SEDESOL 2018

Mónica Zamudio López

Título del proyecto: USO DE DATOS MASIVOS PARA LA EFICIENCIA DEL ESTADO Y LA INTEGRACIÓN REGIONAL

Clave: ATN/OC 15822-RG

Puesto: Científico de Datos Junior

Recolección y limpieza de información

Entregable número: 2

Acrónimo del proyecto:	Estimación de Ingreso
Nombre completo del proyecto:	USO DE DATOS MASIVOS PARA LA EFICIENCIA DEL ESTADO Y LA INTEGRACIÓN REGIONAL
Referencia:	ATN/OC 15822-RG
URL del Proyecto:	http://www.plataformapreventiva.gob.mx

Tipo de Entregable:	Reporte (R)
Fecha de Entrega Contractual:	Agosto - 2018
Fecha de Entrega	17 de agosto de 2018
Número de Páginas:	66
Keywords:	estimación ingreso ciencia datos ingesta
Autor:	Mónica Zamudio López, Laboratorio de Datos, SEDESOL

Resumen

Este proyecto nace de la intención de la Secretaría de Desarrollo Social de construir una metodología de focalización más eficiente y así distribuir mejor los recursos de los programas sociales. Para el desarrollo de esta metodología se tomó en cuenta el problema de sub y sobrerreportaje, considerando distintas especificaciones para construir redes bayesianas. Se utilizaron como principales fuentes de datos la Encuesta de Características Socioeconómicas de los Hogares (ENCASEH), su módulo de verificaciones domiciliarias, y los datos de la Encuesta Nacional de Ingreso y Gasto de los Hogares (ENIGH).

Una vez concluído el proceso de obtención e integración de estos datos, se procedió a la exploración y modelado. A la par, se comenzó a desarrollar la infraestructura para que el flujo de datos fuera reproducible hasta esta etapa, y a la vez las poblaciones descritas por los datos fueran lo más comparables posible.

En este documento se detalla parte del desarrollo que permitió el análisis posterior, así como los resultados de ese análisis.

Índice general

Lista de Acrónimos

DGGPB Dirección General de Geoestadística y Padrones de Beneficiarios

DGAE Dirección General Adjunta de Análisis Espacial

DGAIP Dirección General Adjunta de Integración de Padrones

CUIS Cuestionario Único de Información Socioeconómica

SIFODE Sistema de Focalización de Desarrollo

SEDESOL Secretaría de Desarrollo Social

ENIGH Encuesta Nacional de Ingresos y Gastos de los Hogares

PEA Población Económicamente Activa

LGDS Ley General de Desarrollo Social

INEGI Instituto Nacional de Estadística y Geografía

SISI Sistema de Información Social Integral

PUB Padrón Universal de Beneficiarios

AWS Amazon Web Services

1. Introducción

La Secretaría de Desarrollo Social opera más de 20 programas a nivel federal. Dichos programas están diseñados con el fin específico de incrementar ciertos derechos sociales de la población objetivo. Algunos de esos programas son de cobertura universal[1], mientras que otros utilizan distintas estrategias de focalización. Para tener posibilidad de evaluar el impacto y la eficiencia en el gasto social, ambos tipos de programas requieren de una actualización fidedigna de la información con la que cuentan sobre la población atendida. Además, los programas que hacen focalización generalmente requieren de esa información previo a iniciar operaciones. En ambos casos existe un reto difícil de paliar: tanto las posibles beneficiarias como las personas que operan los programas pueden tener incentivos a sub o sobrereportar ciertos aspectos de su información socioeconómica.

Este proyecto busca generar una solución tecnológica a este tipo de problemas, mejorando así la calidad de los datos con los que se cuenta para dirigir la política social. Utilizando diversos insumos proveídos tanto por la DGGPB como por instancias externas, nos proponemos a construir modelos para corregir por ese sub y sobrereportaje latente. En este documento se presentan las herramientas utilizadas para analizar los datos, así como el flujo de datos diseñado para la reproducibilidad y portabilidad del proceso.

[1] Se dice que un programa es de cobertura universal cuando busca atender a prácticamente toda la población alrededor del lugar de operación del programa. Por otro lado, los programas que utilizan estrategias de focalización buscan atender a la población que cumple con ciertas características específicas.

2. Modelos gráficos

Marco de análisis

Como mencionado en [4], el problema de reportaje incorrecto puede tratarse como un problema de datos faltantes, siguiendo el marco de análisis planteado por [3]. Para este caso, utilizamos el módulo de verificaciones domiciliarias de la ENCASEH, el cual asumimos que nos da las verdaderas características de los hogares. Ajustamos entonces una red bayesiana siguiendo el planteamiento de [2] a la colección de datos que contienen variables reportadas y verificadas. De esta forma, es posible obtener hacer inferencia sobre el ingreso de los hogares para cualquier estimación de ingreso que se proponga, a la vez que se logra capturar la correlación entre las variables verificadas y las reportadas, mejorando la incertidumbre.

Algunas consideraciones importantes

A pesar de que todos los programas utilizan el CUIS como insumo base para sus levantamientos socioeconómicos, estos pueden ser complementados con módulos de preguntas adicionales. Actualmente, los programas sociales operan de una forma relativamente independiente, sin demasiados mecanismos de coordinación en términos de las características de estos levantamientos. Así, es importante asegurarse de que la población de los cuestionarios que sí cuentan con un módulo de verificación sea comparable con la población de los cuestionarios que no, y por ende el modelo no va a tener sesgos sistemáticos que impidan hacer inferencia de forma correcta.

Por otro lado, existen muchas estructuras gráficas distintas que podrían ajustar correctamente la distribución conjunta observada en los datos, pero algunas de ellas pueden ser significativamente más

complejas que otras. En la medida de lo posible, es importante buscar estructuras que no tengan contradicciones teóricas importantes, y partir de un modelo parsimonioso para evitar el sobreajuste. Tomando eso en cuenta, se utilizó un parámetro de regularización relativamente bajo ($k = 20$), y dos restricciones a la estructura de la gráfica:

- Si hay una arista entre una variable reportada y una verificada, la consecuente debe de ser la reportada.
- No pueden existir aristas entre variables reportadas.

Un segundo recurso que se puede utilizar para simplificar la estructura de la gráfica es la construcción y agregación de nuevas variables que garanticen independencia condicional entre los distintos nodos. Con ese propósito, construimos un flujo de datos dedicado a incorporar variables a nivel municipal de forma estandarizada y escalable, que explicamos con mayor detalle en Capítulo 3

Por último, sabemos que este análisis está pensado para ser utilizado en la aplicación de los cuestionarios en tiempo real, por lo que debe de ser completamente replicable y portable entre distintos dispositivos. Estas consideraciones nos llevan a pensar que debemos utilizar un ambiente contenido con la menor cantidad posible de dependencias, por lo que debemos construir una imagen a partir de la cual puedan levantarse contenedores que ejecuten el proceso de predicción.[1]

[1] Para mayor referencia sobre estas consideraciones de infraestructura, ver [1]

3. El proceso de ETL

Buscando que el modelo pueda ser reentrenado periódicamente, diseñamos un proceso de ETL que obtiene los datos, los limpia y aplica transformaciones para modificar variables o crear variables nuevas. Este proceso se lleva a cabo de forma orquestada en un servidor EC2 de AWS, haciendo peticiones a una base de datos RDS alojada en un servidor externo. En la medida de lo posible, el código está optimizado para efectuar la mayor cantidad de operaciones posibles con los recursos del RDS en lugar de utilizar los recursos del servidor de orquestación.

El proceso de ETL tiene tres fases: primero, los datos se extraen de la base de datos cruda y se efectúan procesos sencillos de limpieza de variables y corrección de tipos. Se procede a hacer una recodificación de variables para mayor comparabilidad entre poblaciones, y finalmente se crean variables nuevas a partir de los datos limpios.

Limpieza

El proceso de limpieza se orquesta como una tarea de Luigi que ejecuta un código de R con el nombre de la tabla como parámetro de entrada, así como datos que permiten la conexión a la base de datos. Utilizamos R porque sus herramientas de manipulación de datos facilitan una gran cantidad de operaciones y permiten escribir código extremadamente legible. Este código de R, a su vez, busca en un directorio predeterminado códigos de limpieza específicos para la tabla en cuestión, que pueden estar escritos en R o en SQL. La limpieza se lleva a cabo como una operación remota hacia la base de datos, y el resultado queda guardado en una tabla temporal.

A pesar de que tenemos códigos de limpieza específicos para cada tabla, se diseñó un estándar para facilitar las llamadas a cada programa, sobre todo en el caso de los códigos escritos en R. Así, cada

código de limpieza ejecuta una sola función, llamada *make_clean*.

Recodificación

Para garantizar mayor comparabilidad y facilitar el entrenamiento, construimos un proceso de recodificación, que a través de uniones entre tablas permite homologar los nombres y valores de las variables que describen la misma característica de conjuntos de datos distintos. Así, se toma el resultado de la limpieza y se buscan las variables para las cuales hay códigos nuevos. Se hacen uniones entre la tabla limpia y la tabla auxiliar de códigos (mostrada en la figura abajo), explotando las capacidades de evaluación no estándar que permite R como se puede apreciar en el código expuesto en el anexo.

Para reducir la carga de procesamiento de la base de datos, cada recodificación se guarda de forma temporal y se usa ese resultado para hacer la siguiente recodificación. Una vez recodificada toda la tabla, se escribe el resultado temporal al esquema limpio.

Figura 3.1: Estructura general de la tabla auxiliar de recodificación. Cada fila representa un valor específico de la variable indicada por la columna "variable_inicial", que forma parte de la tabla cruda especificada en la columna "tabla".

tabla	variable_inicial	variable_final	valor_inicial	valor_final	nombre_valor_final
cuis_historico_se_integrante	c_ult_gra	aprobo_ultgrado	[NULL]	[NULL]	[NULL]
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	1.0000	1.0000	kinder o preescolar
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	2.0000	2.0000	primaria
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	3.0000	3.0000	secundaria
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	4.0000	4.0000	preparatoria o bachillerato
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	5.0000	5.0000	normal
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	6.0000	6.0000	carrera técnica o comercial
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	7.0000	6.0000	carrera técnica o comercial
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	8.0000	6.0000	carrera técnica o comercial
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	9.0000	7.0000	profesional
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	10.0000	8.0000	posgrado (maestría o doctorado)
cuis_historico_se_integrante	c_ult_nivel	aprobo_ultnivel	99.0000	0.0000	ninguno
cuis_historico_se_integrante	asis_esc	asiste_escuela	1.0000	1.0000	si
cuis_historico_se_integrante	asis_esc	asiste_escuela	2.0000	0.0000	no
cuis_historico_se_integrante	trab_no_re	con_sueldo	1.0000	1.0000	Sí
cuis_historico_se_integrante	trab_no_re	con_sueldo	2.0000	0.0000	No
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	1.0000	1.0000	Trabajó
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	2.0000	2.0000	Tenía trabajo pero no trabajó
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	3.0000	3.0000	Estudió y trabajó
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	4.0000	4.0000	No trabajó ni buscó trabajo
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	5.0000	5.0000	Buscó trabajo
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	6.0000	6.0000	Estudió
cuis_historico_se_integrante	c_con_tra	realiza_act_lab	7.0000	7.0000	Realizó quehaceres domésticos
cuis_historico_se_integrante	c_ver_con_trab	realiza_act_nolab	1.0000	1.0000	Vender o hacer algún producto o realizar tra
cuis_historico_se_integrante	c_ver_con_trab	realiza_act_nolab	2.0000	2.0000	Prestar o dar algún servicio a cambio de un
cuis_historico_se_integrante	c_ver_con_trab	realiza_act_nolab	3.0000	3.0000	Realizar actividades agropecuarias por su cu
cuis_historico_se_integrante	c_ver_con_trab	realiza_act_nolab	4.0000	4.0000	Ayudar en las tierras de un familiar u otra pe

Creación de variables nuevas

Una vez escrita la tabla limpia a su esquema correspondiente, se utilizan esos datos para la creación de variables nuevas. La creación de variables nuevas está definida en un flujo de datos separado al proceso de ETL, buscando facilitar la idempotencia de las transformaciones. Así, las tablas de

variables nuevas se escriben en un schema distinto en la base de datos, al que llamamos *features*. Este flujo de datos se lleva a través del mismo orquestador, utilizando un archivo auxiliar de tipo *YAML* para definir las dependencias que tienen estas tablas de nuevas variables. Estas dependencias pueden ser tablas limpias, tablas de resultados de modelos u otras tablas del mismo esquema de *features*. El orquestador principal, llamado *Features Pipeline*, recibe como parámetro el nombre de la tabla de variables nuevas, y busca en un archivo separado de configuración las fechas para las que tiene que correr el proceso de creación de variables nuevas. Cuando obtiene esa lista de fechas, busca cuáles son las dependencias de esa tabla en particular en el archivo *YAML* y verifica que estén completas.

