Listings anonymization (price and coords)

David Rey

29 of julio - 2022

Introduction

This markdown file contains the code used for the anonymization of the package's listing records (due to legal requirement). It introduces a small random disturbance in the total price and exact position.

Set parameters

Set process parameters

```
dir_geom_path <- paste0(here::here(),"/data/wkt/")
dir_path <- paste0(here::here(),"/data/assets/raw/")
output_path <- str_remove(dir_path,  "raw/")

#
# Process parameters (expected from a PARAM list)
# here we define some settings
#
COUNTRY = 'es'     # Spain
TYPOLOGY = 'home'  # This type refers to flats/apartments
OPERATION = 'rent' # This refers to rental
YEAR = 2018
OUTPUT_FORMAT = 'csv.gz'   # Format used for the generated data set
CLEAN_CITYNAME = 'Madrid' # City name: Madrid, Barcelona or Valencio

knitr::opts_chunk$set(echo = TRUE)
```

We set the used CRS, input parameters are coded using and EPSG:4326 and we need to use a projected CRS, in this case we will use "+proj=utm +zone=30 +ellps=WGS84 +datum=WGS84 +units=m +no_defs".

```
#
# Set the projections used
#
projcrs_src = '+proj=longlat +datum=WGS84 +no_defs'
projcrs_dest = '+proj=utm +zone=30 +ellps=WGS84 +datum=WGS84 +units=m +no_defs'
```

Load data

Load input data (with actual coordinates and prices).

```
#
# Set input file names
#

file_name = paste0(CLEAN_CITYNAME,'_polygons.',OUTPUT_FORMAT)

#
# Load file and project to make the spatial transformations
#

polygons.sf <- read_delim(paste0(dir_geom_path, file_name),
                          delim = ";") |>
  st_as_sf(wkt = 'WKT',
           crs = projcrs_src) |>
  st_transform(projcrs_dest)

#
# Plot geometry
#

plot(st_geometry(polygons.sf))
```

Load and preprocess listing data

Load data

Load data and remove some extreme values (taken from idealista history data).

```
#
# Variable boundaries come are defined as valid values
#
max_unitprice = list('sale' = 10443.64, 'rent' = 47.3)
min_price = list('sale' = 5000, 'rent' = 100)
max_price = list('sale' = 4E6, 'rent' = 20E3)
range_area = c(20, 500)

file_name = paste0(COUNTRY,'_',TYPOLOGY,
                   '_',OPERATION,'_',CLEAN_CITYNAME,
                   '_',YEAR,'.',OUTPUT_FORMAT)

# Load individual records
assets.sf <- read_delim(paste0(dir_path, file_name),  delim = ";")
```

Fix cadastral construction year

Given we take as right the year user fills in we need to fix it, if we see extreme values < 1500 or in the future we assign the median year in the area:

```
#
# Impute construction year if it is out of bounds ]-inf, MIN_YEAR] or [YEAR, inf]
#

MIN_YEAR = 1500
imputed_years.sf <- assets.sf |>
  dplyr::group_by(LOCATIONID, .drop=F) |>
  dplyr::filter(CADCONSTRUCTIONYEAR > MIN_YEAR & CADCONSTRUCTIONYEAR <= YEAR) |>
  dplyr::summarise(IMPUTED_YEAR=as.integer(median(CADCONSTRUCTIONYEAR), .groups=NULL))

assets.sf <- assets.sf |>
  dplyr::inner_join(imputed_years.sf, by='LOCATIONID') |>
  # 1st attempt: Impute with idealista construction year
  dplyr::mutate(CADCONSTRUCTIONYEAR = ifelse(CADCONSTRUCTIONYEAR < MIN_YEAR
                                             | CADCONSTRUCTIONYEAR > YEAR
                                             | is.na(CADCONSTRUCTIONYEAR),
                                             CONSTRUCTIONYEAR, CADCONSTRUCTIONYEAR)) |>
  dplyr::mutate(-IMPUTED_YEAR)
```

Remove duplicate records, we only take as repeated instance an asset located in the same place with the same price

```
assets.sf <- assets.sf |>
  dplyr::group_by(PERIOD,LATITUDE, LONGITUDE, PRICE, .drop=T) |>
  dplyr::filter(row_number() == 1) |>
  dplyr::ungroup()
```

We turn our regular dataframe to a spatial features data frame, we project the source coordinates.

```
#
# Createsf and reproject
#

assets.sf <- assets.sf |>
  st_as_sf(coords = c("LONGITUDE", "LATITUDE"), crs = projcrs_src) |>
  st_transform(projcrs_dest)

#
# Save coordinates in a separate dataframe for later use
#

assets.coordinates = as_tibble(st_coordinates(assets.sf)) |>
  dplyr::mutate(IDX = row_number())

#
# Remove geometry (turn sf into ordinary a dataframe)
#
st_geometry(assets.sf) <-  NULL
```

Price obfuscation

We will modify slighlty the prices, as price are almost-continuous in a two-step process:

- Randomize price with +/- 2.5%
- Adversisers usually round up figures to multiples of 10 (rent) or 1000 euros (sale), we align prices to multiples of 1000 (sales) and 10 (rent)

Relocation of coordinates

We move the source coordinates with a radius between 30 and 60 meters:

