CS 340 README Template

About the Project

This project is comprised of custom software utilizing Python and MongoDB to perform create, read, update, and delete (CRUD) operations on large-scale datasets. It serves to provide a user-friendly experience to our customers through easy database queries, efficient processing, and rapid results. It specializes in helping customers identify dogs in shelters that would make ideal candidates for search and rescue operations. The databases holding information about dogs in shelters are large and have the capability to rapidly grow. Therefore, it was important to create a project that would allow workers to identify capable dogs much more quickly and efficiently.

Steps to Project Completion

This project, while still able to be improved significantly, has been in development for close to two months. It was important to break this project down into manageable portions and develop incrementally. The first major step was getting all the required packages, plugins, and database pieces ready. In doing so, we heavily researched the functionality provided in each tool to better understand the methods that would best solve problems and create efficient solutions. The next step was to create the CRUD functionality in Python through Jupyter Notebook as it would serve as the backbone for our program. This development was done incrementally with a heavy amount of testing to ensure no problems would arise in the future and all potential outcomes had been planned for. All testing was done through Jupyter Notebook In doing so, error handling was critical as the software expands and new inputs may be utilized which could introduce unexpected side effects. After the CRUD module was complete, we began work on the Dash framework using Jupyter Notebook which was critical as it was the point of contact for users. Here is where Plotly, Pandas, and the Python CRUD module will be

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

imported. It started with creating a data table to house all relevant data from the database and conducting thorough tests to ensure it had been implemented completely. Then it was critical to add sorting features so users could identify relevant lines of data much quicker. From there, a histogram and geolocation chart were developed to add supplemental detail for the user after they had sorted the dataset to their needs. Again, testing in Jupyter Notebook and through trial-and-error was a heavy component of this section to ensure no bugs had been introduced and the outputs matched expectations.

Motivation for Using MongoDB, Python, and Dash

MongoDB is a NoSQL database and provides many benefits necessary to this project. It is flexible with schemas which is valuable with data being received from multiple shelters which may introduce data inconsistencies. MongoDB was designed to be multi-threaded for dynamic environments with the ability to process multiple operations simultaneously which bolster the efficiencies of the query operations. Furthermore, it is a strong tool for querying and analytics while providing horizontal scaling. It also provides a seamless connection with multiple programming languages such as Python. Therefore, it was paired with Python for its compatibility, ease of use, strong data manipulation nature, and flexibility. Python also does not require the types of variables to be predetermined. This is important as CRUD operations will require multiple data types to be sent to various methods which can be handled easily by Python. Dash provided the web framework which would serve as the point of interaction for the customer and the user. It is reliable and responsive to update data outputs to the user based on targeted inputs through callback functions. Dash provides a simple and intuitive programming experience to develop dynamic apps, widgets, and buttons for sorting that significantly improve the user experience. Finally, Dash utilizes Python which ensures compatibility within our MVC design pattern.

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

These factors make this custom software's use intuitive with rapid results to greatly bolster the identification of rescue dogs.

Getting Started

To get a local copy up and running, it is first recommended to have basic MongoDB knowledge prior to working with the software. The first major step is getting an account setup with the appropriate privileges. For this software, users will need read and write privileges to the associated database. Next, the collection, comprised of the information to be manipulated, will need to be imported into MongoDB. Finally, users, with appropriate privileges, can use the accompanying Python code to login and perform CRUD operations on the dataset. It is important to note, users will have to identify the host and port connections and update them in the Python code accordingly. With the create function, users can add documents to the database easily through key-value pairs. The read function allows users to view documents within the collection via a specified key-value pair. This pair can be as restrictive or open as necessary to identify the relevant documents. The update function allows users to target specific documents and update any relevant information necessary. The delete function allows users to identify relevant documents and subsequently delete them from the collection.

Installation

The tools necessary to use the software are MongoDB, Python, Jupyter Notebook, Plotly, Dash, and Pandas. It will also be required to possess an associated user account and access to the dataset in MongoDB. A user account with proper permissions will require basic MongoDB knowledge or a support staff to create it. Finally, the dataset in question will be dependent on user needs and availability. More information regarding the individual tools is described below:

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

MongoDB: Utilize either the Community or Enterprise editions based on the needs of the software. This comprises the NoSQL database for the program. More information on downloading MongoDB can be found at: https://www.mongodb.com/docs/manual/installation/.