Una vez verificado que el proceso de limpieza esté completo para todas las dependencias, se busca un código de R específico para esa tabla de *features*. Buscamos crear las mismas variables para muchas fuentes de datos distintas -en este caso, tenemos CUIS, ENCASEH y ENIGH- por lo que estandarizamos el proceso para auxiliarse de un archivo *YAML* que define todos los conjuntos de variables a crear, así como datos adicionales como variables de agrupamiento o tipo de operación a realizar. Este archivo, así como las funciones auxiliares para procesarlo, se utiliza para todas las fuentes de datos para las que se vayan a crear esas variables. De esta forma, tenemos prácticamente el mismo proceso aplicado a tres fuentes de datos distintas.

En el archivo *YAML* que contiene las variables nuevas, definimos abstracciones que nos permiten efectuar el mismo tipo de operación cuantas veces sea necesario, utilizando la misma función auxiliar. Tenemos tres tipos de funciones auxiliares, que toman una cadena de texto y la analizan sintácticamente para ejecutar las operaciones ahí indicadas, y se diferencian en la forma de analizar esa cadena de texto:

- ***get_dummy***: Analiza la cadena de texto como una condición lógica, verificando su valor de verdad. Regresa una variable dicotómica que responde al valor de verdad encontrado.
- ***get_func***: Analiza la cadena de texto literalmente, como una instrucción válida a realizar en R. Un ejemplo de esto puede ser una multiplicación entre dos variables.
- ***get_cases***: Analiza la cadena de texto como un conjunto de casos a considerar, de modo que la cadena de texto en este tipo de función tendría la forma $condicion \rightarrow resultado$, donde

$condicion$ es una condición lógica, y $resultado$ es el valor que toma la variable en caso de verificarse que la condición es verdadera.

Tomamos entonces el nombre de la nueva variable, la cadena de texto a evaluar, el tipo de función que la va a evaluar y posiblemente un conjunto de datos auxiliares, como si existen variables de agrupamiento o filtros a aplicar antes de hacer la transformación. De esta forma, basta con seleccionar el conjunto de variables que se van a crear a partir del archivo *YAML* y se puede entonces utilizar un conjunto muy sucinto y legible de funciones para crear ese conjunto de variables.

4. Primeros resultados

Comparación entre poblaciones

La primera inquietud a resolver es si los datos de entrenamiento son una muestra suficientemente parecida a los datos para los cuales se van a generar nuevas predicciones. Esta pregunta es relevante porque podrían existir características de las poblaciones atendidas por los programas que hacen verificaciones domiciliarias que no sean compartidas por las poblaciones atendidas por los programas que no las hacen, y si estas inciden en los patrones de sub y sobrerreportaje, la corrección e imputación de variables latentes puede ser incorrecta. Así, comparamos las poblaciones en términos de las variables que podemos observar, compartidas por ambas poblaciones. Como fue mostrado en [4], parece que las poblaciones son razonablemente parecidas, por lo que podríamos suponer que el conjunto de datos de ENCASEH son un conjunto válido de datos de entrenamiento.

Ajuste del modelo

La estructura del modelo propuesta parece ser suficientemente parsimoniosa y no afectar el desempeño del modelo.[1] Las relaciones entre las variables muestran agrupamientos bastante lógicos en términos teóricos, y la estructura de la gráfica es relativamente simple. Como mencionamos en Capítulo 2, utilizamos solamente dos restricciones para la estructura de la gráfica. Sin embargo, existen variables adicionales a considerar que podrían mejorar el ajuste del modelo para el conjunto de prueba: en la medida en que nuevas variables garanticen independencia condicional entre los nodos de la gráfica, se reduce el número de parámetros a estimar y por ende las posibilidades de sobreajuste.

[1] Recordar que medimos el desempeño del modelo con la log-verosimilitud promedio para cada posible valor del parámetro de regularización.

Variables adicionales a considerar

La mayor parte de las variables a considerar se pueden obtener a nivel municipal, y expresan condiciones socioeconómicas del municipio en cuestión:

- **Indicadores de carencias**: como están definidas por CONEVAL, existen seis carencias socioeconómicas. Se tienen datos a nivel municipal de rezago educativo, inseguridad alimentaria, calidad de los espacios en la vivienda, acceso a servicios en la vivienda, acceso a los servicios de salud y acceso a los servicios de seguridad social.
- **Distancia a centros de servicio**: algunas de las variables reportadas, como tenencia de distintos enseres, pueden estar correlacionadas con la distancia que hay a centros de servicio y distribución, para lo que los datos georreferenciados a nivel localidad pueden ser útiles en la construcción de nuevas variables.
- **Información socioeconómica adicional del municipio**: información adicional -disponible en INEGI- para el municipio, como las tasas de pobreza, marginación, y acceso global a distintos servicios pueden ser útiles también para describir los patrones de sub y sobrerreportaje en los cuestionarios.

5. Conclusiones

En este documento se plantearon las herramientas utilizadas para hacer del análisis un proceso escalable, legible, portable y reproducible. Tomamos en cuenta un marco de análisis basado en la idea de modelos gráficos para datos faltantes. Se construyó entonces un flujo de datos que permitió limpiar y homologar valores para fuentes distintas, así como estandarizar el proceso de creación de nuevas variables. La creación de variables nuevas es especialmente importante para procurar mantener un modelo parsimonioso y así evitar el sobreajuste.

Bibliografía

[1] Docker. What is a container. https://www.docker.com/resources/what-container.

[2] Karthika Mohan and Judea Pearl. Graphical models for processing missing data, 2018.

[3] Karthika Mohan, Judea Pearl, and Tian Jin. Missing data as a causal inference problem, 2013.

[4] Luis Felipe González Pérez. Documento guía para modelación de reportaje incorrecto en cuis. Entregable 1.

A. Apéndice

Limpieza de datos

Listing A.1: Código de orquestación del flujo de datos para la limpieza de variables

```
#!/usr/bin/env python
# coding: utf-8

import datetime
import luigi
import os
import random
import subprocess
import logging
import pdb

from luigi import six
from os.path import join, dirname
from luigi import configuration
from luigi.contrib import postgres
from luigi.s3 import S3Target, S3Client
from dotenv import load_dotenv, find_dotenv
from luigi.contrib.postgres import PostgresTarget, PostgresQuery

from politica_preventiva.pipelines.utils.pg_sedesol import parse_cfg_string,\
        download_dir

from politica_preventiva.pipelines.utils.pg_tools import PGWrangler
from politica_preventiva.tasks.pipeline_task import DockerTask, PgRTask
from politica_preventiva.pipelines.ingest.ingest_orchestra import UpdateLineage
from politica_preventiva.pipelines.ingest.tools.ingest_utils import parse_cfg_list,\
    extras, dates_list, get_extra_str, s3_to_pandas, final_dates
from politica_preventiva.pipelines.utils import s3_utils
```

```
# Env Setup
load_dotenv(find_dotenv())

# Logger & Config
configuration.LuigiConfigParser.add_config_path('/pipelines/configs/luigi_models.cfg')
conf = configuration.get_config()

logging_conf = configuration.get_config().get("core", "logging_conf_file")
logging.config.fileConfig(logging_conf)
logger = logging.getLogger("dpa-sedesol")

# AWS
aws_access_key_id = os.environ.get('AWS_ACCESS_KEY_ID')
aws_secret_access_key = os.environ.get('AWS_SECRET_ACCESS_KEY')



class ETLPipeline(luigi.WrapperTask):

    current_date = luigi.DateParameter()
    pipelines = luigi.parameter.ListParameter()
    ptask = luigi.Parameter()
    client = S3Client()
    common_path = luigi.Parameter('DEFAULT')
    local_path = luigi.Parameter('DEFAULT')  # path where csv is located
    historical = luigi.Parameter('DEFAULT')

    def requires(self):
        if self.ptask!='auto':
            self.pipelines = (self.ptask,)

        logger.info('Running the following pipelines: {0}'.format(self.pipelines))
        # loop through pipeline tasks and data dates
        set_pipelines = [(pipeline_task, final_dates(self.historical,
                                                     pipeline_task,
                                                     self.current_date)) for
                         pipeline_task in self.pipelines]
        return [UpdateCleanDB(current_date=self.current_date,
                              pipeline_task=pipeline[0],
                              data_date=dates,
                              suffix=pipeline[1][1])
                for pipeline in set_pipelines for dates in pipeline[1][0]]
```

```
class UpdateCleanDB(postgres.PostgresQuery):
    """
    This Task runs the Clean script in clean folder for
    the pipeline_task, if it doesn't exists then it only updates
    with the last raw table
    """

    current_date = luigi.DateParameter()
    pipeline_task = luigi.Parameter()
    client = S3Client()
    data_date = luigi.Parameter()
    suffix = luigi.Parameter()

    clean_scripts = luigi.Parameter()
    # RDS
    database = os.environ.get("PGDATABASE")
    user = os.environ.get("POSTGRES_USER")
    password = os.environ.get("POSTGRES_PASSWORD")
    host = os.environ.get("PGHOST")

    @property
    def update_id(self):
        return str(self.pipeline_task) + str(self.data_date) +\
               str(self.suffix) + '_clean'

    @property
    def table(self):
        return "clean." + self.pipeline_task

    @property
    def query(self):
        # Read sql command
        path = self.clean_scripts + self.pipeline_task + '.sql'

        try:
            sqlfile = open(path, 'r')
            query = sqlfile.read()

        except:
            query = """DROP TABLE IF EXISTS {0};
            CREATE TABLE {0} AS (SELECT * FROM
                                 raw.{1});""".format(self.table,
```

```
                                                    self.pipeline_task)
        return query


    def requires(self):
        return UpdateLineage(current_date=self.current_date,
                             pipeline_task=self.pipeline_task,
                             data_date=self.data_date,
                             suffix=self.suffix)


    def output(self):
        return PostgresTarget(host=self.host,
                              database=self.database,
                              user=self.user,
                              password=self.password,
                              table=self.table,
                              update_id=self.update_id)
```

Listing A.2: Código principal de limpieza y recodificación de variables

```
#!/usr/bin/env Rscript

library(optparse)
library(dbplyr)
library(dplyr)
library(dbrsocial)
library(DBI)
source("recode_utils.R")

option_list = list(
  make_option(c("--data_date"), type="character", default="",
              help="data date", metavar="character"),
  make_option(c("--database"), type="character", default="",
              help="database name", metavar="character"),
  make_option(c("--user"), type="character", default="",
              help="database user", metavar="character"),
  make_option(c("--password"), type="character", default="",
              help="password for datbase user", metavar="character"),
  make_option(c("--host"), type="character", default="",
              help="database host name", metavar="character"),
  make_option(c("--pipeline_task"), type="character", default="",
              help="pipeline task", metavar="character"),
  make_option(c("--scripts_dir"), type="character", default="",
              help="scripts directory", metavar="character")
)

opt_parser <- OptionParser(option_list=option_list)

opt <- tryCatch(
  {
    parse_args(opt_parser)
  },
  error=function(cond) {
    message("Error: Provide database connection arguments appropriately.")
    message(cond)
    print_help(opt_parser)
    return(NA)
  },
  warning=function(cond) {
    message("Warning:")
```

```
    message(cond)
    return(NULL)
  },
  finally={
    message("Finished attempting to parse arguments.")
  }
)

if(length(opt) > 1){

  if (opt$database=="" | opt$user=="" |
      opt$password=="" | opt$host=="" ){
    print_help(opt_parser)
    stop("Database connection arguments are not supplied.n", call.=FALSE)
  }else{
    PGDATABASE <- opt$database
    POSTGRES_PASSWORD <- opt$password
    POSTGRES_USER <- opt$user
    PGHOST <- opt$host
    PGPORT <- "5432"
  }

  con <- DBI::dbConnect(RPostgreSQL::PostgreSQL(),
                  host = PGHOST,
                  port = PGPORT,
                  dbname = PGDATABASE,
                  user = POSTGRES_USER,
                  password = POSTGRES_PASSWORD
  )

  pipeline_task <- opt$pipeline_task
  scripts_dir <- opt$scripts_dir
  raw_table <- pipeline_task

  # clean files
  rfile <- glue::glue('{scripts_dir}/{pipeline_task}.R')
  sqlfile <- glue::glue('{scripts_dir}/{pipeline_task}.sql')

  # If SQL clean script
  if( file.exists(sqlfile) ){
    # Read query
    query <- readLines(sqlfile) %>%
```

```
            paste(collapse = "") %>% glue::glue()
    # send query
    dbGetQuery(con, sql(query))
    # ToDo: verificar que este nombre funcione
    temp_name <- glue::glue('temp_{pipeline_task}')
    dbCommit(con)


  # If R clean script
  } else if(file.exists(rfile)) {
    source(rfile)
    clean_table <- make_clean(pipeline_task, con)
    temp_name <- glue::glue('temp_{pipeline_task}')
    copy_to(con, clean_table,
            temp_name,
            temporary = TRUE)
  }

  # Recode
  recoded_table <- recode_vars(table_name = temp_name,
                               db_connection = con)

  # Store
  copy_to(con, recoded_table,
          dbplyr::in_schema("clean", pipeline_task),
          temporary = FALSE, overwrite = TRUE)
  dbDisconnect(con)
}
```

