Biodiversity Informatics, 20, 2026, pp. 16-31

BIODUMPY: A COMPREHENSIVE BIOLOGICAL DATA DOWNLOADER

Tommaso Cancellario[*,†], Tomás Golomb Durán[†], Antoni Josep Far, Alejandro Roldán, and Maria Capa

Centre Balear de Biodiversitat, Universitat de les Illes Balears, Palma, Spain

Abstract. In recent years, the expansion of public biodiversity platforms and associated datasets has greatly improved access to ecological and biological information. These resources now cover vast geographic areas, extended temporal scales, and diverse taxonomic groups, becoming essential for ecological studies by enabling more comprehensive analyses and novel hypotheses testing. Concurrently, the development of programming packages has facilitated data access and interaction, optimizing their retrieval processes. However, most existing tools are limited to specific databases, posing challenges for studies requiring smooth data integration from multiple sources. The growing availability of biodiversity records highlight the urgent need for robust tools to efficiently download, process, and analyse, ecological and biological information. To address this limitation, we introduce *biodumpy*, a new Python package developed for the retrieval, management, and integration of biological data from several public databases. *biodumpy* provides access to up-to-date and comprehensive datasets spanning genetic, geographic, taxonomic, and bibliographic sources. It includes specialized modules for efficient data retrieval across taxonomic lists, with the possibility to process multiple modules simultaneously. By integrating diverse data sources, *biodumpy* enhances data acquisition, providing researchers with a powerful framework for comprehensive analyses and supporting ecological research to tackle complex environmental challenges.

Key words.—biodiversity, data mining, data processing, ecology, genetics, python, taxonomy

Introduction

Ecology serves as a bridge that integrates diverse scientific disciplines and its interdisciplinary nature is essential for understanding the complex interactions between organisms and their environments, as well as the relationships among biological entities (Goring et al., 2014; Odum, 1971). A deep comprehension of the ecological dynamics is crucial not only for advancing ecological theories but also for addressing global pressures such as climate change, habitat degradation, and biodiversity loss (Banks-Leite et al., 2020). Addressing these challenges require the development of robust strategies to catalogue, assess, and predict biodiversity variations along with their associated ecosystem services. A key step toward achieving this goal is the collection and sharing of diverse ecological data that span fine to broad spatial and temporal scales, encompass multiple biological levels (e.g., from populations to communities), and include various data types (e.g., species occurrences, genetic information).

In recent years, the scientific community has witnessed a remarkable expansion of public biodiversity platforms and related datasets (Heberling et al., 2021). These digital infrastructures serve as extensive repositories, often containing highly structured biodiversity data encompassing diverse issues, including taxonomy (e.g., Catalogue of Life; Bánki et al., 2025), species occurrences (e.g., Global Biodiversity Information Facility; GBIF.org, 2025), and genetic information (e.g., Barcode of Life Data System; Ratnasingham et al., 2024). Nowadays, such digital resources are indispensable, and many ecological studies use them to assess comprehensive analyses that were previously unfeasible due to limited data availability, poor quality, and low variability. Alongside the growth of these biodiversity platforms, there has also been a notable rise in programming packages, particularly for the R language, designed to connect directly with these databases, simplifying the data retrieval for users. For example, R packages as *rentrez* (Winter, 2017) and *BOLDconnectR* (Padhye et al., 2025) have been created for downloading genetic data. The former provides functions to interact with the National Center for Biotechnology Information API, allowing users to search and download data from databases such as

[*]*Corresponding author:* t.cancellario@uib.eu, tommaso.cancellario@gmail.com.
[†]*Equally shared first authorship.*

GenBank and PubMed. Whereas the latter enables the retrieval of information such as taxonomy, occurrences, and DNA barcode sequences from the Barcode of Life Data Systems database.

Along the same lines, to obtain ecological and biological information exist informatic tools such as *rgbif* (Chamberlain et al., 2025) and *RBIEN* (Maitner et al., 2018) that allows the users to look for and obtain information about taxonomy, distribution, trait and other relevant biological data. All these packages are useful when users need to connect to specific data sources. However, more recently, R packages such as *spocc* (Owens et al., 2025) have been developed to download and integrate data from multiple databases, generating unified and comprehensive outputs accelerating record acquisition for researchers. Despite these advances, and notwithstanding efforts to develop increasingly wide-ranging tools, existing packages often still target a limited number of databases or focus on specific tasks (e.g., downloading occurrence records). Consequently, only their combined use can adequately address the multidisciplinary nature of the ecological research. The development of a toolbox capable of integrating data from several biodiversity databases, embracing different research fields, would significantly enhance data acquisition, providing researchers with a robust foundation for comprehensive and interdisciplinary analyses, facilitating the investigation of complex environmental challenges.

Here, we introduce *biodumpy,* a new multidisciplinary Python package designed to simplify the process of retrieving diverse biological information from several public databases. To demonstrate its capabilities and support users in getting started, in this work, we also provide simplified examples that highlight the package's potential. With *biodumpy*, users can easily download, manage, and integrate data from different sources, ensuring access to the most up-to-date and comprehensive biological information available. The package includes specialized modules designed to efficiently download data for a list of given taxa, with the option to process multiple modules simultaneously. These modules provide the access to different types of biological information, including genetic, taxonomic, geographic, and bibliographic databases. Moreover, *biodumpy* can be used also within the R environment through the *reticulate* package (Ushey et al., 2024), enabling R users to integrate its functionality into existing R-based workflows. In this framework, *biodumpy* will significantly enhance data acquisition by integrating diverse datasets, providing researchers with a robust foundation for comprehensive analyses.

Software Tool Description

The *biodumpy* package is developed in Python programming language (Van Rossum & Drake 2009), and its stable version (v.0.1.9) can be installed from the official repository of Python PyPI using the command: `pip install biodumpy`. For those interested in the development version (v.0.1.19) the package is available from TestPyPI with the following command: `pip install biodumpy -i https://test.pypi.org/simple/ --extra-index-url https://pypi.org/simple/`. *biodumpy* can also be used in R (R Core Team, 2025) through the *reticulate* package, which allows seamless integration of Python code within the R environment.

Starting with biodumpy

The use of *biodumpy* is intuitive, with a general structure that is consistent across all modules. Below we describe an overview of the main steps to get started with the package:

1) Load the package into your Python environment.
2) Import one or more specific modules needed to retrieve the data.
3) Define a list of taxa (except for the Crossref module, which accepts a list of DOIs).
4) Configure the *biodumpy* module/s with the required parameters and start the download.

