cube4health Documentation

Release 0.1.dev56+g5d8f5014a.d20250923

Marcos Lima Rodrigues

Oct 30, 2025

CONTENTS:

docs/figures/cube4health_logo.png

CHAPTER

ONE

CUBE4HEALTH

CHAPTER

TWO

ABOUT

Package **cube4health** is a data acquisition, processing, and publishing toolkit designed to streamline the integration of health, climate, and drone imagery data from Brazil and any other country adhering to standardized protocols. It enables the efficient handling of heterogeneous data sources into a unified structure, supporting scalable workflows for data ingestion, transformation, and dissemination.

CHAPTER

THREE

INSTALLATION

See the installation instructions.

CHAPTER

FOUR

USAGE

The package `cube4health` has an entry point based on the Click package for the execution of functions as a command-line tool. Please see usage for examples of how to execute `cube4health` directly from the terminal.

CHAPTER

FIVE

LICENSE

Copyright (C) 2025 HARMONIZE/INPE.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

5.1 Installation

The `cube4health` relies primarily on the Geospatial Data Abstraction Library (GDAL) for raster processing. Please read the instructions below to install `cube4health`.

5.1.1 Development Installation

Install the GDAL library and its development header files on your system (Linux systems):

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install -y g++ && sudo apt-get install -y gdal-bin libgdal-dev
```

Install the Python dependencies and build numpy-based raster support (Linux systems):

```
export GDAL_VERSION=$(gdal-config --version)
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
pip3 install --upgrade "pip<=25.2" wheel numpy
pip3 install --use-pep517 --no-build-isolation --no-cache-dir --force-reinstall␣
↪gdal[numpy]==`gdal-config --version`
```

Verify that numpy-based raster support has been installed:

```
python -c "from osgeo import gdal, gdal_array ; print(gdal.__version__)"
```

Obs.: If this command raises an ImportError, numpy-based raster support has not been properly installed. Please see these related issues and solutions!

Source code from Github:

```
pip install git+https://github.com/Harmonize-Brazil/cube4health.git
```

Alternative using *Python Virtual Environment*:

1. Clone the software repository:

```
git clone https://github.com/Harmonize-Brazil/cube4health.git
```

2. Go to the source code folder:

```
cd cube4health
```

3. Create a new virtual environment linked to Python 3.10:

```
python3.10 -m venv venv
```

4. Activate the new environment:

```
source venv/bin/activate
```

5. Install using custom setup for GDAL bindings:

```
./setup_env.sh
```

Obs.: This installs the package in development mode, allowing you to modify the code without having to rebuild the package.

Problems with GDAL import? Please see these related issues and solutions!

5.1.2 Build the Documentation

You can automate the documentation process building using Sphinx. Basically, it takes the .rst files and converts them to HTML.

1. Install Sphinx:

```
pip install -e .[docs]
```

2. Build docs:

```
sphinx-build docs docs/_build/html
```

Obs.: The above command will generate the documentation in HTML and it will place it under: `docs/sphinx/_build/html/`

3. Access docs:

```
firefox docs/_build/html/index.html
```

Use your favorite browser (For example Firefox) to open the documentation.

5.2 Usage

5.2.1 Running cube4health package in the Command Line

The `cube4health` package installs a command line tool with the same name `cube4health` that allows processing and publishing health, climate, and drone data produced in the context of the Earth Observation Data Cube tuned for Health Response Systems (EODCtHRS) component of the HARMONIZE project. Important to know that the publishing process is based on predefined protocols and additional services such as BDC-STAC, Geoserver, and Titiler. Please see the documentation available on the Cube4Health GitHub page.

If you want to know the cube4health version, use the option `--version` as in:

```
cube4health --version
```

Each type of data has your specific module (subpackage) for processing and publishing, try (`eddpr` - drone, `ehipr` - health, and `eclimpr` - climate) to see parameter requirements:

```
cube4health run --module eddpr -h


usage: run --module eddpr [-h] --server_type {localhost,remote} --root_path ROOT_PATH --
↪data_path_output
                      DATA_PATH_OUTPUT --publish_data {True,False}