Python: This is the required programming language for the software. It is compatible with MongoDB and Dash. Any Python IDE, such as PyCharm can be utilized to modify the code. More information on downloading Python can be found at: https://www.python.org/downloads/.

Jupyter Notebook: Can be installed from the command line. It can be utilized to run the CRUD operations, unit tests, and scripts to initialize the Dash web framework. More information on downloading Jupyter Notebook can be found at: https://jupyter.org/install.

Plotly: Is required to import into the project script to generate charts such as the histogram and geolocation charts. This tool is primarily utilized for generating graphs and charts in Python and can be imported into the code with Jupyter Notebook. More information on utilizing and importing Plotly can be found at: https://plotly.com/python/getting-started/.

Dash: This is the web framework that will be utilized to build the web application and user side of the program. It allows the use of dynamic widgets and apps in the web application and can be directly imported into the code with Jupyter Notebook. More information on utilizing and importing Dash can be found at: https://dash.plotly.com/installation.

Pandas: This tool is also utilized in the web framework to create and populate data frames in the data table. It enhances and increases the efficiency of the data structures used throughout the program. It can be imported directly into the code with Jupyter Notebook. More information on utilizing Pandas can be found at: https://pandas.pydata.org/docs/getting_started/overview.html.

Usage

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

This application will mostly be interacted with through the Dash framework but is comprised of additional features. First, the CRUD module currently has four main functions. The create method allows users to input new documents into the database based on user input. The read method is heavily relied upon and searches the database for documents relevant to user input based on key-value pairs. The update function lets users update any value in any document held within the database. Finally, the delete function can be used to delete any documents from the database that are no longer needed for whatever reason. On the Dash web application, the user will be presented with sorting options, a data table, a histogram, and a geolocation chart. The sorting options use radio buttons to search the database for relevant dog breeds based on customer requirements. Once a radio button is selected, it will dynamically update the data table to only show relevant search returns. It defaults to “RESET” which returns the data table to an unfiltered state. The histogram and geolocation chart also dynamically update to sorting changes. The histogram displays the quantity and breed of dogs based on the selected sorting option. The geolocation chart will have one marker on the map determined by the coordinates in each document. It initializes to the first document in the data table but will update when the user selects a new row. When clicked on, it displays a selection of data from the row selected.

Below are examples of sorting operations on the Dash web application.

