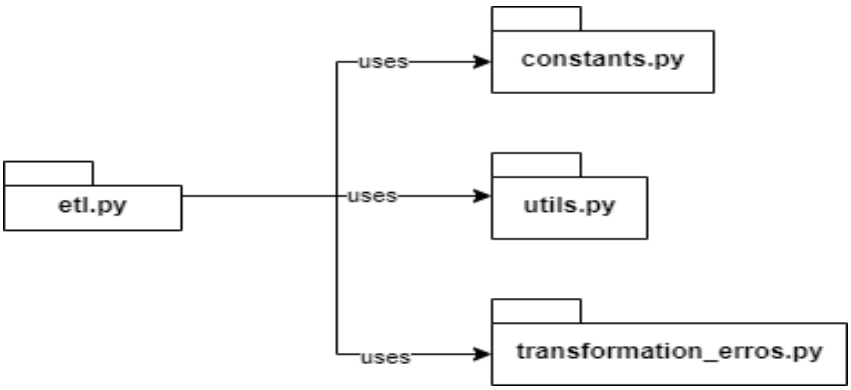
[github]
[pypi]

Auto1 ETL Challenge

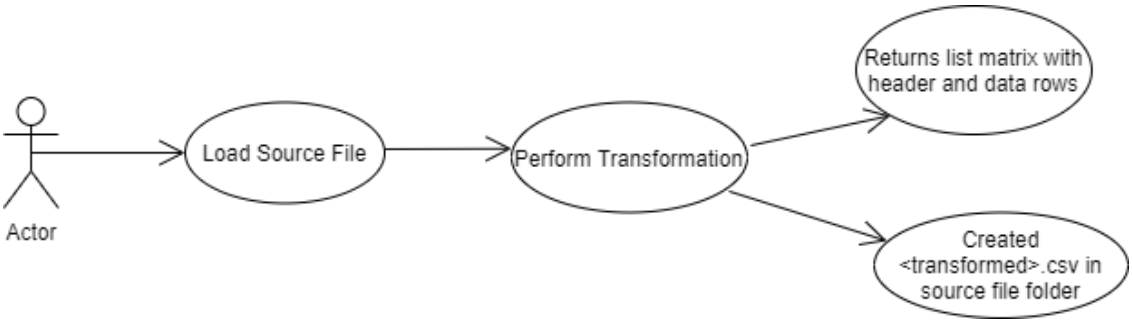
To understand the code design and architecture, I have created some UML diagram illustrations that can sum up the entire code solution very steadily.

Package Diagram



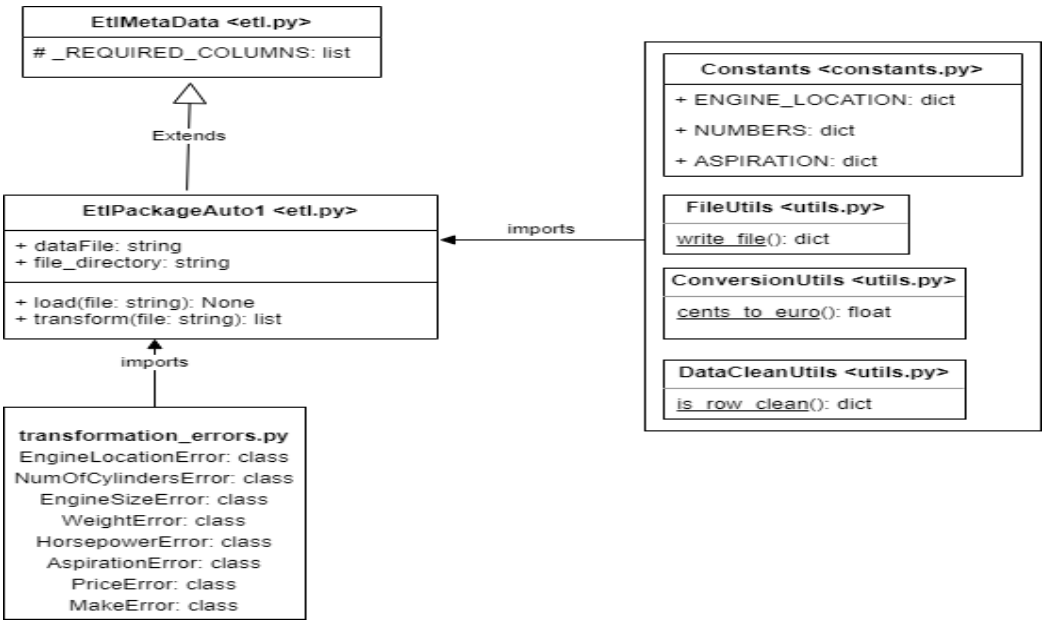
This code base comprises of 4 different modules, where ***etl.py*** uses rest of the three modules.
- **etl.py** performs the main ETL operations like load and transform.
- **constants.py** serves as a file to place constant values that can be used anywhere into the project.
- **utils.py** provides utility functions that helps in performing validated ETL operation.
- **transformation_errors.py** provides customized error message to handle the exception in different columns.

Use Case Diagram



For a sample use case, we may assume that a user would load a file which has some garbage values like **(-)** in different rows and unwanted columns. Invoking a load method on file would clean the garbage rows and load the data with required columns into the *staging file*. Now transform method is invoked which performs required transformation and converts the data into the form that could be used by machine learning models and also creates a *transformed file*.

Class Diagram



- Each module has different classes that serve their own purpose.
- **EtlMetaData** stores the descriptive for the pipeline.
- **EtlPackageAuto1** holds attributes for file loading and methods to perform operations.
- **Constants** class store the constant values that can be used across the pipeline operations.
- **FileUtils** performs files related operations.
- **ConversionUtils** performs conversion on the data if required.
- **DataCleanUtils** has methods that can be used to clean the data while loading the raw file.

Design and Architecture

- Code repository uses the constructor pattern to provide the file path which is loaded into a static variable that can be used across the different operations of the pipeline.
- All the methods of the different classes are set to static since objects are not required.
- Custom exception classes are created to handle the errors differently for each of the required column which consequently provides the way to customize the handling of the different rows.
- Code parts are separated with respect to module pattern where each module serves different purpose during the operation of the pipeline.
- Code is optimized for runtime complexity by avoiding the nested loops and leveraging the extensive use of python dictionaries.