MGWR Python Package Update - Overall Description

Update MGWR support version from Python version 3.6 to version 3.10 or higher.

Reason: The current MGWR codebase is on top of Python version 3.5 or 3.6, I will utilize the advantages of new released versions to improve code readability and efficiency.

Table of Content

Updated MGWR Minimum Supported Version To Python 3.10

Reasons for Updating Python Version

Python is part of one of the fastest-growing and most active communities in the programming world. With each new version release, there comes a wave of updates and enhancements. Comparing Python 3.6 to Python 3.10, (Python 3.11 boasts a remarkable 64% improvement: Python is About to Become 64% Faster) , there are compelling reasons to consider upgrading our supported Python version. This transition not only aligns us with the dynamic Python community but also delivers substantial performance gains, ensuring our code runs efficiently and effectively in this evolving landscape.

Official Support for Python Versions

As Python version 3.7 or early versions have reached their end-of-life (figure below), it's essential to prioritize a supported and stable version. Python 3.8 is the earliest version currently maintained by the Python Software Foundation (PSF) and offers both security and stability. Making the transition to Python 3.10 ensures that we benefit from ongoing support and maintenance, reducing vulnerabilities and enhancing the reliability of our Python environment.

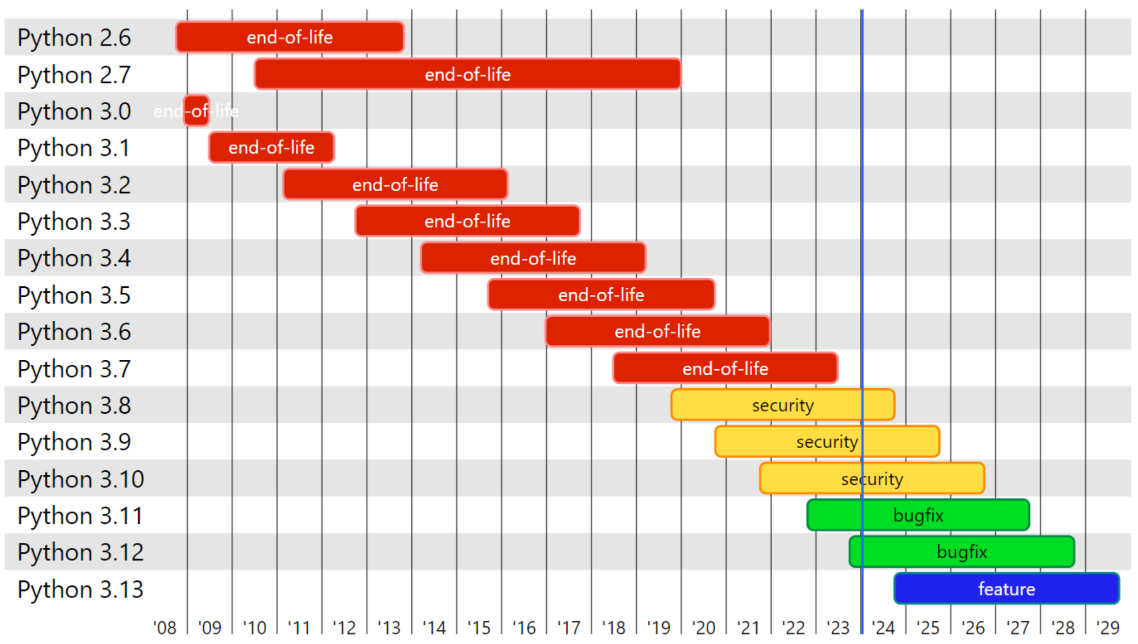
Rationale

The existing MGWR codebase is primarily built on Python version 3.5 or 3.6. By transitioning to a new version, we intend to leverage the advanced features of the newer releases to improve code readability and overall functionality.

Python Release Cycle

As Python version 3.7 and earlier have reached their end-of-life (figure below), it becomes crucial to shift our focus to a supported and stable version to maintain the integrity and security of our project.

