Identification and update of Ecological Infrastructure with JuypterLab

User guide - English version

Audrey Lambiel

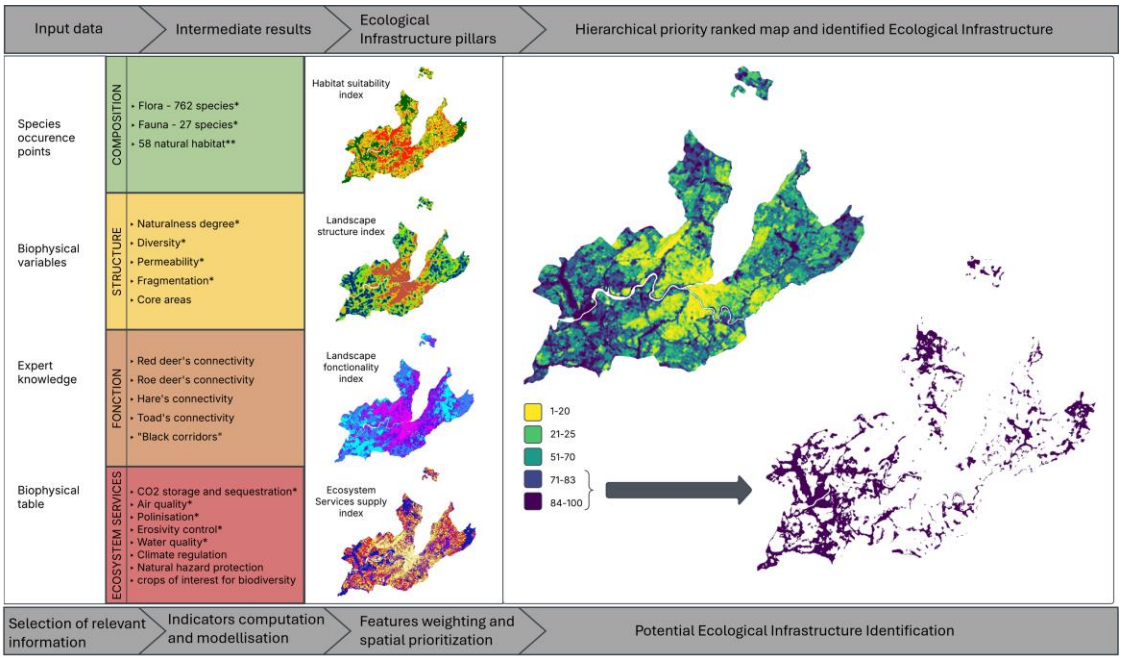
2025-10-15

Introduction

Environmental planning and biodiversity conservation increasingly rely on reproducible, transparent, and scalable digital infrastructures. In this context, **JupyterLab** provides an interactive development environment particularly well-suited for managing complex workflows, especially those involving large amounts of spatial and ecological data.

This user guide accompanies the technical note titled *"A Digital Twin approach for the identification and update of Ecological Infrastructure"* and provides practical instructions to reproduce the analyses presented in the article. It is designed to help users deploy the shared JupyterLab environment, install the necessary software components, and implement the applied method to identify and update the Ecological Infrastructure (EI) of the Canton of Geneva.

The methodology is based on a biodiversity assessment structured around thematic pillars, each composed of ecological indicators. These indicators are computed through an automated **R-based** processing chain, and the results are prioritized using the spatial planning tool **Zonation 5**. All scripts are annotated, and automatic metadata generation ensures traceability and reproducibility of the results.



The figure above presents the general framework used to establish the EI in the canton of Geneva, based on four pillars of biodiversity selon Honeck, Sanguet et al. (2020). After hierarchically ranking all pixels from 1 to 100, only the top 30% are retained and identified as ecological infrastructure. In the figure, indicators marked with a * are those currently selected and fully implemented in the automation process- Those marked with ** are

currently selected and partially implemented (i.e. automation has been completed for all natural habitats, but not for the isolated trees layers) in the automation process.

The available material in the GitHub repository includes R scripts, configuration files, and documentation for the identificatoin and updating of EI, in accordance with the above presented methodology proposed by GE21. This work was designed to be used via a shared environment on JupyterHub, and the scripts were therefor written to be executed in this type of environment.

By following this guide, users will be able to reproduce the analyses by adapting the method to other territories or datasets.

Objectives of this practical guide

The purpose of this guide is to provide users with clear and practical documentation to:

- **Get started** with JupyterLab
- **Install the necessary software environments** to run the scripts
- **Access the scripts available** in the project's GitHub repository
- **Run the scripts** for indicator calculation and spatial prioritization
- **Understand the structure of the generated results**, as well as the associated metadata

It is intended for anyone wishing to reproduce the analyses presented in the publication, adapt the method to other territories, or contribute to the development of FAIR digital infrastructures for environmental planning.

What is JupyterLab ? A biref introduction

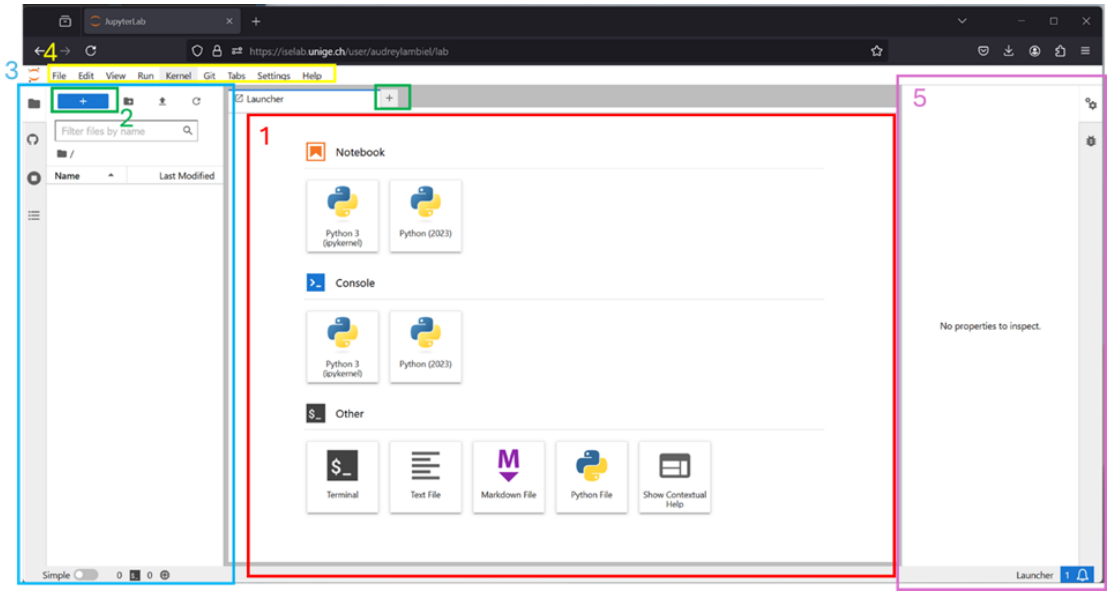
JupyterLab is an open-source interactive development environment designed for scientific research and data analysis. It allows users to write, run, and document code in multiple languages (notably R and Python), while integrating visualizations, text files, consoles, and terminals within a unified interface.

