques aleatorios para clasificación y regresión.
- **`reticulate`**: Permite la integración de Python en R, facilitando el uso de bibliotecas de Python desde el entorno R.

Además, se importan módulos específicos de Python usando `reticulate`:

- **`pandas`**: Para la manipulación de datos en estructuras tabulares.
- **`sklearn.feature_selection`**: Para la selección de características en modelos de machine learning.
- **`sklearn.ensemble`**: Proporciona métodos de ensamblaje, como los bosques aleatorios y gradientes impulsados.

Finalmente, se carga un script adicional, `modelos_freq.R`, que probablemente contiene funciones personalizadas o modelos específicos necesarios para el análisis. Esto asegura que todas las herramientas y funciones requeridas estén disponibles para proceder con el análisis.

```
rm(list =ls())

# Loading required libraries ----------------------------------------------

library(patchwork)
library(nortest)
library(lme4)
library(tidyverse)
```

```r
library(magrittr)
library(caret)
library(car)
library(randomForest)
library(reticulate)

pd <- import("pandas")
sklearn_fs <- import("sklearn.feature_selection")
sklearn_ensemble <- import("sklearn.ensemble")

source("../source/modelos_freq.R")
```

Carga de Datos

En esta fase, se procede a la carga de los datos necesarios para el análisis. Se utilizan las funciones `readRDS()` para leer archivos en formato RDS, que es un formato de almacenamiento de objetos en R.

1. **Carga de la Encuesta:** Se carga el conjunto de datos de la encuesta, almacenado en el archivo `../input/2020/enigh/encuesta_sta.rds`. Este conjunto de datos contiene información sobre las variables de interés para el análisis.

2. **Carga del Censo:** Se lee el archivo `../input/2020/muestra_ampliada/muestra_cuestionario_` que contiene datos censales ampliados. Estos datos se usarán para la calibración y comparación con los datos de la encuesta.

3. **Carga de Predictores Estatales:** Se carga el archivo `../input/2020/predictores/statelevel` que incluye predictores a nivel estatal que se utilizarán para el análisis.

Además, se extraen los nombres de las variables del dataframe `statelevel_predictors_df` y se filtran para eliminar las variables que comienzan con "hog_" o "cve_mun", asegurando que solo se conserven las variables relevantes para el análisis posterior. Esto se realiza mediante la función `grepl()` para identificar y excluir las variables no deseadas.

```r
# Loading data ------------------------------------------------------------

memory.limit(10000000)
encuesta_sta <- readRDS("../input/2020/enigh/encuesta_sta.rds")
censo_sta <- readRDS("../input/2020/muestra_ampliada/muestra_cuestionario_ampliado.rds
statelevel_predictors_df <- readRDS("../input/2020/predictores/statelevel_predictors_d

cov_names <- names(statelevel_predictors_df)
cov_names <- cov_names[!grepl(x = cov_names,pattern = "^hog_|cve_mun")]
```

Selección de Variables

En esta etapa, se definen y seleccionan las variables que serán utilizadas en el análisis posterior.

1. **Variables de Agregación:** Se establece un vector llamado `byAgrega` que contiene las variables de agregación. Estas variables son fundamentales para estructurar y resumir los datos de acuerdo a diferentes dimensiones y características. El vector incluye:
   - `ent` (Entidad Federativa)
   - `cve_mun` (Clave de Municipio)
   - `area` (Área)
   - `sexo` (Sexo)
   - `edad` (Edad)
   - `discapacidad` (Discapacidad)
   - `hlengua` (Lengua Indígena)
   - `nivel_edu` (Nivel Educativo)
2. **Selección de Variables Predictoras:** Se identifican las variables predictoras más relevantes para el análisis. En el código comentado, se había planificado usar un modelo de Random Forest y el método RFE (Recursive Feature Elimination) de Python para seleccionar las variables predictoras más importantes. Sin embargo, el código relacionado con la integración de Python se encuentra comentado.

En lugar de ello, se definen directamente las variables seleccionadas y se especifica un vector `cov_names` que incluye las variables predictoras relevantes para el análisis. Estas variables son: - `modifica_humana` (Modificación Humana) - `acceso_hosp` (Acceso a Hospital) - `acceso_hosp_caminando` (Acceso a Hospital Caminando) - `cubrimiento_cultivo` (Cobertura de Cultivo) - `cubrimiento_urbano` (Cobertura Urbana) - `luces_nocturnas` (Luces Nocturnas) - Variables seleccionadas como `porc_ing_ilpi_urb`, `pob_ind_rur`, `pob_ind_urb`, `porc_hogremesas_rur`, `porc_segsoc15`, `porc_ali15`, `plp15`, `pob_rur`, y `altitud1000`.

Estos pasos aseguran que se utilicen las variables más relevantes para el análisis, optimizando la capacidad predictiva y el rendimiento del modelo.

```
## Selección de variables

byAgrega <-
  c("ent",
    "cve_mun",
    "area",
    "sexo",
    "edad",
    "discapacidad",
```

```
    "hlengua",
    "nivel_edu" )


# encuesta_sta2 <- encuesta_sta %>%   mutate(
#   yk = as.factor(ifelse(ic_ali_nc == 1 ,1,0))) %>%
#   inner_join(statelevel_predictors_df[, c("cve_mun",cov_names)])
#
# # Convertir 'encuesta_sta2' a un dataframe de Python
# encuesta_sta2_py <- pd$DataFrame(encuesta_sta2)
#
# # Obtener 'X' y 'y' del dataframe de Python
# X <- encuesta_sta2_py[cov_names]
# y <- encuesta_sta2_py[['yk']]
#
# # Crear el modelo de clasificación, por ejemplo, un Random Forest
# modelo <- sklearn_ensemble$RandomForestClassifier()
#
# # Crear el selector RFE con el modelo y el número de características a seleccionar
# selector <- sklearn_fs$RFE(modelo, n_features_to_select = as.integer(10))
#
# # Ajustar los datos
# selector$fit(X, y)
# # Obtener las variables seleccionadas
# variables_seleccionadas <- X[selector$support_] %>% names()

variables_seleccionadas <- c(
  "porc_ing_ilpi_urb",
  "pob_ind_rur",
  "pob_ind_urb",
  "porc_hogremesas_rur",
  "porc_segsoc15",
  "porc_ali15",
  "plp15",
  "pob_rur",
  "altitud1000"
)
cov_names <- c(
  "modifica_humana",  "acceso_hosp",
  "acceso_hosp_caminando",  "cubrimiento_cultivo",
  "cubrimiento_urbano",     "luces_nocturnas" ,
  variables_seleccionadas
```

```
)
```

Definición de la Fórmula del Modelo

Se define una fórmula para el modelo que incluye tanto efectos aleatorios como variables predictoras seleccionadas. Primero, se determina el conjunto de variables relevantes excluyendo algunas específicas que no se incluirán en el análisis. Estas exclusiones se hacen utilizando la función `setdiff`, que elimina variables como `elec_mun20`, `elec_mun19`, `transf_gobpc_15_20`, y otras relacionadas con transferencias, electrificación, y características del desempeño y calidad.

El vector resultante de variables seleccionadas se concatena en una cadena de texto para construir la fórmula del modelo. Esta fórmula se configura para ajustar el modelo a datos binomiales, utilizando `cbind(si, no)` como la variable dependiente. La fórmula también incorpora efectos aleatorios para la clave de municipio `(1 | cve_mun)`, la lengua indígena `(1 | hlengua)`, y la discapacidad `(1 | discapacidad)`. Además, incluye efectos fijos para `nivel_edu`, `edad`, `ent`, `area`, y `sexo`. Finalmente, se agregan las variables predictoras seleccionadas.

```
cov_registros <-
  setdiff(
    cov_names,
    c(
      "elec_mun20",
      "elec_mun19",
      "transf_gobpc_15_20",
      "derhab_pea_15_20",
      "vabpc_15_19" ,
      "itlpis_15_20"  ,
      "remespc_15_20",
      "desem_15_20",
      "porc_urb" ,
      "edad65mas_urb",
      "pob_tot" ,
      "acc_muyalto" ,
      "smg1",
      "ql_porc_cpa_rur",
      "ql_porc_cpa_urb"
    )
  )

cov_registros <- paste0(cov_registros, collapse = " + ")
```

La fórmula completa se define como:

```
formula_model <-
  paste0(
    "cbind(si, no) ~ (1 | cve_mun) + (1 | hlengua) + (1 | discapacidad) +  nivel_edu +
    ,
    " + ",
    cov_registros
  )
```

Esta estructura permite capturar la variabilidad entre los municipios y otras dimensiones de la población, mientras que también evalúa el impacto de las variables predictoras sobre la variable dependiente.

Ajuste del Modelo

Se ajusta el modelo utilizando la función `modelo_dummy`, que se aplica a los datos de la encuesta junto con los predictores y el censo. Primero, se transforma la variable `ic_ali_nc` en una variable binaria `yk`, donde se asigna el valor 1 si `ic_ali_nc` es igual a 1, y 0 en caso contrario. Esta transformación permite modelar la variable como un resultado binomial en el análisis.

Luego, se pasa el conjunto de datos transformado (`encuesta_sta` con la nueva variable `yk`), los predictores a nivel estatal (`statelevel_predictors_df`), y el censo estatal (`censo_sta`) a la función `modelo_dummy`, junto con la fórmula del modelo previamente definida y el vector `byAgrega` para la agregación.

Finalmente, se guarda el resultado del ajuste del modelo en un archivo RDS en la carpeta especificada, utilizando el nombre `fit_mrp_ic_ali_nc.rds`. Esto permite almacenar los resultados del modelo para su posterior análisis o uso.

```
fit <- modelo_dummy(
  encuesta_sta = encuesta_sta %>%
    mutate(yk = ifelse(ic_ali_nc == 1 , 1, 0))  ,
  predictors = statelevel_predictors_df,
  censo_sta = censo_sta,
  formula_mod = formula_model,
  byAgrega = byAgrega
)

#--- Exporting Bayesian Multilevel Model Results ---#

saveRDS(fit,
        file = "../output/2020/modelos/fit_mrp_ic_ali_nc.rds")
```

08_Modelo_unidad_py_ic_segsoc

Para la ejecución del presente análisis, se debe abrir el archivo **08_Modelo_unidad_py_ic_segsoc.R** disponible en la ruta `Rcodes/2020/08_Modelo_unidad_py_ic_segsoc.R`.

Este script en R se enfoca en la creación de un modelo multinivel para analizar la carencia en por cobertura de seguridad social usando datos de encuestas y censos. El proceso inicia con la limpieza del entorno de trabajo y la carga de librerías esenciales para el análisis. Se define el límite de memoria con `memory.limit(10000000)` para asegurar suficiente espacio durante el procesamiento de datos. Luego, se cargan los datos necesarios (`encuesta_sta`, `censo_sta`, `statelevel_predictors_df`) y se excluyen ciertas variables de los datos de predictores.

La selección de variables relevantes para el modelo se realiza a través de un procedimiento comentado que emplea la técnica de Recursive Feature Elimination (RFE) usando un modelo Random Forest implementado en Python. Las variables seleccionadas se listan explícitamente y se combinan con otras covariantes para formar cov_names. Posteriormente, se construye una fórmula del modelo (`formula_model`) que incluye efectos aleatorios para cve_mun, hlengua y discapacidad, y efectos fijos para las demás variables, considerando tanto la variabilidad dentro de cada grupo como las características específicas de cada variable.

El ajuste del modelo se realiza mediante la función modelo_dummy, transformando la variable de interés (`ic_segsoc`) en formato binario y utilizando los datos de encuesta y censo junto con la fórmula del modelo y las variables de agregación. Los resultados del modelo se guardan en un archivo RDS (`fit_mrp_ic_segsoc.rds`) y se documentan para futuros análisis. Esto incluye la exportación de los resultados del modelo multinivel y la creación de visualizaciones pertinentes para evaluar el desempeño del modelo y la distribución de las predicciones.

Limpieza del Entorno y Carga de Bibliotecas

En esta etapa, se realiza la limpieza del entorno de R para eliminar objetos previamente cargados y garantizar un inicio limpio para el análisis. Esto se hace utilizando el comando `rm(list = ls())`, que elimina todos los objetos del espacio de trabajo actual.

A continuación, se cargan las bibliotecas necesarias para el análisis. Estas bibliotecas

incluyen:

- **patchwork**: Para la combinación de múltiples gráficos en una sola visualización.
- **nortest**: Proporciona pruebas de normalidad para analizar la distribución de los datos.
- **lme4**: Utilizada para ajustar modelos lineales mixtos, que permiten manejar datos con estructura jerárquica.
- **tidyverse**: Un conjunto de paquetes para la manipulación y visualización de datos.
- **magrittr**: Proporciona el operador **%>%** para facilitar el encadenamiento de operaciones en R.
- **caret**: Para la creación de modelos de machine learning y la selección de variables.
- **car**: Ofrece herramientas para el análisis de regresión y diagnóstico de modelos.
- **randomForest**: Implementa el algoritmo de bosques aleatorios para clasificación y regresión.
- **reticulate**: Permite la integración de Python en R, facilitando el uso de bibliotecas de Python desde el entorno R.

Además, se importan módulos específicos de Python usando `reticulate`:

- **pandas**: Para la manipulación de datos en estructuras tabulares.
- **sklearn.feature_selection**: Para la selección de características en modelos de machine learning.
- **sklearn.ensemble**: Proporciona métodos de ensamblaje, como los bosques aleatorios y gradientes impulsados.

Finalmente, se carga un script adicional, `modelos_freq.R`, que probablemente contiene funciones personalizadas o modelos específicos necesarios para el análisis. Esto asegura que todas las herramientas y funciones requeridas estén disponibles para proceder con el análisis.

```
rm(list =ls())

# Loading required libraries ----------------------------------------

library(patchwork)
library(nortest)
library(lme4)
library(tidyverse)
library(magrittr)
library(caret)
library(car)
library(randomForest)
library(reticulate)
```

```
pd <- import("pandas")
sklearn_fs <- import("sklearn.feature_selection")
sklearn_ensemble <- import("sklearn.ensemble")
```

Carga de Datos

En esta fase, se procede a la carga de los datos necesarios para el análisis. Se utilizan las funciones `readRDS()` para leer archivos en formato RDS, que es un formato de almacenamiento de objetos en R.

1. **Carga de la Encuesta:** Se carga el conjunto de datos de la encuesta, almacenado en el archivo `../input/2020/enigh/encuesta_sta.rds`. Este conjunto de datos contiene información sobre las variables de interés para el análisis.
2. **Carga del Censo:** Se lee el archivo `../input/2020/muestra_ampliada/muestra_cuestionario_` que contiene datos censales ampliados. Estos datos se usarán para la calibración y comparación con los datos de la encuesta.
3. **Carga de Predictores Estatales:** Se carga el archivo `../input/2020/predictores/statelevel` que incluye predictores a nivel estatal que se utilizarán para el análisis.

Además, se extraen los nombres de las variables del dataframe `statelevel_predictors_df` y se filtran para eliminar las variables que comienzan con "hog_" o "cve_mun", asegurando que solo se conserven las variables relevantes para el análisis posterior. Esto se realiza mediante la función `grepl()` para identificar y excluir las variables no deseadas.

```
memory.limit(10000000)
source("../source/modelos_freq.R")
# Loading data ------------------------------------------------------------

memory.limit(10000000)
encuesta_sta <- readRDS("../input/2020/enigh/encuesta_sta.rds")
censo_sta <- readRDS("../input/2020/muestra_ampliada/muestra_cuestionario_ampliado.rds
statelevel_predictors_df <- readRDS("../input/2020/predictores/statelevel_predictors_

cov_names <- names(statelevel_predictors_df)
cov_names <- cov_names[!grepl(x = cov_names, pattern = "^hog_|cve_mun")]
```

Selección de Variables

Se definen las variables de agregación necesarias para el análisis y se seleccionan las variables predictoras más relevantes. El vector `byAgrega` incluye variables clave como `ent`, `cve_mun`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`, y `nivel_edu`, que se utilizarán para la agregación de los datos.

Aunque el código para la selección de variables mediante un proceso de RFE (Recursive Feature Elimination) usando Python está comentado, se detallan los pasos para el uso de esta técnica. En lugar de ejecutar el código Python, se ha seleccionado manualmente un conjunto de variables relevantes basado en el análisis previo. Estas variables seleccionadas incluyen indicadores como `porc_rur`, `porc_urb`, `porc_ing_ilpi_rur`, `porc_ing_ilpi_urb`, `porc_jub_urb`, `porc_segsoc15`, `plp15`, `ictpc15`, `pob_urb`, y `pob_tot`.

El conjunto de variables `cov_names` se compone de las variables seleccionadas más otras variables relevantes para el análisis, excluyendo algunas que no se consideran necesarias para el modelo final. Las variables que se excluyen se encuentran en el conjunto `cov_registros`, y el resultado es un vector de variables que se utilizará en la definición de la fórmula del modelo.

```
## Selección de variables

byAgrega <-
  c("ent",
    "cve_mun",
    "area",
    "sexo",
    "edad",
    "discapacidad",
    "hlengua",
    "nivel_edu" )

# encuesta_sta2 <- encuesta_sta %>%   mutate(
#   yk = as.factor(ifelse(ic_asegsoc == 1 ,1,0))) %>%
#   inner_join(statelevel_predictors_df[, c("cve_mun",cov_names)])
#
# table(encuesta_sta2$yk, encuesta_sta2$ic_segsoc)
#
# # Convertir 'encuesta_sta2' a un dataframe de Python
# encuesta_sta2_py <- pd$DataFrame(encuesta_sta2)
#
# # Obtener 'X' y 'y' del dataframe de Python
# X <- encuesta_sta2_py[cov_names]
# y <- encuesta_sta2_py[['yk']]
#
# # Crear el modelo de clasificación, por ejemplo, un Random Forest
# modelo <- sklearn_ensemble$RandomForestClassifier()
#
# # Crear el selector RFE con el modelo y el número de características a seleccionar
# selector <- sklearn_fs$RFE(modelo, n_features_to_select = as.integer(10))
```

```
#
# # Ajustar los datos
# selector$fit(X, y)
#
# # Obtener las variables seleccionadas
# variables_seleccionadas <- X[selector$support_] %>% names()

variables_seleccionadas <-
  c(
    "porc_rur",
    "porc_urb",
    "porc_ing_ilpi_rur",
    "porc_ing_ilpi_urb",
    "porc_jub_urb",
    "porc_segsoc15",
    "plp15",
    "ictpc15",
    "pob_urb",
    "pob_tot"
  )

cov_names <- c(
  "modifica_humana",
  "acceso_hosp",
  "acceso_hosp_caminando",
  "cubrimiento_cultivo",
  "cubrimiento_urbano",
  "luces_nocturnas",
  variables_seleccionadas
)

cov_registros <-
  setdiff(
    cov_names,
    c(
      "elec_mun20",
      "elec_mun19",
      "transf_gobpc_15_20",
      "derhab_pea_15_20",
      "vabpc_15_19",
      "itlpis_15_20",
      "remespc_15_20",
```

```
      "desem_15_20",
      "porc_urb",
      "edad65mas_urb",
      "pob_tot",
      "acc_muyalto",
      "smg1",
      "ql_porc_cpa_rur",
      "ql_porc_cpa_urb"
    )
  )

cov_registros <- paste0(cov_registros, collapse = " + ")
```

Definición de la Fórmula del Modelo

Se define la fórmula del modelo que será utilizada para el análisis. La fórmula especifica la estructura del modelo incluyendo los efectos aleatorios y las variables predictoras seleccionadas. En este caso, el modelo se formula para predecir una variable dependiente binaria con la estructura `cbind(si, no)`, que representa la proporción de casos positivos y negativos.

La fórmula incluye efectos aleatorios para las variables `cve_mun` (clave de municipio), `hlengua` (lengua indígena), y `discapacidad` (discapacidad), lo cual permite capturar la variabilidad no explicada a nivel de estos factores. Además, se incorporan variables fijas como `nivel_edu` (nivel educativo), `edad` (edad), `ent` (estado), `area` (área), y `sexo` (sexo), que se consideran relevantes para el modelo.

Las variables predictoras adicionales, definidas en el vector `cov_registros`, se añaden a la fórmula para completar el conjunto de variables explicativas del modelo. La fórmula resultante es utilizada para ajustar el modelo y analizar las relaciones entre las variables seleccionadas y la variable dependiente.

```
formula_model <-
  paste0(
    "cbind(si, no) ~ (1 | cve_mun) + (1 | hlengua) + (1 | discapacidad) +  nivel_edu +
    " + ",
    cov_registros
  )
```

Ajuste del Modelo

El ajuste del modelo se realiza utilizando la función `modelo_dummy`, que se aplica a los datos para estimar un modelo basado en la fórmula definida anteriormente. En este paso, la variable dependiente `yk` se crea a partir del indicador `ic_segsoc`, donde se asigna un

valor de 1 si el indicador es positivo y 0 en caso contrario. Esta transformación convierte el problema en una tarea de clasificación binaria.

La función `modelo_dummy` toma como entrada los datos de la encuesta (`encuesta_sta`), los predictores a nivel estatal (`statelevel_predictors_df`), los datos del censo (`censo_sta`), y la fórmula del modelo (`formula_model`). Además, utiliza el vector de variables de agregación (`byAgrega`) para realizar el ajuste del modelo.

Una vez ajustado el modelo, los resultados se guardan en un archivo RDS en la ruta especificada (`../output/2020/modelos/fit_mrp_ic_segsoc.rds`). Este archivo contiene el modelo ajustado, que puede ser utilizado para análisis posteriores o para generar predicciones basadas en los datos de entrada.

```
fit <- modelo_dummy(
  encuesta_sta = encuesta_sta %>%  mutate(yk = ifelse(ic_segsoc == 1 ,1,0)),
  predictors = statelevel_predictors_df,
  censo_sta = censo_sta,
  formula_mod = formula_model,
  byAgrega = byAgrega
)

#--- Exporting Bayesian Multilevel Model Results ---#

saveRDS(fit, file = "../output/2020/modelos/fit_mrp_ic_segsoc.rds")
```

09_PostBenchmarking_ictpc.R

Para la ejecución del presente archivo, debe abrir el archivo **09_PostBenchmarking_ictpc.R** disponible en la ruta *Rcodes/2020/09_PostBenchmarking_ictpc.R*.

Este script en R se centra en la validación y benchmarking de un modelo multinivel para predecir ingresos, utilizando datos de encuestas y censos. Comienza limpiando el entorno de trabajo (`rm(list = ls())`) y cargando las librerías necesarias. Además, se leen las funciones auxiliares desde archivos externos (`Plot_validacion_bench.R` y `Benchmarking.R`). Posteriormente, se carga el modelo preentrenado desde un archivo RDS (`fit_mrp_ictpc.rds`).

