

Proyecto Final: Mapping the geographic distribution of names

Elaborado por: Jorge Gabriel Leyva Durán.

Fecha: Viernes 25 de Abril de 2025.

Objetivo

Se colectarán datos de la distibución geográfica de apellidos en Alemania. Se ha visto que a pesar de la movilidad laboral, los apellidos que han se han originado en cierta región del país, aún continúan en dichas zonas. Además de su valor científico, los mapas de nombre tienen un valor especial a aquellos interesados en las raíces de sus orígenes.

Otro propósito de este trabajo es afrontar problemas tales como:

1. Datos incompletos.
2. Datos que están relacionados pero desperdigados en un árbol HTML.
3. La funcionalidad del límite de entradas (hits) en una página.
4. Parámetros URL no documentados.

Se usará un directorio telefónico para obtener información acerca de los apellidos que se distribuyen en las zonas de toda Alemania. En general, para llevar a cabo esto se seguirán los siguientes pasos:

1. Identificar un directorio online que proporcione la información que necesitamos.
2. Familiarizarse con la estructura de la página y elegir un procedimiento para extraer la información.
3. Aplicar dicho procedimiento: Extraer información, limpiar los datos y documentar problemas que a primera vista no se hayan contemplado que ocurran durante el código.
4. Visualizar y analizar los datos.
5. Generalizar la tarea de “scraping”.

Importación de la base de datos

En este capítulo se discute acerca de la visualización de datos geográficos en R. En particular, los datos se extraerán de la página: https://www.dasoertliche.de/?zvo_ok=&plz=&quarter=&district=&ciid=&kw=Feuerstein&ci=&kgs=&buab=&zbuab=&form_name=search_nat

Nos debemos asegurar que las siguientes paqueterías estén instaladas, si no, instalarlas

- install.packages("RCurl")
- install.packages("XML")
- install.packages("stringr")
- install.packages("maptools") #Cambiar por sf y terra
- install.packages(c("sf", "terra", "spatstat"))
- install.packages("rgdal") #No es necesaria si se tienen las paqueterías sf, terra
- install.packages("maps")
- install.packages("TeachingDemos")
- install.packages("dplyr")
- install.packages("ggplot2")

En particular, la paquetería **maptools** ya es una paquetería que no se usa, por lo que esta se sustituirá por las paqueterías **sf**, **terra** y **spatstat**.

Una vez cargadas las paqueterías, solo se mandan llamar

```

library(RCurl) #si funciona
library(XML) #Si funciona
library(stringr) #Si funciona
library(sf)

## Linking to GEOS 3.13.0, GDAL 3.10.1, PROJ 9.5.1; sf_use_s2() is TRUE
library(terra)

## terra 1.8.29
library(spatstat)

## Cargando paquete requerido: spatstat.data
## Cargando paquete requerido: spatstat.univar
## spatstat.univar 3.1-2
## Cargando paquete requerido: spatstat.geom
## spatstat.geom 3.3-6
##
## Adjuntando el paquete: 'spatstat.geom'
## The following objects are masked from 'package:terra':
##
##      area, delaunay, is.empty, rescale, rotate, shift, where.max,
##      where.min
## Cargando paquete requerido: spatstat.random
## spatstat.random 3.3-3
## Cargando paquete requerido: spatstat.explore
## Cargando paquete requerido: nlme
## spatstat.explore 3.4-2
## Cargando paquete requerido: spatstat.model
## Cargando paquete requerido: rpart
## spatstat.model 3.3-4
## Cargando paquete requerido: spatstat.linnet
## spatstat.linnet 3.2-5
##
## spatstat 3.3-1
## For an introduction to spatstat, type 'beginner'
library(maps) #si funciona
library(TeachingDemos) #si funciona

##
## Adjuntando el paquete: 'TeachingDemos'
## The following object is masked from 'package:terra':
##
##      dots

```

```

library(dplyr)

##
## Adjuntando el paquete: 'dplyr'
## The following object is masked from 'package:nlme':
##
##     collapse

## The following objects are masked from 'package:terra':
##
##     intersect, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
library(ggplot2)

```

Y ahora, debemos crear una lista para almacenar los datos en un archivo HTML.

```
resultados <- list()
```

Dado que en el URL obtenido, se muestran 25 resultados por pantalla y en total son 310 registros, se hace un ciclo para poder guardar todos los resultados de las entradas privadas.

```

for (i in seq(1, 310, by = 25)) {
  res <- getForm("https://www.dasoerliche.de/",
    .params = c(kw = "Feuerstein",
                form_name = "search_nat",
                atfilter = "1",
                recFrom = as.character(i)))
  resultados[[length(resultados) + 1]] <- res
  Sys.sleep(1) # Evitar sobrecargar el servidor
}

```

En este código, se consultan ciertos parámetros para obtener los resultados. En particular tenemos:

- *kw*: La palabra que estamos buscando. En nuestro caso es **Feuerstein**.
- *form_name*: Tipo de búsqueda. Obtenida directamente del URL
- *atfilter*: El filtro de búsqueda. En este caso como *atfilter=1*, el valor 1 significa las entradas privadas.
- *recForm*: Para obtener los caracteres y que sea el directorio en general.