Listing A.3: Archivo de limpieza para la tabla de integrante de CUIS Histórico

```
#!/usr/bin/env Rscript
library(rlang)
library(dplyr)
library(tidyr)

int_columns <- c('c_con_res', 'c_cd_parentesco',
                 'reside', 'padre', 'madre',
                 'c_cd_edo_civil', 'edad',
                 'val_nb_renapo', 'num_per',
                 'conyuge')

text_columns <- c('llave_hogar_h', 'fch_creacion',
                  'usr_creacion')

replace_na <- c('')

# The ETL pipeline will call this function to run UpdateCleanDB task

make_clean <- function(pipeline_task, con){
  df <- tbl(con, dbplyr::in_schema('raw', pipeline_task))
  df %>%
      mutate_at(vars(one_of(int_columns)), as.integer) %>%
      mutate_at(vars(one_of(replace_na)), funs(ifelse(is.na(.), 2, .))) %>%
      mutate(hogar_id = llave_hogar_h) %>%
      mutate(person_id = paste(llave_hogar_h, c_integrante, sep='-')) %>%
      mutate(anio = substr(fch_creacion, 7, 10)) %>%
      mutate(anio = as.integer(anio))
}
```

Listing A.4: Archivo de limpieza para la tabla de datos socioeconómicos de los integrantes de CUIS Histórico

```
#!/usr/bin/env Rscript


int_columns <- c('c_instsal_a', 'c_instsal_b',
                 'c_afilsal_a', 'c_afilsal_b',
                 'c_lengua_ind', 'habl_esp',
                 'indigena', 'leer_escr',
                 'c_ult_nivel', 'c_ult_gra',
                 'asis_esc', 'c_aban_escu',
                 'c_con_tra', 'c_ver_con_trab',
                 'c_pos_ocup', 'c_peri_tra',
                 'c_mot_notr', 'trab_subor',
                 'trab_indep', 'trab_presta_a',
                 'trab_presta_b', 'trab_presta_c',
                 'trab_presta_d', 'trab_presta_e',
                 'trab_presta_f', 'trab_presta_g',
                 'trab_presta_h', 'trab_no_re',
                 'c_periodo', 'seg_volunt_a',
                 'seg_volunt_b', 'seg_volunt_c',
                 'seg_volunt_d', 'seg_volunt_e',
                 'seg_volunt_f', 'seg_volunt_g',
                 'jubilado', 'jubilado_1',
                 'jubilado_2', 'inapam',
                 'am_a', 'am_b', 'am_c', 'am_d',
                 'am_e', 'otr_ing_a', 'otr_ing_b',
                 'otr_ing_c', 'otr_ing_d',
                 'otr_ing_e', 'otr_ing_f', 'otr_ing_g',
                 'tiene_disca', 'disca_ori', 'disca_gra',
                 'tiene_discb', 'discb_ori', 'discb_gra',
                 'tiene_discc', 'discc_ori', 'discc_gra',
                 'tiene_discd', 'discd_ori', 'discd_gra',
                 'tiene_disce', 'disce_ori', 'disce_gra',
                 'tiene_discf', 'discf_ori', 'discf_gra',
                 'enf_art', 'enf_can', 'enf_cir',
                 'enf_ren', 'enf_dia', 'enf_cor',
                 'enf_pul', 'enf_vih', 'enf_def',
                 'enf_hip', 'enf_obe',
                 'p_agri', 'p_manu', 'p_come', 'p_tran',
                 'p_prof', 'p_educ', 'p_sald', 'p_recr',
                 'p_aloj', 'p_comu', 'p_otro',
```

```
                'c_raz_no_trab')

float_columns <- c('monto', 'otr_ing_a_2',
                   'otr_ing_b_2', 'otr_ing_c_2',
                   'otr_ing_c_2', 'otr_ing_d_2',
                   'otr_ing_e_2', 'otr_ing_f_2')

text_columns <- c('llave_hogar_h', 'fch_creacion',
                  'usr_creacion', 'csc_hogar',
                  'actualizacion_sedesol', 'data_date')



replace_na <- c('')

# The ETL pipeline will call this function to run UpdateCleanDB task

make_clean <- function(pipeline_task, con){
  df <- tbl(con, dbplyr::in_schema('raw', pipeline_task))
  df %>%
      mutate_at(vars(one_of(int_columns)), as.integer) %>%
      mutate_at(vars(one_of(float_columns)), as.double) %>%
      mutate_at(vars(one_of(replace_na)), function(x) if_else(is.na(x), 2, x)) %>%
      mutate(hogar_id = llave_hogar_h) %>%
      mutate(person_id = paste(llave_hogar_h, c_integrante, sep='-'))
}
```

Listing A.5: Archivo de limpieza para la tabla de viviendas de CUIS Histórico

```
#!/usr/bin/env Rscript
library(rlang)
library(dplyr)
library(tidyr)

int_columns <- c('c_tipo_viv', 'tot_per_viv',
                 'tot_hog', 'tot_per', 'per_gasto',
                 'per_alim')

text_columns <- c('llave_hogar_h', 'fch_creacion',
                  'usr_creacion', 'csc_hogar')

replace_na <- c('')

# The ETL pipeline will call this function to run UpdateCleanDB task

make_clean <- function(pipeline_task, con){
  df <- tbl(con, dbplyr::in_schema('raw', pipeline_task))
  df %>%
      mutate_at(vars(one_of(int_columns)), as.integer) %>%
      mutate_at(vars(one_of(replace_na)), funs(ifelse(is.na(.), 2, .))) %>%
      mutate(hogar_id = llave_hogar_h) %>%
      mutate(anio = substr(fch_creacion,7,8)) %>%
      mutate(anio = as.integer(anio))
}
```

Listing A.6: Archivo de limpieza para la tabla de datos socioeconómicos de las viviendas de CUIS Histórico

```
#!/usr/bin/env Rscript
library(rlang)
library(tidyverse)

int_columns <- c('c_salud_hoga', 'c_salud_hogb',
                 'ut_cuida1', 'ut_cuida2',
                 'ut_volun1', 'ut_volun2',
                 'ut_repara1', 'ut_repara2',
                 'ut_limpia1', 'ut_limpia2',
                 'ut_acarrea1', 'ut_acarrea2',
                 'con_remesa', 'com_dia',
                 'com_dia_nsnr', 'cereal',
                 'verduras', 'frutas', 'leguminosas',
                 'carne_huevo', 'lacteos', 'grasas',
                 'seg_alim_1', 'seg_alim_2',
                 'seg_alim_3', 'seg_alim_4',
                 'seg_alim_5', 'seg_alim_a',
                 'seg_alim_b', 'seg_alim_c',
                 'seg_alim_d', 'seg_alim_e',
                 'seg_alim_f', 'seg_alim_g',
                 'desay_nin', 'desay_lugar',
                 'desay_razon', 'cuart',
                 'cua_dor', 'coc_duer',
                 'c_piso_viv', 'condi_piso',
                 'cuar_pis_t', 'c_tech_viv',
                 'condi_techo', 'c_muro_viv',
                 'condi_muro', 'c_escusado',
                 'uso_exc', 'c_agua_a',
                 'trat_agua_a', 'trat_agua_b',
                 'trat_agua_c', 'trat_agua_d',
                 'trat_agua_e', 'trat_agua_f',
                 'c_con_drena', 'c_basura',
                 'c_combus_cocin', 'fogon_chim',
                 'ts_refri', 'ts_lavadora',
                 'ts_vhs_dvd_br', 'ts_vehi',
                 'ts_telefon', 'ts_micro',
                 'ts_compu', 'ts_est_gas',
                 'ts_boiler', 'ts_internet',
                 'ts_celular', 'ts_television',
```

```
                'ts_tv_digital', 'ts_tv_paga',
                'ts_tinaco', 'ts_clima',
                'c_luz_ele', 'c_sit_viv',
                'escritura1', 'escritura2',
                'esp_niveles', 'esp_construc',
                'esp_local', 'tie_agri',
                'prop_tierra1', 'prop_tierra2',
                'c_maiz', 'c_frij', 'c_cere',
                'c_frut', 'c_cana', 'c_jito', 'c_chil',
                'c_limn', 'c_papa', 'c_cafe', 'c_cate',
                'c_forr', 'c_otro', 'c_ning',
                'cul_riego', 'cul_maquina', 'cul_anim',
                'cul_ferorg', 'cul_ferquim',
                'cul_plagui', 'uso_hid_tra',
                'caballos', 'burros', 'bueyes',
                'chivos', 'reses', 'gallinas',
                'cerdos', 'conejos', 'proyecto',
                'piso_prog', 'escusado_prog')

float_columns <- c('construc_med', 'local_med',
                   'gas_alim', 'gas_vest', 'gas_educ')

text_columns <- c('llave_hogar_h', 'actualizacion_sedesol',
                  'data_date', 'fch_creacion', 'usr_creacion',
                  'csc_hogar')

replace_na <- c('seg_alim_1', 'seg_alim_2',
                'seg_alim_3', 'seg_alim_4',
                'seg_alim_5', 'seg_alim_a',
                'seg_alim_b', 'seg_alim_c',
                'seg_alim_d', 'seg_alim_e',
                'seg_alim_f', 'seg_alim_g')

# The ETL pipeline will call this function to run UpdateCleanDB task

make_clean <- function(pipeline_task, con){
  df <- tbl(con, dbplyr::in_schema('raw', pipeline_task))
  df %>%
      mutate_at(vars(one_of(int_columns)), as.integer) %>%
      mutate_at(vars(one_of(float_columns)), as.double) %>%
      mutate_at(vars(one_of(replace_na)), funs(ifelse(is.na(.), 2, .))) %>%
      mutate(hogar_id = llave_hogar_h)
```

```
}
```

