PS3

November 7, 2022

CS 61 Lab 3: MongoDB, Javascript

Student: Amittai Siavava

Date: 2022-11-07

NOTE

- I used Jupyter Notebook to write this lab then converted it to pdf, so:
  - The code blocks contain the code that I ran to produce the Markdown cells (or, in pdf, the text, results, or explanations) that are right after.

1 Part 1 (30 Points)

Use the zipcodes dataset.

1.1 1. What is the primary schema represented in zipcodes.json?

```
{
  "_id"   : 5 character numeric string,
  "city"  : string,
  "loc"   : [ longitude: float, latitude: float ],
  "pop"   : integer,
  "state" : 2 character string
}
```

1.2 2. Get the zipcodes.json file loaded into a db on your machine (local) or Atlas (cloud).

[ ]:
```
# Import a bunch of stuff we need.
from pymongo import MongoClient, InsertOne, DeleteOne, ReplaceOne, UpdateOne
import json
from IPython.display import display, Markdown, Latex

display(Markdown('##### Connecting to MongoDB'))

# Connect to the database.
client = MongoClient('localhost', 27017)
```

```
db = client['test']
collection = db['test']

# Clear collection if it has any data.
collection.delete_many({})

# Insert each line in file into the collection.
record_count = 0
display(Markdown('##### Inserting data into MongoDB'))
with open("zipcodes.json") as f:
  for line in f:
    if line.strip(): # ignore blank lines
      record_count += 1
      record = json.loads(line)
      collection.insert_one(record)

# Query the collection
display(Markdown(f"""\n
##### Report on insertions.

  | Metric | Value |
  | :--- | ---: |
  | Records read from file | {record_count} |
  | Records in database    | {collection.count_documents({})} |

##### Sample record:
  ```json
    {collection.find_one({})}
  ```
"""))
```

Connecting to MongoDB

Inserting data into MongoDB

Report on insertions.

Metric	Value
Records read from file	29353
Records in database	29353

Sample record:

{'_id': '01001', 'city': 'AGAWAM', 'loc': [-72.622739, 42.070206], 'pop': 15338, 'state': 'M

1.3 3. Count the number of zip codes in the collection.

```
[ ]: display(Markdown(f"""
#### Strategy

Since zipcodes are the `_id` field, we can use `count_documents`
to get the number of total documents in the collection.

The `_id` field is a unique index, so it will match the number of documents.

#### Full Command
```js
  collection.count_documents({{}})
```

Results:

Total number of zip codes in the collection: {collection.count_documents({})}
"""))
```

Strategy Since zipcodes are the `_id` field, we can use `count_documents` to get the number of total documents in the collection.

The `_id` field is a unique index, so it will match the number of documents.

Full Command

```
collection.count_documents({})
```

Results: Total number of zip codes in the collection: 29353

1.4 4. Count the total number of zip codes in the New England states (CT, RI, MA, VT, NH, and ME).

```
[ ]: display(Markdown(f"""
#### Strategy

We can use the `$in` operator to match any of the values in the list
`["CT", "RI", "MA", "VT", "NH", "ME"]`.

We can then use `count_documents` to get the number of documents that match.

#### Full Command
```js
  collection.count_documents({{"state": {{"$in": ["CT", "RI", "MA", "VT", "NH",␣
 ↪"ME"]}}}})
```
```

```
```

Results:

Total number of zip codes in `CT, RI, MA, VT, NH, ME`: \
{collection.count_documents({"state": {"$in": ["CT", "RI", "MA", "VT", "NH",␣
↪"ME"]}})}
"""))

Strategy We can use the `$in` operator to match any of the values in the list `["CT", "RI", "MA", "VT", "NH", "ME"]`.

We can then use `count_documents` to get the number of documents that match.

Full Command

```
collection.count_documents({"state": {"$in": ["CT", "RI", "MA", "VT", "NH", "ME"]}})
```

Results: Total number of zip codes in `CT, RI, MA, VT, NH, ME`: 1677

1.5 5. Determine the total population of Rhode Island.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use the `collection.aggregate` function, with:

1. `$match` to get documents in `RI`
2. `$group` to group the matched documents.
3. `$sum` to sum the `pop` field while grouping.

