Code guide & explanation

Guillermo Rodríguez-López, Ana Serrano, Ignacio Cazcarro, Miguel Martín-Retortillo

2024-10-03

CLIMATIC VARIABLES SCRIPT

Introduction

This R script performs zonal statistics to analyze climatic raster data at the municipal level. The process extracts statistical information from raster layers and aggregates the results by municipalities, exporting the data for each year. The final output is a merged dataset containing all the computed zonal statistics. The script is modular, allowing scalability and flexibility in handling large datasets of raster files across time periods.

Code Overview

1. Loading Required Libraries

The code begins by loading a set of libraries essential for handling spatial data, raster analysis, and data manipulation. The primary libraries used are:

- `exactextractr`: for efficient zonal statistics computation between raster and vector data.
- `rgdal`, `sf`, and `terra`: for handling spatial vector and raster data formats.
- `raster`: for manipulating raster data.
- `tidyverse`: for data manipulation and cleaning.
- `readxl`, `writexl`: for reading and exporting data in Excel format.

2. Data Loading

In this section, the paths to the raster files and the shapefile containing municipal boundaries are defined:

- `folder_path` points to the directory with climatic raster files.
- `municipalities_path` defines the path to the shapefile of municipal boundaries.

The script uses `list.files()` to load all raster files from the folder, filtering by a specific pattern (`asc` extension). The shapefile representing municipalities is loaded using `st_read()`.

3. Processing the Raster Data

The main processing loop iterates through the list of raster files to extract zonal statistics:

- A year counter (`year = 1900`) is initialized and increments by 10 after processing each file, reflecting the time period for each raster dataset.
- For each raster, the script reads it with the `raster()` function and then computes zonal statistics using `exact_extract()`. This function calculates summary statistics (e.g., mean, sum) for the areas represented by the shapefile.

  The results are stored as a new column (`zs`) in the attribute table of the vector data, and additional columns are added for year and other identifiers.
- The processed data is saved in Excel format using `write_xlsx()` for each time period. The script prints a message confirming the successful completion of each file.

4. Combining the Results

Once all raster files are processed and their results exported, the script aggregates the individual datasets:

- The `ls()` function identifies all objects in memory matching a specific pattern (in this case, related to the processed files).
- These objects are collected into a list with `mget()`, and then combined into a single dataframe using `bind_rows()`.

The final combined results are exported as an Excel file named `combined_result.xlsx`.

Use Case

This script is particularly useful for environmental and geographical studies that require summarizing raster-based variables (such as temperature, precipitation, etc.) by administrative units (e.g., municipalities). The output provides zonal statistics for each municipality and for each time period, facilitating further analysis of temporal trends and spatial patterns.

LAND USE VARIABLES SCRIPT

Introduction

This R script applies zonal statistics to land use raster data at the municipal level, similar to the previously detailed analysis for climatic data. The aim is to summarize land use variables (e.g., agricultural, forest, urban) across municipalities and export the results for further analysis.

Code Overview

1. Loading Data

. The land use raster files and the municipal shapefile are loaded from specified paths using `list.files()` and `st_read()`.

2. Processing the Raster Data

A loop processes each raster file, calculating zonal statistics for each municipality using `exact_extract()`. The data is summarized by year, stored in an attribute table, and exported as Excel files. Each raster represents land use data for a specific time period (e.g., every decade), with the year variable updated accordingly.

3. Combining Results

Once the zonal statistics are computed for all rasters, the individual files are merged into a single dataset using `bind_rows()`. The final dataset is then exported as an Excel file for further analysis of land use patterns over time.

The next part of the script, is oriented to update accurately the irrigation data for de most recent decadde

4. Reading the Global Irrigated Areas Raster File

In this section, the raster file containing Global Irrigated Areas (GIA) data is loaded using the `raster()` function:

- `gia_path` is the path to the raster file.
- The `cellStats()` function checks if the raster file has been reclassified correctly, ensuring that the maximum value is 1, indicating irrigation presence.

4. Calculating Irrigation Statistics

The core of the script focuses on calculating the number of irrigated cells per municipality:

- Zonal statistics are performed using the `exact_extract()` function, which calculates the sum of raster cells with a value of 1 (indicating irrigation) for each municipality.
- The result is stored as a new column (`irrigation_cells`) in the attribute table of the municipal shapefile. This column indicates how many raster cells within each municipality are irrigated.