Although each module of *biodumpy* is characterized by specific parameters, three parameters are common across all modules: `bulk, output_format, and sleep`.

`bulk` is a Boolean parameter allowing users to customize the data structure according to their needs during the saving process (Fig. 1). When `bulk` is *True*, the information downloaded for each taxon is stored into a single file. This configuration may be useful if the amount of the total data is limited, for consolidating data, and simplifying file management. Whereas, if `bulk` is *False*, the information for each taxon is saved in a separate file each one named as the taxon. This option is useful for detailed analysis, particularly when individual taxon-files are required or when the data for each taxon is extensive.

By default, *biodumpy* saves the resulting file in a folder named "downloads" within the user's working directory. Inside this folder, a subfolder is automatically created, named after the current date and, within it, additional subfolders corresponding to each of the modules used are generated.

A – `bulk` = ***True***

```
Root_Folder
└── downloads
    └── YYYY-MM-DD
        ├── Module_1
        │   ├── bulk
        ├── Module_2
        │   └── ...
        ├── Module_3
        │   └── ...
        └── Module_n
            └── ...
```

B – `bulk` = ***False***

```
Root_Folder
└── downloads
    └── YYYY-MM-DD
        ├── Module_1
        │   ├── Taxon_1
        │   ├── Taxon_2
        │   ├── Taxon_3
        │   └── Taxon_n
        ├── Module_2
        │   └── ...
        ├── Module_3
        │   └── ...
        └── Module_n
            └── ...
```

Figure 1. Example illustrating the folder hierarchy structure, with the distinction based on the `bulk` parameter being set to either *True* (A) or *False* (B).

Depending on the module, *biodumpy* provides the option to store files in two output formats: JSON or FASTA. Although the JSON format is not commonly used in biological or ecological fields, we have deliberately chosen it as default format due to its flexibility to support nested and complex data structures, such as arrays and key-value pairs. This makes JSON particularly well-suited for representing hierarchical or relational data, such as taxonomy and observations or nested genetic information, within a single file. Additionally, JSON integrates easily with the raw data returned by the Application Programming Interfaces (APIs) used to develop the *biodumpy* modules and can be easily converted into a tabular format (e.g., .csv).

On the other hand, the FASTA format is specifically designed for storing nucleotide or protein sequences. The BOLD and NCBI modules offer the option to download data in FASTA format, making them ideal to retrieve data to perform sequence-based analysis.

Finally, the `sleep` parameter is useful for controlling the delay between successive data retrieval requests. It enables users to regulate data download speed, thereby preventing server overloads and avoiding rate limit violations. By default, `sleep` is set to 3 seconds. However, we encourage users to adjust this parameter according to the API's rate policies to ensure responsible data access and avoid overwhelming external servers.

Biodumpy modules

The current stable version of *biodumpy* (v.0.1.9) includes 10 modules that cater to different aspects of biodiversity data management, providing functionalities for working with a wide range of data types. Below, we describe the main functionalities and parameters for each module. Detailed information can be found in the *biodumpy* manual[1].

BOLD.—The *BOLD* module allows users to retrieve information from the Barcode of Life Data System (Ratnasingham et al., 2024). To select the output format users can simply set the parameter `output_format` to *'json'* or *'fasta'*. Additionally, this module provides the option to download a summarized version of the data by setting the `summary` parameter to *True*, offering a more concise and manageable dataset (*Supplementary Material 1 - S1*).

COL.—The *COL* module allows users to easily retrieve taxonomic nomenclature from the Catalogue of Life (Bánki et al., 2025). A key parameter for this module is the `dataset_key`, which allows the user to select the Catalogue of Life version to use for the search. This information can be found at the bottom of the Catalogue of Life webpage (https://www.catalogueoflife.org/) under the ChecklistBank label. Additionally, the module allows users to verify taxon synonyms by setting the parameter `check_syn` to *True*. When this parameter is enabled, only the accepted name is stored in the classification section of the JSON file. Conversely, both the synonym and the accepted name are included in the classification section (*Supplementary Material 1 - S2*).

Crossref.—The *Crossref* module allows users to retrieve scientific bibliographic metadata from Crossref

[1] https://biodumpy.readthedocs.io/en/latest/.

database. Unlike other modules that use a list of taxa as an input parameter, this module requires a list of DOIs to download the available bibliographic metadata. Users can also obtain a summarized version of the downloaded data by setting the `summary` parameter to *True* (*Supplementary Material 1 - S3*).

GBIF.—The *GBIF* module enables users to access taxonomic and geographic data from GBIF (GBIF.org, 2025). Taxonomic information can be downloaded using the default parameters, while occurrence information can be obtained by setting the parameter `occ` to *True*. The module supports filtering occurrences based on a specified geographic region using the geometry parameter. When a geometry is provided, only occurrences that fall within the defined polygon are included in the outcome. Finally, this module allows the user to check for possible nomenclature synonyms using the parameter `accepted_only`. If this parameter is *False*, nomenclature includes possible synonyms; conversely, the information retrieved corresponds only to accepted names (*Supplementary Material 1 - S4*).

INaturalist.—The *INaturalist* module permit users to retrieve taxon photo metadata from iNaturalist (iNaturalist, 2025). Notably, the module does not directly download photos from iNaturalist; instead, it retrieves the metadata required to access them (e.g., 34826202/medium.jpg – photo id and its size). To retrieve a photo, the photo-ID (e.g., 34826202) and the image size (e.g., medium) must be concatenated with the following base URL: https://inaturalist-open-data.s3.amazonaws.com/photos/ (e.g., https://inaturalist-open-data.s3.amazonaws.com/photos/34826202/medium.jpg).

To avoid copyright issues, the module retrieves photos classified under public licenses, including: CC0, CC BY, CC BY-NC, CC BY-NC-ND, CC BY-SA, CC BY-ND, and CC BY-NC-SA (*Supplementary Material 1 - S5*).

IUCN.—The *IUCN* module enables users to efficiently retrieve information on species assessments, habitats, and threats from the IUCN Red List (IUCN, 2025) by specifying a species name and providing a personal API key. Moreover, the module allows users to retrieve data for a specific list of species from one or more areas by specifying a list of regions in the `scope` parameter (e.g., scope = *['Global', 'Europe']*; *Supplementary Material 1 - S6*).