Ahora, se deben combinar todos los resultados HTML en una sola cadena

```
tb <- paste(resultados, collapse = "\n")
```

El contenido se guarda en el objeto **tb** que se llamará **phonebook_feuerstein.html** para no usar la página web de nuevo.

```
dir.create("phonebook_feuerstein")
```

```

## Warning in dir.create("phonebook_feuerstein"): 'phonebook_feuerstein' already
## exists
write(tb, file = "phonebook_feuerstein/phonebook_feuerstein.html")

```

Y de esta manera, se puede accesar a la informacion de la página sin necesidad de usar el URL otra vez conectándose a Internet con el siguiente comando.

```
tb_parse <- htmlParse("phonebook_feuerstein/phonebook_feuerstein.html",
                      encoding = "UTF-8")
```

Se deben extraer los apellidos *Feuerstein* con los siguientes comandos y se guardan con el nombre **surnames**

```
xpath <- "//a[@class='hitlnk_name']" #es este comando diferente al otro del libro, pues tiene la clase  
surnames <- xpathSApply(tb_parse, xpath, xmlValue)  
surnames[1:3]
```

Se hace lo mismo para los códigos postales, donde en el código HTML se encuentran en el apartado de **address**

```
xpath <- "//address" #El código postal se encuentra en <address> y se extrae el código postal
address <- xpathSApply(tb_parse, xpath, xmlValue) #idem
zipcodes <- regmatches(address, regexpr("\\b\\d{5}\\b", address)) #regmatches() extrae partes de texto
zipcodes[1:3]
```

```
## [1] "64625" "68549" "68526"
```

Finalmente, se comparan el número de zipcodes y de surnames

`length(surnames)`

```
## [1] 311
```

```
length(zipcodes)
```

[1] 296

Esto se debe a que existen entradas que no tienen una dirección particular, y por tanto no tienen un código postal. De esta manera, debemos extraer del código HTML solo aquellas entradas que tienen entrada de apellido y además código postal. Dado que se busca establecer un mapa de los apellidos en Alemania, un apellido sin código postal no sirve de nada. Para esto usamos los siguientes comandos:

Primero buscamos los bloques que contengan la clase `hit`, es decir `//div[contains(@class, 'hit')]`.

```
hit nodes <- getNodeSet(tb parse, "//div[contains(@class, 'hit')]")
```

Luego, se inicializan los vectores `names_vec` y `zipcodes_vec` y se recorre todo el archivo HTML, donde se extraerán los nombres, los códigos postales, y adicionalmente se hacen los saltos de línea adecuados para una mejor lectura del archivo.

```

# Inicializar vectores vacíos
names_vec <- c()
zipcodes_vec <- c()

# Recorrer resultados
for (hit in hit_nodes) #Bucle que toma cada <div class="hit">
{
  # Extraer nombre
  name_node <- getNodeSet(hit, "./a[@class='hitlnk_name']")
  name <- if (length(name_node) > 0) xmlValue(name_node[[1]]) else NA

```

```

# Extraer dirección y código postal
addr_node <- getNodeSet(hit, "./address")
address <- if (length(addr_node) > 0) xmlValue(addr_node[[1]]) else ""
zipcodes <- str_extract(address, "\\b\\d{5}\\b")

# Limpiar nombre
clean_name <- str_trim(str_replace_all(name, "(\\n|\\t|\\r| {2,})", ""))

# Agregar solo si ambos valores no son NA
if (!is.na(clean_name) && !is.na(zipcodes)) {
  names_vec <- c(names_vec, clean_name)
  zipcodes_vec <- c(zipcodes_vec, as.numeric(zipcodes))
}
}

```

Se crea el data frame: `entries_df`, donde se verifica además que no existan entradas duplicadas.

```

# Crear data frame
entries_df <- data.frame(plz = zipcodes_vec, name = names_vec)

# Eliminar duplicados exactos
entries_df <- unique(entries_df)

```

En particular, en la importación de la base de datos del código HTML. Hay un dato que se duplica sin nombre. Esta se encuentra en el renglón 130 del data frame, por lo que se hace lo siguiente:

```
#Hay un error en la importación del data frame, la línea de código
entries_df <- entries_df[-130, ]
```

Reordenando los renglones, obtenemos el siguiente data frame:

```

#Se reordenan los números de los renglones del data-frame
rownames(entries_df) <- NULL

# Mostrar
head(entries_df)

##      plz                      name
## 1 64625        Bertsch-Feuerstein Lilli
## 2 68549    Bierig-Feuerstein Brigitte u. Feuerstein Norbert
## 3 68526          Blatt Karl u. Feuerstein-Blatt Ursula
## 4 45133                  Feuerstein
## 5 50733                  Feuerstein
## 6 89233                  Feuerstein

```

Mapeo de nombres

Una vez importados los datos, y que ya se haya hecho la extracción y limpieza de datos, se procederá a hacer las gráficas correspondientes.

