Policy Research Working Paper 11115

Building and Managing Local Databases from Google Earth Engine with the geeLite R Package

Marcell T. Kurbucz
Bo Pieter Johannes Andrée

WORLD BANK GROUP
Development Economics
Development Research Group
May 2025

Policy Research Working Paper 11115

Abstract

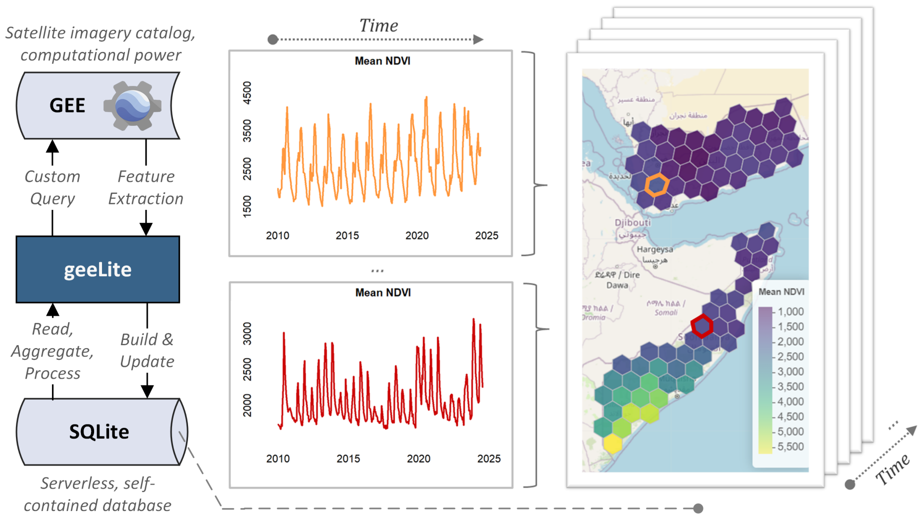
Google Earth Engine has transformed geospatial analysis by providing access to petabytes of satellite imagery and geospatial data, coupled with the substantial computational power required for in-depth analysis. This accessibility empowers scientists, researchers, and non-experts alike to address critical global challenges on an unprecedented scale. In recent years, numerous R packages have emerged to leverage Google Earth Engine's functionalities. However, constructing and managing complex spatio-temporal databases for monitoring changes in remotely sensed data remains a challenging task that often necessitates advanced coding skills. To bridge this gap, geeLite, a novel R package, is introduced to facilitate the construction, management, and updating of local databases for Google Earth Engine-computed geospatial features, which enables users to monitor their evolution over time. By storing geospatial features in SQLite format—a serverless and self-contained database solution requiring no additional setup or administration—geeLite simplifies the data collection process. Furthermore, it streamlines the conversion of stored data into native R formats and provides functions for aggregating and processing created databases to meet specific user needs.

This paper is a product of the Development Research Group, Development Economics. It is part of a larger effort by the World Bank to provide open access to its research and make a contribution to development policy discussions around the world. Policy Research Working Papers are also posted on the Web at http://www.worldbank.org/prwp. The authors may be contacted at bandree@worldbank.org and mkurbucz@worldbank.org.

Graphical Abstract

Building and Managing Local Databases from Google Earth Engine with the `geeLite` R Package

Marcell T. Kurbucz, Bo Pieter Johannes Andrée



Highlights

Building and Managing Local Databases from Google Earth Engine with the `geeLite` R Package

Marcell T. Kurbucz, Bo Pieter Johannes Andrée

- Google Earth Engine offers vast satellite imagery and computational power.
- `geeLite` is an R package designed to leverage the power of Google Earth Engine.
- It enables creating, updating, and managing custom databases for real-time tracking.
- It stores data in SQLite format, enhancing both accessibility and portability.
- It streamlines the reading, aggregation, and processing of created databases.

Building and Managing Local Databases from Google Earth Engine with the `geeLite` R Package

Marcell T. Kurbucz[a,b,1,*], Bo Pieter Johannes Andrée[b,1]

[a] *Institute for Global Prosperity, The Bartlett, University College London, 149 Tottenham Court Road, London, W1T 7NF, United Kingdom*
[b] *Development Economics Data Group, World Bank, 1818 H Street NW, Washington, D.C., 20433, USA*

Keywords: Google Earth Engine, Geographic Information System, Remote Sensing, Raster Data, Spatio-Temporal Data, Software

JEL Codes: C21, C63, C81, C88, L17

---

[*] Corresponding author.
Email addresses: `mkurbucz@worldbank.org` (Marcell T. Kurbucz), `bandree@worldbank.org` (Bo Pieter Johannes Andrée)

[1] These authors contributed equally to this work. Funding by the World Bank's Food Systems 2030 (FS2030) Multi-Donor Trust Fund program (TF0C0728 and TF0C7822) is gratefully acknowledged. We thank Andres Chamorro and Ben P. Stewart for code testing and comments, as well as Steve Penson, David Newhouse and Alia J. Aghjanian for helpful comments and input. This paper reflects the views of the authors and does not reflect the official views of the World Bank, its Executive Directors, or the countries they represent.

1. Introduction

The ever-growing volume of Earth observation data presents both opportunities and challenges for scientific inquiry. While vast datasets hold the potential to revolutionize our understanding of Earth systems, traditional desktop-based analysis methods often struggle with the computational burden associated with such data (Amani et al., 2020). Google Earth Engine (GEE) has emerged as a powerful solution, offering a cloud-based platform for efficient management, analysis, and visualization of geospatial big data (Gorelick et al., 2017). Its core strength lies in its ability to overcome the limitations of traditional approaches by providing (Tamiminia et al., 2020):

- **Petabytes of public geospatial data:** A comprehensive data catalog readily accessible through a web interface, including historical and current satellite imagery, environmental variables, and other geospatial information.
- **High-performance parallel processing:** Leveraging Google's cloud infrastructure for large-scale analysis, enabling researchers to tackle complex problems that would be infeasible on personal computers.
- **Accessible development environment:** A web-based interface and application programming interfaces (APIs) in JavaScript and Python, supporting widely-used programming languages.