In this project, JupyterLab is used as the **central infrastructure** to:

- Automate the calculation of ecological indicators
- Centralize data and scripts
- Facilitate collaborative work among partners
- Ensure traceability and reproducibility of results

The environment is deployed on a shared server (JupyterHub), but can also be installed locally. It allows each user to work in a personal space while accessing shared resources (scripts, data, virtual environments).

JuypterLab Interface



The JupyterLab interface includes five main components:

1. **Main workspace area** with document tabs and activities. In this area, you can organize documents (text files, notebooks, etc.) and other activities such as terminals or code consoles into resizable and subdividable tab panels (simply drag and drop a new tab onto the desired panel). In the screenshot below, the workspace is organized into three distinct blocks: a notebook in the top-left, a new Launcher page in the top-right, and a terminal in the lower part of the screen.

2. **Launcher tab**, which displays available kernels and consoles. It allows you to start a new notebook or console based on any of the available kernels, and also to launch a new terminal.

3. **Left sidebar**, which includes:

   - **File browser**, where the top icons let you open a new Launcher, create a new folder, upload a file, etc. When navigating through your folder structure, any new folder or file is automatically created in the currently selected directory.

   - **List of active kernels, terminals, and open tabs**. When you close a notebook, code console, or terminal, the underlying kernel or terminal running on the server continues to operate. This tab allows you to see which kernels or terminals are still running and to reopen or shut them down if needed.

4. **Top menu bar**, which provides actions for managing active files, kernels, workspace layout, and more.

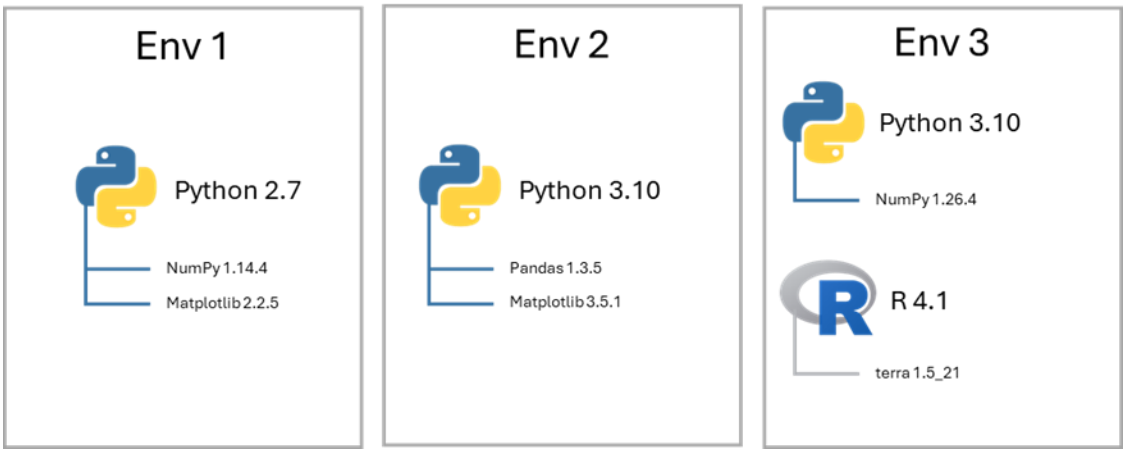
5. **Right sidebar**, which includes a property inspector (active in notebooks) and a debugger.

Terminal

JupyterLab terminals offer full support for command-line interpreters (bash, tsch, etc.) on Mac/Linux and PowerShell on Windows. Terminals run on the system where the Jupyter server is hosted. They are also used to create and manage virtual environments. To open a new terminal, click on the "+" button to open a new Launcher, then click on the terminal icon. Closing a terminal tab does not stop its execution on the server; you can reopen it using the "Running Terminals and Kernels" tab in the left sidebar.

Virtual environment

A virtual environment is an isolated instance of the Python interpreter that allows you to install libraries and dependencies specific to a project without interfering with other projects or with globally installed libraries on the server. When working with JupyterLab, you use a specific kernel (e.g., Python) to execute code. You can have multiple kernels and virtual environments for different projects, ensuring that each project uses the correct versions of the required libraries. Similarly, it is possible to define environment variables tailored to your specific needs.



Kernels

Kernels are separate processes launched by the server that execute your code in different languages and customized programming environments. JupyterLab allows you to connect any open text file to a code console and a kernel. For example, in a virtual environment where both Python and R are installed, you can set up a Python kernel to run Python code, and an R kernel to execute R commands.

Kernels are created from an activated virtual environment using a simple command in the terminal. Once created, the kernel becomes visible and can be used in notebooks and consoles.

Notebook

Notebooks (.ipynb) are documents that combine executable code with narrative text (Markdown), equations (LaTeX), images, and more.

Console

Code consoles allow you to execute code interactively within a kernel. Each line sent to the console is executed immediately, and the variables stored in memory are retained.

System requirements

JupyterLab is cross-platform and runs on:

Windows	10 later (64 bits)
macOS	10.12 (Sierra) or later
Linux	Ubuntu, Debian, Fedora, CentOS, Arch, etc.

You will need Python (version 3.8 or later) and pip (version ≥ 21.0 recommended).

The system can be used via a JupyterHub server (i.e., through a web browser, except Internet Explorer) or installed locally. In that case, download JupyterLab Desktop or run the following commands in your machine's terminal.

