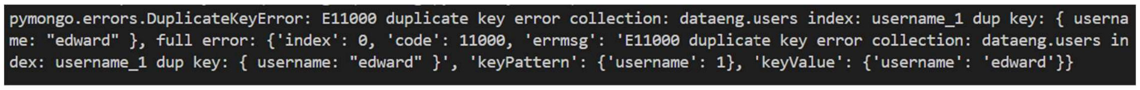
During the implementation of both PyMongo and MongoEngine, I noticed clear differences not only in syntax but also in how each handled errors and validation.

When using PyMongo, the system allowed me to insert any kind of document as long as the MongoDB server accepted it. There was no built-in validation for field types or uniqueness, so any mistakes—such as missing fields or wrong data types—would only become visible during runtime or data inspection. For example, when testing, I initially forgot to convert the _id field to a string, which caused type mismatch issues when reusing it in later queries. After manually adding str(result.inserted_id), the issue was resolved. This experience highlighted that with PyMongo, the developer must explicitly handle all type conversions and constraints.

In contrast, MongoEngine automatically enforced schema-level rules. When I defined my User document with
username = StringField(required=True, unique=True), attempting to insert a user with the same username caused an immediate error:

mongoengine.errors.NotUniqueError:



```
pymongo.errors.DuplicateKeyError: E11000 duplicate key error collection: dataeng.users index: username_1 dup key: { userna
me: "edward" }, full error: {'index': 0, 'code': 11000, 'errmsg': 'E11000 duplicate key error collection: dataeng.users in
dex: username_1 dup key: { username: "edward" }', 'keyPattern': {'username': 1}, 'keyValue': {'username': 'edward'}}
```

This error occurred because both the MongoDB index and MongoEngine's field constraint prevented duplicates.
While PyMongo would have silently allowed such an insert attempt (until MongoDB rejected it), MongoEngine raised a Python exception right away, stopping execution.

Overall, these experiences clearly showed that PyMongo provides more flexibility but requires manual control and error handling, whereas MongoEngine offers automatic validation and stronger safety through schema enforcement.