This combination of features allows scientists to investigate new scientific questions in a wide range of subjects, particularly those requiring large-scale spatial and temporal analysis.[2] Examples include studies on climate change (Banerjee et al., 2024; Kazemi Garajeh et al., 2024), environmental degradation (Andrée et al., 2019), land cover change (Wang et al., 2020; Burgueño et al., 2023), deforestation (Chen et al., 2021; Brovelli et al., 2020), flood monitoring (Hamidi et al., 2023), wildfire prediction (Tavakkoli Piralilou et al., 2022), urbanization (Marconcini et al., 2021; Zheng et al., 2021), food security (Andrée et al., 2020; Wang et al., 2022; Penson et al., 2024), and sustainable development (Burke et al., 2021), to name a few.

Recognizing the broader potential of GEE within the R user community, Aybar et al. (2020) developed the `rgee` R package. Acting as a bridge, `rgee` seamlessly integrates GEE with R's extensive ecosystem of geospatial

[2] For a comprehensive review of current and potential future GEE research trends, refer to Velastegui-Montoya et al. (2023).

packages, making GEE's capabilities accessible to a wider user base. This integration is facilitated by the `reticulate` package (Ushey et al., 2024), allowing `rgee` to interact with GEE's official Python API and utilize all its modules, classes, and functions.

In recent years, numerous R packages have been published to extend the functionality of `rgee` and improve its user experience. Among these, `tidyrgee` (Arno and Erickson, 2022) aids users in filtering, joining, and summarizing GEE image collections, `rgee2` (Kong, 2022) expands upon the original package with additional functions, and `rgeeExtra` (Aybar et al., 2023) simplifies its syntax. Other packages, like `SAEplus` (Team, 2022) and `LandsatTS` (Berner et al., 2023), facilitate the data extraction and processing from GEE or provide geospatial decision-making tools, such as `RePlant alpha` (Morales et al., 2023).

While available third-party libraries improve the accessibility of GEE's functionalities, constructing and managing up-to-date temporal geospatial databases remains a complex time-consuming task that requires advanced coding knowledge. This poses a significant barrier for users interested in monitoring applications but lacking such expertise. To address this gap, our paper introduces `geeLite`, a novel R package that enables users to easily build, maintain, and keep up-to-date local temporal databases of GEE-computed geospatial features. `geeLite` offers a flexible yet user-friendly data collection procedure and stores the gathered features in SQLite format—a serverless, self-contained solution that eliminates the need for additional setup or administration.[3] Furthermore, it streamlines the conversion of stored data into native R formats and provides functions for aggregating and processing created databases to meet specific user needs. The metadata for the new package is detailed in Table 1.

[3]More information can be found at: `https://sqlite.org/features.html` (retrieved: February 2, 2025).

Table 1: Metadata of the geeLite package

Metadata Description	Metadata Contents
Current code version:	0.1.0
Permanent link:	https://github.com/mtkurbucz/geeLite
Legal code license:	Mozilla Public License Version 2.0
Code versioning system:	Git
Software code languages:	R
System requirements:	OS agnostic (Linux, macOS, MS Windows). R 4.3.2 or later.
Required R packages:	cli, crayon, data.table, dplyr, <u>geojsonio</u>, googledrive, h3jsr, jsonlite, knitr, lubridate, magrittr, <u>optparse</u>, progress, purrr, reshape2, reticulate, rgee, RSQLite, sf, stats, stringr, tidyr, tidyrgee, utils, rnaturalearth, <u>rnaturalearthdata</u>.
Suggested R packages:	leaflet, rmarkdown, testthat, withr.
External dependencies:	A virtual Conda environment with the following Python packages: earthengine-api, ee_extra, and numpy. The current version of the geeLite package (0.1.0) specifically requires version 0.1.370 of earthengine-api.
User manual:	https://github.com/mtkurbucz/geeLite/blob/master/README.md
Support email:	mkurbucz@worldbank.org

Note: The underlined R packages are not directly used by the geeLite package; however, they are essential for the database construction process. During the setup of geeLite via its gee_install function, the availability of these packages is checked, and they are automatically installed if not already available.

The rest of this paper is organized as follows. Section 2 outlines the installation steps, workflow, structure, and unit testing of the geeLite package. Section 3 presents an example of the package's application. Finally, Section 4 discusses the impact of the software and provides conclusions.

2. Software Description

2.1. Installation

The geeLite package can be installed from GitHub (Kurbucz and Andrée, 2024) using the devtools R package (Wickham et al., 2022). To interact with GEE, geeLite utilizes the rgee R package, which requires a virtual Conda environment containing the earthengine-api, ee_extra, and numpy Python packages. After installing geeLite, users need to set up this environment using the gee_install function as follows:

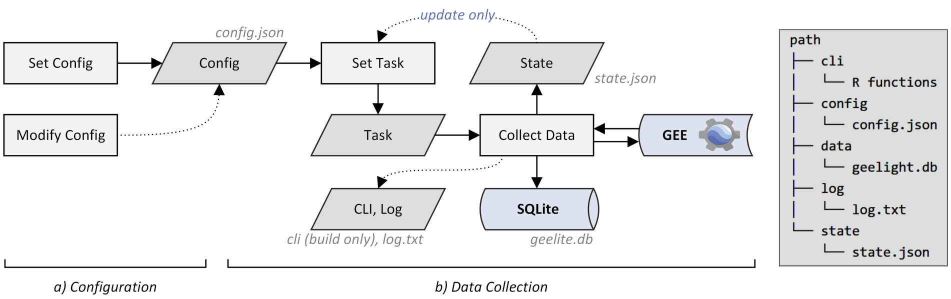
```
# Install geeLite package:                                    [R code]
# install.packages("devtools")
devtools::install_github("mtkurbucz/geeLite")
geeLite::gee_install()
```

Note that the current version of the geeLite package (0.1.0) is specifically compatible with earthengine-api version 0.1.370. The gee_install function ensures the installation of the correct version of this package and also installs the geojsonio and rnaturalearthdata R packages if they are missing.