- with `pip`

```
pip install jupyterlab
```

- or with `conda`

```
conda install -c conda-forge jupyterlab
```

Step 1 : Set up workspace

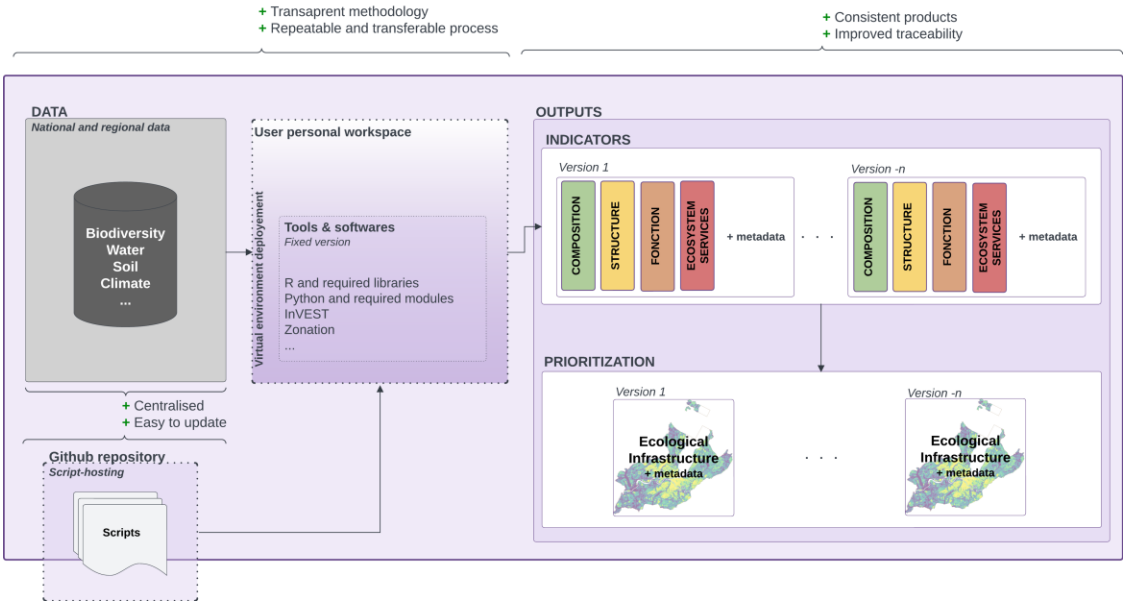
Environment architecture

We propose structuring the workspace around four main components:

- **The GitHub repository**, where scripts and configuration files are stored
- **The user's personal space**, where environments are deployed and scripts are executed
- A `DATA` folder, containing the **data required for indicator** calculations
- An `OUTPUTS` folder, where the generated **results** are automatically saved

In the following sections of this guide, we assume that a shared workspace named `shared_workspace` has been set up on JupyterLab, and that it contains the `DATA` folder. However, it is also possible to work with a `DATA` folder located directly in one's personal space.

Note: A more detailed description of this architecture is available in the associated technical note (see end of guide).



Clone GitHub repository

From your `/home` workspace on JupyterLab, open a **terminal** and clone the GitHub repository into your space.

```
git clone https://github.com/ALambiel/envirospace_IE
```

The repository presents the following structure:

```
envirospace_IE/
├── config/                      # files for deploying required environments
│   ├── rspatial.yml             # environment for spatial analysis with R
│   └── invest3141.yml           # environment to run InVEST 3.14.1
├── run/
│   ├── <ind_pillar_indicator.r> # one script for each indicator
│   ├── template.r                 # template to create new indicator's script
│   ├── prio_zonation.r            # running Zonation prioritization project
│   └── <tools.r>                  # additional scripts for pre-processing
└── README.md
```

Define this repository as working directory

```
cd envirospace_IE
```

Deploying virtual environments

Virtual environments are defined in two configuration files:

- `rspatial.yml`: for spatial analyses in R
- `invest3141.yml`: for indicators based on the InVEST model (R + Python)

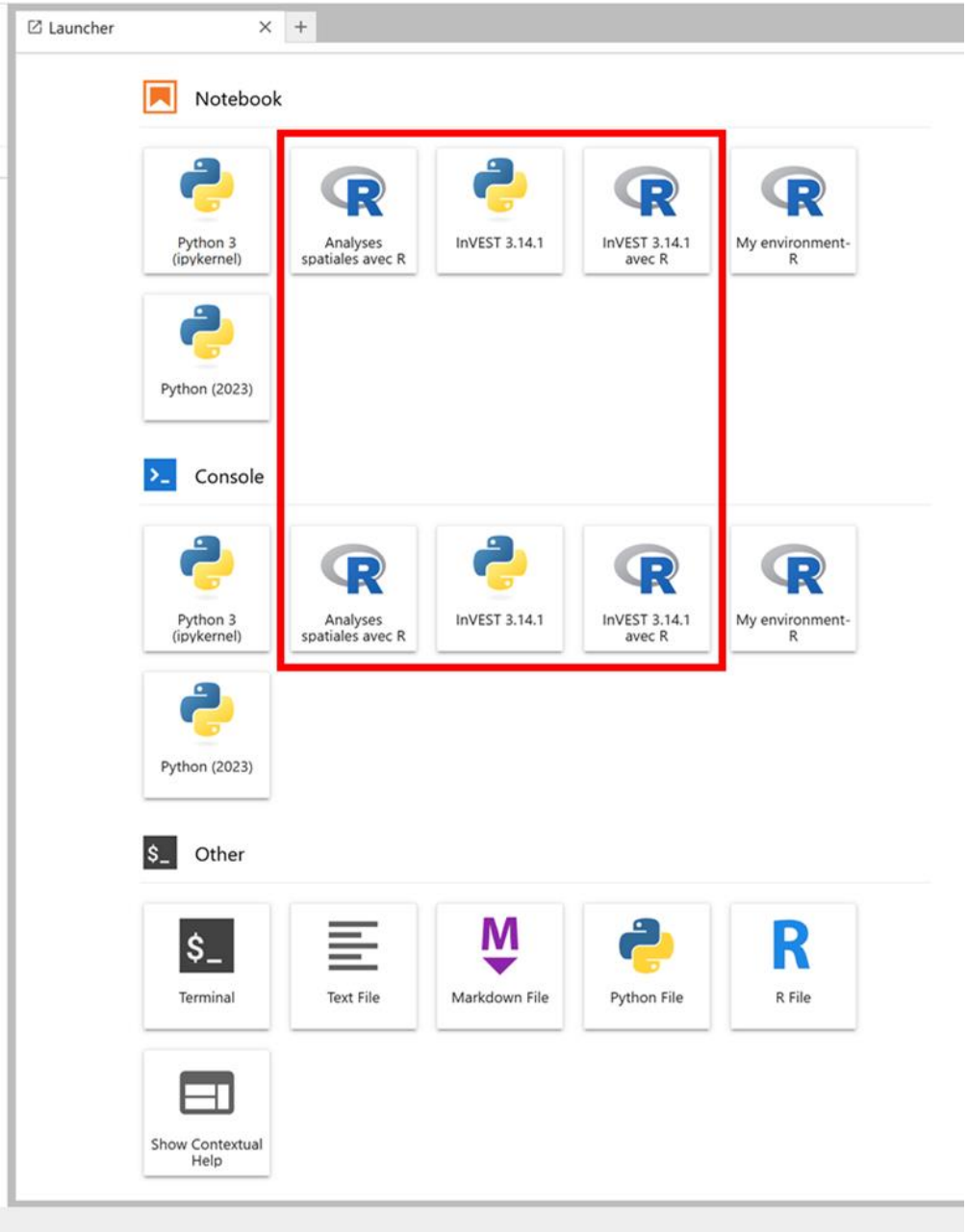
Start by deploying the `rspatial` environment and creating the associated kernel.

```
# create environment from .yml file
conda env create -f config/rspatial.yml

# activate
conda activate rspatial

# launch R
R

# install kernel
IRkernel::installspec(name = 'rspatial', displayname = 'Spatial analysis with
R')

# quit R
q()
```

