Creating NcLibrary

Internship report

David Jiménez Omeñaca

Department of Mathematics
University of Zaragoza
Zaragoza, Spain

2022-2023

Contents

1 Introduction

In the realm of statistics research, managing extensive databases often presents compatibility challenges. Frequently, the data is acquired in formats that lack user-friendliness. Despite the availability of libraries, the existence of diverse database structures results in integration challenges. Consequently, the task of unifying this diverse data becomes an arduous one. The essence of this internship lies in addressing this very issue as well as the retrieval of the data from the databases.

To address these challenges, I developed an R-library aimed at working with reanalysis data from the ERA5 general database and observed values from ECAD. The library facilitates:

- Effortless retrieval of ERA5 data through a user interface.
- Extract data from ERA5 in a user-friendly list to further statistic analysis.
- Incorporation of observed ECAD values into the processed data.
- Sub-setting the previous data based on specific coordinates.

2 Objectives

This internship is driven by a set of well-defined objectives, each aimed at addressing specific challenges and achieving tangible outcomes within the realm of meteorological data management and statistics research. We pursued the following objectives:

1. **Main objective:**

   a To enhance compatibility and accessibility of meteorologic data within statistic research by developing a unified data management system.

2. **Specific objectives:**

   a **Ensure user-friendly interface:**
   - Design a intuitive user interface to retrieve data from ERA5 general database.
   - Prioritize user experience by presenting data in a clear and organized manner.

   b **Design and Implement Data Abstraction Layer:**
   - Develop a versatile data abstraction layer capable of interpreting multiple data formats.
   - Implement methods for data conversion and integration into a standardized format.

   c **Develop Proficiency in Meteorological Databases:**
   - Acquire a comprehensive understanding of various prominent meteorological databases as well as the libraries that work with them.
   - Familiarize oneself with the distinct data formats utilized by different databases.

   d **Create a R-library from scratch.**
   - Familiarize with the R packages and the required structure to create a R-library.
   - Generating comprehensive function documentation to provide users with clear guidance on how to effectively utilize these functions.

   e **Test and Validate System Performance:**
   - Conduct rigorous testing to ensure accurate data conversion and retrieval.
   - Validate the system's efficiency and effectiveness in handling diverse meteorological datasets.

3 Methodology

4 Internship Activities

This section offers a comprehensive exploration of the activities undertaken during the internship, detailing the methodologies employed, obstacles encountered, and the valuable insights gained.

4.1 Data Retrieval

The first task was the data retrieval from the meteorological database. The databases we aimed at were:

- Complete ERA5 global atmospheric reanalysis
- ECAD daily data.

The initial challenge encountered while trying to get the data from this databases revolved around the absence of a user-friendly interface to retrieve the data. Instead data request was through an API requiring to use a Python library called `cdsapi`. Moreover certain request details required consultation from a separate web page. Consequently, a decision was made to streamline this process by creating a simple user interface aimed at simplifying these operations.

On a contrasting note, the second database boasted a user-friendly interface that significantly simplified the data retrieval process. The output of this request was a `.zip` file with the data stored in `.txt` files within it.

4.1.1 User interface

For the creation of the user interface, the Python library `tkinter` was used. This library offers an extensive and straightforward list of tools to create a simple user interface, though grasping its intricacies initially proved challenging. As previously mentioned, the request's variable names were accessible via a webside. One must decide which variables wants to retrieve and then take note of their ID's which is quite a challenge. Therefore, a HTML-to-XLSX converter was used to generate a `.xlsx` file with the data of every variable, including their names and IDs. Within the user interface, a `Combobox` widget was employed. It is a auto-complete entry selector. As a result, users can effortlessly pick the variables of interest, which in turn generates a list of corresponding IDs. Attempts were made to extract HTML data using Python libraries such as `BeautifulSoup`. However, this approach posed more challenges than initially anticipated.

After the user fills in all the required fields and clicks the "Submit all" button, a request is made to the ERA5 database. Note that this petition is rather slow, specially when dealing with large amounts of data. In such cases, it's advisable to consider implementing multi-session or multi-threading techniques to ensure you can continue working while the request is being processed.

Moreover, a data validation control mechanism was integrated into the user interface to prevent errors during the request process. This step was taken to enhance the overall reliability of the process and to reduce the likelihood of encountering issues.