El proceso de benchmarking se realiza mediante la función `benchmarking`, utilizando covariables específicas como `ent`, `area`, `sexo`, `edad`, `discapacidad` y `hlengua`. El resultado se guarda en `list_bench`. A continuación, se crea un diseño de encuesta (`diseno_encuesta`) y se realizan validaciones a nivel nacional comparando los resultados del modelo ajustado (`yk_lmer`) y del modelo ajustado con benchmarking (`yk_bench`). Estas validaciones se llevan a cabo mediante la comparación de medias ponderadas.

Para validar el modelo a nivel de subgrupos, se definen diversos subgrupos como `ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua` y `nivel_edu`. Se crean gráficos de validación para cada subgrupo utilizando la función `plot_uni_validacion`. Estos gráficos se combinan y se guardan en un archivo JPG (`plot_uni.jpg`). Además, se guarda la lista de gráficos y el dataframe postestratificado en archivos RDS (`plot_uni.rds` y `poststrat_df_ictpc.rds` respectivamente), lo que permite una fácil recuperación y análisis posterior.

Limpieza del Entorno y Carga de Bibliotecas

El entorno de R se limpia al eliminar todos los objetos actuales y se limpia la consola para asegurar que el análisis comience desde un estado limpio. A continuación, se cargan las bibliotecas necesarias para el análisis, que incluyen `scales`, `patchwork`, `srvyr`, `survey`, `haven`, `sampling`, y `tidyverse`. Estas bibliotecas proporcionan herramientas para la manipulación de datos, la creación de gráficos, el análisis de encuestas y la realización de muestreos.

```
rm(list =ls())
cat("\f")
###############################################################
# Loading required libraries ------------------------------------------
###############################################################

library(scales)
library(patchwork)
library(srvyr)
library(survey)
library(haven)
library(sampling)
library(tidyverse)
```

Lectura de Funciones

Se cargan dos archivos de funciones que serán utilizadas en el análisis: `Plot_validacion_bench.R` y `Benchmarking.R`. Estos archivos contienen funciones personalizadas para la validación y el análisis comparativo.

```
source("../source/Plot_validacion_bench.R", encoding = "UTF-8")
source("../source/Benchmarking.R", encoding = "UTF-8")
```

Lectura del Modelo

Se lee el modelo ajustado previamente, guardado en un archivo RDS (`fit_mrp_ictpc.rds`). Este modelo será utilizado para realizar análisis adicionales o generar nuevas predicciones basadas en los datos de entrada.

```
fit <- readRDS("../output/2020/modelos/fit_mrp_ictpc.rds")
```

proceso de benchmarking

En esta sección se realiza el proceso de benchmarking utilizando la función `benchmarking`, aplicada al modelo previamente ajustado. La función evalúa el rendimiento del modelo y compara los resultados basados en las variables seleccionadas. Se especifican las variables que se consideran en el análisis: `ent`, `area`, `sexo`, `edad`, `discapacidad`, y `hlengua`. El método de benchmarking utilizado es `logit`, que se emplea para analizar el ajuste del modelo en función de estas variables.

El resultado del proceso de benchmarking se guarda en un archivo RDS (`list_bench.rds`), que contiene las evaluaciones del modelo. Aunque el código para guardar y cargar este archivo está comentado, se proporciona la estructura para guardar los resultados y posteriormente cargarlos si es necesario.

```
list_bench <- benchmarking(modelo = fit,
             names_cov =   c("ent",
                             "area",
                             "sexo",
                             "edad",
                             "discapacidad",
                             "hlengua"),
             metodo = "logit")

# saveRDS(list_bench,"output/2020/plots/ictpc/list_bench.rds")
# list_bench <- readRDS("output/2020/plots/ictpc/list_bench.rds")
```

Validaciones benchmarking

En esta sección se llevan a cabo validaciones del modelo y se comparan los resultados con los datos de la encuesta. Primero, se prepara un DataFrame `poststrat_df` que incluye las predicciones del modelo (`yk_lmer`) y un ajuste de benchmarking (`yk_bench`). Esto permite comparar las estimaciones del modelo con los datos observados.

Luego, se crea un objeto de diseño de encuesta `diseno_encuesta` utilizando la función `as_survey_design` para aplicar los pesos de encuesta a los datos. Se calculan las medias de las predicciones del modelo y de los datos directos para verificar la consistencia entre ambos.

Para la validación nacional, se comparan las estimaciones nacionales obtenidas de la encuesta y del modelo. Esto se realiza mediante la comparación de las medias estimadas a nivel nacional del diseño de encuesta (`Nacional_dir`), el modelo ajustado (`Nacional_lmer`), y el benchmarking (`Nacional_bench`). Los resultados se resumen en un `cbind` para facilitar la comparación.

Este proceso de validación permite evaluar la precisión del modelo y asegurar que sus estimaciones sean consistentes con los datos observados y con los ajustes de benchmarking.

```
poststrat_df <- fit$poststrat_df %>% data.frame() %>%
  mutate(yk_lmer = yk,
         gk_bench = list_bench$gk_bench,
         yk_bench = yk * gk_bench )

diseno_encuesta <- fit$encuesta_mrp %>%
  mutate(yk_dir = yk) %>%
  as_survey_design(weights = fep)

mean(predict(fit$fit_mrp,type = "response"))
```

```r
mean(diseno_encuesta$variables$yk)

## validación nacional.
cbind(
  diseno_encuesta %>% summarise(Nacional_dir = survey_mean(yk_dir)) %>%
    select(Nacional_dir),
  poststrat_df %>% summarise(
    Nacional_lmer = sum(n * yk_lmer) / sum(n),
    Nacional_bench = sum(n * yk_bench) / sum(n*gk_bench),
  )) %>% print()
```

Validaciones por subgrupo completo

En esta sección se lleva a cabo la validación del modelo a nivel de subgrupos, comparando las estimaciones del modelo con los datos directos para cada subgrupo definido.

Se definen los subgrupos de interés en el vector `subgrupo`, que incluye variables como `ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`, y `nivel_edu`. Se generan gráficos y tablas de validación para cada uno de estos subgrupos utilizando la función `plot_uni_validacion`.

Cada gráfico y tabla permite comparar las estimaciones del modelo (`stan_lmer`) con los datos directos (`directo`) y calcular el error relativo (ER), que muestra la diferencia porcentual entre las dos estimaciones. La tabla de validación para el subgrupo `sexo` muestra estas diferencias de manera detallada, mientras que los gráficos combinados permiten una visualización comparativa entre los diferentes subgrupos.

Finalmente, se combinan los gráficos de validación en un solo objeto `plot_uni` utilizando la función `patchwork`. El gráfico combinado se guarda como una imagen JPG en el directorio de salida, y los objetos `plot_subgrupo` y `poststrat_df` se almacenan en archivos RDS para su uso futuro. Esta sección asegura que las estimaciones del modelo se validen de manera exhaustiva para cada subgrupo y se registren adecuadamente para futuras referencias.

```r
subgrupo <- c("ent",
              "area",
              "sexo",
              "edad",
              "discapacidad",
              "hlengua","nivel_edu")


plot_subgrupo <- map(
```

```
  .x = setNames(subgrupo, subgrupo),
  ~ plot_uni_validacion(
    sample_diseno = diseno_encuesta,
    poststrat = poststrat_df,
    by1 = .x
  )
)


plot_subgrupo$sexo$tabla %>% arrange(desc(n_sample) ) %>%
  mutate(ER = abs((directo - stan_lmer)/directo )*100) %>%
  data.frame() %>% select(sexo:directo_upp,stan_lmer, ER)

plot_subgrupo$ent$gg_plot

plot_subgrupo$sexo$gg_plot + plot_subgrupo$nivel_edu$gg_plot +
  plot_subgrupo$edad$gg_plot + plot_subgrupo$hlengua$gg_plot +
  plot_subgrupo$area$gg_plot + plot_subgrupo$discapacidad$gg_plot

plot_uni <- plot_subgrupo$ent$gg_plot  /
  (
    plot_subgrupo$sexo$gg_plot + plot_subgrupo$nivel_edu$gg_plot +
      plot_subgrupo$edad$gg_plot + plot_subgrupo$hlengua$gg_plot +
      plot_subgrupo$area$gg_plot + plot_subgrupo$discapacidad$gg_plot
  )

ggsave(plot = plot_uni,
       filename = "../output/2020/modelos/plot_uni.jpg",
       scale = 3)

saveRDS(object =  plot_subgrupo,
        file = "../output/2020/modelos/plot_uni.rds")

saveRDS(object =  poststrat_df,
        file = "../input/2020/muestra_ampliada/poststrat_df_ictpc.rds")
```

10_PostBenchmarking_ic_ali_nc.R

Para la ejecución del presente archivo, debe abrir el archivo **10_PostBenchmarking_ic_ali_nc.R** disponible en la ruta *Rcodes/2020/10_PostBenchmarking_ic_ali_nc.R*.

Este script en R se centra en la validación y benchmarking de un modelo multinivel para predecir la carencia en alimentos nutritivos y de calidad (ic_ali_nc) utilizando datos de encuestas y censos. Comienza limpiando el entorno de trabajo y cargando librerías necesarias. Además, se leen funciones auxiliares desde archivos externos para la validación y el benchmarking. Luego, se carga el modelo preentrenado desde un archivo RDS.

El proceso de benchmarking se realiza con la función `benchmarking`, utilizando covariables específicas como `ent`, `area`, `sexo`, `edad`, `discapacidad` y `hlengua`, y aplicando el método "logit". Los resultados se almacenan en `list_bench`. A continuación, se crea un diseño de encuesta y se realizan validaciones a nivel nacional comparando los resultados del modelo ajustado (`yk_stan_lmer`) y del modelo ajustado con benchmarking (`yk_bench`). Estas validaciones se llevan a cabo mediante la comparación de medias ponderadas.

Para validar el modelo a nivel de subgrupos, se definen diversos subgrupos como `ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua` y `nivel_edu`. Se crean gráficos de validación para cada subgrupo utilizando la función `plot_uni_validacion`. Estos gráficos se combinan y se guardan en un archivo JPG (`plot_uni_ic_ali_nc.jpg`). Además, se guarda la lista de gráficos y el dataframe postestratificado en archivos RDS (`plot_uni_ic_ali_nc.rds` y `poststrat_df_ic_ali_nc.rds` respectivamente), lo que permite una fácil recuperación y análisis posterior.

Limpieza del Entorno y Carga de Bibliotecas

El entorno de R se limpia al eliminar todos los objetos actuales y se limpia la consola para asegurar que el análisis comience desde un estado limpio. A continuación, se cargan las bibliotecas necesarias para el análisis, que incluyen `scales`, `patchwork`, `srvyr`, `survey`, `haven`, `sampling`, y `tidyverse`. Estas bibliotecas proporcionan herramientas para la manipulación de datos, la creación de gráficos, el análisis de encuestas y la realización de muestreos.

```
rm(list =ls())
cat("\f")
###############################################################
# Loading required libraries ------------------------------------------
###############################################################

library(scales)
library(patchwork)
library(srvyr)
library(survey)
library(haven)
library(sampling)
library(tidyverse)
```

Lectura de Funciones

Se cargan dos archivos de funciones que serán utilizadas en el análisis: `Plot_validacion_bench.R` y `Benchmarking.R`. Estos archivos contienen funciones personalizadas para la validación y el análisis comparativo.

```
source("../source/Plot_validacion_bench.R", encoding = "UTF-8")
source("../source/Benchmarking.R", encoding = "UTF-8")
```

Lectura del Modelo

Se lee el modelo ajustado previamente, guardado en un archivo RDS (`fit_mrp_ictpc.rds`). Este modelo será utilizado para realizar análisis adicionales o generar nuevas predicciones basadas en los datos de entrada.

```
fit <- readRDS("../output/2020/modelos/fit_mrp_ic_ali_nc.rds")
```

proceso de benchmarking

En esta sección se realiza el proceso de benchmarking utilizando la función `benchmarking`, aplicada al modelo previamente ajustado. La función evalúa el rendimiento del modelo y compara los resultados basados en las variables seleccionadas. Se especifican las variables que se consideran en el análisis: `ent`, `area`, `sexo`, `edad`, `discapacidad`, y `hlengua`. El método de benchmarking utilizado es `logit`, que se emplea para analizar el ajuste del modelo en función de estas variables.

El resultado del proceso de benchmarking se guarda en un archivo RDS (`list_bench.rds`), que contiene las evaluaciones del modelo. Aunque el código para guardar y cargar este archivo está comentado, se proporciona la estructura para guardar los resultados y posteriormente cargarlos si es necesario.

```
list_bench <- benchmarking(modelo = fit,
              names_cov =   c("ent",
                              "area",
                              "sexo",
                              "edad",
                              "discapacidad",
                              "hlengua"),
              metodo = "logit")
```

Validaciones benchmarking

En esta sección se llevan a cabo validaciones del modelo y se comparan los resultados con los datos de la encuesta. Primero, se prepara un DataFrame `poststrat_df` que incluye las predicciones del modelo (`yk_lmer`) y un ajuste de benchmarking (`yk_bench`). Esto permite comparar las estimaciones del modelo con los datos observados.

Luego, se crea un objeto de diseño de encuesta `diseno_encuesta` utilizando la función `as_survey_design` para aplicar los pesos de encuesta a los datos. Se calculan las medias de las predicciones del modelo y de los datos directos para verificar la consistencia entre ambos.

Para la validación nacional, se comparan las estimaciones nacionales obtenidas de la encuesta y del modelo. Esto se realiza mediante la comparación de las medias estimadas a nivel nacional del diseño de encuesta (`Nacional_dir`), el modelo ajustado (`Nacional_lmer`), y el benchmarking (`Nacional_bench`). Los resultados se resumen en un `cbind` para facilitar la comparación.

Este proceso de validación permite evaluar la precisión del modelo y asegurar que sus estimaciones sean consistentes con los datos observados y con los ajustes de benchmarking.

```
poststrat_df <- fit$poststrat_df %>% data.frame() %>%
  mutate(yk_stan_lmer = yk,
         gk_bench = list_bench$gk_bench,
         yk_bench = yk * gk_bench )

diseno_encuesta <- fit$encuesta_mrp %>%
  mutate(yk_dir = yk) %>%
  as_survey_design(weights = fep)

mean(predict(fit$fit_mrp,type = "response"))
mean(diseno_encuesta$variables$yk)

## validación nacional.
```

```
cbind(
  diseno_encuesta %>% summarise(Nacional_dir = survey_mean(yk_dir)) %>%
    select(Nacional_dir),
  poststrat_df %>% summarise(
    Nacional_stan_lmer = sum(n * yk_stan_lmer) / sum(n),
    Nacional_bench = sum(n * yk_bench) / sum(n*gk_bench),
  )) %>% print()
```

Validaciones por subgrupo completo

En esta sección, se realiza una validación exhaustiva del modelo ajustado, evaluando su desempeño a nivel de diversos subgrupos. Los subgrupos se definen en el vector `subgrupo`, que incluye variables como `ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`, y `nivel_edu`.

La función `plot_uni_validacion` se utiliza para generar gráficos y tablas de validación para cada uno de estos subgrupos. Estos gráficos y tablas permiten comparar las estimaciones obtenidas mediante el modelo (`stan_lmer`) con los datos directos (`directo`) para cada subgrupo, calculando también el error relativo (ER), que refleja la diferencia porcentual entre las dos estimaciones.

Para el subgrupo `sexo`, se genera una tabla que muestra el error relativo calculado para cada categoría dentro del subgrupo, ordenado por el tamaño de la muestra (`n_sample`). Los gráficos generados para cada subgrupo se combinan en un único gráfico utilizando `patchwork`. Este gráfico combinado permite una visualización comparativa clara entre todos los subgrupos definidos.

Finalmente, el gráfico combinado se guarda como una imagen JPG en el directorio de salida, y los objetos `plot_subgrupo` y `poststrat_df` se almacenan en archivos RDS. Estos archivos proporcionan una referencia detallada de la validación del modelo a nivel de subgrupo, permitiendo revisiones y análisis adicionales si es necesario.

```
subgrupo <- c("ent",
              "area",
              "sexo",
              "edad",
              "discapacidad",
              "hlengua","nivel_edu")


plot_subgrupo <- map(
  .x = setNames(subgrupo, subgrupo),
  ~ plot_uni_validacion(
    sample_diseno = diseno_encuesta,
```

```
    poststrat = poststrat_df,
    by1 = .x
  )
)


plot_subgrupo$sexo$tabla %>% arrange(desc(n_sample) ) %>%
  mutate(ER = abs((directo - stan_lmer)/directo )*100) %>%
  data.frame() %>% select(sexo:directo_upp,stan_lmer, ER)

plot_subgrupo$ent$gg_plot

plot_subgrupo$sexo$gg_plot + plot_subgrupo$nivel_edu$gg_plot +
  plot_subgrupo$edad$gg_plot + plot_subgrupo$hlengua$gg_plot +
  plot_subgrupo$area$gg_plot + plot_subgrupo$discapacidad$gg_plot

plot_uni <- plot_subgrupo$ent$gg_plot  /
  (
    plot_subgrupo$sexo$gg_plot + plot_subgrupo$nivel_edu$gg_plot +
      plot_subgrupo$edad$gg_plot + plot_subgrupo$hlengua$gg_plot +
      plot_subgrupo$area$gg_plot + plot_subgrupo$discapacidad$gg_plot
  )

ggsave(plot = plot_uni,
       filename = "../output/2020/modelos/plot_uni_ic_ali_nc.jpg",
       scale = 3)

saveRDS(object =  plot_subgrupo,
        file = "../output/2020/modelos/plot_uni_ic_ali_nc.rds")

saveRDS(object =  poststrat_df,
        file = "../input/2020/muestra_ampliada/poststrat_df_ic_ali_nc.rds")
```

11_PostBenchmarking_ic_segsoc.r

Para la ejecución del presente análisis, se debe abrir el archivo **11_PostBenchmarking_ic_segsoc.R** disponible en la ruta *Rcodes/2020/11_PostBenchmarking_ic_segsoc.R*.

Este script en R se centra en el proceso de benchmarking y validación de un modelo multinivel previamente ajustado para lala carencia en por cobertura de seguridad social. Inicia con la limpieza del entorno de trabajo y la carga de librerías esenciales como `scales`, `patchwork`, `srvyr`, `survey`, `haven`, `sampling` y `tidyverse`. Luego, se cargan funciones adicionales desde archivos externos para la validación y el benchmarking del modelo.

El modelo ajustado se lee desde un archivo RDS (`fit_mrp_ic_segsoc.rds`) y se procede a la etapa de benchmarking utilizando la función benchmarking, aplicando el método logit sobre un conjunto de covariables específicas (`ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`). Los resultados del benchmarking se incorporan al dataframe de post-estratificación y se recalculan las predicciones ponderadas.

El script también realiza validaciones a nivel nacional y por subgrupos (`ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`, `nivel_edu`). Para cada subgrupo, se generan gráficos de validación univariada utilizando la función plot_uni_validacion, que compara las estimaciones directas y ajustadas del modelo. Los gráficos resultantes se combinan y se guardan en un archivo JPG, proporcionando una visualización completa del desempeño del modelo. Finalmente, tanto los gráficos de subgrupos como el dataframe de post-estratificación se guardan en archivos RDS para futuros análisis y referencia. #### Limpieza del Entorno y Carga de Bibliotecas{-}

El entorno de R se limpia al eliminar todos los objetos actuales y se limpia la consola para asegurar que el análisis comience desde un estado limpio. A continuación, se cargan las bibliotecas necesarias para el análisis, que incluyen `scales`, `patchwork`, `srvyr`, `survey`, `haven`, `sampling`, y `tidyverse`. Estas bibliotecas proporcionan herramientas para la manipulación de datos, la creación de gráficos, el análisis de encuestas y la realización de muestreos.

```
rm(list =ls())
cat("\f")
##############################################################
```

```
# Loading required libraries ---------------------------------------------
##############################################################

library(scales)
library(patchwork)
library(srvyr)
library(survey)
library(haven)
library(sampling)
library(tidyverse)
```

Lectura de Funciones

Se cargan dos archivos de funciones que serán utilizadas en el análisis: `Plot_validacion_bench.R` y `Benchmarking.R`. Estos archivos contienen funciones personalizadas para la validación y el análisis comparativo.

```
source("../source/Plot_validacion_bench.R", encoding = "UTF-8")
source("../source/Benchmarking.R", encoding = "UTF-8")
```

Lectura del Modelo

Se lee el modelo ajustado previamente, guardado en un archivo RDS (`fit_mrp_ictpc.rds`). Este modelo será utilizado para realizar análisis adicionales o generar nuevas predicciones basadas en los datos de entrada.

```
fit <- readRDS("../output/2020/modelos/fit_mrp_ic_segsoc.rds")
```

proceso de benchmarking

En esta sección se realiza el proceso de benchmarking utilizando la función `benchmarking`, aplicada al modelo previamente ajustado. La función evalúa el rendimiento del modelo y compara los resultados basados en las variables seleccionadas. Se especifican las variables que se consideran en el análisis: `ent`, `area`, `sexo`, `edad`, `discapacidad`, y `hlengua`. El método de benchmarking utilizado es `logit`, que se emplea para analizar el ajuste del modelo en función de estas variables.

El resultado del proceso de benchmarking se guarda en un archivo RDS (`list_bench.rds`), que contiene las evaluaciones del modelo. Aunque el código para guardar y cargar este archivo está comentado, se proporciona la estructura para guardar los resultados y posteriormente cargarlos si es necesario.

```
list_bench <- benchmarking(modelo = fit,
             names_cov =   c("ent",
```

```
                              "area",
                              "sexo",
                              "edad",
                              "discapacidad",
                              "hlengua"),
              metodo = "logit")
```

Validaciones benchmarking

En esta sección se llevan a cabo validaciones del modelo y se comparan los resultados con los datos de la encuesta. Primero, se prepara un DataFrame `poststrat_df` que incluye las predicciones del modelo (`yk_lmer`) y un ajuste de benchmarking (`yk_bench`). Esto permite comparar las estimaciones del modelo con los datos observados.

Luego, se crea un objeto de diseño de encuesta `diseno_encuesta` utilizando la función `as_survey_design` para aplicar los pesos de encuesta a los datos. Se calculan las medias de las predicciones del modelo y de los datos directos para verificar la consistencia entre ambos.

Para la validación nacional, se comparan las estimaciones nacionales obtenidas de la encuesta y del modelo. Esto se realiza mediante la comparación de las medias estimadas a nivel nacional del diseño de encuesta (`Nacional_dir`), el modelo ajustado (`Nacional_lmer`), y el benchmarking (`Nacional_bench`). Los resultados se resumen en un `cbind` para facilitar la comparación.