2.2. Workflow

The workflow of the geeLite package consists of two primary steps. First, the configuration step (a) allows users to create and modify a configuration file that specifies the dimensions, variables, and spatial aggregation methods for the desired database. Subsequently, this file guides the second step, data collection (b), which begins with user authentication and then efficiently builds a new database or updates an existing one according to the user's specifications. Additionally, the package provides basic tools for managing the resulting database within an R session (c) and offers utilities to streamline the automation of database updates (d). The workflow and folder structure of the generated database are illustrated in Figure 1 and detailed in the following points.

Figure 1: Workflow of and the folder structure of the generated database



Note: The geeLite package operates through two primary steps: (a) configuration, where users define the parameters for the database to be generated, and (b) data collection, which begins with user authentication. Upon successful authentication, the package collects and preprocesses data from GEE, tailored to the user's specifications, to build or update a custom SQLite database.

a) <u>Configuration</u>

A configuration file, created using the set_config function, is stored locally in JSON format. This file allows users to define regions of interest (regions) for zonal statistics calculations at both the administrative level 0 (countries) and administrative level 1 (subnational states). These regions are identified using ISO 3166-2 codes, which consist of two letters for countries and additional characters for states.[4] Users can select multiple data sources (`source`) by specifying one or more GEE datasets, along with their bands of interest and the statistical indicators for spatial aggregation.

The configuration file also includes settings for the resolution of the H3 grid system (`resol`),[5] the start date for data collection (`start`), and a scaling parameter to adjust image quality before processing (`scale`). Additionally, to avoid interruptions caused by exceeding GEE's limit on concurrent zonal statistics calculations, users can set a maximum number of calculations to be performed simultaneously in the configuration file (`limit`), with the default limit set to 10,000.

To simplify the configuration process for data collection, the `get_state` function allows users to view the complete list of available region codes, while the `get_config` function displays the current contents of the configuration file. Users can also modify the selected regions (`regions`), data sources (`source`), and the limit on concurrent zonal statistics calculations (`limit`) either manually (outside the R session) or using the `modify_config` function. However, modifying other configuration parameters requires rebuilding the database based on a new configuration file.

b) Data Collection (with Authentication)

Data collection involves user authentication, as well as the building and updating of the database using the `run_geelite` function.

Authentication

Building on the authentication protocol of the `rgee` package, the `run_geelite` function initiates the authentication process by directing users to a web browser, where they are prompted to grant permission to link their Google accounts with GEE. Users are then instructed to copy the resulting token into the provided input field. Upon successful authentication, a directory is created under the path `~/.config/earthengine/`, named according to the specified username. This directory securely stores all credentials associated

[4] Shapefiles for the selected regions are retrieved using the `rnaturalearth` package (Massicotte and South, 2024).

[5] `geeLite` applies Uber's H3 grid system (Uber, 2021) using the `h3jsr` package (O'Brien, 2023) to divide user-defined regions into hexagonal bins, from which zonal statistics are calculated.

with the user's Google account. If no username is provided, the credentials are stored directly in the ~/.config/earthengine/ folder. As long as the credentials remain valid, `geeLite` will automatically use them to initialize the selected Google account.

In addition to the standard authentication method, users have the option to authenticate using Google Cloud service account credentials. This method allows them to create a service account via the Google Cloud Console and download a corresponding JSON key file. By configuring the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to reference the downloaded JSON key file, users can bypass the interactive, browser-based authentication process.[6] Once the environment variable is set, `geeLite` will automatically use the service account credentials for authentication. This method is particularly useful for automated workflows and production environments where graphical interfaces may not be available.

Database Building and Updating

After user authentication, the database-building operation gathers GEE-computed geospatial features based on the configuration file and stores them in SQLite format (*geelite.db*).[7] Depending on the value of the `mode` parameter, the extraction process operates in either `"local"` mode—processing data in smaller chunks directly within R—or `"drive"` mode, which leverages Google Drive for exporting larger datasets. The resulting database contains information about H3 bins (referred to as `grid`) and collected datasets, organized into separate tables named according to their associated GEE datasets. The `grid` table includes an Open Geospatial Consortium (OGC) feature identifier (`ogc_fid`), an H3 index (`id`), a corresponding region code (`iso`), and a geometry shape (`geometry`). Other tables, containing zonal statistics, are structured in a wide format. These tables include the H3 index (`id`) of the associated bins, the band name (`band`), the statistical indicator employed (`zonal_stat`), and the zonal statistics computed on specific dates (formatted as `YYYY_MM_DD`) when the dataset was updated.

In addition to the configuration file and SQLite database, the package generates two supplementary files: *state.json* and *log.txt*. These files respec-

[6] To set this environment variable, use the following R code: `Sys.setenv(GOOGLE_APPLICATION_CREDENTIALS = "path/to/service-account-key.json")`.

[7] SQLite objects are managed by the `RSQLite` package (Müller et al., 2024).

tively record the current state of the database and log session types (build or update) along with their timestamps. To facilitate dataset management and scheduled updates, the `geeLite` package organizes a `cli` folder alongside the database. This folder contains an R script for each main function of the package—except for `set_cli` and `read_db`—allowing them to be executed from the Command Line Interface (CLI). To run these functions via the CLI, use the following structure: `Rscript [cli/function] --parameter [parameter]`. These scripts automatically use the path to the root directory of the generated database as an input parameter, eliminating the need for manual definition.[8]

Once the database is built, it can be updated (`rebuild = FALSE`) or rebuilt (`rebuild = TRUE`). For updates, the process begins with a comparison of the configuration and state files. If the configuration file has been modified since the last data collection procedure, the database will be adjusted and updated accordingly. A successful update overwrites the state file and modifies the log file. In the case of rebuilding, the data collection is based solely on the configuration file, and it overwrites the existing database and its supplementary files.

> Note that while some parameters of the configuration file (`regions`, `source`, and `limit`) can be modified manually, the **state file is vulnerable to any manual changes**. Manual modification or deletion of this file can corrupt future updates of the database and necessitate a rebuild.

c) Data management in R

The `geeLite` package provides two main functions to simplify the analysis of the generated SQLite database: `fetch_vars` and `read_db`. The `fetch_vars` function allows users to retrieve detailed information about the available variables in the database. This information includes variable names—comprising the database, band, and statistical indicator names separated by slashes—along with variable IDs, source details, start and end dates, and average frequencies in days. The `read_db` function is designed to read, aggregate, and process data from the database according to user-defined parameters.

