Southern New Hampshire University

CS-340: Assignment 2-1: Module 2 Journal

Prepared on: September 14, 2025

Prepared for: Prof. Jeff H. Sanford

Prepared by: Alexander Ahmann

Contents

1 Introduction

This journal serves as documentation for the tasks that I was assigned in module 2's assignment regarding further data *extraction, loading, and transformation* with the MongoDB non-relational database system (CS-340, n.d.). This assignment had students work with hypothetical "city inspections" data — which is presumably some kind of database for keeping track of unethical or illegal conduct of various for-profit institutions. Concepts in non-relational databases, along with their implementation in the MongoDB service, covered include: *insertion* of new documents, further development of *find* queries, *updating* existing documents at an individual and mass scale, and *removal* of specific documents.

1.1 Brief description of the Figures

To demonstrate that the tasks were carried out, I have embedded a number of figures depicting screenshots that show the results of executing the given tasks on an online `mongosh` shell. In the next section, I elaborate more on what was specifically done, but the following are a brief description of the figures:

- Figure 1 depicts a solution where I make use of the `mongoimport` utility to take a JSON dataset and put them into the *MongoDB* instance.
- Figures 2, 3, and 4 depicts the results to basic data modelling with the `find*()` of queries.
- Figure 5 depicts the use of `insertOne` queries to insert a new document with specific fields and their respective values.
- Figures 6, 7, 8, and 9 depicts more data modelling witht the `find*()` class of queries, along with the `distinct` query.
- Figure 10 depicts a solution where I use the `updateOne()` query to update a document that meets a specific filter conditional.
- Figures 11, 12, and 13 depict the results of using `updateMany` to update multiple `inspections` documents, and the use of the `deleteOne` query to remove a single document that meets a specific filter conditional.

2 Problem Set

2.1 Task 1: Importing the `city` database and its respective `inspections` collection

> "Access the dataset for this assignment, `city_inspections.json`, in the datasets directory in your Codio environment. Using the `mongoimport` tool and documents found in the `city_inspections.json` file, load the database `city` into the inspections collection. Complete this task by typing the following commands in the Linux terminal to perform the import in the right directory." —CS-340 (n.d., §1 of prompt)

The given solution is straightforward: I logged into the *Codio* database, and issued the following command to import the database into the *MongoDB* service:

```
mongoimport --db city --collection inspections
    --drop ./city_inspections.json
```

The given command will import the `./city_inspections.json` dataset into the `city` database, and specifically store each entry as a document in the `inspections` collection. Figure 1 depicts a screenshot that demonstrates the execution of this task.

2.2 Task 2: Confirming the status of the import

> "Verify your load by "using" the "city" database, and issuing the following queries in the mongo shell:
>
> ```
> 1. db.inspections.find({
>     "id" : "10021-2015-ENFO"
> })
>
> 2. db.inspections.find(
>     {"result":"Out of Business"},
>     {"business_name":1
> }).limit(10)
> ```
>
> Provide screenshots of the results as evidence." —CS-340 (n.d., §2 of prompt)

Like with the first task, this one was pretty straightforward. I simply entered in the given commands into the `mongosh` interface. Figures 2, 3, and 4 depicts multiple screenshots that demonstrates the execution and results of this task. Specifically, figure 2 depicts the results of the first executed command, and figures 3 and 4 depicts the results of the second executed command.

2.3 Task 3: Inserting an arbitrary document

> "Using the appropriate commands in the mongo shell, insert a document to the database named `city` within the collection named `inspections`. Use the following key-value pairs [as depicted in Table 1] as data for your document. Be sure to insert the address as a subdocument and use the JavaScript function `Date()` for 'Today's date.' Verify your database creation and insertion using the `findOne()` function in the mongo shell. Provide a screenshot as evidence." —CS-340 (n.d., §3 of the prompt)

Key	**Value**
id	"20032-2020-ACME"
certificate_number	9998888
business_name	"ACME Explosives"
date	Today's date
result	"Business Padlocked"
sector	"Explosive Retail Dealer-999"
address	number -> 1721 street -> Boom Road city -> BRONX zip -> 10463

Table 1: Hypothetical data to be inserted as a document into `inspections`.

The data to be inserted is shown in table 1. The exact MongoDB database query that I used to insert the data as depicted in this table is:[1]

[1]See Giamas (2022, §3.2: *CRUD using the shell* for constructing queries regarding inserting new documents). Also consult the following resource: MongoDB Docs. (n.d.). *db.collection.insertOne()*. Last retrieved on Sept. 14, 2025: https://www.mongodb.com/docs/manual/reference/method/db.collection.insertOne/

```
db.inspections.insertOne({

    _id:"20032-2020-ACME",
    certificate_number:9998888,
    business_name:"ACME Explosives",
    date:new Date(),
    result:"Business Padlocked",
    sector:"Explosive Retail Dealer-999",
    address:{
        number:1721,
        street:"Boom Road",
        city:"BRONX",
        zip:10463
    }
})
```

Some "interesting" fields that constitute the document to be inserted are `_id`, `date`, and `address`. The `_id` is a unique string that identifies the document in the collection, the date is something of a "dynamic wildcard" that is dependent on the state of the database system—- specifically the current date and time, and the address is another dictionary within the current document, which itself is expressed in terms of a dictionary.

After executing the insert query statement, the command line returned the following bit of information to confirm that a transaction was completed:

```
{ acknowledged: true, insertedId: '20032-2020-ACME' }
```

I then confirmed that the transaction completed successfully with the following query statement:[2]

[2]See Giamas (2022, §2-4) as a reference to data modelling and constructing `find*()` non-relational queries. Also see the following resources:

- Factor, P. (Feb. 17, 2020). *MongoDB find Method: Introduction & Examples.* Studio 3T Knowledge Base. Last retrieved on Sept. 14, 2025 from: https://studio3t.com/knowledge-base/articles/mongodb-find-method/
- MongoDB Docs (n.d.). *db.collection.find().* Last retrieved on Sept. 14, 2025 from: https://www.mongodb.com/docs/manual/reference/method/db.collection.find/
- MongoDB Docs (n.d.). *db.collection.findOne().* Last retrieved on Sept. 14, 2025 from: https://www.mongodb.com/docs/manual/reference/method/db.collection.findOne/

```
db.inspections.findOne({
    _id:"20032-2020-ACME"
})
```

Figure 5 depicts a screenshot demonstrating the execution of this task, and its respective results.

2.4 Task 4: Working with MongoDB search queries

This task is composed of three smaller tasks that centre on the theme of data modelling and the `find*()` class of MongoDB queries. The first one concerns itself with counting distinct entities: "*[w]hat is the distinct list of inspection 'sector' in the current inspections collection? How many are in the list? Do not count by hand*" (CS-340, n.d., §4.1). To answer this, I began with the following MongoDB query:[3]

```
db.inspections.distinct("sector")
```

This was just to get an idea of what distinct sectors look like. I then got the exact count of the distinct sectors by appending a `.length` to the previous query:

```
db.inspections.distinct("sector").length
```

This resulted in `87` distinct kinds of sectors. Figure 6 depicts a screenshot which demonstrates the execution of the procedure.