The usage of this app within the R-library through the function named `create_nc_file_app` which creates a `.nc` file with the information requested through the UI.

However, since maybe some users find more comfortable to create the request as a raw function, `create_nc_file_data` was created to fulfill this requirements. This function takes as a parameters directly all the fields that one must fill in the UI except the variables IDs, which extraction must be manual through the link `https://apps.ecmwf.int/codes/grib/param-db/`.

A screenshot of the user interface can be depicted in [Figure 1].

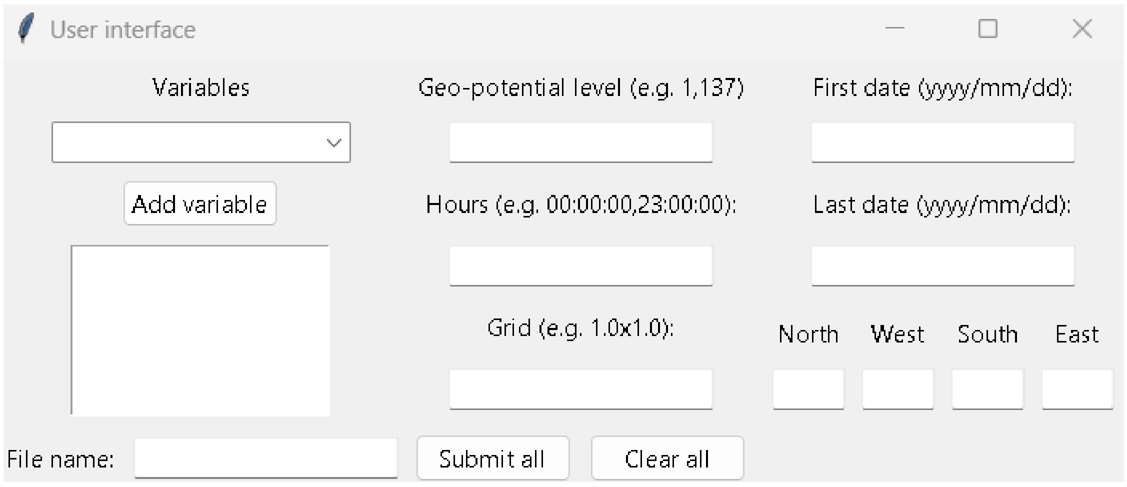

Figure 1: Screenshot of the user interface.

4.2 R Reticulate

The data retrieval presented a quite challenging problem. Python's environment became essential for creating the user interface and executing requests.

To solve this, the R-library called `Reticulate` emerged as a solution. It grants the capability to establish a Python environment within R, enabling the execution of Python scripts. Initially, this seemed promising. However, multiple problems surfaced while working with it. The final solution was to create a new Python environment and download in it all the Python-libraries needed to execute the code. This is made via the function `set_reticulate`. Remember that once all the data is requested from ERA5 it is advisable to remove this new environment through the function `close_reticulate`.

4.3 Data Manipulation

Once the data is requested, two distinct file types are obtained:

- A `.nc` file containing ERA5 database information.
- A `.zip` file with the data from ECAD database.

Each of these files presented unique challenges, along with distinct solutions.

4.3.1 NC files

A `.nc` or `netCDF` file is multi-dimensions format for storing climate data. It operates on the premise of dimensions and variables. Each variable has a list of dimensions and a multi-dimension matrix containing the data. Moreover, each dimension has a length and an array of values (of this exact length). The R-library `ncdf4` was used to read the data from such files

However, this matrix-representation does not suit typical statistical research needs, which would rather to have a single 2-D object with the data within. The main idea behind this transformation is flatten the matrix into a two dimensions data frame. In this configuration, rows will represent temporal data, while the remaining dimensions will be encapsulated in the columns. Is noteworthy that the general number of dimension is 4: longitude, latitude, level and time. This will create a huge problem that will be covered later.

Since the columns must hold 3 dimensions, column names adhere to the syntax:

$$\text{varname\_level\_longitudeStr\_latitudeStr\_hour} \tag{1}$$

where `longitudeStr` and `latitudeStr` are the string representation of the longitude and latitude respectively (see more here). Variable name are crucial because if there is more than one variable, they would act as an additional dimension, also flattened into columns. Finally, the hours are also represented in the columns due to the same reason. In this way each row will be daily which will be very useful further on. However, there are configurations to delete this default behaviour that can be depicted in the library documentation.