#### Full Command:
```js
  collection.aggregate([
    {{"$match": {{"state": "RI"}}}},
    {{"$group": {{"_id": "RI", "totalPop": {{"$sum": "$pop"}}}}}}]
  ).next()["totalPop"]
```

Results:

The total population in RI is: \
{collection.aggregate([{"$match": {"state": "RI"}}, {"$group": {"_id": "RI",␣
↪"totalPop": {"$sum": "$pop"}}}]).next()["totalPop"]}
"""))
```

Strategy We can use the `collection.aggregate` function, with:

1. `$match` to get documents in `RI`
2. `$group` to group the matched documents.
3. `$sum` to sum the `pop` field while grouping.

Full Command:

```
collection.aggregate([
  {"$match": {"state": "RI"}},
  {"$group": {"_id": "RI", "totalPop": {"$sum": "$pop"}}}]
).next()["totalPop"]
```

Results: The total population in RI is: 1003218

1.6 6. Determine which zip code has the smallest population.

[ ]:
```
display(Markdown(f"""

#### Strategy

We can use the `collection.aggregate` function, with:

1. `$sort` with `"pop": 1` to sort by population in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

#### Full Command:
```js
  collection.aggregate([{{"$sort": {{"pop": 1}}}}])
            .next()["_id"]
```


Results:

The zip code with the smallest population is: \
{collection.aggregate([{ "$sort": { "pop": 1}}]).next()["_id"] }

"""))
```

Strategy We can use the `collection.aggregate` function, with:

1. `$sort` with `"pop": 1` to sort by population in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

Full Command:

```
collection.aggregate([{"$sort": {"pop": 1}}])
          .next()["_id"]
```

Results: The zip code with the smallest population is: 02163

1.7 7. Determine the southernmost zip code in the database.

[ ]:
```
display(Markdown(f"""
#### Strategy

The southernmost zip code will have the smallest longitude.
The longitude is the second index in the `loc` array.
We can use the `collection.aggregate` function, with:

1. `$sort` with `"loc.1": 1` to sort by longitude in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

#### Full command:
```js
  collection.aggregate([{"$sort": {{"loc.1": 1}}}}])
            .next()["_id"]
```

Results:

The southernmost zip code is: \
{collection.aggregate([{ "$sort": { "loc.1": 1}}]).next()["_id"] }

"""))
```

Strategy The southernmost zip code will have the smallest longitude. The longitude is the second index in the `loc` array. We can use the `collection.aggregate` function, with:

1. `$sort` with `"loc.1": 1` to sort by longitude in ascending order.
2. `.next()` to get the first document.
3. `["_id"]` to get the zip code (`_id`) field.

Full command:

```
collection.aggregate([{"$sort": {"loc.1": 1}}])
          .next()["_id"]
```

Results: The southernmost zip code is: 96772

1.8 8. Determine the average population of states beginning with the letter 'M'.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use the `collection.aggregate` function, with:

1. `$group` to group the documents by `state`, summing `pop`.
2. `$match` with `$regex: "^M"` to match states that start with `M`.
3. `$group` with `_id: null` to group all aggregations and average the total␣
 ↪populations.


#### Full Command:
```js
  collection.aggregate([
    {{"$group": {{"_id": "$state", "totalPop": {{"$sum": "$pop"}}}}}},
    {{"$match": {{"_id": {{"$regex": "^M"}}}}}},
    {{"$group": {{"_id": "null", "avgPop": {{"$avg": "$totalPop"}}}}}}
  ]).next()["avgPop"]
```

Results:

The average population of states beginning with `M` is: \
{collection.aggregate([
  { "$group": { "_id": "$state", "totalPop": { "$sum": "$pop"}}},
  { "$match": { "_id": { "$regex": "^M"}}},
  { "$group": { "_id": "null", "avgPop": { "$avg": "$totalPop"}}}
]).next()["avgPop"]}

"""))
```

Strategy We can use the `collection.aggregate` function, with:

1. `$group` to group the documents by `state`, summing `pop`.
2. `$match` with `$regex: "^M"` to match states that start with `M`.
3. `$group` with `_id: null` to group all aggregations and average the total populations.

Full Command:

```
  collection.aggregate([
    {"$group": {"_id": "$state", "totalPop": {"$sum": "$pop"}}},
    {"$match": {"_id": {"$regex": "^M"}}},
```

```
    {"$group": {"_id": "null", "avgPop": {"$avg": "$totalPop"}}}
  ]).next()["avgPop"]
```

Results: The average population of states beginning with `M` is: 4271942.875

1.9 9. Which zip codes have more than $50,000$ population?

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use `collection.find` with `"pop": $gt 50000` to get all documents
with a population greater than 50,000.

We can select only the `_id` field with `"_id": 1`.

