Specification for Uploading Dataset to MongoDB

Deadline: August 9th

Overview

The task involves downloading a 3GB CSV file, converting it to a more efficient format, and then uploading the data to a MongoDB database using a defined schema. This is a one-off task but the schema for the mongoDB may change in the future so it may need to be adapted for final usage to match any changes in the schema.

Tasks and Requirements

1. **Download the data**
    - We want to upload the data in this 3GB CSV file from here: https://osf.io/2f857 (metadata also available at the link).
    - A smaller file that you can work from is available at this link. You can read this with `pd.read_feather("all_indicators.feather")`

2. **Dataframe Conversion Pipeline**
    - Create a pipeline to convert the output of the previous step to a dataframe ideal for use with the provided schema.
    - Abstract operations into functions as required to keep the command chain concise and readable.
    - Utilize Polars or Pandas for the dataframe operations.

3. **Schema Mapping and Data Upload**
    - Use the output of the previous step with the preliminary schema provided for MongoDB. Specifically use the Invocation class for the document model.

4. **Testing**
    - Write tests for the database upload step using a stub for the database. These tests do not need to test for unforeseen issues, this dataset is the only one the script will be used for.
    - Use `pytest` with fixtures as required.

5. **Module Structure**

- All code, including tests, should be contained within a single Python module.
- Ensure the conversion/upload functionality is accessible from Python as an importable function from the module.
- Write the script in a way that it can easily be altered to deal with schema changes.

Schema

```
from odmantic import ObjectId
from pydantic import BaseModel, EmailStr, Field

# mriqc applies a unique id to each person so that you can aggregate and detect
# duplication. One can have different views of same work
# (pubmedcentral and nature). Each would have different filenames, different
# provenance, different md5sum. Usually there is a final version of record,
# ideally we would be analysing that. Generally for high throughput we analyse
# the open access pubmed central.

class Work(BaseModel):
    """
    Unique reference for each publication/study/work. For each  “work”,
    pmid, doi (normalized), openalex ids are approaches to referencing such a
    study uniquely but any one of them may be used by a user. Versioning of the
    publications (as in pubmed vs Nature vs addendums) should all be handled
    naturally as part of an array of referenced “user input documents” (let’s say
    a pdf) provided as part of each "Invocation" or cli call.
    """
    id: ObjectId = Field(alias="_id")
    user_defined_id: str
    pmid: str
    doi: str
    openalex_id: str
    scopus_id: str
    file: bytes
    content_hash: str
    timestamp: str


class Derivative(BaseModel):
    """
    Gridfs can avoid issues with size limitations. Each derivative is an output of
    execution of a single container with the “preceding document” or “parent”
    referenced (this could be a primary document or another derivative). A primary
    document could have several different derivatives (scibeam and rtransparent)
    and/or several versions of the same derivative type (scibeam and rtransparent
    across different releases or rtransparent or modifications of our docker
    image). A text label would be useful here but a docker image id is likely the
    sanest way to track derivatives (which would mean that all derivatives must be
    computed in a docker container).
    """

    id: ObjectId = Field(alias="_id")
    component_id: str
    text_label: str
    version: str
```

```
class Component(BaseModel):
    id: ObjectId = Field(alias="_id")
    name: str
    version: str
    docker_image: str
    docker_image_id: str


class Metrics(BaseModel):
    """Potentially link to other databases for extra metadata"""

    id: ObjectId = Field(alias="_id")
    metrics: dict


class Client(BaseModel):
    id: ObjectId = Field(alias="_id")
    compute_context_id: str
    email: EmailStr
    invocation_id: str


class Workflow(BaseModel):
    """
    Schema to describe an analysis that was run by a user. This represents a
    chain of steps that culminate in the bibliometric output json. It will
    contain a reference to the primary document of the "work" supplied as input,
    the steps run, and derivatives created.
    """
    id: ObjectId = Field(alias="_id")
    output: Metrics
    work_id: str
    steps: list[Component]
    derivatives_created: list[Derivative]


class Invocation(BaseModel):
    """
    Approximate document model. This may evolve but provides a starting point
    for the Odmantic document model used to interact with mongodb.
    """
    id: ObjectId = Field(alias="_id")
    osm_version: str
    timestamp: str
    client: Client
    work: Work
    workflow: Workflow
    output_id: str
    user_comment: str
```