The next subtask concerns itself with data types: "*What is the difference in the date data type for the business named 'AUSTIN 2012' versus your business document insertion of 'Acme Explosives?'*" (CS-340, n.d., §4.2). To answer this question, I used the `typeof` operator in MongoDB queries for the `AUSTIN 2012` and `ACME Explosives` databases:

[3]See Giamas (2022, §7) for working with `distinct` values. Also, see the following resources:

- Geeks4Geeks (c.a. Jul. 23, 2025) *MongoDB - Distinct() Method*. Last Retrieved on Sept. 14, 2025 from: https://www.geeksforgeeks.org/mongodb/mongodb-distinct-method/
- MongoDB Docs (n.d.). *db.collection.distinct()*. Last Retrieved on Sept. 14, 2025 from: https://www.mongodb.com/docs/manual/reference/method/db.collection.distinct/

```
typeof db.inspections.findOne({
    business_name: "AUSTIN 2012"
}).date // for the date type of "AUSTIN 2012"

typeof db.inspections.findOne({
    business_name: "ACME Explosives"
}).date // for the date type of "ACME Explosives"
```

Regarding the `AUSTIN 2012` business, it returned a `string` data type for the data field, and regarding the `ACME Explosives` business, it returned a `object` data type for the date field. From this, I can infer that MongoDB type databases are more flexible in how they process fields.[4] Figure 7 depicts a screenshot that demonstrates the results after executing the task.

The final subtask concerns itself with `distinct` queries that return data conditional to some criteria: "*[h]ow many businesses have a 'Violation Issued?'*" To answer this question, I used the following `distinct` query to return the count documents conditional on `business_name` having a value of `Violation Issued`:

```
db.inspections.distinct(
    "business_name",
    { result:"Violation Issued"}
).length
```

The result was `10602` distinct records, see Figure 8 for a sample of the records return, and Figure 9 for the count of total distinct records that meet the criteria.

2.5 Task 5: Updating an arbitrary document

> "Using the appropriate command in the mongo shell, update the document with the ID `20032-2020-ACME` in the collection `inspections` in the database `city` with the information [in Table 2.5]. Verify your database update using the appropriate `find()` function in the mongo shell." —CS-340 (n.d., §5)

[4] In previous experience working with the JSON file format, without the use of MongoDB, I noticed that, when iterating through a JSON flat file, not all entries have a certain field. For example, when using a Python for-loop to iterate through a `.jsonl` file and searching for a hypothetical field "field1", some entries did not have a "field1", causing earlier versions of my scripts to crash. I overcame this problem with an exception handler.

Key	Value
business_name	"New ACME Flowers"
result	"Business Re-opened"
comments	"Flowers after the explosion"

Table 2: Hypothetical key-value pairs by which to update the "20032-2020-ACME" document in the `inspections` table.

To answer this question, I had to use the `updateOne` query which is conditional to the `_id` of the company that I am interested in making changes to.[5] The exact query that I used to update the record with the `{ "_id" : "20032-2020-ACME" }` key-value pair is:

```
db.inspections.updateOne( { "_id":"20032-2020-ACME" }, {
    $set: {
        "business_name":"New ACME Flowers",
        "result":"Business Re-opened",
        "comments":"Flowers after the explosion"
    }
})
```

I then used the following query to confirm that the update procedure was successful:

```
db.findOne( {"_id": "20032-2020-ACME"} )
```

Figure 10 depicts a screenshot which demonstrates this task's completion and its results.

[5]See Giamas (2022, §3.2) for working with `update*() delete*()` classes of queries. Also see the following resources:

- Geeks4Geeks (Jul. 23, 2025). *MongoDB - updateOne() Method*. Last Retrieved on Sept. 14, 2025 from: https://www.geeksforgeeks.org/mongodb/mongodb-updateone-method-db-collection-updateone/
- MongoDB Docs (n.d.). *db.collection.updateOne()*. Last retrieved on Sept. 14, 2025 from: https://www.mongodb.com/docs/manual/reference/method/db.collection.updateOne/
- MongoDB Docs (n.d.). *Delete Documents*. Last retrieved on Sept. 14, 2025 from: https://www.mongodb.com/docs/drivers/node/current/crud/delete/

2.6 Task 6: Updating many documents, and removing an arbitrary document

This task is divided up into two subtasks: one involving updating many documents in the `inspections` collection, and the other is removing a specific document that meets a filter criteria. The first subtask states to: "*[u]pdate all the documents that contain the key-value pair* `{ "city":"ROSEDALE" }` *in the address subdocument by changing the zip code in the address subdocument to* `76114`" (CS-340, n.d., §6.1). To accomplish this task, I used an `updateMany` query.[6] Specifically, the exact query that I used to update all ZIP codes in documents that contain the `{ "address.city" : "ROSEDALE" }` entry is:

```
db.inspections.updateMany({
    "address.city":"ROSEDALE"
},
{
    $set: {
        "address.zip": 76114
    }
})
```

To verify the successfulness of this transaction, I then use the following query:

```
db.inspections.findOne( {"address.zip": 76114} )
```

Figures 11 and 12 depicts screenshots showing the successfulness of the `updateMany` transaction, and the results of the `findOne` query.

The final subtask, and indeed the final job of this assignment, is to remove an arbitrary document from the `inspections` collection: "[r]emove the first document with the key-value pair `"result" : "Violation Issued."`" (CS-340, n.d., §6.2). To accomplish this task, I used a `deleteOne()` query, and then verified its successfulness with a `find()` queries before and after the execution. Specifically, I used the following `deleteOne` query:

```
db.inspections.deleteOne({"result":"Violation Issued"})
```

Figure 13 depicts screenshots that demonstrates the results after executing these queries.

[6]See the following resource when referencing `updateMany` queries:
MongoDB Docs (n.d.). *db.collection.updateMany()*. Last retrieved on Sept. 14, 2025 from: https://www.mongodb.com/docs/manual/reference/method/db.collection.updateMany/

3 Summary

This assignment acts to demonstrate this student's competence in the extracting, loading, and transformation of data with the `mongoimport` utility, and specific use cases for *Create, Read, Update, and Remove* features of the MongoDB database management system. Regarding CRUD features:

- I used the `find*()` class of queries, and the `distinct()` query, to return documents that meet some filter conditional, and to remove redundancies.
- I used the `insertOne()` query to insert new documents into a MongoDB collection.
- I used the `update*()` class of queries to update values for specifc keys across a specific document, or even many document, based on some filter conditional.
- I used the `deleteOne()` query to remove a single document that meets a filter conditional.

The bulk of this assignment was more focused on CRUD features of MongoDB, as opposed to ETL methods and utilities. Nonetheless, the importance of knowing how to use the `mongoimport` utility is emphasised in the first task.