Este proceso de validación permite evaluar la precisión del modelo y asegurar que sus estimaciones sean consistentes con los datos observados y con los ajustes de benchmarking.

```
poststrat_df <- fit$poststrat_df %>% data.frame() %>%
  mutate(yk_stan_lmer = yk,
         gk_bench = list_bench$gk_bench,
         yk_bench = yk * gk_bench )

diseno_encuesta <- fit$encuesta_mrp %>%
  mutate(yk_dir = yk) %>%
  as_survey_design(weights = fep)

mean(predict(fit$fit_mrp,type = "response"))
mean(diseno_encuesta$variables$yk)

## validación nacional.
cbind(
  diseno_encuesta %>% summarise(Nacional_dir = survey_mean(yk_dir)) %>%
```

```
    select(Nacional_dir),
  poststrat_df %>% summarise(
    Nacional_stan_lmer = sum(n * yk_stan_lmer) / sum(n),
    Nacional_bench = sum(n * yk_bench) / sum(n*gk_bench),
  )) %>% print()
```

Validaciones por subgrupo completo

En esta sección, se realiza una validación exhaustiva del modelo ajustado, evaluando su desempeño a nivel de diversos subgrupos. Los subgrupos se definen en el vector `subgrupo`, que incluye variables como `ent`, `area`, `sexo`, `edad`, `discapacidad`, `hlengua`, y `nivel_edu`.

La función `plot_uni_validacion` se utiliza para generar gráficos y tablas de validación para cada uno de estos subgrupos. Estos gráficos y tablas permiten comparar las estimaciones obtenidas mediante el modelo (`stan_lmer`) con los datos directos (`directo`) para cada subgrupo, calculando también el error relativo (ER), que refleja la diferencia porcentual entre las dos estimaciones.

Para el subgrupo `sexo`, se genera una tabla que muestra el error relativo calculado para cada categoría dentro del subgrupo, ordenado por el tamaño de la muestra (`n_sample`). Los gráficos generados para cada subgrupo se combinan en un único gráfico utilizando `patchwork`. Este gráfico combinado permite una visualización comparativa clara entre todos los subgrupos definidos.

Finalmente, el gráfico combinado se guarda como una imagen JPG en el directorio de salida, y los objetos `plot_subgrupo` y `poststrat_df` se almacenan en archivos RDS. Estos archivos proporcionan una referencia detallada de la validación del modelo a nivel de subgrupo, permitiendo revisiones y análisis adicionales si es necesario.

```
subgrupo <- c("ent",
              "area",
              "sexo",
              "edad",
              "discapacidad",
              "hlengua","nivel_edu")


plot_subgrupo <- map(
  .x = setNames(subgrupo, subgrupo),
  ~ plot_uni_validacion(
    sample_diseno = diseno_encuesta,
    poststrat = poststrat_df,
    by1 = .x
```

```
  )
)



plot_subgrupo$sexo$tabla %>% arrange(desc(n_sample) ) %>%
  mutate(ER = abs((directo - stan_lmer)/directo )*100) %>%
  data.frame() %>% select(sexo:directo_upp,stan_lmer, ER)

plot_subgrupo$ent$gg_plot

plot_subgrupo$sexo$gg_plot + plot_subgrupo$nivel_edu$gg_plot +
  plot_subgrupo$edad$gg_plot + plot_subgrupo$hlengua$gg_plot +
  plot_subgrupo$area$gg_plot + plot_subgrupo$discapacidad$gg_plot

plot_uni <- plot_subgrupo$ent$gg_plot  /
  (
    plot_subgrupo$sexo$gg_plot + plot_subgrupo$nivel_edu$gg_plot +
      plot_subgrupo$edad$gg_plot + plot_subgrupo$hlengua$gg_plot +
      plot_subgrupo$area$gg_plot + plot_subgrupo$discapacidad$gg_plot
  )

ggsave(plot = plot_uni,
       filename = "../output/2020/modelos/plot_uni_ic_segsoc.jpg",
       scale = 3)

saveRDS(object =  plot_subgrupo,
        file = "../output/2020/modelos/plot_uni_ic_segsoc.rds")

saveRDS(object =  poststrat_df,
        file = "../input/2020/muestra_ampliada/poststrat_df_ic_segsoc.rds")
```

12_Imputacion_censal.R

Para la ejecución del presente análisis, se debe abrir el archivo **12_Imputacion_censal.R** disponible en la ruta **Rcodes/2020/12_Imputacion_censal.R**.

Este script en R se centra en el análisis y la predicción de datos utilizando modelos estadísticos ajustados a encuestas y censos. Primero, se limpia el entorno de trabajo y se cargan diversas librerías necesarias para el análisis. Luego, se leen modelos previamente ajustados (`fit_ingreso`, `fit_alimento`, `fit_salud`) y datos de encuestas y líneas de bienestar. Se realiza una combinación de datos entre las encuestas y predictores de nivel estatal, y se calculan predicciones basadas en estos modelos. Los resultados de estas predicciones se almacenan en un archivo RDS para su posterior análisis.

El script también incluye un paso para ajustar y validar los datos de la encuesta ampliada mediante simulaciones de variables predictoras y el cálculo de medidas de desviación estándar residual. Posteriormente, se lleva a cabo un proceso iterativo para generar nuevas predicciones y comparar estas predicciones con las originales. Finalmente, el script prepara los datos para un análisis posterior al actualizar las variables con valores simulados basados en las predicciones y desviaciones calculadas. Este proceso permite validar y ajustar los modelos utilizados, asegurando la precisión de las predicciones en la encuesta ampliada.

Limpieza del Entorno y Carga de Bibliotecas

Primero, se limpia el entorno de trabajo para evitar cualquier interferencia de datos o variables residuales. Esto se hace utilizando `rm(list = ls())`, que elimina todos los objetos del espacio de trabajo en R. Esta acción asegura que el análisis se realice en un entorno limpio y libre de residuos de sesiones anteriores.

Luego, se cargan las bibliotecas necesarias para el análisis. Se utiliza **`tidyverse`** para la manipulación y visualización de datos, que incluye paquetes como `ggplot2`, `dplyr`, y `tidyr`. **`data.table`** se usa para la manipulación eficiente de grandes conjuntos de datos. La biblioteca **`openxlsx`** permite leer y escribir archivos de Excel, mientras que **`magrittr`** proporciona el operador `%>%` para encadenar operaciones de manera fluida. Se carga **`haven`** para importar datos desde formatos de Stata, SPSS y SAS, y **`labelled`** para manejar variables con etiquetas, lo que facilita la interpretación de datos etiquetados.

sampling es útil para procedimientos de muestreo y selección de muestras, mientras que **lme4** se utiliza para ajustar modelos lineales y no lineales mixtos. Finalmente, **survey** y **srvyr** son empleadas para el análisis de datos de encuestas complejas y para manipular estos datos en el contexto de `dplyr`.

Para asegurar que la consola esté limpia y el entorno de trabajo esté listo para el análisis, se utiliza `cat("\f")`, que borra cualquier salida previa en la consola.

Finalmente, se cargan funciones personalizadas desde un archivo de script R utilizando `source("../source/benchmarking_indicador.R")`. Este paso garantiza que las funciones necesarias para el análisis de indicadores de benchmarking estén disponibles y listas para su uso.

```
rm(list = ls())
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(haven)
library(labelled)
library(sampling)
library(lme4)
library(survey)
library(srvyr)
cat("\f")
source("../source/benchmarking_indicador.R")
```

Carga de Modelos y Datos

En la sección se procede a leer los modelos ajustados previamente para los indicadores de ingresos, alimentación y salud. Esta etapa es crucial para el análisis posterior, ya que permite acceder a los modelos ya entrenados y preparados para realizar validaciones o nuevos cálculos.

Primero, se cargan los modelos ajustados para el indicador de ingresos. El archivo `fit_mrp_ictpc.rds` contiene el modelo para el Índice de Condiciones de Vida (ICTPC), y se extrae el objeto `fit_mrp` del archivo usando `readRDS`.

Luego, se realiza una operación similar para el modelo de alimentación. El archivo `fit_mrp_ic_ali_nc.rds` incluye el modelo correspondiente al Índice de Condiciones de Alimentación y Nutrición (IC-ALI-NC). De nuevo, se extrae el objeto `fit_mrp` del archivo cargado.

Finalmente, se carga el modelo para el indicador de seguridad social, que se encuentra en el archivo `fit_mrp_ic_segsoc.rds`. Se sigue el mismo proceso de extracción del objeto `fit_mrp`.

```
fit_ingreso <- readRDS("../output/2020/modelos/fit_mrp_ictpc.rds")$fit_mrp
fit_alimento <- readRDS("../output/2020/modelos/fit_mrp_ic_ali_nc.rds")$fit_mrp
fit_salud <- readRDS("../output/2020/modelos/fit_mrp_ic_segsoc.rds")$fit_mrp
```

Carga de Datos

En esta sección, se cargan los datos esenciales necesarios para el análisis de bienestar y pobreza. Primero, se lee la encuesta `ENIGH` del archivo `encuesta_sta.rds`, la cual se actualiza con una columna adicional llamada `ingreso`, que se define a partir del índice de condiciones de vida (ICTPC).

A continuación, se importan las líneas de bienestar desde un archivo CSV llamado `Lineas_Bienestar.csv`. Este archivo contiene información relevante sobre las líneas de bienestar, y los datos se procesan para asegurarse de que la columna `area` sea de tipo carácter.

Se procede a cargar los predictores a nivel estatal desde el archivo `statelevel_predictors_df.rds`, que incluye variables a nivel de estado que se usarán en el análisis.

Finalmente, se lee una muestra ampliada del censo desde el archivo `encuesta_ampliada.rds`. Este conjunto de datos se une con los predictores estatales y las líneas de bienestar para enriquecer el dataset y asegurar que toda la información relevante esté disponible para el análisis.

```
encuesta_enigh <- readRDS("../input/2020/enigh/encuesta_sta.rds") %>%
  mutate(ingreso = ictpc)

# lineas de bienestar

LB <- read.delim(
  "input/2020/Lineas_Bienestar.csv",
  header = TRUE,
  sep = ";",
  dec = ","
) %>% mutate(area = as.character(area))

statelevel_predictors <- readRDS("../input/2020/predictores/statelevel_predictors_df.r

############################################################
# Lectura del muestras intercensal o  muestra_ampliada del censo
############################################################

muestra_ampliada <- readRDS("../output/2020/encuesta_ampliada.rds")
col_names_muestra <- names(muestra_ampliada)
```

```
muestra_ampliada <-  inner_join(muestra_ampliada, statelevel_predictors)
muestra_ampliada <- muestra_ampliada %>% inner_join(LB)
```

Predicción y Evaluación

Se realizan predicciones utilizando los modelos ajustados previamente para evaluar diferentes indicadores de bienestar social. Primero, se emplea el modelo para ingresos, cargado previamente en `fit_ingreso`, para generar predicciones sobre la muestra ampliada. Estas predicciones se obtienen mediante la función `predict`, especificando el nuevo conjunto de datos (`muestra_ampliada`) y el tipo de respuesta esperado.

De manera similar, se aplican los modelos para los indicadores de alimentación (`fit_alimento`) y para seguridad social (`fit_salud`) en la misma muestra ampliada. Al usar `allow.new.levels = TRUE`, se asegura que las predicciones puedan realizarse incluso para niveles no presentes en los datos de entrenamiento, lo que permite una evaluación más completa y precisa sobre el conjunto de datos actual.

```
pred_ingreso <- predict(fit_ingreso,
                        newdata = muestra_ampliada,
                        allow.new.levels = TRUE,
                        type = "response")

pred_ic_ali_nc <- predict(fit_alimento,
                        newdata = muestra_ampliada,
                        allow.new.levels = TRUE,
                        type = "response")

pred_segsoc <- predict(fit_salud,
                        newdata = muestra_ampliada,
                       allow.new.levels = TRUE,
                       type = "response")
```

Cálculo de la Desviación Estándar Residual para el Modelo de Ingreso

Se calcula la desviación estándar residual para el modelo de ingresos ajustado para evaluar la precisión de las predicciones. Primero, se agrega la variable de predicción `pred_ingreso` al conjunto de datos `encuesta_enigh`. Luego, se define un diseño de encuesta utilizando la función `svydesign` del paquete `survey`, que incluye información sobre el diseño muestral, como los identificadores de unidades primarias de muestreo (UPM), los estratos y los pesos.

A continuación, se agrupan los datos por el nivel municipal (`cve_mun`) para calcular la media observada y la media predicha de los ingresos. La desviación estándar residual se obtiene al calcular la raíz cuadrada de la media de las diferencias

cuadráticas entre las medias observadas y las predicciones. Finalmente, se compara la desviación estándar calculada con la desviación estándar del error residual del modelo (`sigma(fit_ingreso)`), y se toma el valor mínimo de ambos para una evaluación más robusta.

```
paso <- encuesta_enigh %>% mutate(pred_ingreso = pred_ingreso) %>%
  survey::svydesign(
    id =  ~ upm ,
    strata = ~ estrato ,
    data = .,
    weights = ~ fep
  ) %>%
  as_survey_design()

sd_1 <- paso %>% group_by(cve_mun) %>%
  summarise(media_obs = survey_mean(ingreso),
            media_pred = survey_mean(pred_ingreso)) %>%
  summarise(media_sd = sqrt(mean(c(
    media_obs - media_pred
  ) ^ 2)))

desv_estandar_residual <-
  min(c(as.numeric(sd_1), sigma(fit_ingreso)))
```

Cálculo del Índice de Pobreza Multidimensional (IPM)

Para calcular el Índice de Pobreza Multidimensional (IPM), se utiliza una combinación de las predicciones de los modelos y los datos originales. Primero, se realiza una unión con los datos de las líneas de bienestar (`LB`). Luego, se crea una nueva variable llamada `tol_ic` que suma los indicadores de carencia.

Con la variable `tol_ic` calculada, se define el índice IPM mediante la función `case_when`. Esta función clasifica a la población en cuatro categorías:

1. **Situación de pobreza (Categoría I)**: Se asigna a aquellos cuyo ingreso es menor que el umbral de pobreza (`lp`) y tienen al menos una carencia social (`tol_ic >= 1`).
2. **Vulnerabilidad por carencias sociales (Categoría II)**: Se asigna a quienes tienen un ingreso igual o superior al umbral de pobreza pero presentan al menos una carencia social.
3. **Vulnerabilidad por ingresos (Categoría III)**: Se asigna a quienes tienen un ingreso menor o igual al umbral de pobreza pero no presentan carencias sociales.
4. **No pobre multidimensional y no vulnerable (Categoría IV)**: Se asigna a aquellos con ingresos igual o superior al umbral de pobreza y sin carencias

sociales.

```
encuesta_enigh <- encuesta_enigh %>% inner_join(LB) %>%
   mutate(
   tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv +  ic_sbv + ic_rezedu,
   ipm    = case_when(
     # Población en situación de pobreza.
     ingreso < lp & tol_ic >= 1 ~ "I",
     # Población vulnerable por carencias sociales.
     ingreso >= lp & tol_ic >= 1 ~ "II",
     # Poblacion vulnerable por ingresos.
     ingreso <= lp & tol_ic < 1 ~ "III",
     # Población no pobre multidimensional y no vulnerable.
     ingreso >= lp & tol_ic < 1 ~ "IV"
     )
   )
```

Predicción Final y Almacenamiento:

Para completar el análisis, se realiza una predicción final utilizando los modelos ajustados sobre la muestra ampliada. Esto incluye la predicción para los modelos de ingreso, alimentación y seguridad social (`pred_ingreso`, `pred_ic_ali_nc`, y `pred_segsoc`, respectivamente).

Además, se calcula la desviación estándar residual (`desv_estandar_residual`) para el modelo de ingreso, que se usa para ajustar y validar el modelo.

Finalmente, se guarda un archivo en formato RDS con todas las predicciones y la desviación estándar residual calculada. Este archivo se almacena en la ruta especificada: `"../output/2020/modelos/predicciones.rds"`. Este archivo facilitará el acceso a los resultados para futuras etapas del análisis o para reportes posteriores.

```
saveRDS(list(pred_ingreso = pred_ingreso,
          pred_ic_ali_nc = pred_ic_ali_nc,
          pred_segsoc = pred_segsoc,
          desv_estandar_residual = desv_estandar_residual),
     file =  "../output/2020/modelos/predicciones.rds")
```

Ajuste y Validación de las Predicciones en la Muestra Ampliada

Una vez realizadas las predicciones con los modelos ajustados, se procede a ajustar los valores en la muestra ampliada utilizando las predicciones obtenidas. Para cada observación en la muestra ampliada, se ajustan las variables relacionadas con las carencias y el ingreso:

1. **Ajuste de las Carencias**: Las variables de carencia (`ic_segsoc` e `ic_ali_nc`) se generan utilizando la función `rbinom`, que permite modelar estas variables como variables binomiales con probabilidades dadas por las predicciones (`pred_segsoc` y `pred_ic_ali_nc`).
2. **Ajuste del Ingreso**: El ingreso se ajusta sumando a las predicciones de ingreso (`pred_ingreso`) un término de error normalmente distribuido con media cero y desviación estándar residual (`desv_estandar_residual`).
3. **Cálculo de la Variable `tol_ic`**: Se calcula la suma de todas las carencias para obtener `tol_ic`.

Finalmente, se valida la precisión de las predicciones comparando las medias de las variables ajustadas (`muestra_ampliada_pred`) con las medias de las predicciones originales (`pred_ic_ali_nc`, `pred_segsoc`, y `pred_ingreso`). La diferencia entre estas medias proporciona una indicación de la precisión de las predicciones.

Este proceso asegura que las predicciones realizadas se ajusten adecuadamente a la muestra ampliada y permite evaluar la exactitud de las predicciones en el contexto de la muestra.

```
muestra_ampliada_pred <-
  muestra_ampliada %>%
  select(all_of(col_names_muestra), li, lp) %>%
  mutate(
    ic_segsoc = rbinom(n = n(), size = 1, prob = pred_segsoc),
    ic_ali_nc = rbinom(n = n(), size = 1, prob = pred_ic_ali_nc),
    ingreso = pred_ingreso + rnorm(n = n(), mean = 0, desv_estandar_residual),
    tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv +
      ic_sbv + ic_rezedu
  )
# Validación de los predict
mean(muestra_ampliada_pred$ic_ali_nc) - mean(pred_ic_ali_nc)
mean(muestra_ampliada_pred$ic_segsoc) - mean(pred_segsoc)
mean(muestra_ampliada_pred$ingreso) - mean(pred_ingreso)
```

13_Estimacion_ent_02.R

Para la ejecución del presente archivo, debe abrir el archivo **13_Estimacion_ent_02.R** disponible en la ruta *Rcodes/2020/13_Estimacion_ent_02.R*.

Inicialización y Carga de Librerías

En esta primera sección, se limpia el entorno de trabajo eliminando todas las variables y objetos existentes mediante `rm(list = ls())`. A continuación, se cargan diversas librerías necesarias para el análisis de datos y modelado, como `patchwork` para la visualización, `lme4` para modelos lineales mixtos, `tidyverse` para manipulación de datos, y otras librerías específicas para análisis de encuestas y predicción. También se incluye un archivo externo de funciones llamado `modelos_freq.R`.

```
rm(list = ls())
library(patchwork)
library(nortest)
library(lme4)
library(tidyverse)
library(magrittr)
library(caret)
library(car)
library(survey)
library(srvyr)
source("../source/modelos_freq.R")
```

Configuración del Nivel de Agregación y Carga de Datos

Aquí se define un vector `byAgrega` que especifica los niveles de agregación para el análisis, como entidad, municipio, área, y variables demográficas.

```
byAgrega <-
  c("ent",
    "cve_mun",
    "area",
    "sexo",
```

```
    "edad",
    "discapacidad",
    "hlengua",
    "nivel_edu" )

memory.limit(10000000)
```

Esta sección carga y filtra los datos necesarios desde archivos RDS y CSV. Se establece un límite de memoria y se cargan varios conjuntos de datos relevantes para el análisis, como la encuesta `enigh`, datos del censo, y predictores a nivel estatal. También se carga un archivo de líneas de bienestar y se actualizan los nombres de las variables para el análisis.

```
encuesta_enigh <- readRDS("../input/2020/enigh/encuesta_sta.rds") %>%
  filter(ent == "02", ictpc <= 10000)
censo_sta <- readRDS("../input/2020/muestra_ampliada/muestra_cuestionario_ampliado.rds
  filter(ent == "02")
statelevel_predictors_df <- readRDS("../input/2020/predictores/statelevel_predictors_d
  filter(ent == "02")
muestra_ampliada <- readRDS("output/2020/encuesta_ampliada.rds") %>%
  filter(ent == "02")
LB <-
  read.delim(
    "../input/2020/Lineas_Bienestar.csv",
    header = TRUE,
    sep = ";",
    dec = ","
  ) %>% mutate(area = as.character(area))
cov_names <- names(statelevel_predictors_df)
cov_names <- cov_names[!grepl(x = cov_names, pattern = "^hog_|cve_mun")]
```

Preparación de Datos y Modelado

Aquí se preparan los datos combinando diferentes conjuntos mediante `inner_join` para integrar información de predictores estatales y líneas de bienestar con la muestra ampliada.

```
col_names_muestra <- names(muestra_ampliada)
muestra_ampliada <-  inner_join(muestra_ampliada, statelevel_predictors_df)
muestra_ampliada <- muestra_ampliada %>% inner_join(LB)
```

Modelado del Ingreso

Se seleccionan y preparan las variables para modelar el ingreso. Se define la fórmula del modelo que incluye efectos aleatorios y fijos, y se ajusta un modelo usando la función `modelo_ingreso`. Los resultados del modelo se guardan en `fit_ingreso`.

```
variables_seleccionadas <-
  c(
    "prom_esc_rel_urb",
    "smg1",
    "gini15m",
    "ictpc15",
    "altitud1000",
    "prom_esc_rel_rur",
    "porc_patnoagrnocal_urb",
    "acc_medio",
    "derhab_pea_15_20",
    "porc_norep_ing_urb"
  )
cov_names <- c(
  "modifica_humana", "acceso_hosp",
  "acceso_hosp_caminando", "cubrimiento_cultivo",
  "cubrimiento_urbano", "luces_nocturnas",
  variables_seleccionadas
)
cov_registros <- setdiff(
  cov_names,
  c(
    "elec_mun20",
    "elec_mun19",
    "transf_gobpc_15_20",
    "derhab_pea_15_20",
    "vabpc_15_19",
    "itlpis_15_20",
    "remespc_15_20",
    "desem_15_20",
    "smg1",
    "ql_porc_cpa_urb",
    "ql_porc_cpa_rur"
  )
)
cov_registros <- paste0(cov_registros, collapse = " + ")
formula_model <-
  paste0("ingreso ~ (1 | cve_mun) + (1 | hlengua) + (1 | discapacidad)  + nivel_edu +
```

```
      " + ", cov_registros)
encuesta_enigh$ingreso <- (as.numeric(encuesta_enigh$ictpc))
fit <- modelo_ingreso(
  encuesta_sta = encuesta_enigh,
  predictors = statelevel_predictors_df,
  censo_sta = censo_sta,
  formula_mod = formula_model,
  byAgrega = byAgrega
)
fit_ingreso <- fit$fit_mrp
```