Se deben emparejar las coordenadas geográficas de Alemania en conjunto con los códigos postales obtenidos. Para realizar esto, se usa una base de datos que vincula dichas coordenadas con los códigos postales. Esta se encuentra con el siguiente link: <https://ratopi.github.io/opengeodb/PLZ.tab>, pero antes, se debe crear un directorio llamado: `geo_germany` con el comando siguiente

```

dir.create("geo_germany")

## Warning in dir.create("geo_germany"): 'geo_germany' already exists

```

Y ahora, usando el link mostado, se descarga y se guarda con el nombre `plz_de.txt`. De modo que leyendo este último archivo con

```
plz_df <- read.delim("geo_germany/plz_de.txt", stringsAsFactors=FALSE, encoding = "UTF-8")
plz_df[1:3, ]
```

```
##   X.loc_id  plz      lon      lat      Ort
## 1      5078 1067 13.72107 51.06003 Dresden
## 2      5079 1069 13.73891 51.03956 Dresden
## 3      5080 1097 13.74397 51.06675 Dresden
```

donde obtenemos las longitudes, latitudes y lugares de acuerdo al código postal (ordenado desde el código postal con valor más pequeño encontrado hasta el de mayor valor).

Se fusiona ahora, la información del data frame anterior con `plz_df` usando la variable identificadora adjunta `plz`:

```
places_geo <- merge(entries_df, plz_df, by = "plz", all.x = TRUE)
places_geo[1:3, ]
```

```
##   plz           name X.loc_id      lon      lat      Ort
## 1 4924    Feuerstein Jana     5343 13.37692 51.51678 Bad Liebenwerda
## 2 6712    Feuerstein Reinhold  5408 12.13265 51.03134 Zeitz
## 3 7318    Feuerstein Horst   12863 11.32997 50.62594 Saalfeld / Saale
```

Para enriquecer el mapa con fronteras administrativas usamos información de GADM. Se descarga el archivo `gadm41_DEU.shp.zip` que es el mapa con la forma de Alemania. Este archivo se extrajo de la página https://gadm.org/download_country.html, donde el `shapefile` y se descomprimió directamente en el directorio creado, `geo_germany`.

Una vez actualizado, el contenido del directorio `geo_germany` es:

```
dir("geo_germany")
```

```
## [1] "gadm41_DEU_0.cpg" "gadm41_DEU_0.dbf" "gadm41_DEU_0.prj" "gadm41_DEU_0.shp"
## [5] "gadm41_DEU_0.shx" "gadm41_DEU_1.cpg" "gadm41_DEU_1.dbf" "gadm41_DEU_1.prj"
## [9] "gadm41_DEU_1.shp" "gadm41_DEU_1.shx" "gadm41_DEU_2.cpg" "gadm41_DEU_2.dbf"
## [13] "gadm41_DEU_2.prj" "gadm41_DEU_2.shp" "gadm41_DEU_2.shx" "gadm41_DEU_3.cpg"
## [17] "gadm41_DEU_3.dbf" "gadm41_DEU_3.prj" "gadm41_DEU_3.shp" "gadm41_DEU_3.shx"
## [21] "gadm41_DEU_4.cpg" "gadm41_DEU_4.dbf" "gadm41_DEU_4.prj" "gadm41_DEU_4.shp"
## [25] "gadm41_DEU_4.shx" "plz_de.txt"
```

De los archivos mostrados se tiene hacen las siguientes observaciones:

- archivo.shp (contiene datos geográficos)
- archivo.dbf (contiene datos de atributos adjuntos a objetos geográficos)
- archivo.shx (contiene el índice de los datos geográficos)
- archivo.cpg (archivo de página de códigos que se usa para identificar el conjunto de caracteres que se usarán en un archivo)
- archivo.prj (opcionales y contienen información acerca del formato de proyección del mapa)