NCBI.—The *NCBI* module enables users to efficiently retrieve data from the National Center for Biotechnology Information database (NCBI, 2025). To obtain data in FASTA format, users must set the rettype and `output_format` parameters to *'fasta'*. By providing a list of taxa, users can refine their searches using the `query_type` parameter, which allows for the combination of multiple search criteria to better target the desired data. For example, to download sequences related to the cytochrome c oxidase subunit I, users can set the `query_type` to *'[Organism] AND ("CO1"[GENE] OR "COI"[GENE] OR "COX1"[GENE] OR "COXI"[GENE])'* (Porter & Hajibabaei, 2018).

Module performance can be further optimized using the `step_id` and `step_seq` parameters, which control the chunk size for downloading IDs and sequences, respectively. While relatively large chunk sizes are manageable for ID downloads (e.g., step_id = *500*), setting a high value for `step_seq` can lead to memory issues due to the potentially large size of the sequence data. To avoid such problems, we recommend using moderate values for the `step_seq` parameter (e.g., `step_seq` = *100*). Lastly, users can retrieve a summary of metadata associated with the retrieved data by setting the `summary` parameter to *True* (*Supplementary Material 1 - S7*).

OBIS.—The *OBIS* module facilitates the retrieval of data from the Ocean Biodiversity Information System (OBIS, 2025). Users can download taxonomic information from OBIS and, like the *GBIF* module, this module enables users to download species occurrences by setting the `occ` parameter to *True*. Moreover, the module supports filtering occurrences based on specific geographic regions using the `geometry` or `areaid` parameters. These spatial parameters can be applied independently or in combination. When a `geometry` or `areaid` is specified, the results include only occurrences that fall within the defined polygons (*Supplementary Material 1 - S8*).

WORMS.—The *WORMS* module enables users to easily retrieve nomenclature information from the World Register of Marine Species database (WoRMS Editorial Board, 2016). Users can set the parameter `marine_only` to *True* to restrict the search only for species belonging to the marine environment. Moreover, it is possible to retrieve the distribution data for the taxa setting the parameter `distribution` to *True* (*Supplementary Material 1 - S9*).

ZooBank.—The *ZooBank* module allows users to easily retrieve scientific bibliographic information about taxa from the Official Registry of Zoological Nomenclature database (International Commission on Zoological Nomenclature, 2025) by providing a taxa name. To optimize the performance of the module, this function includes the parameter `dataset_size`. This parameter can set to *'small'* or *'large'* depending on the amount of data to download. Users can also download additional information setting the parameter `info` to *True*. When this parameter is activated, the outcome includes the main information of a bibliographic resource along with additional details such as the DOI. Examples of retrieved data can be found in *Supplementary Material 1 - S10*.

Customizable modules

Although several modules have been implemented in *biodumpy*, some digital sources may remain unexplored, limiting the scope of data retrieval. To address this issue, we offer users the possibility to create their own modules, leveraging the inherent hierarchical structure of *biodumpy*. The commented code to create your own module is available in *Supplementary Material 1 - S11*.

Biodumpy tests

To further improve the quality and reliability of *biodumpy*, we incorporated a comprehensive suite of tests designed to stress all individual modules and assess their performance under various conditions. These tests are executed through a GitHub Actions workflow, which automatically runs each module whenever a new version is uploaded to the development or deployment branches. For the development branch, tests are performed using Python versions 3.10 and 3.14 on the Ubuntu operating system. For the deployment branch, the package is tested across Python versions 3.10 to 3.14 and on three operating systems: Windows, macOS, and Ubuntu. The testing procedures are conducted using *pytest* (v. 8.3.3; Krekel et al., 2004). Test summaries are available online at dev_test and deploy_test.

Dependencies of biodumpy

The *biodumpy* package relies on several Python libraries, listed below: *Biopython* (v. >=1.84; <2.0): (Cock et al., 2009); *tqdm* (v. >=4.66.4; <5.0): (da Costa-Luis et al., 2024); *requests* (v. >=2.32.4; <3.0): (Reitz, 2024); *beautifulsoup4* (v. >=4.12.3; <5.0): (Richardson, 2024); *pytest* (v. 8.3.2): (Krekel et al., 2004).

Example Applications

To demonstrate the basic usage of *biodumpy*, we provide commented code of three comprehensive examples, which can be useful for introducing the potential of this package.

Briefly, in the "Example 1", we aim to work with several *biodumpy* modules and create a general pipeline to retrieve basic ecological information as taxonomy, occurrences, and bibliography from a list of taxa. In the "Example 2", we focus on genetic modules, and we illustrate how to download FASTA files and perform an alignment with MAFFT (Katoh et al., 2019). In the "Example 3", we show how to use *biodumpy* within the R environment. This can be particularly useful for users not strictly familiarized with Python. To facilitate the integration of the *biodumpy* in R, we utilize the *reticulate* package.

While the examples provided are useful for showing the potential of *biodumpy*, they do not fully capture the complexity or scope required for a comprehensive biological or ecological data management and analysis. Therefore, they should be considered as a simple guide to help the users in performing their own customized pipeline. A Python script and the downloaded data for each example are available[2].

We encourage users to open the example script provided in an Integrated Development Environment (e.g., Visual Studio Code, PyCharm, or RStudio), rather than simply copying and pasting the code below into a new script, to avoid potential issues such as indentation errors or similar problems.

Example 1 – Work with taxonomic, geographic, and bibliographic data

In this example, we aimed to create a simple pipeline to store in a single file the systematics taxonomy classification about a target species, its occurrences and an environmental variable (i.e., bathymetry) across a specified geographic area, and the bibliographic information available in ZooBank for the given taxon. To achieve this, we divided the example into five steps: i) loading the necessary packages; ii) creating a taxa list and downloading information from GBIF, OBIS, COL, and ZooBank; iii) filtering the species-specific information and creating separate datasets containing taxonomy, distribution, and ZooBank bibliographic information; iv) saving each dataset into separate Excel sheets; v) creating two simple plots one to show the distribution of occurrences and another for the environmental variable (Fig. 2). (Data accessed: December 02, 2025.)

