GreenEx_Py: A Python package for modelling multidimensional greenspace exposure

Yúri Grings

1395521

Supervised by:

Dr. S.M. Labib (Utrecht University)

A dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in Applied Data Science

Utrecht University

July, 2023

Acknowledgement

I would first like to express my sincere appreciation to my project supervisor; Dr. S.M. Labib. His expertise, guidance and constructive feedback were fundamental to the successful execution of this research. Additionally, I am grateful to my fellow thesis students who participated in the Spatial Data Science and Geo-AI Lab project, as their valuable insights have greatly contributed to the research process.

Abstract

Greenspace exposure is often modelled from three perspectives; availability, accessibility and visibility. However, existing methodologies applied in this field present challenges related to the transparency, reproducibility and replicability of research due to the usage of diverse software tools. Consequently, we developed an open-source Python package which overcomes these challenges by adhering to FAIR(4RS) principles and enables researchers to model multidimensional greenspace exposure.

We adhere to FAIR(4RS) principles by utilizing open-access databases and Application Programming Interfaces (APIs), thoroughly documenting function processes and openly sharing data used throughout the testing phase. While specifically showing strength in its ability to model eye-level greenness visibility, our package comprises a total of seven functions; four of which are linked to greenspace availability, one to accessibility and two to visibility.

This paper introduces 'GreenEx_Py', a Python package which we strongly believe establishes a robust foundation for advancing research in geospatial analysis and presents opportunities for further development.

Contents

List of abbreviations

Abbreviation	Definition
AoI	Area of Interest
API	Application Programming Interface
CRS	Coordinate Reference System
DSM	Digital Surface Model
DTM	Digital Terrain Model
EPSG	European Petroleum Survey Group
ESA	European Space Agency
FAIR	Findable, Accessible, Interoperable, Reusable
FAIR4RS	Findable, Accessible, Interoperable, Reusable for Research Software
GVI	Greenness Visibility Index
LoS	Line of Sight
NDVI	Normalized Difference Vegetation Index
NIR	Near InfraRed
OSM	OpenStreetMap
PoI	Points/Polygons of Interest
WHO	World Health Organization

1. Motivation and significance

The presence of greenspace holds significant ecological importance due its multifunctional abilities to improve air quality [1], [2], minimization of the urban heat island effect, lowering noise level and more [3], [4]. In addition to its environmental significance, the relevance of greenspace in urban environments extends to encompass a range of social benefits that profoundly impact the well-being and quality of life of urban residents. Existing studies suggest that improved physical health stimulated by exposure to greenspace reduces a variety of health risk factors [5]–[8]. Furthermore, positive effects on mental health are revealed by, among others, decreased levels of anxiety, stress and depression [5], [9], [10].

To examine its ecological and societal effects, studies have used objective measurement of greenspace exposure that is based on three core principles; availability, accessibility and visibility [5], [11]. Availability is defined as the physical amount of greenspace within an area of interest [12], [13]. Accessibility refers to the spatial proximity of greenspace to locations of interest [14], [15]. Visibility quantifies the amount of greenness that can be seen from a particular location of interest [16]–[18].

Research methodologies involve the usage of various tools, including both proprietary software such as ArcGIS [5], [19] and open-source software like QGIS [20], [21] for topics related to availability and accessibility. Regarding studies associated with greenspace visibility, the application of open-source Python code is frequently favoured due to its robust capacity for processing street view images [22]–[24].

The utilization of diverse tools and software in specific presents challenges to the reproducibility and replicability of research. First, proprietary software introduces inherent barriers to accessibility due to additional costs or platform compatibility [25], as well as to interpretation because of the opaque nature of functions for which the internal workings are not disclosed [26]. Second, due to the fixed capabilities of software in general, users are likely to encounter limitations that restrict their ability to employ scientific methods which are based on underlying theory [25], [27]. Third, a series of, often undocumented, decisions is made while operating the software, compromising the possibility of reproducing research [25], [28].

To achieve transparent, reproducible and replicable research, open code libraries and thorough documentation on data and decisions are considered essential [17], [26]. These requirements align with the FAIR and FAIR4RS frameworks which emphasize the principles of Findable, Accessible, Interoperable and Reusable for data and software, respectively [29], [30]. Findable data and tools allow researchers to reproduce a study by locating the exact resources that were used. Being able to retrieve these resources without the need for costly subscriptions and proprietary software makes them accessible and promotes transparency. The demand for open-access data and tools necessitates the interoperability of data, referring to standardized formats and protocols, to facilitate the enrichment process by enabling a seamless information exchange across different platforms. Lastly, for data and tools to be reusable, thorough documentation and proper licensing are of the essence [29], [30].

In an effort to combine existing study methods while overcoming the previously addressed challenges and adhering to the FAIR(4RS) principles, this study aims to introduce the ‘GreenEx_Py’ package, an open-source Python tool for analysing multidimensional greenspace exposure comprising availability, accessibility and visibility. By using open-access data covering the globe, the package effectively overcomes challenges related to robustness and scalability, thus enabling the analysis of study areas on a global scale. Moreover, it offers an integrated and holistic solution for researchers to model greenspace exposure from multiple perspectives as opposed to traditional methodologies that necessitate the usage of various software tools.

2. Software description

The package includes a set of functions designed to model greenspace exposure from multiple perspectives; availability, accessibility and visibility. We aim to provide researchers with an open-source and user-friendly tool that may facilitate analyses concerning the ecological and societal effects of greenspace exposure, among others.

2.1. Software installation

We used the Python programming language to develop the 'GreenEx_Py' package [31], as well as a collection of freely available packages of which the fundamental ones are listed in Appendix I.

The package source code, together with the accompanying documentation, can be accessed through GitHub. Currently, there are two methods for using the package;

1. Execution on a local machine involves cloning the GitHub repository and installing the corresponding Python environment. Installation manuals for Windows and Mac platforms can be found in Appendix II and Appendix III, respectively.
2. Execution on free-to-use Google Colab platform using the designated Jupyter notebooks.

2.2. Software architecture

The functionalities of the GreenEx_Py package are categorized into three aforementioned groups; availability, accessibility and visibility. The package architecture encompasses general workflow components that apply to each of the functionalities. These components include;

- *Processing user inputs*
  To ensure the proper execution of functions, user inputs are validated by verifying if the provided files and values align with listed criteria. More specifically, this phase includes reprojection of the data if these are not provided with a projected Coordinate Reference System (CRS). Also, the majority of function parameters has a limited number of valid values of which one needs to be used in order for the functions to work properly.

- *Data enrichment*
  If the user can or does not provide the complete set of necessary data, the function automatically retrieves the part which is missing from open-access databases and via Application Programming Interfaces (APIs) to facilitate calculations.

- *Processing acquired data*
  The data are processed once acquired. This phase generally involves computations related to the area of interest (AoI) for which calculations are to be made.

- *Performing requested calculations*
  Algorithms and statistical methods are used to generate the requested values related to greenspace exposure based on the processed data.

- *Returning results to user*
  The results are presented to the user in the original format that was initially provided to the function. Additionally, if applicable and desired, supporting files and interactive plots are also returned.

An overview of the general functionality components, as well as a visual representation of the greenspace exposure principles [5], is provided in Figure 1.

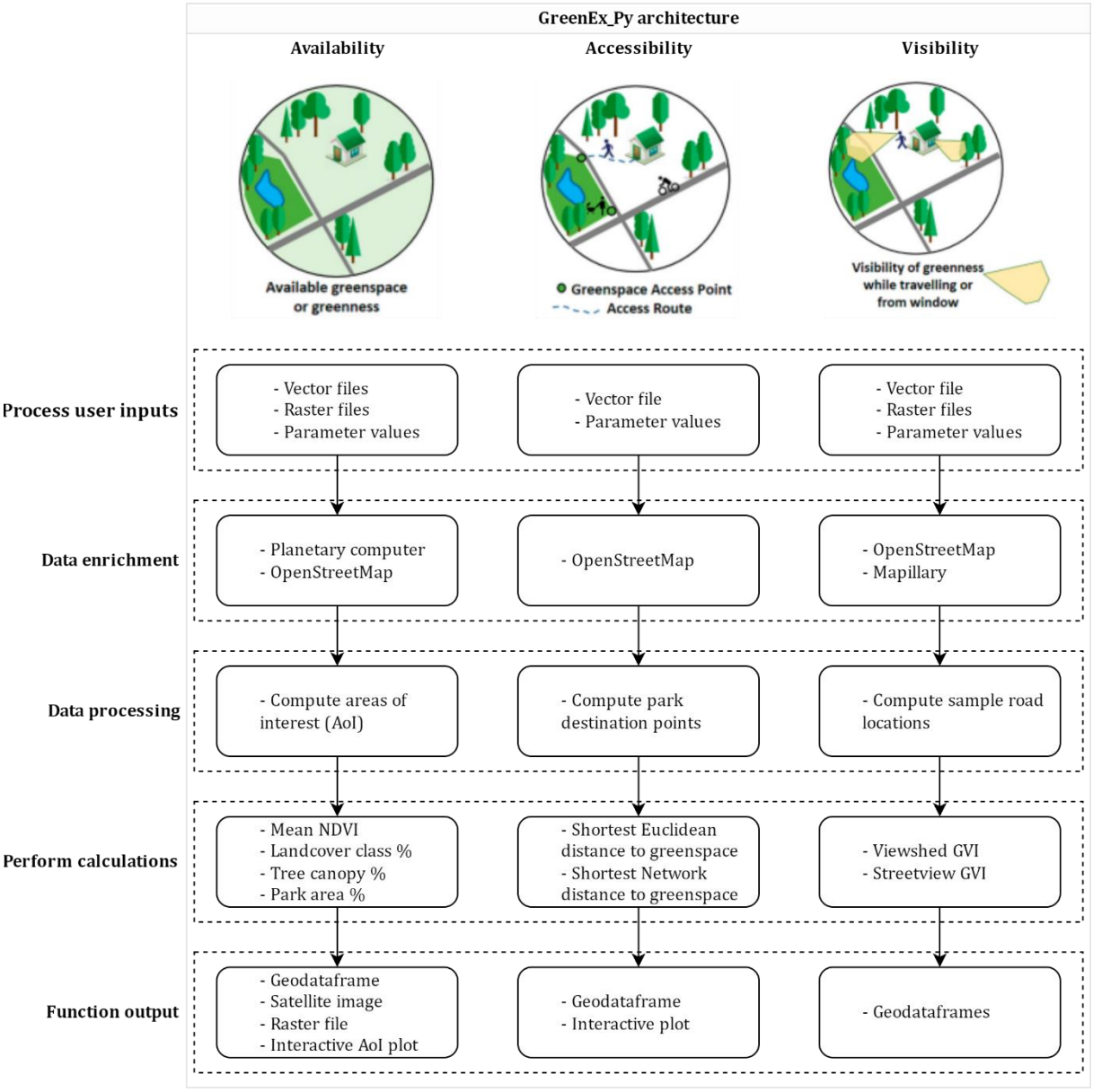

Figure 1. GreenEx_Py package architecture

A comprehensive description of the function inputs, processes and outputs is given in the subsequent sections of this chapter.