6. Converting Irrigated Cells to Hectares

The next step is to convert the number of irrigated cells into hectares, accounting for the geographic location of each municipality:

- The script calculates pixel size in meters, adjusting for latitude and longitude using trigonometric functions (`cos()`).
- The area of each pixel (in square meters) is computed based on the degree-to-meter conversion for both latitude and longitude.
- The number of irrigated cells is then multiplied by the pixel area to estimate the total irrigated area in square meters.
- Finally, the total irrigated area is converted from square meters to hectares (`irrigation_GIA_ha`).

7. Exporting the Results

The processed data, containing the calculated irrigated hectares for each municipality, is exported to an Excel file for further analysis. Each file is generated dynamically based on the current decade being processed.

Improving the accuracy with official statistics (ESYRCE)

1. Loading Data

The script begins by loading the necessary datasets for estimated and official irrigation areas for the years 2000 and 2010. The `read_excel()` function is used to read Excel files containing the estimated irrigation area for both years and the official irrigation area from the ESYRCE data, storing them as `rest2010`, `rest2000`, `rok2002`, and `rok2010`.

2. Adjusting Official Values from 2002 to 2000

To homogenize the year, the official values for 2002 are adjusted to match the totals for 2000. The variable `tot2000` is set to represent the total official irrigated hectares in Spain for the year 2000. The script utilizes the `mutate()` function from the `dplyr` package to calculate the adjusted irrigation values for 2000, scaling the regional totals to align with this national total. This adjustment ensures that regional values are consistent, providing a solid foundation for further analysis.

3. Calculating Change Rate from 2000 to 2010

The next step is to calculate the evolution rate of irrigation values by region. The script merges the official irrigation data from 2000 and 2010 using the `left_join()` function, resulting in a new data frame `difok_0010`. The irrigation values are converted to numeric format using `as.numeric()`, and the change in irrigation values over the decade is calculated using the `mutate()` function again. This evolution rate is crucial for understanding how irrigation has changed at the regional level, allowing for comparisons against the adjusted estimated values.

4. Calculating Adjusted Irrigation Values

In this section, the script calculates the adjusted irrigation for 2010 based on the estimated values for 2000. The `left_join()` function is used to combine the data frames containing the adjusted irrigation values and the calculated change rates. The `mutate()` function is applied once more to compute the adjusted irrigation values for 2010, resulting in a new column `irrigation_ajusted2010`. This step ensures that the estimates reflect both official statistics and estimated data, improving the overall accuracy of the results.

5. Exporting the Results

The processed data, containing the calculated adjusted irrigation areas, is prepared for export. This final dataset includes the adjusted irrigation values and is ready for further analysis or reporting, typically exported using functions such as `write.csv()` or `writexl::write_xlsx()`.

Calculation of the area under cultivation and irrigated land in adjacent municipalities

1. Defining the Path and Loading the Shapefile

In this section, the path to the shapefile is defined, and the shapefile containing cultivated area data is loaded:

- `ruta` is the path to the shapefile.
- The `st_read()` function is used to load the shapefile into an `sf` object (`df`).

2. Initializing New Column for Neighboring Areas

A new column is created in the data frame to store the cultivated area of neighboring municipalities:

- `supcultivada_ha_vecinos_2010s` is initialized to hold the sum of cultivated areas from adjacent municipalities.

3. Calculating Cultivated Area of Adjacent Municipalities

The core of the script focuses on calculating the cultivated area for each municipality:

- A loop iterates through each municipality, extracting its geometry.
- The `st_touches()` function identifies neighboring municipalities that share boundaries with the current municipality.
- The current municipality's code is excluded from the neighbors list.
- The cultivated area of the neighboring municipalities is summed, ignoring any `NA` values.
- The total cultivated area of neighbors is stored in the corresponding row of the new column.

4. Exporting the Results

The processed data, which includes the calculated cultivated area for each municipality, is exported to a CSV file for further analysis:

- The `write.csv()` function saves the updated data frame to a CSV file named `supcultivada_vecinos2010s.csv`.