When executing `read_db`, the selected variables are first converted to

[8]For more detailed information on using these CLI scripts, please refer to Section 3.1 or visit the package's GitHub repository (Kurbucz and Andrée, 2024).

a daily frequency using a preprocessing function (`prep_fun`)—such as the default linear interpolation. The data is then aggregated at a specified frequency (`freq`) using one or more aggregation functions (`aggr_funs`); by default, monthly mean values are calculated. Additionally, `read_db` provides the option to apply further transformations through post-processing functions (`postp_funs`), which can be especially useful for feature extraction in machine learning applications.

To enhance flexibility, `geeLite` allows users to fully customize both the aggregation and post-processing functions. This can be achieved not only by adjusting default settings, but also by applying specific functions to different variables. Furthermore, `geeLite` includes an additional main function, `init_postp`, which provides the option to define post-processing functions externally, in separate files, thereby improving the transparency of the transformation process. This function creates a *postp* folder in the root directory of the generated database, containing an editable R script (*functions.R*) where users can define custom post-processing functions, along with a JSON file (*structure.json*) that specifies which functions to apply to which variables during post-processing.

After the database is imported, R's full range of capabilities for processing, modeling, and plotting the data can be utilized.

d) <u>Automation</u>

The `geeLite` package allows users to execute its functions via the command-line interface (CLI). Each database created with `geeLite` includes a *cli* folder containing CLI-compatible versions of all functions, except for `set_cli` and `read_db`. Alternatively, these functions can be generated directly using the `set_cli` function. This feature is designed to support automatic database updates through job-scheduling utilities, such as Linux Crontab (see, e.g., Bradley, 2016).

2.3. Structure

The eleven main functions of the `geeLite` package, along with their parameters, are detailed in Table 2.

Table 2: Description of main functions and parameters

Functions

Name	Parameters	File	Description
`gee_install`	1	*gee_install.R*	Creates a Conda environment and installs required dependencies.
`set_config`	2-9	*set_config.R*	Creates the configuration file.
`run_geelite`	1-2, 9-12	*run_geelite.R*	Collects and stores data, and updates the state and log files.
`modify_config`	2, 9, 13, 14	*modify_config.R*	Modifies the configuration file.
`set_cli`	2, 9	*set_cli.R*	Creates scripts to make main functions callable through the CLI.
`get_config`	2	*get_json.R*	Prints the configuration file.
`get_state`	2	*get_json.R*	Prints the state file.
`fetch_regions`	15	*fetch_regions.R*	Displays ISO 3166-2 region codes.
`fetch_vars`	2, 16	*access_db.R*	Displays information on the available variables in the SQLite database.
`read_db`	2, 17-21	*access_db.R*	Reads, aggregates, and processes data from the SQLite database.
`init_postp`	2, 9	*access_db.R*	Initializes the file structure for external post-processing.

Parameters

ID	Name	Type	Description
1	`conda`	*character*	Name of the virtual Conda environment.
2	`path`	*character*	Path to the root directory for the generated database.
3	`regions`	*character*	ISO 3166-2 codes of regions of interest, with two letters for countries and extra characters for subdivisions like states.
4	`source`	*list*	Detailed description of the GEE datasets of interest. This is a nested list with three levels: `names`, `bands`, and `zonal_stats`.[a]
4.1	`names`	*list*	Names of the GEE datasets to be used (e.g., `"MODIS/061/MOD13A1"`).
4.1.1	`bands`	*list*	Specific data bands from the datasets (e.g., `"NDVI"`).
4.1.1.1	`zonal_stats`	*character*	Type of spatial statistics to be computed for each region (options: `"mean"`, `"sum"`, `"median"`, `"min"`, `"max"`, `"sd"`).
5	`resol`	*integer*	Spatial resolution of the H3 grid system.[b]
6	`scale`	*integer*	The nominal resolution (in meters) for processing the image projection. If left as `NULL` (the default), a resolution of 1000 is used.[c]
7	`start`	*date*	Starting date for data collection (default: `"2020-01-01"`).
8	`limit`	*integer*	Controls batch size for concurrent zonal statistics calculations. In `local` mode, it is ⌊`limit`/number of dates⌋. In `drive` mode, it sets the max H3 bins per export. Default: `10000`.
9	`verbose`	*logical*	Displays messages (default: `TRUE`).
10	`rebuild`	*logical*	If `TRUE`, overwrites the database and associated files based on the configuration file (default: `FALSE`).
11	`user`	*character*	Generates a folder in ∼`/.config/earthengine/` to store credentials for a specific Google identity. Default is the root directory: `NULL`.
12	`mode`	*character*	Specifies the mode of data extraction. Acceptable values are `"local"` and `"drive"`. In `"local"` mode, data is extracted in smaller chunks directly within R, which is suitable for modest data volumes. In `"drive"` mode, data is exported via Google Drive, enabling the handling of larger datasets that require parallel processing or higher export limits. Defaults to `"local"`.
13	`keys`	*list*	Paths to the values designated for replacement.[d]
14	`new_values`	*list*	New values to replace the original entries at the specified paths.[d]
15	`admin_lvl`	*integer*	Specifies the administrative level: `0` for country, `1` for state, or `NULL` for all regions (default: `0`).
16	`format`	*character*	Specifies the output format. Options include `"data.frame"` (default), `"markdown"`, `"latex"`, `"html"`, `"pipe"` (Pandoc pipe tables), `"simple"` (Pandoc simple tables), or `"rst"`.
17	`vars`	*character*	Names or IDs of the selected variables. Use the `fetch_vars` function to list available variables (default: `"all"`).
18	`freq`	*character*	Output frequency. Options are `"day"`, `"week"`, `"month"` (default), `"bimonth"`, `"quarter"`, `"season"`, `"halfyear"`, `"year"`, or `NULL` (disable aggregation).
19	`prep_fun`	*function*	A pre-processing function for time series data, applied before aggregation. For daily frequency, missing values are handled using the specified method. By default, if set to `NULL`, linear interpolation is applied.
20	`aggr_funs`	*function or list*	A function or list of functions to aggregate data at the specified frequency (`freq`). The default is `mean: function(x) mean(x, na.rm = TRUE)`.[e]
21	`postp_funs`	*function, list, or character*	A function or list of functions applied to the time series data of a single bin after aggregation. The default is `NULL`, indicating no post-processing.[e] Alternatively, set to `"external"` to apply post-processing from external files initialized with the `init_postp` function. See Appendix A for more details.