References

CS-340 (n.d.). *CS-340 Module Two Assignment Guidelines and Rubric.*

Giamas, A. (2022). *Mastering MongoDB 6.x: Expert Techniques to Run High-volume and Fault-tolerant Database Solutions Using MongoDB 6.x.* Birmingham, UK: Packt Publishing, 2022. Last retrieved on Sept. 12, 2025 from: https://research.ebsco.com/linkprocessor/plink?id=a5bcc20e-3306-36b5-ad4f-0d0bd1f1567e

A Appendix: Figures depicting the results of tasks worked out

A.1 Figure 1: Importing the `inspections` collection

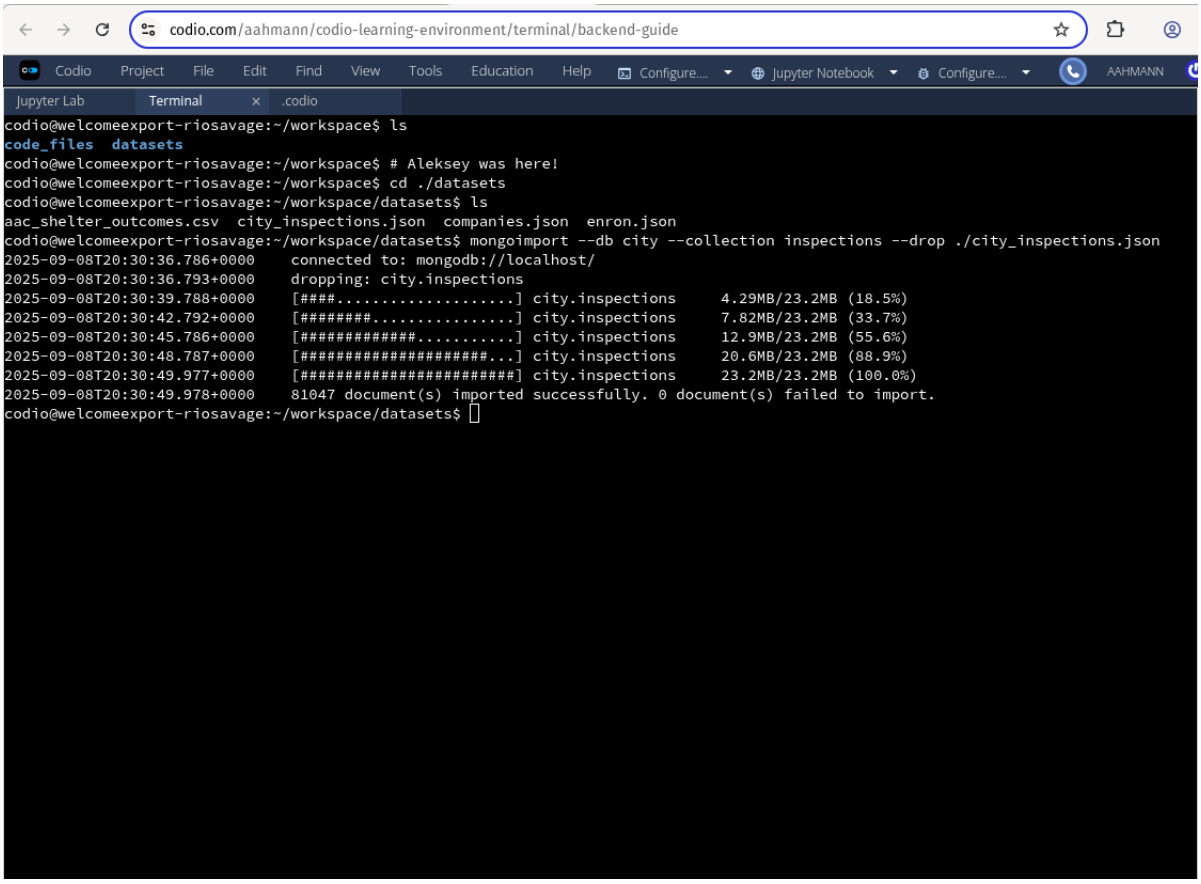

Figure 1: Importing the `inspections` collection.

A.2 Figures 2, 3 & 4: Examples of `find*()` queries against the `inspections` collection