Do the same with the second virtual environment `invest3141`

```
conda env create -f config/invest3141.yml
conda activate invest3141
pip install natcap.invest==3.14.1
R
IRkernel::installspec(name = 'invest3141r', displayname = 'InVEST 3.14.1 with
R')
q()

# installer python kernel
python -m ipykernel install --user --name invest3141--display-name "InVEST
3.14.1"
```

In your `Launcher`, the new icons corresponding to the environments and kernels you just created should now appear.



Install Zonation 5

The prioritization step relies on the use of the Zonation 5 software, which must be installed as an AppImage within the JuypterLab environment. First, download the Zonation 5 AppImage by clicking on the following link: Zonation5_Linux.zip (latest release).

Once the ZIP file is downloaded, unzip it and move the `zonation5` file into a folder within your JupyterLab environment (for example, `envirospace_IE/zonation_folder`).

Finally, make it executable and extract the AppImage by opening a terminal and running the following commands:

```
cd /path/to/zonation_folder
chmod +x zonation5
./zonation5 --appimage-extract
```

A new folder named `squashfs-root/` should appears now in your folder `zonation_folder`, and contains Zonation exectuable.

Access to your shared data base

Note: As mentioned earlier, we recommend working with a shared database. This means that a shared environment on JupyterHub has been set up, and your administrator has granted you access to this folder and its contents.

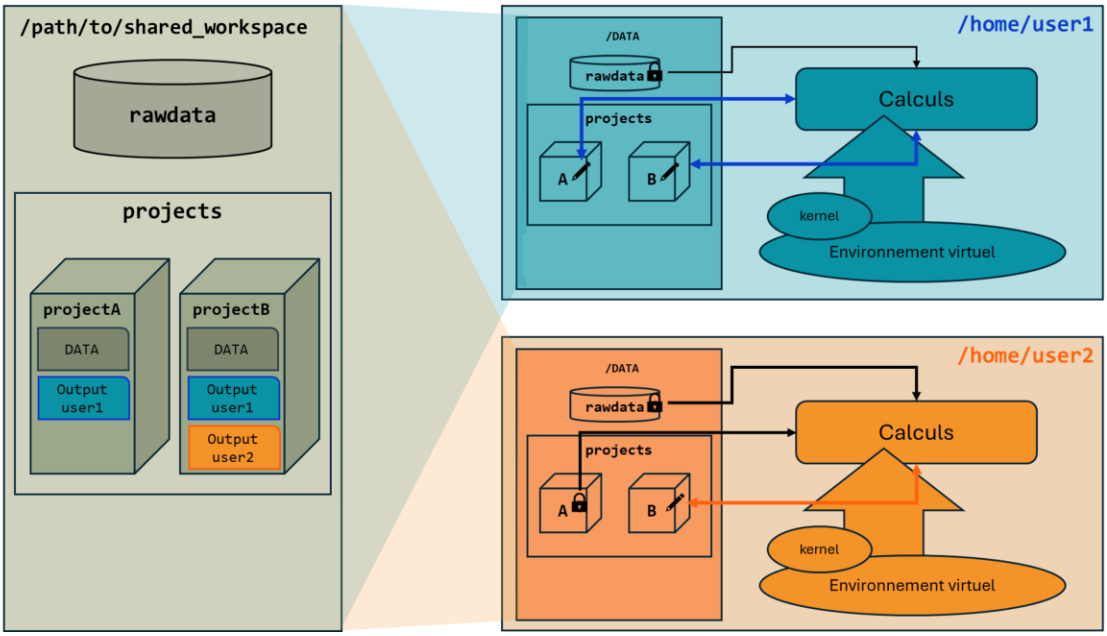
In the file explorer, navigate to `envirospace_IE` and open a terminal from the Launcher. Then, run the following command:

```
ln -s /path/to/shared_workspace shared_env
```

This creates a shortcut (or symbolic link) in your personal space to `shared_workspace`: a folder named `shared_env` now appears at the root of your personal workspace. You can interact with it just like any other personal folder.

The `shared_env` folder is not physically located in your personal space. By creating a symbolic link named `envirospace`, you have created a shortcut to the shared environment within your personal workspace. When you access the symbolic link (`/home/username/shared_env`), the system automatically redirects you to the target folder, i.e., `/path/to/shared_workspace`. As a result, the contents of `shared_env` are exactly the same as those of `shared_workspace`. If you create a file in `shared_env`, it is actually created in `shared_workspace`. Two users can each have full access to the shared environment and its data, but depending on their permissions, they may only be able to write to certain folders.

For example, in the image below, `user1` has write access to the project folders `projectA` and `projectB`. They can retrieve data from the `rawdata` folder, perform calculations in their personal space (`/home/user1`) using their own virtual environments and kernels, and then save the results in `projectA` and `projectB`. In contrast, `user2` only has write access to the `projectB` folder. However, they can still access the results saved by `user1` in `projectA` and use them for their own calculations.



If you want to remove the symbolic link, just run the following command in the terminal

```
rm shared_env
```

Note: This will not delet the folder for other users.

Step 2 : run scripts

Indicators

Now that the workspace is ready, we can begin running the indicator and prioritization scripts.

Navigate to the `run` folder located in `envirospace_IE`. This folder contains the R scripts used to calculate the ecological infrastructure indicators. Using a single programming language and a common structure for all scripts simplifies the process (one script per indicator) and makes it easier to understand what has been done for each one. The script names follow a standardized format: `ind_<pillar>_<indicator>.r`. For the following explanations, we will use the script `ind_es_pollination.r`.