Recodificación de datos

Listing A.7: Archivo de funciones auxiliares de recodificación de datos.

```
#!/usr/bin/env Rscript

##############################
# Auxiliary functions
##############################

source('setup.R')
#source('connect.R')
source('pipeline_utils.R')



get_table_name <- function(table_name){
  raw_table_name <- sub('temp_', '', table_name)

  # Define which schema to look for table
  if (grepl('temp_', table_name)){
    schema <- 'public'
  } else{
    schema <- 'raw'
  }

  return(list(raw_table_name, schema))
}



get_table <- function(table_name, db_connection, schema = 'public'){
  # Lazy query for fetching table
  if (schema == 'public'){
    dplyr::tbl(db_connection, table_name)
  } else{
    dplyr::tbl(db_connection, dbplyr::in_schema(schema, table_name))
  }
}



get_variable_name <- function(table_name, old_name, recode_table){
  # Reads new variable name
  matches <- recode_table %>%
```

```
                dplyr::filter(tabla  == table_name, variable_inicial == old_name)

  check_rows <- matches %>%
                  dplyr::summarise(n = n()) %>%
                  dplyr::pull(n)

  if(check_rows == 0) return(NA)

  matches %>%
    dplyr::pull(variable_final) %>%
    unique()
}



get_varlist <- function(table_name, recode_table){
  # Fetch list of variables to recode
  matches <- recode_table %>%
              dplyr::filter(tabla == table_name)

  check_rows <- matches %>%
                  dplyr::summarise(n = n()) %>%
                  dplyr::pull(n)

  if(check_rows == 0) return(NA)

  matches %>%
    dplyr::pull(variable_inicial) %>%
    unique()
}



join_codes <- function(raw_table, table_name, old_name, new_name, recode_table){
  # Recoding function
  if(is.na(new_name)) return(raw_table)

  name <- eval(old_name)
  var_enquo <- rlang::enquo(name)
  by <- purrr::set_names('valor_inicial', rlang::quo_name(var_enquo))

  filter_recode <- recode_table %>%
                    dplyr::filter(tabla == table_name, variable_inicial == !! var_enquo)
  valores <- filter_recode %>% dplyr::pull(valor_inicial)
```

```
  if (is.na(valores[1])){
    old_name_sym <- rlang::sym(old_name)
    old_name_quo <- rlang::quo(!!old_name_sym)
    return_table <- raw_table %>%
                    dplyr::mutate(!!new_name := !!old_name_quo) %>%
                    dplyr::select(-old_name) %>%
                    dplyr::compute()
  } else {
    return_table <- raw_table %>%
                    dplyr::left_join(filter_recode, by = by) %>%
                    dplyr::select(-variable_inicial) %>%
                    dplyr::rename(temp = !!name, !!new_name := valor_final) %>%
                    dplyr::select(-temp, -tabla, -nombre_valor_final, -variable_final) %>%
                    dplyr::compute()
  }
  return(return_table)
}


cast_int <- function(table_name, raw_table_name, db_connection, schema){
    recode_table <- get_table(table_name = 'recode',
                              db_connection = db_connection,
                              schema = 'raw')
    clean_table <- get_table(table_name = table_name,
                             db_connection = db_connection,
                             schema = schema)
    varlist <- get_varlist(raw_table_name, recode_table)

    clean_table %>%
      dplyr::mutate_at(vars(varlist), as.integer)
}


recode_vars <- function(table_name,
                        recode_table_name = 'recode',
                        db_connection = con){

    # Get raw_table_name and schema
    c(raw_table_name, schema) := get_table_name(table_name)

    # Get tables
```

```
    recode_table <- get_table(recode_table_name, db_connection, 'raw')
    raw_table <- get_table(table_name, db_connection, schema)
    # Get selected variables
    varlist <- get_varlist(raw_table_name, recode_table)

    if (is.na(varlist[1])) return(raw_table)

    # Separate variables to be recoded in order to simplify computing process
    raw_table <- raw_table %>%
                 dplyr::mutate(temp_id = row_number())
    not_recoded <- raw_table %>%
                   dplyr::select(-one_of(varlist))
    new_vars <- raw_table %>%
                dplyr::select(temp_id)

    for(old_name in varlist){
      # Get new varname from recode_table
      new_name <- get_variable_name(raw_table_name, old_name, recode_table)
      to_recode <- raw_table %>%
                   dplyr::select(one_of(c('temp_id', old_name)))
      # Perform variable recode
      new_var <- join_codes(to_recode, raw_table_name, old_name, new_name, recode_table)
      new_vars <- new_vars %>%
                   dplyr::left_join(new_var) %>%
                   dplyr::compute()
      print(glue::glue('Renamed variable {old_name} to {new_name}'))
    }

    new_table <- not_recoded %>%
                 dplyr::left_join(new_vars)

    new_table
}
```

Creación de nuevas variables

Listing A.8: Código de orquestación del flujo de datos para la creación de nuevas variables

```
#!/usr/bin/env python
# coding: utf-8

import datetime
import luigi
import os
import random
import subprocess
import logging
import pdb
import yaml

from luigi import six
from os.path import join, dirname
from luigi import configuration
from luigi.contrib import postgres
from luigi.s3 import S3Target, S3Client
from dotenv import load_dotenv, find_dotenv
from luigi.contrib.postgres import PostgresTarget, PostgresQuery

from politica_preventiva.pipelines.utils.pg_sedesol import parse_cfg_string,\
        download_dir
from politica_preventiva.tasks.pipeline_task import DockerTask
from politica_preventiva.pipelines.ingest.tools.ingest_utils import parse_cfg_list,\
    extras, dates_list, get_extra_str, s3_to_pandas, final_dates
from politica_preventiva.pipelines.utils import s3_utils
from politica_preventiva.pipelines.etl.etl_orchestra import ETLPipeline

# Environment Setup
load_dotenv(find_dotenv())

# Logger and Config
conf = configuration.get_config()
logging_conf = configuration.get_config().get("core", "logging_conf_file")

logging.config.fileConfig(logging_conf)
logger = logging.getLogger("dpa-sedesol")
```

```
# AWS
aws_access_key_id = os.environ.get('AWS_ACCESS_KEY_ID')
aws_secret_access_key = os.environ.get('AWS_SECRET_ACCESS_KEY')

# Semantic Schema

with open("pipelines/configs/features_dependencies.yaml", "r") as file:
    composition = yaml.load(file)


class FeaturesPipeline(luigi.WrapperTask):

    features = parse_cfg_list(conf.get("FeaturesPipeline", "pipelines"))
    current_date = luigi.DateParameter()
    client = S3Client()
    ptask = luigi.Parameter()

    def requires(self):
        return [UpdateFeaturesDB(features_task, self.current_date)
                for features_task in self.features]

class UpdateFeaturesDB(postgres.PostgresQuery):

    features_task = luigi.Parameter()
    current_date = luigi.DateParameter()
    client = S3Client()
    features_scripts = luigi.Parameter()

    # AWS RDS
    database = os.environ.get("PGDATABASE")
    user = os.environ.get("POSTGRES_USER")
    password = os.environ.get("POSTGRES_PASSWORD")
    host = os.environ.get("PGHOST")

    @property
    def update_id(self):
        return str(self.features_task) + '_features'

    @property
    def table(self):
        return "features." + self.features_task
```

```
@property
def cmd(self):
    # Read features script
    features_script = self.features_scripts +\
            self.features_task + '.R'

    if not os.path.isfile(features_script):
        return

    command_list = ['Rscript', features_script,
                    '--database', self.database,
                    '--user', self.user,
                    '--password', "'{}'".format(self.password),
                    '--host', self.host]
    cmd = " ".join(command_list)
    return cmd
@property

def requires(self):
    # Check table dependencies
    dep_types = [dt for dt in composition[self.features_task].keys()]

    for dt in dep_types:
        if 'features_dependencies' in dep_types:
            features_tables = composition[self.features_task]['features_dependencies']
            yield [FeaturesPipeline(current_Date=self.current_date,
                feature_task=feature_task) for feature_task in features_tables]

        if 'clean_dependencies' in dep_types:
            clean_tables = composition[self.clean_task]['clean_dependencies']
            yield [ETLPipeline(current_date=self.current_date,
                pipelines=pipeline_task) for pipeline_task in clean_tables]

        if 'model_dependencies' in dep_types:
            models_tables = composition[self.models_task]['models_dependencies']
            yield [ModelsPipeline(current_date=self.current_date,
                pipelines=pipeline_task) for pipeline_task in models_tables]

def output(self):
    return PostgresTarget(host=self.host,
                          database=self.database,
                          user=self.user,
```

```
password=self.password,
table=self.table,
update_id=self.update_id)
```

Listing A.9: Archivo auxiliar para definir parámetros adicionales en el flujo de creación de nuevas variables

```
[cuis_carencias]
periodicity=None

[cuis_ingreso]
periodicity=None

[enigh_carencias]
periodicity=None

[enigh_ingreso]
periodicity=None
```

Listing A.10: Archivo auxiliar de definición de dependencias para la creación de nuevas variables

```
# YAML file indicating feature task dependencies.
# Dependencies are to be listed according to the schema
# their PostgreSQL table is located in:
# clean_dependencies: "clean"
# features_dependencies: "features"
# models_dependencies: "models"
cuis_carencias:
    clean_dependencies:
        - cuis_historico_integrante
        - cuis_historico_se_integrante
        - cuis_historico_se_vivienda
        - cuis_historico_vivienda
cuis_ingreso:
    clean_dependencies:
        - cuis_historico_integrante
        - cuis_historico_se_integrante
        - cuis_historico_se_vivienda
        - cuis_historico_vivienda
enigh_carencias:
    clean_dependencies:
        - enigh_hogares
        - enigh_ingresos
        - enigh_poblacion
        - enigh_trabajos
        - enigh_viviendas
enigh_ingreso:
    clean_dependencies:
        - enigh_hogares
        - enigh_ingresos
        - enigh_poblacion
        - enigh_trabajos
        - enigh_viviendas
```

Listing A.11: Archivo de funciones auxiliares para la creación de nuevas variables

```
#!/usr/bin/env Rscript

library(dplyr)
library(purrr)
library(rlang)
library(yaml)

apply_condition <- function(df, ids, name, condition, groupby, dofilter){
    #' @title Create variable from condition.
    #' @description This function creates new variable by parsing
    #' condition from yaml and applying it to a dataframe
    #' @param df data.frame (df)
    #' @param ids table ids column name (string)
    #' @param name name of new variable to be created (string)
    #' @condition condition to be evaluated to create new variable (string)
    #' @groupby name of df variable to groupby (string)
    #' @dofilter text to parse in order to filter df rows and apply
    #' condition to those rows (string)

  condition <- eval(parse(text=condition))

  if (is.null(dofilter)) {
    if (!is.null(groupby)){
      df_tmp <- df %>%
                  dplyr::group_by(!!!rlang::syms(groupby)) %>%
                  dplyr::summarise(!!name := !!condition) %>%
                  dplyr::compute()
      df <- df %>%
              dplyr::left_join(df_tmp, by = groupby) %>%
              dplyr::mutate_at(vars(one_of(name)), funs(coalesce(., 0)))
    } else {
      df <- df %>%
              dplyr::mutate(!!name := !!condition)
    }
  } else {
    dofilter <- glue::glue("quote({dofilter})")
    dofilter <- eval(parse(text = dofilter))

    if (!is.null(groupby)){
      df_tmp <- df %>%
```

```
                dplyr::filter(!!dofilter) %>%
                dplyr::group_by(!!!rlang::syms(groupby)) %>%
                dplyr::summarise(!!name := !!condition) %>%
                dplyr::compute()
      df <- df %>%
              dplyr::left_join(df_tmp, by = groupby) %>%
              dplyr::mutate_at(name, funs(coalesce(., 0)))
    } else {
      df_tmp <- df %>%
                dplyr::filter(!!dofilter) %>%
                dplyr::mutate(!!name := !!condition) %>%
                dplyr::select(!!!c(rlang::sym(name), ids)) %>%
                dplyr::compute()
      df <- df %>% dplyr::left_join(df_tmp, by = ids)
    }
  }
  return(df)
}

getDummy <- function(df, ids, name, condition, groupby = NA, dofilter = NA){
  condition <- glue::glue("quote(ifelse(({condition}), 1, 0))")

  df %>%
    apply_condition(ids, name, condition, groupby, dofilter) %>%
    dplyr::compute()
}

getCases <- function(df, ids, name, condition, groupby = NA, dofilter = NA){
  condition <- glue::glue("quote(case_when({condition}))")

  df %>%
    apply_condition(ids, name, condition, groupby, dofilter) %>%
    dplyr::compute()
}

getFunc <- function(df, ids, name, condition, groupby = NA, dofilter = NA){
  condition <- glue::glue("quote({condition})")

  df %>%
    apply_condition(ids, name, condition, groupby, dofilter) %>%
    dplyr::compute()
}
```

```
aux <- function(df=NULL, ids=NULL, transmute=NULL, condition=NULL, name=NULL){
  fun <- get(transmute$fun)
  groupby <- try(transmute$groupby)
  dofilter <- try(transmute$filter)

  fun(df = df,
      ids = ids,
      name = name,
      condition = condition,
      groupby = groupby,
      dofilter = dofilter)
}

make_features <- function(yml_name){
  # run chunks
  yml <- yaml::read_yaml(yml_name)
  doChunk <- function(df=NULL, chunkname=NULL, vars=NULL, ids=NULL, return.all=FALSE){
    #' @title Apply chunk
    #' @description Apply functions to a complete chunk of the yaml
    #' @param df data.frame (df)
    #' @param chunkname key name in yaml (string)
    #' @param ids table ids column name (string)
    #' @param return.all  return all df or just new columns (bool)
    #' @return data.frame with new columns (df)
    params <- yml[[c(chunkname)]]

    names <- c()
    for (i in seq_along(params)){
      if (params[[i]]$name  %in% vars){
        df <- aux(df=df,
                  ids=ids,
                  transmute=params[[i]]$transmute,
                  condition=params[[i]]$condition,
                  name=params[[i]]$name)  %>% compute()
        print(params[[i]]$name)
        names <- c(names, params[[i]]$name)
        }
    }
    if(return.all) return(df)
```

```
    df %>% select(c(ids, names))
  }
}
```

