Steps:

# Step-1: setup environment

Create Modular Directory Structure

* Setup Github repo: <https://github.com/AkileshVishnu/mlproject1>
* Setup local dir: C:\Users\Akilesh\OneDrive\Desktop\Data\_Projects\ML\_Projects\mlproject
* Setup virtual env: C:\Users\Akilesh\OneDrive\Desktop\Data\_Projects\data\_venv
* Sync github code repo and local code dir
* Create proper production setup like Modules, packages
* Setup project folders like to publish packages for opensource by deploying with the following process like (setup.py, src->\_\_init\_\_.py )
* Src folder -> build the package
* Created the below essential folders for keeping project modular and organized

**artifacts** for storing outputs like models or data files.

**logs** for recording logs of the project.

**notebook** for Jupyter notebooks.

**src** for source code that handles the machine learning pipeline.

# Setp-2: Code for the Modules

## Order of writing Modules

This is dynamic not all code is written at once and then the next one is written. It is more like connecting the dots backwards. So First the major functionalities are coded and then they are configured and linked between modules, unit tested and then the ultimate pipeline is executed

These two are first coded, explored and tested before the final modular files are created

notebook\data\1. EDA STUDENT PERFORMANCE .ipynb

notebook\data\2. MODEL TRAINING.ipynb

Now after the code and plan of action is tested we will start writing code as Modules and packages that can be used dynamically and deployed

src\exception.py

src\logger.py

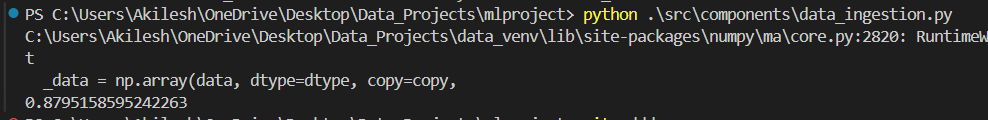
src\components\data\_ingestion.py

src\components\data\_transformation.py

src\utils.py -> save\_object created first

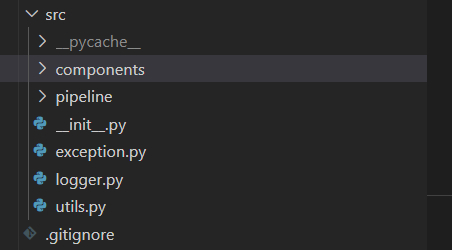
src\components\model\_trainer.py

Now all modules are Imported into src\components\data\_ingestion.py and a demo test is run till here if we are able to see the r2\_score

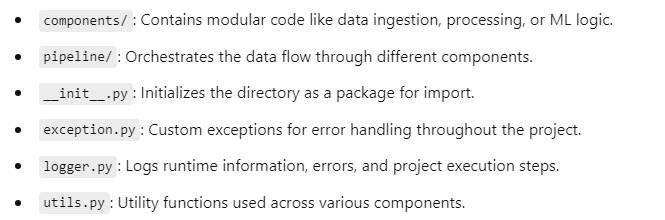


## Various Packages

### src



Explanation)



# Doubts I had during project and coding

1. @dataclass

The **@dataclass** decorator in Python simplifies class creation by automatically generating special methods like \_\_init\_\_, \_\_repr\_\_, and \_\_eq\_\_. In this example, **DataTransformationConfig** stores the path of a preprocessor object (preprocessor.pkl). It uses **os.path.join** to dynamically create file paths for storage.