The `tools.r` scripts are used to preprocess the data (if needed) before using it in the indicator scripts.

Finally, `template.r` allows you to write a new script following the same structure as the others, helping maintain a consistent format.

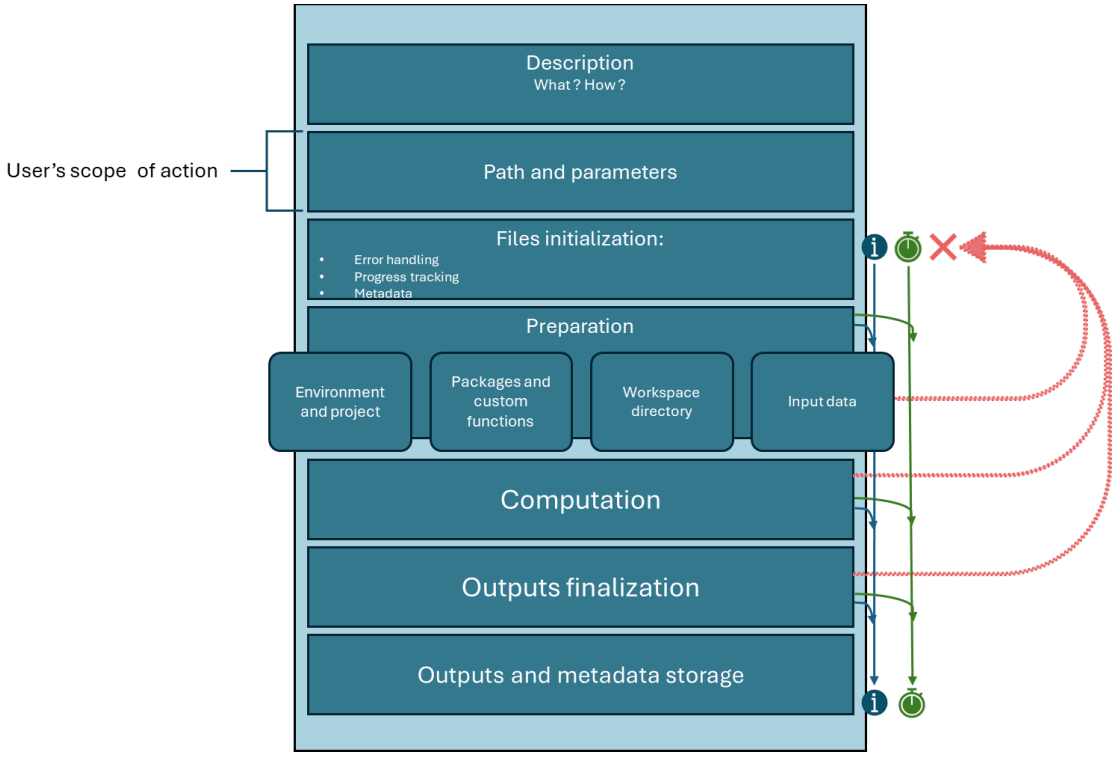
Scripts general structure

It is important to read the script thoroughly before running it for the first time, in order to understand the elements involved.

Open it and read it in full. The R scripts in the *run* folder all follow the same structure and are designed to:

- Minimize the number of actions required from the user.
- Facilitate understanding of the method through clear naming conventions and commented lines.
- Make the chosen methodology transparent.
- Automatically generate tracking files for calculations and documentation of results.
- Organize folders and outputs in a coherent manner.

Schematically, they are structured as follows:



The first part of the script provides general information about the calculation process, as well as the kernel required for proper script execution.

The user interacts with the second block, where they can modify the input data and adjust parameters specific to the chosen method.

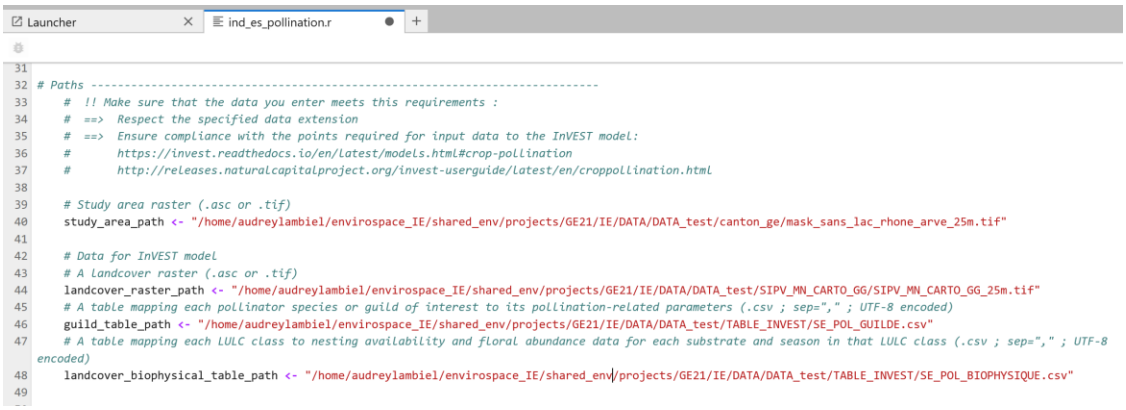
Running the script automatically generates metadata, as well as folders where intermediate and final results are saved, ensuring that the overall structure within the project folder is maintained. Each step is monitored in terms of execution time, and in case of an error, the returned message is logged in a document to facilitate debugging.

Modify scripts

The paths to the input data required for calculating the indicator must be listed in the `#Path` section of the script. In our example, the following are needed:

- A mask raster covering the study area
- A land cover raster
- A table with the ecological preferences of the considered guilds/pollinators
- A biophysical table

Enter the appropriate file paths. To do this, you can browse your file explorer to locate the data, then right-click on the file and select `Copy Path`. Then, paste it directly into the script.