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

Reset Filter – Initial State

GRAZIOSO SALVARE

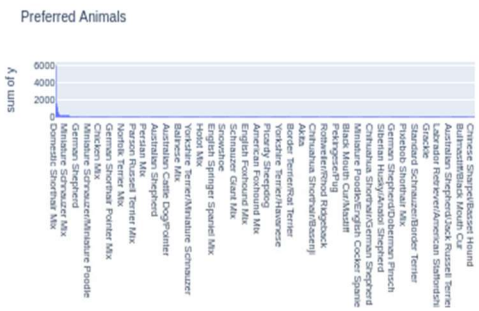
Christian Henshaw Global Rain Dashboard

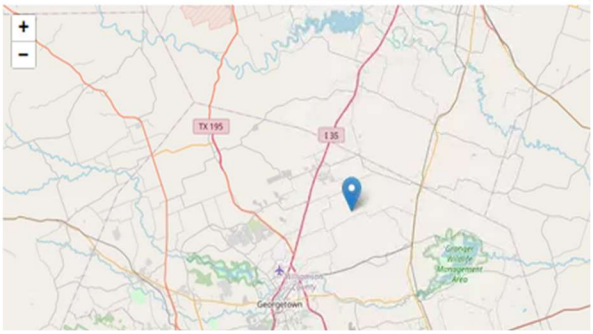
CS-340 Dashboard

○Water Rescue ○Mountain or Wilderness Rescue ○Disaster or Individual Tracking Rescue ◉Reset Filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome
	filter												
◉	3	2 years	A716330	Dog	Chihuahua Shorthair Mix	Brown/White	2013-11-18	2015-12-28 18:43:00	2015-12-28T18:43:00	Frank		Adoption	Neutered Male
○	1	3 years	A746874	Cat	Domestic Shorthair Mix	Black/White	2014-04-10	2017-04-11 09:00:00	2017-04-11T09:00:00		SCRP	Transfer	Neutered Male
○	9	3 years	A720214	Dog	Labrador Retriever Mix	Red/White	2013-02-04	2016-02-11 12:41:00	2016-02-11T12:41:00	Blessing		Adoption	Spayed Female
○	10	3 months	A664290	Cat	Domestic Shorthair Mix	Tortie	2013-09-01	2013-12-08 14:58:00	2013-12-08T14:58:00	*Taylor		Adoption	Spayed Female
○	11	1 year	A721199	Dog	Dachshund Wirehair Mix	Tan/White	2015-02-23	2016-02-27 17:49:00	2016-02-27T17:49:00	Belle		Adoption	Spayed Female

« ‹ 1 / 4002 › »





Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

Water Rescue:

GRAZIOSO
SALVARE

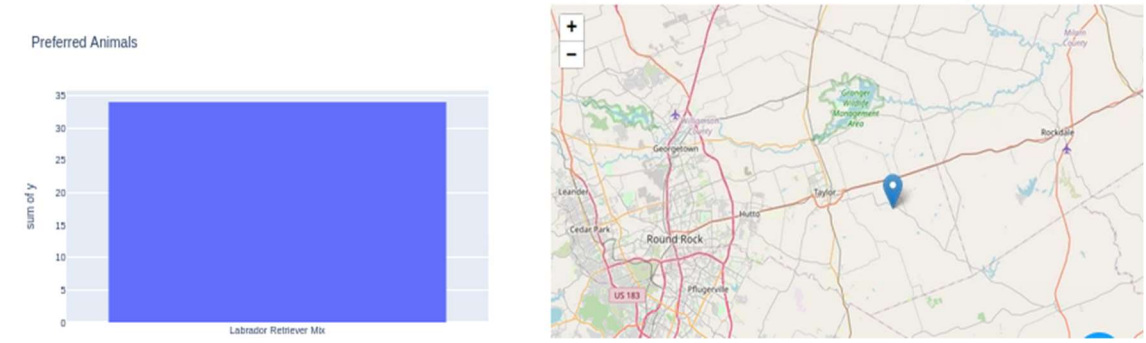
Christian Henshaw Global Rain Dashboard

CS-340 Dashboard

◉ Water Rescue ○ Mountain or Wilderness Rescue ○ Disaster or Individual Tracking Rescue ○ Reset Filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcom
	filter												
◉	36	6 months	A706953	Dog	Labrador Retriever Mix	Yellow	2014-12-06	2015-07-06 11:33:00	2015-07-06T11:33:00		Medical	Euthanasia	Intact Femal
○	732	2 years	A749782	Dog	Labrador Retriever Mix	Tan/White	2015-05-19	2017-07-25 14:59:00	2017-07-25T14:59:00	*Catalina		Return to Owner	Intact Femal
○	1121	1 year	A757158	Dog	Labrador Retriever Mix	White/Black	2016-08-30	2017-08-31 14:12:00	2017-08-31T14:12:00	Pirata		Return to Owner	Intact Femal
○	1628	9 months	A740471	Dog	Labrador Retriever Mix	Tan/White	2016-03-17	2016-12-23 17:13:00	2016-12-23T17:13:00	Mika		Adoption	Intact Femal
○	1757	7 months	A742767	Dog	Labrador Retriever Mix	Black	2016-06-27	2017-02-14 15:20:00	2017-02-14T15:20:00	Marley		Return to Owner	Intact Femal

« ‹ 1 / 7 › »



Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

Mountain or Wilderness Rescue:

GRAZIOSO
SALVARE

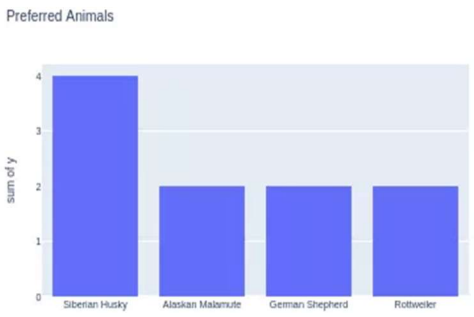
Christian Henshaw Global Rain Dashboard

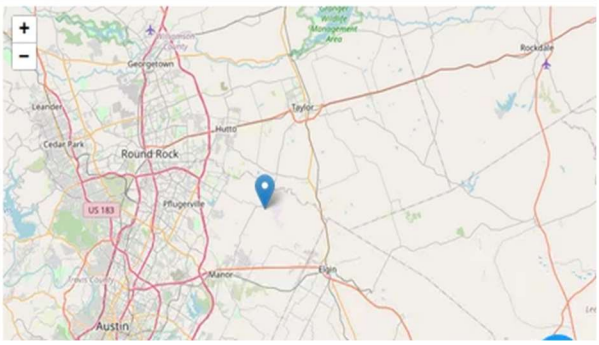
CS-340 Dashboard

○Water Rescue ◉Mountain or Wilderness Rescue ○Disaster or Individual Tracking Rescue ○Reset Filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome	l
	filter													
◉	5315	2 years	A708726	Dog	Alaskan Malamute	Sable/White	2013-07-30	2015-08-02 17:24:00	2015-08-02T17:24:00	Papa		Return to Owner	Intact Male	30.4
○	5315	2 years	A708726	Dog	Alaskan Malamute	Sable/White	2013-07-30	2015-08-02 17:24:00	2015-08-02T17:24:00	Papa		Return to Owner	Intact Male	30.4
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.
○	6021	2 years	A728165	Dog	Rottweiler	Black	2015-05-31	2017-09-23 11:23:00	2017-09-23T11:23:00	Zeke		Return to Owner	Intact Male	30.4

« ‹ 1 / 2 › »





Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

Disaster or Individual Tracking Rescue:

GRAZIOSO SALVARE

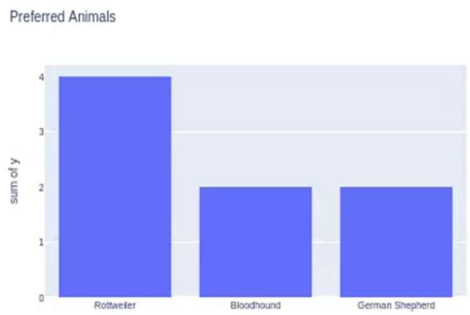
Christian Henshaw Global Rain Dashboard

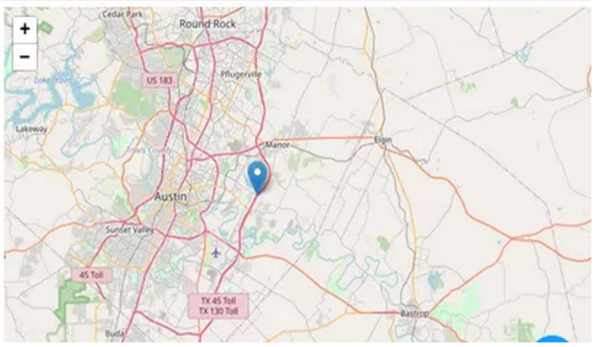
CS-340 Dashboard

○Water Rescue ○Mountain or Wilderness Rescue ◉Disaster or Individual Tracking Rescue ○Reset Filter

	rec_num	age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome	lo
	filter													
◉	3767	4 years	A712291	Dog	Bloodhound	Red	2011-09-20	2015-09-22 15:43:00	2015-09-22T15:43:00	Boomer		Return to Owner	Intact Male	30.270
○	3767	4 years	A712291	Dog	Bloodhound	Red	2011-09-20	2015-09-22 15:43:00	2015-09-22T15:43:00	Boomer		Return to Owner	Intact Male	30.270
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.4
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.4
○	2987	4 years	A694614	Dog	Rottweiler	Black/Brown	2011-01-01	2015-01-01 14:25:00	2015-01-01T14:25:00	Striker		Return to Owner	Intact Male	30.32

« ‹ 1 / 2 › »