Listing A.12: Archivo de funciones auxiliares para la definición de objetos parecidos a las tuplas

```
#!/usr/bin/env Rscript



':=' <- function(lhs, rhs) {
  frame <- parent.frame()
  lhs <- as.list(substitute(lhs))
  if (length(lhs) > 1)
    lhs <- lhs[-1]
  if (length(lhs) == 1) {
    do.call('=', list(lhs[[1]], rhs), envir=frame)
    return(invisible(NULL))
  }
  if (is.function(rhs) || is(rhs, 'formula'))
    rhs <- list(rhs)
  if (length(lhs) > length(rhs))
    rhs <- c(rhs, rep(list(NULL), length(lhs) - length(rhs)))
  for (i in 1:length(lhs))
    do.call('=', list(lhs[[i]], rhs[[i]]), envir=frame)
  return(invisible(NULL))
}
```

Listing A.13: Archivo auxiliar para la creación de variables de carencias a nivel hogar

```
# ------------------------------------------------
# Rezago educativo
# ------------------------------------------------

rezago_educativo:
    # Variable auxiliar para CUIS
    - name: antec_esc
      condition: '1'
      transmute:
        fun: getFunc
    # Nivel Educativo
    - name: nivel_edu
      condition: 'aprobo_ultnivel < 2 | (aprobo_ultnivel == 2 & aprobo_ultgrado < 6) ~ 0,
                  (aprobo_ultnivel == 2 & aprobo_ultgrado == 6) | (aprobo_ultnivel == 3 &
                      aprobo_ultgrado < 3) |
                  (aprobo_ultnivel == 5 | aprobo_ultnivel == 6) & aprobo_ultgrado < 3 & antec_esc ==
                      1 ~ 1,
                  (aprobo_ultnivel == 3 & aprobo_ultgrado == 3) | (aprobo_ultnivel == 4) |
                   (aprobo_ultnivel == 5 & antec_esc == 1 & aprobo_ultgrado >= 3) | (aprobo_ultnivel
                       == 6 & antec_esc == 1 & aprobo_ultgrado >= 3) |
                   (aprobo_ultnivel == 5 & antec_esc >= 2) | (aprobo_ultnivel == 6 & antec_esc >= 2)
                       | (aprobo_ultnivel >= 7) ~ 2'
      transmute:
        fun: getCases
    # ano nacimiento
    - name: a_nacimiento
      condition: 'anio-edad'
      transmute:
        fun: getFunc


    ## Carencia por rezago Educativo
    # 1. Entre los 3 y 5 anos y no ha terminado la educacion obligatoria (secundaria terminada) y no
         siste a la escuela
    # 2. Nacio antes de 1982 y no cuenta con el nivel de educacion obligatoria vigente en el momento
         en que debia haberla cursado (primaria completa).
    # 3. Nacio a partir de 1982, es mayor de 15 anos y no cuenta con el nivel de educacion
        obligatoria (secundaria completa).
    - name: ic_rezago_educativo
      condition: '( (edad >= 3 & edad <= 15) & (asiste_escuela == 0 & (nivel_edu == 0 | nivel_edu ==
          1)) ) |
                  ( edad >= 16 & a_nacimiento >= 1982 & (nivel_edu == 0 | nivel_edu == 1) ) |
```

```
                  ( edad >= 16 & a_nacimiento <= 1981 & nivel_edu == 0 )'
      transmute:
        fun: getDummy

# ------------------------------------------------
# Carencia por Acceso a los Servicios Basicos en la Vivienda
# ------------------------------------------------

carencia_servicios:
    # Indicadora por servicio de agua
    - name: ic_servicio_agua
      condition: 'servicio_agua <= 3 | !is.na(servicio_agua)'
      transmute:
        fun: getDummy
    # Indicadora por servicio de drenaje
    - name: ic_servicio_drenaje
      condition: 'servicio_drenaje <= 3'
      transmute:
        fun: getDummy
    # Indicadora por servicio de electricidad
    - name: ic_servicio_electricidad
      condition: 'servicio_elect == 1'
      transmute:
        fun: getDummy
    # Indicadora por servicio de combustible
    - name: ic_combustible
      condition: '(fuente_combustible == 1) & (estufa_sinchimenea == 1)'
      transmute:
        fun: getDummy

    ## Indicador de Carencia por servicios basicos
    # 1. Indicador de carencia por servicio de agua
    # 2. Indicador de carencia de servicio de drenaje
    # 3. Indicador de carencia de servicios de electricidad
    # 4. Indicador de combustible
    - name: ic_servicios_basicos
      condition: 'ic_servicio_agua == 1 | ic_servicio_drenaje == 1 | ic_servicio_electricidad == 1 |
           ic_combustible == 1'
      transmute:
        fun: getDummy

# ------------------------------------------------
```

```
# Carencia en Calidad y Espacios en la Vivienda
# ------------------------------------------------

carencia_vivienda:
    # Indice de hacinamiento
    - name: indice_hacinamiento
      condition: 'total_personas / total_cuartosdor'
      transmute:
        fun: getFunc
    # Indicadora de hacinamiento
    - name: ic_hacinamiento
      condition: 'indice_hacinamiento > 2.5'
      transmute:
        fun: getDummy
    # Indicadora de carencia por material de pisos
    - name: ic_material_piso
      condition: 'material_pisos == 1'
      transmute:
        fun: getDummy
    # Indicadora de carencia por material de muros
    - name: ic_material_muros
      condition: 'material_muros <= 5'
      transmute:
        fun: getDummy
    # Indicador de carencia por material de techos de la vivienda
    - name: ic_material_techos
      condition: 'material_techos <= 2'
      transmute:
        fun: getDummy
    ## Indicador de Carencia por Calidad de Vivienda
    # 1. Indicador de carencia por indice de hacinamiento
    # 2. Indicador de carencia por material de piso de la vivienda
    # 3. Indicador de carencia por material de techos de la vivienda
    # 4. Indicador de carencia por material de muros de la vivienda
    - name: ic_vivienda
      condition: '(ic_hacinamiento == 1) | (ic_material_piso == 1) | (ic_material_techos == 1) | (
          ic_material_muros == 1)'
      transmute:
        fun: getDummy

# ------------------------------------------------
# Carencia Alimentaria
```

```
# ------------------------------------------------

carencia_alimentaria:
    - name: min_edad
      condition: 'min(edad, na.rm = TRUE)'
      transmute:
        fun: getFunc
        groupby: [fecha_creacion, hogar_id]
    # hogares menores de 18 anos
    - name: i_menores18
      condition: 'min_edad >= 0 & min_edad <= 17'
      transmute:
        fun: getDummy
    # Para hogares sin menores de edad:
    # 1. Algun adulto tuvo una alimentacion basada en muy poca variedad de alimentos?
    # 2. Algun adulto deja de desayunar, comer o cenar?
    # 3. Algun adulto comio menos de lo que debia comer?
    # 4. El hogar se quedo sin comida?
    # 5. Algun adulto sintio hambre pero no comio?
    # 6.  Algun adulto sintio hambre pero no comio?

    # Para hogares con menores de edad:
    # 1. Alguien de 0 a 17 anios tuvo una alimentacion basada en muy poca variedad de alimentos?
    # 2. Alguien de 0 a 17 anios comio menos de lo que debia?
    # 3. Se tuvo que disminuir la cantidad servida en las comidas a alguien de 0 a 17 anios?
    # 4. Alguien de 0 a 17 anios sintio hambre pero no comio?
    # 5. Alguien de 0 a 17 anios se acosto con hambre?
    # 6. Alguien de 0 a 17 anios comio una vez al dia o dejo de comer todo un dia?

    # escala total
    - name: total_ia
      condition: 'alim_pocavar + alim_dejocomida + alim_comiomenos +
                  alim_sincomida + alim_nocomio + alim_comiouna +
                  alim_menor_pocavar + alim_menor_comiomenos + alim_menor_disminuyo +
                  alim_menor_nocomio + alim_menor_acosto + alim_menor_comiouna'
      transmute:
        fun: getFunc
    # grado inseguridad alimentaria
    - name: inseguridad_alimentaria
      condition: 'total_ia == 0 ~ 0,
                total_ia == 1 | total_ia == 2 | (total_ia == 3 & i_menores18 ==1) ~ 1,
```

```
                (i_menores18 == 0 & (total_ia == 3 | total_ia == 4 )) | (i_menores18 == 1 & (
                    total_ia==4 | total_ia==5 |total_ia==6 |total_ia==7)) ~ 2,
                (i_menores18 == 0 & (total_ia == 5 | total_ia == 6)) | total_ia >= 8 ~ 3'
      transmute:
        fun: getCases

    ## Indicador de Carencia por inseguridad alimentaria
    # Presente inseguridad alimentaria moderada o severa
    - name: ic_alimentacion
      condition: "(inseguridad_alimentaria == '2') | (inseguridad_alimentaria == '3')"
      transmute:
        fun: getDummy

# ------------------------------------------------
# Carencia por Acceso a Servicios de Salud
# ------------------------------------------------

trabajo_enigh:
    # poblacion economicamente activa
    - name: pea
      condition: "trabajo == 1 & edad >= 16 & !is.na(edad) & is.na(act_pnea1) == TRUE & is.na(
          act_pnea2) == TRUE ~ 1,
                 edad >= 16 & !is.na(edad) &
                 ((act_pnea1 == 1 & !is.na(act_pnea1)) | (act_pnea2 == 1 & !is.na(act_pnea2))) ~ 2,
                 (edad >= 16 & !is.na(edad)) &
                 ((act_pnea2 >= 2 & act_pnea2 <= 6 & !is.na(act_pnea2)) |
                 (act_pnea1 >= 2 & act_pnea1 <= 6 & !is.na(act_pnea1))) ~ 0"
      transmute:
        fun: getCases
    # Tipo de trabajo: subordinado, indep que recibe pago o indep que no recibe pago
    - name: tipo_trab
      condition: 'is.na(trabaja_subord) trabaja_subord == 1 ~ 1,
                  ((trabaja_subord == 0 & !is.na(trabaja_subord) & trabaja_indep == 1 & !is.na(
                      trabaja_indep)) &
                      ((con_sueldo == 1 & !is.na(con_sueldo)) | (tipo_pago == 1 & !is.na(tipo_pago)
                          ))) ~ 2,
                   ((trabaja_subord == 0 & !is.na(trabaja_subord)) &
                  ( (trabaja_indep == 1 & con_sueldo == 0 & !is.na(trabaja_indep) & !is.na(
                      con_sueldo)) |
                    (trabaja_indep == 0 & !is.na(trabaja_indep) & !is.na(tipo_pago) & (tipo_pago ==
                        2 | tipo_pago == 3)))) ~ 3'
      transmute:
```

```
        fun: getCases