GEOGRAPHICAL VARIABLES SCRIPT

Altitude and Ruggedness Calculation for Spanish Municipalities

This R script aims to calculate the average altitude and ruggedness (standard deviation of altitude) for each municipality in Spain using a digital elevation model (DEM) and a shapefile of the municipalities. The `exactextract` package is utilized for zonal statistics extraction.

1. Data Reading

In this section, the necessary data is read into R:

- **Path to Raster**: The variable `ruta` specifies the path where the digital elevation model (DEM) raster file is located.
- **Read Vector Base**: The `st_read()` function is used to load the shapefile containing the municipalities (in the `Municipios_IGN_ETRS89.shp` file) into a spatial object `v`. This shapefile serves as the vector layer for the analysis.
- **Read Raster**: The `raster()` function reads the DEM raster file into the object `r`, enabling raster operations.

2. Zonal Statistics

This section calculates the statistics of altitude for each municipality:

- **Mean Altitude**: The `exact_extract()` function calculates the mean altitude for each municipality by summing the altitude values within each municipality's boundaries. The results are stored in a new column `zs_mean` in the spatial object `v`.
- **Ruggedness**: The same `exact_extract()` function is used again to compute the standard deviation of altitude values within each municipality, which serves as a measure of ruggedness. This value is stored in the `zs_sd` column of the object `v`.

3. Attribute Table Conversion

- The attribute table of the spatial object `v` is converted into a tibble format (a type of data frame from the `tidyverse`) using `as_tibble()`. This conversion facilitates easier manipulation and analysis of the data.

4. Exporting Results

- Finally, the attribute table, which now contains the average altitude and ruggedness values for each municipality, is exported to an Excel file named `altitude_ruggedness` using the `write_xlsx()` function. This allows for further analysis or reporting of the results in other applications.

Distance variables

This methodology outlines the detailed steps taken to compute geographical variables, which focus on distance, location, and relief characteristics of Spanish municipalities. The key geographical variables included in this analysis are the distances to Madrid (the capital of Spain), the provincial capital, and the coast, as well as the latitude (y) and longitude (x) coordinates of each municipality.

1. Extraction of Municipal Centroids:

The first step involves obtaining the centroids of each municipality using the shapefile provided by the National Geographic Institute of Spain. To achieve this, the QGIS algorithm "Generate points (pixel centroids) inside polygons" was utilized. In R, this can be implemented using the `st_centroid()` function from the `sf` package. This function calculates the geometric centroid of polygon features, ensuring that we accurately represent each municipality's central point. The centroids serve as the basis for all subsequent distance calculations.

2. Calculating X and Y Coordinates:

After extracting the centroids, the next step is to calculate their X and Y coordinates. This was accomplished using the `st_coordinates()` function from the `sf` package, which extracts coordinate information directly from the spatial objects. The output is a data frame containing the coordinates of each centroid, which is essential for distance calculations.

3. Distance Calculation to Madrid and Provincial Capital:

To calculate the distances to Madrid and the provincial capital, separate layers were created for each capital municipality. This was achieved using the `filter()` function from the `dplyr` package, which allows for attribute selection based on specific criteria. For instance, the relevant municipalities for Madrid and the provincial capital were isolated from the shapefile attribute table loaded as an `sf` object.

Once the capital centroids were identified, the distance from all municipality centroids to each capital centroid was computed. The `qgis:distancetonearesthub` algorithm derived from the function `qgis_run_algorithm` from the `qgisprocess` package facilitated this calculation by measuring distances between the geometric points representing the centroids.

Furthermore, to ensure that the distance calculated for the provincial capital matched the corresponding municipality's province, an additional filtering step was implemented using the `filter()` function again. This ensures the integrity of the distance data, aligning it correctly with the relevant municipality.

4. Distance to the Coast Calculation:

The methodology for calculating the distance from the centroids of municipalities to the coastline was slightly different. In this case, the "Distance to nearest hub (line to hub)" algorithm from the `qgisprocess` package was employed. This algorithm efficiently calculates the nearest distance from a point (the municipality centroid) to a line feature (the coastline). By utilizing the `qgis_run_algorithm()` function from the `qgisprocess` package, we can implement this analysis seamlessly within the R environment. The output is a dataset containing the calculated distances from each municipality to the coastline.