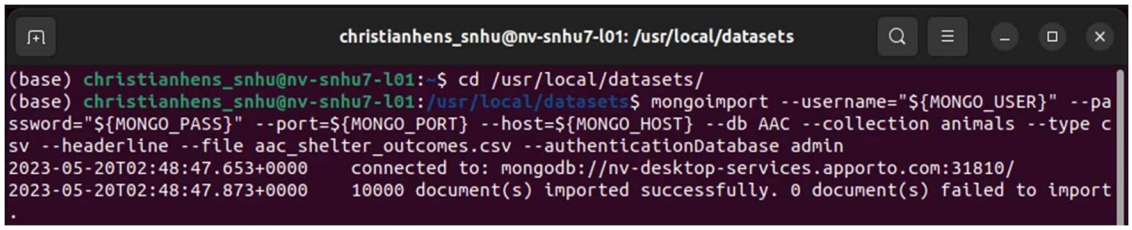




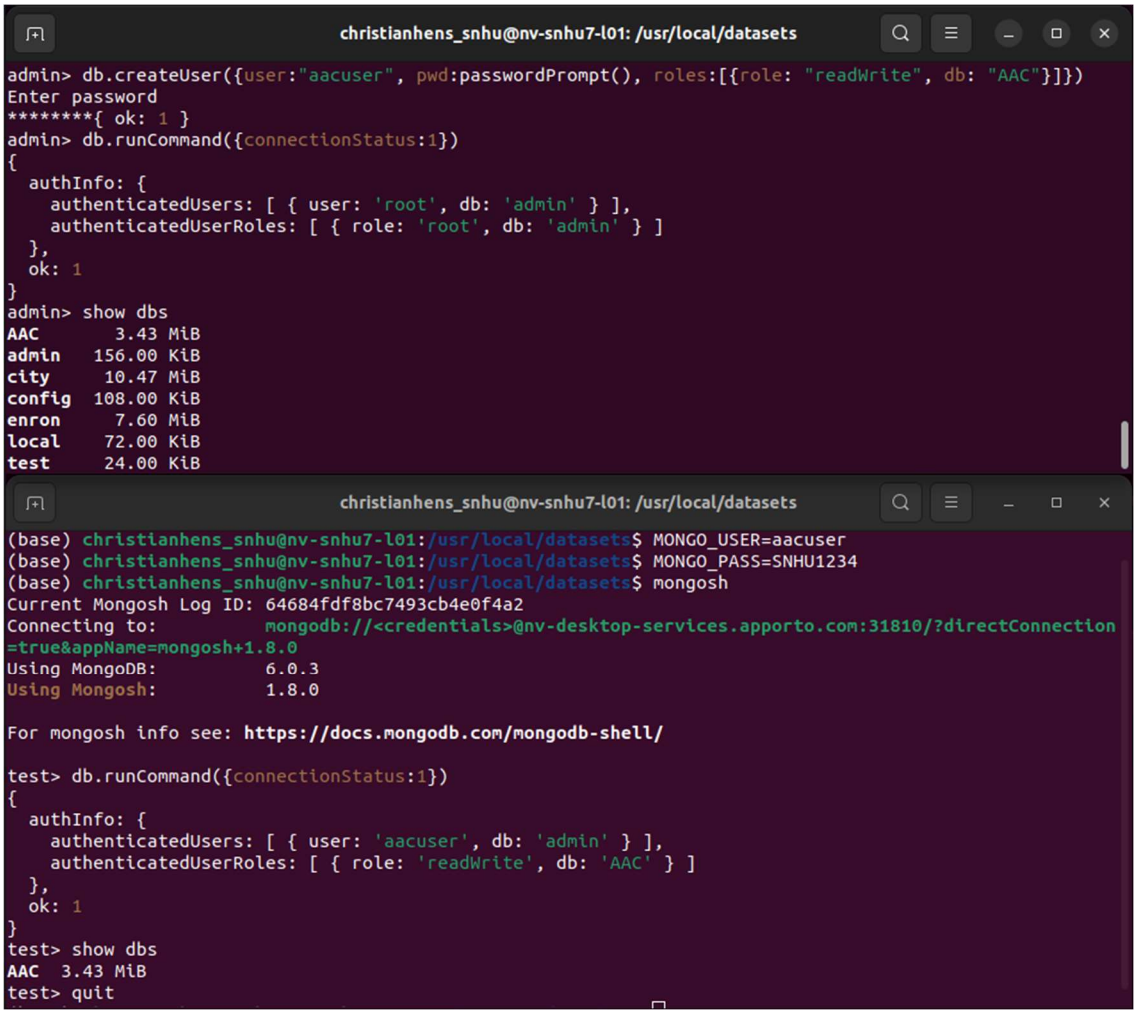
Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

Below are examples of how the project works and how it can be used.