Modelado de Alimentos de Calidad y Seguridad Social

Esta sección es similar a la anterior, pero se enfoca en modelar la calidad de alimentos y seguridad social. Se ajustan modelos para alimentos de calidad y seguridad social, con fórmulas adecuadas y utilizando la función `modelo_dummy`.

```
variables_seleccionadas <- c(
  "porc_ing_ilpi_urb",
  "pob_ind_rur",
  "pob_ind_urb",
  "porc_hogremesas_rur",
  "porc_segsoc15",
  "porc_ali15",
  "plp15",
  "pob_rur",
  "altitud1000"
)

cov_names <- c(
  "modifica_humana", "acceso_hosp",
  "acceso_hosp_caminando", "cubrimiento_cultivo",
  "cubrimiento_urbano", "luces_nocturnas",
  variables_seleccionadas
)

cov_registros <- setdiff(
  cov_names,
  c(
    "elec_mun20",
    "elec_mun19",
    "transf_gobpc_15_20",
    "derhab_pea_15_20",
```

```
    "vabpc_15_19",
    "itlpis_15_20",
    "remespc_15_20",
    "desem_15_20",
    "porc_urb",
    "edad65mas_urb",
    "pob_tot",
    "acc_muyalto",
    "smg1",
    "ql_porc_cpa_rur",
    "ql_porc_cpa_urb"
  )
)
cov_registros <- paste0(cov_registros, collapse = " + ")
formula_model <-
  paste0(
    "cbind(si, no) ~ (1 | cve_mun) + (1 | hlengua) + (1 | discapacidad) + nivel_edu +
    " + ",
    cov_registros
  )
fit <- modelo_dummy(
  encuesta_sta = encuesta_enigh %>%
    mutate(yk = ifelse(ic_ali_nc == 1 , 1, 0)),
  predictors = statelevel_predictors_df,
  censo_sta = censo_sta,
  formula_mod = formula_model,
  byAgrega = byAgrega
)
fit_alimento <- fit$fit_mrp
```

```
################################################################################
# modelo para seguridad social  #
################################################################################

variables_seleccionadas <-
  c(
    "porc_rur",
    "porc_urb",
    "porc_ing_ilpi_rur",
    "porc_ing_ilpi_urb",
    "porc_jub_urb",
    "porc_segsoc15",
    "plp15",
```

```
    "ictpc15",
    "pob_urb",
    "pob_tot"
  )
cov_names <- c(
  "modifica_humana",
  "acceso_hosp",
  "acceso_hosp_caminando",
  "cubrimiento_cultivo",
  "cubrimiento_urbano",
  "luces_nocturnas" ,
  variables_seleccionadas
)

cov_registros <-
  setdiff(
    cov_names,
    c(
      "elec_mun20",
      "elec_mun19",
      "transf_gobpc_15_20",
      "derhab_pea_15_20",
      "vabpc_15_19" ,
      "itlpis_15_20"  ,
      "remespc_15_20",
      "desem_15_20",
      "porc_urb" ,
      "edad65mas_urb",
      "pob_tot" ,
      "acc_muyalto" ,
      "smg1",
      "ql_porc_cpa_rur",
      "ql_porc_cpa_urb"
    )
  )


cov_registros <- paste0(cov_registros, collapse = " + ")

formula_model <-
  paste0(
    "cbind(si, no) ~ (1 | cve_mun) + (1 | hlengua) + (1 | discapacidad) +  nivel_edu +
```

```
    ,
    " + ",
    cov_registros
  )




fit <- modelo_dummy(
  encuesta_sta = encuesta_enigh  %>%
    mutate(yk = ifelse(ic_segsoc == 1 ,1,0))  ,
  predictors = statelevel_predictors_df,
  censo_sta = censo_sta ,
  formula_mod = formula_model,
  byAgrega = byAgrega
)

fit_segsoc <- fit$fit_mrp
```

Predicción y Validación

En esta parte, se realiza la predicción del ingreso utilizando el modelo ajustado y se evalúa la precisión de las predicciones. Se calcula el error estándar residual y se ajustan los valores en función de las predicciones realizadas.

```
encuesta_sta <- inner_join(encuesta_enigh, statelevel_predictors_df)
pred_ingreso <- (predict(fit_ingreso, newdata = encuesta_sta))
sum(pred_ingreso < 3560)
rm(encuesta_sta)
paso <- encuesta_enigh %>% mutate(pred_ingreso = pred_ingreso) %>%
  survey::svydesign(id =~ upm , strata = ~ estrato , data = ., weights = ~ fep) %>%
  as_survey_design()
sd_1 <- paso %>% group_by(cve_mun) %>%
  summarise(media_obs = survey_mean(ingreso),
            media_pred = survey_mean(pred_ingreso)) %>%
  summarise(media_sd = sqrt(mean(c(media_obs - media_pred)^2)))
desv_estandar_residual <- min(c(as.numeric(sd_1), sigma(fit_ingreso)))
```

Cálculo del Índice de Pobreza Multidimensional (IPM)

Finalmente, se calcula el índice de pobreza multidimensional (IPM) basado en varias dimensiones, se ajusta la encuesta con estos datos, y se calcula la media del IPM.

```
encuesta_enigh <-
  encuesta_enigh %>% inner_join(LB)  %>%
  mutate(
    tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv +  ic_sbv + ic_rezedu,
    ipm   = case_when(
      # Población en situación de pobreza.
      ingreso < lp  &  tol_ic >= 1 ~ "I",
      # Población vulnerable por carencias sociales.
      ingreso >= lp & tol_ic >= 1 ~ "II",
      # Poblacion vulnerable por ingresos.
      ingreso <= lp & tol_ic < 1 ~ "III",
      # Población no pobre multidimensional y no vulnerable.
      ingreso >= lp & tol_ic < 1 ~ "IV"
    ),
    # Población en situación de pobreza moderada.
    pobre_moderada = ifelse(c(ingreso > li & ingreso < lp) &
                              tol_ic > 2, 1, 0),
    pobre_extrema = ifelse(ipm == "I" & pobre_moderada == 0, 1, 0)

  )
```

Predicción con los Modelos Ajustados

Se generan predicciones para las variables de interés (`ingreso`, `ic_ali_nc`, `ic_segsoc`) usando los modelos ajustados (`fit_ingreso`, `fit_alimento`, `fit_segsoc`) y el conjunto de datos `muestra_ampliada`. Se permite el uso de nuevos niveles en las predicciones.

```
pred_ingreso <- predict(fit_ingreso,
                        newdata = muestra_ampliada ,
                        allow.new.levels = TRUE,
                        type = "response")

pred_ic_ali_nc <- predict(fit_alimento,
                          newdata = muestra_ampliada ,
                          allow.new.levels = TRUE,
                          type = "response")

pred_segsoc <- predict(fit_segsoc,
                       newdata = muestra_ampliada ,
                       allow.new.levels = TRUE,
                       type = "response")
```

Creación de Variables Dummy y Preparación de Datos

Aquí, se crean variables dummy para indicadores de salud, seguridad social, y otras características. Luego, se preparan datos adicionales para `muestra_ampliada_pred`, generando variables simuladas (`ic_segsoc`, `ic_ali_nc`) y ajustando el ingreso con ruido aleatorio. Se calcula el total del índice de carencias (`tol_ic`).

```
muestra_ampliada %<>% mutate(
  ic_asalud = ifelse(ic_asalud == 1, 1,0),
  ic_cv = ifelse(ic_cv == 1, 1,0),
  ic_sbv = ifelse(ic_sbv == 1, 1,0),
  ic_rezedu = ifelse(ic_rezedu == 1, 1,0))

muestra_ampliada_pred <-
  muestra_ampliada %>%
  select(all_of(col_names_muestra), li, lp) %>%
  mutate(
    ic_segsoc = rbinom(n = n(), size = 1, prob = pred_segsoc),
    ic_ali_nc = rbinom(n = n(), size = 1, prob = pred_ic_ali_nc),
    ingreso = pred_ingreso + rnorm(n = n(), mean = 0, desv_estandar_residual),
    tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv +
      ic_sbv + ic_rezedu)
```

Validación de las Predicciones

Se validan las predicciones comparando las medias de las variables simuladas con las predicciones generadas por los modelos. Se calcula la diferencia entre estas medias para verificar la precisión de las predicciones.

```
mean(muestra_ampliada_pred$ic_ali_nc) -
  mean(pred_ic_ali_nc)

mean(muestra_ampliada_pred$ic_segsoc) -
  mean(pred_segsoc)

mean(muestra_ampliada_pred$ingreso) -
  mean(pred_ingreso)
```

Cálculo del Índice de Pobreza Multidimensional (IPM) con Predicciones

Se clasifica la población en diferentes categorías de pobreza multidimensional (`ipm`) y se identifican las personas en pobreza moderada y extrema. Se calcula la proporción de cada categoría en `muestra_ampliada_pred`.

```
muestra_ampliada_pred <- muestra_ampliada_pred %>% mutate(
  ipm    = case_when(
    ingreso < lp  &  tol_ic >= 1 ~ "I",
    ingreso >= lp & tol_ic >= 1 ~ "II",
    ingreso <= lp & tol_ic < 1 ~ "III",
    ingreso >= lp & tol_ic < 1 ~ "IV"
  ),
  pobre_moderada = ifelse(c(ingreso > li & ingreso < lp) &
                            tol_ic > 2, 1, 0),
  pobre_extrema = ifelse(ipm == "I" & pobre_moderada == 0, 1, 0) )
prop.table(table(muestra_ampliada_pred$ipm))
```

Estimación y Comparación del IPM con Predicciones

Se estima el IPM para las predicciones y se compara con la estimación real basada en `encuesta_enigh`. Se calculan las proporciones de cada categoría IPM para ambas estimaciones.

```
muestra_ampliada_pred %>% filter() %>% group_by(ipm) %>%
  summarise(num_ipm =sum(factor)) %>%
  mutate(est_ipm = num_ipm/sum(num_ipm))

encuesta_enigh %>% group_by(ipm) %>%
  summarise(num_ipm =sum(fep)) %>%
  mutate(est_ipm = num_ipm/sum(num_ipm))
```

Ajuste de las Predicciones y Calibración

Se actualizan las variables de `muestra_ampliada_pred` con información adicional y se elimina cualquier dato faltante en `encuesta_enigh`.

```
muestra_ampliada_pred   %<>% mutate(
  tol_ic4 = ic_asalud + ic_cv +  ic_sbv + ic_rezedu,
  pred_segsoc = pred_segsoc,
  pred_ingreso = pred_ingreso,
  desv_estandar_residual = desv_estandar_residual,
  pred_ic_ali_nc = pred_ic_ali_nc
)

ii_ent = "02"
encuesta_enigh %<>% na.omit()
```

Cálculo de la Población y Pobreza por Municipio

Se calcula la densidad de población por municipio y se determina el porcentaje de población en cada categoría de IPM. Se estima la pobreza moderada y extrema en función de las encuestas y la población total.

```
total_mpio  <- muestra_ampliada_pred %>% group_by(cve_mun) %>%
  summarise(den_mpio = sum(factor),.groups = "drop") %>%
  mutate(tot_ent = sum(den_mpio))

tot_pob <- encuesta_enigh %>%
  group_by(ipm) %>%
  summarise(num_ent = sum(fep),.groups = "drop") %>%
  transmute(ipm,   prop_ipm = num_ent/sum(num_ent),
            tx_ipm =  sum(total_mpio$den_mpio)*prop_ipm) %>%
  filter(ipm != "I")

pobreza <-  encuesta_enigh %>% summarise(
  pobre_ext = weighted.mean(pobre_extrema, fep),
  pob_mod = weighted.mean(pobre_moderada, fep),
  pobre_moderada = sum(total_mpio$den_mpio)*pob_mod,
  pobre_extrema = sum(total_mpio$den_mpio)*pobre_ext)
```

Iteraciones de Calibración

Este bloque realiza iteraciones para ajustar y calibrar los datos, actualizando las predicciones en cada iteración y comparando los resultados con la población real. Se calculan y guardan los resultados en archivos `.rds` para cada iteración.

```
for( iter in 1:200){
  cat("\n iteracion = ", iter,"\n\n")

  muestra_ampliada_pred %<>%
    mutate(
      ic_segsoc = rbinom(n = n(), size = 1, prob = pred_segsoc),
      ic_ali_nc = rbinom(n = n(), size = 1, prob = pred_ic_ali_nc),
      ingreso = pred_ingreso + rnorm(n = n(),mean = 0,
                                     desv_estandar_residual),
      tol_ic = tol_ic4 + ic_segsoc + ic_ali_nc
    ) %>% mutate(
      ipm   = case_when(
        ingreso < lp  &  tol_ic >= 1 ~ "I",
        ingreso >= lp & tol_ic >= 1 ~ "II",
        ingreso <= lp & tol_ic < 1 ~ "III",
```

```
        ingreso >= lp & tol_ic < 1 ~ "IV"
      ),
      pobre_moderada = ifelse(c(ingreso > li & ingreso < lp) &
                                tol_ic > 2, 1, 0),
      pobre_extrema = ifelse(ipm == "I" & pobre_moderada == 0, 1, 0)
    )

  Xk <- muestra_ampliada_pred %>% select("ipm", "cve_mun") %>%
    fastDummies::dummy_columns(select_columns = c("ipm", "cve_mun")) %>%
    select(names(Tx_hat[-c(1:2)]))

  diseno_post <- bind_cols(muestra_ampliada_pred,Xk) %>%
    mutate(fep = factor) %>%
    as_survey_design(
      ids = upm,
      weights = fep,
      nest = TRUE,
      # strata = estrato
    )

  mod_calib <-  as.formula(paste0("~ -1+",paste0(names(Tx_hat), collapse = " + ")))

  diseno_calib <- calibrate(diseno_post, formula = mod_calib,
                            population = Tx_hat,calfun = "raking")

  estima_calib_ipm  <- diseno_calib %>% group_by(cve_mun,ipm) %>%
    summarise(est_ipm = survey_mean(vartype ="var"))

  estima_calib_pob  <- diseno_calib %>% group_by(cve_mun) %>%
    summarise(est_pob_ext = survey_mean(pobre_extrema , vartype ="var"),
              est_pob_mod = survey_mean(pobre_moderada  , vartype ="var" ))

  estima_calib

<- pivot_wider(
    data = estima_calib_ipm,
    id_cols = "cve_mun",
    names_from = "ipm",
    values_from = c("est_ipm", "est_ipm_var"),values_fill = 0
  ) %>% full_join(estima_calib_pob)

  valida_ipm <- estima_calib_ipm %>% inner_join(total_mpio, by = "cve_mun") %>%
```

```
    group_by(ipm) %>%
    summarise(prop_ampliada = sum(est_ipm*den_mpio)/unique(tot_ent)) %>%
    full_join(tot_pob, by = "ipm")

  valida_pob <- estima_calib_pob %>% select(cve_mun, est_pob_ext , est_pob_mod) %>%
    inner_join(total_mpio, by = "cve_mun") %>%
    summarise(pob_ext_ampliada = sum(est_pob_ext * den_mpio) / unique(tot_ent),
              pob_mod_ampliada = sum(est_pob_mod * den_mpio) / unique(tot_ent),
              ipm_I = pob_ext_ampliada + pob_mod_ampliada) %>%
    bind_cols(pobreza[,1:2])

  saveRDS(list(valida = list(valida_pob = valida_pob,
                             valida_ipm = valida_ipm),
               estima_calib = estima_calib),
          file = paste0( "../output/2020/iteraciones/mpio_calib/",
                         ii_ent,"/iter",iter,".rds"))
}
fin <- Sys.time()
tiempo_total <- difftime(fin, inicio, units = "mins")
print(tiempo_total)
cat("#########################################################################\n")
```

14_estimacion_municipios_ipm_pobreza.l

Para la ejecución del presente análisis, se debe abrir el archivo **14_estimacion_municipios_ipm_pob** disponible en la ruta *Rcodes/2020/14_estimacion_municipios_ipm_pobreza.R*.

Este script en R realiza un análisis exhaustivo para calibrar y validar indicadores de pobreza a nivel municipal, utilizando modelos predictivos y datos de encuestas. Primero, el entorno de trabajo se limpia y se cargan las librerías necesarias para la manipulación de datos (`tidyverse`, `data.table`, `magrittr`, etc.) y para realizar cálculos estadísticos (`lme4`, `survey`, `srvyr`). Luego, se leen datos de predicciones y encuestas ampliadas, junto con información adicional sobre las líneas de bienestar.

En la sección inicial de procesamiento, se unen y mutan los datos de la encuesta ENIGH para calcular indicadores de pobreza y vulnerabilidad multidimensional. Se asignan categorías de pobreza (`ipm`) y se crean nuevas variables que reflejan la pobreza moderada y extrema, basadas en umbrales de ingreso y carencias sociales.

El script luego se centra en la preparación de datos para la calibración de modelos. La muestra ampliada se ajusta para incluir variables adicionales y predicciones de ingreso y carencias sociales. A continuación, se inicia un bucle que itera sobre un conjunto de códigos de entidad territorial. Para cada entidad, se filtran y ajustan los datos de la encuesta ampliada y se simulan nuevas variables (como ingresos y carencias sociales) utilizando distribuciones probabilísticas basadas en las predicciones anteriores.

Dentro de cada iteración, se realiza una calibración de los datos utilizando el método de "raking", que ajusta las estimaciones de los indicadores a las estimaciones poblacionales conocidas. Los resultados de la calibración se calculan para cada municipio y se comparan con los datos originales para validar la precisión de las estimaciones. Finalmente, los resultados de cada iteración se guardan en archivos RDS separados para su análisis posterior y se reporta el tiempo total de ejecución del proceso.

Este enfoque permite realizar un ajuste fino de los modelos predictivos y asegurar que las estimaciones de pobreza sean lo más precisas posible, utilizando simulaciones y técnicas avanzadas de calibración.

Limpieza del Entorno y Carga de Bibliotecas

Antes de iniciar el análisis, se realiza una limpieza del entorno de trabajo para asegurar que no queden variables o datos residuales que puedan interferir con el análisis actual. Se utiliza `rm(list = ls())` para eliminar todos los objetos del entorno de R.

A continuación, se cargan las bibliotecas necesarias para el análisis. Estas bibliotecas incluyen:

- **`tidyverse`**: Para manipulación y visualización de datos.
- **`data.table`**: Para operaciones eficientes con datos en tablas.
- **`openxlsx`**: Para leer y escribir archivos de Excel.
- **`magrittr`**: Para usar el operador de tubería (`%>%`) que facilita la cadena de operaciones.
- **`haven`**: Para importar y exportar datos en formatos de software estadístico como SPSS, Stata, y SAS.
- **`labelled`**: Para gestionar y manipular etiquetas de variables.
- **`sampling`**: Para técnicas de muestreo y análisis relacionados.
- **`lme4`**: Para ajustar modelos lineales y no lineales de efectos mixtos.
- **`survey`**: Para análisis de datos de encuestas complejas.
- **`srvyr`**: Para aplicar métodos de análisis de encuestas utilizando una sintaxis similar a `dplyr`.

Finalmente, se carga un script adicional desde una ubicación específica, `benchmarking_indicador.R`, que probablemente contiene funciones y procedimientos específicos para el análisis de benchmarking que se realizará a continuación.

```
rm(list = ls())
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(haven)
library(labelled)
library(sampling)
library(lme4)
library(survey)
library(srvyr)
source("../source/benchmarking_indicador.R")
```

Lectura de Datos

Se inicia el análisis cargando los datos necesarios y las predicciones de los modelos ajustados previamente.

Primero, se leen las predicciones almacenadas en el archivo `predicciones.rds`, que

contiene las predicciones para ingresos, alimentación, y seguridad social, así como la desviación estándar residual.

Luego, se carga la muestra ampliada desde el archivo `encuesta_ampliada.rds`. Esta muestra ampliada es esencial para ajustar los valores y validar las predicciones en un contexto más amplio.

A continuación, se carga la encuesta ENIGH desde `encuesta_sta.rds`. Se realiza una mutación en los datos para agregar la variable de ingreso (`ingreso`) basada en el indicador `ictpc`.

Finalmente, se lee el archivo `Lineas_Bienestar.csv`, que contiene las líneas de bienestar. Estos datos se importan utilizando la función `read.delim` con las especificaciones adecuadas para el separador de campos y el decimal. Posteriormente, se convierte la columna `area` a formato de carácter para su uso en el análisis.

```
predicciones <- readRDS( "../output/2020/modelos/predicciones.rds")
muestra_ampliada <- readRDS("../output/2020/encuesta_ampliada.rds")
encuesta_enigh <- readRDS("../input/2020/enigh/encuesta_sta.rds") %>%
  mutate(ingreso = ictpc)

LB <- read.delim(
    "../input/2020/Lineas_Bienestar.csv",
    header = TRUE,
    sep = ";",
    dec = ","
  ) %>% mutate(area = as.character(area))
```

Cálculo del IPM en la Encuesta

En esta sección, se calcula el Índice de Pobreza Multidimensional (IPM) y se definen nuevas variables relacionadas con la pobreza en la encuesta ENIGH.

Primero, se unen los datos de la encuesta ENIGH con las líneas de bienestar (`LB`) mediante la función `inner_join`. Esto permite integrar la información adicional necesaria para el cálculo del IPM.

Luego, se añade una nueva columna `tol_ic` que representa la suma de los indicadores de carencia social (`ic_segsoc`, `ic_ali_nc`, `ic_asalud`, `ic_cv`, `ic_sbv`, `ic_rezedu`). A continuación, se calcula el IPM basado en las siguientes condiciones: - **Categoría I:** Población en situación de pobreza, donde el ingreso es menor que el umbral de pobreza (`lp`) y el total de carencias sociales (`tol_ic`) es mayor o igual a 1. - **Categoría II:** Población vulnerable por carencias sociales, donde el ingreso es mayor o igual al umbral de pobreza y el total de carencias sociales es mayor o igual a 1. - **Categoría III:** Población vulnerable por ingresos, donde el ingreso es menor o igual al umbral de pobreza y el total de carencias sociales es menor a 1. - **Categoría IV:** Población no

pobre multidimensional y no vulnerable, donde el ingreso es mayor o igual al umbral de pobreza y el total de carencias sociales es menor a 1.

Además, se crean dos nuevas variables: - **pobre_moderada:** Indica si una persona está en situación de pobreza moderada, donde el ingreso está entre el límite inferior (`li`) y el límite de pobreza (`lp`), y el total de carencias sociales (`tol_ic`) es mayor a 2. - **pobre_extrema:** Indica si una persona está en situación de pobreza extrema, definida como aquellos en la categoría "I" del IPM y que no cumplen con los criterios de pobreza moderada.

Estas variables proporcionan una visión más detallada sobre los niveles de pobreza y las carencias sociales en la muestra.

```
##############################################################################
# IPM en la enigh
##############################################################################
encuesta_enigh <-
  encuesta_enigh %>% inner_join(LB)  %>%
  mutate(
    tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv +  ic_sbv + ic_rezedu,
    ipm   = case_when(
      # Población en situación de pobreza.
      ingreso < lp  &  tol_ic >= 1 ~ "I",
      # Población vulnerable por carencias sociales.
      ingreso >= lp & tol_ic >= 1 ~ "II",
      # Poblacion vulnerable por ingresos.
      ingreso <= lp & tol_ic < 1 ~ "III",
      # Población no pobre multidimensional y no vulnerable.
      ingreso >= lp & tol_ic < 1 ~ "IV"
    ),
    # Población en situación de pobreza moderada.
    pobre_moderada = ifelse(c(ingreso > li & ingreso < lp) &
                              tol_ic > 2, 1, 0),
    pobre_extrema = ifelse(ipm == "I" & pobre_moderada == 0, 1, 0)
  )
```

Preparación de la Muestra Intercensal

En esta sección, se ajusta y se preparan los datos de la muestra ampliada, incorporando las predicciones obtenidas previamente y calculando nuevas variables necesarias para el análisis.