5. Final Variables Compilation:

After computing all relevant distances, the final step involves compiling the results into a cohesive data frame. This data frame includes the municipal centroids, their respective X and Y coordinates, and the computed distances to Madrid, the provincial capital, and the coast. By consolidating these geographical variables, we create a comprehensive dataset that serves as a valuable foundation for further spatial analysis and modeling.

In summary, this methodology effectively integrates spatial data handling and distance calculations through the utilization of R packages such as `sf`, `dplyr`, and `qgisprocess`. By meticulously following these steps, we establish a robust framework for analyzing geographical variables at the municipal level, which is essential for understanding spatial dynamics and their implications in various contexts.

HYDROLOGICAL VARIABLES SCRIPT

Reservoir variables

Introduction

This document explains the process of matching dam datasets, assigning construction years, and calculating the area and volume of dams by municipality using R. The analysis involves a combination of data wrangling and spatial data operations using various packages.

1. Loading Libraries

To begin, we load the necessary packages for data manipulation and spatial operations:

- `dplyr`: Used for data wrangling, filtering, summarizing, and creating new variables.
- `readxl` and `readr`: For reading Excel and CSV files.
- `stringdist` and `stringr`: Used to manipulate and match dam names between datasets.
- `janitor`: Helps clean and format data for easier analysis.
- `rgdal`: Provides functions for reading and manipulating spatial data (shapefiles).
- `tidyr`: Assists in reshaping data.
- `openxlsx`: For writing data to Excel files.

2. Filtering and Matching Datasets

We load dam datasets using functions from `readxl`. We filter the datasets to remove rows that do not contain useful information, such as empty dam names, using the `filter()` function from `dplyr`.

We then use `mutate()` and `match()` to match dam names between two datasets and assign construction years. Dams without a match in the first dataset are supplemented using a second manually matched dataset. The function `coalesce()` from `dplyr` is used to handle multiple potential sources for construction years.

3. Handling Missing Matches

We address missing matches by importing a secondary dataset. The `coalesce()` function combines the primary and secondary datasets, ensuring that the best available construction year is assigned to each dam.

4. Spatial Data Integration

The `readOGR()` function from `rgdal` is used to import a shapefile containing geographical data of dams. To standardize dam names, string manipulation functions such as `gsub()` and `toupper()` are applied.

We use `mutate()` and `match()` to assign construction year data to the corresponding dams in the shapefile.

5. Calculating Proportional Area and Volume

We load data on the proportion of dam area and volume by municipality. Using `mutate()` from `dplyr`, we calculate these proportions and then summarize the data by municipality using `group_by()` and `summarise()`.

The `across()` function allows us to apply summary functions to multiple columns, simplifying the aggregation of area and volume data for each municipality.

6. Calculating Accumulated and Non-Accumulated Data by Decade

Two types of summaries are created: - **Accumulated data**: This includes all dams constructed up to the end of each decade. We use `filter()` to subset the data by decade and then `group_by()` and `summarise()` to generate the summaries. - **Non-Accumulated data**: This considers only the dams constructed within a specific decade. The same filtering and summarizing functions are applied to isolate the data for each time period.

7. Exporting Results

Finally, the processed data is exported to Excel format using the `write.xlsx()` function from the `openxlsx` package. This allows for further analysis or presentation of the results outside of R.

Water courses variables

This section explains the main steps involved in calculating hydrological variables for municipalities, particularly focusing on their proximity to watercourses and hydrographic basins. The methodology leverages geospatial operations using R with the `sf` and `qgisprocess` packages.

Data Preparation

- **Municipal centroids**: These represent the geographical center of each municipality. They are essential for spatial operations where distance to various water features is calculated.
- **Hydrological shapefiles**: Two key shapefiles are used in the process. The first one contains the complete water network, while the second one contains the main rivers. Both shapefiles are loaded using the `sf::st_read()` function and are initially in polyline format.

1. Converting Polylines to Points

Using the QGIS tool **Points along geometry**, the watercourses (polylines) from both shapefiles are converted into points. This step ensures that we have reference points along each river or watercourse for distance calculations. This operation is performed using the `qgisprocess` package, which interfaces with QGIS to carry out the conversion process.