2.3. Software functionalities

We designed the functions in a way that enables users to analyse specific locations or areas of interest. Consequently, the common core parameter of all functions, *point_of_interest_file*, requests a vector file containing either point or polygon geometries (PoI), implying that a mixture of geometry types is not supported. These PoIs may for example represent house addresses or neighbourhoods. For the file to be properly processed, it is important that the CRS is specified in the file's metadata.

2.3.1. Availability

The package accommodates four functions to measure greenspace availability, i.e. the physical amount of greenspace within an area of interest;

1. **get_mean_ndvi()**: calculates the average Normalized Difference Vegetation Index (NDVI).

2. **get_landcover_percentages()**: calculates the percentage of area covered by each landcover class.
3. **get_canopy_perentage()**: calculates the percentage of area covered by tree canopy.
4. **get_greenspace_percentage()**: calculates the percentage of area covered by greenspaces.

Below, we elaborate on the general workflow of these functions based on the components depicted in Figure 1 and while using a selection of common function parameters;

- **point_of_interest_file**: the file which contains the PoI geometries.
- **buffer_type**: the way in which to compute the areas of interest, options are "euclidean", "network" and None (Note: None is only valid if PoI file contains polygon geometries that represent neighbourhoods).
- **network_type**: applicable if buffer_type is set to "network", the travel mode for which the network should be retrieved, options are "walk", "bike", "drive" and "all".
- **buffer_dist**: the distance in meters that will be considered when computing the area of interest. Note: if buffer_type is set to "network", buffer_dist should not be set when trip_time and travel_speed are set.
- **trip_time**: the trip time in minutes that will be considered when computing the area of interest. Note: if buffer_type is set to "network", trip_time and travel_speed should not be set when buffer_dist is set.
- **travel_speed**: the travel speed in km/h that will be considered when computing the area of interest. Note: if buffer_type is set to "network", trip_time and travel_speed should not be set when buffer_dist is set.

Complementary to these, we refer to unique function parameters and properties when relevant. The complete collections of parameters for the four availability functions are documented in Appendices IV to VII, respectively.

Process user inputs

Parameter values are evaluated individually and combined, considering unsupported combinations. Additionally, if needed, the PoI file is reprojected into a projected CRS. Following this, the European Petroleum Survey Group (EPSG) code is obtained from the CRS, establishing it as the designated reference for reprojecting all data employed in subsequent analysis steps.

The function *get_mean_ndvi()* includes an optional parameter *ndvi_raster_file* that allows users to provide a file where each raster pixel contains an NDVI value. If provided, the rioxarray package is used to read the file, reproject the data when needed and verify that all PoIs fall within the raster's extent [32]. Similarly, the function *get_landcover_percentages()* permits users to optionally bring in a raster file where each pixel is assigned a specific landcover class through the *landcover_raster_file* parameter.

The function *get_canopy_percentage()* requires users to provide a vector file containing canopy geometries of either Polygon or Multipolygon type. The file can be passed through the *canopy_vector_file* parameter. Likewise, the function *get_greenspace_percentage()* includes an optional parameter, *greenspace_vector_file*, to pass a vector file containing geometries for greenspace areas.

Data enrichment

Each function requires a specific set of data to perform the requested calculations. This set may be split into three categories based on the user inputs;

1. PoIs
2. Raster/vector data
3. Network (if *buffer_type* set to "network")

After the user inputs have been processed, we utilize the osmnx package and planetary computer API to acquire data that is still considered missing. The osmnx package automatically processes network topology from raw OpenStreetMap (OSM) data whereas the planetary computer is a platform through which satellite images can be retrieved [33], [34].

Before extracting the data, we determine the region for which the data should be obtained. We do this by creating a polygon that encompasses the total bounds of the PoI file. This polygon is expanded by a buffer distance if specified through either the *buffer_dist* parameter or the *travel_speed* and *trip_time* parameters.

The function *get_mean_ndvi()* retrieves satellite images from the Sentinel-2 program, specifically the Red and Near InfraRed (NIR) bands, to calculate NDVI values at a 10-meter resolution [33]. On the other hand, for the function *get_landcover_percentages()*, the landcover image is obtained from the European Space Agency (ESA) WorldCover collection which provides global maps for 2020 and 2021 at 10-meter resolution [33], [35].

The function *get_greenspace_percentage()* obtains geometries for greenspace areas from OSM if the user does not provide them. The greenspace query for OSM is based on conditions stated by Bart Breekveldt [36]. All functions also use OSM to retrieve the road network for the travel mode passed to the *network_type* parameter if a network buffer is desired.

Data processing

Regardless of the geometry type that is present within the *point_of_interest_file*, each function performs its calculations for an AoI. The AoI of each PoI can be composed in three distinct ways;

- AoI(s) provided by user (i.e. polygon geometries), no buffer zone created
- AoI(s) created by defining Euclidean buffer zone
- AoI(s) created by defining Network buffer zone

The difference between the latter two is illustrated in Figure 2.

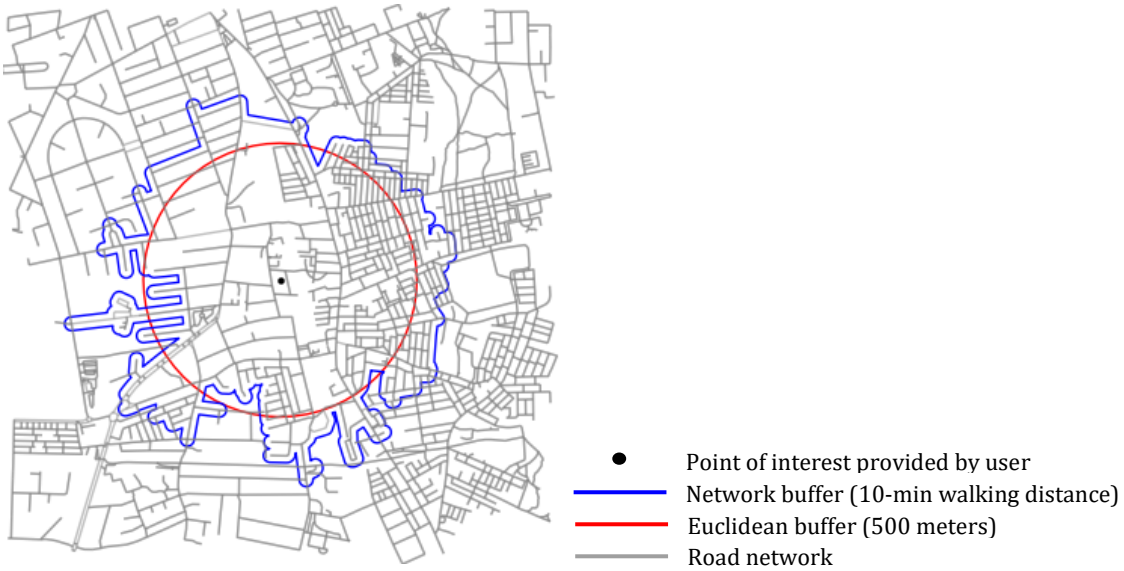

Figure 2. Euclidean vs. Network buffer

The euclidean buffer is based on a straight-line distance calculated from a point or the edges of a polygon. In contrast, a network buffer considers the actual travel distance along a road network from a specific location, resulting in an irregular-shaped buffer [37]. An induced road network is retrieved for

the functions by utilizing the networkx package. This package applies Dijkstra's algorithm to compute shortest paths between a source node and all other nodes present within a network [38].

Perform calculations
The polygon geometries of the previously created buffer zones are used to perform a clipping operation on each function's raster/vector data for all PoIs. The clipped data then serves as input for;

- Calculating the mean NDVI, together with the standard deviation, for *get_mean_ndvi()*
- Computing the proportion of area covered by each distinct landcover class, expressed as a percentage for *get_landcover_percentages()*
- Calculating the percentage of canopy tree coverage for *get_canopy_percentage()*
- Calculating the percentage of greenspace coverage for *get_greenspace_percentage()*

Function output
Throughout the execution process, real-time updates regarding the ongoing function steps are presented. Finally, each function returns a Geodataframe similar to the one which was originally provided by the user. It encompasses additional columns for PoI ID (if not already present) and the desired values pertaining to the four corresponding functions.

2.3.2. Accessibility

The package accommodates one function to measure greenspace accessibility, i.e. the spatial proximity of greenspace to locations of interest;

1. **get_shortest_distance_greenspace()**: assesses whether or not greenspaces are located within a threshold distance from the PoIs.

Below, we elaborate on the workflow of this function based on the components depicted in Figure 1 and while using a selection of the function's parameters;

- **point_of_interest_file**: the file which contains the PoI geometries.
- **greenspace_vector_file**: optional, the file which contains the greenspace geometries.
- **distance_type**: the way in which the shortest distance should be calculated, options are "euclidean" and "network".
- **destination**: the greenspace destination points, options are "centroids" and "entrance". If "entrance", distances will be computed between PoI and the network nodes that are within twenty meters of the greenspace boundaries – therefore considered as pseudo-entry points.
- **network_type**: applicable if buffer_type is set to "network", the travel mode for which the network should be retrieved, options are "walk", "bike", "drive" and "all".

The complete collection of parameters for the accessibility function is documented in Appendix VIII.

Process user inputs
In case the PoI file contains polygon geometries, centroids are computed since point geometries are required for the distance calculation. In addition, the PoI file is reprojected if it does not have a projected CRS yet. Following this, the EPSG code is obtained from the CRS, establishing it as the designated reference for reprojecting all data employed in subsequent analysis steps.

Data enrichment
The function obtains geometries for greenspace areas from OSM if the user does not provide them. The greenspace query for OSM is based on conditions stated by Bart Breekveldt [36]. OSM is also used to

retrieve the road network for the travel mode passed to the *network_type* parameter. The latter only applies if either *distance_type* is set to "network" or *destination* is set to "entrance".

Before extracting these data, we determine the region for which the data should be obtained. We do this by creating a polygon that encompasses the total bounds of the PoI file and expanding this polygon by the target distance.

Data processing

Depending on the user inputs, greenspace destination points are created by computing either centroids or pseudo-entry points. The pseudo-entry points are represented by network nodes that are located within twenty meters of the greenspace boundaries. Destination points are only created for greenspaces that intersect the area of interest which is defined by the PoI involved surrounded by a straight-line buffer of *target_dist* meters.

Perform calculations

If *distance_type* is set to "euclidean", the function calculates the distance between the PoI and the nearest destination point which is identified by using a k-d tree for euclidean nearest neighbour search, provided by the SciPy package [39].

If *distance_type* is set to "network", the network distance is calculated between the PoI's nearest network node and the pseudo-entry points of the greenspaces. This is done using the networkx package that applies Dijkstra's algorithm to compute the shortest path [38]. To minimize the distance error, the euclidean distance between the PoI and its nearest network node is added to the previously calculated network distance. Similarly, the euclidean distance between the greenspace's centroid and its pseudo-entry points is added if *destination* is set to "centroids".

The difference between a network distance and a euclidean one is illustrated in Figure 3.

