Assignment #3:

Create an R or Python Package for a Frequently Used Function

Objective:

Choose a function that you (or your lab) have developed and use frequently. Create an R or Python package that implements this function, focusing on usability, documentation, and adherence to best practices.

Instructions:

1. **Select your function:**
    - Choose a function that is commonly used in your work or one that provides a significant benefit in your research. Ensure that the function is well-defined and can stand alone as a useful utility.
2. **Version control [optional]:**
    - Initialize a Git repository for your package and make regular commits to document your development process.
3. **Package structure:**
    - Create a new package using either:
        - **R:** Use `usethis::create_package("path/to/package")` to set up your package structure.
        - **Python:** Use `cookiecutter` with the cookiecutter-pypackage template to scaffold your package.
        - **From scratch**: this is often the best method, since you have complete control
4. **Implement your function:**
    - Write the function code and place it in the appropriate directory:
        - **R:** Place your function in the `R/` directory.
        - **Python:** Place your function in the `your_package/` directory.
5. **Documentation:**
    - Write clear and comprehensive documentation for your function:
        - **R:** Use `roxygen2` comments to document your function. Make sure to include details about parameters, return values, and examples of usage.
        - **Python:** Use docstrings within your function to explain its purpose, parameters, and return values. Consider using Sphinx for generating additional documentation.
6. **Testing [optional]:**
    - Create a set of tests to ensure your function behaves as expected:

        - **R:** Use the `testthat` package to write unit tests for your function, ensuring it handles various input scenarios correctly.
        - **Python:** Use `pytest` to write tests for your function, covering edge cases and expected outputs.
7. **Package metadata:**
    - Fill in the metadata for your package:
        - **R:** Add information to the `DESCRIPTION` file, including the package name, version, author, and description.
        - **Python:** Update `setup.py` (or `pyproject.toml` if using Poetry) with the package name, version, author, and description.
8. **Install and test your package:**
    - Install your package locally to ensure it works as intended:
        - **R:** Use `devtools::install()` to install your package.
        - **Python:** Use `pip install .` to install your package in editable mode.
9. **Publish your package (optional):**
    - If you wish, consider publishing your package to:
        - **R:** CRAN or GitHub.
        - **Python:** PyPI or GitHub.

Deliverables:

- A compressed package (e.g., `tar.gz`).
    - In R, this is output from `devtools::build()`
    - In python, this is output from `poetry build`
- [Optional] Instead of the compressed package, send your completed R or Python package in a Git repository, including:
    - **The implemented function.**
    - **Comprehensive documentation.**
    - **Tests for your function.**
    - **Metadata files (e.g., DESCRIPTION or setup.py).**
- **Brief report (½ to 1 page)**: Include the following details:
    - **Purpose**: Describe what your package does. Why is your package useful?
    - **Limitations:** What are some limitations of your package? How can this improve in the future?

Resources:

- **R Package Development:**
    - R Packages by Hadley Wickham
    - devtools Package Documentation
    - testthat Package Documentation
- **Python Package Development:**
    - Python Packaging User Guide
    - pytest Documentation
    - Sphinx Documentation

Tips:

- Focus on writing clean, reusable code and clear documentation.
- Make sure your package is user-friendly and easy to install.
- Don’t hesitate to ask for feedback from peers or mentors as you develop your package.