UNIVERSIDADE DA CORUÑA

Data Engineer
Master in Artificial Intelligence

P3: Dimensional modelling & ETL data processing

Students:
Antonio Vila Leis, antonio.vila@udc.es
David Florez Mazuera, d.florez@udc.es

1. Introduction

1.1 Project Context and Objective

This report details the design and implementation of an ETL (Extract, Transform, Load) pipeline and a dimensional data model for analyzing medical imaging metadata. The project's core objective is to transform raw DICOM (Digital Imaging and Communications in Medicine) file metadata into a structured data warehouse environment using a star schema, leveraging MongoDB as the NoSQL database for storage.

1.2 Business Need

The primary goal is to support analytical queries focused on three key areas of medical imaging operations:

1. **Imaging Quality:** Assessing image characteristics like pixel spacing, rows, and columns.
2. **Equipment Performance:** Analyzing station details, manufacturer, model, and exposure parameters.
3. **Protocol Consistency:** Evaluating the consistency of scan protocols (body part, patient position, contrast agent usage) across patients and over time.

This structured data model will enable users to perform rapid, historical analysis to monitor equipment efficiency, ensure compliance with scanning standards, and correlate imaging parameters with patient outcomes.

2. Project Overview

2.1 Data Source

The primary data source is the **SIIM Medical Images Kaggle dataset**, specifically the metadata contained within the DICOM files. DICOM is the standard format for medical images, and its files contain not only the image data but also extensive metadata tags (e.g., (0010,0040) for Patient Sex, (0008,0070) for Manufacturer).

2.2 Technology Stack

The ETL and modeling process utilizes the following technologies:

- **Database:** MongoDB Community Edition (NoSQL) for storing the dimensional model (medical_imaging_dw).
- **ETL Tool:** Python 3.12, utilizing the pymongo library for database interaction and pydicom for parsing and extracting metadata from DICOM files.

- **Configuration:** Connection details (host, port) and data directories are managed using environment variables and the config.py file to ensure portability and easy configuration.
- **Tools:** MongoDB Compass for database exploration and query validation.

2.3 Project structure

```
p3/
├── .env                   # Environment variables, user configuration
├── pyproject.toml         # Project config and dependencies
├── README.md
├── data/                  # Project data
│   ├── dicom_dir/         # CT scans
│   ├── tiff_images/       # TIFF images
│   ├── overview.csv       # Metadata (Age, Contrast, IDs)
│   ├── full_archive.npz   # Numpy archive
│   └── output/            # Generated results
│       └── jpeg_images/   # Converted JPEG
└── src/                   # Source Code
    └── de_p3              # Package
        ├── __init__.py    # Inicialización del paquete
        ├── processing.py  # Full ETL pipeline
        ├── config.py      # Package configuration file
        └── utils.py       # Utilities for the ETL
```

2.4 DICOM format

DICOM, or Digital Imaging and Communications in Medicine, is the international standard for medical images and related information. It defines the format for medical images and how they can be exchanged with the associated metadata.

A DICOM file contains both image data (pixel data) and extensive metadata tags, such as patient information (Patient ID, Name, Age, Sex), study details (Study Date, Modality, Body Part Examined), and equipment information (Manufacturer, Model). This comprehensive metadata makes DICOM files self-contained and highly interoperable, crucial for clinical workflows and medical imaging analysis.