Firstly, we import the required package for carrying out the analysis. If packages are not already installed in the Python virtual environment, they can be installed using the general command: `pip install package_name==version_number` (or see the file named "readme.md" contained in the biodumpy_Example.zip for instructions on installing all packages required to run the examples).

```
# 1) Import packages and modules #

import json
import statistics
import seaborn as sns
import matplotlib.pyplot as plt
import xlsxwriter
```

[2] https://osf.io/tn3au/overview?view_only=59718b5494d7405a965ac9d1e09252d0.

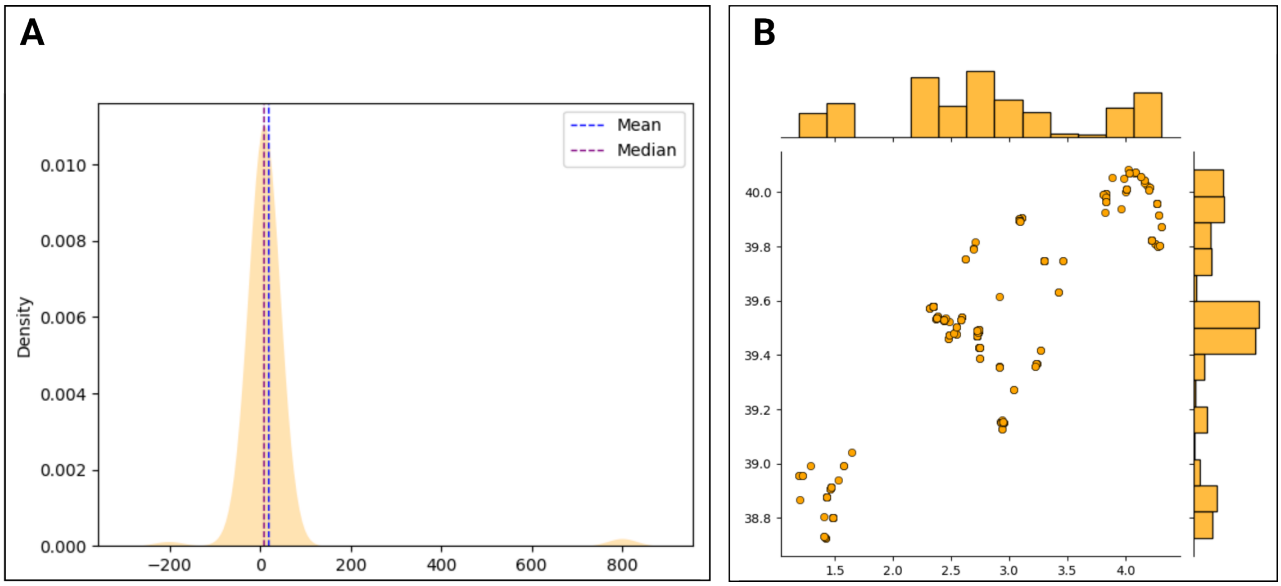

Figure 2. A) Density plot representing the distribution of depth data. The x-axis indicates the depth values, while the y-axis represents the corresponding density. A vertical dashed blue line marks the mean depth, whereas a purple line indicates the median depth. The shaded area under the curve highlights the overall density distribution. B) Joint plot illustrating the relationship between latitude and longitude variables. The central scatter plot shows individual data points, with longitude on the x-axis and latitude on the y-axis, indicating the spatial distribution of occurrences. Marginal histograms are displayed along the axes, showing the frequency distributions of longitude and latitude independently.

```
from biodumpy import Biodumpy
from biodumpy.inputs import GBIF,
OBIS, COL, ZooBank
```

After imported the packages, we define a list of taxa to start the download. In this case, we used a single taxon (e.g., *Pinna nobilis*) to expedite the analysis. However, users can include a list of taxa as desired. We utilized the GBIF, OBIS, COL, and ZooBank modules to retrieve different types of information, such as taxonomy, taxon occurrences, and bibliography. To limit the occurrence download, we defined a polygon around the Balearic Islands.

```
# 2) Create a taxa list and download
data #

taxa = ['Pinna nobilis']
polygon = ('POLYGON((1.1169
38.56042,4.41447 38.56042,4.41447
40.19769,1.1169 40.19769,1.1169
38.56042))')

bdp = Biodumpy([
    GBIF(geometry=polygon, occ=True,
accepted_only=True),
    OBIS(geometry=polygon, occ=True),
    COL(dataset_key=313100),
    ZooBank(dataset_size='small', in-
fo=True)])
bdp.start(taxa, output_path='./down-
loads_example_1/{module}/{name}')
```

After downloading the data, we can filter each JSON file and create specific objects for each data type.

```
# Extract taxonomy of Pinna nobilis
provided by COL.
f = open('downloads_example_1/COL/Pin-
na nobilis.json', 'r')
data_col = json.load(f)[0]
f.close()

# Extract taxonomic classification lev-
els.
classification = data_col['classifica-
tion']

# Create an object "taxonomy" with the
taxonomic information of the species
(i.e., id, name, and taxonomic rank).
taxonomy = []
for item in classification:
    taxonomy.append({
```

```
        'id': item['id'],
        'name': item['name'],
        'rank': item['rank']
    })

# Extract DOI.
f = open('downloads_example_1/ZooBank/
Pinna nobilis.json', 'r')
data_zoo = json.load(f)
f.close()

# Create and object "doi" with the DOI
links.
doi = [item['info'][0]['IdentifierURL']
for item in data_zoo]

# 3) Prepare occurrence data #

f = open('downloads_example_1/GBIF/
Pinna nobilis.json', 'r')
data_gbif = json.load(f)
f.close()

f = open('downloads_example_1/OBIS/
Pinna nobilis.json', 'r')
data_obis = json.load(f)
f.close()

# Filter GBIF and OBIS data and merge
into a single dataset.
data_gbif_filtered = [
    {
        'scientificName': item.get('ac-
ceptedScientificName', None),
        'decimalLatitude': item.
get('decimalLatitude', None),
        'decimalLongitude': item.
get('decimalLongitude', None),
        'depth': item.get('depth',
None),
        'source': 'GBIF'
    } for item in data_gbif
]

data_obis_filtered = [
    {
        'scientificName': item.
get('scientificName'),
        'decimalLatitude': item.
get('decimalLatitude'),
        'decimalLongitude': item.
get('decimalLongitude'),
        'depth': item.get('bathyme-
try'),
        'source': 'OBIS'
    } for item in data_obis
]

# Join the objects data_gbif_filtered
and data_obis_filtered to create a
# comprehensive object containing the
geographic data.
data = data_gbif_filtered + data_obis_
filtered

# Create an object with depth informa-
tion
depth = [item['depth'] for item in
data if item['depth']]
mean_depth = round(statistics.
mean(depth), ndigits=2)
median_depth = round(statistics.medi-
an(depth), ndigits=2)
```