Convert raw images and mosaics to Cloud Optimized GeoTIFF (COG) and create JSON files to␣
↪build catalogs using
SpatioTemporal Asset Catalog (STAC) specification

required arguments:
--server_type {localhost,remote}
                      Required server target type localhost or remote
--root_path ROOT_PATH
                      Required path to raw and mosaic images from drone. Example /home/
↪user/Desktop/HARMONIZE-
                      Br_Project/src/FieldWorkCampaigns
--data_path_output DATA_PATH_OUTPUT
                      Required path to save Cloud Optimized GeoTIFF (COG) files.␣
↪Example /home/user/Docker-
                      Compose/geoserver/data
--publish_data {True,False}
                      Required parameter to specify a supplementary processing step␣
↪for automatically publishing
                      data via the BDC STAC service and Geoserver. Note: Additional␣
↪parameters will be requested
                      after data processing.

optional arguments:
-h                    show this help message and exit
```

This is an example of using the `eddpr` module to process and publish data at localhost:

```
cube4health run --module eddpr  --server_type localhost --root_path /home/user/Desktop/
↪HARMONIZE-Br_Project/src/FieldWorkCampaigns --data_path_output /home/user/Docker-
↪Compose/geoserver/data  --publish_data True
```

Obs.: the of processing drone data is based on the Harmonize protocol that shows how to provide images and auxiliary information from fieldwork campaigns available in the folder especified through the `--root_path` option. The creation of images or mosaics collections depends on templates files in JSON format for each device (e.g. Phantom 3 Advanced, Mavic 3M, Mavic 3 Enterprise), type (scene, mosaic or thermal) and flight height.

5.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.3.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/Harmonize-Brazil/cube4health/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

cube4health could always use more documentation, whether as part of the official cube4health docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Harmonize-Brazil/cube4health/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.3.2 Get Started!

Ready to contribute? Here's how to set up *cube4health* for local development.

1. Fork the *cube4health* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/cube4health.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv cube4health
$ cd cube4health/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
$ make test
Or
$ make test-all
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/Harmonize-Brazil/cube4health/pull_requests and make sure that the tests pass for all supported Python versions.

5.3.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_cube4health
```

5.3.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

5.3.6 Code of Conduct

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

5.4 Credits

5.4.1 Development Leaders

- Adeline Maciel <adelinemaciel22@gmail.com>
- Marcos Lima Rodrigues <marcos.mlr@gmail.com>
- Yuri Domaradzki <yuridomaradzki@gmail.com>

5.4.2 Contributors

- Luana Becker <luanabeckerdaluz@gmail.com>
- Gabriel Sansigolo <gabrielsansigolo@gmail.com>

5.5 cube4health

5.5.1 cube4health package

Subpackages

cube4health.eddpr package

Submodules

cube4health.eddpr.eddpr module

Utility for Drone images collection creation

Build JSON files to support STAC catalog creation from drone data (RGB/Multispectral/NDVI/Thermal scenes and mosaics). The Assets (spatiotemporal representation of Earth Observation data) files in COG format that is associated with collections are made using EXIF and XMP tags with metadata: height, width, center coordinates (GPS position of aircraft when capturing the image), camera focal length, flight yaw degree; and auxiliaries information: width size of the sensor (millimeters) and flying height (meters) available at a JSON file in the path of the source data or passed using parameters.

This approach is based on JSON files examples from Sentinel available in the examples -> fixtures folder at https://github.com/brazil-data-cube/bdc-catalog.

cube4health.eddpr.eddpr.**calc_ndvi**(*out_fname*, *fname*)

Calculate NDVI from multispectral mosaic and save to COG file

Parameters

- **out_fname** (*String*) – filename output with mission and data identification.
- **fname** (*String*) – raster multispectral.

Note: The output file uses EPSG:3395 - WGS 84/World Mercator is global coordinate system with unit in meters. Source: https://epsg.io/3395

cube4health.eddpr.eddpr.**create_png_from_raster**(*raster_tif=None*, *output_file=None*, *color_png_file=None*)

Generate a PNG file from a raster GeoTIFF file using a custom color scale.