3. Data Model

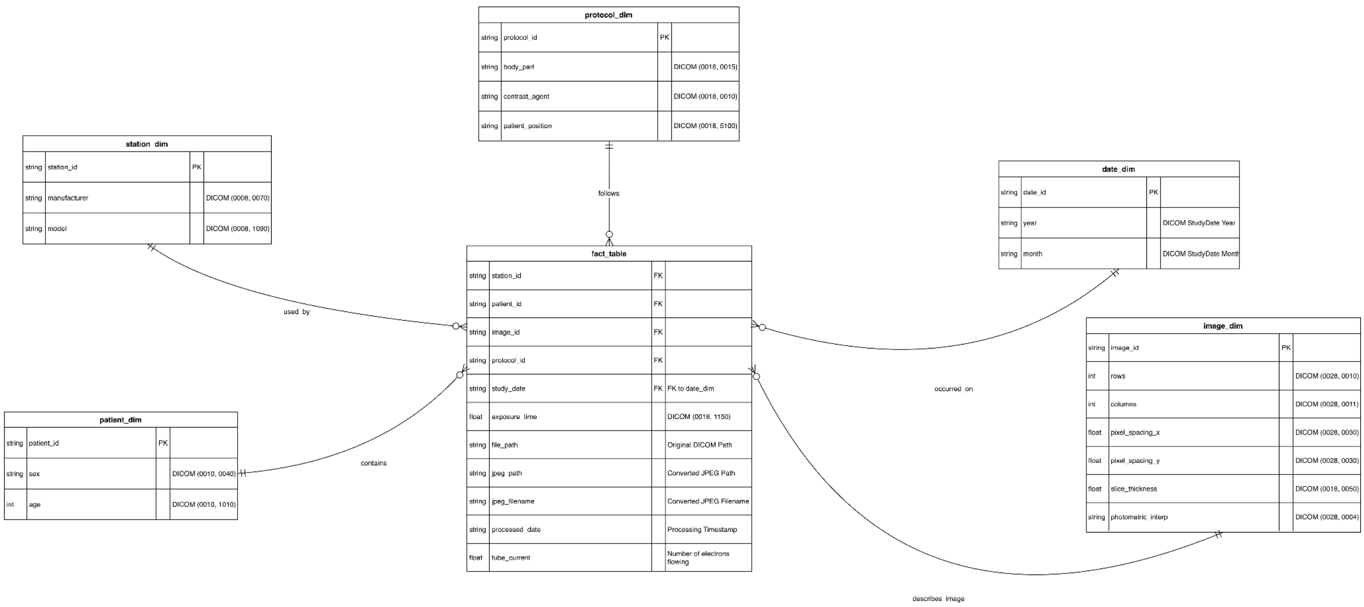
3.1 Dimensional Model Design

The project utilizes a **Star Schema** centered around the **STUDY Fact Table**. This model is designed to optimize query performance for analytical workloads.

The names of the collections from the initial design established in the guide have been changed as shown in the next table.

Component	Collection Name	Description
Fact Table	fact_table	**STUDY**: Represents a single medical scan event. It contains foreign keys (surrogate keys) linking it to the dimension tables and the measures relevant to the study.
Dimensions	patient_dim	**PATIENT**: Describes the subject of the study (e.g., sex, age).
	station_dim	**STATION**: Describes the equipment used for the scan (e.g., manufacturer, model).
	protocol_dim	**PROTOCOL**: Describes the scanning configuration (e.g., body part, contrast agent).
	image_dim	**IMAGE**: Describes the physical characteristics of the image slice (e.g., rows, pixel spacing).
	date_dim	**DATE**: A simple date dimension used for temporal filtering and grouping (year, month).

Schema Visualization:



Key Relationships (Fact Table to Dimensions):

- fact_table links to patient_dim via patient_id (FK to PK).
- fact_table links to station_dim via station_id (FK to PK).
- fact_table links to protocol_dim via protocol_id (FK to PK).
- fact_table links to image_dim via image_id (FK to PK).
- fact_table links to date_dim via study_date (FK to PK).

3.2 MongoDB Collection Explanations and Visual Examples

In the MongoDB implementation, the get_or_create function ensures that the Primary Key (PK) field in the dimension collections (e.g., patient_id) and the Foreign Key (FK) field in the fact_table both contain the same **surrogate hash key**. This approach simplifies lookups and joins across the collections.

1. patient_dim Collection (Dimension)

- **Purpose:** Stores unique patient demographic information. The PK (patient_id) stores the unique hash of the attributes (sex, age).
- **Primary Key (PK):** patient_id (Surrogate Hash Key).
- **Fields:** sex ((0010, 0040)), age ((0010, 1010), *formatted to integer*).
- **Visual Example:**

```
_id: ObjectId('68fa338584e6695c1514df07')
patient_id : "6c44def74e39240c1683575ad497242a"
sex : "M"
age : 67
```

2. station_dim Collection (Dimension)

- **Purpose:** Stores unique equipment details used for the scans.
- **Primary Key (PK):** station_id (Surrogate Hash Key).
- **Fields:** manufacturer ((0008, 0070)), model ((0008, 1090)).
- **Visual Example:**

```
_id: ObjectId('68fa338584e6695c1514df08')
station_id : "f4404e900fb062ddceffb8834d8476ec"
manufacturer : "GE MEDICAL SYSTEMS"
model : "LightSpeed VCT"
```

3. protocol_dim Collection (Dimension)