Note: Parameters marked with underlines are considered optional. (Referenced pages retrieved on February 2, 2025.)

[a] The complete data catalog of GEE is accessible at: https://developers.google.com/earth-engine/datasets/catalog.

[b] Allowable values and related dimensions can be found at: https://h3geo.org/docs/core-library/restable/.

[c] More information can be found at: https://developers.google.com/earth-engine/guides/scale.

[d] For example, use `keys = list("regions", c("source", "MODIS/061/MOD13A2", "NDVI"))` and `new_values = list(c("SO", "KE"), c("mean", "max"))` to update the `regions` to Somalia and Kenya and modify the `zonal_stats` for `"NDVI"` to `"mean"` and `"max"`.

[e] Both the `aggr_funs` and `postp_funs` parameters allow users to apply different functions to different variables. To do this, users can define a nested list, such as `aggr_funs <- list("default" = FUN_1, "Var_1" = list(FUN_1, FUN_2))`, where the default function is applied to any variable not explicitly specified. In `postp_funs`, the functions defined in `aggr_funs` can be referenced by their index, corresponding to the order in which they were listed. For example, `postp_funs <- list("default" = NULL, "Var_1/2" = FUN_3)` applies `FUN_3` as a post-processing function after the data is aggregated using `FUN_2`.

2.4. Unit Test

The `geeLite` package includes a test file generated using the `testthat` package (Wickham, 2011). This file automatically tests nearly all functions provided by `geeLite`, with the exception of `set_cli`, `fetch_regions`, and `init_postp`. The tests cover a range of tasks, including configuration generation, database building, database reading, configuration modification, and database updates.

If the user's Google account is not authenticated through a regular session of the `geeLite` package before running the tests, they can authenticate manually using the internal function `geeLite:::set_depend(conda = "rgee", user = NULL, drive = TRUE, verbose = TRUE)`.[9]

3. Illustrative Example

This section presents a practical example of the typical workflow for the `geeLite` package, demonstrating key steps such as configuration, database creation, updates, data retrieval, and processing. The example generates an up-to-date database containing the mean and standard deviation zonal statistics for the Normalized Difference Vegetation Index (NDVI) in Somalia and the Republic of Yemen. These statistics are calculated from January 1, 2010, using a grid system with a resolution level of three, which corresponds to areas of approximately $12,393$ square kilometers.[10]

3.1. Related Workflow

a) Configuration

First, the `fetch_regions` function is used to identify the ISO 3166-2 country codes for Somalia and Yemen. These codes are then used to configure the data generation process as follows:

[9]The parameters are described in detail in Table 2.

[10]NDVI data are sourced from the Moderate Resolution Imaging Spectroradiometer (MODIS) instrument. The dataset's profile is available at: https://developers.google.com/earth-engine/datasets/catalog/MODIS_061_MOD13A2 (retrieved: February 2, 2025).

```r
# Setting the config file:                                    [R code]
set_config(path = "path/to/db",
           regions = c("SO", "YE"),
           source = list(
              "MODIS/061/MOD13A2" = list(
                 "NDVI" = c("mean", "min")
              )
           ),
           resol = 3,
           start = "2010-01-01")
```

As a result, the configuration file (*config/config.json*) is generated at the specified target path.

Alternatively, the data generation process can be configured via the CLI. First, the CLI versions of the `geeLite` functions are set up using the `set_cli` function. Then, the data generation process is configured through the CLI as follows:[11]

```bash
# Setting the CLI files:                                      [Bash code]
Rscript ./geeLite/cli/set_cli.R --path "path/to/db"

# Change directory:
cd path/to/db

# Setting the configuration file:
Rscript cli/set_config.R --regions "SO YE" --source "list('MODIS/061/MOD13A2' = list('NDVI' = c('mean', 'min')))" --resol 3 --start "2010-01-01"
```

b) <u>Data Collection</u>

Once the configuration file is defined, the current database and its supplementary files (*state/state.json*, *log/log.txt*, along with the CLI versions of the functions) can be generated using the `run_geelite` function:

```r
# Building or updating the database:                          [R code]
run_geelite(path = "path/to/db")
```

Re-running the `run_geelite` function updates the dataset. Both the building and updating processes can be easily handled through the corresponding CLI script, as shown below:[12]

[11]The CLI versions of the functions automatically load all necessary dependencies.
[12]Using the CLI script is particularly useful for scheduling regular database updates.

```bash
# Building or updating the database:                          [Bash code]
Rscript cli/run_geelite.R
```

c) Configuration Modification

To update regions of interest, data sources, or the limit parameter, modify the configuration file using the `modify_config` function, then execute the `run_geelite` function to refresh the database. In the configuration, mean (`mean`) and minimum (`min`) statistics were initially selected for zonal statistics calculations. To tailor the database for the original task, the following code replaces the minimum statistic with the standard deviation (`sd`) in the configuration file and updates the database accordingly:

```r
# Modifying the configuration file:                              [R code]
modify_config(path = "path/to/db",
              keys = list(c("source", "MODIS/061/MOD13A2", "NDVI"))
              new_values = list(c("mean","sd")))

# Updating the database:
run_geelite(path = "path/to/db")
```

Alternatively, the same modifications to the configuration file and database can be made via the CLI using the following code:

```bash
# Modifying the configuration file:                           [Bash code]
Rscript cli/modify_config.R --keys "list(c('source', 'MODIS/061/
    MOD13A2', 'NDVI'))" --new_values "list(c('mean', 'sd'))"

# Updating the database:
Rscript cli/run_geelite.R
```

After the modifications, the generated SQLite database occupies 1.37 MB of disk space. Note that increasing the spatial resolution, number of indicators, aggregation methods, or geographic coverage will lead to greater disk usage. For example, raising the hexagonal resolution from size 3 to size 4 in the original configuration (reducing the bin size from approximately $12,393$ to $1,770$ square kilometers) would increase the file size to 9.33 MB. Despite the higher resolution, the file size would still be manageable, even for multiple countries.