We now create a tabular file (e.g., .xlsx) containing the downloaded information, organized into separate sheets.

```
# 4) Save files into an .xlmx #

workbook = xlsxwriter.Workbook('down-
loads_example_1/biodumpy_ex_1.xlsx')
worksheet = workbook.add_work-
sheet('taxonomy')
worksheet.write_row('A1', ['Rank',
'Name'])

# Write taxonomy data
for i, item in enumerate(taxonomy,
start=2):
    worksheet.write_row(f'A{i}',
[item['rank'], item['name']])

worksheet = workbook.add_work-
sheet('occurrences')

worksheet.write_row('A1', ['scientif-
icName', 'decimalLatitude', 'decimal-
Longitude', 'depth', 'source'])

# Write occurrence data
for i, item in enumerate(data,
start=2):
```

```
        worksheet.write_row(f'A{i}',
[item['scientificName'],
item['decimalLatitude'], item['dec-
imalLatitude'], item['depth'],
item['source']])

worksheet = workbook.add_work-
sheet('bibliography')
worksheet.write('A1', 'doi')

# Write bibliographic data
for i, item in enumerate(doi,
start=2):
    worksheet.write(f'A{i}', item)

# Save the workbook
workbook.close()
```

Finally, we produce two simple plots. The first is a density plot illustrating the depth data's distribution, along with its associated mean and median (Fig. 2A). The second is a joint plot showing the relationship between latitude and longitude (Fig. 2B).

```
# 5) Create density plot of depth and
another joint plot for longitude and #
latitude data #

sns.kdeplot(depth, color='orange',
fill=True, alpha=.3, linewidth=0)
plt.axvline(x=mean_depth, col-
or='blue', linestyle='--', line-
width=1, label='Mean')
plt.axvline(x=median_depth, col-
or='purple', linestyle='--', line-
width=1, label='Median')
plt.legend()
plt.show()

lon = [item['decimalLongitude'] for
item in data]
lat = [item['decimalLatitude'] for
item in data]
sns.jointplot(x=lon, y=lat,
kind='scatter', color='orange', edge-
color='black')
plt.show()
```

Example 2 – Genetics

Here we illustrate how to download FASTA files and perform an alignment with MAFFT. We divided the process in five main steps: i) loading the necessary packages; ii) creating a function to remove duplicate records; iii) downloading the FASTA files from BOLD and NCBI modules; iv) extracting only the sequences labeled as COI and merging the data into a single dataset; v) performing an alignment using MAFFT. (Data accessed: December 02, 2025.)

First, we load the packages, and the modules needed to perform the analysis.

```
# 1) Import packages #

import subprocess

from biodumpy import Biodumpy
from biodumpy.inputs import BOLD, NCBI
from biodumpy.utils import read_fasta,
save_fasta

# Install MAFFT following the instruc-
tions provided at https://mafft.cbrc.
jp/alignment/software/
```

Next, we create a Python function to remove possible duplicate records shared between the two datasets. This function has two input parameters: `dicts_list`, which specifies the name of the list containing the sequences, and `key`, which indicates the name of the field containing the values to check for duplicates. In this example, we use the NCBI accession number as the value to check.

```
# 2) Create 'remove_duplicates' func-
tion #

def remove_duplicates(dicts_list:
list, key: str):
    seen = set()
    unique_dicts = []

    for d in dicts_list:
        value = d.get(key)
        if value not in seen:
            unique_dicts.append(d)
            seen.add(value)

    return unique_dicts
```

Now, we download the sequences in FASTA format for the Odonata species *Anax imperator* using the *biodumpy* modules BOLD and NCBI.

```
# 3) Create a taxa list and start the
download #

taxa = ['Anax imperator']

# Set the modules BOLD and NCBI and
start the download.
bdp = Biodumpy([
    BOLD(bulk=True, output_for-
mat='fasta'),
    NCBI(bulk=True, rettype='fas-
ta', output_format='fasta', query_
type='[Organism]', mail='your_mail@
git.hub')])

bdp.start(taxa, output_path='down-
loads_example_2/{module}/{name}')
```

Once we download the files, we can open the two FASTA files using the function `read_fasta`, specifying the file paths of the downloaded data as parameters. Next, we extract only the sequences labeled with COI and combine the data obtained from BOLD and NCBI modules into a single dataset removing duplicated records.

```
# 4) Extract only COI Marker #

# Read the fasta files downloaded from
BOLD and NCBI respectively.
seq_bold = read_fasta('downloads_exam-
ple_2/BOLD/bulk.fasta')
seq_ncbi = read_fasta('downloads_exam-
ple_2/NCBI/bulk.fasta')

# Filtered only sequences labeled as
COI (this is a filter example)
filtered_bold = [item for item in seq_
bold if 'COI-5P' in item['id']]
filtered_ncbi = [item for item in seq_
ncbi if 'cytochrome oxidase subunit 1'
in item['id']]

# Remove shared records between BOLD
and NCBI
for item in filtered_bold:
    item['ncbi_accession'] =
item['id'].split('|')[-1] if
len(item['id'].split('|')) == 4 else
None

for item in filtered_ncbi:
    item['ncbi_accession'] =
item['id'].split('.')[0]

# Join the filtered data
sequences = remove_duplicates(filtered_
ncbi + filtered_bold, 'ncbi_accession')

# Save the final file
save_fasta('downloads_example_2/se-
quences.fasta', sequences)
```

Finally, we perform sequence alignment using MAFFT and save the resulting output. In this step, we create a command to run MAFFT, setting the alignment strategy parameter to *auto* (for more information about MAFFT parameter options and installation, please visit the official webpage[3]).

```
# 5) Alignment #

# Specify the input file for the align-
ment process and define the output file
path.

# Path to the FASTA file containing se-
quences to be aligned with MAFFT
input_file = 'downloads_example_2/se-
quences.fasta'

# Path where the aligned sequences
will be saved
output_file = 'downloads_example_2/out-
put.fasta'

# Construct the MAFFT command
# Note: If 'mafft' is not found in
PATH,
# specify the full installation path:
#   Windows: mafft_command = [r'C:\Pro-
gram Files\mafft-win\mafft.bat', '—
#   auto', input_file]
#   macOS:   mafft_command = ['/usr/lo-
cal/bin/mafft', '--auto', input_file]
#   Linux:   mafft_command = ['/usr/
bin/mafft', '--auto', input_file]
mafft_command = ['mafft', '--auto', in-
put_file]

# Run MAFFT
```