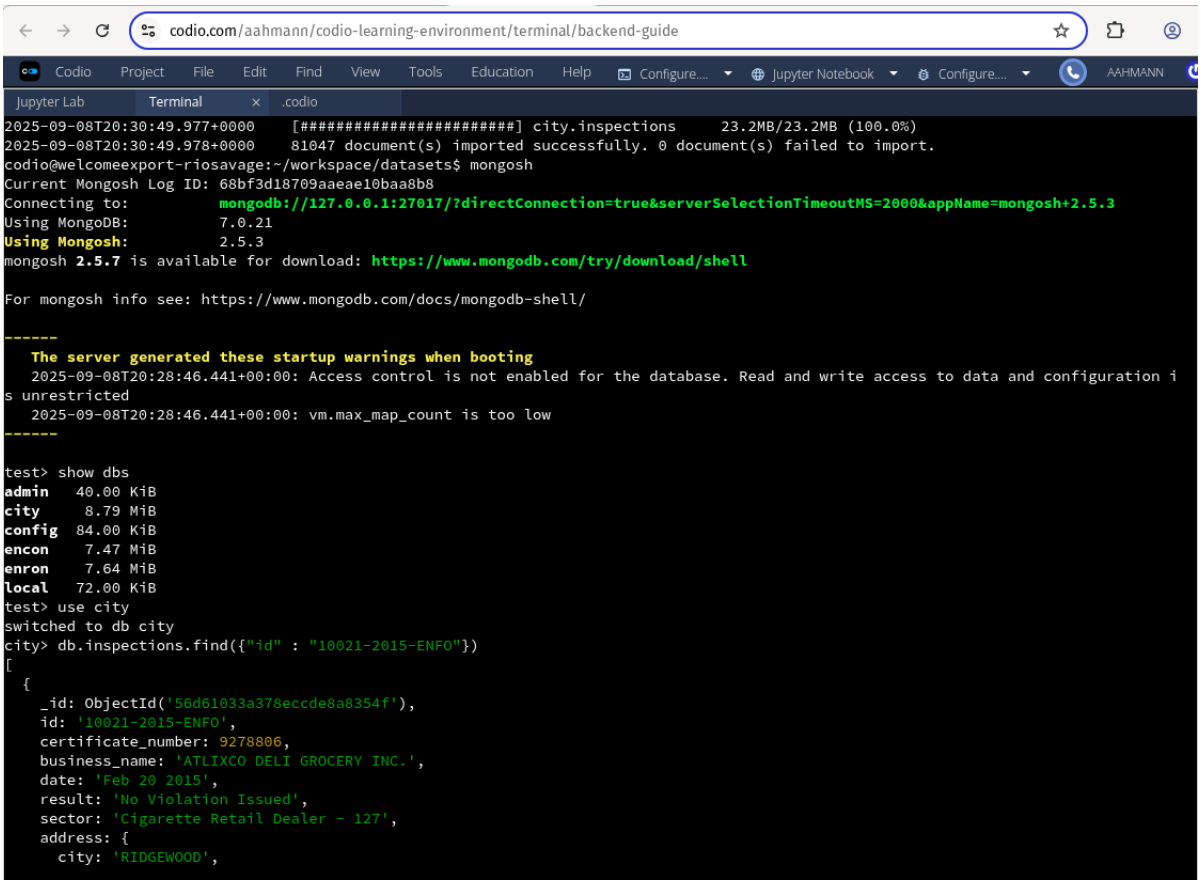

Figure 2: Results of `db.inspections.find({"id":"10021-2015-ENFO"})`

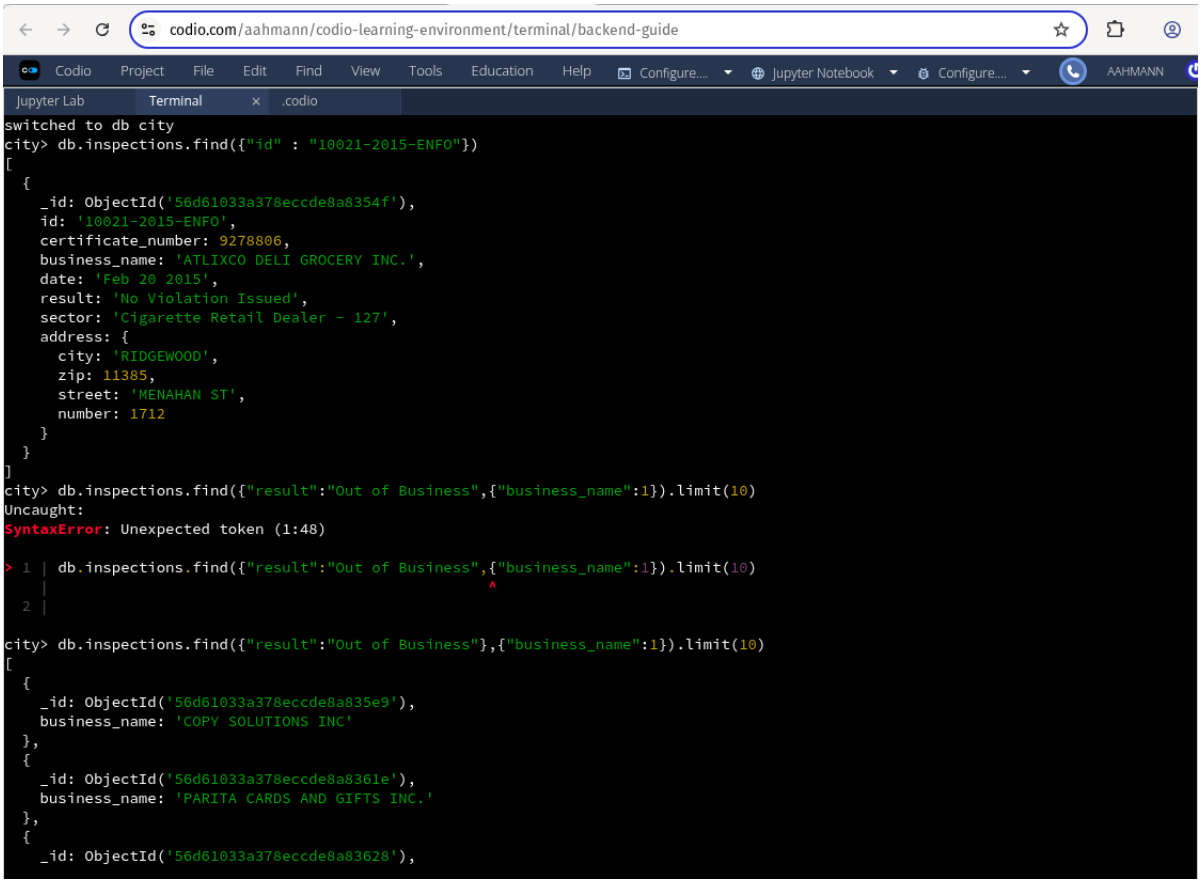

Figure 3: Results of `db.inspections.find({"result":"Out of Business"}, {"business_name":1}).limit(10)` (with mistakes, for the sake of full disclosure).

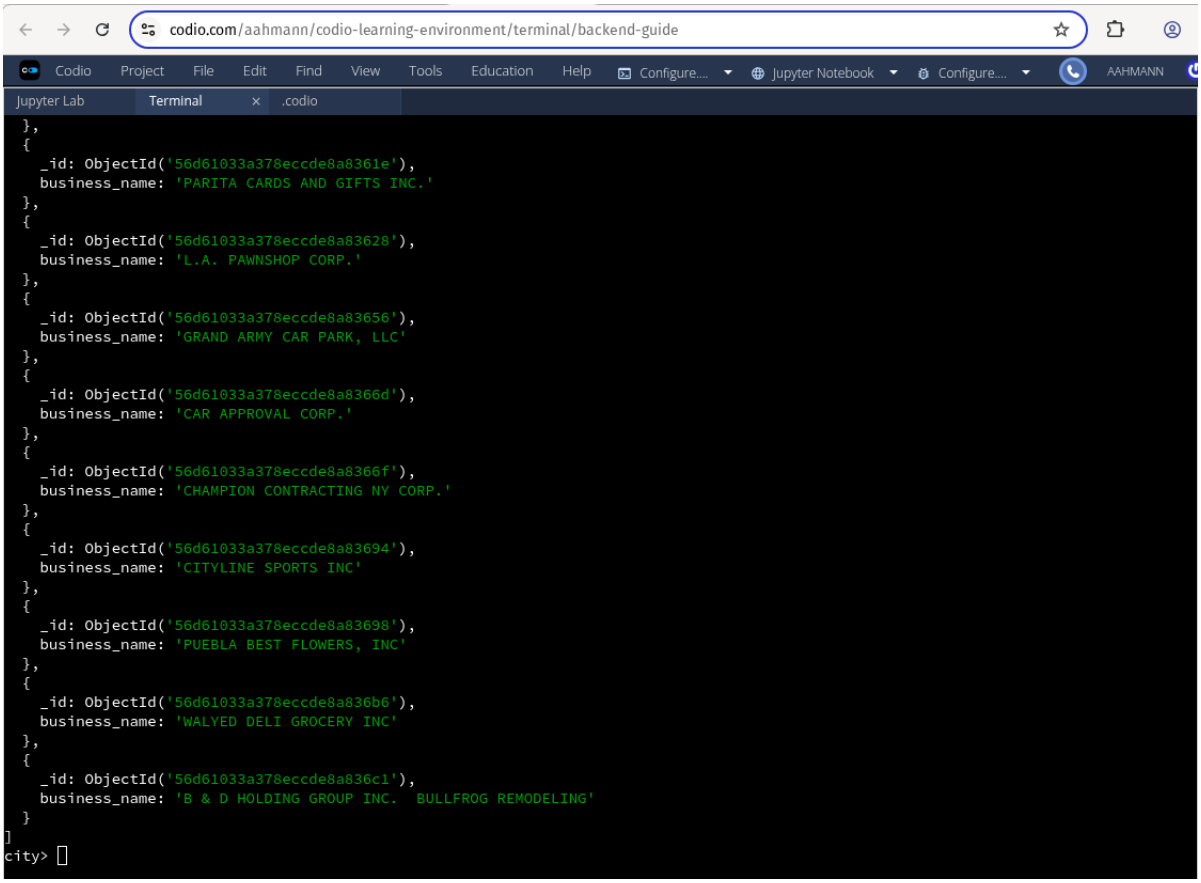

Figure 4: Results of `db.inspections.find("result":"Out of Business", "business_name":1).limit(10)` (cont.)