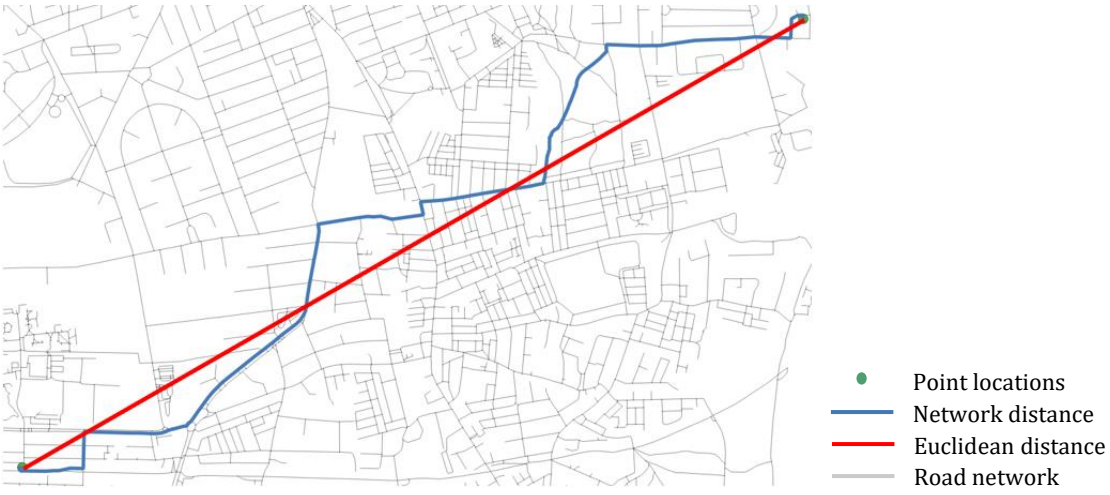

Figure 3. Euclidean vs. Network distance

Function output

Throughout the execution process, real-time updates regarding the ongoing function steps are presented. Finally, the function returns a Geodataframe similar to the one which was originally provided by the user. It encompasses additional columns for PoI ID (if not already present), a boolean value indicating whether or not greenspace is located within the target distance and the (pseudo-)distance to greenspace.

2.3.3. Visibility

The package accommodates two functions to measure greenspace visibility, i.e. the amount of greenness that can be seen from a specific location of interest;

1. **get_streetview_GVI()**: calculates the average Greenness Visibility Index (GVI) based on a streetview analysis.
2. **get_viewshed_GVI()**: calculates the average GVI based on a viewshed analysis. The function is built upon research conducted by Labib et al. [24].

Below, we elaborate on the general workflow of these functions based on the components depicted in Figure 1 and while using a selection of common function parameters;

- **point_of_interest_file**: the file which contains the PoI geometries.
- **buffer_dist**: the distance in meters that will be considered when computing road network locations.
- **network_file**: the file which contains the road network to consider.

Complementary to these, we refer to unique function parameters and properties when relevant. The complete collections of parameters for the two visibility functions are documented in Appendices IX and X, respectively.

Process user inputs

The PoI file is reprojected if it does not have a projected CRS yet. Following this, the EPSG code is obtained from the CRS, establishing it as the designated reference for reprojecting all data employed in subsequent analysis steps.

The function *get_viewshed_GVI()* requires users to provide three files containing a Digital Terrain Model (DTM), Digital Surface Model (DSM) and a binary greenspace raster. The corresponding parameters are *dtm_raster_file*, *dsm_raster_file* and *greendata_raster_file*, respectively. All PoIs should fall within these raster extents. The function *get_streetview_GVI()* requests users to provide an access token for the Mapillary API.

Data enrichment

The function *get_streetview_GVI()* obtains streetview images for road network locations, which result from the data processing phase, through Mapillary. Furthermore, both functions use OSM to retrieve the road network if no network file is provided by the user. Before extracting the network, we determine the region for which it should be obtained. We do this by creating a polygon that encompasses the total bounds of the PoI file and expanding this polygon by the buffer distance if specified.

Data processing

The function *get_viewshed_GVI()* includes a parameter *sample_dist* via which users can specify a distance interval in meters that needs to be considered when computing sample road locations in the vicinity of the PoIs. GVI values will be calculated for these sample road locations. The same applies to the function *get_streetview_GVI()*, the only difference being that it uses fifty meters as standard interval rather than an interval specified by the user.

Perform calculations

The function *get_streetview_GVI()* processes both panoramic and non-panoramic images obtained from Mapillary. Each panoramic image is split into two based on the identification of road centres. Images are then segmented using the pre-trained Masked-attention Mask Transformer model [40]. Lastly, image segments serve as input for calculating the proportion of visible green pixels, expressed as a percentage.

The function *get_viewshed_GVI()* uses the Midpoint Circle Algorithm to identify boundary cells of the viewshed area [24], [41], [42] and Bresenham's line algorithm to determine a Line-of-Sight (LoS) from the centre cell to each boundary cell [24], [43], [44]. It also considers the reducing visual prominence of objects in space with increasing distance from the location of interest prior to calculating the percentage of visible green cells. The concept is illustrated in Figure 4 [24].

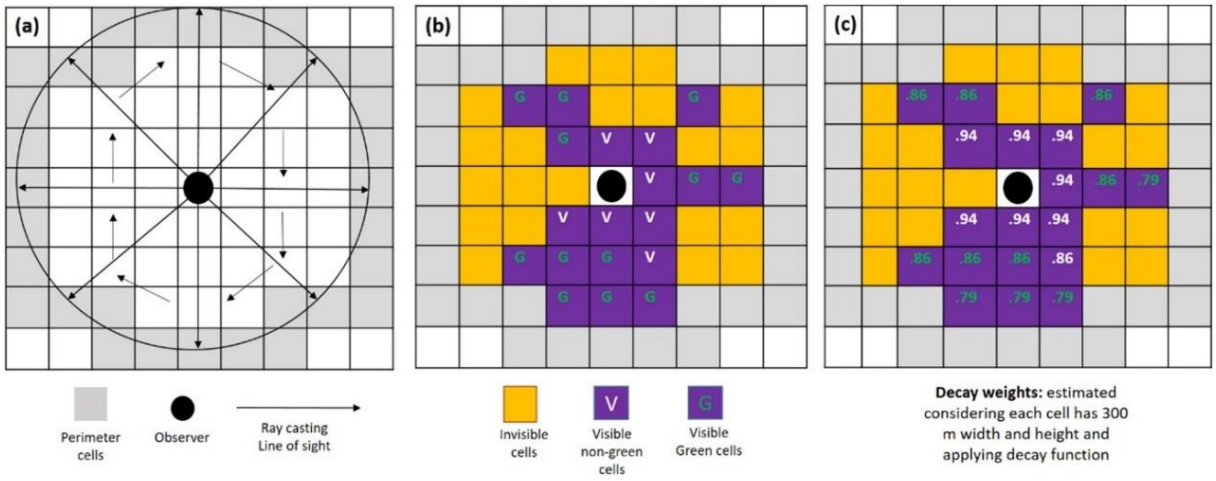

Figure 4. Example of (a) Line-of-sight algorithm for viewshed analysis for a given viewing distance, (b) viewshed outcome for observer cell; (c) the decay weights associated with the visible cells [24].

Function output

Throughout the execution process, real-time updates regarding the ongoing function steps are presented. Finally, each function returns two Geodataframes. The first one is similar to the one which was originally provided by the user. It encompasses additional columns for PoI ID (if not already present) and the average GVI values pertaining to the two corresponding functions. The second geodataframe contains the sample road locations upon which these average GVI values were based.

3. Illustrative examples

The purpose of the following examples is to provide a clearer understanding of how to utilize the functions of the 'GreenEx_Py' package. For a more detailed overview on function inputs, parameters and outputs, users are invited to check the corresponding documentation on GitHub.

We provide the majority of examples in the context of three specific locations within the city of Amsterdam. Consequently, we store the file path of the example data, which serves as core input for all functions, using code snippet 1.

Code snippet 1

```
poi_file = "C:/Users/ygrin/Example_data/AMS_example_data.gpkg"
```

Due to limited availability of canopy data, the function *get_canopy_percentage()* will be exhibited using a file which contains a single point location for Amsterdam and is stored using code snippet 2. The file containing the tree canopy geometries is stored as well.

Code snippet 2

```
poi_file_canopy = "C:/Users/ygrin/Example_data/AMS_example_data_single.gpkg"
canopy_file = "C:/Users/ygrin/Example_data/tree_canopy.gpkg"
```

As the examples solely serve for demonstration purposes, we include parameter values which prevent files from being saved locally and interactive plots to be created by the functions.

3.1. Availability

3.1.1. get_mean_ndvi()

Suppose that we want to retrieve the mean NDVI values for the three locations of the example data using a euclidean buffer of 300 meters. This can be achieved by applying the code from code snippet 3.

Code snippet 3

```
availability.get_mean_NDVI(point_of_interest_file=poi_file,
                           buffer_type="euclidean",
                           buffer_dist=300,
                           write_to_file=False,
                           save_ndvi=False,
                           plot_aoi=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Retrieving NDVI raster through planetary computer...
Information on the satellite image retrieved from planetary computer, used to calculate NDVI values:
  Date on which image was generated: 2023-06-14T18:28:31.597880Z
  Percentage of cloud cover: 0.084894
  Percentage of pixels with missing data 3e-05
Done, running time: 0:00:13.447168

Calculating mean NDVI values...
Done, running time: 0:00:01.881099
```

This information confirms that the NDVI raster was created based on a satellite image retrieved via the planetary computer API as we did not provide the raster file ourselves. The function output, including the mean NDVI values and their standard deviations, is shown in Table 1.

Table 1. *get_mean_ndvi()* function output

	geometry	id	mean_NDVI	std_NDVI
0	POINT (118883.345 485054.641)	1	0.204	0.136
1	POINT (118246.855 488082.089)	2	0.187	0.129
2	POINT (122483.550 487728.517)	3	0.047	0.073

A visual representation of the PoIs, AoIs and NDVI values upon which the results are based, is provided in Figure 5. As expected, NDVI values are higher for locations that are in the vicinity of parks and trees (point id 1 and 2) compared to those that are mainly surrounded by water and built-up area (point id 3).

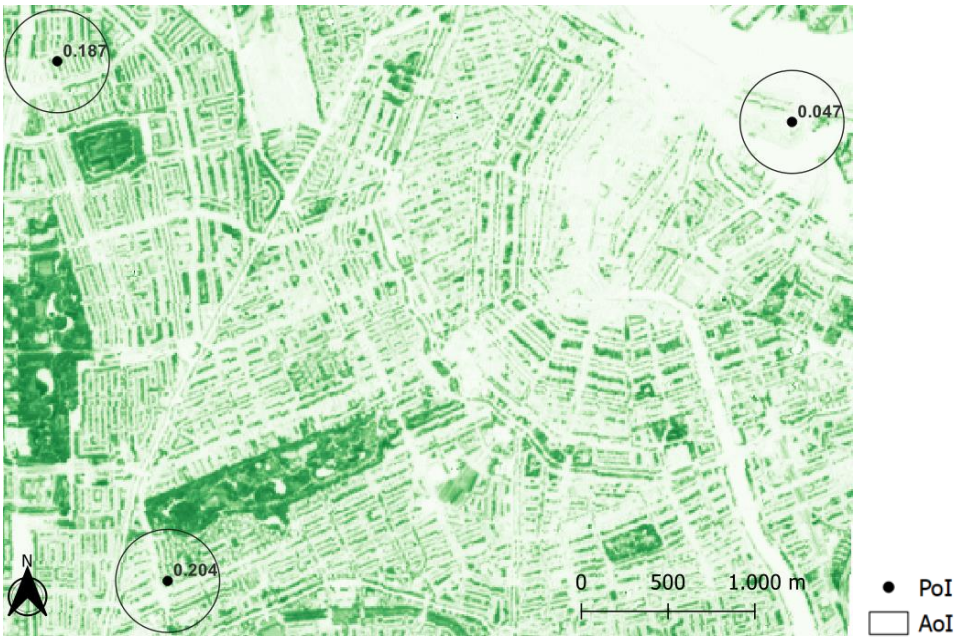

Figure 5. Visual representation of the *get_mean_ndvi()* results

3.1.2. get_landcover_percentages()

Suppose that we want to retrieve the percentage of area that is covered by each distinct landcover class based on a 500-meter radius surrounding the PoIs. This can be achieved by applying the code from code snippet 4.

