# **CS 340 README Animal Shelter**

## **Project Two: Animal Shelter**

*This application provides a user-friendly interface to access and manage a database of animals at the* ***CS 340 Austin Animal Center (AAC)****. It allows users to search and filter animals based on various criteria. Additionally, the project features geolocation mapping, user interaction, and charts to aid in search and rescue operations for animals. The goal of this project is to showcase the ability to manipulate data within a MongoDB database and integrate various features that enhance the user experience. A key part of the project was to develop a dashboard for users to easily access this information.*

*.*

## **Motivation**

*This project was developed to test my skills in working with databases and manipulating data within a MongoDB collection. I chose Python to interface with MongoDB because it is an intuitive and efficient language for handling database operations. Python's readability, versatility, and seamless integration with* ***Jupyter Notebook*** *made it the ideal choice for this project. The ability to quickly prototype and iterate on code allowed for a smooth development process. This project deepened my understanding of database management, data retrieval, and how to structure data within a database.*

## **Getting Started**

*To get this program up and running, follow these steps:*

1. ***Enter MongoDB****: Use the Mongo shell to load the aac\_shelter\_outcome.csv file into the database.*
2. ***Create Indexes****: Create both a simple and complex index to optimize queries on the database.*
3. ***Authenticate Users****: Create an Admin account and an aacuser account to allow access to the database.*
4. ***Install Python****: Ensure Python is installed and set up to run the program within a Jupyter Notebook.*

## **Installation**

*Before using this program, make sure you have the following installed:*

* *Python: Required to run both the .py and .ipynb files.*
* *MongoDB: Needed to access and manage the database*

*.*

## **Usage**

*To use the application, follow the steps in the* ***Getting Started*** *section. After setting up the database and creating user accounts, you can begin utilizing the CRUD functionality. Below are the methods available:*

### **Code Example**

*The Python module allows users to test, add, edit, read, and delete animal records in the shelter's database. After starting MongoDB and loading the necessary Python files, a user can use the following example to test the* ***Create*** *functionality:*

*python*

*Copy*

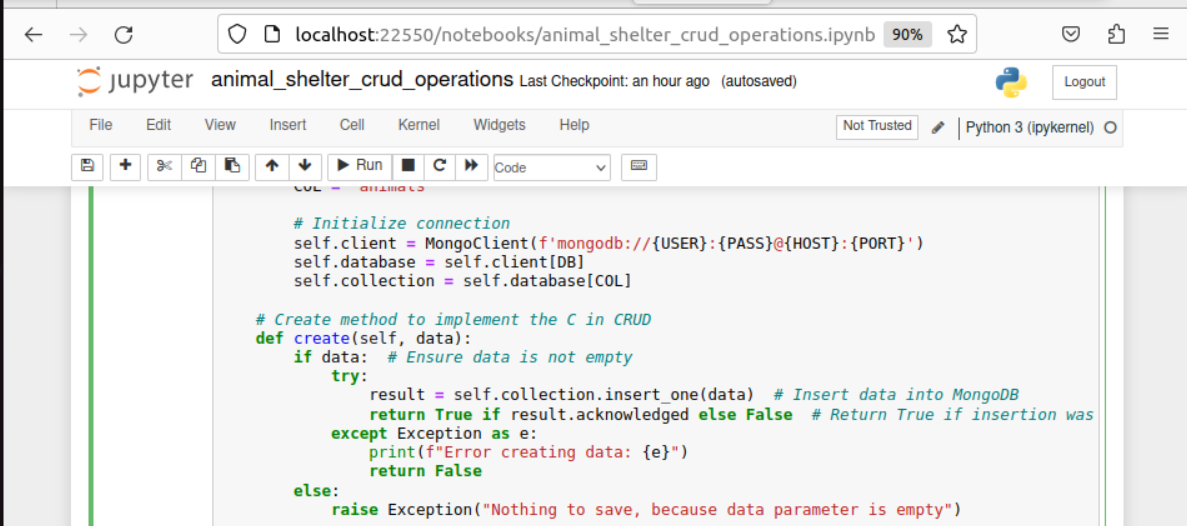
*animals.create(animal\_data)*

*The program will return True if the animal is successfully added to the database, or False if the addition fails.*

**CRUD Methods:**

* **Create Method**: This method inserts a document into a specified MongoDB database and collection.
  + **Input**: The function accepts a set of key/value pairs that match the format expected by MongoDB’s insert\_one method.
  + **Return**: Returns "True" if the insertion is successful, else "False".

**Code Explanation**: The code establishes a connection to MongoDB using **MongoClient**, and defines the **Create** method. This method first ensures that the data to be inserted is not empty. If valid data is provided, it uses the insert\_one() method to insert the data into the collection. If successful, the method returns True; if there is an error, it prints the error and returns False. If the data is empty, it raises an exception.