    # Jubilados o pensionados
    # si no trabajo el mes pasado y es pensionado o jubilado
    - name: jubilado
      condition: 'trabajo == 0 & !is.na(trabajo) & ((act_pnea1 == 2 & !is.na(act_pnea1)) | (act_pnea
          2 == 2 & !is.na(act_pnea2)))'
      transmute:
        fun: getDummy
# Definicion de variables relacionadas al trabajo para tablas de CUIS
trabajo_cuis:
    # Realiza actividad laboral, de acuerdo al CUIS
    - name: trabajo
      condition: "(!is.na(realiza_act_lab) & (realiza_act_lab <= 3))|(!is.na(realiza_act_nolab) & (
          realiza_act_nolab <= 4))"
      transmute:
        fun: getDummy
    # Poblacion Economicamente Activa (poblacion incorporada al mercado de trabajo, ya sea ocupada o
        desocupada y en busqueda de una actividad laboral
    - name: pea
      condition: "edad >= 16 & (!is.na(trabajo) & trabajo == 1) ~ 1,
                  edad >= 16 & (!is.na(realiza_act_lab) & (realiza_act_lab == 5 | realiza_act_lab ==
                       6)) & (!is.na(realiza_act_nolab) & realiza_act_nolab <= 4) ~ 2,
                  edad >= 16 & (!is.na(realiza_act_nolab) & (realiza_act_nolab == 5 |
                      realiza_act_nolab == 6)) & (!is.na(realiza_act_lab) & realiza_act_lab == 5) ~
                      0"
      transmute:
        fun: getCases
    # Tipo de trabajo: subordinado, indep que recibe pago o indep que no recibe pago
    - name: tipo_trab
      condition: "!is.na(trabaja_subord) & trabaja_subord == 1 ~ 1,
                  (!is.na(trabaja_indep) & trabaja_indep == 1) | (!is.na(realiza_act_nolab) &
                      realiza_act_nolab <= 4) & (!is.na(con_sueldo) & con_sueldo == 1) ~ 2,
                  (!is.na(trabaja_indep) & trabaja_indep == 1) & (!is.na(con_sueldo) & con_sueldo ==
                       0) ~ 3"
      transmute:
        fun: getCases
    - name: jubilado
      condition: "jubilado == 1"
      transmute:
        fun: getDummy
carencia_salud:
    # Prestaciones laborales (servicios medicos)
```

```
# Personas ocupadas que estan afiliadas o inscritas a alguna institucion que proporciona
    atencion medica:
# IMSS o ISSSTE o ISSSTE estatal o Institucion de PEMEX por medio de una prestacion en el
    trabajo
- name: atencion_medica
  condition: '!is.na(c_instsal_a) & c_instsal_a == 99'
  transmute:
    fun: getDummy
- name: seguro_popular
  condition: '!is.na(c_instsal_a) & c_instsal_a == 1'
  transmute:
    fun: getDummy
# Inscrito al IMSS
- name: am_imss
  condition: '(!is.na(c_instsal_a) & c_instsal_a == 2) | (!is.na(c_instsal_b) & c_instsal_b == 2
      )'
  transmute:
    fun: getDummy
- name: am_issste
  condition: '(!is.na(c_instsal_a) & c_instsal_a == 3) | (!is.na(c_instsal_b) & c_instsal_b == 3
      )'
  transmute:
    fun: getDummy
- name: am_issste_estatal
  condition: '0'
  transmute:
    fun: getFunc
- name: am_pemex
  condition: '(!is.na(c_instsal_a) & c_instsal_a == 4) | (!is.na(c_instsal_b) & c_instsal_b == 4
      )'
  transmute:
    fun: getDummy
- name: am_imss_prospera
  condition: '(!is.na(c_salud_hoga) & c_salud_hoga == 3) | (!is.na(c_salud_hogb) & c_salud_hogb
      == 3)'
  transmute:
    fun: getDummy
- name: am_otra
  condition: '(!is.na(c_instsal_a) & c_instsal_a == 5) | (!is.na(c_salud_hoga) & (c_salud_hoga
      == 6 | c_salud_hoga == 7)) | (!is.na(c_salud_hogb) & (c_salud_hogb == 6 | c_salud_hogb ==
      7)'
  transmute:
```

```
    fun: getDummy
- name: inscrito_prestacion_lab
  condition: '(!is.na(c_afilsal_a) & c_afilsal_a == 1) | (!is.na(c_afilsal_b) & c_afilsal_b == 1
      )'
  transmute:
    fun: getDummy
- name: inscrito_jubilacion
  condition: '(!is.na(c_afilsal_a) & (c_afilsal_a == 2 | c_afilsal_a == 3)) | (!is.na(
      c_afilsal_b) & (c_afilsal_b == 2 | c_afilsal_b == 3))'
  transmute:
    fun: getDummy
- name: inscrito_familiar
  condition: '(!is.na(c_afilsal_a) & c_afilsal_a == 4) | (!is.na(c_afilsal_b) & c_afilsal_b == 4
      )'
  transmute:
    fun: getDummy
- name: inscrito_muerte_aseg
  condition: '(!is.na(c_afilsal_a) & c_afilsal_a == 5) | (!is.na(c_afilsal_b) & c_afilsal_b == 5
      )'
  transmute:
    fun: getDummy
- name: inscrito_estudiante
  condition: '(!is.na(c_afilsal_a) & c_afilsal_a == 6) | (!is.na(c_afilsal_b) & c_afilsal_b == 6
      )'
  transmute:
    fun: getDummy
- name: inscrito_contratacion
  condition: '(!is.na(c_afilsal_a) & c_afilsal_a == 7) | (!is.na(c_afilsal_b) & c_afilsal_b == 7
      )'
  transmute:
    fun: getDummy
- name: inscrito_familiar_otro
  condition: '(!is.na(c_afilsal_a) & c_afilsal_a == 8) | (!is.na(c_afilsal_b) & c_afilsal_b == 8
      )'
  transmute:
    fun: getDummy
- name: sm_lab
  condition: 'trabajo == 1 & atencion_medica == 1 & (am_imss == 1 | am_issste == 1 | !is.na(
      am_issste_estatal) & am_issste_estatal == 1 | am_pemex == 1) & (inscrito_prestacion_lab ==
       1)'
  transmute:
    fun: getDummy
```

```
# contratacion voluntaria: servicios medicos
# Personas que estan afiliadas o inscritas a alguna institucion que proporciona atencion medica:
# IMSS o ISSSTE o ISSSTE estatal o Institucion de PEMEX inscritos por contratacion propia
- name: sm_cv
  condition: 'atencion_medica == 1 &
              (am_imss == 1 | am_issste == 1 | am_issste_estatal == 1 | am_pemex == 1) & (
                  inscrito_contratacion == 1) & (edad >= 12)'
  transmute:
    fun: getDummy
# acceso directo a servicios de salud
# suboridinado con prestacion laboral o
# independiente que recibe pago con prestacion o voluntario o
# independiente sin pago con prestacion o voluntario
- name: salud_dir
  condition: '(tipo_trab == 1 & sm_lab == 1) |
              (tipo_trab == 2 & (sm_lab == 1 | sm_cv == 1)) |
              (tipo_trab == 3 & (sm_lab == 1 | sm_cv == 1))'
  transmute:
    fun: getDummy
# Jefe del hogar con acceso directo
# excepto contrataciones propias del ISSSTE o ISSSTE estatal
# que no este inscrito a otra ni por ningun otro medio
- name: jefe_sm
  condition: '(parentesco == 1 & salud_dir == 1) &
              !(((am_issste == 1 | am_issste_estatal == 1) & inscrito_contratacion == 1) &
              (is.na(am_imss) == TRUE & is.na(am_pemex) == TRUE & is.na(am_otra) == TRUE) &
              (is.na(inscrito_prestacion_lab) == TRUE & is.na(inscrito_jubilacion) == TRUE & is.
                  na(inscrito_familiar) == TRUE &
              is.na(inscrito_muerte_aseg) == TRUE & is.na(inscrito_estudiante) == TRUE & is.na(
                  inscrito_familiar_otro) == TRUE))'
  transmute:
    fun: getDummy
# Conyugue del hogar con acceso directo
# excepto contrataciones propias del ISSSTE o ISSSTE estatal
# que no este inscrito a otra ni por ningun otro medio
- name: cony_sm
  condition: '(parentesco == 2 & salud_dir == 1) &
              !(((am_issste == 1 | am_issste_estatal == 1) & inscrito_contratacion == 1) &
              (is.na(am_imss) == TRUE & is.na(am_pemex) == TRUE & is.na(am_otra) == TRUE) &
              (is.na(inscrito_prestacion_lab) == TRUE & is.na(inscrito_jubilacion) == TRUE & is.
                  na(inscrito_familiar) == TRUE &
```

```
                is.na(inscrito_muerte_aseg) == TRUE & is.na(inscrito_estudiante) == TRUE & is.na(
                    inscrito_familiar_otro) == TRUE))'
    transmute:
      fun: getDummy
  # Hijo del hogar con acceso directo
  # excepto contrataciones propias del ISSSTE o ISSSTE estatal
  # que no este inscrito a otra ni por ningun otro medio
  - name: hijo_sm
    condition: '(parentesco == 3 & salud_dir == 1) &
                !(((am_issste == 1 | am_issste_estatal == 1) & inscrito_contratacion == 1) &
                (is.na(am_imss) == TRUE & is.na(am_pemex) == TRUE & is.na(am_otra) == TRUE) &
                (is.na(inscrito_prestacion_lab) == TRUE & is.na(inscrito_jubilacion) == TRUE & is.
                    na(inscrito_familiar) == TRUE &
                is.na(inscrito_muerte_aseg) == TRUE & is.na(inscrito_estudiante) == TRUE & is.na(
                    inscrito_familiar_otro) == TRUE))'
    transmute:
      fun: getDummy
  # Acceso directo a los servicios de salud de la jefatura del hogar
  - name: acceso_jefe_sm
    condition: 'max(jefe_sm, na.rm = TRUE)'
    transmute:
      fun: getFunc
      groupby: [fecha_creacion, hogar_id]
  # Acceso directo a los servicios de salud de conyuge de la jefatura del hogar
  - name: acceso_cony_sm
    condition: 'max(cony_sm, na.rm = TRUE)'
    transmute:
      fun: getFunc
      groupby: [fecha_creacion, hogar_id]
  # Acceso directo a los servicios de salud de hijos(as) de la jefatura del hogar
  - name: acceso_hijo_sm
    condition: 'max(hijo_sm, na.rm = TRUE)'
    transmute:
      fun: getFunc
      groupby: [fecha_creacion, hogar_id]
  # Otros nucleos familiares:
  # Personas que estan afiliadas o inscritas a alguna institucion que proporciona atencion medica:
  # IMSS o ISSSTE o ISSSTE estatal o institucion PEMEX, e inscritas por:
  # familiar en el hogar o muerte del asegurado o algun otro familiar
  - name: acceso_otros_sm
    condition: 'atencion_medica == 1 & (am_imss == 1 | am_issste == 1 | am_issste_estatal == 1 |
        am_pemex == 1) &
```

```
                (inscrito_familiar == 1 | inscrito_muerte_aseg == 1 | inscrito_familiar_otro == 1)
                    &
                 !is.na(atencion_medica)'
      transmute:
        fun: getDummy
    # Indicador de Carencia de Servicios de Salud
    # No se encuentra afiliada o inscrita al Seguro Popular o alguna institucion
    # que proporcione servicios medicos, ya sea por prestacion laboral,
    # contratacion voluntaria o afiliacion de un familiar por parentesco directo
    # O sin acceso reportado con seguro popular o seguro popular y atencion medica o Seguro
