Project Instructions

Used Cars Dataset

Objective:

In this final project, you will combine the skills we have developed over the duration of this course to incorporate and analyze two real datasets.

Deliverables:

To receive credit for this lab, please submit the following files:

- `createCarInsertScripts.py`
- `createCityInsertScript.py`
- `A_locationCars.js`
- `B_locationCities.js`
- `C_carClosestCity.js`
- `D_cityFarthestCar.js`
- `E_mostDistantCars.js`

Instructions:

For this project, we will be using two public datasets:

- Kaggle: Used Cars Dataset: https://www.kaggle.com/austinreese/craigslist-carstrucks-data
- Kaggle: California Cities Dataset: https://www.kaggle.com/camnugent/california-housing-feature-engineering

To ensure that the data doesn't change from the time I create this project to when you begin it, I have uploaded to Canvas the datasets as they are at the time of writing. Download the following files from Canvas to get started:

- `vehicles.zip`
- `cal_cities_lat_long.csv`

Extract the `vehicles.csv` file by right clicking on the zip file and selecting Extract All…. This file contains 480,154 rows. You can see this with the following command in the PowerShell:

```
gc vehicles.csv | Measure-Object -Line
```

However, as you will find as you work with the data, there are actually only 458,213 rows of actual CSV data. Some of the data contains newline characters that throw off the line count.

Each row of data in this CSV file details a used car ad on Craigslist. You need to insert this data into a MongoDB collection called `carData` in a database called `FinalProject`. Write a Python file called `createCarInsertScripts.py` to convert the data into MongoDB insert commands (`insertOne` or `insertMany`) stored in one or more JavaScript files.

MongoDB has a maximum of 100,000 documents in an `insertMany` command, so you will at least need to break it up into 5 chunks.[1] You will probably find that an `insertMany` command with 80,000 documents runs too slowly, so you will probably need to have more `insertMany` commands with less documents. You could try 458,000 `insertOne` commands. You are the database manager; it is your choice. Both your python code will need to finish creating insertion scripts and your insertion scripts will need to finish inserting all 458,213 documents within 15 minutes each. One thing that may help improve performance is to disable the insert acknowledgements using the `writeConcern` feature of `insertMany`.

Since this dataset is so large (1.4GB), it's a good idea to prune out as much unnecessary detail as possible before importing it into MongoDB. You should exclude the following columns from your import:

- The first column (no heading), it is a row number we don't need
- `url`
- `region_url`
- `image_url`
- `description`

All of the data in the CSV file is stored as a string. Make sure to store the following as number datatypes in MongoDB (use the Python `float()` function):

- `price`
- `year`
- `odometer`
- `lat`
- `long`

And be sure to store the `posting_date` as a date in MongoDB.

When opening the CSV file in Python, you will need to specify the encoding as utf-8, as there are some non ASCII characters in the data.

It may be a good idea to include in your insertion code some print statements so that you can track the progress. If you write the code inefficiently, it can take hours to import the data. If you write it efficiently, you can get it down to about 5 minutes.

Once the vehicles data has been imported into the `carData` collection, your data should look like:

[1] https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/

```
_id: ObjectId("603ae7c20b1edfb7682e1da1")
id: "7240372487"
region: "auburn"
price: 35990
year: 2010
manufacturer: "chevrolet"
model: "corvette grand sport"
condition: "good"
cylinders: "8 cylinders"
fuel: "gas"
odometer: 32742
title_status: "clean"
transmission: "other"
VIN: "1G1YU3DW1A5106980"
drive: "rwd"
size: ""
type: "other"
paint_color: ""
state: "al"
lat: 32.59
long: -85.48
posting_date: 2020-12-02T14:11:30.000+00:00
```


Next, write a Python file called `createCityInsertScript.py` to import the `cal_cities_lat_long.csv` data into a MongoDB collection called `cityData` in the `FinalProject` database. Store the `Latitude` and `Longitude` values as numbers. Once the city data has been imported into the `cityData` collection, your data should look like:


```
_id: ObjectId("603aea055ba97e64883d68e9")
Name: "Adelanto"
Latitude: 34.582769
Longitude: -117.409214
```

Now that the vehicle and city data has been imported into MongoDB, you need to write JavaScript code to perform the following operations on the data:

1. Create a JavaScript file called `A_locationCars.js`. The purpose of this code is to create a location based index on `carData`. To create a location based index, you will need to:
    a. Remove the cars with null or blank locations (7448 documents) and put them in a separate collection called `carDataNoLocation`.

    b. Store the `carData` latitude and longitude values in the appropriate GeoJSON format for a 2dsphere Point. For example, the `loc` sub-document in:

```
{
    "name" : "New York City",
    "loc" : {
        "type" : "Point",
        "coordinates" : [50, 2]
    }
}
```

    The order in the array is: `[longitude, latitude]`.