In the `#Parameters` section, the `#Project` subsection allows you to modify certain project-related parameters, including the location of the shared environment, the name of the project (or in other words, the version you are producing), the indicator you are working on, and the pillar to which it belongs. Some parameters may not necessarily need to be changed.

```
# Parameters ----------
    # Project
    # Name of the main shared project folder
    shared_directory <- "/home/audreylambiel/envirospace_IE/shared_env/projects/GE21/IE"
    # Specify the name of an existing project or choose your new project's name
    # Please note that if you enter an existing project name, previously calculated results for this indicator may be overwritten.
    project_name <- "test2025"
    # Name of the pillar
    pillar_name <- "ES"
    # Name of the indicator
    indicator_name <- "POLLINATION"
    # Give a short descrition of the indicator
    description <- paste0("Capacity of ecosystems to support pollinators (InVEST model).", "\n",
                          "http://releases.naturalcapitalproject.org/invest-userguide/latest/en/croppollination.html")
```

The `#Datas and computing parameters` section allows you to specify the parameters specific to the applied method (which is why it's important to read the entire script beforehand).

```
# Datas and computing parameters
# CRS of your projet, e.g. to which your data are
mycrs <- "epsg:2056"
# Pattern used to select which of the outputs of the InVEST model should be aggregated to obtain the final result.
# See here: http://releases.naturalcapitalproject.org/invest-userguide/latest/en/croppollination.html#interpreting-results
pattern_invest_output <- "total_pollinator_abundance_"
# Function to be applied to InVEST outputs to aggregate them. Not used if only one output of the InVEST model is retained.
aggregation_function <- "sum"
```

Finally, the `#Clean up options` section allows you to manage the contents of the `scratch` folder, the folder where intermediate results are stored during the process, and the progress tracking file generated at the end of the calculation.

Set both options to `NO` so that you can explore them afterwards.

```
# Clean up options
# Do you want to delete the contents of the "scratch" folder at the end of the calculation? 'YES' or 'NO'
scratch_to_trash <- "NO"
# Do you want to delete the progress and error files generated while the script is running when it finishes? 'YES' or 'NO'
# n.b.: If "YES" but an error occurs, the two files will not be deleted in all cases to allow debugging.
track_to_trash <- "NO"
```

Note: Additional options may be available depending on the script. For example, in the case of indicators from the structure pillar, an option to aggregate the resulting raster to another resolution is provided. This depends on the specific script.

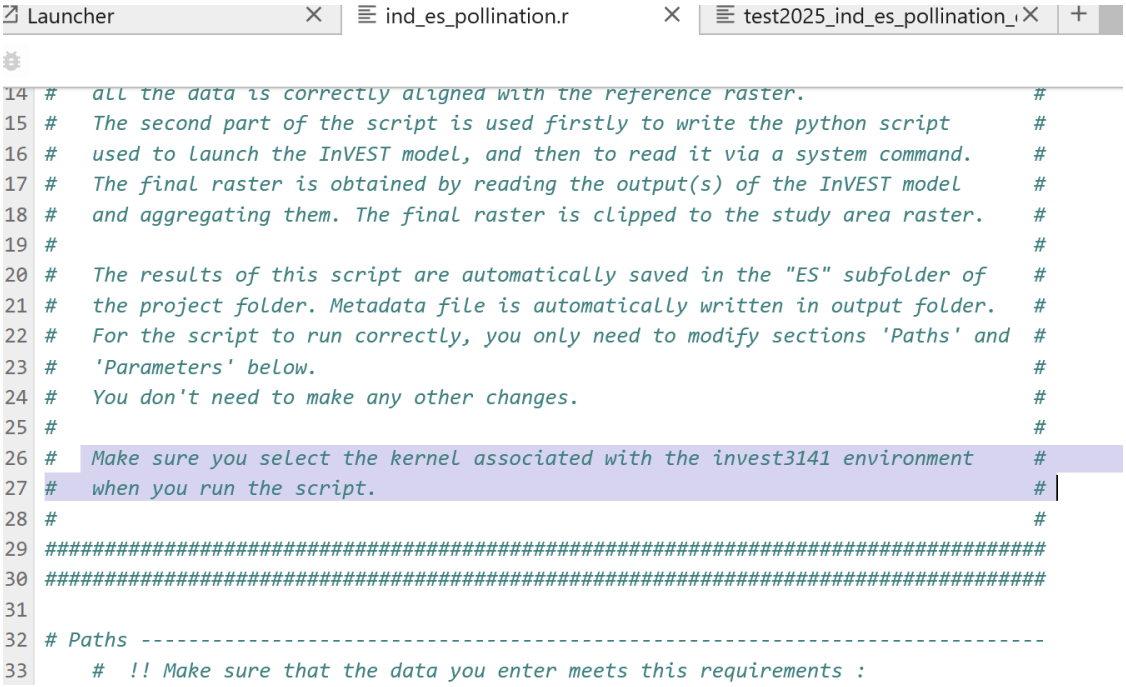
The rest of the script is not meant to be modified, except in cases of major changes to the method or when debugging is required.

Run script to compute indicator

To run the script, you must first create a console associated with the appropriate kernel. The required kernel for running the script is indicated at the end of the description.

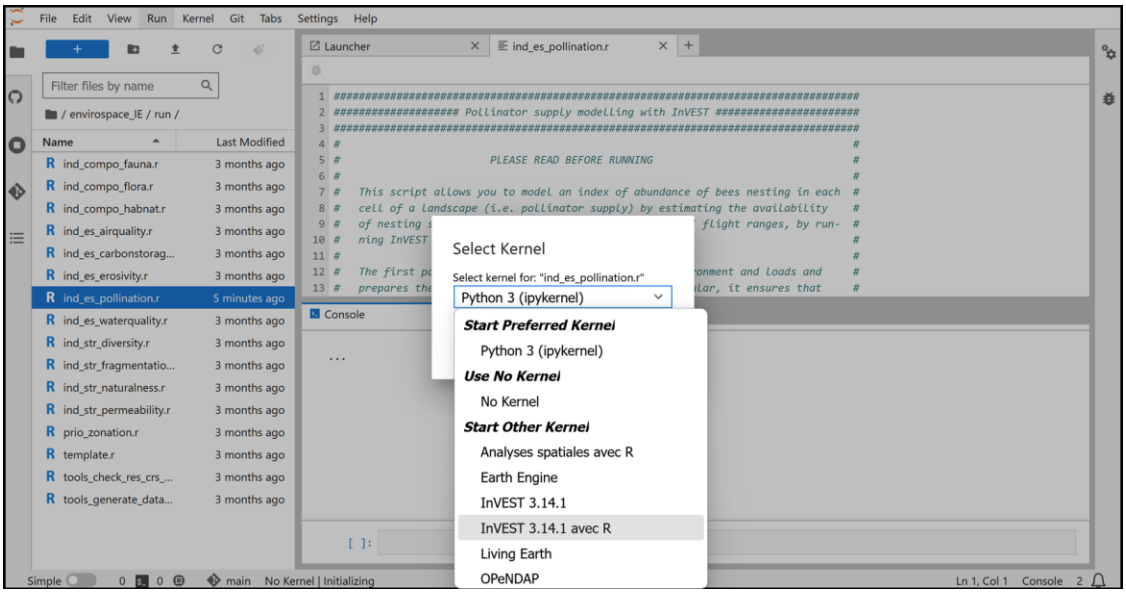
In this case, the script is based on the InVEST "Crop Pollination" model, a Python model that we will execute through R. Therefore, we need to use the `invest3141r` kernel, which is linked to the `invest3141` environment.