[3] https://mafft.cbrc.jp/alignment/software/.

```
with open(output_file, 'w') as output:
    subprocess.run(mafft_command, stdout=output)
```

Example 3 – biodumpy in R

In the following example, we describe the procedure for integrating *biodumpy* into the R environment. This example is run within the R environment. Therefore, the code syntax will be slightly different compared to the examples above. We divide the following code into three main steps: i) install and load the *reticulate* package; ii) create a virtual environment in R; iii) use a couple of *biodumpy* modules to download the data. (Data accessed: December 02, 2025.)

As the first step we install and load the pack *reticulate*.

```
# 1) Install and load the reticulate pack #

install.packages('reticulate', dependencies=TRUE)
library(reticulate)
```

Once the package is installed and loaded, users can set up a specific virtual environment to perform the analysis. While this step may be unfamiliar to many R users, it is important to understand its purpose. Similar to R, Python has a vast ecosystem of external packages that extend its core functionality. However, Python allows only one version of a package to be installed at a time, which can create conflicts when working on multiple projects requiring different package versions or dependencies. To address this issue, Python enables the creation of virtual environments, which allow users to manage packages and their dependencies independently for each project. By default, *reticulate* uses an isolated Python virtual environment named 'r-reticulate'.

```
# 2) Create a new environment

# Create and activate the virtual environment (macOS)
# /opt/homebrew/bin/python3.11 -m venv ./biodumpy_example
# source ./biodumpy_example/bin/activate
use_virtualenv('./biodumpy_example', required = TRUE)

# Set a specific virtualenv and install biodumpy (v 0.1.9)
virtualenv_install('./biodumpy_example', 'biodumpy==0.1.9')
# py_list_packages()
```

Once we set the virtual environment containing the *biodumpy* package it is possible to start to work with it. Similarly to the previous examples, we download occurrences and sequences from GBIF and BOLD respectively.

```
# 3) Import biodumpy and its modules

biodumpy <- import('biodumpy')
inputs <- import('biodumpy.inputs')

# Define the taxa vector
taxa <- list('Anax imperator')


# Define a polygon to limit the download from GBIF
polygon = 'POLYGON((1.1169 38.56042,4.41447 38.56042,4.41447 40.19769,1.1169 40.19769,1.1169 38.56042))'

# Set the BOLD and GBIF modules
bold <- inputs$BOLD(bulk=FALSE, output_format='fasta')
gbif <- inputs$GBIF(bulk=FALSE, occ=TRUE, geometry=polygon)

# Start the download
bdp <- biodumpy$Biodumpy(list(bold, gbif))
bdp$start(taxa, output_path = 'downloads_example_3/{module}/{name}')
```

DISCUSSION

Recent advancements in computational power, coupled with the growing application of big data in various ecological domains, have significantly improved our capacity to analyse and understand the natural environment. Here, we introduced *biodumpy*, a novel and customizable Python package developed for the retrieval of biological data from multiple public databases. Unlike existing tools, which are often tailored to interact with a single data source, *biodumpy* includes several modules that enable users to download, manage, and integrate diverse types of biological information, primarily starting from a user-defined list of taxa. With the ability to retrieve different kinds of information simultaneously, *biodumpy* facilitate the

data collection and ensures compatibility across datasets, providing a robust foundation for ecological and biological analyses.

Overall, *biodumpy* represents a significant step toward enhancing the efficiency of data acquisition in ecological research, promoting interdisciplinary studies and permitting scientists to focus more on analysis and outcome interpretation rather than on data download.
Future versions of the package will enhance flexibility for more biological data sources, integrate advanced data cleaning functions, and optimize performance for large-scale data processing.

Acknowledgments

This work has been partially funded and promoted by the Comunitat Autonoma de les Illes Balears through the Conselleria d'Educació i Universitats and by the European Union- Next Generation EU/PRTR-C17. I1 (SINCO2022/6717). Nevertheless, the views and opinions expressed are solely those of the author or authors, and do not necessarily reflect those of the Conselleria d'Educació i Universitats, the European Union or the European Commission. Therefore, none of these organizations shall be held liable. This study has been partially funded by GOIB/ Conselleria d'Educació i Universitats through the project "SINCO2022/18146" and co-funded by the European Union. Many thanks to Simone Guareschi and Enrique Arboleda for their useful comments on the final version of the manuscript.

Author Contributions

CT and GDT conceived the ideas, designed methodology, created the documentation and tested the software. CT wrote the original draft and led the writing of the manuscript. FA and RA designed methodology, tested the software, reviewed and edited the manuscript. CM reviewed and edited the manuscript. All authors contributed critically to the final version of manuscript and gave final approval for publication.

Conflict of Interest Statement

The authors have declared that no competing interests exist.

Data Accessibility

OSF (Python scripts[4]). GitHub: The code is available[5]. To report a bug or issue, please contact the corresponding author or open an issue or pull request on the GitHub project.

[4] https://osf.io/tn3au/overview?view_only=59718b5494d7405-a965ac9d1e09252d0.
[5] https://github.com/centrebalearbiodiversitat/biodumpy.