        voluntario de gastos medicos
    - name: asalud
      condition: ' (salud_dir == 1) |
                (parentesco == 1 & acceso_cony_sm == 1 ) | (parentesco == 1 & pea == 0 &
                    acceso_hijo_sm == 1) |
                (parentesco == 2 & acceso_jefe_sm == 1) | (parentesco == 2 & pea == 0 &
                    acceso_hijo_sm == 1) |
                (parentesco == 3 & edad < 16 & (acceso_jefe_sm == 1 | acceso_cony_sm == 1)) |
                (parentesco == 3 & edad >= 16 & edad <= 25 & asiste_escuela == 1 & (acceso_jefe_sm
                    == 1 | acceso_cony_sm == 1)) |
                (parentesco == 4 & pea == 0 & (acceso_jefe_sm == 1 | acceso_cony_sm == 1)) |
                (acceso_otros_sm == 1) |
                (seguro_popular == 1 | (seguro_popular == 0 & atencion_medica == 1 &
                  (am_imss == 1 | am_issste == 2 | am_issste_estatal == 1 | am_pemex == 1 |
                      am_imss_prospera == 1 | am_otra == 6)) | segvol_medico == 1)'
      transmute:
        fun: getDummy
    - name: ic_asalud
      condition: ' asalud == 0 | is.na(asalud)'
      transmute:
        fun: getDummy

# ------------------------------------------------
# Carencia por seguridad social
# ------------------------------------------------

ingresos_enigh:
    # Suma de ingresos de pam
    - name: ingreso_pam_temp
      condition: '(ing_1 + ing_2 + ing_3 + ing_4 + ing_5 + ing_6) / 6'
      transmute:
        fun: getFunc
```

```
        filter: "clave == 'P044' | clave == 'P045'"
    - name: ingreso_pam
      condition: 'sum(ingreso_pam_temp, na.rm=TRUE)'
      transmute:
        fun: getFunc
        groupby: [fecha_creacion, person_id]
    # Programas sociales de pensiones para adultos mayores
    - name: pam
      condition: 'edad >= 65 & ingreso_pam > 0 & !is.na(ingreso_pam)'
      transmute:
        fun: getDummy
    # Suma de ingresos de pension
    - name: ingreso_pens_temp
      condition: '(ing_1 + ing_2 + ing_3 + ing_4 + ing_5 + ing_6) / 6'
      transmute:
        fun: getFunc
        filter: "clave == 'P032' | clave == 'P033'"
    - name: ingreso_pens
      condition: 'sum(ingreso_pens_temp, na.rm=TRUE)'
      transmute:
        fun: getFunc
        groupby: [fecha_creacion, person_id]

ingresos_cuis:
    - name: pam
      condition: '(!is.na(am_a) & am_a == 1) | (!is.na(am_b) & am_b == 1) | (!is.na(am_c) & am_c ==
          1)'
      transmute:
        fun: getDummy
    - name: ingreso_pens
      condition: '0'
      transmute:
        fun: getFunc

carencia_seguridad_social:
    # Prestaciones basicas:
    # Prestaciones laborales (Servicios medicos): IMSS, ISSSTE, ISSSTE estatal o inst PEMEX por
        prestacion laboral
    - name: sm_lab
      condition: 'trabajo == 1 & atencion_medica == 1  &
                  (am_imss == 1 | am_issste == 1 | am_issste_estatal == 1 | am_pemex == 1) & (
                      inscrito_prestacion_lab == 1)'
```

```
  transmute:
    fun: getDummy
# Contratacion voluntaria: servicios medicos y SAR o Afore
# servicios medicos: IMSS, ISSSTE, ISSSTE estatal o inst PEMEX por contratacion propia
- name: jubilado
  condition: ' jubilado == 1 | (ingreso_pens > 0 & !is.na(ingreso_pens)) |
                  (inscrito_jubilacion == 1 & !is.na(inscrito_jubilacion))'
  transmute:
    fun: getDummy
- name: sm_cv
  condition: 'atencion_medica == 1 &
             (am_imss == 1 | am_issste == 1 | am_issste_estatal == 1 | am_pemex == 1) & (
                 inscrito_contratacion == 1) & (edad >= 12)'
  transmute:
    fun: getDummy
# Seguro voluntario SAR, AFORE
- name: afore_cv
  condition: 'segvol_afore == 1 & edad >= 12'
  transmute:
    fun: getDummy
# Acceso directo a la seguridad social
- name: ss_dir
  condition: '(tipo_trab == 1 & !is.na(tipo_trab) &
                (sm_lab == 1 & !is.na(sm_lab) & prestacion_incapacidad == 1 & !is.na(
                    prestacion_incapacidad) &
                prestacion_afore == 1 & !is.na(prestacion_afore) )) |
                 ( tipo_trab == 2 & !is.na(tipo_trab) &
                  ( (sm_lab == 1 & !is.na(sm_lab)) | (sm_cv == 1 & !is.na(sm_cv))) &
                  ( (prestacion_afore == 1 & !is.na(prestacion_afore)) | (afore_cv == 1 & !is.na
                      (afore_cv)))) |
              (tipo_trab == 3 & !is.na(tipo_trab) &
              ((( sm_lab == 1 & !is.na(sm_lab)) | (sm_cv == 1 & !is.na(sm_cv))) & afore_cv == 1
                  & !is.na(afore_cv))) |
              (jubilado == 1)'
  transmute:
    fun: getDummy
# Jefe con acceso directo a seguridad social
# excepto contrataciones propias del ISSSTE o ISSSTE estatal
# que no este inscrito a otra ni por ningun otro medio
- name: jefe_ss
  condition: 'parentesco == 1 & ss_dir == 1 &
```

```
                !(((am_issste == 1 | am_issste_estatal == 3) & inscrito_contratacion == 1) & (is.
                    na(am_imss) == TRUE &
                is.na(am_pemex) == TRUE & is.na(am_otra) == TRUE) & (is.na(inscrito_prestacion_lab
                    ) == TRUE & is.na(inscrito_jubilacion) == TRUE &
                is.na(inscrito_familiar) == TRUE & is.na(inscrito_muerte_aseg) == TRUE & is.na(
                    inscrito_estudiante) == TRUE & is.na(inscrito_familiar_otro) == TRUE))'
  transmute:
    fun: getDummy
# Conyugye con acceso directo a seguridad social
# excepto contrataciones propias del ISSSTE o ISSSTE estatal
# que no este inscrito a otra ni por ningun otro medio
- name: cony_ss
  condition: 'parentesco == 2 & ss_dir == 1 &
                !(((am_issste == 1 | am_issste_estatal == 3) & inscrito_contratacion == 1) & (is.
                    na(am_imss) == TRUE &
                is.na(am_pemex) == TRUE & is.na(am_otra) == TRUE) & (is.na(inscrito_prestacion_lab
                    ) == TRUE & is.na(inscrito_jubilacion) == TRUE &
                is.na(inscrito_familiar) == TRUE & is.na(inscrito_muerte_aseg) == TRUE & is.na(
                    inscrito_estudiante) == TRUE & is.na(inscrito_familiar_otro) == TRUE))'
  transmute:
    fun: getDummy
# Hijo con acceso directo a seguridad social
# excepto contrataciones propias del ISSSTE o ISSSTE estatal
# que no este inscrito a otra ni por ningun otro medio
- name: hijo_ss
  condition: '(parentesco == 3 & ss_dir == 1 & (jubilado == 0 | (jubilado == 1 & edad > 25))) &
                !(((am_issste == 1 | am_issste_estatal == 3) & inscrito_contratacion == 1) & (is.
                    na(am_imss) == TRUE &
                is.na(am_pemex) == TRUE & is.na(am_otra) == TRUE) & (is.na(inscrito_prestacion_lab
                    ) == TRUE & is.na(inscrito_jubilacion) == TRUE &
                is.na(inscrito_familiar) == TRUE & is.na(inscrito_muerte_aseg) == TRUE & is.na(
                    inscrito_estudiante) == TRUE & is.na(inscrito_familiar_otro) == TRUE))'
  transmute:
    fun: getDummy
# Acceso directo a la seguridad social de la jefatura del hogar
- name: acceso_jefe_ss
  condition: 'max(jefe_ss, na.rm = TRUE)'
  transmute:
    fun: getFunc
    groupby: [fecha_creacion, hogar_id]
# Acceso directo a la seguridad social de conyuge de la jefatura del hogar
- name: acceso_cony_ss
```

```
  condition: 'max(cony_ss, na.rm = TRUE)'
  transmute:
    fun: getFunc
    groupby: [fecha_creacion, hogar_id]
# Acceso directo a la seguridad social de hijos(as) de la jefatura del hogar
- name: acceso_hijo_ss
  condition: 'max(hijo_ss, na.rm = TRUE)'
  transmute:
    fun: getFunc
    groupby: [fecha_creacion, hogar_id]
# Otros nucleos familiares
- name: acceso_otros_ss
  condition: 'atencion_medica == 1 & (am_imss == 1 | am_issste == 1 | am_issste_estatal == 1 |
      am_pemex == 1) &
             (inscrito_familiar == 1 | inscrito_muerte_aseg == 1 | inscrito_contratacion == 1 |
                 inscrito_familiar_otro == 1)'
  transmute:
    fun: getDummy
## Indicador carencia por inseguridad social
# 1. PEA: servicio medico, SAR o AFORE e incapacidad laboral con goce de sueldo
# 2. poblacion trabajadora no asalariada o independiente tenga: servicios medicos, SAR o AFORE
# 3. reciba jubilacion o pension, independientemente de su edad
# 4. programa de adultos mayores
# 5. por parentezco directo reciban servicios medicos
# 6. Las personas que gozan de alguna jubilacion, pension o que reciben servicios medicos en el
    IMSS, ISSSTE, ISSSTE estatal o PEMEX por parte de algun familiar, o consecuencia de muerte
    de una persona asegurada o por contratacion propia.
- name: seguridad_social
  condition: '( (ss_dir == 1) |
                (parentesco == 1 & (acceso_cony_ss == 1 | (pea == 0 & !is.na(pea) &
                    acceso_hijo_ss == 1))) |
                (parentesco == 2 & (acceso_jefe_ss == 1 | (pea == 0 & !is.na(pea) &
                    acceso_hijo_ss == 1))) |
                (parentesco == 3 & (( edad < 16 & acceso_jefe_ss == 1) | (edad < 16 &
                    acceso_cony_ss == 1) |
                           ( edad >= 16 & edad <= 25 & asiste_escuela == 1 & !is.na(
                               asiste_escuela) & acceso_jefe_ss == 1) |
                           (edad >= 16 & edad <= 25 & asiste_escuela == 1 & !is.na(
                               asiste_escuela) &  acceso_cony_ss == 1))) |
              (parentesco == 4 & pea == 0 & !is.na(pea) & acceso_jefe_ss == 1) |
              (parentesco == 5 & pea == 0 & !is.na(pea) & acceso_cony_ss == 1) |
              (acceso_otros_ss == 1) |
```

```
                (pam == 1 & !is.na(pam)))'
    transmute:
      fun: getDummy
  - name: ic_seguridad_social
    condition: 'seguridad_social == 0 & !is.na(seguridad_social)'
    transmute:
      fun: getDummy
```