A.3 Figure 5: Insertion of a new document

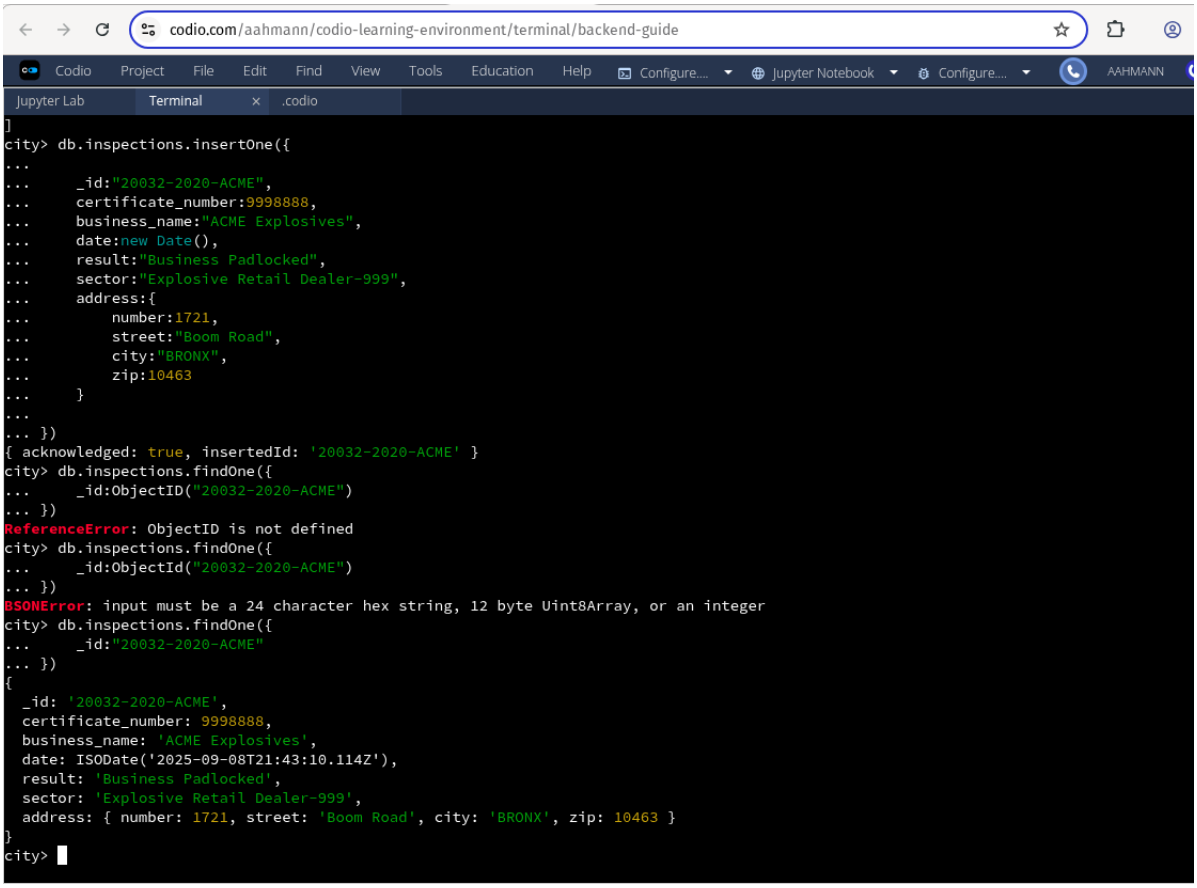

Figure 5: Results of inserting a new document in accordance with Table 1 (with mistakes, for the sake of full disclosure).

A.4 Figures 6, 7, 8, & 9: Sub-Tasks involved in data modelling

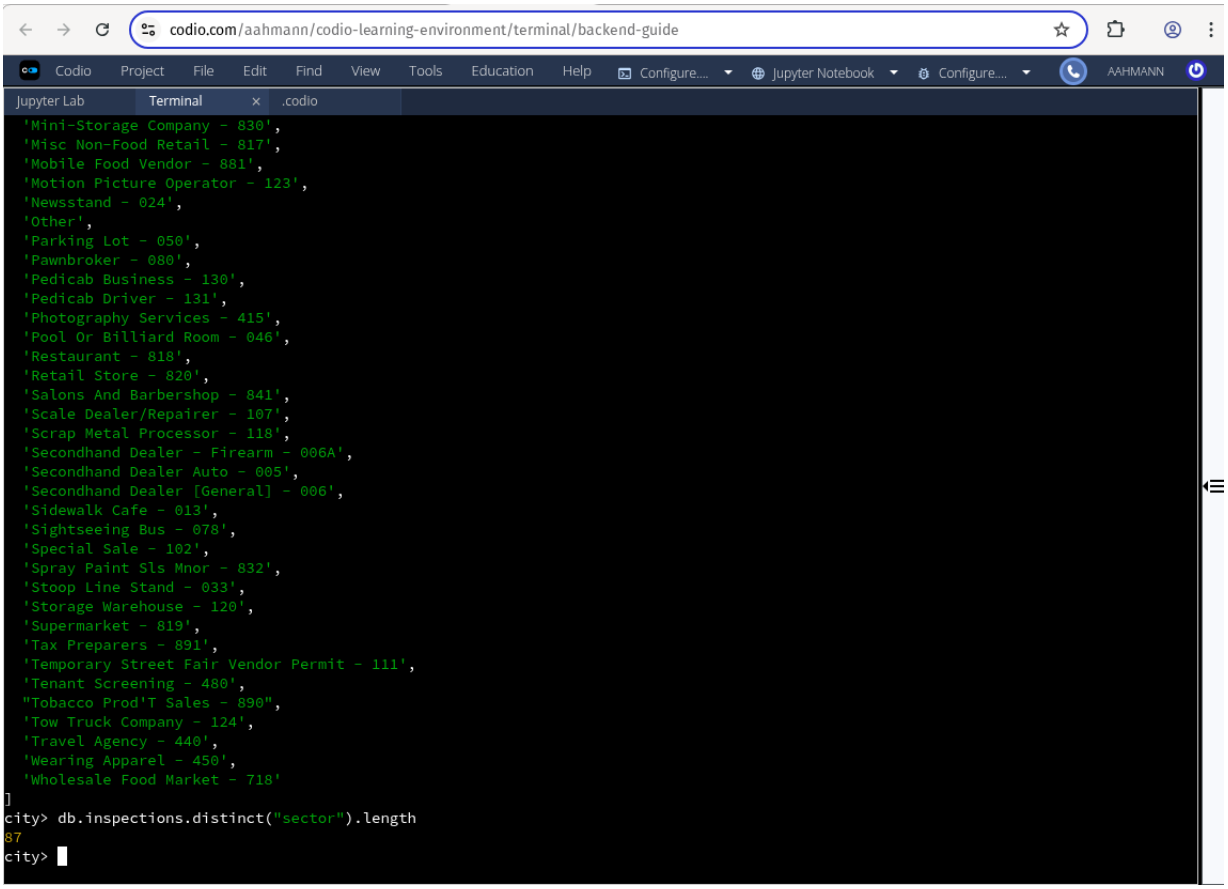

Figure 6: Distinct count of documents with the `sector` field.