Code snippet 4

```
availability.get_landcover_percentages(point_of_interest_file=poi_file,
                                       buffer_dist=500,
                                       buffer_type="euclidean",
                                       save_lulc=False,
                                       write_to_file=False,
                                       plot_aoi=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Retrieving landcover class raster through planetary computer...
Information on the land cover image retrieved from planetary computer:
  Image description: ESA WorldCover product at 10m resolution
  Image timeframe: 2021-01-01T00:00:00Z - 2021-12-31T23:59:59Z
Done, running time: 0:00:02.506165

Calculating landcover class percentages...
Done, running time: 0:00:00.509130
```

This information confirms that the landcover map was obtained via the planetary computer API as we did not provide a landcover raster ourselves. The function output, including the percentages for each distinct landcover class, is shown in Table 2.

Table 2. *get_landcover_percentages()* function output

	geometry	id	0	Tree cover	Grassland	Built-up	Bare / sparse vegetation	Permanent water bodies	Cropland
0	POINT (118883.345 485054.641)	1	21.843%	21.37%	0.771%	53.247%	0.013%	2.757%	NaN
1	POINT (118246.855 488082.089)	2	21.302%	18.135%	1.687%	57.813%	0.114%	0.928%	0.021%
2	POINT (122483.550 487728.517)	3	21.357%	2.091%	0.514%	40.361%	0.034%	35.643%	NaN

Missing data is indicated by the '0' column, suggesting that data quality is an additional aspect to consider when depending on open-access data. A visual representation of the PoIs, AoIs and landcover classes upon which the results are based, is provided in Figure 6. This figure also confirms what was said about the NDVI values in Figure 5; the location with point id 3 is mainly surrounded by water and built-up area whereas the other two locations have higher percentages for tree coverage.

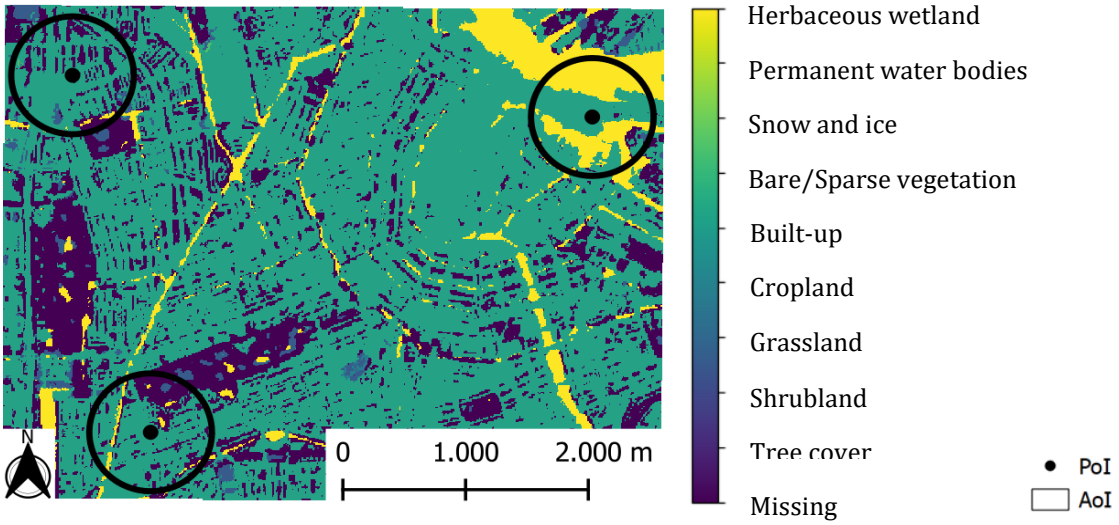

Figure 6. Visual representation of the *get_landcover_percentages()* results

3.1.3. get_canopy_percentage()

Suppose that we want to retrieve the percentage of tree canopy coverage within a circular area of 250 meters centred around the PoI. This can be achieved by applying the code from code snippet 5.

Code snippet 5

```
availability.get_canopy_percentage(point_of_interest_file=poi_file_canopy",
                                   canopy_vector_file=canopy_file,
                                   buffer_type="euclidean",
                                   buffer_dist=250,
                                   write_to_file=False,
                                   plot_aoi=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Adjusting CRS of Greenspace file to match with Point of Interest CRS...
Done

Calculating percentage of tree canopy coverage...
Done, running time: 0:00:00.813638
```

This confirms that the tree canopy file was reprojected as the CRS did not match the CRS of the PoI file. The function output, including the percentage of tree canopy coverage, is shown in Table 3.

Table 3. *get_canopy_percentage()* function output

	geometry	id	canopy_cover
0	POINT (122906.402 487497.569)	1	12.31%

A visual representation of the PoI, AoI and tree canopy geometries upon which the results are based, is provided in Figure 7.

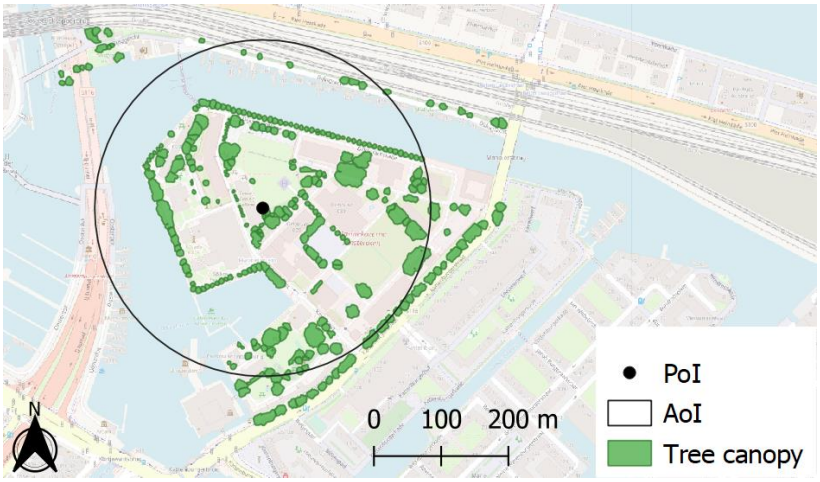

Figure 7. Visual representation of the *get_canopy_percentage()* output

3.1.4. get_greenspace_percentage()

Suppose that we want to retrieve the percentage of greenspace coverage for an area located within a 15-minute walking distance from the PoIs. This can be achieved by applying the code from code snippet 6.

Code snippet 6

```
availability.get_greenspace_percentage(point_of_interest_file=poi_file,
                                       buffer_type="network",
                                       travel_speed=5,
                                       trip_time=15,
                                       network_type="walk",
                                       write_to_file=False,
                                       plot_aoi=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Retrieving greenspaces within total bounds of Point(s) of interest, extended by buffer distance if specified...
Done, running time: 0:00:09.669733

Retrieving network within total bounds of Point(s) of interest, extended by buffer distance as specified...
Done, running time: 0:00:44.845513

Retrieving isochrone for point(s) of interest: 100%
3/3 [00:17<00:00, 5.31s/it]
Note: creation of isochrones based on code by gboeing, source: https://github.com/gboeing/osmnx-examples/blob/main/notebooks/13-isolines-isochrones.ipynb

Calculating percentage of greenspace area coverage...
Done, running time: 0:00:00.597858
```

This confirms that both the greenspace geometries and network are retrieved through OSM. The function output, including the percentage of greenspace coverage, is shown in Table 4.

Table 4. *get_greenspace_percentage()* function output

	geometry	id	greenspace_cover
0	POINT (118883.345 485054.641)	1	21.34%
1	POINT (118246.855 488082.089)	2	13.38%
2	POINT (122483.550 487728.517)	3	1.14%

A visual representation of the PoIs, AoIs and greenspace geometries upon which the results are based, is provided in Figure 8.

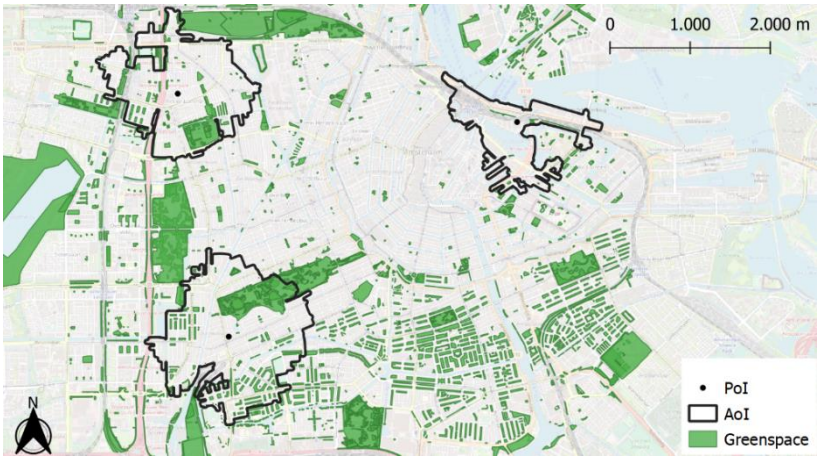

Figure 8. Visual representation of the *get_greenspace_percentage()* output

3.2. Accessibility

3.2.1. get_shortest_distance_greenspace()

Suppose that we want to assess whether or not greenspace centroids are located within a 300-meter euclidean distance from the PoIs. Additionally, we only want to consider greenspaces that cover at least an area of 400 square meters. This can be achieved by applying the code from code snippet 7.

Code snippet 7

```
accessibility.get_shortest_distance_greenspace(point_of_interest_file=poi_file,
                                               target_dist=300,
                                               distance_type='euclidean',
                                               destination='centroids',
                                               min_greenspace_area=400,
                                               write_to_file=False,
                                               plot_aoi=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Retrieving greenspaces within total bounds of point(s) of interest, extended by a 450.0m buffer to account for edge effects...
Done, running time: 0:00:08.386876

Calculating shortest distances...
Warning: no greenspace centroids could be detected within euclidean distance of 300m for PoI with id 1, distance_to_greenspace is therefore set to target distance. Consider for further analysis.
Done, running time: 0:00:00.162571
```

The user has been informed that there is no greenspace centroid within a 300-meter euclidean distance from the PoI with ID 1. Since the nearest centroid cannot be determined with certainty due to edge effects, the distance_to_greenspace is therefore set to the target distance. In this case, edge effects imply that greenspaces might be available just outside the target distance. These greenspaces may not be retrieved from OSM as they did not fall within the total bounds' polygon created for the data enrichment process of the function, as elaborated upon in Section 2.3.2.