Primero, la muestra ampliada se une con los datos de las líneas de bienestar (`LB`) para asegurar que toda la información relevante esté disponible. A continuación, se ajustan las variables de carencia social en la muestra ampliada para que sean binarias, es decir,

se establecen como 1 si el indicador es positivo y 0 en caso contrario. Esto incluye las variables de `ic_asalud`, `ic_cv`, `ic_sbv`, y `ic_rezedu`.

Se añade una nueva variable llamada `tol_ic4`, que es la suma de los indicadores ajustados de carencia social. Esta variable representa el total de carencias sociales en la muestra ampliada.

Además, se incorporan las predicciones de los modelos ajustados, que incluyen: - `pred_segsoc`: Predicción para el indicador de seguridad social. - `pred_ingreso`: Predicción para el ingreso. - `desv_estandar_residual`: Desviación estándar residual utilizada para ajustar el modelo de ingreso. - `pred_ic_ali_nc`: Predicción para el indicador de alimentación y necesidades básicas.

Finalmente, se elimina el objeto `predicciones` del entorno para liberar memoria, ya que los datos necesarios han sido incorporados en la muestra ampliada.

Esta preparación asegura que la muestra intercensal esté completa y lista para el análisis posterior.

```
##############################################################################
# preparando encuesta intercensal
##############################################################################

muestra_ampliada <- muestra_ampliada %>% inner_join(LB)
muestra_ampliada   %<>% mutate(
  ic_asalud = ifelse(ic_asalud == 1, 1,0),
  ic_cv = ifelse(ic_cv == 1, 1,0),
  ic_sbv = ifelse(ic_sbv == 1, 1,0),
  ic_rezedu = ifelse(ic_rezedu == 1, 1,0),
  tol_ic4 = ic_asalud + ic_cv +  ic_sbv + ic_rezedu,
  pred_segsoc = predicciones$pred_segsoc,
  pred_ingreso = predicciones$pred_ingreso,
  desv_estandar_residual = predicciones$desv_estandar_residual,
  pred_ic_ali_nc = predicciones$pred_ic_ali_nc
)

rm(predicciones)
```

Iteración por Entidades Federativas

En esta sección, se realiza un proceso iterativo para ajustar y calibrar la muestra post-calibración en cada entidad federativa. El objetivo es obtener estimaciones precisas a nivel municipal y realizar validaciones. El proceso se repite para cada entidad federal especificada.

1. Inicialización y Configuración:

El código comienza con una lista de entidades federativas, representadas por sus códigos (`ii_ent`). Se establece un bucle que itera sobre cada uno de estos códigos. Para cada entidad federal, se imprime la hora de inicio de la iteración y se realiza el filtrado de datos específicos para la entidad actual.

2. Filtrado y Cálculo de Totales:

Para la entidad actual (`ii_ent`), se filtra la muestra ampliada (`muestra_ampliada`) para incluir solo los datos correspondientes. Luego, se calcula el total de cada municipio (`total_mpio`) sumando una variable de ponderación (`factor`). También se calcula el total para la entidad (`tot_ent`).

La encuesta de referencia (`encuesta_enigh`) se filtra para la entidad actual y se eliminan las observaciones con valores faltantes (`na.omit`). A partir de esta encuesta, se calcula el total de población (`tot_pob`) por tipo de pobreza multidimensional (`ipm`). Se estima la proporción y el total ajustado para cada categoría de pobreza.

3. Cálculo de Pobreza:

Se calculan las tasas de pobreza moderada y extrema utilizando los datos de la encuesta. La pobreza moderada se define como aquellos con ingresos entre los umbrales de pobreza (`li` y `lp`) y con un índice de carencia social mayor que 2. La pobreza extrema se define como aquellos en situación de pobreza multidimensional sin pobreza moderada.

4. Ajuste y Calibración:

En cada iteración, se ajusta la muestra post-calibración (`muestra_post`) con las predicciones de los modelos (`pred_segsoc`, `pred_ic_ali_nc`, `pred_ingreso`) y la desviación estándar residual (`desv_estandar_residual`). Se calculan nuevas variables relacionadas con el índice de pobreza multidimensional (IPM) y la pobreza moderada y extrema.

Se crean variables dummy para los municipios y las categorías de IPM y se incorporan al diseño de la encuesta (`diseno_post`). Luego, se realiza la calibración utilizando el método de raking (`calibrate`), ajustando el diseño para que coincida con las estimaciones de la población (`Tx_hat`).

5. Estimación y Validación:

Se obtienen las estimaciones calibradas para el IPM y la pobreza a nivel municipal. Estas estimaciones se comparan con los totales calculados previamente para validar la precisión de las predicciones. Se calcula la proporción de cada categoría de IPM y se compara con los totales esperados.

6. Guardado de Resultados:

Los resultados de cada iteración se guardan en un archivo `.rds` en una carpeta específica para la entidad y la iteración actual. El archivo incluye: - `valida`: Resultados de la

validación de la pobreza y el IPM. - `estima_calib`: Estimaciones calibradas para el IPM y la pobreza.

Finalmente, se mide y se imprime el tiempo total que tomó cada iteración. Se libera la memoria utilizada con `gc()` para optimizar el rendimiento durante el proceso iterativo.

Este procedimiento asegura que las estimaciones sean precisas y que se validen adecuadamente para cada entidad federativa.

```
ii_ent <- "02"
iter = 1

for(ii_ent in c("03", "06", "23", "04", "01", "22", "27", "25", "18",
                "05", "17", "28", "10", "26", "09", "32", "08", "19", "29",
                "24", "11", "31", "13", "16", "12", "14", "15", "07", "21",
                "30",  "20" ,"02"
)){
  cat("########################################################################\n")
  inicio <- Sys.time()
  print(inicio)

  # Filtrar muestra post
  muestra_post = muestra_ampliada %>% filter(ent == ii_ent)

  # Cálculo de totales por municipio y población
  total_mpio  <- muestra_post %>% group_by(cve_mun) %>%
    summarise(den_mpio = sum(factor), .groups = "drop") %>%
    mutate(tot_ent = sum(den_mpio))

  encuesta_sta = encuesta_enigh %>% filter(ent == ii_ent)
  encuesta_sta %<>% na.omit()

  tot_pob <- encuesta_sta %>%
    group_by(ipm) %>%
    summarise(num_ent = sum(fep), .groups = "drop") %>%
    transmute(ipm, prop_ipm = num_ent/sum(num_ent),
              tx_ipm =  sum(total_mpio$den_mpio)*prop_ipm) %>%
    filter(ipm != "I")

  pobreza <- encuesta_sta %>% summarise(
    pobre_ext = weighted.mean(pobre_extrema, fep),
    pob_mod = weighted.mean(pobre_moderada, fep),
    pobre_moderada = sum(total_mpio$den_mpio)*pob_mod,
    pobre_extrema = sum(total_mpio$den_mpio)*pobre_ext)
```

```
Tx_ipm <- setNames(tot_pob$tx_ipm,paste0("ipm_",tot_pob$ipm))
Tx_pob <- pobreza[,3:4] %>% as.vector() %>% unlist()

tx_mun <- setNames(total_mpio$den_mpio,
                   paste0("cve_mun_",total_mpio$cve_mun))

Tx_hat <- c(Tx_pob, Tx_ipm, tx_mun)

for(iter in 1:200){
  cat("\n municipio = ", ii_ent,"\n\n")
  cat("\n iteracion = ", iter,"\n\n")

  # Ajuste de la muestra post
  muestra_post %<>%
    mutate(
      ic_segsoc = rbinom(n = n(), size = 1, prob = pred_segsoc),
      ic_ali_nc = rbinom(n = n(), size = 1, prob = pred_ic_ali_nc),
      ingreso = pred_ingreso + rnorm(n = n(), mean = 0, desv_estandar_residual),
      tol_ic = tol_ic4 + ic_segsoc + ic_ali_nc
    ) %>% mutate(
      ipm   = case_when(
        # Población en situación de pobreza.
        ingreso < lp  &  tol_ic >= 1 ~ "I",
        # Población vulnerable por carencias sociales.
        ingreso >= lp & tol_ic >= 1 ~ "II",
        # Poblacion vulnerable por ingresos.
        ingreso <= lp & tol_ic < 1 ~ "III",
        # Población no pobre multidimensional y no vulnerable.
        ingreso >= lp & tol_ic < 1 ~ "IV"
      ),
      # Población en situación de pobreza moderada.
      pobre_moderada = ifelse(c(ingreso > li & ingreso < lp) &
                                tol_ic > 2, 1, 0),
      pobre_extrema = ifelse(ipm == "I" & pobre_moderada == 0, 1, 0))

  Xk <- muestra_post %>% select("ipm", "cve_mun") %>%
    fastDummies::dummy_columns(select_columns = c("ipm", "cve_mun")) %>%
    select(names(Tx_hat[-c(1:2)]))

  diseno_post <- bind_cols(muestra_post, Xk) %>%
    mutate(fep = factor) %>%
    as_survey_design(
```

```
    ids = upm,
    weights = fep,
    nest = TRUE
  )

mod_calib <-  as.formula(paste0("~ -1+",paste0(names(Tx_hat), collapse = " + ")))

diseno_calib <- calibrate(diseno_post, formula = mod_calib,
                          population = Tx_hat, calfun = "raking")

estima_calib_ipm  <- diseno_calib %>% group_by(cve_mun, ipm) %>%
  summarise(est_ipm = survey_mean(vartype ="var"))

estima_calib_pob  <- diseno_calib %>% group_by(cve_mun) %>%
  summarise(est_pob_ext = survey_mean(pobre_extrema , vartype ="var"),
            est_pob_mod = survey_mean(pobre_moderada  , vartype ="var" ))

estima_calib <- pivot_wider(
  data = estima_calib_ipm,
  id_cols = "cve_mun",
  names_from = "ipm",
  values_from = c("est_ipm", "est_ipm_var"), values_fill = 0
) %>% full_join(estima_calib_pob)

valida_ipm <- estima_calib_ipm %>% inner_join(total_mpio, by = "cve_mun") %>%
  group_by(ipm) %>%
  summarise(prop_ampliada = sum(est_ipm*den_mpio)/unique(tot_ent)) %>%
  full_join(tot_pob, by = "ipm")

valida_pob <- estima_calib_pob %>% select(cve_mun, est_pob_ext, est_pob_mod) %>%
  inner_join(total_mpio, by = "cve_mun") %>%
  summarise(pob_ext_ampliada = sum(est_pob_ext * den_mpio) / unique(tot_ent),
            pob_mod_ampliada = sum(est_pob_mod * den_mpio) / unique(tot_ent),
            ipm_I = pob_ext_ampliada + pob_mod_ampliada) %>%
  bind_cols(pobreza[,1:2])

saveRDS(list(valida = list(valida_pob = valida_pob,
                           valida_ipm = valida_ipm),
             estima_calib = estima_calib),
        file = paste0( "../output/2020/iteraciones/mpio_calib/",
                       ii_ent,"/iter",iter,".rds"))
gc()
```

```
  }
  fin <- Sys.time()
  tiempo_total <- difftime(fin, inicio, units = "mins")
  print(tiempo_total)
  cat("#########################################################################\n")
}
```

15_estimacion_municipios_ipm_pobreza_

Para la ejecución del presente análisis, se debe abrir el archivo **15_estimacion_municipios_ipm_pob** disponible en la ruta **Rcodes/2020/15_estimacion_municipios_ipm_pobreza_error.R**.

El script comienza con la limpieza del entorno de trabajo y la carga de librerías necesarias, como `tidyverse` y `survey`. A continuación, se cargan y preparan los datos de `muestra_ampliada`, `encuesta_enigh`, y `LB`. Se calculan las variables relacionadas con la pobreza multidimensional (IPM) en la encuesta, incluyendo la clasificación en categorías como "I", "II", "III", y "IV", y se identifican los casos de pobreza moderada y extrema.

Luego, se procesa la información de los resultados de estimación por municipio, que se leen desde archivos generados en iteraciones previas. Se calculan y combinan las estimaciones, varianzas y medias, y se consolidan en un solo conjunto de datos. Estos resultados se guardan en archivos RDS y Excel para su posterior análisis.

Finalmente, se realiza una visualización comparativa de las estimaciones de pobreza multidimensional. Utilizando `ggplot2`, se crean gráficos que comparan las estimaciones obtenidas de la ENIGH con las de CEPAL para cada estado, y estos gráficos se exportan en formato PNG. Este paso permite la evaluación visual de la precisión y consistencia de las estimaciones realizadas.

El código que compartiste está orientado a procesar y analizar datos de una encuesta para calcular y comparar las estimaciones de pobreza y el Índice de Pobreza Multidimensional (IPM). A continuación, te proporciono un desglose de las principales secciones y acciones del código:

Preparación del Entorno

En esta etapa se realiza la limpieza del entorno de trabajo y se cargan las bibliotecas necesarias para llevar a cabo el análisis. Primero, se elimina cualquier objeto que pueda haber quedado en el entorno global, utilizando el comando `rm(list = ls())`. Este paso es crucial para evitar interferencias de variables o datos residuales de análisis previos.

A continuación, se cargan las bibliotecas necesarias para el análisis. Las bibliotecas incluyen `tidyverse`, un conjunto de paquetes diseñado para facilitar la manipulación

y visualización de datos, `magrittr`, que proporciona el operador de encadenamiento `%>%` para escribir código más legible y fluido, `survey`, que ofrece herramientas para el análisis de datos de encuestas complejas, y `srvyr`, que permite realizar operaciones sobre datos de encuestas utilizando una sintaxis similar a `dplyr`. La carga de estas bibliotecas asegura que se disponga de las herramientas adecuadas para manejar y analizar los datos de manera efectiva.

```
rm(list = ls())
library(tidyverse)
library(magrittr)
library(survey)
library(srvyr)
```

Lectura de Datos

En esta sección se lleva a cabo la carga y preparación de los datos necesarios para el análisis. Se leen tres conjuntos de datos clave: `muestra_ampliada`, `encuesta_enigh`, y `LB`.

Primero, se carga el archivo de muestra ampliada, almacenado en el archivo `../output/2020/encuesta_ampliada.rds`, utilizando la función `readRDS()`. Este archivo contiene la muestra ampliada del censo, que incluye una serie de variables importantes para el análisis posterior.

Luego, se carga el archivo `encuesta_enigh`, ubicado en `../input/2020/enigh/encuesta_sta.rds`. Este archivo se lee también mediante `readRDS()`, y se le añade una columna nueva llamada `ingreso`, que toma los valores de la columna `ictpc`. Esta columna `ingreso` se usará para realizar cálculos y análisis en los pasos siguientes.

Finalmente, se lee el archivo `Lineas_Bienestar.csv` con la función `read.delim()`, especificando que el archivo está separado por punto y coma (`sep = ";"`) y que los decimales están representados por comas (`dec = ","`). Este archivo contiene datos sobre las líneas de bienestar, y se convierte la columna `area` a formato carácter para asegurar una correcta manipulación de los datos en el análisis.

Estos datos, una vez cargados, se preparan para el análisis mediante la adición de columnas y el ajuste de formatos necesarios.

```
muestra_ampliada <- readRDS("../output/2020/encuesta_ampliada.rds")
encuesta_enigh <- readRDS("../input/2020/enigh/encuesta_sta.rds") %>%
  mutate(ingreso = ictpc)

LB <- read.delim(
    "input/2020/Lineas_Bienestar.csv",
    header = TRUE,
    sep = ";",
```

```
    dec = ","
  ) %>% mutate(area = as.character(area))
```

Cálculo del Índice de Pobreza Multidimensional (IPM) en `encuesta_enigh`

Esta sección del código realiza el cálculo del Índice de Pobreza Multidimensional (IPM) y de variables asociadas utilizando los datos de la encuesta `encuesta_enigh` y la información sobre las líneas de bienestar `LB`. El objetivo es integrar la información de las carencias y el ingreso para clasificar a los individuos en diferentes categorías de pobreza y vulnerabilidad.

1. **Integración de Datos**: Primero, se realiza una unión interna (`inner_join()`) entre `encuesta_enigh` y `LB`, añadiendo las variables de las líneas de bienestar (`LB`) a los datos de la encuesta.
2. **Cálculo de `tol_ic`**: Se calcula la variable `tol_ic`, que representa el total de las carencias sociales que un individuo enfrenta. Esta variable es la suma de varias carencias (`ic_segsoc`, `ic_ali_nc`, `ic_asalud`, `ic_cv`, `ic_sbv`, y `ic_rezedu`).
3. **Determinación del IPM**:
   - **Pobreza Multidimensional (`ipm`)**: Se clasifica a cada individuo en una de las cuatro categorías de pobreza multidimensional basadas en el ingreso (`ingreso`) y el total de carencias (`tol_ic`).
4. **Variables de Pobreza Moderada y Extrema**:
   - **Pobreza Moderada (`pobre_moderada`)**: Se define como 1 si el ingreso está entre la línea de pobreza extrema (`li`) y la línea de pobreza (`lp`), y el total de carencias sociales (`tol_ic`) es mayor que 2; en caso contrario, se define como 0.
   - **Pobreza Extrema (`pobre_extrema`)**: Se define como 1 si el individuo está en la categoría de pobreza multidimensional "I" y no es considerado pobre moderado; en caso contrario, se define como 0.

Estas transformaciones y cálculos permiten clasificar y analizar el nivel de pobreza y las condiciones de vulnerabilidad de los individuos en la encuesta `encuesta_enigh`.

```
encuesta_enigh <-
  encuesta_enigh %>% inner_join(LB)  %>%
  mutate(
    tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv +  ic_sbv + ic_rezedu,
    ipm   = case_when(
      ingreso < lp  &  tol_ic >= 1 ~ "I",
      ingreso >= lp & tol_ic >= 1 ~ "II",
      ingreso <= lp & tol_ic < 1 ~ "III",
```

```
      ingreso >= lp & tol_ic < 1 ~ "IV"
    ),
    pobre_moderada = ifelse(c(ingreso > li & ingreso < lp) &
                              tol_ic > 2, 1, 0),
    pobre_extrema = ifelse(ipm == "I" & pobre_moderada == 0, 1, 0)
  )
```

Resumen de Datos por Municipio

El análisis de los datos de la `muestra_ampliada` se centra en calcular la cantidad total de personas en cada municipio. Para lograr esto, se agrupan los datos por entidad federativa y por clave de municipio. Luego, se suma el número de personas, representado por la variable `factor`, para cada municipio específico. Esta operación proporciona un resumen detallado de la población en cada municipio, generando un `data frame` que incluye las columnas `ent` (entidad federativa), `cve_mun` (clave del municipio) y `N_pers_mpio` (número total de personas en el municipio). Este resumen es fundamental para la correcta calibración de los modelos y la interpretación de los resultados a nivel municipal.

```
N_mpio <- muestra_ampliada %>% group_by(ent, cve_mun) %>%
  summarise(N_pers_mpio = sum(factor))
```

Lectura de Archivos de Iteraciones y Cálculo de Estadísticas

Se procede a la lectura de los archivos generados durante las iteraciones del ajuste de modelos y calibración. Primero, se obtienen todos los archivos ubicados en el directorio `../output/2020/iteraciones/mpio_calib/`, que contienen los resultados de las iteraciones realizadas para cada entidad federativa. A continuación, se crea un `data frame` que incluye el nombre completo de cada directorio de iteración y la cantidad de archivos presentes en cada uno. Este `data frame` se filtra para mantener únicamente aquellos directorios que contienen al menos un archivo, lo que indica que se han generado resultados en esas iteraciones. Este proceso facilita el análisis posterior al asegurar que solo se consideren las iteraciones que tienen datos válidos para el análisis de las estimaciones.

```
list_estiacion <- list.files("../output/2020/iteraciones/mpio_calib/", full.names = TR

list_estiacion <- data.frame(list_estiacion,
iter = list_estiacion %>% map_dbl(~list.files(.x) %>% length())
) %>% filter(iter > 0)
```

Proceso Iterativo para Cada estado

Para cada entidad federativa, se lleva a cabo un análisis iterativo para calcular estimaciones promedio, varianza y otros indicadores estadísticos relevantes. Este proceso comienza con la lectura de los archivos de iteraciones para cada estado, que contienen resultados de las estimaciones calibradas. Los archivos se cargan y se combinan en un solo `data frame` para su procesamiento.

Para cada estado, se realiza una serie de cálculos estadísticos. Primero, se obtiene la estimación promedio para cada indicador por municipio, calculando la media de las estimaciones en los datos. Posteriormente, se calcula el tamaño de la muestra para cada municipio y la varianza de las estimaciones. La varianza se descompone en la varianza entre las iteraciones (B) y la varianza promedio (Ubar) dentro de las iteraciones.

Se realiza una combinación de estos datos para calcular la varianza total de las estimaciones, sumando la varianza promedio (Ubar) con la varianza entre las iteraciones ajustada por el tamaño de la muestra. La estimación del error estándar se obtiene como la raíz cuadrada de la varianza.

Finalmente, se organiza toda la información en un `data frame` que incluye el número de observaciones, las estimaciones promedio, las varianzas y los errores estándar para cada indicador por municipio. Todos estos resultados se guardan en una lista para su posterior análisis y evaluación.

```
resul_ent_ipm <- list()

for(ii_ent in list_estiacion$list_estiacion){

  archivos <- list.files(ii_ent, full.names = TRUE)
  datos <- map_df(archivos, ~ readRDS(.x)$estima_calib)

  dat_estima <- datos %>% group_by(cve_mun) %>%
    summarise_at(vars(!matches("var")), mean) %>% data.frame()

  dat_n <- datos %>% group_by(cve_mun) %>% tally()
  dat_B <- datos %>% group_by(cve_mun) %>%
    summarise_at(vars(!matches("var")), var) %>% data.frame()

  dat_Ubar <- datos %>% group_by(cve_mun) %>%
    summarise_at(vars(matches("var")), mean) %>% data.frame()

  names(dat_Ubar) <- gsub("_var", "", names(dat_Ubar))

  dat_var <- dat_B %>%
    gather(key = "Indicador", value = "B", -cve_mun) %>%
```

```
    inner_join(
      dat_Ubar %>% gather(key = "Indicador", value = "Ubar", -cve_mun)
    ) %>% inner_join(dat_n)

  var_est <-  dat_var %>%
    transmute(cve_mun, Indicador,
              var = Ubar + (1 +1/n)*B,
              ee = sqrt(var)) %>%
    pivot_wider(
      data = .,
      id_cols = "cve_mun",
      names_from = "Indicador",
      values_from = c("var", "ee"), values_fill = 0
    )

  dat_var <- pivot_wider(
    data = dat_var,
    id_cols = "cve_mun",
    names_from = "Indicador",
    values_from = c("B", "Ubar"), values_fill = 0
  )

  resul_ent_ipm[[ii_ent]] <- dat_n %>% inner_join(dat_estima) %>%
    inner_join(dat_var) %>%
    inner_join(var_est)
}
```

Guardar Resultados y Crear Archivos Excel

Una vez calculadas todas las estimaciones y estadísticas para cada entidad federativa, los resultados se consolidan y se guardan en varios formatos para su posterior análisis y distribución.