Parameters: - raster_tif (str): Path to the input raster GeoTIFF file. - output_dir (str): Directory where the PNG file will be saved. - color_png_file (str): Path to the file containing color definitions.

cube4health.eddpr.eddpr.**decimal_coords**(*coords*, *ref*)

cube4health.eddpr.eddpr.**get_exif_info**(*img_path*)

cube4health.eddpr.eddpr.**get_flight_info**(*fname_out*)

cube4health.eddpr.eddpr.**get_local_utc**(*exif_info*)

Returns a location's datetime from UTC based on lat and lon coordinators

param exif_info
contais information from EXIF tags of image.

type exif_info
dict

cube4health.eddpr.eddpr.**get_raster_info**(*raster_file*)

cube4health.eddpr.eddpr.**get_raster_nodata**(*fname*)

cube4health.eddpr.eddpr.**get_tags_info**(*img_path*)

cube4health.eddpr.eddpr.**is_decimal**(*s*)

cube4health.eddpr.eddpr.**main**(*argv*)

Read JSON files with template information for processing drone data to produce COGs files to publish using Geoserver/Titiler and JSON files used for STAC catalog creation.

param argv.server_type
parameter that defines if data will be published using the services (STAC, Geoserver, and Titiler) running with Docker on localhost or a remote server. Because for a remote server, it will be necessary to supply additional information at runtime for remote connection to upload files and connection with Geoserver.

type argv.server_type
String

param argv.root_path
root path filename with drone data to processing (NADIR or Mosaic) of RGB, multispectral, or thermal rasters. Attention: it's required to follow the EODCtHRS protocol defined for drone data organization! Please see the document available at <https://docs.google.com/document/d/1pZ_yBBRXJBnyq6wTk4bMXDYmdHe-aP1BHejxg4-wc4U/edit?usp=sharing> for further information.

type argv.root_path
String

param argv.data_path_output
path filename to output files processed (COGs and thumbnails).

type argv.data_path_output
String

cube4health.eddpr.eddpr.**prepare_thumbnail_v4**(*filename_out*, *file_in*, *composition*)

Creates a thumbnail from GeoTIFF multispectral raster mosaic from drone data

param filename_out
filename output with mission and data identification.

type filename
String

param file_in
raster multispectral.

type file_in
String

param composition
list of band names to create a composition.

type composition
List

cube4health.eddpr.eddpr.**process_flights**(*flights_path*, *collections_template*, *catalog_path*, *prefix_geoserver_data*, *publish*)

Processing drone data to produce COGs to publish using Geoserver/Titiler and JSON files used for STAC catalog creation.

param flights_path
List with path names containing drone data for each flight.

type flights_path
Path

param collections_template
A dictionary of dictionaries containing template information from different collections of scenes/mosaics (RGB, Multispectral, NDVI, and Thermal) for supported drone models. Attention: To include support for a new device, it's required to create template JSON files with the required information for the drone and the collection wanted. See file example for Mavic 3M Multispectral data (mavic3m_flight_height120m_multispectral_template.json).

type collections_template
Dict

param catalog_path
Path name for output files (COGs and thumbnails).

type catalog_path
String

param prefix_geoserver_data
String with the parent path name for the data that will be published with Geoserver. For example, our address for Geoserver is <https://geolab.inpe.br/big/geoserver> by default, the service points to a path containing data using the prefix "harmonize".

type prefix_geoserver_data
String

param publish
String with True or False condition to publish the data collections created using STAC catalogs and Geoserver layers.

type publish
String

cube4health.eddpr.eddpr.**write_cogtiff_v2**(*fname*, *out_fname*, *type=None*)

Convert the Geotiff to COG using gdal TILED <boolean>: Switch to tiled format COPY_SRC_OVERVIEWS <boolean>: Force copy of overviews of source dataset COMPRESS=[NONE/DEFLATE]: Set the compression to use.