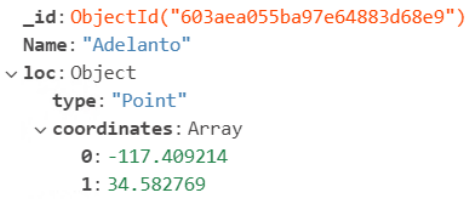
    c. Remove the `lat` and `long` keys as well, so that the only location data is in the `loc` sub-document. When you are done, your `carData` collection should look like:

```
_id: ObjectId("603ae7c20b1edfb7682e1da1")
id: "7240372487"
region: "auburn"
price: 35990
year: 2010
manufacturer: "chevrolet"
model: "corvette grand sport"
condition: "good"
cylinders: "8 cylinders"
fuel: "gas"
odometer: 32742
title_status: "clean"
transmission: "other"
VIN: "1G1YU3DW1A5106980"
drive: "rwd"
size: ""
type: "other"
paint_color: ""
state: "al"
posting_date: 2020-12-02T14:11:30.000+00:00
loc: Object
   type: "Point"
   coordinates: Array
      0: -85.48
      1: 32.59
```

    d. Then you can finally create the location-based index.

2. Create a JavaScript file called `B_locationCities.js`. This code should update each city entry so that the `Latitude` and `Longitude` values are converted into the appropriate GeoJSON format for a 2dsphere Point.

   Remove the `Latitude` and `Longitude` keys as well, so that the only location data is in the `loc` sub-document. Create an index on the cities by their location. When you are done, the data in `cityData` should look like:



3. Create a JavaScript file called `C_carClosestCity.js`. This code should find the closest California city for each car listed in CA and then store that city `name` and `loc` in the car document with the key `nearestCity`. Every car listed in CA will have a `nearestCity` key, and those in other states will not. When you are done, a car in CA in `carData` should look like:


```
_id: ObjectId("603ae7d10b1edfb7682e741e")
id: "7240678054"
region: "bakersfield"
price: 74995
year: 2019
manufacturer: "ford"
model: "f-250sd"
condition: ""
cylinders: ""
fuel: "diesel"
odometer: 13388
title_status: "clean"
transmission: "automatic"
VIN: "1FT7W2BT5KEE29401"
drive: "4wd"
size: ""
type: ""
paint_color: ""
state: "ca"
posting_date: 2020-12-02T22:01:31.000+00:00
loc: Object
    type: "Point"
    coordinates: Array
        0: -117.446303
        1: 34.070311
nearestCity: Object
    name: "Fontana"
    loc: Object
        type: "Point"
        coordinates: Array
            0: -117.435047
            1: 34.092233
```

4. Create a JavaScript file called `D_cityFarthestCar.js`. This will be used to calculate cities with the most distant cars. From the previous code, each car in CA has a nearest city, but the distance from the city to the actual car varies. Each city will have a most distant car that still considers that city its `nearestCity`. It is difficult to measure the straight line distance between two points on a sphere. So we will measure the difference between their latitude and longitude, which will be the angle between them on the Earth's surface.

   The `loc` coordinates are stored as

   ```
   [longitude, latitude]
   ```

   The vertical (north/south) distance would be

   ```
   latitude(car) - latitude(city)
   ```

   If this number is positive, the car is to the north, if it is negative, the car is to the south of the city.

   Your code should find and print out the 5 cities with the most distant northern, southern, eastern and western cars (total of 20 cities). Your output should look like:

```
---Most northern cars---
Calexico has a car 79.76 degrees N
Imperial Beach has a car 29.17 degrees N
Blythe has a car 20.58 degrees N
Needles has a car 1.32 degrees N
Avalon has a car 1.25 degrees N

---Most sourthern cars---
Tulelake has a car -30.66 degrees S
Crescent City has a car -21.02 degrees S
Needles has a car -10.04 degrees S
Alturas has a car -6.66 degrees S
Dorris has a car -6.47 degrees S

---Most eastern cars---
Imperial Beach has a car 2.87 degrees E
Dorris has a car 1.16 degrees E
Susanville has a car 0.82 degrees E
Redding has a car 0.77 degrees E
Bishop has a car 0.74 degrees E

---Most western cars---
Crescent City has a car -275.1 degrees W
Calexico has a car -87.84 degrees W
Blythe has a car -70.72 degrees W
Needles has a car -42.86 degrees W
Tulelake has a car -29.5 degrees W
```

5. Create a JavaScript file called `E_mostDistantCars.js`. There are a couple of very distant cars discovered in the previous script. This code should find the coordinates of the v1st most northern car and the three most western cars and their corresponding cities. Your output should look like:

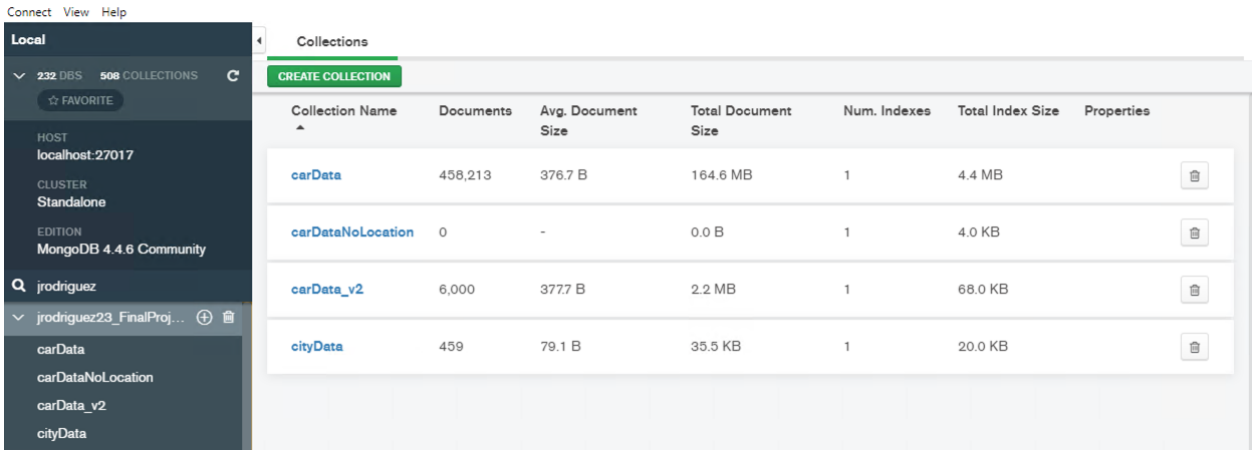
```
---Most northern cars---
Calexico is at -115.498883,32.678947 and has a car at -27.662376, -47.082977

---Most western cars---
Crescent City is at -124.201747,41.755947 and has a car at 150.898969, 62.773733
Calexico is at -115.498883,32.678947 and has a car at -27.662376, -47.082977
Blythe is at -114.589175,33.617233 and has a car at -43.870361, 13.034829
```

Some of the operations in these JavaScript files apply permanent changes to the data. You should write these scripts so that if they are run one after the other (`A` to `E`), the data in the database is in the form described. For example, at the end of `A_locationCars.js`, there are two indexes on the `carData` (`_id` and `loc`) and the `carDataNoLocation` collection holds all the null and blank location cars.

Grading:

The final project will be graded using the rubric provided below:

Task	Description	Evaluation Score: Missing = 0; Inadequate = .25; Average = .5; Proficient = .75; Excellent = 1	Score
createCarInsertScripts.py	Creates JS files that import car data		2.5%
	Data inserted into correct collection/database		1.25%
	Documents split up amongst multiple files		2.5%
	Unnecessary fields excluded		2.5%
	Data stored as appropriate datatypes		2.5%
	Script files created within 15 minutes		5%
	Scripts insert data within 15 minutes		5%
createCityInsertScript.py	Creates JS file that imports city data		2.5%
	Data stored as appropriate datatypes		2.5%
	Data inserted into correct collection/database		1.25%
A_locationCars.js	a) Null or blank location cars moved to other collection		5%
	b) Location data stored appropriately as GeoJSON		5%
	c) Lat and long keys removed		2.5%
	d) Location index created correctly		2.5%
B_locationCities.js	a) Location data stored appropriately as GeoJSON		2.5%
	b) Latitude and Longitude keys removed		2.5%
	c) Location index created correctly		2.5%
C_carClosestCity.js	nearestCity found for every car in CA		15%
D_cityFarthestCar.js	Most distant cars found in all directions		25%
E_mostDistantCars.js	Coordinates of cars and cities found		10%
Grade			100%


Connect View Help
Local
232 DBS 508 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 4.4.6 Community
jrodriguez
jrodriguez23_FinalProj...
carData
carDataNoLocation
carData_v2
cityData
Collections
CREATE COLLECTION
Collection Name
Documents
Avg. Document Size
Total Document Size
Num. Indexes
Total Index Size
Properties
carData 458,213 376.7 B 164.6 MB 1 4.4 MB
carDataNoLocation 0 - 0.0 B 1 4.0 KB
carData_v2 6,000 377.7 B 2.2 MB 1 68.0 KB
cityData 459 79.1 B 35.5 KB 1 20.0 KB