- Take a random angle 0 - 360 degrees
- Scale the unitary radius to a value between 30 and 60 meters

The process makes sure listings are not moved our their neighborhood boundary (for this matter we use the zoning schema used on www.idealista.com).

```
#
# Set distance parameters
#

MIN = 30
MAX = 60
DIFF = MAX - MIN

#
# Set location codes for each record: join between polygon areas and listings' cooordinates
#

coordinates.sf <- assets.coordinates |>
    st_as_sf(coords = c("X", "Y"), crs = projcrs_dest) |>
    st_join(polygons.sf, left = T)

#
# Remove spatial features from the dataframe
#

st_geometry(coordinates.sf) <- NULL

#
# It should not necessary if polygons do not overlap, just in case they do we take the first location
#  the listing is located within
#

coordinates.sf <- coordinates.sf |>
  dplyr::group_by(IDX, .drop=T) |>
  dplyr::summarize(LOCATIONID = min(LOCATIONID), groups = NULL) |>
  dplyr::arrange(IDX)

#
# We assign the idealista's location code (called LOCATIONID)
#

assets.coordinates$LOCATIONID = coordinates.sf$LOCATIONID
```

We move the coordinates, bearing in mind we must keep the item in the same polygon that it was, if not the case try get new coordinates again (to a maximum of 10 fold, in practice it needed 2 iteration maximum).

```
#
# Set a maximum reprocessing iteration
# a reprocess takes care of all items that are moved out from their original idealista polygons (LOCATIONID)
#
MAX_ITERATIONS = 10

#
# Init the process
#
pending.coordinates = assets.coordinates
new.coordinates = NULL
iteration = 1

cat(paste0('Start process objective ', nrow(pending.coordinates), ' rows\n'))

while(nrow(pending.coordinates) > 0 | iteration > MAX_ITERATIONS){

    cat(paste0('\tPerforming relocation iteration ', iteration,
             ' pending rows: ',nrow(pending.coordinates),'\n' ))

  #
  # Relocate coordinates has two steps:
  #
  #    1) Get a random angle alpha
  #    2) Calculate a distance between d in [MIN, MAX]
  #    3) Set new coordinates at (X + cos(alpha) * d, Y + sin(alpha) * d)
  #
  candidate = pending.coordinates |>
      dplyr::mutate(ANGLE_RADIANS = runif(nrow(.), 0, 360) * pi / 180) |>
      dplyr::mutate(DX = cos(ANGLE_RADIANS) * (MIN + DIFF * runif(nrow(.), 0, 1)),
                    DY = sin(ANGLE_RADIANS) * (MIN + DIFF * runif(nrow(.), 0, 1))) |>
      dplyr::mutate(NEW_X = round(X + DX),
                    NEW_Y = round(Y + DY)) |>
      dplyr::select(-ANGLE_RADIANS, -DX, -DY)

  #
  # Append the new location to the data frame`
  #
  candidates.sf = candidate |>
      st_as_sf(coords = c("NEW_X", "NEW_Y"), crs = projcrs_dest) |>
      st_join(polygons.sf) |>
      dplyr::rename(LOCATIONID = LOCATIONID.x,
                    NEW_LOCATIONID = LOCATIONID.y)

  # Take locationid use the min
  candidates.sf <- candidates.sf |>
    dplyr::group_by(IDX, LOCATIONID, .drop = T) |>
    dplyr::summarize(NEW_LOCATIONID = min(NEW_LOCATIONID), .groups = NULL) |>
    dplyr::arrange(IDX)

  candidate$NEW_LOCATIONID = candidates.sf$LOCATIONID

  # Add created end redo the pending
  pending.coordinates <- candidate |>
    dplyr::filter(NEW_LOCATIONID != LOCATIONID & !is.na(LOCATIONID))
```

```
  done.coordinates  <- candidate |>
    dplyr::filter((NEW_LOCATIONID == LOCATIONID) | is.na(LOCATIONID))

  if (is.null(new.coordinates)){
    new.coordinates = done.coordinates
  } else {
    new.coordinates <- new.coordinates |>
      dplyr::bind_rows(done.coordinates)
  }
  cat(paste0('\tPerformed relocation iteration ', iteration,
             ' total rows: ',nrow(done.coordinates),'\n-------\n' ))
  iteration = iteration + 1

}
new.coordinates <- new.coordinates |>
  mutate(MOVED_DISTANCE = sqrt((NEW_X - X)^2 + (NEW_Y - Y)^2))
rm(pending.coordinates, done.coordinates, candidates.sf, candidate)
```

Assign the new coordinates to the listings.

```
#
# Save coordinates
#
assets.coordinates = as_tibble(st_coordinates(assets.sf))

#
# Remove geometry
#
st_geometry(assets.sf) <-  NULL

#
# Reasign coordinates once geom was removed
#
assets.sf <- assets.sf |>
  dplyr::mutate(LONGITUDE = assets.coordinates$X,
                LATITUDE = assets.coordinates$Y)
```

Save results

Create final dataframe and save results

```
#
# Save results
#
assets.sf |>
  fwrite(file = paste0(output_path, iconv(file_name,from="UTF-8",to="ASCII//TRANSLIT")),
         sep = ";",
         compress = "gzip")
```