Code Example



The example above demonstrates importing the dataset, in this case the 'animals' collection, into MongoDB. Note it is important to have proper user account authorization to import a collection.



Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

The example above demonstrates creating a new user account with the appropriate privileges to utilize this software. This user account has read and write privileges to the associated database to perform CRUD operations on.

CRUD Tests

```
print('Attempting Valid Animal Creation:')
create_Test = CRUD.create({"name": "xyz", "animal_type": "Dog", "outcome_type": "Transfer"})
print(create_Test)
```

```
Attempting Valid Animal Creation:
True
```

```
print('Attempting Invalid Animal Creation:')
create_Test = CRUD.create({0:0})
print(create_Test)
```

```
Attempting Invalid Animal Creation:
False
```

The example above highlights the functionality of the software tests for valid and invalid create operations on the dataset utilizing Jupyter notebook. In the first scenario, the input is valid so a document can be created accordingly. In the second scenario, the input is invalid which catches the exception block. The Boolean return value describes whether the process was completed or ran into issues.

```
print('Attempting Valid Animal Read:')
read_Test = CRUD.read({"name": "xyz"})
for iter in read_Test:
    print(iter)
```

```
Attempting Valid Animal Read:
{'name': 'xyz', 'outcome_type': 'Adopted'}
{'name': 'xyz', 'animal_type': 'Dog', 'outcome_type': 'Adopted'}
{'name': 'xyz', 'animal_type': 'Dog', 'outcome_type': 'Transfer'}
```

```
print('Attempting Invalid Animal Read:')
read_Test = CRUD.read({"name": "abcdefg"})
for iter in read_Test:
    print(iter)
```

```
Attempting Invalid Animal Read:
```

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

The example above highlights the functionality of the software tests for valid and invalid read operations on the dataset. In the first scenario, the input is valid and relevant results are found which are displayed to the user. This information will describe all pertinent information about potential rescue dogs located in the shelters. The second scenario demonstrates invalid input information where no hits were identified in the search. Therefore, the user will be presented with an empty list. Should the key-value pair be invalid, an exception will be thrown accordingly.

```
print('Attempting Valid Animal Update:')
update_Test = CRUD.update({"name": "xyz"}, {"outcome_type": "Adopted"})
print("Documents updated: ", update_Test.modified_count)
```

```
Attempting Valid Animal Update:
Documents updated:  1
```

```
print('Attempting Invalid Animal Update:')
update_Test = CRUD.update({"name": "abcdefg"}, {"outcome_type": "Adopted"})
print("Documents updated: ", update_Test.modified_count)
```

```
Attempting Invalid Animal Update:
Documents updated:  0
```

The example above highlights the functionality of the software tests for valid and invalid update operations on the dataset. The first scenario demonstrates valid input that has an already present document in the collection. Dot notation is used on the MongoDB cursor object to identify the number of documents that were updated. The second scenario demonstrates invalid input without an associated document in the collection. Subsequently, the dot notation reveals that no documents were updated. Should the key-value pairs be invalid, an exception will be thrown accordingly.

```
print('Attempting Valid Animal Delete:')
delete_Test = CRUD.delete({"name": "xyz"})
print("Documents deleted: ", delete_Test.deleted_count)
```

```
Attempting Valid Animal Delete:
Documents deleted:  3
```

```
print('Attempting Invalid Animal Delete:')
delete_Test = CRUD.delete({"name": "abcdefg"})
print("Documents deleted: ", delete_Test.deleted_count)
```

```
Attempting Invalid Animal Delete:
Documents deleted:  0
```

The example above highlights the functionality of the software tests for valid and invalid delete operations on the dataset. In the first scenario, the input is valid with relevant documents contained within the collection. Therefore, they are identified, and the number of documents deleted is elicited with dot notation from the MongoDB cursor object. The second scenario demonstrates invalid input that has no relevant documents to identify in the collection. Subsequently, dot notation reveals no documents had been deleted. Similarly, if the key-value pairs are invalid, an exception will be thrown.