The function output, including a boolean value indicating whether or not greenspace centroids are available within a euclidean distance of 300 meters and the (pseudo-)distance, is shown in Table 5.

Table 5. *get_shortest_distance_greenspace()* function output

	geometry	id	greenspace_within_300m	distance_to_greenspace
0	POINT (118883.345 485054.641)	1	False	300.0
1	POINT (118246.855 488082.089)	2	True	177.0
2	POINT (122483.550 487728.517)	3	True	135.0

A visual representation of the PoIs, AoIs and greenspace centroids (destinations) upon which the results are based, is provided in Figure 9. Note that only centroids are computed for greenspaces that intersect at least one of the AoIs.

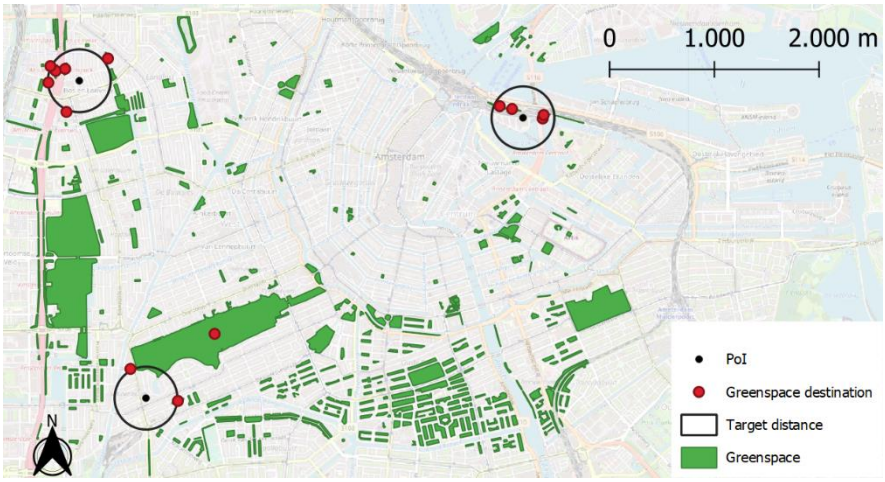

Figure 9. Visual representation of *get_shortest_distance_greenspace()* output

3.3. Visibility

3.3.1. get_streetview_GVI()

Suppose that we want to retrieve the average GVI value for road locations that fall within a 150-meter radius of the PoIs. This can be achieved by applying the code from code snippet 8.

Code snippet 8

```
visibility.get_streetview_GVI(point_of_interest_file=poi_file,
                              access_token="MAPILLARY_API_TOKEN",
                              buffer_dist=150,
                              write_to_file=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Retrieving network within total bounds of Point(s) of interest, extended by the buffer_dist in the case provided...
Done, running time: 0:00:47.329337
```

```
Computing sample points for roads within the area of interest's network...
Done, running time: 0:00:01.145286

Downloading StreetView images for road sample points...
Downloading tiles: 100%
5/5 [00:20<00:00, 3.44s/it]
Downloading images: 100%
86/86 [43:10<00:00, 23.47s/it]
Done, running time: 0:44:08.827193

Calculating StreetView GVI score...
Done, running time: 0:00:00.373096

Note: workflow for calculating Streetview GVI based on code by Ilse A. Vázquez Sánchez
source: https://github.com/Spatial-Data-Science-and-GEO-AI-Lab/StreetView-NatureVisibility
```

This confirms that sample road locations are computed within the vicinity of the PoIs. Consequently, streetview images are obtained based on these locations. The function results are split into two dataframes that are presented in Tables 6 and 7, respectively. The first dataframe contains the original PoIs, the average GVI values and the number of sample road locations upon which these are based. The second dataframe contains the sample road locations, their respective GVI values and information on the streetview images used. Note that GVI values for sample road locations may result in "NaN" in case no streetview image was found.

Table 6. *get_streetview_GVI()* function output pt. 1 (PoIs)

	id	geometry	GVI	nr_of_points
0	1	POINT (118883.345 485054.641)	0.153368	33
1	2	POINT (118246.855 488082.089)	0.212064	36
2	3	POINT (122483.550 487728.517)	0.003300	13

Table 7. *get_streetview_GVI()* function output pt. 2 (sample road locations)

	id	geometry	GVI	is_panoramic	missing
0	1	POINT (119009.625 484981.604)	0.077009	True	False
1	1	POINT (118963.280 484962.840)	0.070314	True	False
2	1	POINT (118916.934 484944.076)	0.102058	True	False
...	...	...	...	...	...
83	3	POINT (122393.881 487732.156)	0.000000	True	False
84	3	POINT (122408.407 487615.927)	0.001916	False	False
85	3	POINT (122353.549 487686.836)	NaN	None	True

Visualizations are not supported by this function. However, to illustrate the fundamental process of segmenting the streetview images, Figure 10 was created in which a panoramic image was split into two prior to segmentation.

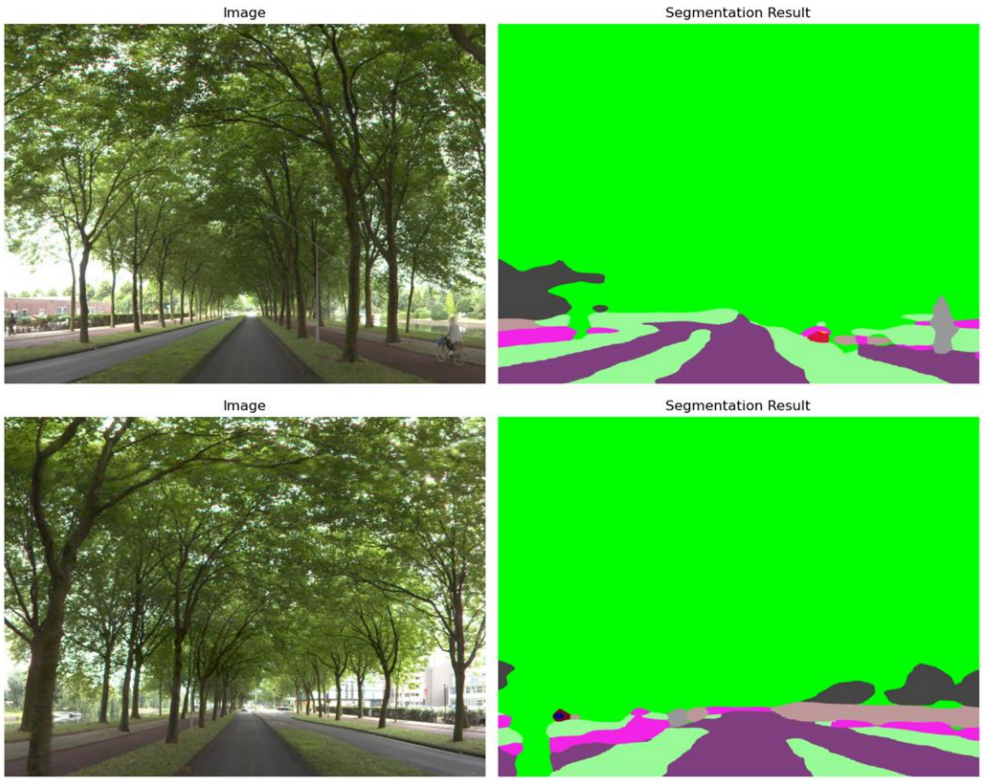

Figure 10. Segmentation of panoramic streetview image taken in Amsterdam, GVI value: 0.801

3.3.1. get_viewshed_GVI()

Suppose that we want to retrieve the average GVI value for road locations that fall within a 100-meter radius of the PoIs. Moreover, we consider a viewing distance range of 250 meters, an observer height of 1.7 meters and the sample road locations should be spaced at intervals of 50 meters. This requires additional files to be passed to the function and can be achieved by applying the code from code snippet 9.

Code snippet 9

```
greendata_binary_file = "C:/Users/ygrin/Example_data/AMS_greenspace_binary.tif"
dtm_file = "C:/Users/ygrin/Example_data/AMS_dtm.tif"
dsm_file = "C:/Users/ygrin/Example_data/AMS_dsm.tif"

visibility.get_viewshed_GVI(point_of_interest_file=poi_file,
                            greendata_raster_file=greendata_binary_file,
                            dtm_raster_file=dtm_file,
                            dsm_raster_file=dsm_file,
                            buffer_dist=100,
                            viewing_dist=250,
                            sample_dist=50,
                            observer_height=1.7,
                            write_to_file=False)
```

While running, the function provided feedback on which steps were being executed and the time spent;

```
Retrieving network within total bounds of Point(s) of interest, extended by the buffer_dist in case provided...
Done, running time: 0:00:28.402266

Computing sample points for roads within area of interest's network...
Note: creation of sample points based on code by Ondrej Mlynarcik
```

```
source: https://github.com/Spatial-Data-Science-and-GEO-AI-Lab/2.5D-GreenViewIndex-Netherlands/blob/main/sample_points_linestrings.ipynb
Done, running time: 0:00:00.160945

Calculating GVI for Point 1: 100%
117/117 [00:15<00:00, 7.82it/s]
Calculating GVI for Point 2: 100%
86/86 [00:11<00:00, 8.53it/s]
Calculating GVI for Point 3: 100%
48/48 [00:06<00:00, 7.09it/s]
Note: calculation of Viewshed GVI based on code by Johnny Huck and Labib
source: https://github.com/jonnyhuck/green-visibility-index/blob/master/gvi.py
```

This confirms that sample road locations are computed within the vicinity of the PoIs. Consequently, the viewshed analysis is based on these locations. The function results are split into two dataframes that are presented in Tables 8 and 9, respectively. The first dataframe contains the original PoIs, the average GVI values and the number of sample road locations upon which these are based. The second dataframe contains the sample road locations and their respective GVI values.

Table 8. *get_viewshed_GVI()* function output pt. 1 (PoIs)

	geometry	id	GVI	nr_of_points
0	POINT (118883.345 485054.641)	1	0.276	117
1	POINT (118246.855 488082.089)	2	0.231	86
2	POINT (122483.550 487728.517)	3	0.024	48

Table 9. *get_viewshed_GVI()* function output pt. 2 (sample road locations)

	id	geometry	GVI
0	1	POINT (118602.894 485014.928)	0.157408
1	1	POINT (118656.183 485033.863)	0.185535
2	1	POINT (118708.299 485057.627)	0.159684
...	...	...	...
248	3	POINT (122446.453 487664.938)	0.0
249	3	POINT (122441.012 487732.964)	0.007802
250	3	POINT (122467.668 487726.301)	0.0

Visualizations are not supported by this function. However, to give a visual representation of the study area, Figure 11 was created using an overlay of the DSM and the binary green data file. As expected, the average GVI values are higher for locations that are surrounded by greenness (point id 1 and 2).

Figure 11. Visual representation of *get_viewshed_GVI()* study area

4. Impact and discussion

The primary goal of the 'GreenEx_Py' package is to provide researchers with a tool that is both user-friendly and open source, enabling them to conduct thorough and transparent analyses pertaining to multidimensional greenspace exposure while adhering to FAIR(4RS) principles. Example analyses may investigate the impact of exposure to greenspace on house prices [45], [46] or public health outcomes such as mental well-being and physical activity levels [47], [48].