Note: The output file uses EPSG:3395 - WGS 84/World Mercator is global coordinate system with unit in meters.
Source: https://epsg.io/3395

Module contents

Python package for processing Drone data at HARMONIZE project scope.

cube4health.edpu package

Submodules

cube4health.edpu.geoserver module

class cube4health.edpu.geoserver.**GeoServer**(*service_url: str*, *workspace: str*, *hostname: str = 'localhost'*, *username: str = 'admin'*, *store: str = 'public'*, *db_settings: dict | None = None*)

Bases: `object`

Attributes

service_url
[str] The URL for the GeoServer instance.

workspace
[str] Workspace name to group similar layers.

hostname
[str] Hostname for server with collections files. Default values is localhost.

username
[str] Login name for session.

store
[str] Store name to connects to a data source that contains raster or vector data.

db_settings: Optional[dict] = None
Database settings (name, user, schema and port)

create_coverage_store(*data: str*, *workspace: str | None = None*, *file_type: str | None = 'GeoTIFF'*, *content_type: str | None = 'image/tiff'*) → None

Create coverage store.

Parameters

data
[str,] The path to the data.

workspace
[str, default value is None] The name of workspace.

file_type
[str, default value is 'GeoTIFF'.] The type of file.

content_type
[str, default value is 'image/tiff'] The content type.

Returns

None

create_feature_store(*workspace: str | None = None*, *store: str | None = None*) → bool

Create a new feature store in geoserver.

Parameters

workspace
[str, default value is None.] The name of workspace.

store
[str, default value is None.] The name of Store.

Returns

Boolean value of existance of store.

static create_health_feature_style(*layer_name: str*, *column_name: str*, *gdf: GeoDataFrame*, *template_name: str | None = None*) → str

Create style for layer in geoserver.

Parameters

layer_name
[str,] Name of the layer to which the style will be associated.

column_name
[str,] Name of the data column that will be used to create ranges.

gdf
[gpd.GeoDataFrame,] Layer GeoDataFrame.

Returns

The path of the style.

create_imagemosaic_store(*data: str*, *layer_name: str*, *store_name: str | None = None*, *workspace: str | None = None*, *title: str | None = None*, *time_regex: str | None = 'regex=[0-9]{8}'*, *style: str | None = None*, *is_root_path: bool | None = True*) → bool | str

Create a new coverage store in geoserver.

Parameters

data
[str] Path where the raster are stored.

layer_name
[str] Layer name.

store_name
[Optional[str], default = None,] Store name.

workspace
[Optional[str], default = None,] Workspace name.

time_regex
[Optional[str], default = 'regex=[0-9]{8}',] Time regex.

style
[Optional[str], default = None,] Style path.

is_root_path
[Optional[bool], default = False] If false, the data is not stored in the root path. Otherwise, the data is stored in the root path.

Return

True if the coveragestore was created successfully, False otherwise. And a message that indicates the status of the operation.

Example

data_path = '/home/yuri/Docker-Compose/geoserver/data/Phantom3Adv_120m_RGB' name = 'Phantom3Adv_120m_RGB' time_regex = 'regex=[0-9]{8}T[0-9]{6}'

created, message = geo.create_imagemosaic_store(data=data_path,
layer_name=name, store_name=name, time_regex=time_regex)

`create_workspace`(*workspace: str | None = None*) → bool

Create a new workspace in geoserver. The geoserver workspace url will be same as the name of the workspace.

Parameters

workspace
[str, default value is None.] The name of workspace.

Returns

Booleand value that indicates if the workspace was created.

`get_password`() → str

Get the Geoserver password

Returns

A string that contains the Geoserver password

`get_username`() → str

Get the GeoServer username

Returns

A string that contains the GeoServer username

make_thumbnail(*url: str*, *layers: List[dict]*, *time_regex: str | None = 'regex=[0-9]{8}'*, *workspace: str | None = None*) → None

Make thumbnail from Layer.