- **Purpose:** Stores unique scan protocol configurations.
- **Primary Key (PK):** protocol_id (Surrogate Hash Key).
- **Fields:** body_part ((0018, 0015)), contrast_agent ((0018, 0010), *normalized*), patient_position ((0018, 5100)).
- **Visual Example:**

```
_id: ObjectId('68fa338584e6695c1514df09')
protocol_id : "657eb2b009666d7647887c8daa3d901d"
body_part : "LUNG"
contrast_agent : "No contrast agent"
patient_position : "FFS"
```

4. image_dim Collection (Dimension)

- **Purpose:** Stores standardized physical attributes of the image slices.
- **Primary Key (PK):** image_id (Surrogate Hash Key).
- **Fields:** rows ((0028,0010)), columns ((0028,0011)), pixel_spacing_x/y ((0028,0030), *normalized to predefined bins*), slice_thickness ((0018,0050)), photometric_interp ((0028,0004)).
- **Visual Example:**

```
_id: ObjectId('68fa338584e6695c1514df0b')
image_id : "e2f41248fc9cc4059f801df1522ffaa8"
rows : 512
columns : 512
pixel_spacing_x : 0.7
pixel_spacing_y : 0.7
slice_thickness : 2.5
photometric_interp : "MONOCHROME2"
```

---

5. date_dim Collection (Dimension)

- **Purpose:** Stores time components for easy query slicing.
- **Primary Key (PK):** date_id (Surrogate Hash Key, typically a hash of the date string).
- **Fields:** year, month (both from (0008,0020)).
- **Visual Example:**

```
_id: ObjectId('68fa338584e6695c1514df0a')
date_id : "823aacb951471c34777224528869af26"
year : "1995"
month : "01"
```

6. fact_table Collection (Fact Table)

- **Purpose:** Stores the individual study/scan event, linking all dimensions and containing the key performance measures.
- **Foreign Keys (FKs):** patient_id, station_id, protocol_id, image_id, study_date (all storing the respective dimension's hash key).
- **Measures:** exposure_time ((0018, 1150)), tube_current ((0018, 1151)) file_path (original DICOM), jpeg_path (converted JPEG path).
- **Visual Example:**

```
_id: ObjectId('68fa6c5f5cd6d03afebc7d2c')
station_id : "65e9e00458da0c949638d3dadbb5b75c"
patient_id : "e22afe0efdd28303b3f0496619d4c8e3"
image_id : "548a5e11339b226ca7e1587fec7c6cac"
protocol_id : "57d48c7da1c5ab6a4107f418b54a0cae"
study_date : "f7689288edb92b2832806c43ec3055cd"
exposure_time : 750
tube_current : 206
file_path : "data\dicom_dir\ID_0000_AGE_0060_CONTRAST_1_CT.dcm"
jpeg_path : "data\output\jpeg_images\ID_0000_AGE_0060_CONTRAST_1_CT.jpg"
jpeg_filename : "ID_0000_AGE_0060_CONTRAST_1_CT.jpg"
processed_date : "2025-10-23T19:56:47.722943"
```

4. ETL Data Processing

The ETL pipeline is fully implemented in Python and orchestrated from processing.py, relying on centralized configuration from config.py and specific utilities in utils.py. The script iterates over the DICOMs located in the configured directory, performs an initial visual and metadata inspection, establishes a connection with MongoDB, and finally processes each file to populate a dimensional model.

4.1 Utility Functions (utils.py)

The utils.py file encapsulates a set of functions used throughout the data processing:

- **surrogate_key**: generates a stable MD5 key from the ordered (key, value) pairs of a dictionary. The stability of the hash guarantees that the same combination of attributes will always produce the same surrogate key.

- **get_or_create**: acts as a guardian of the dimensions. From the values of a dimension, it generates a stable surrogate key using surrogate_key that uniquely identifies that combination, uses it to check whether an identical document already exists, and if it does, reuses it; if not, it inserts a new one with that same key and values. This prevents duplicates, makes the load idempotent even if the pipeline is executed multiple times, and always returns that key so the fact table can use it as a reference.

- **format_age**: converts the DICOM age format "NNNY" (for example, "061Y") to an integer, returning None if it is invalid or missing. This avoids typing errors and standardizes the field.

- **normalize_pixel_spacing**: attempts to convert the value to float and approximates it to the nearest bin within {0.6, 0.65, 0.7, 0.75, 0.8}, returning None when it cannot be converted. In this way, close resolutions are discretized and analysis is facilitated.

