

Master IASD, 2025

Practice Session:

Document-oriented Databases

*Exploring an Anomaly Detection Methods Taxonomy**Paul Boniol*

This lab uses a real dataset of **133 time-series anomaly detection methods** and **13 algorithmic families** from <https://bonioldp.github.io/TSADtaxonomy/>. All queries are written in **Python** using the pymongo driver. **Part 1** is completed in class; **Part 2** is a take-home assignment on your **personalised dataset** (generated from a automatically computed seed number). If you do not manage to finish **Part 1** in class, you will need to finish it at home. The practice lab is long, and we will take it into account during the evaluation.

1 Setup

Installing MongoDB

Choose the procedure for your operating system. In all cases you need **MongoDB Community Edition 7.x** and the Python driver **pymongo**.

macOS — Homebrew

```
# One-time: trust and tap the MongoDB formula
brew trust mongodb/brew
brew tap mongodb/brew
# Install and start
brew install mongodb-community
brew services start mongodb-community
```

If `brew tap` fails with “invalid syntax”, run `brew trust mongodb/brew` first (needed on Homebrew 4.x+). If you see a “Command Line Tools too outdated” warning after installation, please update the Command Line Tools. You can stop MongoDB using the following command: `brew services stop mongodb-community`.

Linux — Ubuntu / Debian (apt)

```
# Import the MongoDB public key
sudo apt-get install -y gnupg curl
curl -fsSL https://www.mongodb.org/static/pgp/server-7.0.asc \
| sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg --dearmor
# Add the repository and install
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg ] \
https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" \
| sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list
sudo apt-get update && sudo apt-get install -y mongodb-org
# Start the service
sudo systemctl start mongod && sudo systemctl enable mongod
```

Windows — MSI Installer

1. Download the **MongoDB Community Server** MSI from <https://www.mongodb.com/try/download/community> (choose “Windows”, version 7.0).
2. Run the installer, select **Complete** setup, and check *Install MongoDB as a Service*. Make sure to untick **install MongoDB Compass**.
3. MongoDB starts automatically as a Windows service. To manage it, use the following command in the PowerShell:
`net start MongoDB / net stop MongoDB`
4. Add `C:\Program Files\MongoDB\Server\7.0\bin` to your PATH to use `mongosh` from any terminal.

Python environment (all platforms)

Create a dedicated **conda** environment, install the requirements, then launch JupyterLab:

```
# 1. Create and activate a fresh environment
conda create -n tsad_lab python==3.10 -y
conda activate tsad_lab

# 2. Install the required packages
pip install -r requirements.txt

# 3. Open the lab notebook
jupyter lab lab_notebook.ipynb
```

The `requirements.txt` file provided with the lab contains:

```
pymongo==4.7.*
jupyterlab>=4.0
```

Make sure the `tsad_lab` environment is active before running `jupyter lab`. If JupyterLab asks you to select a kernel, choose **Python 3 (ipykernel)** — it corresponds to the active conda environment.

Connecting and loading the data

Open the provided Jupyter notebook `lab_notebook.ipynb`. The first cells handle:

1. Connecting to MongoDB (`localhost:27017`).
2. Loading all JSON files from the `data/` folder into two collections: `methods` (133 documents) and `families` (13 documents).

Note: For **Part 2** you must reload the data with your seed before answering questions H1–H8. A dedicated cell in the notebook handles this — just set `your_name` to your last name (i.e., family name).

2 Data Schema

The methods collection

Each document represents one anomaly detection algorithm. The latter may contain the following attributes:

Field	Type	Description
name	string	Short identifier, e.g. "AE", "LSTM-AD"
full_name	string	Full algorithm name
category	string	Algorithmic family: "Reconstruction-based", "Forecasting-based", ...
Dim	string	"Univariate" or "Multivariate"
Sup	string	"Unsupervised", "Semi-supervised", or "Supervised"
Stream	boolean	True if the method supports online/streaming processing
year	integer	Publication year
authors	array	List of author name strings
paper	string	Paper title and venue
description	string	Plain-language description
url	string	URL to the paper
code	string	URL to an open-source implementation (when available)
snippet	string	Python code example (present for 13 methods only)

The families collection

Each document describes a high-level algorithmic family:

Field	Type	Description
name	string	Family name, e.g. "Reconstruction-based"
text	string	Description of the family and its general principles
year	integer	Year the family concept was first established

The `category` field in `methods` matches the `name` field in `families`: every method belongs to a family that has a corresponding family document. This link will be useful in Part 2.

Warning

The database contains inconsistencies **on purpose**. Some words have inconsistent capitalization (Sup or sup), and some names are either fully written out or use only the initial (Themis Palpanas or T. Palpanas). Despite these small inconsistencies, they all refer to the same entity (e.g., Themis Palpanas and T. Palpanas are the same person, and Sup and sup are the same category).