The package specifically showcases strength in its usage of global open-access data and ability to assess eye-level greenness visibility. Also, the accessibility function in particular supports policy directives regarding the selection of size of greenspace and the distance to greenspace as issued by the World Health Organization [49]. Moreover, the package can be used both locally and through the free-to-use cloud platform of Google Colab.

The illustrative examples demonstrate the successful execution of the functions in accordance with their intended purposes. Nevertheless, it should be noted that the package is currently in an ongoing development phase, and as such, it possesses inherent limitations that need to be considered.

First, the functions *get_mean_ndvi()* and *get_landcover_percentages()* currently rely on the planetary computer API if no raster files are provided by the user. Depending on the study area, this might not be adequate as it retrieves satellite images that cover a specific region. Therefore, all PoIs should fall within that image's region, captured using a tile. This aspect can be accounted for by integrating the Google Earth Engine into the two functions as it allows tiles to be merged prior to further analysis.

Second, greenspace accessibility is currently assessed using a single function that does not account for aspects like supply and demand as highlighted by Bart Breekveldt [36]. To get a more comprehensive understanding of greenspace accessibility, an additional function should be developed that does consider the supply and demand related to greenspace areas.

Third, the availability and accessibility functions do not yet support network files to be provided by users. Enabling users to provide local networks may improve the analysis findings as the road coverage provided by OSM may be inaccurate for specific areas. This introduces a limitation that applies to all package functions given the potential variability in data quality across open-access databases and APIs such as the planetary computer, OSM and Mapillary.

Lastly, the package functions are not yet optimized to process large amount of data. Therefore, the usage of many PoIs (>5000) may lead to extensive processing times depending on the accompanying parameter values.

5. Conclusions

In this software paper, we introduce the open-source Python package 'GreenEx_Py'. The package includes diverse functions which enable researchers to model greenspace exposure from three perspectives; availability, accessibility and visibility. We strongly believe that this package aligns with FAIR(4RS) principles by effectively addressing barriers related to transparent, reproducible and replicable research. We achieve this by utilizing various open-access databases and APIs, thoroughly documenting function processes, and openly sharing data used throughout the testing phase. Efficiency and accuracy of the package functions may be further enhanced by accounting for the known limitations of this moment.

References

[1] J. N. Georgi and D. Dimitriou, 'The contribution of urban green spaces to the improvement of environment in cities: Case study of Chania, Greece', *Build Environ*, vol. 45, no. 6, pp. 1401–1414, Jun. 2010, doi: 10.1016/J.BUILDENV.2009.12.003.

[2] W. Selmi, C. Weber, E. Rivière, N. Blond, L. Mehdi, and D. Nowak, 'Air pollution removal by trees in public green spaces in Strasbourg city, France', *Urban For Urban Green*, vol. 17, pp. 192–201, Jun. 2016, doi: 10.1016/J.UFUG.2016.04.010.

[3] M. Razzaghmanesh, S. Beecham, and T. Salemi, 'The role of green roofs in mitigating Urban Heat Island effects in the metropolitan area of Adelaide, South Australia', *Urban For Urban Green*, vol. 15, pp. 89–102, Jan. 2016, doi: 10.1016/J.UFUG.2015.11.013.

[4] C. Wang *et al.*, 'Efficient cooling of cities at global scale using urban green space to mitigate urban heat island effects in different climatic regions', *Urban For Urban Green*, vol. 74, p. 127635, Aug. 2022, doi: 10.1016/J.UFUG.2022.127635.

[5] S. M. Labib, S. Lindley, and J. J. Huck, 'Estimating multiple greenspace exposure types and their associations with neighbourhood premature mortality: A socioecological study', *Science of The Total Environment*, vol. 789, p. 147919, Oct. 2021, doi: 10.1016/j.scitotenv.2021.147919.

[6] M. Kondo, J. Fluehr, T. McKeon, and C. Branas, 'Urban Green Space and Its Impact on Human Health', *Int J Environ Res Public Health*, vol. 15, no. 3, p. 445, Mar. 2018, doi: 10.3390/ijerph15030445.

[7] M. P. Jimenez *et al.*, 'Associations between Nature Exposure and Health: A Review of the Evidence', *Int J Environ Res Public Health*, vol. 18, no. 9, p. 4790, Apr. 2021, doi: 10.3390/ijerph18094790.

[8] D. Rojas-Rueda, M. J. Nieuwenhuijsen, M. Gascon, D. Perez-Leon, and P. Mudu, 'Green spaces and mortality: a systematic review and meta-analysis of cohort studies', *Lancet Planet Health*, vol. 3, no. 11, pp. e469–e477, Nov. 2019, doi: 10.1016/S2542-5196(19)30215-3.

[9] P. Dadvand *et al.*, 'Green spaces and cognitive development in primary schoolchildren', *Proceedings of the National Academy of Sciences*, vol. 112, no. 26, pp. 7937–7942, Jun. 2015, doi: 10.1073/pnas.1503402112.

[10] K. C. Fong, J. E. Hart, and P. James, 'A Review of Epidemiologic Studies on Greenness and Health: Updated Literature Through 2017', *Curr Environ Health Rep*, vol. 5, no. 1, pp. 77–87, Mar. 2018, doi: 10.1007/s40572-018-0179-y.

[11] P. Dadvand and M. Nieuwenhuijsen, 'Green Space and Health', in *Integrating Human Health into Urban and Transport Planning*, Cham: Springer International Publishing, 2019, pp. 409–423. doi: 10.1007/978-3-319-74983-9_20.

[12] Z. Yu *et al.*, 'A simple but actionable metric for assessing inequity in resident greenspace exposure', *Ecol Indic*, vol. 153, p. 110423, Sep. 2023, doi: 10.1016/j.ecolind.2023.110423.

[13] G. N. Bratman *et al.*, 'Nature and mental health: An ecosystem service perspective', *Sci Adv*, vol. 5, no. 7, Jul. 2019, doi: 10.1126/sciadv.aax0903.

[14] M. Buckland and D. Pojani, 'Green space accessibility in Europe: a comparative study of five major cities', *European Planning Studies*, vol. 31, no. 1, pp. 146–167, Jan. 2023, doi: 10.1080/09654313.2022.2088230.

[15] E. D. Ekkel and S. de Vries, ‘Nearby green space and human health: Evaluating accessibility metrics’, *Landsc Urban Plan*, vol. 157, pp. 214–220, Jan. 2017, doi: 10.1016/j.landurbplan.2016.06.008.

[16] E.-H. Yoo, J. E. Roberts, Y. Eum, X. Li, and K. Konty, ‘Exposure to urban green space may both promote and harm mental health in socially vulnerable neighborhoods: A neighborhood-scale analysis in New York City’, *Environ Res*, vol. 204, p. 112292, Mar. 2022, doi: 10.1016/j.envres.2021.112292.

[17] S. M. Labib, S. Lindley, and J. J. Huck, ‘Spatial dimensions of the influence of urban green-blue spaces on human health: A systematic review’, *Environ Res*, vol. 180, p. 108869, Jan. 2020, doi: 10.1016/j.envres.2019.108869.

[18] S. T. Brinkmann, D. Kremer, and B. B. Walker, ‘Modelling eye-level visibility of urban green space: Optimising city-wide point-based viewshed computations through prototyping’, *AGILE: GIScience Series*, vol. 3, pp. 1–7, Jun. 2022, doi: 10.5194/agile-giss-3-27-2022.

[19] A. Pallathadka, L. Pallathadka, S. Rao, H. Chang, and D. Van Dommelen, ‘Using GIS-based spatial analysis to determine urban greenspace accessibility for different racial groups in the backdrop of COVID-19: a case study of four US cities’, *GeoJournal*, vol. 87, no. 6, pp. 4879–4899, Dec. 2022, doi: 10.1007/s10708-021-10538-8.

[20] A. Gou *et al.*, ‘Spatial association between green space and COPD mortality: a township-level ecological study in Chongqing, China’, *BMC Pulm Med*, vol. 23, no. 1, p. 89, Mar. 2023, doi: 10.1186/s12890-023-02359-x.

[21] W. Shih, ‘Greenspace patterns and the mitigation of land surface temperature in Taipei metropolis’, *Habitat Int*, vol. 60, pp. 69–80, Feb. 2017, doi: 10.1016/j.habitatint.2016.12.006.

[22] A. Larkin and P. Hystad, ‘Evaluating street view exposure measures of visible green space for health research’, *J Expo Sci Environ Epidemiol*, vol. 29, no. 4, pp. 447–456, Jul. 2019, doi: 10.1038/s41370-018-0017-1.

[23] A. C. O’Regan, R. F. Hunter, and M. M. Nyhan, ‘“Biophilic Cities”: Quantifying the Impact of Google Street View-Derived Greenspace Exposures on Socioeconomic Factors and Self-Reported Health’, *Environ Sci Technol*, vol. 55, no. 13, pp. 9063–9073, Jul. 2021, doi: 10.1021/acs.est.1c01326.

[24] S. M. Labib, J. J. Huck, and S. Lindley, ‘Modelling and mapping eye-level greenness visibility exposure using multi-source data at high spatial resolutions’, *Science of The Total Environment*, vol. 755, p. 143050, Feb. 2021, doi: 10.1016/j.scitotenv.2020.143050.

[25] M. Fleischmann, A. Feliciotti, and W. Kerr, ‘Evolution of Urban Patterns: Urban Morphology as an Open Reproducible Data Science’, *Geogr Anal*, vol. 54, no. 3, pp. 536–558, Jul. 2022, doi: 10.1111/gean.12302.

[26] C. Brunsdon and A. Comber, ‘Opening practice: supporting reproducibility and critical spatial data science’, *J Geogr Syst*, vol. 23, no. 4, pp. 477–496, Oct. 2021, doi: 10.1007/s10109-020-00334-2.

[27] R. Harris *et al.*, ‘More bark than bytes? Reflections on 21+ years of geocomputation’, *Environ Plan B Urban Anal City Sci*, vol. 44, no. 4, pp. 598–617, Jul. 2017, doi: 10.1177/2399808317710132.

[28] G. Boeing, ‘The right tools for the job: The case for spatial science tool-building’, *Transactions in GIS*, vol. 24, no. 5, pp. 1299–1314, Oct. 2020, doi: 10.1111/tgis.12678.

[29] M. D. Wilkinson *et al.*, 'The FAIR Guiding Principles for scientific data management and stewardship', *Sci Data*, vol. 3, no. 1, p. 160018, Mar. 2016, doi: 10.1038/sdata.2016.18.

[30] M. Barker *et al.*, 'Introducing the FAIR Principles for research software', *Sci Data*, vol. 9, no. 1, p. 622, Oct. 2022, doi: 10.1038/s41597-022-01710-x.

[31] G. van Rossum and J. de Boer, 'Interactively testing remote servers using the Python programming language', *CWI Quarterly*, vol. 4, no. 4, pp. 283–304, 1991.

[32] A. D. Snow *et al.*, 'corteva/rioxarray: 0.14.1 Release', Apr. 2023, doi: 10.5281/ZENODO.7829704.

[33] M. O. Source, M. McFarland, R. Emanuele, D. Morris, and T. Augspurger, 'microsoft/PlanetaryComputer: October 2022', Oct. 2022, doi: 10.5281/ZENODO.7261897.

[34] G. Boeing, 'OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks', *Comput Environ Urban Syst*, vol. 65, pp. 126–139, Sep. 2017, doi: 10.1016/j.compenvurbsys.2017.05.004.