References

Bánki, O., Roskov, Y., Döring, M., Ower, G., Hernández Robles, D. R., Plata Corredor, C. A., Stjernegaard Jeppesen, T., Örn, A., Pape, T., Hobern, D., Garnett, S., Little, H., DeWalt, R. E., Miller, J., Orrell, T., Aalbu, R., Abbott, J., Abreu, C., Acero P, A., et al. (2025). *Catalogue of Life (2025-12-20 XR)*. Catalogue of Life Foundation, Amsterdam, Netherlands. https://doi.org/10.48580/dgvtg

Banks-Leite, C., Ewers, R. M., Folkard-Tapp, H. and Fraser, A. (2020). Countering the effects of habitat loss, fragmentation, and degradation through habitat restoration. *One Earth*, 3(6), 672-676. https://doi.org/10.1016/j.oneear.2020.11.016

da Costa-Luis, C., Larroque, S. K., Altendorf, K., Mary, H., richardsheridan, Korobov, M., Yorav-Raphael, N., Ivanov, I., Bargull, M., Rodrigues, N., Shawn, Dektyarev, M., Górny, M., mjstevens777, Pagel, M. D., Zugnoni, M., JC, CrazyPython, Newey, C., Lee, A., pgajdos, Todd, Malmgren, S., redbug312, Desh, O., Nechaev, N., Boyle, M., Nordlund M., MapleCCC and McCracken, J. (2024). tqdm: A fast, Extensible Progress Bar for Python and CLI (v4.67.0). Zenodo. https://doi.org/10.5281/zenodo.14047011

Chamberlain, S., Barve, V., Mcglinn, D., Oldoni, D., Desmet, P., Geffert, L. and Ram, K. (2025). rgbif: Interface to the Global Biodiversity Information Facility API. R package version 3.8.4.3. https://CRAN.R-project.org/package=rgbif.

Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedbeg, I., Hamelryck, T., Kauff, F., Wilczynski, B. and De Hoon, M. J. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11), 1422. https://doi.org/10.1093/bioinformatics/btp163

GBIF.org. (2025). *GBIF Home Page*. https://www.gbif.org

Goring, S. J., Weathers, K. C., Dodds, W. K., Soranno, P. A., Sweet, L. C., Cheruvelil, K. S., Kominoski, J. S., Rüegg, J., Thorn, A. M. and Utz, R. M. (2014). Improving the culture of interdisciplinary collaboration in ecology by expanding measures of success. *Frontiers in Ecology and the Environment*, 12(1), 39-47. https://doi.org/10.1890/120370

Heberling, J. M., Miller, J. T., Noesgaard, D., Weingart, S. B. and Schigel, D. (2021). Data integration enables global biodiversity synthesis. *Proceedings of the National Academy of Sciences USA*, 118(6), e2018093118. https://doi.org/10.1073/pnas.2018093118

iNaturalist. (2025). iNaturalist. https://www.inaturalist.org/

International Commission on Zoological Nomenclature.

(2025). *ZooBank: The Official Registry of Zoological Nomenclature*. https://www.zoobank.org

IUCN. (2025). IUCN Red List of Threatened Species. https://www.iucnredlist.org

Katoh, K., Rozewicki, J. and Yamada, K. D. (2019). MAFFT online service: multiple sequence alignment, interactive sequence choice and visualization. *Briefings in Bioinformatics*, 20(4), 1160-1166. https://doi.org/10.1093/bib/bbx108

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B. and Bruhin, F. (2004). Pytest 8.3.3. https://github.com/pytest-dev/pytest

Maitner, B. S., Boyle, B., Casler, N., Condit, R., Donoghue, J., Durán, S. M., Guaderrama, D., Hinchliff, C. E., Jørgensen, P. M., Kraft, N. J. B., McGill, B., Merow, C., Morueta-Holme, N., Peet, R. K., Sandel, B., Schildhauer, M., Smith, S. A., Svenning, J. C., Thiers, B., Violle C., Wiser, S. and Enquist, B. J. (2018). The bien r package: A tool to access the Botanical Information and Ecology Network (BIEN) database. *Methods in Ecology and Evolution*, 9(2), 373-379. https://doi.org/10.1111/2041-210X.12861

NCBI. (2025). National Center for Biotechnology Information (NCBI). https://www.ncbi.nlm.nih.gov/

OBIS. (2025). Ocean Biodiversity Information System. Intergovernmental Oceanographic Commission of UNESCO. https://obis.org.

Odum, E. P. (1971). Fundamentals of ecology. Philadelphia: W.B. Saunders Company.

Owens, H., Barve, V. and Chamberlain, S. (2025). spocc: Interface to Species Occurrence Data Sources. R package version 1.2.4. https://github.com/ropensci/spocc.

Padhye, S., Ballesteros-Mejia, L. and Ratnasingham, S. (2025). BOLDconnectR: Retrieve, Transform and Analyze the Barcode of Life Data Systems Data. R package version 1.0.0. https://cran.r-project.org/web/packages/BOLDconnectR

Porter, T. M. and Hajibabaei, M. (2018). Over 2.5 million COI sequences in GenBank and growing. *PLoS One*, 13(9), e0200177. https://doi.org/10.1371/journal.pone.0200177

R Core Team. (2025). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Ratnasingham, S., Wei, C., Chan, D., Agda, J., Agda, J., Ballesteros-Mejia, L., Boutou, H. A., El Bastami, Z. M., Ma, E., Manjunath, R., Rea, D., Ho, C., Telfer, A., McKeowan, J., Rahulan, M., Steinke, C., Dorsheimer, J., Milton, M. and Hebert, P. D. (2024). BOLD v4: A centralized bioinformatics platform for DNA-based biodiversity data. In *DNA Barcoding: Methods and Protocols* (pp. 403-441). New York, NY: Springer US. https://doi.org/10.1007/978-1-0716-3581-0_26

Reitz, K. A. (2024). Requests: HTTP for Humans™ (Version v2.32.3). https://requests.readthedocs.io/en/latest/

Richardson, L. (2024). Beautifulsoup4 (Version latest). Computer software. https://www.crummy.com/software/BeautifulSoup/

Ushey, K., Allaire, J. J. and Tang, Y. (2024). reticulate: Interface to 'Python.' https://rstudio.github.io/reticulate/

Van Rossum, G. and Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.

Winter, D. J. (2017). rentrez: An R package for the NCBI eUtils API (No. e3179v2). PeerJ Preprints. https://doi.org/10.7287/peerj.preprints.3179v2

WoRMS Editorial Board (2016). World Register of Marine Species. Available from http://www.marinespecies.org at VLIZ. Accessed 2025. doi:10.14284/170

Supplementary Material 1

S1. Examples of output for the BOLD module. (A) data in JSON format; (B) FASTA format; (C) summarized data using the parameter `summary` = *True* and (D) its field description.

S2. Examples of output for the COL module. (A) data retrieved using the parameter `check_syn` = *True*; (B) data retrieved setting the parameter `check_syn` = *False.*

S3. Examples of output for the Crossref module. (A) standard retrieved data; (B) summarized data and (C) its field description.

S4. Examples of output for the GBIF module. (A) data retrieved using the parameter `accepted_only` = *True*; (B) data retrieved using the parameter `accepted_only` = *False*; (C) occurrences downloaded within a specific polygon.