Haciendo uso de la paquetería `sf` se procede a hacer la lectura de los archivos `shapefile`, del contorno de Alemania y también el mapa que contiene las divisiones por estados de Alemania (`gadm41_DEU_0.shp` y `gadm41_DEU_1.shp`, respectivamente)

```
# Leer el shapefile de Alemania (admin 0 - fronteras del país)
map_germany <- st_read(str_c(getwd(), "/geo_germany/gadm41_DEU_0.shp"))
```

```
## Reading layer `gadm41_DEU_0` from data source
##   `D:\Programación de Aplicaciones Web Orientada a Objetos 25-I\Proyecto Final\geo_germany\gadm41_DEU_0.shp'
```

```

##   using driver `ESRI Shapefile'
## Simple feature collection with 1 feature and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: 5.866251 ymin: 47.2707 xmax: 15.04181 ymax: 55.05653
## Geodetic CRS:  WGS 84

# Establecer la proyección en WGS84 (longitud y latitud)
map_germany <- st_transform(map_germany, crs = "+proj=longlat +ellps=WGS84 +datum=WGS84")

# Leer el shapefile de las regiones de Alemania (admin 1 - estados)
map_germany_laender <- st_read(str_c(getwd(), "/geo_germany/gadm41_DEU_1.shp"))

## Reading layer `gadm41_DEU_1` from data source
##   `D:\Programación de Aplicaciones Web Orientada a Objetos 25-I\Proyecto Final\geo_germany\gadm41_DEU_1.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 16 features and 11 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: 5.866251 ymin: 47.2707 xmax: 15.04181 ymax: 55.05653
## Geodetic CRS:  WGS 84

# Establecer la proyección también para este mapa
map_germany_laender <- st_transform(map_germany_laender, crs = "+proj=longlat +ellps=WGS84 +datum=WGS84")

```

NOTA: WGS84 es un sistema de coordenadas que permite determinar la posición de cualquier punto en la Tierra.

Ahora, se transforman las coordenadas de los apellidos “Feuerstein” encontrados a puntos de coordenadas que se puedan visualizar en el mapa:

```
coords_sf <- st_as_sf(places_geo, coords = c("lon", "lat"), crs = 4326)
```

NOTA: EPSG 4326 es el código que identifica al sistema de referencia WGS84, el cual se utiliza para representar la Tierra a nivel mundial. Es equivalente a WGS84.

A continuación, se extraerán las ciudades alemanas con una población mayor a 450,000 habitantes. Se usan los comandos siguientes:

```
data("world.cities")
```

```
## 9903 Duisburg      Germany 502251      0 POINT (6.75 51.43)
```

Se grafica ahora el contorno de Alemania:

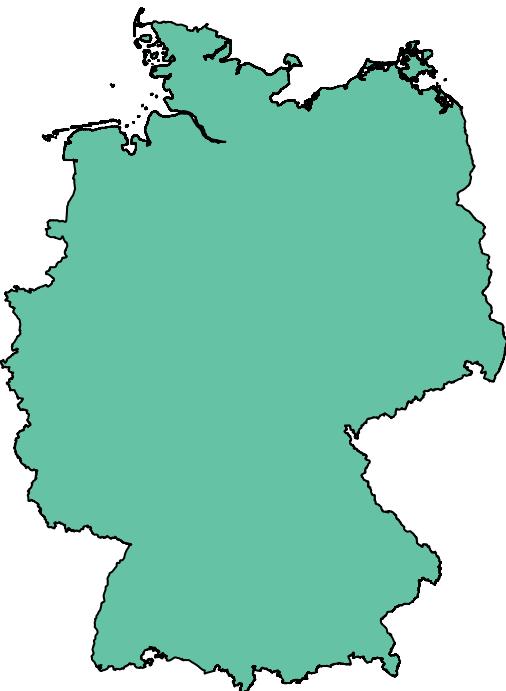
```
map_germany <- st_read(str_c(getwd(), "/geo_germany/gadm41_DEU_0.shp"))
```

```
## Reading layer `gadm41_DEU_0' from data source  
##   'D:\Programación de Aplicaciones Web Orientada a Objetos 25-I\Proyecto Final\geo_germany\gadm41_DEU_0.shp'  
##   using driver 'ESRI Shapefile'  
## Simple feature collection with 1 feature and 2 fields  
## Geometry type: MULTIPOLYGON  
## Dimension: XY  
## Bounding box: xmin: 5.866251 ymin: 47.2707 xmax: 15.04181 ymax: 55.05653  
## Geodetic CRS: WGS 84  
plot(map_germany)
```

GID_0



COUNTRY



Luego se grafica la división geopolítica de Alemania:

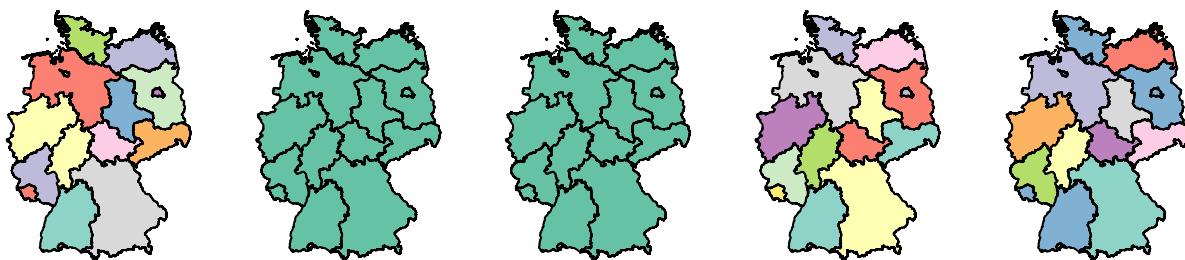
```
map_germany_laender <- st_read(str_c(getwd(), "/geo_germany/gadm41_DEU_1.shp"))
```

```
## Reading layer `gadm41_DEU_1' from data source  
##   'D:\Programación de Aplicaciones Web Orientada a Objetos 25-I\Proyecto Final\geo_germany\gadm41_DEU_1.shp'  
##   using driver 'ESRI Shapefile'  
## Simple feature collection with 16 features and 11 fields  
## Geometry type: MULTIPOLYGON  
## Dimension: XY  
## Bounding box: xmin: 5.866251 ymin: 47.2707 xmax: 15.04181 ymax: 55.05653  
## Geodetic CRS: WGS 84
```

```
plot(map_germany_laender)
```

```
## Warning: plotting the first 10 out of 11 attributes; use max.plot = 11 to plot
## all
```

GID_1 GID_0 COUNTRY NAME_1 VARNAME_1



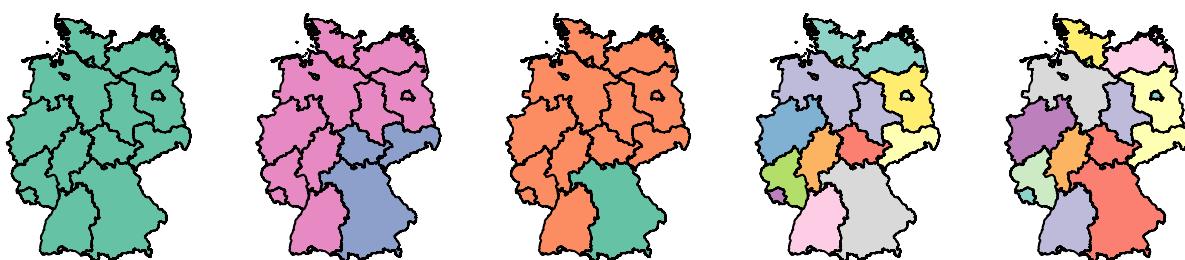
NL_NAME_1

TYPE_1

ENGTYPE_1

CC_1

HASC_1



Se añaden las coordenadas de las ciudades

```
coords_cities_sf <- st_as_sf(cities_ger, coords = c("long", "lat"), crs = 4326)
```

Se grafica la distribución geográfica del apellido “Feuerstein” en Alemania:

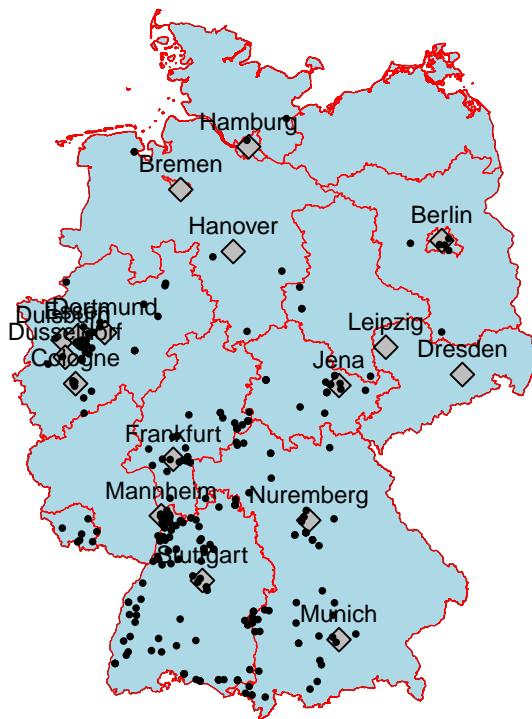
```
# Crear el gráfico
ggplot() +
  # Dibujar el mapa de Alemania
  geom_sf(data = map_germany, fill = "lightblue", color = "black") +
  # Dibujar las regiones (Laender) sobre el mapa de Alemania
  geom_sf(data = map_germany_laender, fill = NA, color = "red") +
  # Agregar puntos para las coordenadas de las ciudades de Alemania
  geom_sf(data = coords_cities_sf, color = "black", fill = "grey", shape = 23, size = 3) +
  # Agregar los valores de Feuerstein
  geom_sf(data = coords_sf, color = "black", fill = "black", shape = 16, size = 1) +
  
  # Etiquetar las ciudades
  geom_text(data = cities_ger, aes(x = long, y = lat, label = name), size = 3, vjust = -1) +
  theme_minimal() + theme(panel.grid.major = element_blank(), # Eliminar cuadrícula mayor
                         panel.grid.minor = element_blank(), # Eliminar cuadrícula menor
                         axis.text.x = element_blank(), # Eliminar números en el eje x
                         axis.text.y = element_blank(), # Eliminar números en el eje y
```

```

axis.title.x = element_blank(),          # Eliminar título del eje x
axis.title.y = element_blank() +          # Eliminar título del eje y
labs(title = "Mapa de Alemania con la distribución geográfica de 'FeuerStein'")