This operations are covered by the function `read_nc_file` which will return a list with some important information alongside the data frame. The function offers numerous parameters to customize results, as detailed in the function's documentation. Additionally, the function `write_csv` offers the capability to write the data frame into a `.csv` file.

As it was mentioned before, a dimension-related issue arose. When making a request to the ERA5 database with just one geo-potential level, the file didn't detect it as a dimension due to its length of one. This contrasted with other dimensions, where a length of one still retained dimension status. To rectify this, the `.nc` file required reformatting to add this new dimension. This task required the use of Python and the `netCDF` library.However, this library extracted the data in a notably distinct format which complicated every part of the process. Despite the complexity, the task was successfully accomplished.

DAY	**MONTH**	**YEAR**	**DATE**	**t_150_15N_15E**	**...**
1	2	2000	2000/02/01	100	...
...	...	...	...	...	...

Table 1: Example of the resulting data frame.

An example of the data frame obtained can be depicted in [Table 1] and an example of the result list of the function in [Figure 2].

4.3.2 ECAD data

The data from the ECAD database was stored in `.txt` files and compressed in a `.zip` file. There is always common files, such that `stations.txt` which have information about all the stations mentioned in the file. Each variable and its data is stored in an independent file. This made everything much easier because functions such that `readLine` and `utils::read.table` could be used to read the data frame and additional information of each variable. However, this idea was very optimistic because initially it was thought that every file would follow the same structure. However, there was slight differences between the variables files. Blended and not blended files differs and within the non-blended variables there were differences too. Taking everything into account a function called `read_eca_zip` was created. It returns a list with useful information stored in `.zip file`. Its structure can be depicted in [Figure 2].

4.4 Merge observed values

Once the data is correctly extracted the following step was to merge this information together. In the realm of statistics it is important to be able to work with reanalysis data along with observed values. As a result of the previous section two different list contain this two types of information. A function called `get_merge_observations` join to the ERA5-list a variable data frame from the ECAD-list. While creating it a important decision arose: both list have a distinct temporal support. Therefore it was decided that the new time set would be the intersection of both, warning if the ERA5-list time shrinks after the operation. It is important to recall what it was said about "daily rows". Since the ECAD data is daily, it was important to represent the hours as a new dimension so the tables could fit together. The observed values joined will be represented as a new row in the data frame and its name would be depicted as an element of the array

```
era5_list
|__ temporal_info
|   |__ daily
|   |__ first
|   |__ last
|   |__ vals
|__ df
|__ vars
|__ geo_levels
|__ creation_time
|__ extended_hours
|__ hours
|__ grid
|__ latitude
|   |__ range
|   |   |__ min
|   |   |__ max
|   |__ vals
|__ longitude
|   |__ range
|   |   |__ min
|   |   |__ max
|   |__ vals
|__ file
|__ sep
|__ obs_values
```

```
ecad_list
|__ vars
|   |__ var1
|   |__ ...
|__ nvar
|__ creation_time
|__ origin_file
|__ blended
|__ stations_info
    |__ STAID
    |__ STANAME
    |__ CN
    |__ LAT
    |__ LON
    |__ HGHT
```

```
NcLibrary
|__ DESCRIPTION
|__ NAMESPACE
|__ inst
|   |__ Non R-Code
|__ R
|   |__ R code
|__ man
    |__ Rd files
```

Figure 2: List structure of **read_nc_file** (left), list structure of **read_eca_zip** (mid) and package configuration (right)

"obs_values". This operation can be recursive and multiple observed values can be added.

4.5 Locate the data frame

The previous section manifested a new needed feature. Most of the times, while doing statistics, we work with rather large amounts of data. However, sometimes we want to focus our studies on a particular region where we have observations, so we would not need every shred of information. A function that returns a subset of the all the provided data given a city was thought to be very handy. So `get_subset_city` was created to fulfil this requirement. This function, subsets the main ERA5-list given a city name and a depth. The idea of depth can be depicted in [Figure 3]. In contrast of the previous function mentioned, this one is meant to be used once per list.