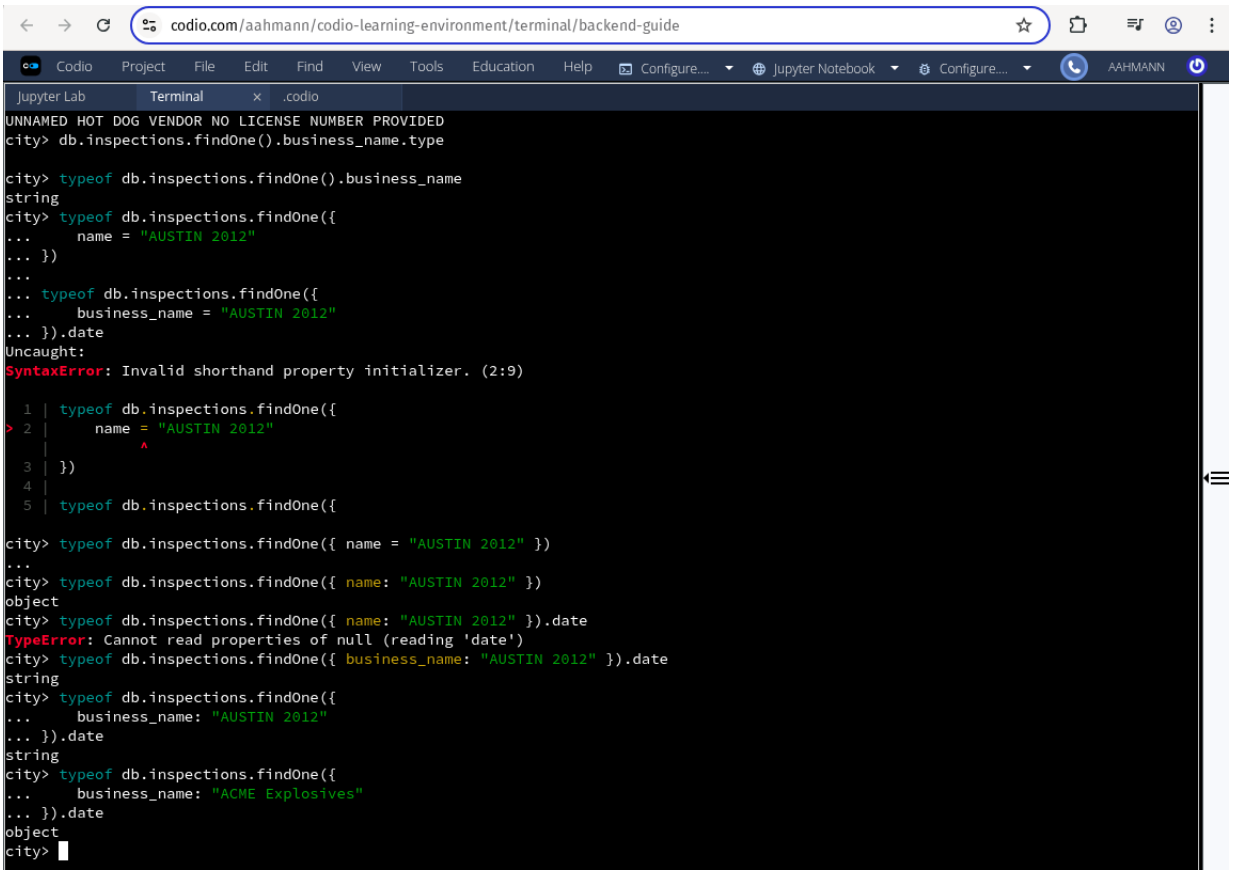

Figure 7: Comparison of the `typeof _id` fields for "AUSTIN 2012" and "ACME Explosive" (with mistakes, for the sake of full disclosure).

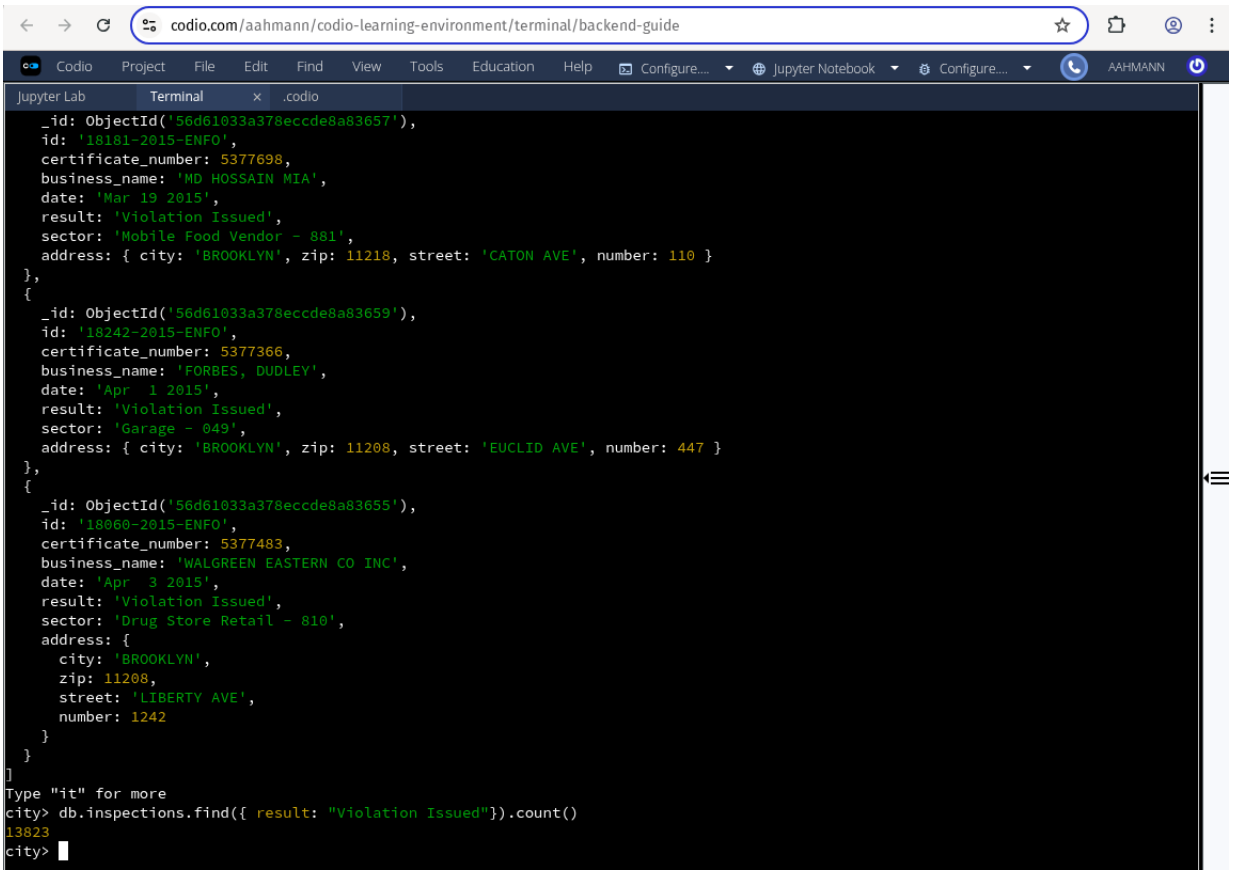

Figure 8: Sample of businesses with "Violations Found".