```

Mapa de Alemania con la distribución geográfica de 'FeuerStein'



Naturalmente, uno buscaría repetir este ejercicio una y otra vez con ligeras modificaciones. Se desarrollará un conjunto de funciones que generalizarán el scraping, parsing y mapeo para casos diferentes. En nuestro caso, la información rara vez cambia con el transcurso del tiempo, por lo tanto se quisiera realizar lo hecho anteriormente con distintos nombres, así también como verificar los mapeos para cada nombre.

Se dividirá el procedimiento en tres funciones:

1. Una función de scraping.
2. Una función de parsing y de limpieza de datos.
3. Una función de mapeo.

Función de scraping.

La función generalizada de scraping se ve de la siguiente manera (para las primeras entradas):

```

namesScrape <- function(phonenumber, update.file = FALSE) {
  ## transform phonenumber
  phonenumber <- tolower(phonenumber)
  ## load libraries
  x <- c("stringr", "RCurl", "XML")
  lapply(x, require, character.only=TRUE)
  ## create folder
  dir.create(str_c("phonebook_", phonenumber), showWarnings = FALSE)
  filename <- str_c("phonebook_", phonenumber, "/phonebook_", phonenumber,

```

```

".html")
if (file.exists(filename) & update.file == FALSE) {
  message("Data already scraped; using data from ", file.info(
    filename)$mtime)
} else {
  ## retrieve and save html
  resultados <- list()
  for (i in seq(1, 310, by = 25)) {
    res <- getForm("https://www.dasoertliche.de/",
      .params = c(kw = phonename,
        form_name = "search_nat",
        atfilter = "1",
        recFrom = as.character(i)))
    resultados[[length(resultados) + 1]] <- res
    Sys.sleep(1) # Evitar sobrecargar el servidor
  }
  tb <- paste(resultados, collapse = "\n")
  write(tb, file = filename)
}
}

```

Función de parsing y de limpieza de datos.

```

namesParse <- function(phonename) {
  filename <- str_c("phonebook_", phonename, "/phonebook_", phonename,
    ".html")
  ## load libraries
  x <- c("stringr", "XML")
  lapply(x, require, character.only = TRUE)
  ## parse html
  tb_parse <- htmlParse(filename, encoding = "UTF-8")

  ## retrieve zipcodes and names
  hit_nodes <- getNodeSet(tb_parse, "//div[contains(@class, 'hit')])" #busca y selecciona los bloques que cumplen la condición

  # Inicializar vectores vacíos
  names_vec <- c()
  zipcodes_vec <- c()

  # Recorrer resultados
  for (hit in hit_nodes) #Bucle que toma cada <div class="hit">
  {
    # Extraer nombre
    name_node <- getNodeSet(hit, "./a[@class='hitlnk_name'])")
    name <- if (length(name_node) > 0) xmlValue(name_node[[1]]) else NA

    # Extraer dirección y código postal
    addr_node <- getNodeSet(hit, "./address")
    address <- if (length(addr_node) > 0) xmlValue(addr_node[[1]]) else ""
    zipcodes <- str_extract(address, "\\b\\d{5}\\b")

    # Limpiar nombre
  }
}

```

```

clean_name <- str_trim(str_replace_all(name, "(\\n|\\t|\\r| {2,})", ""))
# Agregar solo si ambos valores no son NA
if (!is.na(clean_name) && !is.na(zipcodes)) {
  names_vec <- c(names_vec, clean_name)
  zipcodes_vec <- c(zipcodes_vec, as.numeric(zipcodes))
}
}

# Crear data frame
entries_df <- data.frame(plz = zipcodes_vec, name = names_vec)

# Eliminar duplicados exactos
entries_df <- unique(entries_df)

# Se reordenan los números de los renglones del data-frame
rownames(entries_df) <- NULL

## match coordinates to zipcodes
plz_df <- read.delim("geo_germany/plz_de.txt", stringsAsFactors=FALSE, encoding = "UTF-8")
geodf <- merge(entries_df, plz_df, by = "plz", all.x = TRUE)
geodf <- geodf[!is.na(geodf$lon),]
## return data frame
geodf <- geodf[, !names(geodf) %in% "X.loc_id"]
return(geodf)
}