Parameters

url
[str,] The URL for the GeoServer instance.

layers
[list with dicts,] A list that contains dicts with layers informations, keys = (name, title, style, bbox, path)

time_regex
[str, default value is 'regex=[0-9]{8}'] The time regex.

workspace
[str, default value is None.] The name of workspace.

store
[str, default value is None.] The name of Store.

publish_feature_data(*layers: List[dict]*, *time_regex: str*, *attribute: str = 'date'*, *workspace: str | None = None*, *store: str | None = None*, *dynamic_style: bool = False*, *add_tile_cache: bool | None = True*) → None

Publish features in geoserver.

Parameters

layers
[list with dicts,] A list that contains dicts with layers informations, keys = (name, title, style, dates, bbox, path)

time_regex
[str] The regex that matches the time of features.

workspace
[str, default value is None.] The name of workspace.

store
[str, default value is None.] The name of Store.

dynamic_style
[bool, default value is False] Whether to use style that are created dynamically or not.

add_tile_cache
[bool, default value is True] Whether to add a tile cache or not to the layer.

Returns

None

update_imagemosaic_store(*data: str*, *root_path: str*, *layer_name: str*, *store_name: str | None = None*, *workspace: str | None = None*, *title: str | None = None*, *time_regex: str | None = 'regex=[0-9]{8}'*) → bool | str

Update Imagemosaic store

Parameters

data
[str] The path where the data is stored.

root_path
[str] The root path where the data is stored.

layer_name
[str] The name of the layer.

store_name
[str, default value is None] The name of the store.

workspace
[str, default value is None] The name of the workspace.

title
[str, default value is None] The title of the layer.

time_regex
[str, default value is 'regex=[0-9]{8}'] The time regex.

Return

A boolean value indicating if the store was updated and a string with the message.

Example

new_data = '/home/yuri/Docker-Compose/geoserver/data/Phantom3Adv_120m_RGB/2022/11' root_data = '/home/yuri/Docker-Compose/geoserver/data/Phantom3Adv_120m_RGB' name = 'Phantom3Adv_120m_RGB' time_regex = 'regex=[0-9]{8}T[0-9]{6}'

created, message = geo.update_imagemosaic_store(data=new_data,
root_path=root_data, layer_name=name, store_name=name, time_regex=time_regex)

cube4health.edpu.stac module

`class cube4health.edpu.stac.STAC(service_url: str, hostname: str = 'localhost', db_name: str = 'bdc', port: int = 5432, pg_username: str = 'postgres', pg_password: str = 'postgres')`

Bases: `object`

`browse_items(path: str, asset_names: Dict[str, str], root_data_path: str | None = None, footprint: List[int] | Dict[str, List[int]] | None = None, **kwargs) → List[dict] | str`

Browse the items directory.

Parameters

path
[str] The directory path.

asset_names
[Dict[str, str]] The asset names. The key of the dictionary is the asset name, and the value is the asset extension.

root_data_path
[str] The data path where nginx is running.

footprint
[Union[List[int], Dict[str, List[int]]], optional] The footprint of the items. The user can provide the footprint as a list or as a dictionary. If a dictionary, the key is the item name, and the value is the footprint. If a list, it represents coordinates and will be used for all items.

Returns

Union[List[dict], str]
A list of items. Each item is a dictionary with the following keys:

- *name* (str): The name of the item.
- *collection_id* (int): The collection ID.
- *start_date* (datetime): The start date of the item.
- *end_date* (datetime): The end date of the item.
- *footprint* (Polygon): The footprint of the item.
- *assets* (dict): A dictionary with the assets of the item.
- *cloud_cover* (int): The cloud cover of the item.
- *srid* (int): The SRID of the item.

If the return value is a string, it indicates an error.