Primero, los resultados de todas las entidades federativas se combinan en un solo `data frame`. Este `data frame` se enriquece con el número de personas por municipio, lo que permite contextualizar las estimaciones. Se seleccionan las columnas relevantes, que incluyen estimaciones de diversos indicadores, errores estándar y número de personas por municipio.

El `data frame` consolidado se guarda en dos formatos. Se utiliza `saveRDS()` para guardar los datos en un archivo RDS, que es adecuado para el almacenamiento y recuperación eficiente en R. Además, se utiliza `openxlsx::write.xlsx()` para exportar los resultados a un archivo Excel, facilitando su revisión y análisis en herramientas de

hojas de cálculo.

Además, se calcula una estimación agregada a nivel estatal. Para esto, se agrupan los resultados por entidad federativa y se calculan las estimaciones totales para cada indicador, ponderadas por el número de personas en cada municipio. Este resumen estatal se guarda en otro archivo Excel separado, proporcionando una visión general de las estimaciones a nivel estatal.

Estos pasos garantizan que tanto los datos detallados por municipio como las estimaciones agregadas a nivel estatal estén disponibles en formatos accesibles y útiles para la toma de decisiones y la presentación de resultados.

```
temp <- resul_ent_ipm %>% bind_rows()

temp %<>% inner_join(N_mpio) %>%
  select(ent, cve_mun, N_pers_mpio, est_ipm_I,
         est_pob_mod, est_pob_ext, est_ipm_II:est_ipm_IV,
         ee_ipm_I = ee_est_ipm_I,
         ee_pob_mod = ee_est_pob_mod,
         ee_pob_ext = ee_est_pob_ext,
         ee_ipm_II = ee_est_ipm_II,
         ee_ipm_III = ee_est_ipm_III,
         ee_ipm_IV = ee_est_ipm_IV)

temp %>%
  saveRDS("../output/Entregas/2020/result_mpios.RDS")

openxlsx::write.xlsx(temp,
                     paste0(
                       "../output/Entregas/2020/estimacion_numicipal_",
                       Sys.Date() ,
                       ".xlsx"
                     ))

temp2 <- temp %>%
  select(ent, est_ipm_I:est_ipm_IV, N_pers_mpio) %>%
  group_by(ent) %>%
  summarise(across(starts_with("est_"),
                   ~ sum(.x * N_pers_mpio) / sum(N_pers_mpio)))

openxlsx::write.xlsx(temp2,
                     paste0(
                       "../output/Entregas/2020/estimacion_estado_",
                       Sys.Date() ,
```

```
                        ".xlsx"
                      "))
```

Generación de Gráficos

El proceso de generación de gráficos se enfoca en visualizar las estimaciones de pobreza multidimensional por entidad federativa para el año 2020. Los gráficos se crean para diferentes tipos de Índice de Pobreza Multidimensional (IPM) y se guardan en archivos PNG.

1. **Preparación de los Datos para Graficar:** Se utiliza el diseño muestral (`diseno`) de la encuesta ENIGH para calcular las estimaciones directas de los indicadores de pobreza multidimensional y sus intervalos de confianza. Estos cálculos se realizan para cada entidad federativa y se organizan en un `data frame` (`estimad_dir`), que se amplía para incluir las estimaciones directas de pobreza moderada y extrema.

2. **Configuración de los Datos para Cada IPM:** Se crea una lista de tipos de IPM (`ind`) que incluye tanto los tipos de IPM ("I", "II", "III", "IV") como las categorías de pobreza moderada ("mod") y extrema ("ext"). Para cada tipo, se seleccionan los datos relevantes y se preparan para la visualización.

3. **Generación de Gráficos:** Para cada tipo de IPM, se realiza lo siguiente:

    - Se filtran los datos para el tipo de IPM actual y se preparan las columnas para los límites de los intervalos de confianza.
    - Los datos se transforman y se preparan para graficar las estimaciones y sus intervalos de confianza.
    - Se crea un gráfico con `ggplot2` que muestra las estimaciones de pobreza por entidad federativa. Las estimaciones de la encuesta ENIGH se representan con barras de error, mientras que las estimaciones de CEPAL se muestran con puntos. Los gráficos se personalizan con colores distintos, etiquetas, y un tema limpio.
    - Los gráficos se guardan en archivos PNG con un tamaño adecuado para presentaciones y análisis.

Este proceso asegura que los resultados de las estimaciones de pobreza sean visualmente accesibles y comparables entre las distintas entidades federativas y los diferentes tipos de IPM.

```
diseno <- encuesta_enigh %>% na.omit() %>%
  as_survey_design(
    ids = upm,
    weights = fep,
    nest = TRUE,
```

```
    strata = estrato
  )

estimad_dir <- diseno %>% group_by(ent, ipm) %>%
  summarise(direct = survey_mean(vartype = "ci" ))

estimad_dir_ipm <- pivot_wider(
  data = estimad_dir,
  id_cols = "ent",
  names_from = "ipm",
  values_from = c("direct", "direct_low", "direct_upp")
)

estimad_dir <- diseno %>% group_by(ent) %>%
  summarise(direct_mod = survey_mean(pobre_moderada, vartype = "ci"),
            direct_ext = survey_mean(pobre_extrema, vartype = "ci"))

estimad_dir %<>% inner_join(estimad_dir_ipm)

ind <- c("I", "II", "III", "IV", "mod", "ext")
ii_ipm  = 1
for (ii_ipm in 1:6){
    ii_ipm <- ind[ii_ipm]
    paso <- paste0("(_|\\b)", ii_ipm, "(_|\\b)")
    dat_plot <- inner_join(estimad_dir, temp2) %>%
      select(ent, matches(paso))

    dat_lim  <- dat_plot %>% select(ent, matches("upp|low"))

    names(dat_lim) <- c("ent","Lim_Inf", "Lim_Sup")

    dat_plot %<>% select(-matches("upp|low")) %>%
      gather(key = "Origen", value = "Prop", -ent) %>%
      mutate(Origen = ifelse(grepl(pattern = "direct", x = Origen),
                             "ENIGH", "Estimación CEPAL")) %>%
      inner_join(dat_lim)

    gg_plot <- ggplot(data = dat_plot, aes(x = ent, y = Prop, color = Origen)) +
      labs(
        x = "",
        y = "Estimación",
        color = "",
```

```
        title = paste0("Estimación de la pobreza multidimensional 2020 - Tipo ", ii_ip
      )+ theme_bw(20) +
      geom_jitter(width = 0.3) +
      theme(legend.position

 = "bottom",
            plot.title = element_text(hjust = 0.5)) +
      scale_y_continuous(labels = scales::percent_format(scale = 100))

    gg_plot <-  gg_plot +
      geom_errorbar(data = dat_plot %>% filter(Origen   == "ENIGH"),
                    aes(ymin = Lim_Inf, ymax = Lim_Sup, x = ent),
                    width = 0.2, linewidth = 1) +
      scale_color_manual(
         breaks = c("ENIGH", "Estimación CEPAL"),
         values = c("red", "blue3")
      ) +
      theme(
        legend.position = "bottom",
        axis.title = element_text(size = 10),
        axis.text.y = element_text(size = 10),
        axis.text.x = element_text(
          angle = 90,
          size = 8,
          vjust = 0.3
        ),
        legend.title = element_text(size = 15),
        legend.text = element_text(size = 15)
      )

    ggsave(plot = gg_plot, width = 16, height = 9,
           filename = paste0("../output/Entregas/2020/plot_uni/ipm_", ii_ipm, ".png"))
}
```

16_Mapas.R

Para la ejecución del presente análisis, se debe abrir el archivo **16_Mapas.R** disponible en la ruta *Rcodes/2020/16_Mapas.R*.

Este script en R está diseñado para procesar y visualizar datos geoespaciales relacionados con la pobreza multidimensional en México para el año 2020. Primero, se limpia el entorno de trabajo con `rm(list = ls())` para eliminar objetos previos y `gc()` para liberar memoria, lo que garantiza que el script se ejecute en un entorno limpio. Luego, se cargan diversas librerías necesarias para la manipulación de datos (`dplyr`, `data.table`), para trabajar con datos espaciales (`sf`), y para la creación de gráficos (`tmap`).

La memoria disponible se limita a 250 GB con `memory.limit(250000000)`, lo que es crucial para manejar grandes conjuntos de datos y evitar errores por falta de memoria. A continuación, se cargan dos conjuntos de datos importantes: `ipm_mpios`, que contiene estimaciones de pobreza para los municipios, y `ShapeDAM`, que es un archivo shapefile con la geometría de los municipios. En `ShapeDAM`, se renombra y ajusta la columna `cve_mun` para extraer códigos de entidad y se eliminan columnas innecesarias.

El script luego define los cortes de clasificación para las estimaciones de pobreza y genera mapas temáticos para diferentes tipos de pobreza multidimensional (`IPM_I`, `IPM_II`, `IPM_III`, `IPM_IV`) y para pobreza extrema y moderada. Utiliza la función `tm_shape()` de la librería `tmap` para crear los mapas y `tm_polygons()` para definir la apariencia de las áreas en el mapa, incluyendo los colores, la leyenda, y el título. Los mapas se guardan en archivos PNG con `tmap_save()`, especificando dimensiones y resolución para asegurar una alta calidad en la visualización.

Finalmente, el script también visualiza y guarda los errores de estimación asociados con cada tipo de pobreza, utilizando el mismo procedimiento para crear mapas temáticos con los errores de estimación. La visualización de estos errores ayuda a evaluar la precisión de las estimaciones y a identificar áreas con alta incertidumbre. Este proceso proporciona una manera efectiva de comunicar visualmente la distribución de la pobreza y los errores asociados a través de mapas detallados y estéticamente ajustados.

Configuración y carga de bibliotecas

El siguiente código configura el entorno de R, carga las bibliotecas necesarias y establece un límite de memoria para el análisis:

```
### Cleaning R environment ###
rm(list = ls())
gc()

#################
### Libraries ###
#################
library(dplyr)
library(data.table)
library(haven)
library(magrittr)
library(stringr)
library(openxlsx)
library(tmap)
library(sf)
select <- dplyr::select

###------------ Definiendo el límite de la memoria RAM a emplear ------------###
memory.limit(250000000)
```

Primero, se limpia el entorno de trabajo de R eliminando todos los objetos y se ejecuta la recolección de basura para liberar memoria. Luego, se cargan varias bibliotecas esenciales: `dplyr` y `data.table` para manipulación y análisis de datos; `haven` para importar datos de formatos específicos; `magrittr` para operaciones de encadenamiento; `stringr` para manipulación de cadenas de texto; `openxlsx` para trabajar con archivos Excel; y `tmap` y `sf` para análisis y visualización espacial. Finalmente, se establece un límite de memoria RAM de 250 MB para el uso de R, asegurando que el análisis se realice dentro de los recursos disponibles del sistema.

Carga de datos y creación de mapas de IPM

El siguiente código se encarga de cargar los datos de estimaciones de pobreza multidimensional y el archivo de formas geográficas, para luego preparar los datos para la visualización en mapas.

```
ipm_mpios <-
  readRDS("../output/Entregas/2020/result_mpios.RDS") %>%
  mutate(est_pob_ext = ifelse(est_pob_ext < 0, 0, est_pob_ext))

ShapeDAM <- read_sf("../shapefile/2020/MEX_2020.shp")
```

```
ShapeDAM %<>% mutate(cve_mun = CVEGEO,
                     ent = substr(cve_mun, 1, 2),
                     CVEGEO = NULL)
```

Primero, se cargan los resultados de estimaciones de pobreza multidimensional desde un archivo RDS (`result_mpios.RDS`). Se ajusta la variable `est_pob_ext` para asegurarse de que no tenga valores negativos, estableciendo dichos valores en cero.

Luego, se lee el archivo de forma geográfica (`MEX_2020.shp`) que contiene la geometría de los municipios de México. Se renombra la columna `CVEGEO` a `cve_mun` para que coincida con la variable en el conjunto de datos de estimaciones. También se extrae el código de entidad federal (`ent`) de los primeros dos caracteres de `cve_mun`, y se elimina la columna original `CVEGEO` que ya no es necesaria.

Estos pasos preparan los datos para ser usados en la creación de mapas que visualicen las estimaciones de pobreza multidimensional a nivel municipal.

Definición de Cortes Esta sección establece los cortes utilizados para clasificar las estimaciones de pobreza multidimensional en diferentes categorías. Los cortes están definidos como fracciones de 100 (por ejemplo, 0, 15, 30, 50, 80, y 100) para facilitar la visualización en los mapas temáticos. Estos cortes se utilizan para determinar los intervalos de los valores de estimación en los mapas.

```
cortes <- c(0, 15, 30, 50, 80, 100) / 100
```

Creación del Mapa para IPM Tipo I En esta sección, se genera un mapa temático para la estimación del Índice de Pobreza Multidimensional (IPM) Tipo I. Primero, se combina la información geoespacial con las estimaciones de IPM por municipio. Luego, se crea un mapa usando la función `tm_polygons`, donde los colores representan los diferentes intervalos de pobreza Tipo I. El mapa incluye una leyenda detallada con el diseño y formato especificados, y se guarda como una imagen PNG.

```
P1_mpio_norte <-
  tm_shape(ShapeDAM %>%
             inner_join(ipm_mpios, by = "cve_mun"))

Mapa_I <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "est_ipm_I",
    title = "Estimación de la pobreza \nmultidimensional 2020 - Tipo I",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
```

```
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_I,
  filename = "../output/Entregas/2020/mapas/IPM_I.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Creación del Mapa para IPM Tipo II En esta sección, se crea un mapa temático para la estimación del IPM Tipo II. Similar al mapa del IPM Tipo I, se utiliza la función `tm_polygons` para representar los intervalos de pobreza Tipo II con colores específicos. El mapa incluye una leyenda con el formato y diseño configurados para una mejor interpretación. Finalmente, el mapa se guarda como una imagen PNG en la ubicación especificada.

```
Mapa_II <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "est_ipm_II",
    title = "Estimación de la pobreza \nmultidimensional 2020 - Tipo II",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )
```

```
tmap_save(
  Mapa_II,
  filename = "../output/Entregas/2020/mapas/IPM_II.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Creación del Mapa para IPM Tipo III Esta sección está dedicada a la creación del mapa temático para el IPM Tipo III. Se sigue el mismo procedimiento que para los mapas de IPM Tipo I y Tipo II, adaptando el título y la variable correspondiente a la estimación del IPM Tipo III. La visualización se guarda como una imagen PNG, proporcionando una representación clara y detallada de la distribución de pobreza Tipo III.

```
Mapa_III <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "est_ipm_III",
    title = "Estimación de la pobreza \nmultidimensional 2020 - Tipo III",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_III,
  filename = "../output/Entregas/2020/mapas/IPM_III.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Creación del Mapa para IPM Tipo IV Finalmente, esta sección se encarga de generar el mapa temático para el IPM Tipo IV. El proceso es similar al de los mapas anteriores, pero se ajusta para representar la estimación del IPM Tipo IV. El mapa incluye una leyenda informativa y se guarda como un archivo PNG para su distribución y análisis.

```
Mapa_IV <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "est_ipm_IV",
    title = "Estimación de la pobreza \nmultidimensional 2020 - Tipo IV",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_IV,
  filename = "../output/Entregas/2020/mapas/IPM_IV.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Creación pobreza extrema y moderada, y errores de estimació

Esta sección establece los cortes utilizados para clasificar las estimaciones de pobreza en categorías de pobreza extrema y moderada. Los cortes están definidos como fracciones de 100 (por ejemplo, 0, 15, 30, 50, 80, y 100) para facilitar la visualización en los mapas temáticos. Estos cortes ayudan a segmentar los datos en intervalos significativos para su representación gráfica.

```
cortes <- c(0, 15, 30, 50, 80, 100) / 100
```

Aquí se muestra un resumen estadístico de la variable `est_pob_ext`, que representa la estimación de la pobreza extrema en los municipios. El resumen proporciona una visión

general de las estadísticas descriptivas de esta variable, como mínimo, máximo, media, y cuartiles.

```
summary(ipm_mpios$est_pob_ext)
```

Creación del Mapa de Pobreza Extrema En esta sección, se genera un mapa temático para la estimación de pobreza extrema. Se utiliza la función `tm_polygons` para representar los intervalos de pobreza extrema en colores distintos, con una leyenda detallada que facilita la interpretación. El mapa incluye configuraciones específicas para la leyenda y se guarda como una imagen PNG.

```
Mapa_ext <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "est_pob_ext",
    title = "Estimación de la pobreza extrema",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_ext,
  filename = "../output/Entregas/2020/mapas/pob_ext.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Resumen de Pobreza Moderada Esta sección muestra un resumen estadístico de la variable `est_pob_mod`, que representa la estimación de la pobreza moderada en los municipios. Similar al resumen de pobreza extrema, se proporciona una visión general de las estadísticas descriptivas de esta variable.

```
summary(ipm_mpios$est_pob_mod)
```

Creación del Mapa de Pobreza Moderada Aquí se genera un mapa temático para la estimación de pobreza moderada. Se utiliza la función `tm_polygons` para representar los intervalos de pobreza moderada con diferentes colores, junto con una leyenda informativa. Este mapa proporciona una visualización clara de la pobreza moderada en los municipios y se guarda como una imagen PNG.

```
Mapa_mod <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "est_pob_mod",
    title = "Estimación de la pobreza moderada",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_mod,
  filename = "../output/Entregas/2020/mapas/pob_mod.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Definición de Cortes para Errores de Estimación

En esta sección se establecen los cortes utilizados para clasificar los errores de estimación en categorías significativas. Los cortes están definidos como fracciones de 100 (0, 1, 5, 10, 25, 50, y 100) para permitir una visualización detallada de la variabilidad en los errores de estimación de pobreza multidimensional y sus tipos asociados.

```
cortes <- c(0, 1, 5, 10, 25, 50, 100) / 100
```

.0.1 Mapa de Error de Estimación del IPM Tipo I Se genera un mapa temático que visualiza el error de estimación del Índice de Pobreza Multidimensional (IPM) Tipo I. Utilizando la función `tm_polygons`, los errores se representan en diferentes colores según los intervalos definidos por los cortes. El mapa incluye una leyenda detallada y se guarda como una imagen PNG para su posterior análisis y presentación.

```
Mapa_I_ee <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "ee_ipm_I",
    title = "Error de estimación de la pobreza \nmultidimensional 2020 - Tipo I",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_I_ee,
  filename = "../output/Entregas/2020/mapas/IPM_I_ee.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Mapa de Error de Estimación del IPM Tipo II Aquí se crea un mapa temático para visualizar el error de estimación del IPM Tipo II. Los errores se clasifican en diferentes categorías de color según los cortes definidos. La leyenda y la disposición gráfica están configuradas para facilitar la interpretación de los resultados, y el mapa se guarda como una imagen PNG.

```
Mapa_II_ee <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "ee_ipm_II",
    title = "Error de estimación de la pobreza \nmultidimensional 2020 - Tipo II",
    palette = "Greens",
```

```
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_II_ee,
  filename = "../output/Entregas/2020/mapas/IPM_II_ee.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Mapa de Error de Estimación del IPM Tipo III Este mapa temático ilustra el error de estimación del IPM Tipo III, utilizando los cortes para segmentar los errores en diferentes niveles de color. La leyenda está diseñada para mostrar claramente las categorías de errores y el mapa se guarda como un archivo PNG.

```
Mapa_III_ee <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "ee_ipm_III",
    title = "Error de estimación de la pobreza \nmultidimensional 2020 - Tipo III",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )
```

```
tmap_save(
  Mapa_III_ee,
  filename = "../output/Entregas/2020/mapas/IPM_III_ee.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

5. Mapa de Error de Estimación del IPM Tipo IV Se genera un mapa temático que representa el error de estimación del IPM Tipo IV. Los errores se visualizan mediante diferentes colores según los intervalos definidos por los cortes. La leyenda y el diseño del mapa están ajustados para facilitar la interpretación visual de los errores, y el mapa se guarda como una imagen PNG.

```
Mapa_IV_ee <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "ee_ipm_IV",
    title = "Error de estimación de la pobreza \nmultidimensional 2020 - Tipo IV",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_IV_ee,
  filename = "../output/Entregas/2020/mapas/IPM_IV_ee.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Mapa de Error de Estimación de la Pobreza Extrema Este mapa temático ilustra el error de estimación para la pobreza extrema. Los datos se representan

en diferentes colores según los cortes definidos, con una leyenda que facilita la interpretación. El mapa se guarda como un archivo PNG para su uso en informes y análisis.

```
Mapa_ext_ee <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "ee_pob_ext",
    title = "Error de estimación de la pobreza extrema",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_ext_ee,
  filename = "../output/Entregas/2020/mapas/pob_ext_ee.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

Mapa de Error de Estimación de la Pobreza Moderada Finalmente, se genera un mapa temático que visualiza el error de estimación para la pobreza moderada. Los intervalos de error están representados por diferentes colores, y la leyenda está configurada para mostrar claramente las categorías de error. El mapa se guarda como un archivo PNG para su análisis y presentación.