4.6 Extra conversion functions

As it was mentioned before, there was two ways to represent a longitude and latitude. The numerical version where positive numbers stands for east and north coordinates and negative numbers, west and south respectively. However, sometimes the string representation (using the characters N,W,E,S) is more intuitive and easily recognized. Therefore it was very useful to create functions which transform the longitude and latitude between the two representations. Firstly, this functions were meant to be private, as much others, but it was thought that they could be quite handy in some scenarios. The functions are: `lat_to_str`, `str_to_lat`, `lon_to_str`, `str_to_lon`.

4.7 R Library

Once each function demonstrated its proper functionality, the next step was to assemble them into a cohesive library. This task required the use of tools like `roxygen2` or `devtools` alongside the creation of comprehensive documentation for each function. However, meticulous organization of files was essential. The resultant file configuration is outlined in [Figure 2].

The main folder consists in two 3 new directories: **R** for R code, **inst** for Python and .csv files and **man** directory, for the generated `.Rd` files. This configuration

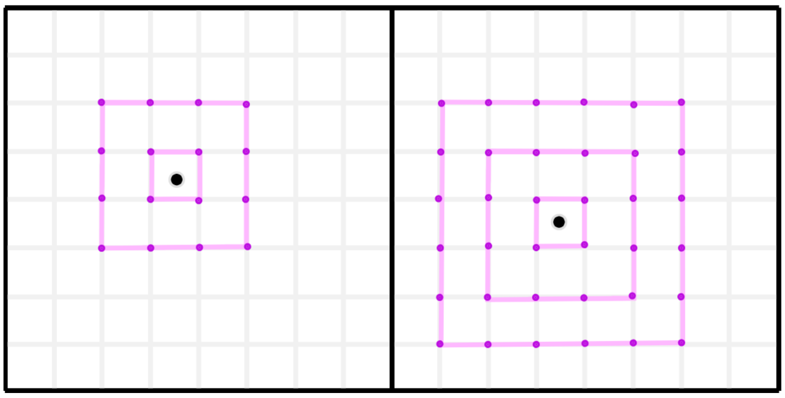

Figure 3: Grid depth of two(left) and tree(right).

wasn't arbitrary; it adhered to specific requirements.

Additionally it was compulsory to create a `DESCRIPTION` file containing the library configurations. Since our data required non-R files, it was needed to copy this files to the library by adding the relative path of this files in the `DESCRIPTION` file. After all preparatory steps, it was time to build the `.Rd` files with the function `devtoos::document` and then, check the package with `devtools::check`. Post resolution of errors, warnings, and notes, the library was created using `R-Studio` "Build" window. Once completed, the library was accessible from every R-Script in the local computer using `library(NcLibrary)`.

Notably, adjustments were necessary after employing the library in other scripts due to path considerations. Formerly, the Python files were referenced by using relatives paths, assuming the working path was the library's. However, to access these files from diverse paths, `system.file` was indispensable. This function facilitates path retrieval from the library using the `package` parameter.

With the library successfully formed, the next phase involved generating an `.html` page using the package `pkgdown`. However, the lack of documentation and the previous knowledge needed, made everything rather difficult. Fortunately, a template discovered on GitHub (explore the template here) offered assistance. Modifying the template provided insight into the intricacies of developing a package site. However, following the guide, it was not very challenging.Guided by the template, the process was more manageable. After refining the layout and creating a basic logo for the library, the final page was successfully created.

4.8 Testing

A critical aspect of our endeavor involved thoroughly testing the R-library, a phase that demanded considerable time. While R-libraries offer specialized mechanisms for constructing robust test benches, such as utilizing tools like `testthat`, this avenue was ultimately not pursued. This decision was influenced by the unique nature of the functions being tested. Instead, meticulous manual tests were conducted to ensure the dependability of the resulting code and the accuracy of its outcomes.

5 Results and System Performance

In this section we delve into the performance of the system when dealing with substantial volumes of data. There are essentially two metrics under consideration: the time taken for data retrieval from the ERA5 database and the time required to process this data using the function `read_nc_data`. Crucially, plotting these two graphs together could reveal any potential correlation between them.

With this aim in mind, a R-script was made. This script generates 30 queries, systematically increasing in size, and records both retrieval and processing times. Upon execution, the outcomes are illustrated in [Figure 4]. The x-axis represent the number of cells within the `.nc` matrix, i.e. the product of all dimensions lengths. Meanwhile, the y-axis represent the time spent in minutes. It is easy to see a common behaviour: while the petition time increases considerably in proportion to size, the manipulation time remains relatively constant or with a lower order increasing rate. However, this test can fluctuate depending on the ERAS5 server, the internet speed or even the computer performance.