3 Part 1 — Getting Started (in-class)

These questions should be answered **during the lab session** and submitted as part of the answer sheet. They cover insertion, retrieval, filtering, projection, sorting, aggregation, and schema validation.

Section A — Inserting and Exploring the Data

1. Run the setup cells in the notebook to load both collections. Verify the insertion by displaying all documents from each collection (use `find()` with no filter).
2. Count the total number of documents in the `methods` collection.
3. Count the total number of documents in the `families` collection.
4. List all **distinct category names** present in the `methods` collection.

T1 - Relational Model

Suppose that, instead of MongoDB, you must store the same data (the `methods` and `families` collections, including the `rating` and `citations` fields of Part 2) in a **relational database** (e.g. PostgreSQL).

- (a) Propose a relational schema able to store these data. For each table, give its attributes and their types, its **primary key**, its **foreign keys**, and any relevant integrity constraints (`NOT NULL`, `UNIQUE`, `CHECK`, ...). You may use the relational notation, e.g. `TABLE(key, attribute, ..., #foreign_key)`, or SQL `CREATE TABLE` statements.
- (b) Justify your design. Which fields of the JSON documents cannot be stored *as is* in a relational table, and how do you handle them? Which normal form does your schema satisfy? How does your schema handle the inconsistencies mentioned in Section 2?

Section B — Filtering and Projection

5. Find all methods that handle **Multivariate** time series. Display only `name` and `category`.
6. Find all methods published **after 2018** that support **streaming** (`Stream=True`). Show `name`, `year`, and `category` only.
7. Find the **5 most recently published** methods, showing `name`, `year`, and `category`. Sort them first by year descending then by name ascending.
8. Find all methods whose `paper` field contains the word “**neural**” (case-insensitive).
9. Find all methods that have a code repository — i.e., the `code` field exists *and* is not an empty string.
10. Find all methods that do **not** have a `snippet` field. Display only their names.
11. Find methods published between **2015 and 2020 inclusive** that are **Unsupervised**. Sort by year ascending then name ascending.
12. Find all methods where **Eamonn Keogh** is listed as an author.

T2 — Sharding Strategy

We now imagine a much larger version of the `methods` collection (e.g. hundreds of millions of documents describing papers), which no longer fits on a single server and must therefore be **distributed (sharded)** over several servers. We assume that the **most frequent queries** retrieve papers published within a given **time range**.

- (a) What is the most commonly used sharding strategy in MongoDB? Describe how documents are distributed among the servers with this strategy, and how the query above would be executed.
- (b) Which sharding strategy would you choose for this workload? Justify your choice by comparing it with the strategy of (a).
- (c) What are the limits or risks of the strategy you proposed in (b), and how could you mitigate them?

Section C — Aggregation

- 13.Count the number of methods **per category**. Sort from the largest to the smallest group. In case of equality, sort by lexicographical order based on the category name.
- 14.Find all categories that contain **more than 10 methods**.
- 15.Compute the **number of methods per supervision type** (Sup field). *Hint: inspect the field values carefully — there may be inconsistencies in capitalisation.*
- 16.List all **distinct author names** across all method documents. How many unique authors are there in total?
- 17.For each category, compute the **average publication year**. Sort by average year descending.
- 18.Count the number of **streaming** methods per category. Only show categories with at least one streaming method. Sort in descending order based on the number of streaming methods per category. In case of equality, sort by lexicographical order based on the category name.

T3 — Ranking Documents for a Natural-Language Search Engine

We want to build a **search engine** on top of the `methods` and `families` collections. A user types a free-text query in natural language, for instance:

“unsupervised streaming methods based on distance by author X.”

and the engine must return the methods sorted from the most to the least relevant.

This is an **open question**. Explain how you would compute the **relevance score** of each method for a given query, and therefore the ranking.

Section D — Updates and Schema Validation

- 19.Add a field `era` to every method document: `"classic"` for methods published before 2010, `"modern"` for 2010 or later. Verify by counting the documents in each era.
- 20.Create a new collection `validated_methods` with a `$jsonSchema` validator enforcing: (a) `name` is required and must be a string; (b) `year` is required and must be an integer; (c) `Dim`, if present, must be `"Univariate"` or `"Multivariate"`. Insert one valid and one invalid document to confirm the schema works.

4 Part 2 — Take-Home Assignment (personalised)

These questions require advanced aggregation pipelines (`$group`, `$lookup`, `$unwind`, `$project`, `$sort`, `$limit`, ...). Each student works on a **unique dataset**: answers cannot be shared.

For each question **H1–H7**, you must also provide a brief written explanation of your approach (see the prompt below each question). For **H8**, a written discussion is the central part of the answer. **H9** is an open-ended design question; no MongoDB query is expected.