```

Función de mapeo.

```

namesPlot <- function(geodf, phonename, show.map = TRUE, save.pdf = TRUE,
                      minsize.cities = 450000, add.cities = "", print.names = FALSE) {
  # Cargar las librerías necesarias
  packages <- c("stringr", "sf", "terra", "spatstat", "maps",
               "TeachingDemos", "dplyr", "ggplot2")
  lapply(packages, require, character.only = TRUE)

  # Convertir el data frame geodf en un objeto sf
  coords_sf <- st_as_sf(geodf, coords = c("lon", "lat"), crs = 4326)

  # Preparar los mapas: leer y transformar los shapefiles a WGS84
  map_germany <- st_read(file.path(getwd(), "geo_germany/gadm41_DEU_0.shp"), quiet = TRUE)
  map_germany <- st_transform(map_germany, crs = "+proj=longlat +ellps=WGS84 +datum=WGS84")

  map_germany_laender <- st_read(file.path(getwd(), "geo_germany/gadm41_DEU_1.shp"), quiet = TRUE)
  map_germany_laender <- st_transform(map_germany_laender, crs = "+proj=longlat +ellps=WGS84 +datum=WGS84")

  # Añadir ciudades grandes a partir del dataset world.cities
  data("world.cities", package = "maps")
  cities_ger <- subset(world.cities,
                        country.etc == "Germany" & (pop > minsize.cities | name %in% add.cities))
  coords_cities_sf <- st_as_sf(cities_ger, coords = c("long", "lat"), crs = 4326)

  # Construir el mapa

```

```

p <- ggplot() +
  geom_sf(data = map_germany, fill = "lightblue", color = "black") +
  geom_sf(data = map_germany_laender, fill = NA, color = "red") +
  geom_sf(data = coords_cities_sf, color = "black", fill = "grey", shape = 23, size = 3) +
  geom_sf(data = coords_sf, color = "black", fill = "black", shape = 16, size = 1) +
  geom_text(data = cities_ger, aes(x = long, y = lat, label = name), size = 3, vjust = -1) +
  theme_minimal() + theme(panel.grid.major = element_blank(), # Eliminar cuadrícula mayor
                         panel.grid.minor = element_blank(), # Eliminar cuadrícula menor
                         axis.text.x = element_blank(),      # Eliminar números en el eje x
                         axis.text.y = element_blank(),      # Eliminar números en el eje y
                         axis.title.x = element_blank(),    # Eliminar título del eje x
                         axis.title.y = element_blank()) +   # Eliminar título del eje y
  labs(title = str_c("Mapa de Alemania con la distribución geográfica de ", toupper(phoneName)))

if (print.names){
  p <- p + geom_text(data = geodf, aes(x = lon, y = lat, label = name),
                      size = 2.5, vjust = -0.5, check_overlap = TRUE)
}

# Guardar en PDF si se solicita
if (save.pdf) {
  pdf_file <- file.path(str_c("phonebook_", phoneName), str_c("map-", phoneName, ".pdf"))
  # Crear el directorio si no existe
  if (!dir.exists(file.path("phonebook_", phoneName))) {
    dir.create(file.path("phonebook_", phoneName), recursive = TRUE)
  }
  pdf(pdf_file, height = 10, width = 7.5, family = "URWTimes")
  print(p)
  dev.off()
}

# Mostrar el mapa en pantalla si se solicita
if (show.map) {
  print(p)
}

```

Hacemos distintas pruebas

Prueba 1

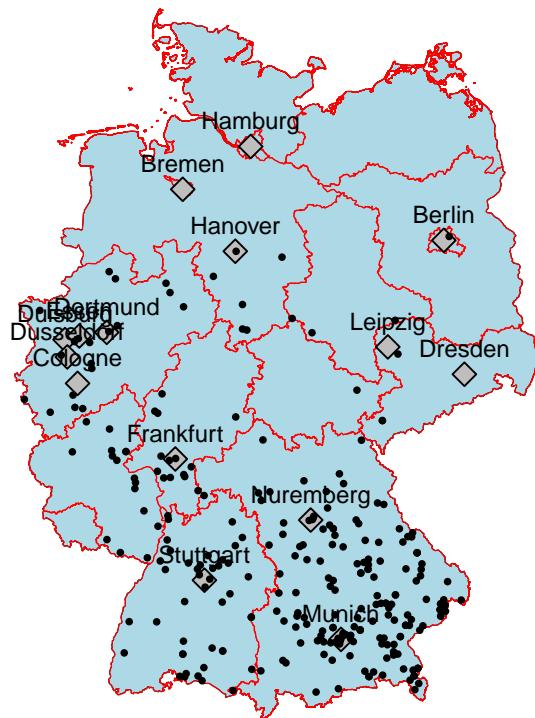
```

namesScrape("Gruber")

## Data already scraped; using data from 2025-04-21 17:28:50
gruber_df <- namesParse("Gruber")
namesPlot(gruber_df, "Gruber", save.pdf = FALSE, show.map = TRUE, print.names = FALSE)