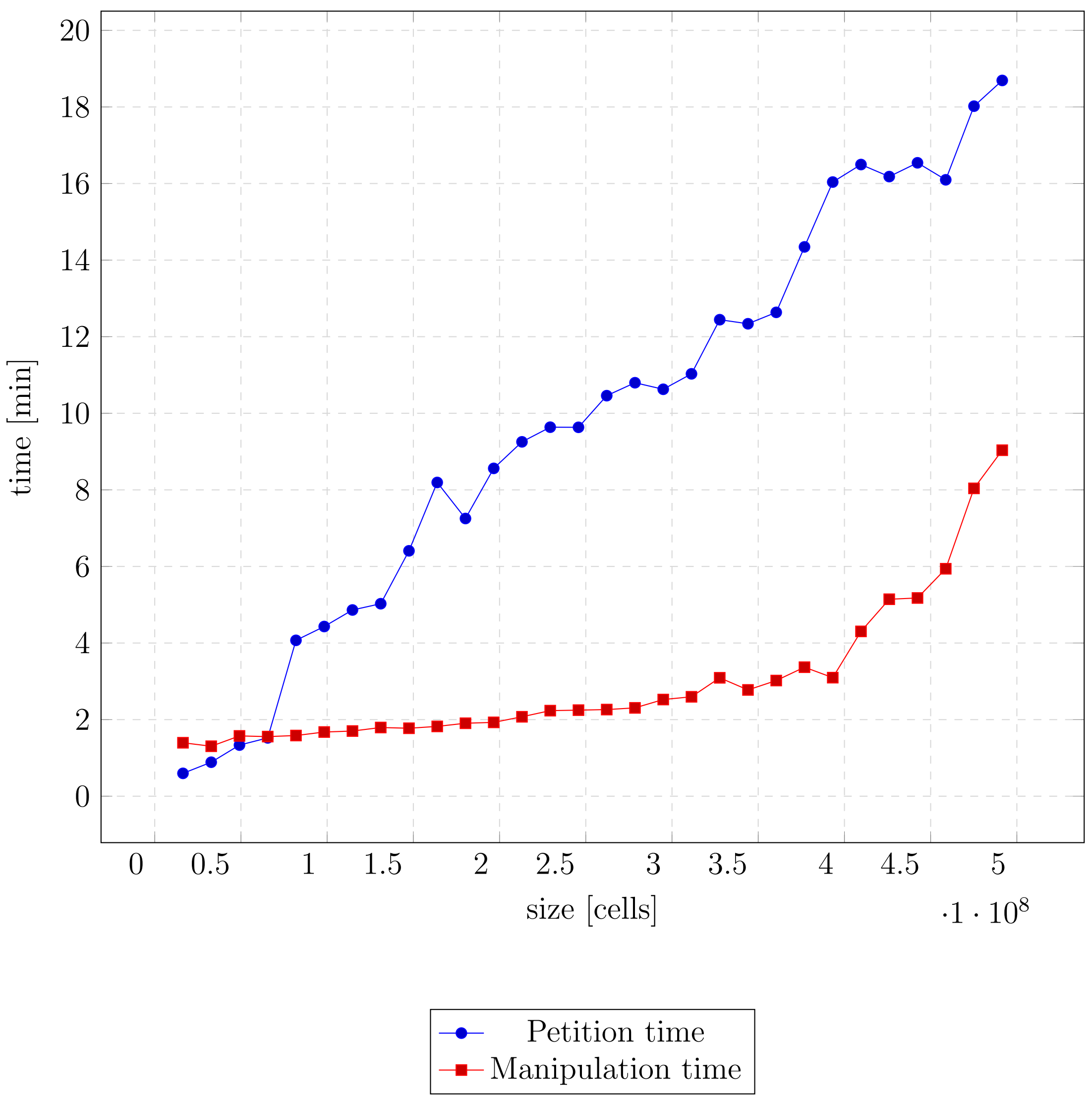

Figure 4: Graph comparing the time expended in the manipulation of the time and in the petition respect to the number of cells of the .nc matrix

6 Conclusion

References