3.2. Reading and Processing the Database

The generated SQLite database (*data/geelite.db*) contains two tables: `grid`, which stores information about the H3 bins, and `MODIS/061/MOD13A2`, which holds the collected zonal statistics.[13] To read, aggregate, and pre-process the database in R, the `read_db` function from the `geeLite` package can be used. In this example, the dataset was converted to a daily frequency using the default pre-processing method, which applies linear interpolation:

```r
# Reading SQLite tables:                                        [R code]
db <- read_db(path = "path/to/db", freq = "day")
```

This code snippet generates a list containing three elements: a simple features (sf) object representing the `grid` table, and two data frames corresponding to the variables `MODIS/061/MOD13A2/NDVI/mean` and `MODIS/061/MOD13A2/NDVI/sd`.

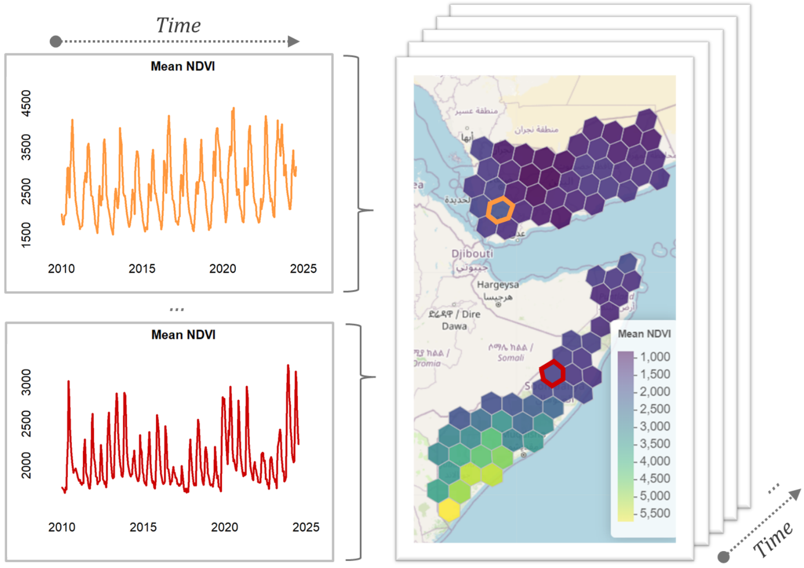
3.3. Visualizing the Database

After joining these tables using the H3 index (`id`), the spatiotemporal characteristics of the generated database are illustrated in Figure 2.[14]

[13] The contents of the SQLite database can be reviewed using the `fetch_vars` function.

[14] A code example for visualizing the mean NDVI values from the generated database is provided in Appendix B. To highlight the capabilities of the `geeLite` package, Figure A1 in Appendix C presents examples of higher-resolution H3 grid systems applied to NDVI data from Yemen.

Figure 2: Unscaled mean NDVI data retrieved from GEE (scale conversion factor: 0.00001)



Note: The generated database consists of forty-two H3 bins per country, each covering approximately 12, 393 square kilometers. It tracks NDVI time series data across these bins, with 5, 306 consecutive values derived from linear interpolation of the original 335 measurements collected between January 1, 2010, and July 11, 2024.

As shown in Figure 2, forty-two H3 bins were defined for both countries, each covering approximately 12, 393 square kilometers. The number of time series generated for each bin corresponds to the number of variables collected. The original dataset, collected between January 1, 2010, and July 11, 2024, comprised 335 measurements. After converting the data to a daily frequency using linear interpolation, the number of measurements increased to 5, 306.

3.4. Automation

To automate the updating of the generated database, Linux Crontab and the CLI version of the `run_geelite` function are used. For a monthly schedule, the following script can be utilized:[15]

```
# Monthly update with Crontab:                                    [Bash code]
(crontab -l 2>/dev/null; echo "0 0 1 * * Rscript cli/run_geelite.R") | crontab -
```

[15]For more information about Linux Crontab, see (Bradley, 2016).

4. Impacts and Conclusions

The development and release of `geeLite` represent a significant advancement in geospatial analysis, providing a user-friendly tool that simplifies the collection, management, aggregation, and processing of GEE data. By bridging the gap between GEE's powerful data capabilities and R users with varying levels of technical expertise, `geeLite` offers several key benefits:

- **Facilitating real-time geospatial monitoring and early warning systems:** Users can maintain continuously updated local geospatial databases, critical for real-time applications such as disaster response, environmental monitoring, and crisis management. Access to dynamic spatial data supports the development of early warning systems in these fields.
- **Improving research efficiency and data accessibility:** `geeLite` reduces the need for extensive coding, simplifying the collection, aggregation, and management of GEE data. Its intuitive design enables researchers to quickly gather and process large geospatial datasets, saving time and effort across multiple disciplines.
- **Enhancing reproducibility and collaboration:** With SQLite-based storage, databases are portable and self-contained, making it easy to share data and collaborate among research teams. This feature promotes reproducibility, allowing others to easily replicate experiments and validate findings.
- **Supporting long-term geospatial studies:** Designed for longitudinal research, `geeLite` excels at tracking geospatial features over time in studies such as climate change, land-use patterns, and biodiversity monitoring. Its ability to manage and update local databases ensures consistency for long-term analysis.

In conclusion, `geeLite` marks a significant step forward in making geospatial analysis more accessible, efficient, and reproducible. As the scale and complexity of Earth observation data continue to grow, tools like `geeLite` will be essential in transforming raw data into actionable insights that support informed decision-making and impactful research.

Appendix A. Post-Processing with External Files

The `init_postp` function generates a *postp* folder in the root directory of the database. This folder includes an editable R script (*functions.R*), where users can define custom post-processing functions, and a JSON file (*structure.json*) that outlines which functions should be applied to each variable during post-processing. When the `postp_funs` parameter in the `read_db` function is set to `"external"`, the post-processing logic specified in these external files is applied.