CRUD Screenshots

```
    # Create method to implement the C in CRUD.
    def create(self, data):
        try:
            if data is not None:
                # Insert user passed data into the collection
                insert = self.database.animals.insert_one(data)  # data should be dictionary
                if insert is not None:
                    # return true indicating procedure finished properly
                    return True
            else:
                raise Exception("Nothing to save, because data parameter is empty")
        except:
            # return false indicating procedure failed
            return False

    # Read method to implement the R in CRUD.
    def read(self, readData):
        if readData is not None:
            # Find all matching data suppressing the id variable
            result = list(self.database.animals.find(readData, {"_id": False}))
            if result is not None:
                # return list of matching data
                return result
            else:
                # return an empty list
                return []
        else:
            raise Exception("Nothing to search, because data parameter is empty")
```

The example above highlights the functionality of the create and read functions within the software. In creating a document, users can pass key-value pairs which will be inserted into the collection. An associated Boolean is returned informing the user of successful or unsuccessful creation. In reading a document, a user can pass as restrictive as necessary key-values pairs which will return relevant documents from within the collection. Relevant options will be returned if found, otherwise, a blank list will be produced indicating no hits were identified.

```
    # Update method to implement the U in CRUD.
    def update(self, dataToUpdate, newData):
        if dataToUpdate or newData is not None:
            # Find and update all matching data
            updated = self.database.animals.update_many(dataToUpdate, {"$set": newData})
            # return cursor object containing updated count
            return updated
        else:
            raise Exception("Nothing to update, because data parameter is empty")

    # Delete method to implement the D in CRUD.
    def delete(self, dataToDelete):
        if dataToDelete is not None:
            # Find and delete all matching data
            deleted = self.database.animals.delete_many(dataToDelete)
            # return cursor object containing deleted count
            return deleted
        else:
            raise Exception("Nothing to delete, because data parameter is empty")
```

The example above highlights the functionality of the update and delete functions within the software. In updating a document, users can pass two separate key-value pairs to the appropriate method. The first key-value pair will be utilized to identify matching documents that will be updated. The second key-value pair will contain the information that will be updated in the previously identified documents. The MongoDB cursor object will be returned which contains the modified count that can be elicited through dot notation as demonstrated later. In deleting a document, users can pass key-value pairs which will identify relevant documents to be deleted. Similarly, to updating documents, the MongoDB cursor object will be returned which contains the deleted count that can be elicited through dot notation as demonstrated later.

Roadmap/Features

Multiple features have been proposed for future releases to include:

- Adding functionality to allow users to select multiple rows at once and have multiple markers displayed on the geolocation map.
- Limit the number of x-values displayed in the initial histogram to prevent compression.

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.

- Add custom icons on the geolocation chart markers depending on which breed of animal was selected.
- Add more information regarding what will be queried for based on the sorting option.
- Fix the duplication of rows during sorting.

Challenges During Development:

There were many challenges faced during development. It was initially difficult to create the histogram and geolocation charts as the syntax was unfamiliar. It was necessary to set aside a fair amount of time for testing as many errors were thrown out and had to be fixed accordingly. The geolocation chart also required a significant amount of time to troubleshoot due to an error. It stated an index was out of bounds for one of the coordinates to place a marker on the map during the initial load. However, it would still place the marker properly and was able to identify other values perfectly fine in the same manner which would then be displayed. It was not exactly clear why this error was occurring. One line of code suppressed the built-in ID column from being displayed in the data table which was removed. While it did not seem relevant, it fixed this issue. Another major challenge faced was rows duplicating after sorting the dataset and it isn't entirely clear why this occurs. When selecting a sorting option, besides "Reset Filter," each row will be populated in the table twice yet contain the same data. Subsequently, the histogram values are doubled which doesn't change the percentages but may impact user experience. This issue is still in troubleshooting but will remain in the software until a fix can be implemented.

Contact

Christian Henshaw

Note: This template has been adapted from the following sample templates: Make a README, Best README Template, and A Beginners Guide to Writing a Kickass README.