- **normalize_contrast_agent**: standardizes the contrast agent, assigning "No contrast agent" when the value is empty, null, or a single character, and returning the cleaned string otherwise. This stabilizes a notoriously noisy field.

- **dicom_to_jpeg**: produces a consistent visual derivative: normalizes image intensities to 0–255, converts to 8 bits, resizes to 256×256, and saves a .jpg in the specified directory, returning the final path so it can be persisted in the fact table.

4.2 Data Extraction

Extraction begins with the discovery of DICOM files. The script imports RAW_DATA_DIR from config.py as DATA_PATH and builds the *.dcm pattern to search for files using glob. Once the files are found, it constructs a pandas DataFrame that contains for each row the full path and file name, which facilitates both inspection and preview visualization.

Before connecting to the database, the script provides a quick visual verification of the data. It takes the first sixteen paths from the DataFrame, opens each DICOM with pydicom.dcmread, and displays its pixel_array in a 4×4 grid using matplotlib. Instead of showing the interactive window, it saves the mosaic as dicom_grid_output.png in the parent of DATA_PATH and closes the figure. This allows visual auditing that the files are being read correctly and that the images look as expected.

Next, it performs metadata inspection on a single sample file. It selects the first path, loads it with pydicom, and displays a header with the file name. It then prints only a subset of representative fields: PatientID, PatientName, PatientAge, PatientSex, Modality, and Manufacturer. This allows a quick validation of the presence and format of some attributes and confirms that DICOM reading is going properly.

4.3 Data Transformation

With the preliminary inspection completed, the script prepares the connection to MongoDB, performs a ping to confirm it is operational, and clears the target tables. Once done, it begins the main loop that processes each DICOM independently. For each file, it opens the dataset with pydicom.dcmread and extracts the fields of interest using getattr, always providing safe default values.

- For **patient_dim**, it takes PatientSex and PatientAge directly from the dataset, but transforms the age with format_age() to convert the DICOM format (for example, “061Y”) to an integer (61) and return None when the value is not interpretable.

- For **station_dim**, it uses Manufacturer and ManufacturerModelName, which identify the manufacturer and model of the equipment.

- For **protocol_dim**, it captures BodyPartExamined, PatientPosition, and the contrast agent; the latter is normalized with normalize_contrast_agent to replace empty, null, or single-character strings with the literal “No contrast agent”, thus stabilizing the domain of values.

- **date_dim** is derived from StudyDate: the script takes the string and extracts year and month by slicing, mitigating the absence of the field with empty strings.

- **image_dim** adds purely technical characteristics. Rows and Columns are typed as integers. PixelSpacing is handled with special care because DICOM exposes it as a list of two values [row_spacing, col_spacing]; the code decomposes this array into pixel_spacing_x and pixel_spacing_y, converting each element to float only if it exists, otherwise defaulting to 0.0. It then applies normalize_pixel_spacing to both axes, which approximates the value to the nearest bin in a predefined set (0.6, 0.65, 0.7, 0.75, 0.8), with the goal of consolidating resolutions into discrete categories. It also captures SliceThickness as float and PhotometricInterpretation as string, preserving information about sampling and photometric representation.

- The measures intended for the fact table include times (ExposureTime) and tube current (XRayTubeCurrent), which are converted to float.

Before loading the data, the pipeline generates a visual derivative for each study with dicom_to_jpeg. This function reads the pixel_array, normalizes its intensities to the 0–255 range per image using min–max normalization, converts to uint8, transforms it into a grayscale PIL image, and resizes it to 256×256 using the LANCZOS filter, saving it as .jpg in data/output/jpeg_images. The conversion returns both the full path and the JPEG file name; both are stored in the fact table for traceability and to facilitate consumption by external tools.

4.4 Data Loading

With the transformed values, the script proceeds to conform each dimension using get_or_create. For each dimension, it constructs a dictionary with the final attributes, internally computes a stable surrogate key for that set, and queries whether a document with that identifier already exists. If it does not, it inserts it; in any case, it returns the identifier to be used as a foreign key in the fact table.