2. Spatial Join: Assigning Basins

To calculate the basin variable, a **spatial join** operation is used. The shapefile of the hydrographic basins is joined with the municipalities using the `sf::st_join()` function. This assigns each municipality the value of the basin it falls into. If a municipality spans two basins, the basin covering the largest area is selected using geospatial area comparison techniques (`sf::st_area()`).

3. Distance to Water Networks

The `qgisprocess::qgis_run_algorithm()` function is used to calculate the distances from each municipality centroid to the nearest point along the river networks (both the complete network and the main rivers). This is done with the **Distance to nearest hub (points)** QGIS algorithm. The result provides the nearest river (or main river) and its distance for each municipality centroid.

4. Output Variables

After the calculations, the following key variables are generated: - **Basin**: The hydrographic basin in which each municipality is located. - **nearest_all_river**: The ID of the nearest watercourse from the complete water network. - **Dist_all_rivers**: The distance from the municipality centroid to the nearest watercourse in the complete network. - **nearest_main_river**: The ID of the nearest main river. - **Dist_main_rivers**: The distance to the nearest main river.

SOCIOECONOMIC VARIABLES SCRIPT

Introduction

This document provides a comprehensive overview of the methodology used to calculate socio-economic variables related to the population of municipalities. The focus is on classifying municipalities based on population size, calculating distances to those with significant populations, and assigning Simpson area classifications.

1. Load Necessary Libraries

The analysis begins with loading the required R libraries to perform data manipulation, spatial analysis, and GIS operations. The following packages are utilized:

- **dplyr**: For data manipulation and transformation.
- **sf**: For handling spatial data.
- **qgisprocess**: For executing QGIS algorithms within R.

2. Import Population Data

The total population data is imported using the appropriate R functions. This data, sourced from Goerlich's municipal population series, contains the population counts for each municipality without modifications.

3. Classify Municipalities

Municipalities are classified into three categories based on their population size using the `case_when()` function from the `dplyr` package. The classification criteria are as follows:

- **Rural**: Municipalities with less than 2000 inhabitants.
- **Intermediate**: Municipalities with a population between 2000 and 10000.
- **Urban**: Municipalities with more than 10000 inhabitants.

The resulting variable, `urban_rural`, captures this classification, facilitating further analysis.

4. Calculate Distances to Significant Populations

4.1 Define Population Thresholds To calculate distances to municipalities with significant populations, two thresholds are defined: **5000** and **10000** inhabitants. The variable `p` represents these thresholds.

4.2 Create Municipal Subsets For each decade of data, the centroids of municipalities are utilized to generate two subsets:

- **Municipalities below p**: Contains municipalities with populations less than `p`.
- **Municipalities above p**: Contains municipalities with populations greater than or equal to `p`.

This is achieved through the `filter()` function from the `dplyr` package, allowing for efficient selection based on population criteria.

4.3 Calculate Distances Using QGIS Tools For each subset created, the QGIS tool **Nearest Hub** from the `qgisprocess` package is employed to calculate distances:

- **nearest_municipality_below_p**: Distance from municipalities below `p` to the nearest municipality above `p`.
- **distance_pop_5000_** and **distance_pop_10000_**: Variables capturing these distances.

The use of the QGIS tool allows for accurate spatial calculations, ensuring that the distances are computed based on the geometric locations of the municipalities.

5. Assign Simpson Area Classifications

The Simpson area classifications are established according to Simpson (1995). This classification groups provinces into regions based on socio-economic and geographical characteristics.

5.1 Define Area Classifications Two classifications are defined based on the degree of aggregation:

- **Simpson_Areas_4**: Corresponds to broader regional classifications.
- **Simpson_Areas_10**: Corresponds to more detailed classifications.

5.2 Allocate Simpson Areas to Municipalities

Using a conditional statement, each municipality is assigned to its corresponding Simpson area based on the province it belongs to. This is facilitated through the use of `ifelse()` or similar conditional functions to ensure that each municipality receives the correct classification.

6. Compile the Dataset

After calculating all the required variables, the final dataset is compiled. This includes:

- Total population counts.
- Urban-rural classification.
- Distances to municipalities with populations over the specified thresholds.
- Corresponding Simpson area classifications.