S5. Examples of output for the INaturalist module. (A) standard retrieved data.

S6. Examples of output for the IUCN module. (A) data retrieved using a list of regions; (B) data retrieved using the parameter `assess_details` = *True*.

S7. Examples of output for the NCBI module. (A) standard retrieved data; (B) data retrieved in FASTA format using the parameters `retype` = *"fasta"* and `output_format` = "*fasta"*; (C) summarized data using the parameter `summary` = *True* and (D) its field description.

S8. Examples of output for the OBIS module. (A) standard retrieved data; (B) occurrence data retrieved setting the `occ` = *True*, `geometry` = *'POLYGON((0.248 37.604, 6.300 37.604, 6.300 41.472, 0.248 41.472, 0.248 37.604))'*, and `areaid` = *33322*.

S9. Examples of output for the WORMS module. (A) standard retrieved data; (B) data retrieved coupled with the distribution information setting the parameter `distribution` = *True*.

S10. Examples of output for the ZooBank module. (A) data retrieved setting the parameter `dataset_size` = *"small"*; (B) data retrieved setting the parameter `dataset_size` = *"large"*; (C) data retrieved setting the parameter `dataset_size` = *"small"* and `info` = *True.*

S11. Commented Python script to create your own custom module.

S1. Examples of output for the BOLD module. (A) data in JSON format; (B) FASTA format; (C) summarized data using the parameter `summary` = *True* and (D) its field description.

(A) Example of data in JSON format: BOLD_A
(B) Example of data in FASTA format: BOLD_B
(C) Example of summarized data: BOLD_C
(D) Summarized data field description.

Field name	Description
record_id	The unique identifier for the BOLD record.
processid	The process ID associated with the BOLD record.
bin_uri	The BIN (Barcode Index Number) URI.
taxon	The name of the lower taxonomic level.
country	The country where the collection event took place.
province_state	The province or state of the collection event.
region	The region of the collection event.
lat	The latitude of the collection event.
lon	The longitude of the collection event.
markercode	The marker code from the sequences data.
genbank_accession	The GenBank accession number from the sequences data.

S2. Examples of output for the COL module. (A) data retrieved using the parameter `check_syn` = *True*; (B) data retrieved with the parameter `check_syn` = *False.*

(A) Example of data retrieved using the parameter `check_syn` = *True*: COL_A
(B) Example of data retrieved with the parameter `check_syn` = *False*: COL_B

S3. Examples of output for the Crossref module. (A) standard retrieved data; (B) summarized data and (C) its field description.

(A) Example of standard retrieved data: Crossref_A
(B) Example of summarized data: Crossref_B
(C) Summarized data field description.

Field name	Description
publisher	The name of the publishing entity responsible for releasing the publication.
container-title	The title of the journal or book in which the research is published.
DOI	The Digital Object Identifier assigned to the publication.
type	The publication type, such as article, book chapter, report, etc.
language	The language in which the publication is written.
URL	A direct link to the publication.
published	The date when the research was published.
title	The title of the publication.
author	The names of the authors who contributed to the research along with their main academic information.
abstract	The publication abstract.

S4. Examples of output for the GBIF module. (A) data retrieved using the parameter `accepted_only` = *True*; (B) data retrieved using the parameter `accepted_only` = *False*; (C) occurrences downloaded within a specific polygon. `geometry` = "*POLYGON((0.248 37.604, 6.300 37.604, 6.300 41.472, 0.248 41.472, 0.248 37.604))"*

(A) Example of data retrieved using the parameter `accepted_only` = *True*: GBIF_A
(B) Example of data retrieved using the parameter `accepted_only` = *False*: GBIF_B
(C) Example of occurrence download: GBIF_C

S5. Examples of output for the INaturalist module. (A) standard retrieved data.

(A) Example of standard retrieved data: INaturalist_A

S6. Examples of output for the IUCN module. (A) data retrieved using a list of regions; (B) data retrieved using the parameter `assess_details` = *True*.

(A) Example of data retrieved using a list of regions: IUCN_A
(B) Example of data retrieved using the parameter `assess_details` = *True*: IUCN_B

S7. Examples of output for the NCBI module. (A) standard retrieved data; (B) data retrieved in FASTA format using the parameters `retype` = *"fasta"* and `output_format` = "*fasta"*; (C) summarized data using the parameter `summary` = *True* and (D) its field description.

(A) Example of standard retrieved data: NCBI_A
(B) Example of data retrieved in FASTA format: NCBI_B
(C) Example of summarized data: NCBI_C
(D) Summarized data field description.

Field name	Description
Id	A numerical identifier (GI Number) that used to be assigned to each sequence version (e.g., "345678912").
Caption	A unique identifier (accession number) assigned to a sequence when it is submitted to GenBank (e.g., "NM_001256789").
Title	A short description or title of the sequence, often including information about the gene, organism, and type of sequence.
Length	The length of the sequence in base pairs (for nucleotide sequences) or amino acids (for protein sequences).
query	The original search term or query string used to retrieve this result.

S8. Examples of output for the OBIS module. (A) standard retrieved data; (B) occurrence data retrieved setting the `occ` = *True*, `geometry` = "*POLYGON((0.248 37.604, 6.300 37.604, 6.300 41.472, 0.248 41.472, 0.248 37.604))"*, and `areaid` = *33322*.

(A) Example of standard retrieved data: OBIS_A
(B) Example of occurrence download: OBIS_B

S9. Examples of output for the WORMS module. (A) standard retrieved data; (B) data retrieved coupled with the distribution information setting the parameter `distribution` = *True*.

(A) Example of standard retrieved data: WORMS_A
(B) Example of data retrieved with distribution information: WORMS_B

S10. Examples of output for the ZooBank module. (A) data retrieved setting the parameter `dataset_size` = *"small"*; (B) data retrieved setting the parameter `dataset_size` = *"large"*; (C) data retrieved setting the parameter `dataset_size` = *"small"* and `info` = *True.*

(A) Example of data retrieved setting the parameter `dataset_size` = *"small"*: ZooBank_A
(B) Example of data retrieved setting the parameter `dataset_size` = *"large"*: ZooBank_B
(C) Example of data retrieved setting the parameter `dataset_size` = *"small"* and `info` = *True:* ZooBank_C

S11. Commented Python script to create your own custom module: Python_script