```
Mapa_mod_ee <-
  P1_mpio_norte + tm_polygons(
    breaks = cortes,
    "ee_pob_mod",
    title = "Error de estimación de la pobreza moderada",
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(
```

```
    legend.show = TRUE,
    legend.text.size = 1.5,
    legend.outside.position = 'left',
    legend.hist.width = 1,
    legend.hist.height = 3,
    legend.stack = 'vertical',
    legend.title.fontface = 'bold',
    legend.text.fontface = 'bold'
  )

tmap_save(
  Mapa_mod_ee,
  filename = "../output/Entregas/2020/mapas/pob_mod_ee.png",
  width = 4000,
  height = 3000,
  asp = 0
)
```

17_estimacion_municipios_carencias.R

Para la ejecución del presente análisis, se debe abrir el archivo **17_estimacion_municipios_carencias** disponible en la ruta *Rcodes/2020/17_estimacion_municipios_carencias.R*.

El código proporcionado realiza una serie de procesos para la estimación y calibración de indicadores de carencias a nivel municipal en base a datos de encuestas. Comienza limpiando el entorno de trabajo y cargando las librerías necesarias, como `tidyverse`, `data.table`, y `survey`, además de leer los datos de predicciones y encuestas. Se preparan las bases de datos con variables calculadas para las carencias, incluyendo indicadores de carencias sociales y líneas de pobreza.

Posteriormente, el script filtra y prepara los datos de la encuesta para cada entidad (municipio) específica. Para cada municipio, se realiza un proceso iterativo de simulación donde se actualizan las variables del modelo (como ingreso y carencias sociales) con base en predicciones previas. Se calcula la población total y se ajusta la muestra para realizar estimaciones calibradas utilizando el método de "raking", que ajusta los pesos de la muestra para que coincidan con las estimaciones de la población.

Finalmente, se guarda cada iteración de los resultados calibrados en archivos RDS y se calcula el tiempo total de ejecución. Cada archivo contiene estimaciones para los indicadores de pobreza en cada municipio. El script utiliza técnicas de muestreo y calibración para asegurar que las estimaciones sean consistentes con las características de la población y se gestionan errores potenciales durante el proceso de calibración.

Configuración y carga de librerías

El código comienza limpiando el entorno de trabajo en R para eliminar cualquier objeto residual que pudiera interferir con el análisis, utilizando la función `rm(list = ls())`. A continuación, carga una serie de librerías esenciales: `tidyverse` para manipulación de datos y visualización, `data.table` para el manejo eficiente de grandes conjuntos de datos, `openxlsx` para trabajar con archivos Excel, `magrittr` para el uso del operador `%>%` en la encadenación de operaciones, `haven` para leer y escribir datos en formatos de software estadístico, `labelled` para manejar etiquetas de variables, `sampling` para técnicas de muestreo, `lme4` para modelos lineales y no lineales con efectos mixtos, `survey` para el análisis de datos de encuestas, y `srvyr` para la integración de datos de

encuestas con `dplyr`. Finalmente, el script `benchmarking_indicador.R` se carga usando la función `source()` para incluir funciones adicionales o configuraciones necesarias para el análisis.

```
### Cleaning R environment ###
rm(list = ls())
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(haven)
library(labelled)
library(sampling)
library(lme4)
library(survey)
library(srvyr)
source("../source/benchmarking_indicador.R")
```

Lectura de datos

El código comienza con la lectura de varios archivos de datos necesarios para el análisis. Primero, se cargan las predicciones guardadas en el archivo `predicciones.rds` ubicado en la carpeta de modelos de 2020. Luego, se lee el archivo `encuesta_ampliada.rds`, que contiene datos ampliados de encuestas, y el archivo `encuesta_sta.rds`, que se transforma para incluir la variable `ingreso`, calculada a partir de `ictpc`. Finalmente, se importa el archivo `Lineas_Bienestar.csv` con delimitadores específicos y se convierte la columna `area` en formato de texto. Estos pasos preparan los datos necesarios para realizar el análisis posterior.

```
predicciones <- readRDS("../output/2020/modelos/predicciones.rds")
encuesta_ampliada <- readRDS("../output/2020/encuesta_ampliada.rds")
encuesta_enigh <- readRDS("../input/2020/enigh/encuesta_sta.rds") %>%
  mutate(ingreso = ictpc)

LB <- read.delim(
  "../input/2020/Lineas_Bienestar.csv",
  header = TRUE,
  sep = ";",
  dec = ","
) %>% mutate(area = as.character(area))
```

Índice de Pobreza Multidimensional en la ENIGH

En esta sección del código se calcula el Índice de Pobreza Multidimensional (IPM) utilizando la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH).

Primero, se realiza una combinación de la base de datos `encuesta_enigh` con el archivo `LB` que contiene las líneas de bienestar. Luego, se crean nuevas variables para evaluar la pobreza multidimensional:

1. **`tol_ic`**: Suma de los indicadores de bienestar social y económico (`ic_segsoc`, `ic_ali_nc`, `ic_asalud`, `ic_cv`, `ic_sbv`, `ic_rezedu`).
2. **`pobrea_lp`**: Variable binaria que indica si el ingreso es menor que el umbral de pobreza (lp), asignando un valor de 1 si es el caso, y 0 en caso contrario.
3. **`pobrea_li`**: Variable binaria que indica si el ingreso es menor que el umbral de pobreza extrema (li), asignando un valor de 1 si es el caso, y 0 en caso contrario.
4. **`tol_ic_1`**: Variable binaria que indica si la suma de los indicadores de bienestar (`tol_ic`) es mayor que 0, asignando un valor de 1 si es el caso, y 0 en caso contrario.
5. **`tol_ic_2`**: Variable binaria que indica si la suma de los indicadores de bienestar (`tol_ic`) es mayor que 2, asignando un valor de 1 si es el caso, y 0 en caso contrario.

Estas transformaciones permiten evaluar la pobreza multidimensional en función de los umbrales establecidos y los indicadores de bienestar.

```
##############################################################################
# IPM en la enigh
##############################################################################
encuesta_enigh <- encuesta_enigh %>%
  inner_join(LB) %>%
  mutate(
    tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv + ic_sbv + ic_rezedu,
    pobrea_lp = ifelse(ingreso < lp, 1, 0),
    pobrea_li = ifelse(ingreso < li, 1, 0),
    tol_ic_1 = ifelse(tol_ic > 0, 1, 0),
    tol_ic_2 = ifelse(tol_ic > 2, 1, 0)
  )
```

Preparación de encuesta ampliada

En esta sección, se lleva a cabo la preparación de la base de datos `encuesta_ampliada` para el análisis de pobreza multidimensional. En primer lugar, la base `encuesta_ampliada` se une con el archivo `LB`, que contiene las líneas de bienestar. A continuación, se realizan varias transformaciones importantes en las variables de la encuesta. Se transforman los indicadores relacionados con salud (`ic_asalud`), cobertura de vivienda (`ic_cv`),

servicios básicos (`ic_sbv`), y educación (`ic_rezedu`) en variables binarias, donde un valor de 1 se mantiene como 1 y un valor distinto se convierte en 0. Además, se calcula `tol_ic4`, que representa la suma de estos indicadores de bienestar.

Se incorporan también las predicciones generadas por modelos previos, incluyendo las predicciones de seguridad social (`pred_segsoc`), ingresos (`pred_ingreso`), y el indicador de alimentos no consumidos (`pred_ic_ali_nc`), junto con el error estándar residual (`desv_estandar_residual`). Aunque se ha planificado crear una estructura de directorios para almacenar los resultados de las iteraciones de calibración de carencia por municipio, esta parte del código está comentada y no se ejecuta en este fragmento.

Finalmente, se realiza una limpieza de memoria al eliminar el objeto `predicciones`, y se configuran variables adicionales. Se define un código de entidad específico (`ii_ent`) y se prepara una lista (`list_yks`) que incluye las variables clave para el análisis, tales como indicadores de pobreza y bienestar. Esta preparación es esencial para facilitar el análisis posterior y la organización de los resultados en directorios específicos.

```
encuesta_ampliada <- encuesta_ampliada %>%
  inner_join(LB) %>%
  mutate(
    ic_asalud = ifelse(ic_asalud == 1, 1, 0),
    ic_cv = ifelse(ic_cv == 1, 1, 0),
    ic_sbv = ifelse(ic_sbv == 1, 1, 0),
    ic_rezedu = ifelse(ic_rezedu == 1, 1, 0),
    tol_ic4 = ic_asalud + ic_cv + ic_sbv + ic_rezedu,
    pred_segsoc = predicciones$pred_segsoc,
    pred_ingreso = predicciones$pred_ingreso,
    desv_estandar_residual = predicciones$desv_estandar_residual,
    pred_ic_ali_nc = predicciones$pred_ic_ali_nc
  )

# dir.create("output/2020/iteraciones/mpio_calib_carencia")
# map(
#   paste0(
#     "output/2020/iteraciones/mpio_calib_carencia/",
#     unique(encuesta_enigh$ent)
#   ),
#   ~ dir.create(path = .x)
# )

rm(predicciones)

ii_ent <- "20"
```

```
list_yks <- list(
  c("pobrea_lp"),
  c("pobrea_li"),
  c("tol_ic_1", "tol_ic_2", "ic_segsoc", "ic_ali_nc")
)
```

Iteraciones y calibración

En esta sección del código, se realiza un proceso iterativo para la calibración de estimaciones de pobreza multidimensional a nivel municipal. Este proceso se ejecuta para una lista específica de entidades, representadas por sus códigos (`ii_ent`).

A Descripción General

1. **Inicio del Ciclo de Iteración:**
   - Se comienza un ciclo `for` que itera sobre una lista de códigos de entidades (`ii_ent`), que representan diferentes áreas geográficas para las cuales se calibrarán los modelos.
2. **Configuración y Preparación de Datos:**
   - Para cada entidad (`ii_ent`), se registra el tiempo de inicio y se filtran los datos correspondientes a la entidad actual en `encuesta_ampliada` y `encuesta_enigh`.
   - Se agrupan los datos por municipio y se calcula la población total de cada municipio (`total_mpio`) y la población total a nivel estatal (`tot_pob`).
   - Se crean los vectores `top_caren` y `tx_mun`, que se combinan en `Tx_hat` para la calibración de los modelos.
3. **Iteraciones de Calibración:**
   - Se inicia un ciclo `for` para realizar 200 iteraciones de calibración.
   - En cada iteración:
     - Se simulan nuevas observaciones para las variables `ic_segsoc`, `ic_ali_nc`, y `ingreso` utilizando distribuciones aleatorias basadas en las predicciones previas.
     - Se actualizan las variables de pobreza y bienestar (`pobrea_lp`, `pobrea_li`, `tol_ic_1`, `tol_ic_2`) en función de los nuevos valores generados.
4. **Calibración del Modelo:**
   - Para cada conjunto de variables de interés (`list_yks`), se preparan los datos para la calibración:
     - Se generan variables dummy para los códigos de municipios y se construye un diseño de encuesta usando `as_survey_design`.
     - Se define una fórmula para la calibración y se intenta ajustar el modelo con el método de `raking`.

  - Si ocurre un error durante la calibración, se captura el error y se continúa con la siguiente iteración.

5. **Resumen y Guardado de Resultados:**
   - Los resultados calibrados se agrupan por municipio y se guardan en un archivo `.rds` en un directorio específico para cada entidad y cada iteración.
   - Se limpia la memoria (`gc()`) para liberar espacio.
6. **Registro de Tiempo:**
   - Al final de cada iteración de entidad, se calcula y se imprime el tiempo total transcurrido.

Los resultados de cada iteración se guardan en archivos que permiten la evaluación de la calibración en diferentes áreas geográficas y en diferentes iteraciones, lo que facilita el análisis de la variabilidad y la precisión de las estimaciones de pobreza multidimensional.

```
for(ii_ent in c("03", "06", "23", "04", "01", "22", "27", "25", "18",
                "05", "17", "28", "10", "26", "09", "32", "08", "19", "29",
                "24", "11", "31", "13", "16", "12", "14", "15", "07", "21",
                "30", "20", "02")) {

  cat("#####################################################################\n")
  inicio <- Sys.time()
  print(inicio)

  yks <- unlist(list_yks)
  muestra_post <- encuesta_ampliada %>% filter(ent == ii_ent)

  total_mpio <- muestra_post %>% group_by(cve_mun) %>%
    summarise(den_mpio = sum(factor), .groups = "drop") %>%
    mutate(tot_ent = sum(den_mpio))

  encuesta_sta <- encuesta_enigh %>% filter(ent == ii_ent) %>% na.omit()

  tot_pob <- encuesta_sta %>%
    summarise_at(.vars = yks, .funs = list(
      ~ weighted.mean(., w = fep, na.rm = TRUE) * sum(total_mpio$den_mpio)
    ))

  top_caren <- unlist(as.vector(tot_pob))
  tx_mun <- setNames(total_mpio$den_mpio, paste0("cve_mun_", total_mpio$cve_mun))
  Tx_hat <- c(top_caren, tx_mun)

  for(iter in 1:200) {
    cat("\n municipio = ", ii_ent, "\n\n")
    cat("\n iteracion = ", iter, "\n\n")
```

```r
##########################################
muestra_post <- muestra_post %>%
  mutate(
    ic_segsoc = rbinom(n = n(), size = 1, prob = pred_segsoc),
    ic_ali_nc = rbinom(n = n(), size = 1, prob = pred_ic_ali_nc),
    ingreso = pred_ingreso + rnorm(n = n(), mean = 0, sd = desv_estandar_residual)
    tol_ic = tol_ic4 + ic_segsoc + ic_ali_nc
  ) %>%
  mutate(
    pobrea_lp = ifelse(ingreso < lp, 1, 0),
    pobrea_li = ifelse(ingreso < li, 1, 0),
    tol_ic_1 = ifelse(tol_ic > 0, 1, 0),
    tol_ic_2 = ifelse(tol_ic > 2, 1, 0)
  )

estima_calib_ipm <- list()
for(ii in 1:length(list_yks)) {
  yks <- list_yks[[ii]]

  var_calib <- c(yks, names(tx_mun))
  Xk <- muestra_post %>% select("cve_mun") %>%
    fastDummies::dummy_columns(select_columns = c("cve_mun"), remove_selected_colu

  diseno_post <- bind_cols(muestra_post, Xk) %>%
    mutate(fep = factor) %>%
    as_survey_design(
      ids = upm,
      weights = fep,
      nest = TRUE
    )

  mod_calib <- as.formula(paste0("~ -1 +", paste0(var_calib, collapse = " + ")))
  diseno_calib <- tryCatch({
    calibrate(diseno_post, formula = mod_calib,
              population = Tx_hat[var_calib], calfun = "raking",
              maxit = 50)
  }, error = function(e) {
    message("Error en la calibración: ", yks)
    return(NULL)
  })

  if (is.null(diseno_calib)) {
```

```
        next
      }

      estima_calib_ipm[[ii]] <- diseno_calib %>% group_by(cve_mun) %>%
        summarise_at(.vars = yks, .funs = list(
          ~ survey_mean(., vartype = "var", na.rm = TRUE))
        )
    }

    estima_calib_ipm <- keep(estima_calib_ipm, ~ !is.null(.)) %>%
      reduce(inner_join)

    saveRDS(list(estima_calib = estima_calib_ipm),
            file = paste0("../output/2020/iteraciones/mpio_calib_carencia/",
                          ii_ent, "/iter", iter, ".rds"))
    gc()
  }

  fin <- Sys.time()
  tiempo_total <- difftime(fin, inicio, units = "mins")
  print(tiempo_total)
  cat("#################################################################\n")
}
```

18_estimacion_municipios_carencias_erro

Para la ejecución del presente análisis, se debe abrir el archivo **18_estimacion_municipios_carencias** disponible en la ruta *Rcodes/2020/18_estimacion_municipios_carencias_error.R*.

Este script en R está diseñado para procesar y analizar datos de encuestas sobre pobreza en México para el año 2020. El código comienza limpiando el entorno de trabajo con `rm(list = ls())` y cargando las bibliotecas necesarias (`tidyverse`, `magrittr`, `survey`, `srvyr`, `tidyselect`) para la manipulación de datos y análisis estadístico.

La primera sección del script se enfoca en la lectura de datos. Se cargan dos conjuntos de datos usando `readRDS()`: `encuesta_ampliada` y `encuesta_enigh`, este último con una modificación en la columna `ingreso`. También se lee un archivo CSV con `read.delim()`, que contiene datos sobre las líneas de bienestar, y se convierte la columna `area` a formato de carácter.

En la sección del Índice de Pobreza Multidimensional en la encuesta ENIGH, se realiza una combinación interna (`inner_join()`) entre `encuesta_enigh` y `LB` para añadir variables relacionadas con carencias sociales y pobreza. Se crean nuevas variables en `encuesta_enigh`, tales como `tol_ic` (suma de indicadores de carencia social), `pobrea_lp` (indicador de pobreza por ingresos) y `pobrea_li` (indicador de pobreza extrema por ingresos). Además, se genera un resumen del número de personas por municipio en `N_mpio`. A continuación, el script lee una lista de archivos de estimación de carencia desde un directorio específico, y procesa estos archivos para calcular diversas estadísticas como media, varianza y errores estándar de las estimaciones. Utiliza la función `map_df()` para combinar datos de múltiples archivos, y realiza cálculos de varianza y errores estándar para cada indicador de pobreza. Los resultados se almacenan en una lista y se combinan en un único marco de datos.

Los resultados combinados (`temp`) se enriquecen con información de población de `N_mpio` y se guardan en archivos RDS y Excel utilizando `saveRDS()` y `write.xlsx()`. Se calcula también un resumen a nivel estatal, agregando los resultados por entidad. En la última sección del script, se realiza un análisis gráfico de las estimaciones de pobreza. Se convierte `encuesta_enigh` en un diseño de encuesta usando `as_survey_design()`, y se calculan las estimaciones de pobreza por entidad. Luego, se generan gráficos con `ggplot2` para comparar las estimaciones de carencia entre los datos de la encuesta

ENIGH y las estimaciones de CEPAL. Los gráficos se guardan como archivos PNG utilizando `ggsave()`.

Limpieza del Entorno y Carga de Bibliotecas

En esta sección, el código se ocupa de limpiar el entorno de trabajo y cargar las bibliotecas necesarias para la manipulación y análisis de datos, así como para la visualización. Este proceso garantiza que el entorno esté libre de interferencias de objetos previos y que las herramientas requeridas estén disponibles para ejecutar el resto del script de manera eficiente.

```
rm(list = ls())
library(tidyverse)
library(magrittr)
library(survey)
library(srvyr)
library(tidyselect)
```

- **Limpieza del entorno**: El comando `rm(list = ls())` se utiliza para eliminar todos los objetos en el entorno de trabajo de R. Esto asegura que no haya residuos de datos anteriores que puedan afectar el nuevo análisis.
- **Carga de bibliotecas**:
  - `tidyverse`: Conjunto de paquetes que incluye `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, entre otros. Estos paquetes son esenciales para la manipulación de datos, análisis y visualización.
  - `magrittr`: Proporciona el operador de pipe (`%>%`), facilitando la escritura de código legible y conciso al encadenar funciones.
  - `survey`: Ofrece herramientas para el análisis de datos de encuestas complejas, permitiendo el diseño de muestras y la estimación de estadísticas ponderadas.
  - `srvyr`: Extiende las funcionalidades del paquete `survey` utilizando la sintaxis de `dplyr`, lo que facilita la manipulación y resumen de datos de encuestas.
  - `tidyselect`: Utilizado para facilitar la selección de columnas en operaciones de manipulación de datos dentro del ecosistema de tidyverse.

Este paso es crucial para asegurar un entorno limpio y preparado para realizar análisis de datos precisos y eficientes.

Lectura de Datos En esta sección, el código se encarga de leer y cargar los datos necesarios para el análisis desde archivos RDS y CSV. Estos datos incluyen encuestas ampliadas, encuestas ENIGH y líneas de bienestar.

```
encuesta_ampliada <- readRDS("../../output/2020/encuesta_ampliada.rds")
encuesta_enigh <-
  readRDS("../input/2020/enigh/encuesta_sta.rds") %>%
  mutate(ingreso = ictpc)
```

```
LB <-
  read.delim(
    "../input/2020/Lineas_Bienestar.csv",
    header = TRUE,
    sep = ";",
    dec = ","
  ) %>%
  mutate(area = as.character(area))
```

- **Lectura de encuesta ampliada**:
  - `encuesta_ampliada <- readRDS("../../output/2020/encuesta_ampliada.rds")`: Carga el archivo RDS que contiene la encuesta ampliada. Este archivo se encuentra en la carpeta de salida `output/2020` y contiene datos detallados sobre las encuestas realizadas en 2020.
- **Lectura de encuesta ENIGH**:
  - `encuesta_enigh <- readRDS("../input/2020/enigh/encuesta_sta.rds")`: Carga el archivo RDS que contiene los datos de la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH) de 2020.
  - `mutate(ingreso = ictpc)`: Añade una nueva columna `ingreso` a la encuesta ENIGH, utilizando la columna `ictpc` que representa el ingreso total per cápita.
- **Lectura de líneas de bienestar**:
  - `LB <- read.delim("../input/2020/Lineas_Bienestar.csv", header = TRUE, sep = ";", dec = ",")`: Lee el archivo CSV que contiene las líneas de bienestar. Este archivo se encuentra en la carpeta `input/2020` y contiene información sobre las líneas de pobreza y bienestar.
  - `mutate(area = as.character(area))`: Convierte la columna `area` en tipo de dato carácter, asegurando una correcta manipulación de los datos en análisis posteriores.

Este paso es esencial para preparar y estructurar los datos que se utilizarán en el análisis, asegurando que toda la información necesaria esté disponible y correctamente formateada.

Cálculo de Indicadores de Pobreza

En esta sección, se calculan diversos indicadores de pobreza utilizando los datos de la encuesta ENIGH y las líneas de bienestar. El código une los datos de la encuesta ENIGH con las líneas de bienestar (`LB`) mediante una unión interna, asegurando que solo se conserven las observaciones presentes en ambos conjuntos de datos. A continuación, se calcula el total de carencias sociales (`tol_ic`) sumando las diferentes carencias individuales (`ic_segsoc`, `ic_ali_nc`, `ic_asalud`, `ic_cv`, `ic_sbv`, `ic_rezedu`). Se determinan indicadores binarios de pobreza basados en el ingreso, como `pobrea_lp`,

que indica si un hogar está por debajo de la línea de pobreza (`lp`), y `pobrea_li`, que indica si un hogar está por debajo de la línea de indigencia (`li`). Asimismo, se calculan indicadores de pobreza basados en el número de carencias sociales, como `tol_ic_1`, que indica si un hogar tiene al menos una carencia social, y `tol_ic_2`, que indica si un hogar tiene más de dos carencias sociales. Estos cálculos permiten evaluar diferentes dimensiones de la pobreza, combinando tanto el ingreso como las carencias sociales para ofrecer una visión más completa de la situación socioeconómica de los hogares.

```
encuesta_enigh <- encuesta_enigh %>% inner_join(LB) %>%
mutate(
tol_ic = ic_segsoc + ic_ali_nc + ic_asalud + ic_cv + ic_sbv + ic_rezedu,
pobrea_lp = ifelse(ingreso < lp, 1, 0),
pobrea_li = ifelse(ingreso < li, 1, 0),
tol_ic_1 = ifelse(tol_ic > 0, 1, 0),
tol_ic_2 = ifelse(tol_ic > 2, 1, 0)
)
```

Procesamiento de Resultados Iterativos

Esta sección se encarga de listar y procesar los directorios que contienen los resultados de iteraciones previas, almacenados en archivos. Primero, se listan todos los directorios dentro de la ruta especificada (`../output/2020/iteraciones/mpio_calib_carencia//`) utilizando la función `list.files`, incluyendo sus rutas completas. Luego, se crea un data frame (`list_estiacion`) que contiene las rutas de los directorios y la cantidad de archivos en cada uno de ellos, que se determina aplicando la función `length` a la lista de archivos en cada directorio. Se filtran los directorios para conservar solo aquellos que contienen al menos un archivo de iteración (es decir, aquellos con `iter > 0`). Finalmente, se inicializa una lista vacía (`resul_ent_ipm`) para almacenar los resultados procesados de estas iteraciones.