`close_connection`() → str
Close the connection

Returns

The message that the connection was closed

`static format_date`(*date: str*) → str
Format the date

Parameters

date
[str,] The date to be formatted

Returns

The formatted date

`publish_collection`(*data: dict*, *template: str*, *items_path: str*, *asset_names: Dict[str, str]*, *root_data_path: str | None = None*, *additional_path: str | None = None*, *footprint: List[int] | Dict[str, List[int]] | None = None*, *output_file: str | None = None*, *del_output_file: bool | None = True*, *workspace: str | None = 'bdc_lcc'*) → int | str
Update the the collection's metadata JSON file template

Parameters

data
[dict,] The collection's metadata JSON file

template
[str,] The collection's template

output_file
[str,] The path where the collection's metadata JSON file will be created.

del_output_file
[bool, default value is True,] If True, delete the output_file.

workspace
[Optional[str], default = 'bdc_lcc',] The name of geoserver workspace.

items_path: str,
The directory path where the items are.

asset_names: Dict[str, str],
The asset names. The key of the dictionary is the asset name and the value is the asset extension.

root_data_path: str,
The data path where nginx is running.

footprint: Union[List[int], Dict[str, List[int]]], default value is None,
The footprint of the items. The user can inform the footprint in a list or in a dictionary with footprints. If the footprint is in a dictionary, the key of the dictionary is the item name and the value is the footprint. If the footprint is in a list, the footprint is a list of coordinates and it will be used to all items.

Returns

The collection id or a string with the error.

cube4health.edpu.utils module

cube4health.edpu.utils.**check_existence_remote_path**(*ssh: SSHClient*, *remote_path: str*) → bool

Check if a remote path exists.

Parameters

ssh
[SSHClient] SSH client object.

remote_path
[str] Remote path to check.

Returns

Boolean value indicating if the remote path exists.

cube4health.edpu.utils.**clean_raster_directory**(*path: str*, *conn: SSHClient | None = None*) → bool

Clean the raster directory.

Parameters

path
[str] The directory path.

Returns

True if the directory is empty. False otherwise.

cube4health.edpu.utils.**connect_ssh**(*hostname: str*, *username: str*, *password: str*, *port: int | None = 22*) → Tuple[bool, str | SSHClient]

Connect to a remote server using SSH.

Parameters

hostname
[str] Hostname of the remote server.

username
[str] Username to connect to the remote server.

password
[str] Password to connect to the remote server.

port
[int] Port to connect to the remote server. Default value is 22.

Returns

A tuple containing a boolean value indicating if the connection was successful and a string or SSHClient object depending on the success of the connection.

cube4health.edpu.utils.**execute_command**(*command: str*) → None

Execute a command in the terminal.

Parameters

command
[str] Command to execute in the terminal.

Returns

None

cube4health.edpu.utils.**files_ssh**(*ssh: SSHClient*, *method: str*, *local_path: str*, *remote_path: str*, *filename: str*) → bool

Send or retrieve files from local path to remote path, or delete files remotely.

Parameters

ssh
[SSHClient] SSH client object.

method
[str] Method to be used: 'get' for download, 'put' for upload, 'delete' for removing a file.

local_path
[str] Local path where the file is located or where it will be stored.

remote_path
[str] Remote path where the file will be sent or retrieved from.

filename
[str] Name of the file to be sent, retrieved, or deleted.

Returns

bool
Boolean value indicating if the operation was successful.

cube4health.edpu.utils.**get_ip_container_db**(*name: str = 'postgis'*, *platform: str = 'bdc_net'*) → str

Get the IP address of the container where the database is hosted.

Parameters

name
[str, default value is 'postgis'] The name of the container where the database is hosted.

platform
[str, default value is 'bdc_net'] The platform of the container where the database is hosted.

Returns

A string that contains the IP address of the container where the database is hosted.

cube4health.edpu.utils.**get_ports_container_db**(*name: str = 'postgis'*) → str

Get the ports of the container where the database is hosted.

Parameters

name
[str, default value is 'postgis'] The name of the container where the database is hosted.

Returns

A string that contains the ports of the container where the database is hosted.

cube4health.edpu.utils.**get_round_value**(*gdf: GeoDataFrame*, *column_name: str = 'vl'*, *only_round: bool = False*) → Tuple[str, float, float, int] | int

Rounds a point number of a column from a GeoDataFrame.

Parameters

gdf
[gpd.GeoDataFrame] The GeoDataFrame that contains the column to be rounded.

column_name
[str, optional] The column name to be rounded. Default is 'vl'.

only_round
[bool, optional] If True, returns only the number of decimal places to round. If False, returns a tuple containing the column name, minimum value, interval value, and number of decimal places.