4.1 Generating Your Personal Dataset

You are assigned a **personal seed number** (computed automatically from your last name). In the notebook, set `your_name = <your_name>` in the Part 2 setup cell and run it. This will reload the `methods` collection with two extra fields added to every document:

- **rating** — integer in `[1, 5]`, simulated peer-review score.
- **citations** — integer in `[50, 5000]`, simulated citation count.

These values are deterministic for your seed but differ between students.

4.2 Questions

↪ **Process explanation:** *For each question, describe the main steps of your pipeline. Which operators did you use and why? Were there any edge cases or pitfalls to watch out for?*

H1 — Category Performance Summary

For each category, compute (a) the **average rating** (rounded to 2 decimal places) and (b) the **total number of citations**. Return only categories with **at least 5 methods**, sorted by average rating descending, then total citations descending.

H2 — Prolific Cross-Category Authors

Find all authors who have co-authored papers in **at least 3 different categories**. For each such author, return their name and the **sorted list of distinct categories** they published in. Sort the final result by number of categories descending, then by author name alphabetically.

H3 — Highest-Rated Category

Find the category where the **proportion of methods with rating ≥ 4** to the total number of methods in that category is the highest. In case of a tie, return the category with the highest **total citations**. Return the category name, the proportion (rounded to 3 decimal places), and the total citations.

H4 — Top Cited Methods in Recent High-Performing Categories

Consider only methods published **between 2015 and 2020 inclusive**. Among these, find the **top-3 categories** by average citations. For each of these 3 categories, return the **single method with the highest individual citation count** (name, category, citations, year).

H5 — Family–Method Cross-Collection Join

For each document in the `families` collection whose `name` matches a `category` in `methods`, return: (a) the family name, (b) the number of methods in that category, (c) the average rating (rounded

to 2 decimal places), (d) whether any method in that category has a **snippet** field. Sort by average rating descending.

H6 — Most Productive Year

Find the **single year** in which the largest number of **distinct authors** published at least one method. Return that year, the count of distinct authors, and the **list of author names** (in lexicographical order).

H7 — Supervision-Type Deep Dive

For each **supervision type** (Sup field), produce a summary containing: (a) the supervision type, (b) total number of methods, (c) the **most common category** among those methods, (d) the average publication year (rounded to 1 decimal place), (e) the **names of the top-2 most cited methods**.

Sort the results by total methods, descending.

H8 — Co-authorship Shortest Path

Two authors are **co-authors** if they appear together in the **authors** list of at least one method. This defines a graph where authors are nodes and edges connect co-authors. The objective is to find the shortest path between two authors of your choice, including all *intermediate* nodes.

(a) **A Pure MongoDB attempt.** Build a **co_authors** collection in MongoDB, one document per author, with a **co_authors** field listing that author's collaborators. Write a **\$graphLookup** aggregation that tries to address the objective above.

(b) **A Python attempt.** Implement a function **find_path(start, end)** on an adjacency dictionary built from MongoDB. Test it on the following three pairs and report the path and number of hops:

Pair	Author A	Author B
1	'Eamonn Keogh'	'John Paparrizos'
2	'Eamonn Keogh'	'D. Papadopoulos'
3	'Pavel Senin'	'D. Papadopoulos'

(c) Discuss the feasibility, the advantages and the limits of both (a) and (b). Can you imagine an alternative to (a) and (b)?

H9 — Academic Descendants

Consider the following query:

“Which author has the most descendants that published at least one paper in the database?”

Here, **descendant** means an academic descendant in the following sense (i.e., Math Genealogy Project): a PhD student advised by the author, or a student of a student, etc. This is **not** a co-authorship relationship.

(a) Can you answer this query directly with the current database? Explain why or why not.

(b) Propose a high-level solution that would allow you to answer the query above. You do *not* need to implement the solution; a clear, technically precise description is sufficient. Such a solution

should be as much independent of the database content as possible (i.e., applicable to any new database sharing the same schema). Discuss the feasibility of your proposed solution.

5 Useful Resources

- MongoDB manual:
<https://www.mongodb.com/docs/manual/tutorial/query-documents/>
- MongoDB aggregation pipeline:
<https://www.mongodb.com/docs/manual/core/aggregation-pipeline/>
- Aggregation operators:
<https://www.mongodb.com/docs/manual/reference/operator/aggregation/>
- \$lookup:
<https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/>
- \$graphLookup:
<https://www.mongodb.com/docs/manual/reference/operator/aggregation/graphLookup/>
- pymongo documentation:
<https://pymongo.readthedocs.io/>
- TSADtaxonomy:
<https://boniolp.github.io/TSADtaxonomy/>