Insertion into the fact table is performed at the end of processing each file, once all surrogate keys are available. The script builds a document with station_id, patient_id, image_id, protocol_id, and study_date (which is actually the date_id from date_dim), and adds the measures exposure_time and tube_current. For traceability, it includes file_path, jpeg_path, and jpeg_filename, along with processed_date. That document is inserted into the fact_table collection. If any exception occurs during the processing of a file, it is caught, the name of the failed file is reported, the error counter is incremented, and execution continues with the next DICOM.

At the end of the batch, the script prints a summary stating the total number of files processed and the number of errors, and queries the document counts in each collection (patient_dim, station_dim, protocol_dim, date_dim, image_dim, fact_table). It also confirms the path where the JPEGs were saved. This conclusion provides a quick and quantitative verification of the execution's success and clearly indicates the locations of the generated artifacts.

5. Queries and Analysis

5.1 Business Questions

The following business questions are designed to leverage the medical imaging dimensional model to analyze equipment performance, protocol consistency, and resource usage across patients and time.

1. **Protocol Efficiency by Manufacturer**
    - **Question:** What is the average **exposure time** for scans

- **Modality**, grouped by the **manufacturer** of the scanning station, and which manufacturer has the lowest average?
- **Business Value:** Assesses the efficiency and consistency of different equipment models/manufacturers. Lower exposure times can indicate newer, more efficient equipment or consistent protocol adherence, which is important for patient safety (reduced radiation) and throughput.

2. **Popularity of Contrast Agents by Age Group**
   - **Question:** Among male patients (sex: "M"), what is the **count of studies** where a contrast agent was used, grouped into age buckets (e.g., "50-59", "60-69", "70+"), and which age group had the highest usage?
   - **Business Value:** Provides insights into clinical trends and resource utilization. This analysis helps determine if contrast agent usage correlates with specific demographics, which can inform inventory management, procurement, and protocol review.
3. **Image Dimensions for Chest Scans**
   - **Question:** What are the minimum, maximum, and average **pixel dimensions** (rows and columns) for all images where the **body part** is the "CHEST"?
   - **Business Value:** Determines the range and standard size of images for a specific body part. This is critical for data quality checks, ensuring consistency in image acquisition, and optimizing downstream image processing/AI model training, as models often require standardized input sizes.
4. **Temporal Analysis of Slice Thickness**
   - **Question:** What is the **average slice thickness** used for scans in each **month** of the dataset, and how has this average changed over time?
   - **Business Value:** Tracks changes in standard imaging protocols over time. Significant shifts in average slice thickness could indicate new clinical guidelines, updated equipment settings, or inconsistencies in how different dates/operators run the protocols.
5. **Protocol Variance by Patient Position**
   - **Question:** How many distinct **protocols** (by protocol_id) are used when the **patient position** is "HFP" (Head First Prone) versus "HFS" (Head First Supine), and what is the most common body_part for each?
   - **Business Value:** Measures the complexity and standardization of imaging protocols. A large number of distinct protocols for the same position/modality might signal protocol proliferation or lack of standardization, requiring review by the clinical team.

5.2 SQL/NoSQL Queries and Explanations

The queries are implemented using **MongoDB's Aggregation Framework**, which processes data in multiple stages, similar to SQL queries. The fact table is assumed to be named study_fact, and dimensions are named dim_patient, dim_station, dim_protocol, and dim_image.

Query 1: Protocol Efficiency by Manufacturer

Goal: Calculate the average exposure time for scans per manufacturer.

```
cdb.getCollection('fact_table').aggregate(
  [
    {
      $lookup: {
        from: 'station_dim',
        localField: 'station_id',
        foreignField: 'station_id',
        as: 'station_info'
      }
    },
    { $unwind: '$station_info' },
    {
      $group: {
        _id: '$station_info.manufacturer',
        average_exposure_time: {
          $avg: '$exposure_time'
        }
      }
    },
    { $sort: { average_exposure_time: 1 } }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Explanation: This query links the STUDY fact table to the STATION dimension using the foreign key station_id. The $group stage uses the **manufacturer** as the grouping key to compute the average of the exposure_time field. The final $sort arranges the results, making it easy to see which manufacturer has the most efficient (lowest) average exposure time.

Result:

```
_id: "SIEMENS"
average_exposure_time : 613.7313432835821
```

```
_id: "GE MEDICAL SYSTEMS"
average_exposure_time : 774.7272727272727
```

Query 2: Popularity of Contrast Agents by Age Group

Goal: Count studies where a contrast agent was used among male patients, grouped into decade-based age buckets.