```
  list_estiacion <- list.files("../output/2020/iteraciones/mpio_calib_carencia//", fu
  list_estiacion <- data.frame(list_estiacion, iter = list_estiacion %>% map_dbl(~li
    filter(iter > 0)

  resul_ent_ipm <- list()
```

Combinación de Resultados Iterativos

Esta sección combina los resultados de las iteraciones en un único conjunto de datos. Primero, se recorre cada directorio listado en `list_estiacion$list_estiacion`. Para cada directorio, se obtiene la lista de archivos dentro de él y se leen los datos contenidos en cada archivo utilizando la función `readRDS`, combinando los datos en un solo data frame con `map_df`. Luego, se agrupan los datos por `cve_mun` y se calculan diversas estadísticas: las medias (`summarise_at(vars(!matches("var")), mean)`), el

conteo (`tally()`), las varianzas (`summarise_at(vars(!matches("var")), var)`), y las medias de las varianzas (`summarise_at(vars(matches("var")), mean)`).

Posteriormente, los datos se transforman y combinan para calcular las varianzas totales y los errores estándar (`transmute(cve_mun, Indicador, var = Ubar + (1 + 1 / n) * B, ee = sqrt(var))`). Los resultados se reestructuran utilizando `pivot_wider` para obtener el formato deseado y se guardan en la lista `resul_ent_ipm` para cada entidad procesada. Finalmente, se unen los diversos data frames generados (`dat_n`, `dat_estima`, `dat_var`, `var_est`) en un único data frame por entidad y se almacenan en la lista de resultados.

```
for (ii_ent in list_estiacion$list_estiacion) {
  archivos <- list.files(ii_ent, full.names = TRUE)
  datos <- map_df(archivos, ~ readRDS(.x)$estima_calib)

  dat_estima <-
    datos %>% group_by(cve_mun) %>% summarise_at(vars(!matches("var")), mean) %>% dat
  dat_n <- datos %>% group_by(cve_mun) %>% tally()
  dat_B <-
    datos %>% group_by(cve_mun) %>% summarise_at(vars(!matches("var")), var) %>% data

  dat_Ubar <-
    datos %>% group_by(cve_mun) %>% summarise_at(vars(matches("var")), mean) %>% data
  names(dat_Ubar) <- gsub("_var", "", names(dat_Ubar))

  dat_var <-
    dat_B %>% gather(key = "Indicador", value = "B", -cve_mun) %>%
    inner_join(dat_Ubar %>% gather(key = "Indicador", value = "Ubar", -cve_mun)) %>%
    inner_join(dat_n)

  var_est <-
    dat_var %>% transmute(cve_mun,
                          Indicador,
                          var = Ubar + (1 + 1 / n) * B,
                          ee = sqrt(var)) %>%
    pivot_wider(
      id_cols = "cve_mun",
      names_from = "Indicador",
      values_from = c("var", "ee"),
      values_fill = 0
    )

  dat_var <-
    pivot_wider(
```

```
      data = dat_var,
      id_cols = "cve_mun",
      names_from = "Indicador",
      values_from = c("B", "Ubar"),
      values_fill = 0
    )

resul_ent_ipm[[ii_ent]] <-
    dat_n %>% inner_join(dat_estima) %>% inner_join(dat_var) %>% inner_join(var_est)
}
```

Guardado de Resultados

En esta sección se guardan los resultados combinados en archivos RDS y Excel. Primero, se unen todos los data frames contenidos en la lista `resul_ent_ipm` utilizando `bind_rows()`. A continuación, se hace un `inner_join` con el data frame `N_mpio` y se seleccionan las columnas relevantes: `ent`, `cve_mun`, los indicadores de pobreza y carencias (`pobrea_lp`, `pobrea_li`, `tol_ic_1`, `tol_ic_2`, `ic_segsoc`, `ic_ali_nc`), `N_pers_mpio` y las columnas que coinciden con `ee`. Este conjunto de datos se guarda en un archivo RDS con la función `saveRDS`.

Luego, se guarda el mismo conjunto de datos en un archivo Excel utilizando la función `write.xlsx` de la biblioteca `openxlsx`. Se genera un nombre de archivo que incluye la fecha actual obtenida con `Sys.Date()`.

Para obtener un resumen a nivel de entidad, se seleccionan las columnas relevantes (`ent`, `pobrea_lp`, `pobrea_li`, `tol_ic_1`, `tol_ic_2`, `ic_segsoc`, `ic_ali_nc`, `N_pers_mpio`) y se agrupan por `ent`. Se calculan las medias ponderadas de los indicadores de pobreza y carencias, ponderadas por el número de personas en cada municipio (`N_pers_mpio`). Este resumen se guarda en otro archivo Excel, también con un nombre que incluye la fecha actual.

```
temp <- resul_ent_ipm %>% bind_rows()
temp %<>% inner_join(N_mpio) %>%
  select(
    ent,
    cve_mun,
    "pobrea_lp",
    "pobrea_li",
    "tol_ic_1",
    "tol_ic_2",
    "ic_segsoc",
    "ic_ali_nc",
    N_pers_mpio,
```

```
    matches("ee")
  )

temp %>% saveRDS("../output/Entregas/2020/result_mpios_carencia.RDS")

openxlsx::write.xlsx(
  temp,
  paste0(
    "../output/Entregas/2020/estimacion_numicipal_carencia_",
    Sys.Date(),
    ".xlsx"
  )
)

temp2 <- temp %>%
  select(ent, pobrea_lp:ic_ali_nc, N_pers_mpio) %>%
  group_by(ent) %>%
  summarise(across(
    c(
      "pobrea_lp",
      "pobrea_li",
      "tol_ic_1",
      "tol_ic_2",
      "ic_segsoc",
      "ic_ali_nc"
    ),
    ~ sum(.x * N_pers_mpio) / sum(N_pers_mpio)
  ))

openxlsx::write.xlsx(
  temp2,
  paste0(
    "../output/Entregas/2020/estimacion_estado_carencia_",
    Sys.Date(),
    ".xlsx"
  )
)
```

Visualización de Resultados

En esta sección se generan y guardan gráficos que comparan los indicadores de pobreza entre diferentes fuentes de datos. Primero, se crea un diseño de encuesta utilizando

la base de datos `encuesta_enigh` con la función `as_survey_design`, eliminando previamente los valores faltantes. Con este diseño, se calculan estimaciones directas de los indicadores de pobreza y carencias (`pobrea_lp`, `pobrea_li`, `tol_ic_1`, `tol_ic_2`, `ic_segsoc`, `ic_ali_nc`) para cada entidad (`ent`), incluyendo intervalos de confianza (`ci`).

Los indicadores de interés se almacenan en el vector `ind`, y se itera sobre cada uno de ellos para generar los gráficos correspondientes. Para cada indicador, se filtran y preparan los datos de comparación entre las estimaciones directas (ENIGH) y las estimaciones calculadas (CEPAL), utilizando `inner_join` y funciones de manipulación de datos de `dplyr` y `tidyr`. Se calculan los límites inferior y superior de los intervalos de confianza y se reorganizan los datos para su visualización.

Se genera un gráfico utilizando `ggplot2`, con las entidades en el eje x y las proporciones estimadas en el eje y. Las estimaciones de ENIGH y CEPAL se diferencian por colores. Se añaden barras de error para los intervalos de confianza de ENIGH y se ajustan los elementos visuales del gráfico para mejorar la presentación. Finalmente, se guarda cada gráfico como un archivo PNG con un nombre que incluye el indicador correspondiente.

```
diseno <- encuesta_enigh %>% na.omit() %>%
  as_survey_design(ids = upm,
                   weights = fep,
                   nest = TRUE)

estimad_dir <- diseno %>% group_by(ent) %>%
  summarise_at(
    .vars = c(
      "pobrea_lp",
      "pobrea_li",
      "tol_ic_1",
      "tol_ic_2",
      "ic_segsoc",
      "ic_ali_nc"
    ),
    .funs = list(dir = ~ survey_mean(., vartype = "ci", na.rm = TRUE))
  )

ind <-
  c("pobrea_lp",
    "pobrea_li",
    "tol_ic_1",
    "tol_ic_2",
    "ic_segsoc",
    "ic_ali_nc")
```

```
for (ii_ipm in 1:6) {
  ii_ipm <- ind[ii_ipm]
  paso <- paste0("(_|\\b)", ii_ipm, "(_|\\b)")
  dat_plot <- inner_join(estimad_dir, temp2) %>%
    select(ent, matches(paso))

  dat_lim <- dat_plot %>% select(ent, matches("upp|low"))
  names(dat_lim) <- c("ent", "Lim_Inf", "Lim_Sup")

  dat_plot %<>% select(-matches("upp|low")) %>%
    gather(key = "Origen", value = "Prop", -ent) %>%
    mutate(Origen = ifelse(grepl(pattern = "dir", x = Origen), "ENIGH", "Estimación C
    inner_join(dat_lim)

  gg_plot <-
    ggplot(data = dat_plot, aes(x = ent, y = Prop, color = Origen)) +
    labs(
      x = "",
      y = "Estimación",
      color = "",
      title = paste0("Estimación de la carencia 2020 -", ii_ipm)
    ) +
    theme_bw(20) +
    geom_jitter(width = 0.3) +
    theme(legend.position = "bottom",
          plot.title = element_text(hjust = 0.5)) +
    scale_y_continuous(labels = scales::percent_format(scale = 100))

  gg_plot <- gg_plot +
    geom_errorbar(
      data = dat_plot %>% filter(Origen == "ENIGH"),
      aes(ymin = Lim_Inf, ymax = Lim_Sup, x = ent),
      width = 0.2,
      linewidth = 1
    ) +
    scale_color_manual(
      breaks = c("ENIGH", "Estimación CEPAL"),
      values = c("red", "blue3")
    ) +
    theme(
      legend.position = "bottom",
      axis.title = element_text(size = 10),
```

```
      axis.text.y = element_text(size = 10),
      axis.text.x = element_text(
        angle = 90,
        size = 8,
        vjust = 0.3
      ),
      legend.title = element_text(size = 15),
      legend.text = element_text(size = 15)
    )

  print(gg_plot)
  ggsave(
    plot = gg_plot,
    width = 16,
    height = 9,
    filename = paste0("../output/Entregas/2020/plot_uni/ipm_", ii_ipm, ".png")
  )
}
```

19_Mapas_carencias.R

Para la ejecución del presente análisis, se debe abrir el archivo **19_Mapas_carencias.R** disponible en la ruta *Rcodes/2020/19_Mapas_carencias.R*.

El código proporciona un procedimiento para la visualización de estimaciones de carencias y errores de estimación en un formato de mapa utilizando `tmap` y datos geoespaciales. Primero, limpia el entorno de trabajo en R y establece un límite para el uso de memoria RAM. Luego, carga las librerías necesarias y los datasets relevantes, incluyendo los resultados de estimación de carencia y un shapefile de México que se ajusta para incluir las claves municipales y regionales.

A continuación, define los cortes para la clasificación de los datos en porcentajes y genera mapas temáticos para cada indicador de carencia (como pobreza por ingresos, carencias sociales, etc.). Utiliza el objeto `ShapeDAM` combinado con `ipm_mpios` para crear mapas que visualizan estos indicadores con una paleta de colores verde. Cada mapa se guarda como una imagen PNG con alta resolución.

Finalmente, el código realiza un proceso similar para visualizar los errores de estimación asociados a cada indicador. Los cortes para los errores se definen de manera diferente, y se generan mapas temáticos que muestran la magnitud de estos errores. Cada mapa se guarda en formato PNG con especificaciones similares.

Limpieza del Entorno y Carga de Bibliotecas

Este fragmento de código realiza una serie de preparativos esenciales antes de ejecutar un análisis de datos. Primero, limpia el entorno de trabajo eliminando todos los objetos existentes en la memoria y fuerza la recolección de basura para liberar espacio en el sistema, asegurando que no queden residuos de ejecuciones anteriores que puedan afectar la ejecución actual. Luego, carga una serie de bibliotecas necesarias para el análisis y la visualización de datos. Entre las bibliotecas cargadas se encuentran `dplyr` y `data.table` para la manipulación de datos, `haven` para la importación de datos, `magrittr` para el uso de operadores encadenados, `stringr` para el manejo de cadenas de texto, `openxlsx` para la manipulación de archivos Excel, y `tmap` y `sf` para la creación de mapas y el manejo de datos espaciales. Además, se redefine la función `select` para asegurarse de que se utilice la versión de `dplyr`. Finalmente, el código establece un límite de memoria

RAM de 250 GB para la ejecución del script, lo cual es crucial para evitar problemas de memoria durante el análisis de grandes volúmenes de datos.

```
rm(list = ls())
gc()

#################
### Libraries ###
#################
library(dplyr)
library(data.table)
library(haven)
library(magrittr)
library(stringr)
library(openxlsx)
library(tmap)
library(sf)
select <- dplyr::select

###------------ Definiendo el límite de la memoria RAM a emplear ------------###

memory.limit(250000000)
```

Lectura de Datos Primero, carga un archivo de resultados de pobreza a nivel municipal en formato RDS, utilizando la función `readRDS`. El archivo cargado, `ipm_mpios`, contiene los resultados de los indicadores de pobreza a nivel municipal que se han generado previamente.

En segundo lugar, lee un archivo de forma (shapefile) en formato `.shp` utilizando la función `read_sf` de la biblioteca `sf`. Este archivo, ubicado en la carpeta "../shapefile/2020/", contiene datos espaciales que describen la geografía de México para el año 2020. Una vez cargado el shapefile, el código realiza dos transformaciones en el objeto `ShapeDAM`: extrae el código de entidad (`ent`) a partir de los primeros dos dígitos del campo `CVEGEO`, y elimina la columna `CVEGEO`, ya que no es necesaria para el análisis posterior.

Finalmente, define un vector `cortes` que contiene los puntos de corte para la clasificación de datos en intervalos porcentuales (0%, 15%, 30%, 50%, 80%, 100%). Estos cortes serán utilizados para categorizar o visualizar los datos de pobreza en diferentes rangos de intensidad.

```
ipm_mpios <-
  readRDS("../output/Entregas/2020/result_mpios_carencia.RDS")
```

```
ShapeDAM <- read_sf("../shapefile/2020/MEX_2020.shp")
ShapeDAM %<>% mutate(cve_mun = CVEGEO ,
                     ent = substr(cve_mun, 1, 2),
                     CVEGEO = NULL)

cortes <- c(0,  15, 30, 50, 80, 100 )/100
```

Mapas de las carencias estimadas

Primero, se crea un objeto `P1_mpio_norte` utilizando `tm_shape`, que carga un mapa basado en el shapefile `ShapeDAM` y une este mapa con los datos de pobreza `ipm_mpios` a través de la columna `cve_mun`. Los datos de pobreza son incorporados al mapa utilizando un `left_join`.

Se define un vector `ind` que contiene los nombres de los indicadores de pobreza que se desean visualizar en los mapas. Estos indicadores incluyen varias medidas de pobreza y carencias sociales.

Luego, se ejecuta un bucle `for` que recorre cada indicador en `ind`. Para cada indicador (`yks_ind`), se crea un mapa temático (`Mapa_I`) usando `tm_polygons`. Este mapa visualiza los datos de pobreza con una paleta de colores verde (`palette = "Greens"`) y clasifica los datos en intervalos definidos por el vector `cortes`. El título del mapa incluye el nombre del indicador actual.

La función `tm_layout` ajusta la presentación del mapa, configurando la leyenda para que sea visible, con un tamaño de texto específico, y colocada en la posición izquierda. La leyenda se presenta en una disposición vertical y con estilos de fuente en negrita.

Finalmente, el mapa se guarda en un archivo PNG utilizando `tmap_save`, con un tamaño de imagen de 4000x3000 píxeles y una relación de aspecto de 0. El archivo se nombra de acuerdo con el indicador de pobreza actual y se guarda en la carpeta "../output/Entregas/2020/mapas/".

```
P1_mpio_norte <-
  tm_shape(ShapeDAM %>%
             left_join(ipm_mpios,   by = "cve_mun"))
names(ipm_mpios)

ind <- c("pobrea_lp", "pobrea_li","tol_ic_1","tol_ic_2","ic_segsoc",
         "ic_ali_nc")


for(yks_ind in ind){

  Mapa_I <-
```

```
    P1_mpio_norte + tm_polygons(
      breaks  = cortes,
      yks_ind,
      # style = "quantile",
      title =  paste0("Estimación de la carencia 2020 -", yks_ind ),
      palette = "Greens",
      colorNA = "white"
    ) + tm_layout(legend.show = TRUE,
                  #legend.outside = TRUE,
                  legend.text.size =  1.5,
                  legend.outside.position = 'left',
                  legend.hist.width = 1,
                  legend.hist.height = 3,
                  legend.stack = 'vertical',
                  legend.title.fontface = 'bold',
                  legend.text.fontface = 'bold')


  tmap_save(
    Mapa_I,
    filename =  paste0("../output/Entregas/2020/mapas/carencia_", yks_ind, ".png"),
    width = 4000,
    height = 3000,
    asp = 0
  )


}
```

Mapas del error de la estimación de las carencias

Primero, se redefine el vector `cortes` para establecer los intervalos en los que se clasificarán los datos de error de estimación. Los cortes incluyen valores como 0%, 1.5%, 10%, 25%, 50% y 100%.

Luego, se genera un vector `ind_ee` que contiene los nombres de las variables de error de estimación, obtenidos al anteponer "ee_" a cada indicador de pobreza en `ind`. Por ejemplo, si `ind` incluye "pobrea_lp", entonces `ind_ee` incluirá "ee_pobrea_lp".

A continuación, se ejecuta un bucle `for` que recorre cada indicador de error de estimación en `ind_ee`. Para cada indicador (`yks_ind`), se crea un mapa temático (`Mapa_I_ee`) usando `tm_polygons`. Este mapa visualiza el error de estimación con la paleta de colores verde (`palette = "Greens"`) y clasifica los datos en intervalos

definidos por el vector `cortes`. El título del mapa incluye el nombre del indicador de error de estimación actual.

La función `tm_layout` ajusta la presentación del mapa, configurando la leyenda para que sea visible, con un tamaño de texto específico y ubicada en la posición izquierda. La leyenda se presenta en una disposición vertical y con estilos de fuente en negrita.

Finalmente, el mapa se guarda en un archivo PNG usando `tmap_save`, con un tamaño de imagen de 4000x3000 píxeles y una relación de aspecto de 0. El archivo se nombra según el indicador de error de estimación actual y se guarda en la carpeta "../output/Entregas/2020/mapas/".

```
cortes <- c(0,  1,5,10, 25,  50,  100 )/100

ind_ee <- paste0("ee_", ind)

for(yks_ind in ind_ee){

  Mapa_I_ee <-
  P1_mpio_norte + tm_polygons(
    breaks  = cortes,
    yks_ind,
    # style = "quantile",
    title = paste0("Error de estimación de la carencia 2020 - ", yks_ind),
    palette = "Greens",
    colorNA = "white"
  ) + tm_layout(legend.show = TRUE,
                #legend.outside = TRUE,
                legend.text.size =  1.5,
                legend.outside.position = 'left',
                legend.hist.width = 1,
                legend.hist.height = 3,
                legend.stack = 'vertical',
                legend.title.fontface = 'bold',
                legend.text.fontface = 'bold')


tmap_save(
  Mapa_I_ee,
  filename =  paste0("../output/Entregas/2020/mapas/carencia_", yks_ind, ".png"),
  width = 4000,
  height = 3000,
  asp = 0
)
```

```
}
```

20_estimacion_directa_carencias.R

Para la ejecución del presente archivo, debe abrir el archivo **20_estimacion_directa_carencias.R** disponible en la ruta *Rcodes/2020/20_estimacion_directa_carencias.R*.

Descripción General del Código Este código está diseñado para procesar y analizar datos de encuestas con el objetivo de generar estimaciones de carencia para diferentes entidades dentro de un país. A continuación, se detalla cada sección del código:

Preparación del Entorno Al inicio, el código limpia el entorno de trabajo de R mediante el siguiente bloque de código:

```
rm(list = ls())
```

Esto elimina todos los objetos existentes en el espacio de trabajo, asegurando que el análisis se realice en un entorno limpio y sin interferencias de datos anteriores.

Carga de Bibliotecas Se cargan varias bibliotecas necesarias para el análisis de datos mediante el bloque de código:

```
library(tidyverse)
library(data.table)
library(openxlsx)
library(magrittr)
library(haven)
library(labelled)
library(sampling)
library(lme4)
library(survey)
library(srvyr)
```

Estas bibliotecas proporcionan herramientas para la manipulación de datos, manejo de archivos, técnicas de muestreo y análisis de encuestas.

Lectura y Preparación de Datos Se lee un archivo de datos en formato RDS que contiene la encuesta ampliada y se prepara la encuesta intercensal con el siguiente código:

```
encuesta_ampliada <- readRDS("output/2020/encuesta_ampliada.rds")

encuesta_ampliada %<>% mutate(
  ic_asalud = ifelse(ic_asalud == 1, 1,0),
  ic_cv = ifelse(ic_cv == 1, 1,0),
  ic_sbv = ifelse(ic_sbv == 1, 1,0),
  ic_rezedu = ifelse(ic_rezedu == 1, 1,0)
)
```

Aquí, se ajustan ciertos indicadores para que solo tomen valores binarios (0 o 1).

Creación de Directorios y Definición de Variables Se crea un directorio para almacenar los resultados de estimación y se define un vector con códigos de entidades:

```
dir.create("output/2020/estimacion_dir")

c("03", "06", "23", "04", "01", "22", "27", "25", "18",
  "05", "17", "28", "10", "26", "09", "32", "08", "19", "29",
  "24", "11", "31", "13", "16", "12", "14", "15", "07", "21",
  "30",  "20" ,"02")
```

Análisis y Estimación Se realiza un bucle `for` que itera sobre cada código de entidad con el siguiente bloque de código:

```
for(ii_ent in c("03", "06", "23", "04", "01", "22", "27", "25", "18",
  "05", "17", "28", "10", "26", "09", "32", "08", "19", "29",
  "24", "11", "31", "13", "16", "12", "14", "15", "07", "21",
  "30",  "20" ,"02" )){

  cat("##############################################################################
  inicio <- Sys.time()
  print(inicio)

  muestra_post = encuesta_ampliada %>% filter(ent == ii_ent)

  cat("\n Estado = ", ii_ent,"\n\n")

  diseno_post <-  muestra_post %>%
    mutate(fep = factor) %>%
    as_survey_design(
```

```
      ids = upm,
      weights = fep,
      nest = TRUE,
      # strata = estrato
    )

  estima_carencia <-  diseno_post %>% group_by(cve_mun) %>%
    summarise_at(.vars = yks, .funs = list(
      ~ survey_mean(., vartype ="var", na.rm = TRUE))
    )

  saveRDS(estima_carencia,
          file = paste0( "output/2020/estimacion_dir/estado_",
                         ii_ent,".rds"))
  gc(TRUE)
  fin <- Sys.time()
  tiempo_total <- difftime(fin, inicio, units = "mins")
  print(tiempo_total)
  cat("##############################################################################
}
```