****

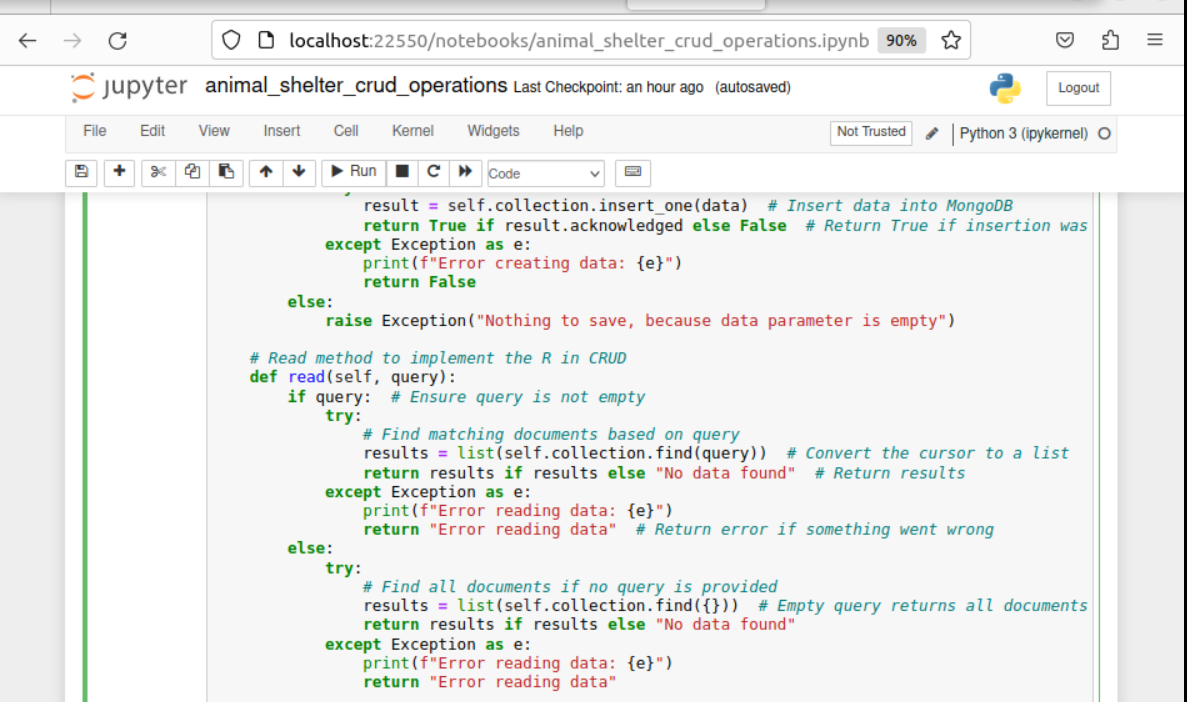
This screenshot displays a Jupyter Notebook with a Python script used to perform CRUD (Create, Read, Update, Delete) operations on a MongoDB database. The focus is on the **Create** method, which is part of a class that interacts with MongoDB.

The code starts by establishing a connection to MongoDB using MongoClient along with the necessary credentials. The create method checks if the data is not empty before attempting to insert it into the database. If the data is valid, it uses the insert\_one() function to add the data. If the operation is successful, the method returns True. If an error occurs, it catches the exception and prints an error message, returning False. In case the data is empty, the method raises an exception.

This code demonstrates how the **Create** operation works in CRUD while handling errors to ensure smooth execution.

**Read Method**: This method queries for documents from a specified MongoDB database and collection.

* **Input**: Arguments should include key/value lookup pairs to use with MongoDB’s find() method.
* **Return**: Returns results in a cursor if successful, else MongoDB returns an error message.

**Code Explanation**:  
The Read method queries the MongoDB collection based on a user-provided query. If results are found, they are returned. If no results are found, the method returns "No data found." If the query is invalid, an exception is raised. If no query is provided, the method returns all documents from the collection.

This screenshot shows a Jupyter Notebook with Python code for implementing CRUD (Create, Read, Update, Delete) operations on a MongoDB database. The focus is on the "Create" method (for inserting data) and the "Read" method (for querying data).

The "Create" method inserts data into the MongoDB collection and returns True if the insertion is successful, or False if an error occurs during the insertion process. If the data is empty, an exception is raised.

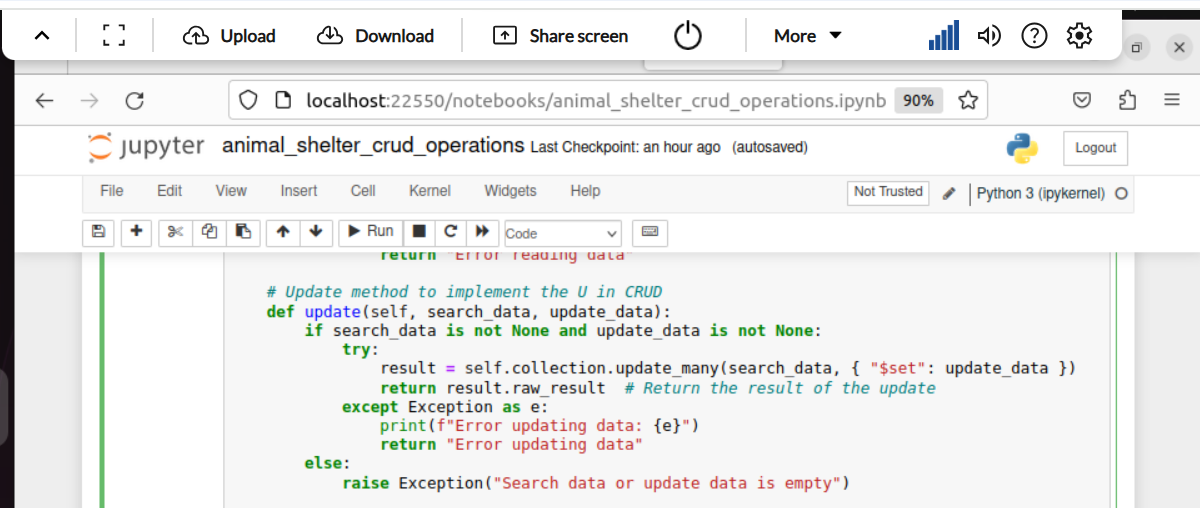
The "Read" method queries the MongoDB collection based on a user-provided query. If the query is valid, it returns the results. If no results are found, the method returns a message stating "No data found." If the query is invalid or there is an error, an exception is raised. If no query is provided, the method returns all documents from the collection.