This information is always provided in the script description.

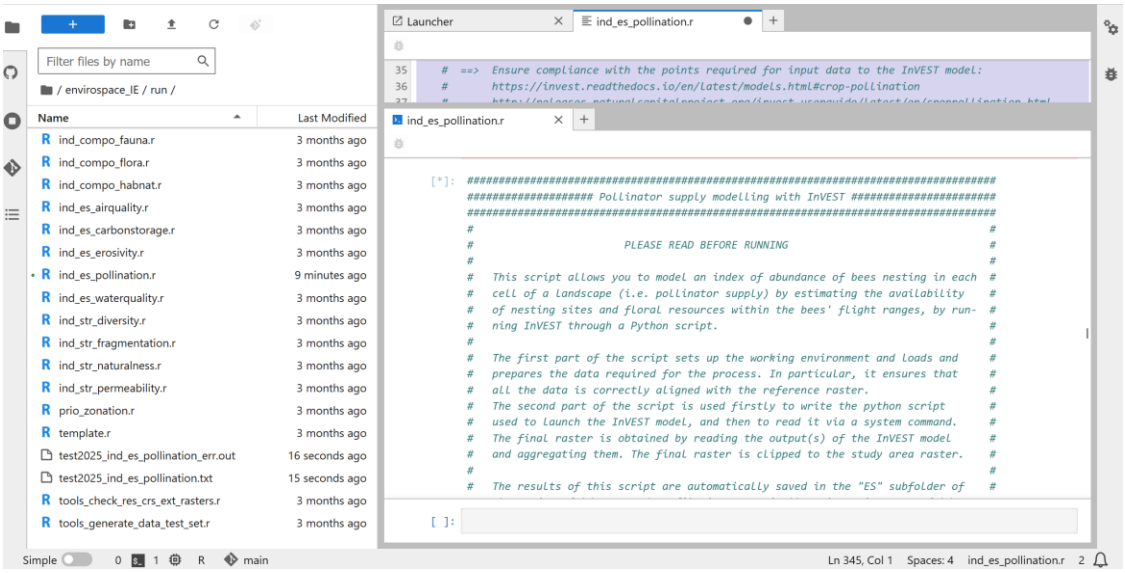


Right-click on the script and select `Create Console for Editor`, then choose the `Invest 3.14.1 with R` kernel.

To launch the indicator calculation, click on the script again and select all lines using `Ctrl+A`. Then, run the code using `Shift+Enter`.



While `[*]` is still presents, script is running. Once completed, the cell turn into `[1]`.



It is possible to run multiple calculations simultaneously without them interfering with each other. To do so, follow the same steps: read the script, update the file paths and parameters, open a console, and execute the script.

Just keep in mind that calculation speed partly depends on server resources. If you launch several processes at once, the computation time will likely increase.

You can also start scripts and then close the server page, do other activities, or even shut down your computer, the scripts will continue running as long as you don't stop the associated consoles/kernels.

Errors handling

You may have noticed that two new files appeared in your file explorer:

- The error management file `test2025_ind_es_pollination_err.out`, which should remain empty if everything runs smoothly.
- The progress tracking file `test2025_ind_es_pollination.txt`.

If an error occurs in the script or during its execution, a failure message will appear in `test2025_ind_es_pollination.txt`, while the actual error message will be recorded in `test2025_ind_es_pollination_err.out`.

You can confirm that the process has successfully completed when the word `COMPLETED` appears in the progress tracking file, or when both files disappear, if you have set `track_to_trash` to `YES`.

Prioritization

Once you have generated all the indicators needed for prioritization, you can run the prioritization process to rank all pixels in your study area and identify your ecological infrastructure.

To better understand how Zonation works, we recommend reading the associated documentation.

The steps are similar to those described in the previous **Indicators** section.

Open the `prio_zonation.r` script located in the *run* folder, and start by making sure the path to the Zonation executable is correctly set in your `zonation_folder`.

```
z5_exe <- "path/to/zonation_folder/squashfs-root/usr/bin/z5"
```

Next, adjust the input parameters according to the indicator layers you want to use (for example, those generated in the previous step or others from later versions), including weights, groups, and any masks. Also, make sure all file paths are correct.

Once everything is ready, you can run the script in the same way as the indicator scripts.

Outputs and metadata

An `OUTPUTS` folder is automatically created in the directory you specified as the shared folder in the `#parameters` section (in our example, this is `shared_env`), and it is always organized in the same way.

```
<shared_env>/              # depending on the folder you enter in the script
└── OUTPUTS/
    ├── INDICATORS/
    │   └── <pillar>/ # either "STRUCTURE", "ES", 'FUNCTION' or "COMPOSITION"
    │       └── <indicator>/     # one for each indicator that forms a pillar
    │           └── <version>/           # one sub-folder per indicator version
    │               ├── <result.tif>     # final layer(s)
    │               ├── METADATA.txt     # summary of processing and parameters
    │               └── scratch/         # optional: intermediate results
    └── PRIORITIZATION/
        └── <version>/             # one sub-folder per prioritization version
            ├── zonation_output/
            │   ├── rankmap.tif        # final prioritization output raster
            │   └── METADATA.txt       # summary of processing and parameters
            └── zonation_settings/
                ├── settings.z5        # Zonation settings file
                ├── all_files.txt # list of input rasters with weights/groups
                └── weight_group.txt   # optional: group weights file
```

Go to the `OUTPUTS` folder and navigate to the folder containing the result for the indicator `ind_es_pollination`. You will find a text file named `METADATA.txt`. Click on it to open.

This file is intended to centralize key information to help understand how the indicator was produced, when, and by whom. While the specific content may vary from one indicator/script to another, the overall structure remains consistent:

- A first section describing the characteristics of the version/project in which the indicator was calculated (project/version name, date, user who executed the code, indicator)
- A list of input data used
- The parameters applied / information about the methodology followed
- Details about the final results (name, save location, resolution, CRS, etc.)
- The duration of each step (typically input data reading and preparation, calculations, and finalization), as well as the total time (at the end of the document)

Ressources

Glossary

Console: An interactive environment that allows you to execute code commands sequentially (enter code and get results immediately).