This dataset is then saved for further analysis or reporting.

Transport Distance Variables

Overview

This script calculates the minimum distance from municipalities to various transport infrastructures, including airports, high-speed rail stations, standard railway stations, and narrow-gauge railway lines. The distances are computed for each decade from 1900 to 2021.

Requirements

To run this script, ensure you have the following R packages installed:

```
install.packages(c("tidyverse", "sf", "units", "writexl", "readxl"))
```

Usage

1. Place the required shapefiles in the appropriate directories:
   - `./data/Municipalities.shp` (municipality boundaries)
   - `./GIS/Aeropuertos/Aeropuertos.shp` (airport locations)
   - `./GIS/estaciones/HSR/HighSpeed_Stations.shp` (high-speed rail stations)
   - `./GIS/estaciones/IberianGauge/IberianGauge_Stations.shp` (standard railway stations)
   - `./GIS/estaciones/NarrowGauge/NarrowGauge_Lines.shp` (narrow-gauge railway lines)
2. Run the script in an R environment. It will:
   - Transform shapefiles to a common coordinate system (EPSG:25830 - UTM for Spain).
   - Compute the distance from each municipality's centroid to the nearest transport infrastructure.
   - Store the results in individual data frames for each decade.
   - Merge the data and save the results as Excel files:
     - `distancia_aeropuertos_year.xlsx`
     - `distancia_ave.xlsx`
     - `distancia_estaciones.xlsx`

  - `distancia_feve.xlsx`
  - `distancia_ferrocarril_year.xlsx`

3. The final dataset, `distancia_ferrocarril_year.xlsx`, contains the integrated transport distances for all rail networks.

Output

The script generates a dataset containing: - `CODIGOINE`: Municipality code - `Year`: Year of analysis - `airport_distance`: Distance to the nearest airport (km) - `highspeedstation_distance`: Distance to the nearest high-speed rail station (km) - `nearest_highspeed_name`: Name of the nearest high-speed rail station - `railwaystation_distance`: Distance to the nearest standard railway station (km) - `nearest_station_name`: Name of the nearest standard railway station - `narrowrail_line_distance`: Distance to the nearest narrow-gauge railway line (km) - `narrowrail_line_name`: Name of the nearest narrow-gauge railway line

Notes

- If no transport infrastructure is available for a given decade, the distance is set to `Inf`.
- For narrow-gauge railways, missing closure years are assumed to be 3000 (indicating they are still active).

Yearly Database

Overview

This script processes historical economic and land use data by annualizing static variables, calculating cultivated and irrigated land for neighboring municipalities, integrating climate and infrastructure data, and generating a final dataset.

Requirements

Ensure the following R packages are installed:

```
install.packages(c("tidyverse", "sf", "units", "writexl", "readxl"))
```

Steps

1. Load Data

- **Municipal Boundaries**: `./data/Municipalities.shp`
- **Irrigated Land**: `./BD_anual/results/Irrigated_year.xlsx`
- **Dryland**: `./BD_anual/results/Dryland_year.xlsx`
- **Historical Economic Data**: `Historeco+.csv`
- **Reservoir Data**: `./BD_anual/data/anual_embalses.xlsx`
- **Infrastructure Data**:
  - Airports: `distancia_aeropuertos_year.xlsx`
  - Railways: `distancia_ferrocarril_year.xlsx`

2. Process Data

- **Annualizing Static Variables**: Expands static variables up to 2021.
- **Calculate Neighboring Land Areas**:
  - Sum irrigated and dryland areas.
  - Use spatial neighbors to compute neighboring cultivated areas.
- **Merge Climate & Land Use Data**:
  - Load climate data (`Spei`, `Precipitation`, etc.).
  - Merge with land use data.
  - Integrate infrastructure distances.
- **Final Processing**:
  - Rename key variables (`spei`, `pp`, `t_average`, etc.).
  - Export final dataset as CSV.

3. Outputs

- **Complete dataset**: `./BD_anual/Historeco_Year.csv`
- **2021 subset**: `bd2021.xlsx`

Notes

- Missing values for closed infrastructures are set to `Inf`.
- If data is missing for certain years, nearest available data is used.

Author

This script was developed for economic geography and land use analysis.