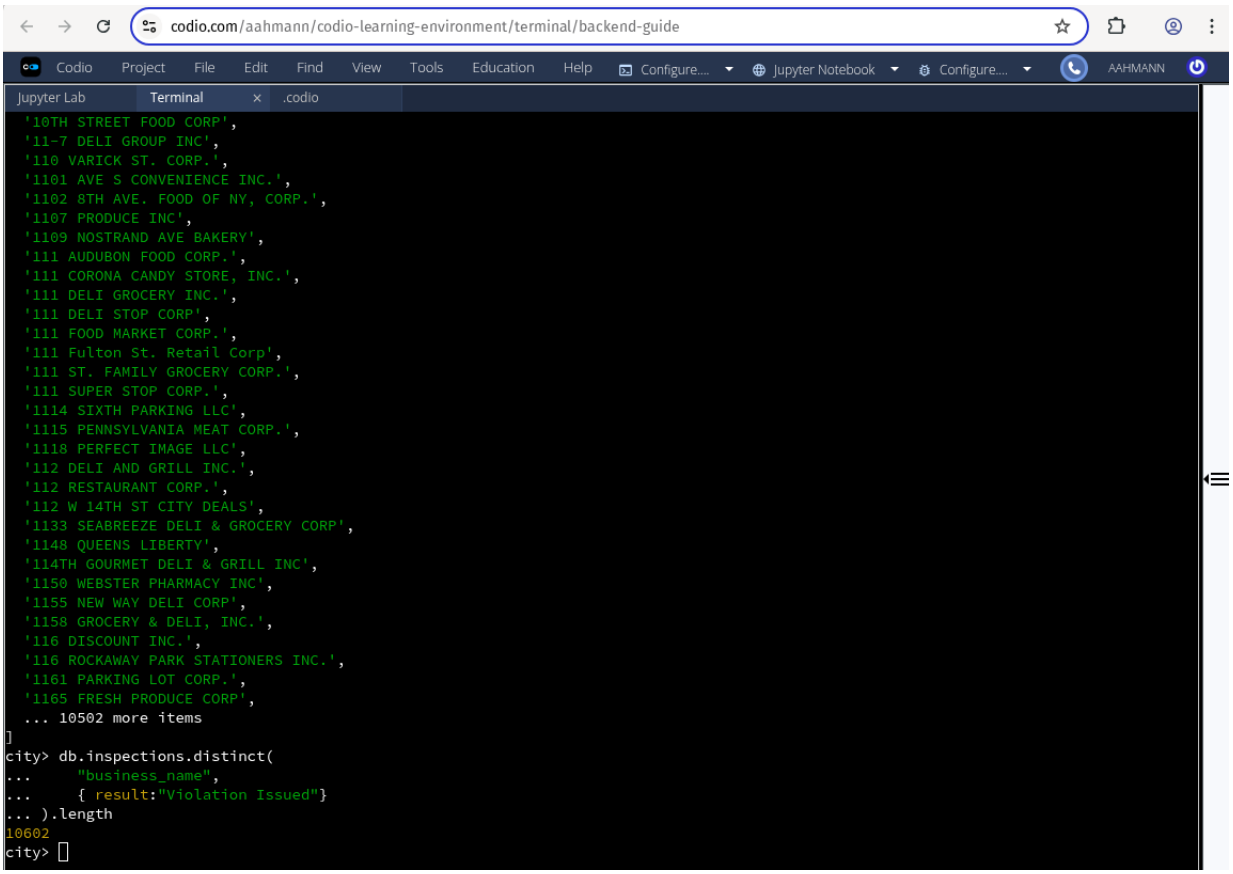

Figure 9: Count of businessed with “Violations Found” (with a few more samples).

A.5 Figure 10: Updating a single document

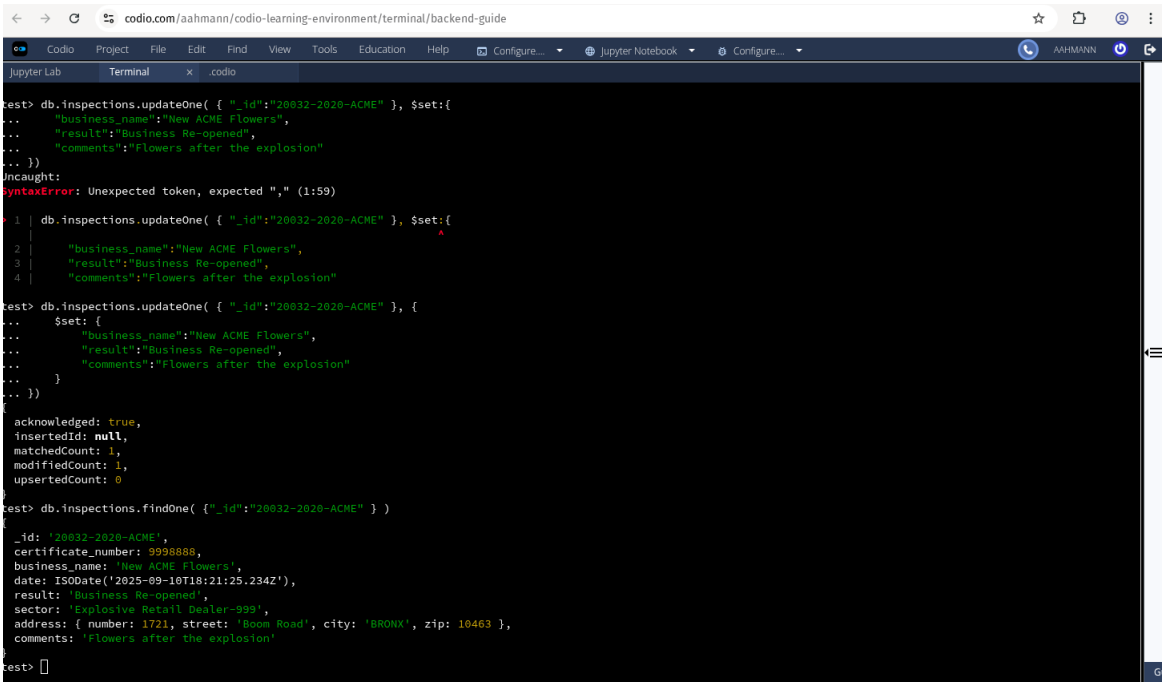

Figure 10: Updating a single document in the `inspections` collection (with mistakes, for the sake of full disclosure).

A.6 Figures 11, 12 & 13: Sub-Tasks relating to mass updating and removal of a specific document

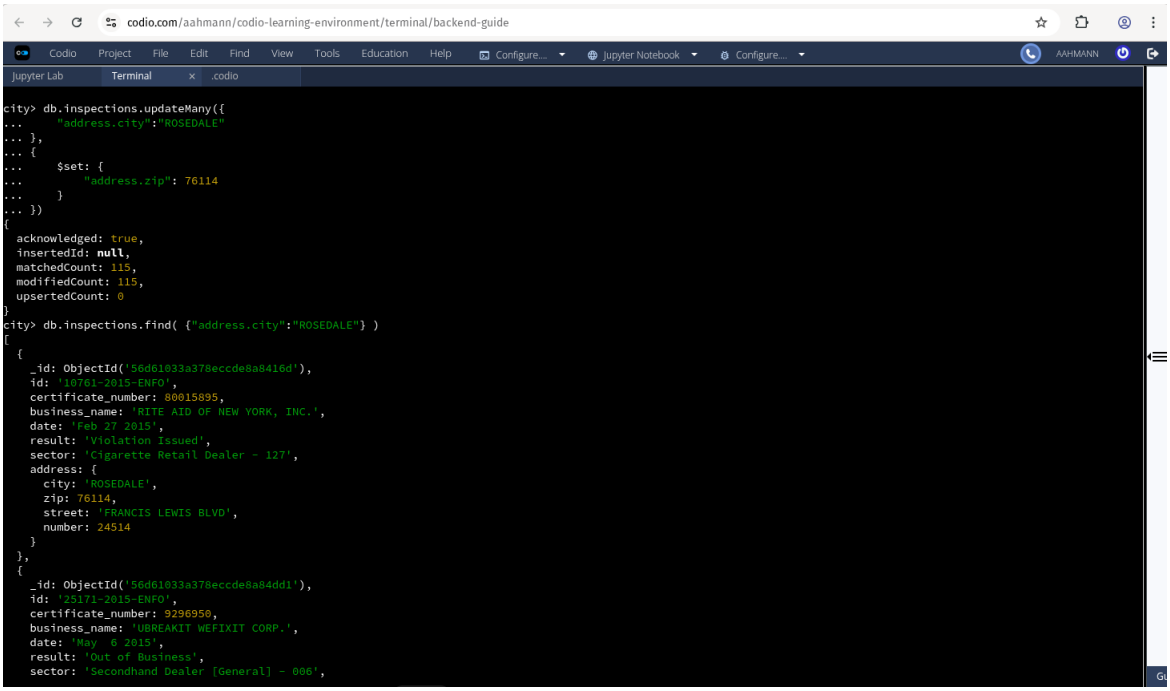

Figure 11: Updating documents in the `inspections` collection that meet the criteria `{ "address.city":"ROSEDALE" }` with a ZIP code of `76114`.