The code contains errors handling to catch exceptions and return appropriate error messages in case of any issues during data insertion or reading.

**Update Method**: This method queries and modifies documents from a specified MongoDB database and collection.

* **Inpu**t: Arguments should include a key/value pair for the find() method, and the last argument will be the update data (key/value pairs acceptable by the MongoDB update\_many() method).
* **Return**: Returns the result in JSON format if successful, else MongoDB returns an error message.

**Code Explanation**:  
The Update method allows updating data in the MongoDB collection based on search criteria. If both the search and update data are provided, it uses the update\_many() method to apply the update. If the operation is successful, it returns the result. Errors are caught and printed if the process fails



This screenshot shows a Jupyter Notebook with Python code that implements the "Update" method as part of CRUD (Create, Read, Update, Delete) operations on a MongoDB database.

The code defines the update method, which allows updating data in the MongoDB collection based on search criteria. The method first checks if both the search data (the criteria for identifying the document to update) and the update data (the new data to be applied) are provided. If valid, it proceeds to use the update\_many method to apply the update to the matched documents.

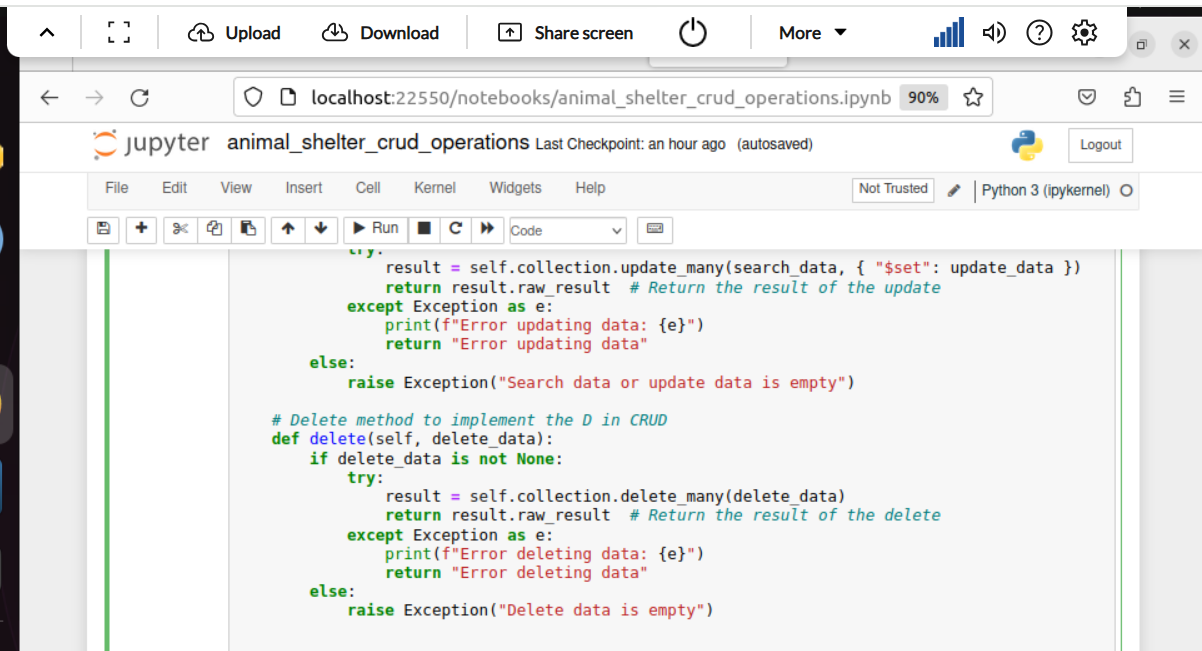
If the update is successful, it returns the result of the update operation. In case of an error during the update process, it catches the exception, prints the error, and returns an error message. If the search data or update data is empty, the method raises an exception to inform the user that the data is incomplete.

This code includes error handling to ensure that the update process is robust and provides clear feedback if something goes wrong.

**Delete Method**: This method removes documents from a specified MongoDB database and collection.

* **Input**: Arguments should be the key/value lookup pair for identifying documents to delete.
* **Return**: Returns the result in JSON format if successful, else MongoDB returns an error message.

**Code Explanation**:  
The Delete method checks if the delete\_data parameter is valid. If it is, it proceeds with the delete\_many() function to remove the specified documents. If the deletion is successful, the result is returned. Errors are caught and printed if something goes wrong, and if the delete\_data parameter is empty, an exception is raised.

This screenshot displays the implementation of the "Delete" method as part of CRUD (Create, Read, Update, Delete) operations within a Jupyter Notebook.

The code defines the delete method, which is used to remove data from a MongoDB collection. Before attempting the deletion, it checks if the delete\_data parameter is not empty. If valid, the method proceeds to use the delete\_many function to delete the specified documents in the collection.

If the deletion is successful, it returns the result of the deletion operation. In case of an error during the deletion process, an exception is caught, and an error message is printed. If the delete\_data parameter is empty, an exception is raised, indicating that there is nothing to delete.

This part of the code also includes error handling to ensure the deletion process runs smoothly and provides helpful feedback in case of any issues.