To illustrate the proper structure of these files, consider an example where the `"MODIS/061/MOD13A2/NDVI/mean"` variable is binarized based on its upper and lower deciles, while all other variables remain unchanged. First, define the functions `FUN_1 <- function(x) as.numeric(x < quantile(x, 0.1))` and `FUN_2 <- function(x) as.numeric(x > quantile(x, 0.9))` in the *functions.R* script. Then, to apply these functions to the mean NDVI variable, the *structure.json* file should be configured as follows:[16]

```
{                                                    [JSON file]
  "default": null,
  "MODIS/061/MOD13A2/NDVI/mean": [
    ["FUN_1", "FUN_2"]
  ]
}
```

Appendix B. Code Example for Data Visualization

Similar to Figure 2, the mean NDVI values from the database generated in Section 3 can be visualized using the following R script:

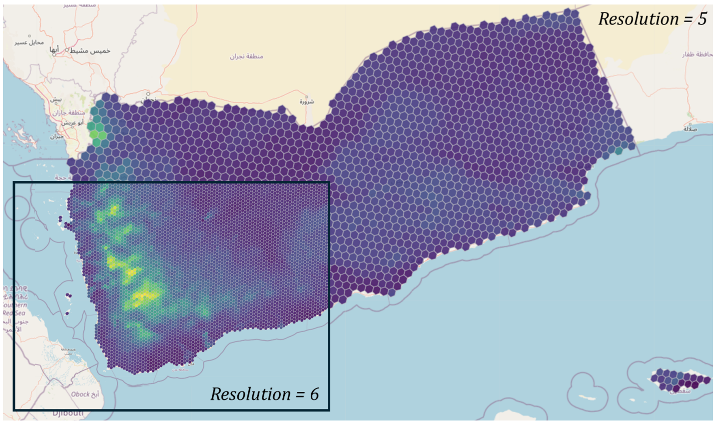
[16] Note that, unlike in the R script, the absence of post-processing is denoted by lowercase `null` in the JSON file.

[R code]

```r
# Install and load required packages:
# install.packages("sf")
# install.packages("dplyr")
# install.packages("leaflet")
library(sf)
library(dplyr)
library(leaflet)

# Read database and merge grid with MODIS data
db <- read_db(path = "path/to/db")
sf <- merge(db$grid, db$`MODIS/061/MOD13A2/NDVI/mean`, by = "id")

# Select the date to visualize
ndvi <- sf$`2020-01-01`

# Create a color palette function based on the values
color_pal <- colorNumeric(palette = "viridis", domain = ndvi)

# Create the leaflet map
leaflet(data = sf) %>%
  addTiles() %>%                                  # Add base tiles
  addPolygons(
    fillColor = color_pal(ndvi),                  # Fill color
    color = "#BDBDC3",                            # Border color
    weight = 1,                                   # Border weight
    opacity = 1,                                  # Border opacity
    fillOpacity = 0.9                             # Fill opacity
  ) %>%
  addScaleBar(position = "bottomleft") %>%        # Add scale bar
  addLegend(
    pal = color_pal,                              # Color palette
    values = ndvi,                                # Data values to map
    title = "Mean NDVI",                          # Legend title
    position = "bottomright"                      # Legend position
  )
```

Appendix C. High-Resolution H3 Grid

Figure A1: Examples of high-resolution H3 grid: unscaled mean NDVI data for Yemen retrieved from GEE (scale conversion factor: 0.00001)



Note: This figure demonstrates the capabilities of the `geeLite` package by collecting NDVI data for Yemen using high-resolution H3 grid systems. It illustrates two resolution levels: `resol = 5` and `resol = 6`, which correspond to grid cells with approximate areas of 252 square kilometers and 36 square kilometers, respectively.

References

Amani, M., Ghorbanian, A., Ahmadi, S.A., Kakooei, M., Moghimi, A., Mirmazloumi, S.M., Moghaddam, S.H.A., Mahdavi, S., Ghahremanloo, M., Parsian, S., et al., 2020. Google Earth Engine Cloud Computing Platform for Remote Sensing Big Data Applications: A Comprehensive Review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 13, 5326–5350.

Andrée, B.P.J., Chamorro, A., Kraay, A., Spencer, P., Wang, D., 2020. Predicting Food Crises .

Andrée, B.P.J., Chamorro, A., Spencer, P., Koomen, E., Dogo, H., 2019. Revisiting the Relation Between Economic Growth and the Environment: A Global Assessment of Deforestation, Pollution and Carbon Emission. Renewable and Sustainable Energy Reviews 114. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-85070497157&partnerID=MN8TOARS, doi:10.1016/j.rser.2019.06.028.

Arno, Z., Erickson, J., 2022. tidyrgee: 'tidyverse' Methods for 'Earth Engine'. R Package Version 0.1.0, https://github.com/r-tidy-remote-sensing/tidyrgee.

Aybar, C., Montero, D., Barja, A., Herrera, F., Gonzales, A., Espinoza, W., 2023. Combining R and Earth Engine, in: Cloud-Based Remote Sensing with Google Earth Engine: Fundamentals and Applications. Springer, pp. 629–651.

Aybar, C., Wu, Q., Bautista, L., Yali, R., Barja, A., 2020. rgee: An R Package for Interacting with Google Earth Engine. Journal of Open Source Software 5, 2272.

Banerjee, A., Ariz, D., Turyasingura, B., Pathak, S., Sajjad, W., Yadav, N., Kirsten, K.L., 2024. Long-Term Climate Change and Anthropogenic Activities Together with Regional Water Resources and Agricultural Productivity in Uganda Using Google Earth Engine. Physics and Chemistry of the Earth, Parts A/B/C 134, 103545.

Berner, L.T., Assmann, J.J., Normand, S., Goetz, S.J., 2023. 'LandsatTS': An R Package to Facilitate Retrieval, Cleaning, Cross-Calibration, and Phenological Modeling of Landsat Time Series Data. Ecography 2023, e06768.