Python release cycle



Supported versions

Dates shown in *italic* are scheduled and can be adjusted.

Branch	Schedule	Status	First release	End of life	Release manager
main	**PEP 719**	feature	*2024-10-01*	*2029-10*	Thomas Wouters
3.12	**PEP 693**	bugfix	2023-10-02	*2028-10*	Thomas Wouters
3.11	**PEP 664**	bugfix	2022-10-24	*2027-10*	Pablo Galindo Salgado
3.10	**PEP 619**	security	2021-10-04	*2026-10*	Pablo Galindo Salgado
3.9	**PEP 596**	security	2020-10-05	*2025-10*	Łukasz Langa
3.8	**PEP 569**	security	2019-10-14	*2024-10*	Łukasz Langa

Code Size Reduction and Performance Enhancement

To enhance the performance and improve the readability of our project, I have optimized the code by utilizing the advanced syntax available in the newer Python versions.

For instance:

(a) Efficient List Generation

Python 3.10 allows for more efficient list generation, exemplified by the more concise expression `[i for i in range(10)]`. This enhancement not only streamlines the code but also augments readability.

(b) Improved Dictionary Iteration

The enhanced dictionary iteration capabilities in Python 3.9 are illustrated by `{key: value for key, value in zip([1, 2, 3], ["val", "val", "val"])}`. Such updates significantly declutter loop functions and elevate code readability, thereby facilitating more user-friendly interactions with dictionaries.

(c) String Formatting

Old String Assignment (will be deprecated in the future):
The existing codebase may have expressions like

```
print("Hi %s, this is old string assignment with number %d assigned" % ("Roy",
5))
```

Updated String Assignment (f-strings), a more pythonic string formatting:
Python 3.8 introduces more intuitive string formatting, exemplified by:

```
print(f"Hi {'Roy'}, this is the new string assignment with number {5}
assigned")
```

These refinements ensure that our code is not only more efficient and readable but also adheres to modern programming practices.

Add type hints, function and class document string explanation to clearly defined and designed functions and classes

One of the most notable updates in Python 3.10 is the significant enhancement of typing. This feature enables precise argument type annotations in function definitions, greatly improving code readability and reliability. In the past, we often relied on docstrings to document expected types, which could be error-prone. However, with Python 3.10, we can explicitly declare the types of function arguments and return values, making the code self-documenting and reducing the risk of type-related errors. This enhanced typing support not only aids in understanding the code but also provides valuable information for static analyzers and development tools. An illustrative example of this improvement is shown below:

Function Definition (Old Versions) (Also in Current MGWR, many functions have no docstrings explanations)

```
def add_two(num_1, num_2):
    """Add two numbers

    num_1: int or float
    first input number
    num_2: int or float
    second input number

    """
    return num_1 + num_2
```

Function Definition (New Versions)

```
def add_two(num_1: int, num_2: int = 2) -> int:
   """Add two numbers

   Parameters:
        num_1: int, the first input number
        num_2: int, the second input number. default 2

    Raises:
        ValueError: Please correctly prepare your input data type.
        if exist raise error inside function add_two, else will not show Raises
section in Docstring

    Warnning:
        if exist Warnning inside function add_two, else will not show Warnning
section in Docstring

    Returns:
        int: the addition result of two integer numbers

    Examples:
        >>>add_two(1,2)
            3
    """
    return num_1 + num_2

```

Code Refactoring

In addition to the language-specific improvements mentioned above, upgrading to Python 3.10 provides an opportunity for code refactoring. This process involves revisiting and restructuring the existing codebase to align with the latest language features and best practices. Code refactoring can enhance maintainability, reduce technical debt, and improve overall code quality. By refactoring our current code, we ensure that it remains efficient, readable, and maintainable in the evolving Python ecosystem, making it easier to adapt to future changes and requirements. It's an essential step in keeping our codebase robust and up-to-date.