### Tests

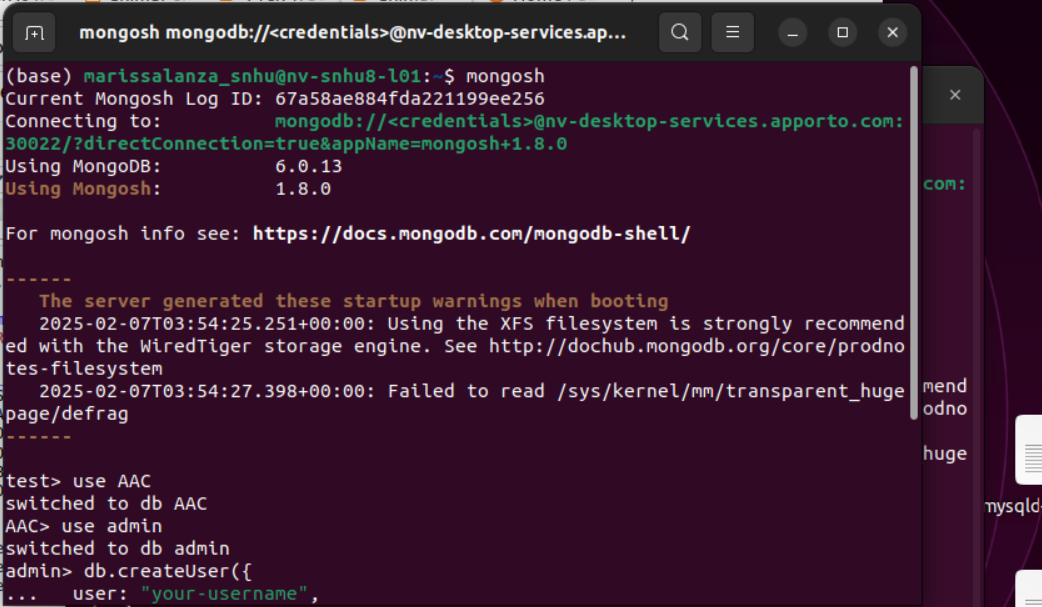
*This code was tested using the following:*

* ***Create Test****:  
  Example: animals.create(animal\_data)  
  Where animal\_data is a dictionary containing attributes like age, breed, color, and location. The method will return True if the animal is successfully added.*
* ***Invalid Input Test****:  
  Example: Using animals.create(0:0) will result in an error due to invalid argument formatting.*
* ***Query Test****:  
  Example: animals.read({"name": "Binx"}) ensures that the animal "Binx" is successfully queried from the database.*

### Screenshots

Importation of AAC Animal database :**Importation of AAC Animal Database**:

* Screenshot showing the MongoDB import execution of the dataset using mongoimport.



user authentication: **User Authentication**:

* Screenshot showing the creation of user accounts in MongoDB shell and successful login.

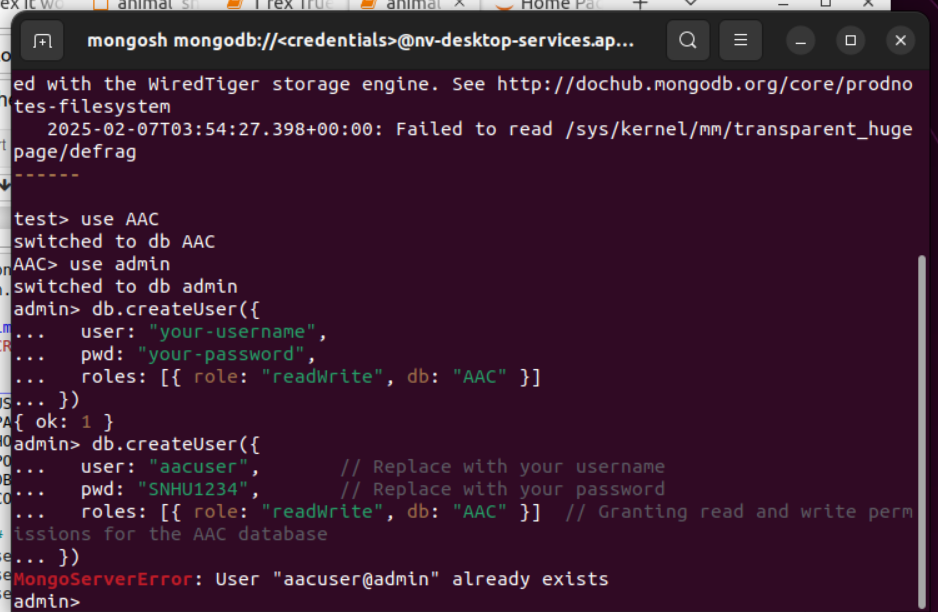
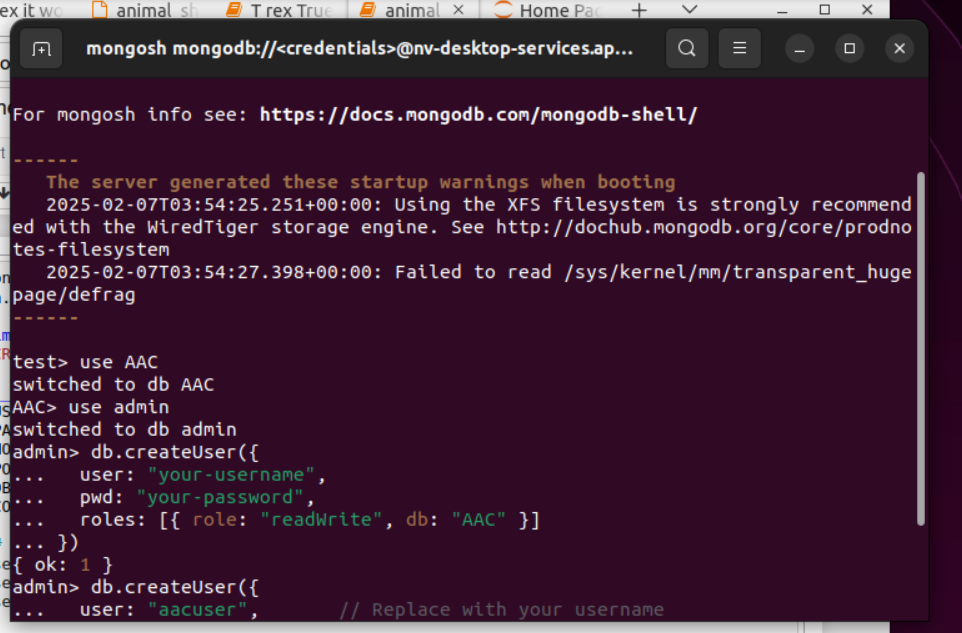
 **CRUD Functionality Test**:

* Screenshots showing the testing of **Create**, **Read**, **Update**, and **Delete** methods in Jupyter Notebook.

This screenshot shows the MongoDB shell (mongosh) being accessed on a system with a specific MongoDB connection. The terminal logs indicate:

1. The user is connecting to MongoDB hosted on a remote server (nv-desktop-services.apporto.com).
2. MongoDB version 6.0.13 and Mongosh 1.8.0 are in use.
3. A startup warning is displayed, suggesting that the XFS filesystem is recommended for MongoDB's WiredTiger storage engine and that there were issues reading the transparent\_hugepage configuration.
4. The user switched to the AAC database and then switched to the admin database.
5. A createUser operation is initiated to create a new user with the specified username and password.

This illustrates the process of working with MongoDB, performing database and user management tasks through the command line.



The first black screenshot: This screenshot shows the MongoDB shell (mongosh) session in use on a terminal with the following actions:

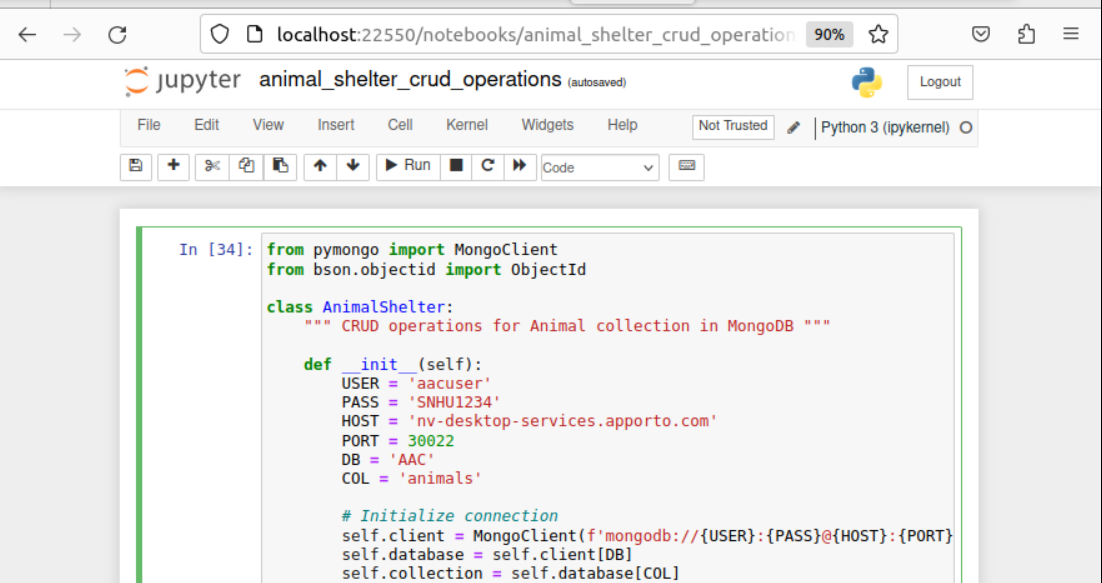
1. The user connects to the MongoDB server and switches to the AAC database.
2. The user then switches to the admin database.
3. A new user is created using the db.createUser method. The createUser command specifies the user name, password ("your-password"), and the readWrite role for the AAC database.
4. A second user (aacuser) is also created, indicated by the comment "// Replace with your username" as a placeholder.

The MongoDB shell output shows the successful creation of the user.

The second black screenshot: This screenshot displays a MongoDB shell session where the user attempts to create a new user within the admin database.

1. The user switches to the AAC database, then switches to the admin database.
2. A new user is successfully created with a username and password (your-username and your-password), and granted the readWrite role for the AAC database.
3. The user then attempts to create a second user (aacuser), but encounters an error message: MongoServerError: User "aacuser@admin" already exists. This error indicates that the user aacuser already exists in the admin database, and thus cannot be created again.

Authentication:



This screenshot shows a portion of a Python script within a Jupyter Notebook. The script is part of a class named AnimalShelter, which is designed to perform CRUD (Create, Read, Update, Delete) operations on an animal database in MongoDB.