Listing A.14: Código de creación de variables de carencias a nivel hogar

```
#!/usr/bin/env Rscript
library(optparse)
library(dbrsocial)
library(dbplyr)
library(dplyr)
library(DBI)
library(yaml)
#source("recode_utils.R")

option_list = list(
  make_option(c("--data_date"), type="character", default="",
              help="data date", metavar="character"),
  make_option(c("--database"), type="character", default="",
              help="database name", metavar="character"),
  make_option(c("--user"), type="character", default="",
              help="database user", metavar="character"),
  make_option(c("--password"), type="character", default="",
              help="password for datbase user", metavar="character"),
  make_option(c("--host"), type="character", default="",
              help="database host name", metavar="character"),
  make_option(c("--pipeline_task"), type="character", default="",
              help="pipeline taks", metavar="character"),
  make_option(c("--scripts_dir"), type="character", default="",
              help="scripts directory", metavar="character")
)

opt_parser <- OptionParser(option_list=option_list)

opt <- tryCatch(
  {
    parse_args(opt_parser)
  },
  error=function(cond) {
    message("Error: Provide database connection arguments appropriately.")
    message(cond)
    print_help(opt_parser)
    return(NA)
  },
  warning=function(cond) {
    message("Warning:")
```

```
    message(cond)
    return(NULL)
  },
  finally={
    message("Finished attempting to parse arguments.")
  }
)

if(length(opt) > 1){

  if (opt$database=="" | opt$user=="" |
      opt$password=="" | opt$host=="" ){
    print_help(opt_parser)
    stop("Database connection arguments are not supplied.n", call.=FALSE)
  }else{
    PGDATABASE <- opt$database
    POSTGRES_PASSWORD <- opt$password
    POSTGRES_USER <- opt$user
    PGHOST <- opt$host
    PGPORT <- "5432"
  }

  con <- DBI::dbConnect(RPostgres::Postgres(),
                        host = PGHOST,
                        port = PGPORT,
                        dbname = PGDATABASE,
                        user = PGHOST,
                        password = POSTGRES_PASSWORD
  )

  source("funciones_auxiliares.R")


################################################################################################
# Tablas LOCALES para tests
################################################################################################
#  se_vivienda_local <- sample_table(connec#tion = con,
                                     # schema #= 'clean',
                                     # the_table = 'cuis_historico_se_vivienda')
#
#  se_integrante_local <- sample_table(connection = con,
#                                      schema = 'clean',
```

```
#                                     the_table = 'cuis_historico_se_integrante')
#
#  edad_integrante_local <- sample_table(connection = con,
#                                        schema = 'clean',
#                                        the_table = 'cuis_historico_integrante') %>%
#                           dplyr::select(hogar_id, person_id, edad, anio, parentesco)
#
#  personas_vivienda_local <- sample_table(connection = con,
#                                          schema = 'clean',
#                                          the_table = 'cuis_historico_vivienda') %>%
#                             dplyr::select(hogar_id, total_personas)
#
#  df_integrante_local <- dplyr::left_join(se_integrante_local,
#                                          edad_integrante_local)
#
#  df_vivienda_local <- dplyr::left_join(se_vivienda_local,
#                                        personas_vivienda_local) %>%
#                       dplyr::select(-one_of(not_selected))
#
#  df_cuis_local <- dplyr::left_join(df_vivienda_local,
#                                              edad_integrante_local)
#
#  df_cuis_local = left_join(df_integrante_local, df_vivienda_local)
#
#  df_cuis_local_recoded = rename(df_cuis_local,
#                                 prestacion_incapacidad = trab_presta_a,
#                                 prestacion_afore = trab_presta_b,
#                                 segvol_medico = seg_volunt_b,
#                                 segvol_afore = seg_volunt_a)
#
  # Load YAML into core function
  carencias_chunk <- make_features('carencias.yaml')

  se_integrante <- large_table(connection = con,
                               schema = 'clean',
                               the_table = 'cuis_historico_se_integrante')

  edad_integrante <- large_table(connection = con,
                                 schema = 'clean',
                                 the_table = 'cuis_historico_integrante') %>%
                     dplyr::select(hogar_id, person_id, edad, anio, parentesco) %>%
                     dplyr::compute()
```

```
personas_vivienda <- large_table(connection = con,
                            schema = 'clean',
                            the_table = 'cuis_historico_vivienda') %>%
                     dplyr::select(hogar_id, total_personas) %>%
                     dplyr::compute()

not_selected <- c('llave_hogar_h', 'usr_creacion', 'csc_hogar',
                  'actualizacion_sedesol', 'data_date', 'fecha_creacion')


df_integrante <- dplyr::left_join(se_integrante, edad_integrante) %>%
                 dplyr::compute()

df_vivienda <- dplyr::left_join(se_vivienda, personas_vivienda) %>%
               dplyr::select(-one_of(not_selected)) %>%
               dplyr::compute()

df_cuis <- left_join(df_vivienda, df_integrante) %>%
           dplyr::compute()
# Rezago educativo
print('Rezago Educativo')
re_vars <- c('antec_esc', 'nivel_edu',
             'a_nacimiento', 'ic_rezago_educativo')
carencia_educacion <- carencias_chunk(df = df_integrante,
                                      chunkname = 'rezago_educativo',
                                      vars = re_vars,
                                      ids = c('hogar_id', 'person_id')) %>%
                      dplyr::compute()

print('Copy to features.cuis_carencia_educacion')
copy_to(con, carencia_educacion,
        dbplyr::in_schema("features",'cuis_carencia_educacion'),
        temporary = FALSE, overwrite = TRUE)

dbDisconnect(con)

# Carencia Alimentaria
con <- prev_connect()
se_vivienda <- large_table(connection = con,
                           schema = 'clean',
                           the_table = 'cuis_historico_se_vivienda')
```

```
edad_integrante <- large_table(connection = con,
                               schema = 'clean',
                               the_table = 'cuis_historico_integrante') %>%
                   dplyr::select(hogar_id, person_id, edad, anio, parentesco) %>%
                   dplyr::compute()

df_alimentaria = left_join(se_vivienda, edad_integrante)

print('Inseguridad Alimentaria')
alimentaria_vars <- c('min_edad', 'i_menores18',
               'total_ia', 'inseguridad_alimentaria',
               'ic_alimentacion')
carencia_alimentaria <- carencias_chunk(df = df_alimentaria,
                                        chunkname = 'carencia_alimentaria',
                                        vars = alimentaria_vars,
                                        ids = 'hogar_id') %>%
                        dplyr::compute()

print('Copy to features.cuis_carencia_alimentaria')
copy_to(con, carencia_alimentaria,
        dbplyr::in_schema("features",'cuis_carencia_alimentaria'),
        temporary = FALSE, overwrite = TRUE)

dbDisconnect(con)

# Carencia en Calidad y Espacios en la Vivienda
con <- prev_connect()

se_vivienda <- large_table(connection = con,
                           schema = 'clean',
                           the_table = 'cuis_historico_se_vivienda')

personas_vivienda <- large_table(connection = con,
                           schema = 'clean',
                           the_table = 'cuis_historico_vivienda') %>%
                     dplyr::select(hogar_id, total_personas) %>%
                     dplyr::compute()

df_vivienda <- dplyr::left_join(se_vivienda, personas_vivienda) %>%
               dplyr::select(-one_of(not_selected)) %>%
               dplyr::compute()
```

```
print('Calidad y Espacios en la Vivienda')
vivienda_vars <- c('indice_hacinamiento', 'ic_hacinamiento',
                   'ic_material_piso', 'ic_material_muros',
                   'ic_material_techos', 'ic_vivienda')
carencia_vivienda <- carencias_chunk(df = df_vivienda,
                                     chunkname = 'carencia_vivienda',
                                     vars = vivienda_vars,
                                     ids = 'hogar_id') %>%
                        dplyr::compute()

copy_to(con, carencia_vivienda,
        dbplyr::in_schema("features",'cuis_carencia_vivienda'),
        temporary = FALSE, overwrite = TRUE)

dbDisconnect(con)

# Carencia por Acceso a los Servicios Basicos en la Vivienda
con <- prev_connect()

df_servicios <- large_table(connection = con,
                            schema = 'clean',
                            the_table = 'cuis_historico_se_vivienda')


print('Acceso a los Servicios Basicos en la Vivienda')
servicios_vars <- c('ic_servicio_agua', 'ic_servicio_drenaje',
                    'ic_servicio_electricidad', 'ic_combustible',
                    'ic_servicios_basicos')
carencia_servicios <- carencias_chunk(df = df_servicios,
                                      chunkname = 'carencia_servicios',
                                      vars = servicios_vars,
                                      ids = 'hogar_id') %>%
                        dplyr::compute()

copy_to(con, carencia_servicios,
        dbplyr::in_schema("features",'cuis_carencia_servicios'),
        temporary = FALSE, overwrite = TRUE)

dbDisconnect(con)

# Carencia por Acceso a los Servicios de Salud
```

```
con <- prev_connect()

selected_trabajo <- c('person_id', 'hogar_id', 'realiza_act_lab',
                      'realiza_act_nolab', 'trabaja_subord', 'trabaja_indep',
                      'con_sueldo')

df_integrante <- large_table(connection = con,
                             schema = 'clean',
                             the_table = 'cuis_historico_se_integrante') %>%
                 dplyr::select(one_of(selected_trabajo))

edad_integrante <- large_table(connection = con,
                               schema = 'clean',
                               the_table = 'cuis_historico_integrante') %>%
                   dplyr::select(hogar_id, person_id, edad)

df_trabajo <- left_join(df_integrante, edad_integrante) %>%
              dplyr::compute()

print('Trabajo')
trabajo_vars <- c('trabajo', 'pea', 'tipo_trab')
trabajo_cuis <- carencias_chunk(df = df_trabajo,
                                chunkname = 'trabajo_cuis',
                                vars = trabajo_vars,
                                ids = c('person_id', 'hogar_id')) %>%
                dplyr::compute()

copy_to(con, trabajo_cuis,
        dbplyr::in_schema("features",'cuis_trabajo'),
        temporary = FALSE, overwrite = TRUE)

dbDisconnect(con)

con <- prev_connect()

not_selected <- c('llave_hogar_h', 'usr_creacion', 'csc_hogar',
                  'actualizacion_sedesol', 'data_date', 'fecha_creacion')

selected_integrante <- c('person_id', 'hogar_id',
                         'c_instsal_a', 'c_instsal_b',
                         'c_afilsal_a', 'c_afilsal_b')
```

```
se_integrante <- large_table(connection = con,
                             schema = 'clean',
                             the_table = 'cuis_historico_se_integrante') %>%
                 dplyr::select(one_of(selected_integrante))

selected_vivienda <- c('hogar_id', 'c_saludhoga', 'c_saludhogb')

se_vivienda <- large_table(connection = con,
                           schema = 'clean',
                           the_table = 'cuis_historico_se_vivienda') %>%
                 dplyr::select(one_of(selected_vivienda))

edad_integrante <- large_table(connection = con,
                               schema = 'clean',
                               the_table = 'cuis_historico_integrante') %>%
                   dplyr::select(hogar_id, person_id, edad, anio, parentesco)

df_trabajo <- large_table(connection = con,
                          schema = 'features',
                          the_table = 'cuis_trabajo')

df_integrante <- dplyr::left_join(se_integrante, edad_integrante)

df_salud <- dplyr::left_join(se_vivienda, df_integrante) %>%
            dplyr::left_join(df_trabajo) %>%
            dplyr::compute()

print('Acceso a los Servicios de Salud')
salud_vars <- c('atencion_medica', 'seguro_popular',
                'am_imss', 'am_issste', 'am_issste_estatal',
                'am_pemex', 'am_imss_prospera',
                'am_otra', 'inscrito_prestacion_lab',
                'inscrito_jubilacion', 'inscrito_familiar',
                'inscrito_muerte_aseg', 'inscrito_estudiante',
                'inscrito_contratacion', 'inscrito_familiar_otro',
                'sm_lab', 'sm_cv', 'salud_dir',
                'jefe_sm', 'cony_sm', 'hijo_sm',
                'acceso_jefe_sm', 'acceso_cony_sm',
                'acceso_hijo_sm', 'acceso_otros_sm',
                'asalud', 'ic_asalud')

print('Acceso a los Servicios de Salud')
```

```
 carencia_salud <- carencias_chunk(df = df_salud,
                                   chunkname = 'carencia_salud',
                                   vars = salud_vars,
                                   ids = c('person_id', 'hogar_id')) %>%
                        dplyr::compute()

 copy_to(con, carencia_salud,
         dbplyr::in_schema("features",'cuis_carencia_salud'),
         temporary = FALSE, overwrite = TRUE)
 dbDisconnect(con)

# Carencia por Acceso a la Seguridad Social
 con <- prev_connect()


 df_ingresos <- large_table(connection = con,
                            schema = 'clean',
                            the_table = 'cuis_historico_se_integrante') %>%
 dplyr::select(one_of(selected_ingresos))

print('Acceso a la Seguridad Social')
ingresos_vars <- c('pam', 'ingreso_pens')
ingresos_cuis <- carencias_chunk(df = df_ingresos,
                                 chunkname = 'ingresos_cuis',
                                 vars = ingresos_vars,
                                 ids = c('person_id', 'hogar_id'),
                                 return.all = TRUE) %>%
                dplyr::compute()


copy_to(con, ingresos_cuis,
         dbplyr::in_schema("features",'cuis_ingresos'),
         temporary = FALSE, overwrite = TRUE)
 dbDisconnect(con)

con <- prev_connect()
seguridad_social_vars <- c('sm_lab', 'jubilado', 'sm_cv',
                           'afore_cv', 'ss_dir',
                           'jefe_ss', 'cony_ss', 'hijo_ss',
                           'acceso_jefe_ss', 'acceso_cony_ss',
                           'acceso_hijo_ss', 'acceso_otros_ss',
                           'seguridad_social', 'ic_seguridad_social')
```

```
carencia_seguridad_social_ind <- carencias_chunk(df = ingresos_cuis,
                                                 chunkname = 'carencia_seguridad_social',
                                                 vars = seguridad_social_vars,
                                                 ids = c('hogar_id', 'person_id')) %>%
                                 dplyr::compute()

carencia_seguridad_social <- carencia_seguridad_social_ind %>%
                             dplyr::group_by(hogar_id) %>%
                             dplyr::summarise_at(seguridad_social_vars,
                                                 funs(max(., na.rm = TRUE))) %>%
                             dplyr::compute()

print('Joining all tables')
cuis_carencias <- carencia_educacion %>%
                  dplyr::left_join(carencia_alimentaria) %>%
                  dplyr::left_join(carencia_vivienda) %>%
                  dplyr::left_join(carencia_servicios) %>%
                  dplyr::left_join(dplyr::select(carencia_salud, -sm_lab, -sm_cv)) %>%
                  dplyr::left_join(carencia_seguridad_social)

print('Copy to features.cuis_carencias')
copy_to(con, cuis_carencias,
         dbplyr::in_schema("features",'cuis_carencias'),
         temporary = FALSE, overwrite = TRUE)
 dbDisconnect(con)

}
```

USO DE DATOS MASIVOS PARA LA EFICIENCIA DEL ESTADO Y LA INTEGRACIÓN REGIONAL

16 de agosto de 2018

FJR-2-ATN/OC 15822-RG

SEDESOL 2018