```

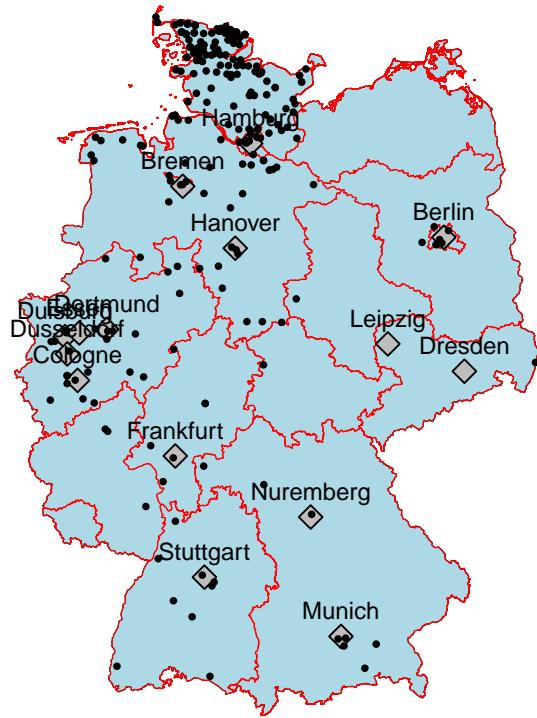
Mapa de Alemania con la distribución geográfica de GRUBI



Prueba 2

```
namesScrape("Petersen")  
  
## Data already scraped; using data from 2025-04-21 17:29:30  
petersen_df <- namesParse("Petersen")  
namesPlot(petersen_df, "Petersen", save.pdf = FALSE, show.map = TRUE, print.names = FALSE)
```

Mapa de Alemania con la distribución geográfica de PETEF



```
### Prueba 3
```

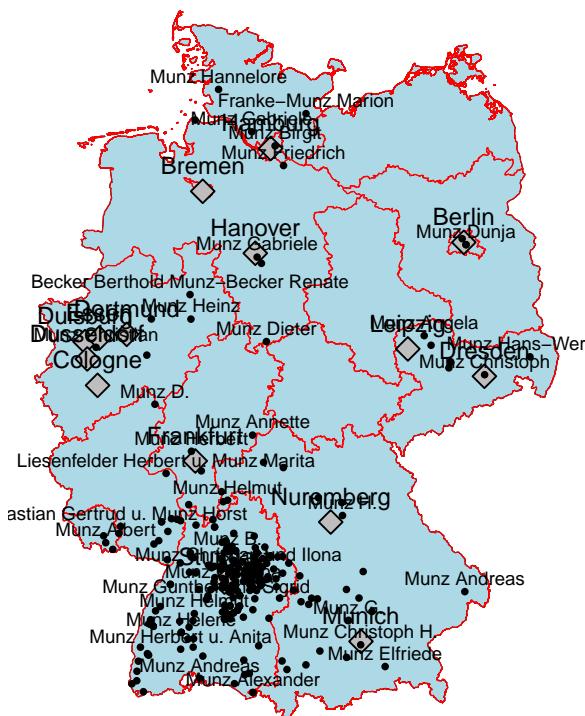
```
namesScrape("Munz")
```

```
## Data already scraped; using data from 2025-04-21 17:30:04
```

```
munz_df <- namesParse("Munz")
```

```
namesPlot(munz_df, "Munz", save.pdf = FALSE, show.map = TRUE, print.names = TRUE)
```

Mapa de Alemania con la distribución geográfica de MUNZ



Resumen

El código presentado es solo el primer paso a un análisis meticuloso para el análisis de la distribución de los apellidos. Hay muchos problemas en la calidad de los datos que debe ser considerada cuando se hace algún tipo de extracción para fines científicos. Primero, la limitación de los datos. Dado que las entradas están ordenadas alfabéticamente, no hay aleatoriedad en la elección de la base de datos (para este caso).

Cuando se trata con contenido dinámico, vale más la pena ver el código fuente de la página que se esté consultando. Se debe de tener un conocimiento básico de cómo los funcionan los parámetros en un GET, cuáles son sus límites, y si hay alguna otra manera de poder extraer la información (quizás más sencilla). Respecto al uso de los datos geográficos, la lección aprendida es que si se tiene un identificador geográfico (código postal), es más fácil enriquecer la base de datos, y por tanto la visualización de los datos será más amigable para el usuario.

Como conclusiones finales, este es un excelente primer acercamiento para obtener información de los directorios telefónicos, y poder mapear los apellidos en zonas de cualquier país. Para poder lograr esto es necesario obtener identificadores geográficos que nos faciliten esta tarea y tener acceso a mapas con coordenadas para lograr el objetivo.

El hecho de que el formato de las páginas que tienen esta información puede llegar a variar, naturalmente la función de scrapping cambiará y se deberá adecuar para que funcione, sin embargo, de ahí en fuera (considerando lo escrito anteriormente) debe ser un método bastante accesible.