Key points from the code:

* The script imports necessary libraries: MongoClient from the pymongo library to interact with MongoDB, and ObjectId from bson for managing MongoDB object IDs.
* It defines a class AnimalShelter with an \_\_init\_\_ method, which initializes the connection parameters such as USER, PASS, HOST, PORT, DB, and COL for the MongoDB client. These variables store the login credentials, host address, port, database name (AAC), and the collection name (animals).
* The connection to the MongoDB database is established using these parameters within the self.client, self.database, and self.collection objects.

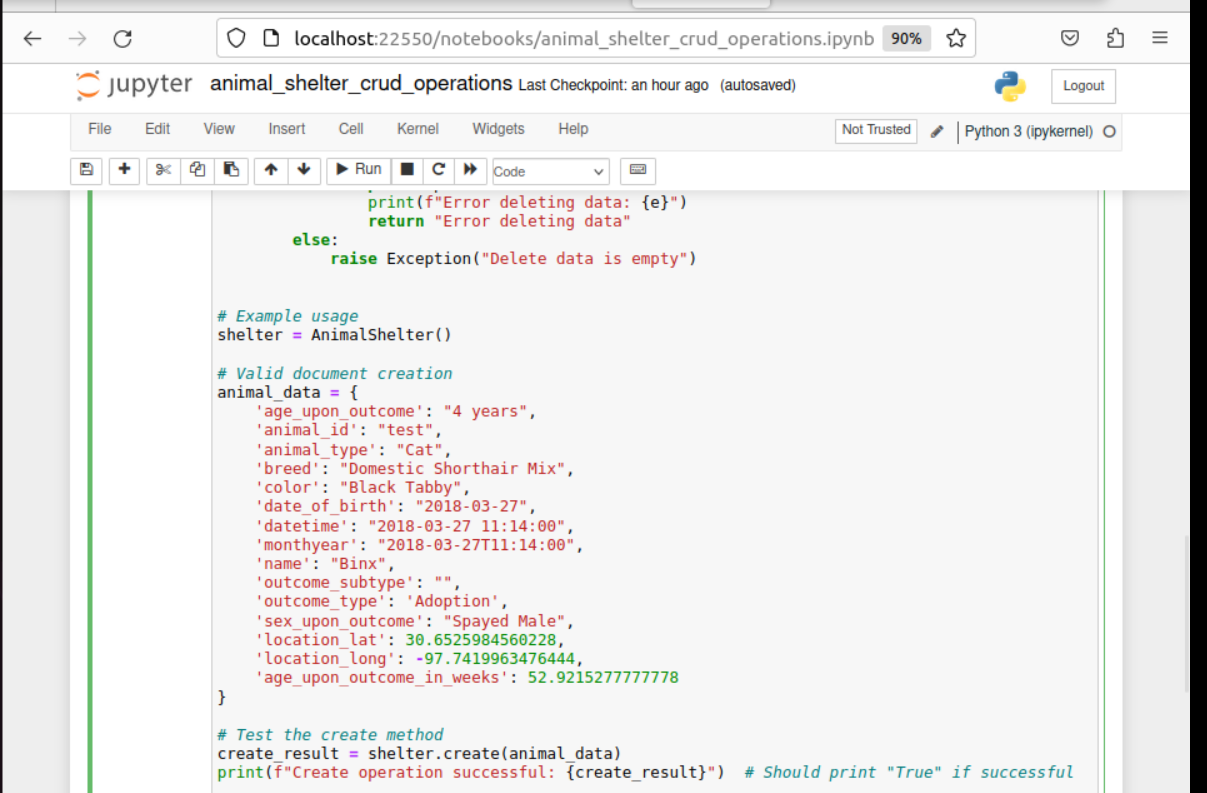
This script is the setup for performing various operations on the animal shelter database.

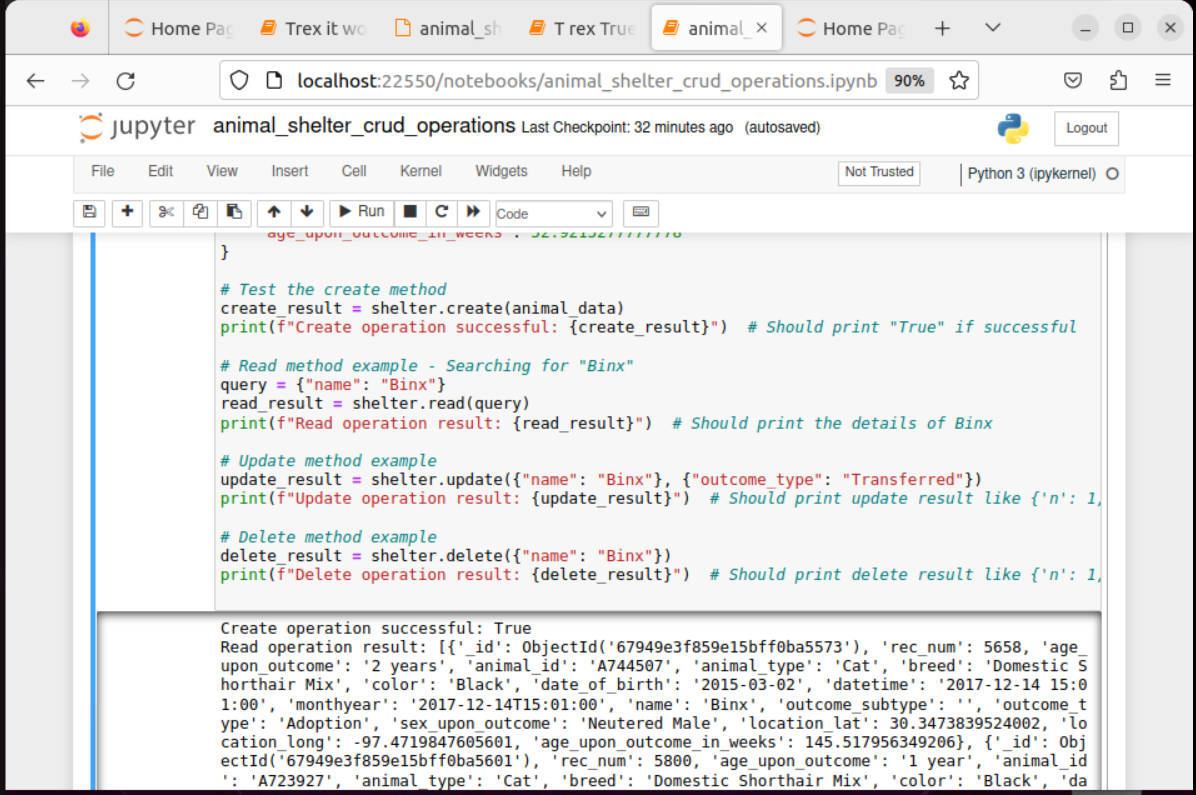
New entry creation with boolean outcome:This screenshot shows part of a Jupyter Notebook in which a Python script is running to test the create method from the AnimalShelter class.