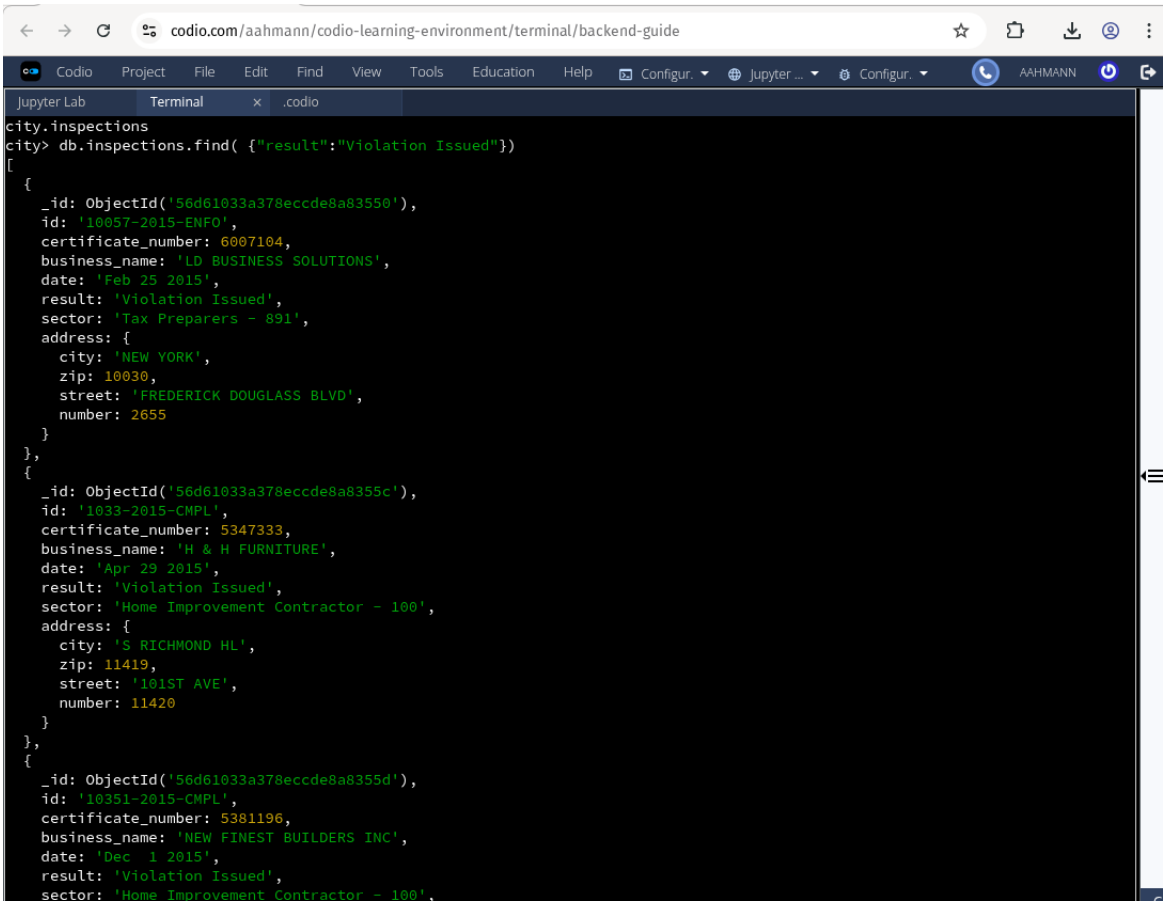

Figure 12: Confirming the `updateMany` transaction.

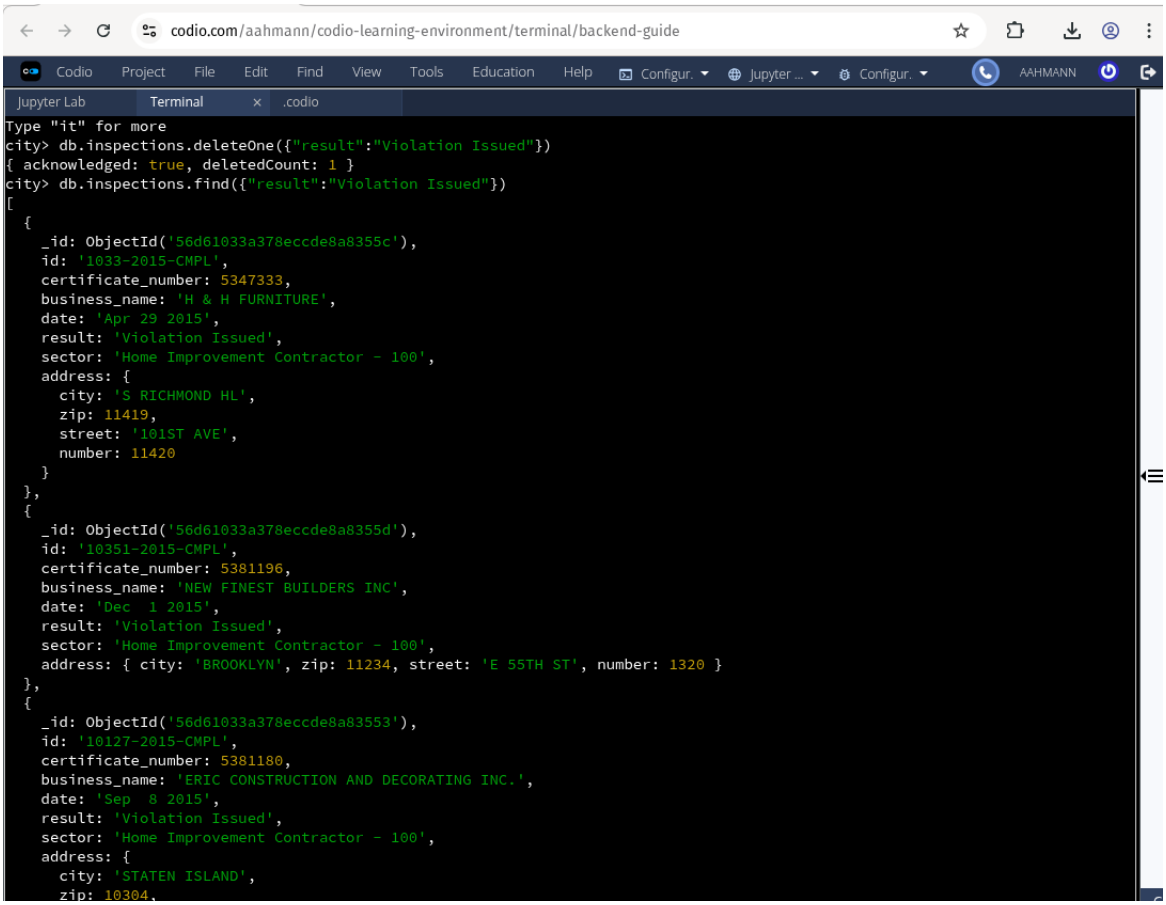

Figure 13: Results of removing a single, specific document from the `inspections` collection.