[35] D. Zanaga *et al.*, 'ESA WorldCover 10 m 2021 v200', Oct. 2022, doi: 10.5281/ZENODO.7254221.

[36] B. E. Breekveldt and S. M. Labib, 'Modelling greenspace accessibility at multi-spatial contexts: a pilot study of comparing the E2SFCA and Gravity models', Apr. 2023, doi: 10.5281/ZENODO.7823447.

[37] J. Li *et al.*, 'Comparing effects of Euclidean buffers and network buffers on associations between built environment and transport walking: the Multi-Ethnic Study of Atherosclerosis', *Int J Health Geogr*, vol. 21, no. 1, p. 12, Sep. 2022, doi: 10.1186/s12942-022-00310-7.

[38] A. A. Hagberg, D. A. Schult, and P. J. Swart, 'Exploring Network Structure, Dynamics, and Function using NetworkX', in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[39] P. Virtanen *et al.*, 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', *Nat Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[40] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, 'Masked-attention Mask Transformer for Universal Image Segmentation'. 2022.

[41] M. Cao, S. Liu, and F. Cao, 'Midpoint Distance Circle Generation Algorithm based on Midpoint Circle Algorithm and Bresenham Circle Algorithm', *J Phys Conf Ser*, vol. 1438, no. 1, p. 012017, Jan. 2020, doi: 10.1088/1742-6596/1438/1/012017.

[42] J. Van Aken, 'An Efficient Ellipse-Drawing Algorithm', *IEEE Comput Graph Appl*, vol. 4, no. 9, pp. 24–35, 1984, doi: 10.1109/MCG.1984.275994.

[43] J. E. Bresenham, 'Algorithm for computer control of a digital plotter', *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965, doi: 10.1147/sj.41.0025.

[44] J. Bresenham, 'A linear algorithm for incremental digital display of circular arcs', *Commun ACM*, vol. 20, no. 2, pp. 100–106, Feb. 1977, doi: 10.1145/359423.359432.

[45] M. N. Daams, F. J. Sijtsma, and P. Veneri, 'Mixed monetary and non-monetary valuation of attractive urban green space: A case study using Amsterdam house prices', *Ecological Economics*, vol. 166, p. 106430, Dec. 2019, doi: 10.1016/j.ecolecon.2019.106430.

[46] C. Wu, Y. Du, S. Li, P. Liu, and X. Ye, ‘Does visual contact with green space impact housing prices? An integrated approach of machine learning and hedonic modeling based on the perception of green space’, *Land use policy*, vol. 115, p. 106048, Apr. 2022, doi: 10.1016/j.landusepol.2022.106048.

[47] V. Gianfredi *et al.*, ‘Association between Urban Greenspace and Health: A Systematic Review of Literature’, *Int J Environ Res Public Health*, vol. 18, no. 10, p. 5137, May 2021, doi: 10.3390/ijerph18105137.

[48] T. Astell-Burt and X. Feng, ‘Association of Urban Green Space With Mental Health and General Health Among Adults in Australia’, *JAMA Netw Open*, vol. 2, no. 7, p. e198209, Jul. 2019, doi: 10.1001/jamanetworkopen.2019.8209.

[49] W. H. Organization. R. O. for Europe, ‘Urban green spaces and health’, World Health Organization. Regional Office for Europe, 2016.

[50] K. Jordahl *et al.*, ‘geopandas/geopandas: v0.8.1’. Zenodo, Jul. 2020. doi: 10.5281/zenodo.3946761.

[51] S. Gillies *et al.*, ‘Shapely’. Jan. 2023. doi: 10.5281/zenodo.5597138.

[52] S. der Walt *et al.*, ‘scikit-image: image processing in Python’, *PeerJ*, vol. 2, p. e453, 2014.

[53] T. Wolf *et al.*, ‘Transformers: State-of-the-Art Natural Language Processing’, in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

Appendix I. Fundamental packages for developing GreenEx_Py

Table 10. Fundamental packages for GreenEx_Py

Package	Description	Source
Geopandas	Powerful package for handling geospatial data	[50]
Networkx	Package for studying complex networks	[38]
Osmnx	Package for retrieving and analyzing OSM networks	[34]
Planetary_computer	Package to interact with Microsoft Planetary Computer	[33]
Rioxarray	Package enabling spatial operations on raster data	[32]
Scipy	Scientific computing library for mathematical tasks	[39]
Shapely	Package to manipulate and analyze geometric objects	[51]
Skimage	Library for image processing	[52]
Transformers	APIs and tools for downloading pre-trained models	[53]

Appendix II. Installation manual for Windows

Cloning GitHub repository using Git Bash
For additional information, please check this link

Prerequisites:
- Git Bash installed

Steps:
1. On GitHub.com, navigate to the main page of the repository.
https://github.com/Spatial-Data-Science-and-GEO-AI-Lab/GreenEx_Py
2. Above the list of files, click Code.
3. Copy the URL for the repository.
4. Open Git Bash.
5. Change the current working directory to the location where you want the cloned directory.
6. Type git clone, and then paste the URL you copied earlier.
$ git clone https://github.com/Spatial-Data-Science-and-GEO-AI-Lab/GreenEx_Py.git
7. Press Enter to create your local clone.

Cloning GitHub repository using GitHub Desktop app
For additional information, please check this link

Prerequisites:
- GitHub desktop installed

Steps:
1. On GitHub.com, navigate to the main page of the repository.
https://github.com/Spatial-Data-Science-and-GEO-AI-Lab/GreenEx_Py
2. Above the list of files, click Code.
3. To clone and open the repository with GitHub Desktop, click Open with GitHub Desktop.
4. Follow the prompts in GitHub Desktop to complete the clone.

Installing and using conda environment
For additional information, please check this link

Prerequisites:
- Either Anaconda or Miniconda installed
- GitHub repository is already cloned

Steps:
1. Open anaconda or miniconda prompt
2. Navigate to the cloned GitHub repository directory using ‘cd’ command, example;
cd Documents/GitHub/GreenEx_Py
3. Create the environment from the GreenExpPy.yml file:
conda env create -f GreenExpPy.yml
4. Activate the new environment:
conda activate Greenex_py
5. Open jupyter notebook by typing: jupyter notebook
6. When jupyter notebook is opened, navigate to cloned GitHub repository
7. Use example.ipynb as an example to run the code

Appendix III. Installation manual for Mac

Cloning GitHub repository
For additional information, please check this link

Steps:
1. Open the main page of the repository in browser. click Clone or download.
https://github.com/Spatial-Data-Science-and-GEO-AI-Lab/GreenEx_Py
2. Above the list of files, click Code.
3. Copy the URL for the repository.
4. Open Terminal on your mac.
5. Type “cd” and the directory where you want the cloned directory to be made. You can right-click the folder in Finder and choose “Copy <the folder name>” to copy the path into clipboard. Then by pressing “Command” and “v” on your keyboard to paste the path into terminal.
6.Type “git clone”, and then paste the URL you copied in step 2. Press Enter. The local clone will be created.

Installing and using conda environment
For additional information, please check this link

Prerequisites:
- Either Anaconda or Miniconda installed
- GitHub repository is already cloned

Steps:
1. Open Terminal
2. Navigate to the cloned GitHub repository directory using ‘cd’ command, example;
cd Documents/GitHub/GreenEx_Py
3. Create the environment from the GreenExpPy_Mac.yml file:
conda env create -f GreenExpPy_Mac.yml
4. Activate the new environment:
conda activate greenexp_py
5. Open jupyter notebook by typing: jupyter notebook
6. When jupyter notebook is opened, navigate to cloned GitHub repository
7. Use example.ipynb as an example to run the code

NOTE: Local issues may be encountered when running the module locally on a Mac device, as was the case during testing. The module is developed on a Windows device and a separate yml file, containing the python environment, had to be created for Mac.

Appendix IV. Specifications for *get_mean_ndvi()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries around and for which to compute mean NDVI values.
- ndvi_raster_file *(string)* – optional, the absolute or relative path to the file containing the NDVI values in raster format. If not provided, the NDVI raster will be obtained by using satellite images from the planetary computer.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- polygon_type *(string {"neighbourhood", "house"})* - to be defined in case point_of_interest_file contains polygon geometries.
- buffer_type *(string {"euclidean", "network"})* – to be defined in case point_of_interest_file contains point geometries and optional in case point_of_interest_file contains polygon geometries, the way in which the area of interest should be composed. If "euclidean", a straight line distance will be used based on the buffer distance as specified by the buffer_dist argument. If "network", isochrone maps will be composed based on the buffer distance or combination of trip_time and travel_speed.
- buffer_dist *(int)* – to be defined if buffer_type is set to "euclidean" and optional if buffer_type is set to "network". The distance in meters that will be used to compute the area of interest surrounding the geometries of the point_of_interest file. NOTE: In case buffer_type is set to "network", buffer_dist should not be specified if travel_speed and trip_time arguments are specified.
- network_type *(string {"walk", "bike", "drive", "all"})* – to be defined in case buffer_type is set to "network", the travel mode for which the network needs to be retrieved from OpenStreetMap.
- trip_time *(int)* – may optionally be defined in case buffer_type is set to "network", trip time in minutes to consider for travel mode specified in network_type. The trip_time, in addition to the travel_speed, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- travel_speed *(float, int)* – may optionally be defined in case buffer_type is set to "network", travel speed in km/h to consider for travel mode specified in network_type. The travel_speed, in addition to the trip_time, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- year *(int)* – optional, may be defined if no ndvi raster file is provided. The year for which to retrieve satellite images using the planetary computer. If not specified while raster file is not provided, the current year will be used by default.
- plot_aoi *(bool {"TRUE", "FALSE"})* - whether or not to plot the areas of interest that have been used for the mean NDVI calculation. If set to TRUE (default), the plot will be shown as part of the function execution.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- save_ndvi *(bool {"TRUE", "FALSE"})* - whether or not to save the satellite image retrieved from planetary computer and NDVI raster that was created afterwards in case NDVI raster was not provided. Files will be written to a new file in the directory specified in the output_dir argument if set to TRUE. By default, argument is set to TRUE.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including column for mean NDVI value(s). Dataframe will also be written to new file in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframe

Appendix V. Specifications for *get_landcover_percentages()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries around and for which to calculate the percentage of area covered by each landcover class.
- landcover_raster_file *(string)* – optional, the absolute or relative path to the raster file containing a landcover classification map, where each pixel is assigned a landcover class. If not provided, the landcover map from the European Space Agency (ESA) will be retrieved through the planetary computer. Note that this map may be outdated and contain missing values.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- polygon_type *(string {"neighbourhood", "house"})* - to be defined in case point_of_interest_file contains polygon geometries.
- buffer_type *(string {"euclidean", "network"})* – to be defined in case point_of_interest_file contains point geometries and optional in case point_of_interest_file contains polygon geometries, the way in which the area of interest should be composed. If "euclidean", a straight line distance will be used based on the buffer distance as specified by the buffer_dist argument. If "network", isochrone maps will be composed based on the additional arguments of trip_time and travel_speed.
- buffer_dist *(int)* – to be defined if buffer_type is set to "euclidean" and optional if buffer_type is set to "network". The distance in meters that will be used to compute the area of interest surrounding the geometries of the point_of_interest file. NOTE: In case buffer_type is set to "network", buffer_dist should not be specified if travel_speed and trip_time arguments are specified.
- network_type *(string {"walk", "bike", "drive", "all"})* – to be defined in case buffer_type is set to "network", the travel mode for which the network needs to be retrieved from OpenStreetMap.
- trip_time *(int)* – may optionally be defined in case buffer_type is set to "network", trip time in minutes to consider for travel mode specified in network_type. The trip_time, in addition to the travel_speed, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- travel_speed *(float, int)* – may optionally be defined in case buffer_type is set to "network", travel speed in km/h to consider for travel mode specified in network_type. The travel_speed, in addition to the trip_time, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- plot_aoi *(bool {"TRUE", "FALSE"})* - whether or not to plot the areas of interest that have been used for the landcover class percentage calculation. If set to TRUE (default), the plot will be shown as part of the function execution.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- save_lulc *(bool {"TRUE", "FALSE"})* - whether or not to save the landcover class image retrieved from planetary computer in case landcover raster was not provided. File will be written to a new file in the directory specified in the output_dir argument if set to TRUE. By default, argument is set to TRUE.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including columns for landcover class percentages. Dataframe will also be written to new file in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframe

Appendix VI. Specifications for *get_canopy_percentage()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries around and for which to compute the greenspace cover percentage.
- canopy_vector_file *(string)* – the absolute or relative path to the vector file containing tree canopy data. Note that geometries should be polygon or multipolygon.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- polygon_type *(string {"neighbourhood", "house"})* - to be defined in case point_of_interest_file contains polygon geometries.
- buffer_type *(string {"euclidean", "network"})* – to be defined in case point_of_interest_file contains point geometries and optional in case point_of_interest_file contains polygon geometries, the way in which the area of interest should be composed. If "euclidean", a straight line distance will be used based on the buffer distance as specified by the buffer_dist argument. If "network", isochrone maps will be composed based on the additional arguments of trip_time and travel_speed.
- buffer_dist *(int)* – to be defined if buffer_type is set to "euclidean" and optional if buffer_type is set to "network". The distance in meters that will be used to compute the area of interest surrounding the geometries of the point_of_interest file. NOTE: In case buffer_type is set to "network", buffer_dist should not be specified if travel_speed and trip_time arguments are specified.
- network_type *(string {"walk", "bike", "drive", "all"})* – to be defined in case buffer_type is set to "network", the travel mode for which the network needs to be retrieved from OpenStreetMap.
- trip_time *(int)* – may optionally be defined in case buffer_type is set to "network", trip time in minutes to consider for travel mode specified in network_type. The trip_time, in addition to the travel_speed, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- travel_speed *(float, int)* – may optionally be defined in case buffer_type is set to "network", travel speed in km/h to consider for travel mode specified in network_type. The travel_speed, in addition to the trip_time, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- plot_aoi *(bool {"TRUE", "FALSE"})* - whether or not to plot the areas of interest that have been used for the tree canopy cover calculation. If set to TRUE (default), the plot will be shown as part of the function execution.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including column for tree canopy cover percentage. Dataframe will also be written to new file in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframe

Appendix VII. Specifications for *get_greenspace_percentage()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries around and for which to compute the percentage of greenspace cover.
- greenspace_vector_file *(string)* – optional, the absolute or relative path to the vector file containing greenspace data. Note that geometries should be polygon or multipolygon. In case no file is provided, greenspace data will be retrieved from OpenStreetMap.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- polygon_type *(string {"neighbourhood", "house"})* - to be defined in case point_of_interest_file contains polygon geometries.
- buffer_type *(string {"euclidean", "network"})* – to be defined in case point_of_interest_file contains point geometries and optional in case point_of_interest_file contains polygon geometries, the way in which the area of interest should be composed. If "euclidean", a straight line distance will be used based on the buffer distance as specified by the buffer_dist argument. If "network", isochrone maps will be composed based on the additional arguments of trip_time and travel_speed.
- buffer_dist *(int)* – to be defined if buffer_type is set to "euclidean" and optional if buffer_type is set to "network". The distance in meters that will be used to compute the area of interest surrounding the geometries of the point_of_interest file. NOTE: In case buffer_type is set to "network", buffer_dist should not be specified if travel_speed and trip_time arguments are specified.
- network_type *(string {"walk", "bike", "drive", "all"})* – to be defined in case buffer_type is set to "network", the travel mode for which the network needs to be retrieved from OpenStreetMap.
- trip_time *(int)* – may optionally be defined in case buffer_type is set to "network", trip time in minutes to consider for travel mode specified in network_type. The trip_time, in addition to the travel_speed, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- travel_speed *(float, int)* – may optionally be defined in case buffer_type is set to "network", travel speed in km/h to consider for travel mode specified in network_type. The travel_speed, in addition to the trip_time, will be used to compose an isochrone map if no buffer distance is provided. NOTE: travel_speed and trip_time arguments should not be specified if buffer_dist is specified.
- plot_aoi *(bool {"TRUE", "FALSE"})* - whether or not to plot the areas of interest that have been used for the greenspace cover calculation. If set to TRUE (default), the plot will be shown as part of the function execution.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including column for greenspace cover percentage. Dataframe will also be written to new file in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframe

Appendix VIII. Specifications for *get_shortest_distance_greenspace()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries for which to compute the shortest distance to greenspaces.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- target_dist *(int)* – threshold distance in meters surrounding the point locations in which greenspaces should be located. If no greenspace destination is detected within threshold distance for the current parameters, the distance to greenspace will be set to the given threshold distance. The actual distance to the nearest greenspace cannot always be retrieved properly due to edge effects concerning the area for which the network and greenspaces are retrieved through OpenStreetMap. A warning is also provided in this case, as it should be considered for further analysis.
- greenspace_vector_file *(string)* – optional, the absolute or relative path to the vector file containing greenspace data. In case no file is provided, greenspace data will be retrieved from OpenStreetMap.
- distance_type *(string {"euclidean", "network"})* – the way in which the shortest distance to a greenspace should be calculated, straight line distance (euclidean) or network distance (network). By default, argument is set to euclidean. Note that if set to network, processing times might increase significantly in case many points/polygons of interest are provided in the point_of_interest file.
- destination *(string {"centroids", "entrance"})* – the destination points for the greenspaces. If "entrance", network distances will be computed between the point location's nearest network node and the network nodes that are within 20 meters of the greenspace boundaries - therefore considered as pseudo-entry points. Euclidean distance between point location and nearest network node will be added. If "centroids", the same procedure applies while also taking into account the euclidean distance between the greenspace's centroid and pseudo-entry points.
- network_type *(string {"walk", "bike", "drive", "all"})* – to be defined in case buffer_type is set to "network", the travel mode for which the network needs to be retrieved from OpenStreetMap.
- min_greenspace_area *(int)* – optional, if set to integer value, only greenspaces with an area greater than or equal to the min_greenspace_area in squared meters will be considered for the shortest distance calculation.
- plot_aoi *(bool {"TRUE", "FALSE"})* - whether or not to plot the areas of interest that have been used for the shortest distance calculation. If set to TRUE (default), the plot will be shown as part of the function execution.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including column indicating whether or not a greenspace is within specified target distance. Dataframe will also be written to new file in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframe

Appendix IX. Specifications for *get_streetview_GVI()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries around and for which to compute the average GVI value.
- access_token *(string)* – Mapillary API token that is required to access the streetview images of Mapillary.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- polygon_type *(string {"neighbourhood", "house"})* - to be defined in case point_of_interest_file contains polygon geometries. In case set to "neighbourhood", buffer_dist argument is optional and if not specified, GVI values will be calculated for road network locations within each polygon geometry. If set to "house", network will be retrieved from OpenStreetMap and based on buffer_dist if not provided.
- buffer_dist *(int)* – to be defined in case point_of_interest_file contains point geometries and optional in case point_of_interest_file contains polygon geometries, the point/polygon of interest surrounding distance in meters that should be considered when defining road network locations for calculating GVI values.
- workers *(int)* – the maximum number of concurrent worker threads to use (i.e. simultaneous tasks that can be executed), providing control over the level of concurrency in the function.
- network_file *(string)* – optional, the absolute or relative path to the file containing the network (transportation infrastructure) to consider. If not specified, the network will be retrieved through OpenStreetMap considering the point/polygon's surrounding straight line distance as specified in the buffer_dist argument.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including columns for mean GVI value(s) and the number of points upon which these are based. A second dataframe will be returned and contains the surrounding road network locations (of the original points/polygons of interest) that were used to calculate the GVI values. The ID column is used to identify the original point/polygon of interest. Both dataframes will also be written to new files in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframes

Appendix X. Specifications for *get_viewshed_GVI()*

Parameters:

- point_of_interest_file *(string)* – the absolute or relative path to the file containing point or polygon geometries around and for which to compute the average GVI value.
- greendata_raster_file *(string)* – the absolute or relative path to the boolean raster file indicating which cells are considered greenspace and which are not.
- dtm_raster_file *(string)* – the absolute or relative path to the raster file containing the Digital Terrain Model.
- dsm_raster_file *(string)* – the absolute or relative path to the raster file containing the Digital Surface Model.
- network_file *(string)* – optional, the absolute or relative path to the file containing the network (transportation infrastructure) to consider. If not specified, the network will be retrieved through OpenStreetMap considering the point/polygon's surrounding straight line distance as specified in the buffer_dist argument.
- crs_epsg *(int)* - optional, to be defined in case provided point of interest file has geographic CRS rather than projected. CRS will be transformed to the projected CRS that is specified. In case crs_epsg is not specified and CRS of file is geographic, CRS will be transformed to EPSG 3395 by default.
- polygon_type *(string {"neighbourhood", "house"})* - to be defined in case point_of_interest_file contains polygon geometries. In case set to "neighbourhood", buffer_dist argument is optional and if not specified, GVI values will be calculated for road network locations within each polygon geometry. If set to "house", network will be retrieved from OpenStreetMap and based on buffer_dist if not provided.
- buffer_dist *(int)* – to be defined in case point_of_interest_file contains point geometries and optional in case point_of_interest_file contains polygon geometries, the point/polygon of interest surrounding distance in meters that should be considered when defining road network locations for calculating GVI values.
- viewing_dist *(int)* - the viewing distance in meters to consider when composing the viewshed.
- sample_dist *(float, int)* - the interval in meters that is used to define road network locations. For each road that is within the area of interest, point locations will be generated using an interval of sample_dist meters. If a road is too short, the road's center location will be used.
- observer_height *(float, int)* - a person's height in meters to consider for the creation of viewsheds and lines of sight.
- write_to_file *(bool {"TRUE", "FALSE"})* - whether or not to write the results to a new file in the directory specified in the output_dir argument. By default, results will be written to file.
- output_dir *(string)* – the absolute or relative path to the directory in which the output file will be written in case write_to_file is set to TRUE. If not specified, the current working directory will serve as default.

Returns:
> Dataframe as obtained from point_of_interest_file including columns for mean GVI value(s) and the number of points upon which these are based. A second dataframe will be returned and contains the surrounding road network locations (of the original points/polygons of interest) that were used to calculate the GVI values. The ID column is used to identify the original point/polygon of interest. Both dataframes will also be written to new files in specified directory (see output_dir argument) if write_to_file set to TRUE.

Return type:

> Geodataframes