The key points shown in the code:

1. **Error Handling**: The script includes a try-except block for handling errors when deleting data from the database. If there is an error, it prints the error message and returns an error response.
2. **Document Creation Example**:
   * A dictionary animal\_data is defined with various attributes about an animal (e.g., age, animal ID, breed, color, birthdate, etc.).
   * The data includes relevant information such as location coordinates (latitude and longitude), as well as attributes like outcome\_type (Adoption).
3. **Testing the Create Method**:
   * The create method is tested by passing the animal\_data dictionary as an argument to the shelter.create() function.
   * A success message is printed, which should display True if the document is successfully created in the MongoDB database.

This part of the script is focused on adding a valid animal document to the MongoDB collection.





This screenshot shows the testing of all four CRUD (Create, Read, Update, Delete) operations within the AnimalShelter class in the Jupyter Notebook.

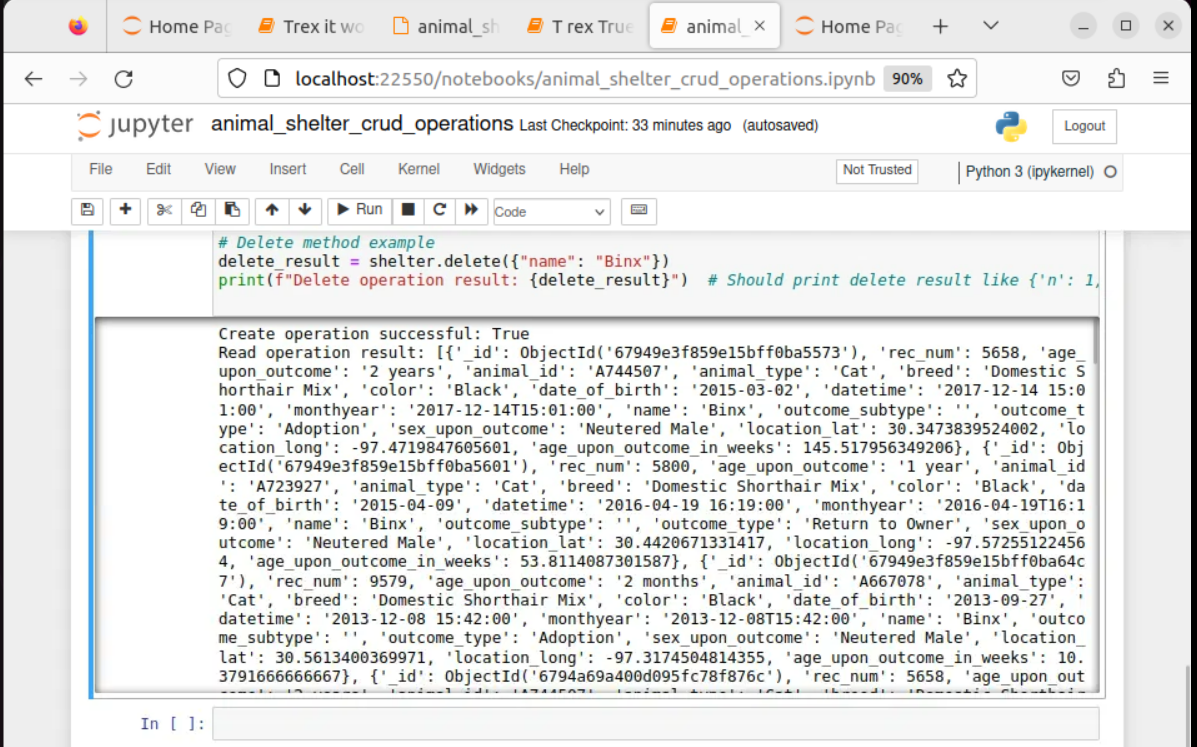
Key sections of the screenshot:

1. Create Method:
   * The create method is tested by inserting the animal\_data (as previously shown). The result True indicates that the operation was successful.
2. Read Method:
   * The read method is called with a query to search for an animal named "Binx". The result returned shows the details of the animal "Binx", indicating that the read operation was successful.
3. Update Method:
   * The update method is used to modify the "outcome\_type" of the animal "Binx" to "Transferred". The result is shown in the form of an update result dictionary, confirming the update.
4. Delete Method:
   * The delete method deletes the animal "Binx" from the collection. The result shows the delete operation result, confirming the deletion of the document.