Conda: A package manager and virtual environment management system. It is used to install, run, and update packages and their dependencies, as well as to efficiently create and manage virtual environments.

Virtual Environment: An isolated space where specific dependencies and packages can be installed for a project without interfering with other projects or system-wide installations. This allows for managing different versions of packages and libraries across projects.

Shared Space: A collaborative workspace where multiple users can access, modify, and manage files and resources together. This space is designed to facilitate teamwork and collaboration on shared projects.

Personal Space: A private workspace where a user can store and manage their files, notebooks, and other resources without access from other users.

JupyterLab: An interactive and flexible user interface for working with notebooks, code, and data. It is the successor to the classic Jupyter Notebook interface, offering a more integrated and feature-rich experience.

Kernel: The computation engine in JupyterLab that runs code in the background. It can be based on different programming languages (such as Python, R, Julia, etc.). The kernel executes the code and returns the results.

Programming Language (e.g., Python, R, Julia, ...): A set of rules and syntax used by developers to write scripts, programs, and applications. In JupyterLab, various programming languages may be available.

Symbolic Link: A shortcut or reference to another file or folder in the file system. A symbolic link allows access to the target file or folder from multiple locations without duplicating the data.

Local Machine: The user's personal computer, where JupyterLab can be installed and run. Unlike a server, the local machine is directly controlled by the user and resources are available locally.

Notebook: An interactive document used in JupyterLab that combines code, text, visualizations, and rich media in a single file.

Package (or Module): Collections of code files that add specific functionalities to a program or application. They may include libraries, tools, scripts, and data files needed to perform certain tasks in a programming language.

Server: A physical or virtual machine that hosts the JupyterLab instance and allows users to access it via a web browser. The server handles user requests, runs kernels, and manages files and resources.

Terminal: A command-line interface that allows direct execution of operating system commands.

Some useful command

Managing environment with Conda

Create environment

```
conda create -n myenv
```

Create environment with a .yml file

```
conda env create -f myenv.yml
```

Activate an environment

```
conda activate myenv
```

Deactivate environment

```
conda deactivate
```

Get a list of existing environments

```
conda info --envs
```

Remove environment

```
conda env remove --name myenv
```

Export environment as a .yml file

```
conda env export > myenv.yml
```

Clone environment

```
conda create --name myclone --clone myenv
```

Managing packages

Get the list of packages

```
conda list
```

Install a package

```
conda install package_name
```

Install a package using a specific chanel

```
conda install -c conda-forge package_name
```

Update package

```
conda update package_name
```

Remove package

```
conda remove package_name
```

Kernels Jupyter

Get the list of existing kernels

```
jupyter kernelspec list
```

Create an R kernel

```
IRkernel::installspec(name = 'nom_kernel', displayname = 'Nom affiché')
```

Create a Python kernel

```
python -m ipykernel install --user --name nom_kernel -- dispay-name "Nom
affiché"
```

Remove kernel

```
juypter kernelspec uninstall nom_kernel
```

Managing folder with terminal

Remove a non-empty folder

```
rm -r nom_dossier
```

Zip a folder

```
shutil.make_archive("chemin/nom_zip", "zip", "chemin/dossier")
```

Unzip a folder

```
shutil.unpack_archive("chemin/zip", "chemin/cible", "zip")
```

References and useful links

Reference article

Lambiel, Audrey, Gregory Giuliani, Nathan Külling, and Anthony Lehmann. In prep. "A Digital Twin approach for the identification and update of Ecological Infrastructure".

Project's GitHub repository

https://github.com/ALambiel/envirospace_IE

Related scientific articles

- Honeck, Erica, Louise Gallagher, Bertrand von Arx, et al. 2021. "Integrating Ecosystem Services into Policymaking – A Case Study on the Use of Boundary Organizations." *Ecosystem Services* 49 (June): 101286. https://doi.org/10.1016/j.ecoser.2021.101286.

- Honeck, Erica, Atte Moilanen, Benjamin Guinaudeau, et al. 2020. "Implementing Green Infrastructure for the Spatial Planning of Peri-Urban Areas in Geneva, Switzerland." *Sustainability* 12 (4): 4. https://doi.org/10.3390/su12041387.

- Honeck, Erica, Arthur Sanguet, Martin A. Schlaepfer, Nicolas Wyler, and Anthony Lehmann. 2020. "Methods for Identifying Green Infrastructure." *SN Applied Sciences* 2 (11): 1916. https://doi.org/10.1007/s42452-020-03575-4.

- Moilanen, Atte, Pauli Lehtinen, Ilmari Kohonen, Joel Jalkanen, Elina A. Virtanen, and Heini Kujala. 2022. "Novel Methods for Spatial Prioritization with Applications in Conservation, Land Use Planning and Ecological Impact Avoidance." *Methods in Ecology and Evolution* 13 (5): 1062–72. https://doi.org/10.1111/2041-210X.13819.

- Moilanen, Atte, Kerrie A. Wilson, and Hugh P. Possingham, eds. 2009. *Spatial Conservation Prioritization: Quantitative Methods and Computational Tools*. Oxford University Press. https://doi.org/10.1093/oso/9780199547760.001.0001.

- Sanguet, Arthur, Nicolas Wyler, Benjamin Guinaudeau, Noé Waller, Loreto Urbina, Laurent Huber, Claude Fischer, and Anthony Lehmann. 2023. "Mapping Ecological Infrastructure in a Cross-Border Regional Context." *Land* 12 (11): 2010. https://doi.org/10.3390/land12112010.

- Urbina, Loreto, Anthony Lehmann, Laurent Huber, and Claude Fischer. 2024. *"Combining multi-species connectivity modelling with expert knowledge to inform the green infrastructure design." Journal for Nature Conservation*, 81: 126654. https://doi.org/https://doi.org/10.1016/j.jnc.2024.126654

Some technical documentation

JupyterLab Documentation - Contains detailed information on installing, using, and customizing JupyterLab.

Conda User Guide - Tasks - Describes various tasks you can perform with Conda, such as managing environments and packages.

Conda Cheatsheet - Provides a concise summary of basic commands for managing environments, installing packages, and importing/exporting environments with Conda.

Installation de kernels Jupyter pour divers langages - Guide for installing Jupyter Kernels for Java, Scala, Go, Rust, and R.

Gestion des fichiers et répertoires avec Bash - Covers essential Bash commands for handling files and directories.