Returns

Union[int, Tuple[str, float, float, int]]

If *only_round* is True:

- An integer indicating the number of decimal places to round.

If *only_round* is False:

- **A tuple containing:**
  - column_name (str): The name of the column.
  - min_value (float): The minimum value in the column.
  - interval (float): The interval value.
  - decimals (int): Number of decimal places to round.

cube4health.edpu.utils.**get_time_list_from_data**(*path: str*, *time_regex: str*, *is_vector: bool | None = False*, *conn: SSHClient | None = None*) → List[str]

Get time list from data.

Parameters

path
[str] The directory path.

time_regex
[str] The time regex.

Returns

A list with the time.

cube4health.edpu.utils.**modify_json**(*data: dict*, *template: dict*, *stac_url: str*) → dict

Modify the json template.

Parameters

data
[dict,] The json with updated values.

template
[dict,] The json template.

stac_url
[str, default value is 'http://localhost:8080/'] The STAC URL.

Returns

The json template with updated values.

cube4health.edpu.utils.**raster_convexhull**(*imagepath: str*, *epsg='EPSG:4326'*) → Polygon

Get the convex hull of a raster image footprint.

Parameters

imagepath
[str] Path to the image file.

epsg
[str] Target EPSG code for geometry.

Returns

shapely.geometry.Polygon: Convex hull of the raster footprint.

cube4health.edpu.utils.**send_files_ssh**(*ssh: SSHClient*, *paths: List[Dict[str, List[str]]]*) → bool

Send files from local path to remote path.

Parameters

ssh
[SSHClient] SSH client object.

paths
[List[Dict[str, List[str]]]] List of paths to send to the remote server.

Returns

Boolean value indicating if the file was sent.

Module contents

Top-level package for edpu.

cube4health.ehipr package

Subpackages

cube4health.ehipr.models package

Submodules

cube4health.ehipr.models.db module

Module contents

Submodules

cube4health.ehipr.config module

cube4health.ehipr.ehipr module

cube4health.ehipr.infodengue module

cube4health.ehipr.lis module

cube4health.ehipr.utils module

cube4health.ehipr.utils.**check_date_format**(*date: str*) → bool

Check if the date is in the correct format.

Parameters

date
[str] The date to check.

Returns

True if the date is in the correct format, False otherwise.

cube4health.ehipr.utils.**chunk_list**(*list_to_chunk: list*, *nchunks: int*) → List

Yield successive n-sized chunks from lst.

Parameters

list_to_chunk
[list] The list to split.

nchunks
[int] The size of the chunks.

Returns

Generator with chunks.

cube4health.ehipr.utils.**date_to_str**(*date: ~.datetime.date*) → str

Convert a datetime.date to string.

Parameters

date
[datetime.date,] The datetime.date object.

Returns

A string that contains the date.

cube4health.ehipr.utils.**shp_to_zip**(*input_path: str*, *output_path: str | None = None*) → str

Converts a shapefile to a ZIP archive.

Parameters

input_path
[str] Path of the shapefile.

output_path
[str, optional] Path of the output ZIP file. If not provided, a ZIP will be created in the same directory as the input.

Returns

str
Path to the created ZIP file.

cube4health.ehipr.utils.**str_to_date**(*date_str: str*) → date

Convert a string to datetime.date.

Parameters

date_str
[str,] The string that contains the date.

Returns

The date in datetime.date format.

Module contents

cube4health.eclimpr package

Submodules

cube4health.eclimpr.download_era5land_data module

cube4health.eclimpr.download_cptec_ftp_data module

cube4health.eclimpr.utils_bd module

cube4health.eclimpr.process_shapefile module

cube4health.eclimpr.process_climate_indicator module

Module contents

Submodules

cube4health.cube4health module

Main module.

class cube4health.cube4health.**Cube4Health**(**request_kwargs*)

Bases: object

Define an interface for cube4health modules.

run(*module=None*)

Module contents

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C