Bradley, J., 2016. OS X Incident Response: Scripting and Analysis. Syngress.

Brovelli, M.A., Sun, Y., Yordanov, V., 2020. Monitoring Forest Change in the Amazon Using Multi-Temporal Remote Sensing Data and Machine Learning Classification on Google Earth Engine. ISPRS International Journal of Geo-Information 9, 580.

Burgueño, A.M., Aldana-Martín, J.F., Vázquez-Pendón, M., Barba-González, C., Jiménez Gómez, Y., García Millán, V., Navas-Delgado, I., 2023. Scalable Approach for High-Resolution Land Cover: A Case Study in the Mediterranean Basin. Journal of Big Data 10, 91.

Burke, M., Driscoll, A., Lobell, D.B., Ermon, S., 2021. Using Satellite Imagery to Understand and Promote Sustainable Development. Science 371, eabe8628.

Chen, S., Woodcock, C.E., Bullock, E.L., Arévalo, P., Torchinava, P., Peng, S., Olofsson, P., 2021. Monitoring Temperate Forest Degradation on Google Earth Engine Using Landsat Time Series Analysis. Remote Sensing of Environment 265, 112648.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., Moore, R., 2017. Google Earth Engine: Planetary-Scale Geospatial Analysis for Everyone. Remote Sensing of Environment 202, 18–27.

Hamidi, E., Peter, B.G., Muñoz, D.F., Moftakhari, H., Moradkhani, H., 2023. Fast Flood Extent Monitoring with SAR Change Detection Using Google Earth Engine. IEEE Transactions on Geoscience and Remote Sensing 61, 1–19.

Kazemi Garajeh, M., Haji, F., Tohidfar, M., Sadeqi, A., Ahmadi, R., Kariminejad, N., 2024. Spatiotemporal Monitoring of Climate Change Impacts on Water Resources Using an Integrated Approach of Remote Sensing and Google Earth Engine. Scientific Reports 14, 5469.

Kong, D., 2022. rgee2. R Package Version 0.1.1, https://github.com/rpkgs/rgee2.

Kurbucz, M.T., Andrée, B.P.J., 2024. geeLite R Package. URL: https://github.com/mtkurbucz/geeLite.

Marconcini, M., Metz-Marconcini, A., Esch, T., Gorelick, N., 2021. Understanding Current Trends in Global Urbanisation: The World Settlement Footprint Suite. GI_Forum 9, 33–38.

Massicotte, P., South, A., 2024. rnaturalearth: World Map Data from Natural Earth. URL: https://docs.ropensci.org/rnaturalearth/. R Package Version 1.0.1.9000, https://github.com/ropensci/rnaturalearth, https://docs.ropensci.org/rnaturalearthhires/.

Morales, N.S., Fernández, I.C., Durán, L.P., Pérez-Martínez, W.A., 2023. RePlant Alfa: Integrating Google Earth Engine and R Coding to Support the Identification of Priority Areas for Ecological Restoration. Land 12, 303.

Müller, K., Wickham, H., James, D.A., Falcon, S., 2024. RSQLite: SQLite Interface for R. URL: https://rsqlite.r-dbi.org. R Package Version 2.3.7, https://github.com/r-dbi/RSQLite.

O'Brien, L., 2023. h3jsr: Access Uber's H3 Library. URL: https://obrl-soil.github.io/h3jsr/. R Package Version 1.3.1.

Penson, S., Lomme, M., Carmichael, Z., Manni, A., Shrestha, S., Andrée, B.P.J., 2024. A Data-Driven Approach for Early Detection of Food Insecurity in Yemen's Humanitarian Crisis. Technical Report. The World Bank.

Tamiminia, H., Salehi, B., Mahdianpari, M., Quackenbush, L., Adeli, S., Brisco, B., 2020. Google Earth Engine for Geo-Big Data Applications: A Meta-Analysis and Systematic Review. ISPRS Journal of Photogrammetry and Remote Sensing 164, 152–170.

Tavakkoli Piralilou, S., Einali, G., Ghorbanzadeh, O., Nachappa, T.G., Gholamnia, K., Blaschke, T., Ghamisi, P., 2022. A Google Earth Engine Approach for Wildfire Susceptibility Prediction Fusion with Remote Sensing Data of Different Spatial Resolutions. Remote Sensing 14, 672.

Team, S.S., 2022. SAEplus. R Package Version 0.1.0, https://github.com/SSA-Statistical-Team-Projects/SAEplus.

Uber, T., 2021. H3: A Hexagonal Hierarchical Geospatial Indexing System.

Ushey, K., Allaire, J., Tang, Y., 2024. reticulate: Interface to 'Python'. URL: https://rstudio.github.io/reticulate/. R Package Version 1.35.0, https://github.com/rstudio/reticulate.

Velastegui-Montoya, A., Montalván-Burbano, N., Carrión-Mero, P., Rivera-Torres, H., Sadeck, L., Adami, M., 2023. Google Earth Engine: A Global Analysis and Future Trends. Remote Sensing 15, 3675.

Wang, D., Andrée, B.P.J., Chamorro, A.F., Spencer, P.G., 2022. Transitions Into and Out of Food Insecurity: A Probabilistic Approach with Panel Data Evidence from 15 Countries. World Development 159, 106035. URL: https://www.sciencedirect.com/science/article/pii/S0305750X2200225X, doi:https://doi.org/10.1016/j.worlddev.2022.106035.

Wang, L., Diao, C., Xian, G., Yin, D., Lu, Y., Zou, S., Erickson, T.A., 2020. A Summary of the Special Issue on Remote Sensing of Land Change Science with Google Earth Engine.

Wickham, H., 2011. testthat: Get Started with Testing. R J. 3, 5.

Wickham, H., Hester, J., Chang, W., Hester, M.J., 2022. Package 'devtools'.

Zheng, Z., Wu, Z., Chen, Y., Yang, Z., Marinello, F., et al., 2021. Analyzing the Ecological Environment and Urbanization Characteristics of the Yangtze River Delta Urban Agglomeration Based on Google Earth Engine. Acta Ecologica Sinica 41, 717–729.