```
db.getCollection('fact_table').aggregate(
  [
    {
      $lookup: {
        from: 'patient_dim',
        localField: 'patient_id',
        foreignField: 'patient_id',
        as: 'patient_info'
      }
    },
    { $unwind: '$patient_info' },
    {
      $lookup: {
        from: 'protocol_dim',
        localField: 'protocol_id',
        foreignField: 'protocol_id',
        as: 'protocol_info'
      }
    },
    { $unwind: '$protocol_info' },
    {
      $match: {
        'patient_info.sex': 'M',
        'protocol_info.contrast_agent': {
          $ne: 'No contrast agent'
        }
      }
    },
    {
      $group: {
```

```
  _id: {
    $switch: {
      branches: [
        {
          case: {
            $and: [
              {
                $gte: [
                  '$patient_info.age',
                  50
                ]
              },
              {
                $lt: [
                  '$patient_info.age',
                  60
                ]
              }
            ]
          },
          then: '50-59'
        },
        {
          case: {
            $and: [
              {
                $gte: [
                  '$patient_info.age',
                  60
                ]
              },
              {
                $lt: [
                  '$patient_info.age',
                  70
                ]
              }
            ]
          },
          then: '60-69'
        }
      ],
      default: '70+'
    }
  },
  study_count: { $sum: 1 }
}
```

```
  },
  { $sort: { study_count: -1 } }
 ],
 { maxTimeMS: 60000, allowDiskUse: true }
);
```

Explanation: This query involves multiple joins ($lookup) to connect the STUDY fact with both the PATIENT and PROTOCOL dimensions. The $match stage filters the data based on sex and the presence of a contrast agent. The final $group stage uses the powerful **$switch** conditional operator to categorize the continuous age field into discrete **age buckets** before performing the final count ($sum: 1\).

Result:

```
_id: "70+"
study_count : 2
```

Query 3: Image Dimensions for Chest Scans

Goal: Find the range (min, max) and average of pixel dimensions (rows and columns) for all scans of the "CHEST" body part.

```
db.getCollection('fact_table').aggregate(
 [
  {
   $lookup: {
    from: 'protocol_dim',
    localField: 'protocol_id',
    foreignField: 'protocol_id',
    as: 'protocol_info'
   }
  },
  { $unwind: '$protocol_info' },
  {
   $match: {
    'protocol_info.body_part': 'CHEST'
   }
  },
  {
   $lookup: {
    from: 'image_dim',
    localField: 'image_id',
    foreignField: 'image_id',
    as: 'image_info'
```

```
      }
    },
    { $unwind: '$image_info' },
    {
      $group: {
        _id: null,
        min_rows: { $min: '$image_info.rows' },
        max_rows: { $max: '$image_info.rows' },
        avg_rows: { $avg: '$image_info.rows' },
        min_columns: {
          $min: '$image_info.columns'
        },
        max_columns: {
          $max: '$image_info.columns'
        },
        avg_columns: {
          $avg: '$image_info.columns'
        }
      }
    },
    { $project: { _id: 0 } }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Explanation: This query performs two $lookup operations to access both the PROTOCOL and IMAGE dimension attributes from the STUDY fact. The data is first filtered using $match to only include studies related to the "CHEST" body part. The final $group stage uses _id: null to treat all filtered documents as a single group, allowing the calculation of aggregate statistics (**$min**, **$max**, and **$avg**) across the entire subset of Chest image dimensions.

Result:

```
min_rows : 512
max_rows : 512
avg_rows : 512
min_columns : 512
max_columns : 512
avg_columns : 512
```

Query 4: Temporal Analysis of Slice Thickness

Goal: Calculate the average slice thickness per month of study.

```
db.getCollection('fact_table').aggregate(
```

```
  [
    {
      $lookup: {
        from: 'date_dim',
        localField: 'study_date',
        foreignField: 'date_id',
        as: 'date_info'
      }
    },
    { $unwind: '$date_info' },
    {
      $lookup: {
        from: 'image_dim',
        localField: 'image_id',
        foreignField: 'image_id',
        as: 'image_info'
      }
    },
    { $unwind: '$image_info' },
    {
      $group: {
        _id: {
          year: '$date_info.year',
          month: '$date_info.month'
        },
        average_slice_thickness: {
          $avg: '$image_info.slice_thickness'
        }
      }
    },
    { $sort: { '_id.year': 1, '_id.month': 1 } }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Explanation: This query performs two $lookup operations to connect the STUDY fact with the DATE and IMAGE dimensions. The $group stage is key, using a **compound key** (year and month) to organize the data chronologically, then calculates the $avg of the slice_thickness for each period.

Result:

```
▾ _id: Object
    year: "1982"
    month: "06"
  average_slice_thickness : 6.5

▾ _id: Object
    year: "1982"
    month: "08"
  average_slice_thickness : 6

▾ _id: Object
    year: "1982"
    month: "11"
  average_slice_thickness : 5

▾ _id: Object
    year: "1982"
    month: "12"
  average_slice_thickness : 8
```

Query 5: Protocol Variance by Patient Position

Goal: Count distinct protocols used for two specific patient positions and identify the most common body part for each position.

```
db.getCollection('fact_table').aggregate(
  [
    {
      $lookup: {
        from: 'protocol_dim',
        localField: 'protocol_id',
        foreignField: 'protocol_id',
        as: 'protocol_info'
      }
    },
    { $unwind: '$protocol_info' },
    {
      $match: {
        'protocol_info.patient_position': {
          $in: ['HFP', 'HFS']
        }
      }
    },
    {
      $group: {
```

```
        _id: {
          position:
            '$protocol_info.patient_position',
          protocol: '$protocol_id',
          body_part: '$protocol_info.body_part'
        },
        count: { $sum: 1 }
      }
    },
    {
      $group: {
        _id: '$_id.position',
        distinct_protocol_count: {
          $addToSet: '$_id.protocol'
        },
        protocol_body_parts: {
          $push: {
            body_part: '$_id.body_part',
            count: '$count'
          }
        }
      }
    },
    {
      $project: {
        _id: 1,
        distinct_protocol_count: {
          $size: '$distinct_protocol_count'
        },
        protocol_body_parts: 1
      }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Explanation: This query uses a $lookup to join the STUDY fact with the PROTOCOL dimension. It uses a **two-stage grouping** similar to Query 6. The first $group establishes the counts for every unique combination of position, protocol, and body_part. The second $group uses **$addToSet** to accurately count the unique protocol_ids for each position and uses **$push** to collect the body part counts for analysis. The final $project uses **$size** to turn the set of distinct protocols into a numeric count.

Result:

```
_id: "HFS"
▸ protocol_body_parts : Array (5)
distinct_protocol_count : 5
```

```
_id: "HFP"
▸ protocol_body_parts : Array (1)
distinct_protocol_count : 1
```

6. How to Run the Code

6.1 Prerequisites

- Python >= 3.12
- UV (package manager)
- **MongoDB Community Edition** (must be running)
- MongoDB Compass (optional, for viewing data)

6.2 Installation

1. Install MongoDB

Download and install from: https://www.mongodb.com/try/download/community

Start MongoDB:

```
# Windows (PowerShell as administrator)
net start MongoDB
# Or run manually:
mongod
```

2. Package Installation Options

Choose **one** of the following options:

Option 1: Local Installation (Recommended for running)

This option uses `uv` to create a virtual environment and sync the exact project dependencies.

```
# 1. Clone the repository

git clone https://github.com/antonv18/DE_p3.git

cd DE_p3

# 2. Sync the environment and dependencies

uv sync
```

Option 2: Install from GitHub (Pip)

This option installs the package directly from GitHub using pip.

```
# Correct syntax using git+

pip install git+https://github.com/antonv18/DE_p3.git
```

Option 3: Local Installation (Editable for development)

This option clones the repository and installs the package in "editable mode," which means changes to the source code are reflected immediately.

```
# 1. Clone the repository

git clone https://github.com/antonv18/DE_p3.git

cd DE_p3

# 2. Install the package in editable mode

# (pip will install the dependencies listed in pyproject.toml)

pip install -e .
```

6.3 Configuring execution

Before executing the script, the user must ensure the configuration in the .env file is adapted to their local configuration.

```
DATA_DIR="data"        # Path of the DICOM dataset
DB_HOST="localhost" #Host of the DB
DB_PORT=207           # Port of the DB
```

6.4 Execution Instructions

Make sure you have MongoDB running and the data in the data/dicom_dir/ folder.

Option 1: Recommended if you used uv sync

```
# Runs the script defined in pyproject.toml
uv run run-pipeline-mongo
```

Option 2: Direct module

```
# Works with any installation method
uv run python -m de_p3.processing
# Or if not using uv:
python -m de_p3.processing
```