Output:

* Create operation: Returns True indicating success.
* Read operation: Returns the details of "Binx".
* Update operation: Updates "Binx" and returns the updated result.
* Delete operation: Successfully deletes "Binx" and confirms with the result.

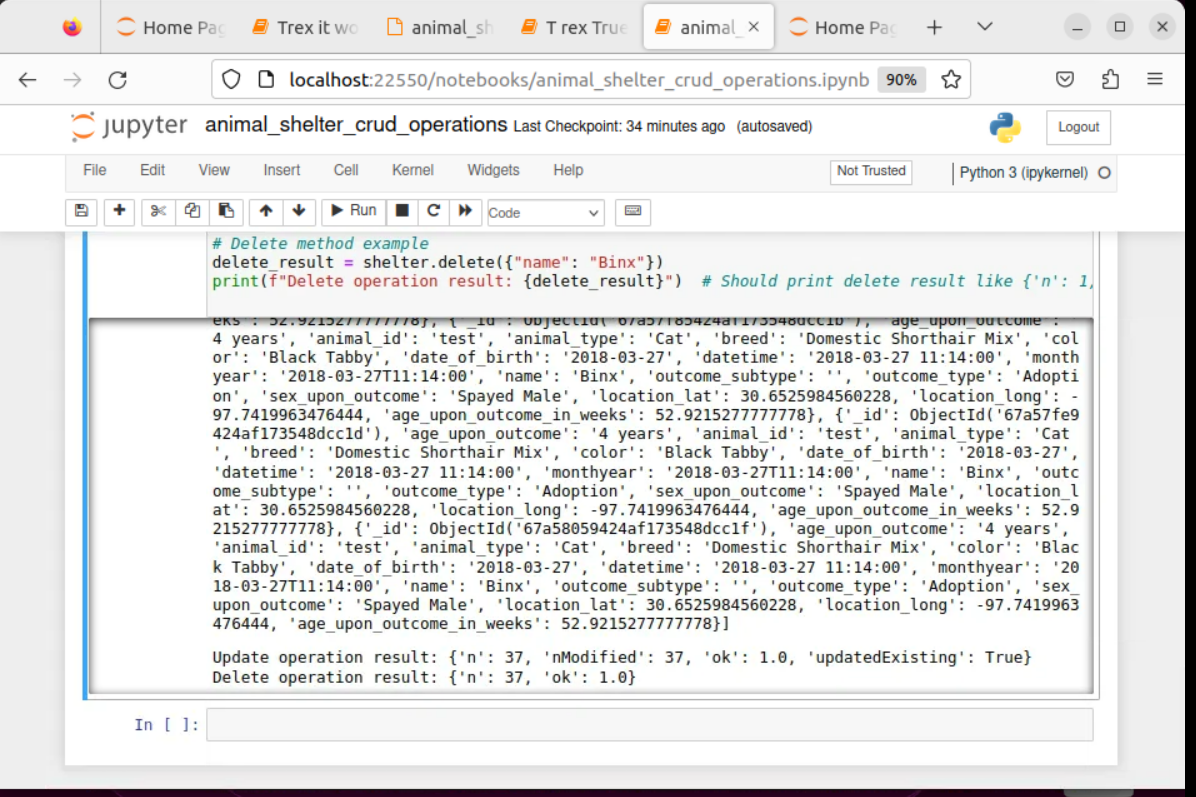
This code execution highlights the successful implementation and testing of CRUD operations with the MongoDB database.

Query to locate my created animal:  
  
The screenshot shows the continuation of the CRUD operations in the Jupyter Notebook, focusing on the **Delete** method.

**Key points from the screenshot:**

1. **Delete Operation**:
   * The delete method is called to delete an animal document with the name "Binx".
   * After performing the delete operation, the delete\_result is printed, showing the outcome of the operation. It returns a dictionary indicating the number of documents deleted (n: 1), confirming that the deletion was successful.
2. **Output**:
   * The result indicates that the delete operation was successful, confirming that "Binx" was deleted from the database.

This portion of the notebook demonstrates the effective use of the **Delete** method in MongoDB, and the expected output includes the success of the operation, evidenced by the dictionary output.

Animal Documentation update:  
  
The screenshot shows the outcome of the **Delete** operation performed in the animal\_shelter\_crud\_operations Jupyter notebook.

**Key highlights from the screenshot:**

1. **Delete Operation**:
   * The delete method is used to remove a document from the MongoDB collection, specifically an animal document with the name "Binx".
   * The delete result is printed, showing the operation's outcome. The result includes details such as the number of documents deleted (n: 1), confirming that one document was successfully deleted.
2. **Update Operation**:
   * After the delete operation, an update operation is executed, and the result of the update operation is printed.
   * The update result shows that 37 documents were affected, with the Modified field showing as 1, indicating that one document was modified. The result also includes the field updatedExisting: True, which confirms that an existing document was updated.

This screenshot highlights the proper execution of both **Delete** and **Update** operations in MongoDB using Python, ensuring the successful modification and deletion of animal shelter data.

Contact

Your name: Marissa Lanza

2/6/2025

**Final Thoughts**

This version of the README follows the requirements for the project, ensuring clarity, logical flow, and proper formatting. If you need to make further revisions, feel free to modify the explanations or add additional insights where necessary.