#### Full Command:
```js
  collection.find({{"pop": {{"$gt": 50000}}}}, {{"_id": 1}})
```

Results:

The zip codes with a population greater than 50,000 are: \
{list(
  collection.find({"pop": {"$gt": 50000}}, {"_id": 1})
)}"""))
```

Strategy We can use `collection.find` with `"pop": $gt 50000` to get all documents with a population greater than 50,000.

We can select only the `_id` field with `"_id": 1`.

Full Command:

```
  collection.find({"pop": {"$gt": 50000}}, {"_id": 1})
```

Results: The zip codes with a population greater than 50,000 are: [{'_id': '01201'}, {'_id': '01701'}, {'_id': '02146'}, {'_id': '02148'}, {'_id': '02154'}, {'_id': '02155'}, {'_id': '02401'}, {'_id': '02895'}, {'_id': '06010'}, {'_id': '06040'}, {'_id': '06450'}, {'_id': '06511'}, {'_id': '06516'}, {'_id': '06902'}, {'_id': '07002'}, {'_id': '07047'}, {'_id': '07055'}, {'_id': '07087'}, {'_id': '07111'}, {'_id': '07305'}, {'_id': '07306'}, {'_id': '08360'}, {'_id': '08753'}, {'_id': '10002'}, {'_id': '10003'}, {'_id': '10009'}, {'_id': '10016'}, {'_id': '10021'}, {'_id': '10023'}, {'_id': '10024'}, {'_id': '10025'}, {'_id': '10027'}, {'_id': '10029'}, {'_id': '10031'}, {'_id': '10032'}, {'_id': '10033'}, {'_id': '10128'}, {'_id': '10314'}, {'_id': '10452'}, {'_id': '10453'}, {'_id': '10456'}, {'_id': '10457'}, {'_id': '10458'}, {'_id': '10462'}, {'_id': '10463'}, {'_id': '10466'}, {'_id': '10467'}, {'_id': '10468'}, {'_id': '10469'}, {'_id': '10472'}, {'_id': '10473'},

{'_id': '10701'}, {'_id': '10940'}, {'_id': '11203'}, {'_id': '11204'}, {'_id': '11206'}, {'_id': '11207'}, {'_id': '11208'}, {'_id': '11209'}, {'_id': '11210'}, {'_id': '11211'}, {'_id': '11212'}, {'_id': '11213'}, {'_id': '11214'}, {'_id': '11215'}, {'_id': '11216'}, {'_id': '11218'}, {'_id': '11219'}, {'_id': '11220'}, {'_id': '11221'}, {'_id': '11223'}, {'_id': '11224'}, {'_id': '11225'}, {'_id': '11226'}, {'_id': '11229'}, {'_id': '11230'}, {'_id': '11233'}, {'_id': '11234'}, {'_id': '11235'}, {'_id': '11236'}, {'_id': '11354'}, {'_id': '11355'}, {'_id': '11368'}, {'_id': '11372'}, {'_id': '11373'}, {'_id': '11375'}, {'_id': '11377'}, {'_id': '11385'}, {'_id': '11432'}, {'_id': '11434'}, {'_id': '11550'}, {'_id': '11691'}, {'_id': '11717'}, {'_id': '11746'}, {'_id': '12180'}, {'_id': '13440'}, {'_id': '14120'}, {'_id': '14150'}, {'_id': '14221'}, {'_id': '15601'}, {'_id': '16001'}, {'_id': '17013'}, {'_id': '17042'}, {'_id': '17055'}, {'_id': '17603'}, {'_id': '17701'}, {'_id': '18042'}, {'_id': '18103'}, {'_id': '18702'}, {'_id': '19020'}, {'_id': '19104'}, {'_id': '19111'}, {'_id': '19120'}, {'_id': '19124'}, {'_id': '19134'}, {'_id': '19140'}, {'_id': '19143'}, {'_id': '19145'}, {'_id': '19464'}, {'_id': '19711'}, {'_id': '20002'}, {'_id': '20011'}, {'_id': '20019'}, {'_id': '20020'}, {'_id': '20906'}, {'_id': '21061'}, {'_id': '21122'}, {'_id': '21206'}, {'_id': '21207'}, {'_id': '21215'}, {'_id': '21217'}, {'_id': '21218'}, {'_id': '21222'}, {'_id': '21224'}, {'_id': '21229'}, {'_id': '21234'}, {'_id': '21801'}, {'_id': '22003'}, {'_id': '22110'}, {'_id': '22901'}, {'_id': '23320'}, {'_id': '23452'}, {'_id': '23454'}, {'_id': '23456'}, {'_id': '23462'}, {'_id': '23464'}, {'_id': '23602'}, {'_id': '26505'}, {'_id': '27292'}, {'_id': '28540'}, {'_id': '28655'}, {'_id': '28677'}, {'_id': '29501'}, {'_id': '29801'}, {'_id': '30032'}, {'_id': '30060'}, {'_id': '30062'}, {'_id': '30223'}, {'_id': '30236'}, {'_id': '30318'}, {'_id': '30906'}, {'_id': '31907'}, {'_id': '32210'}, {'_id': '32211'}, {'_id': '32216'}, {'_id': '33012'}, {'_id': '33023'}, {'_id': '33024'}, {'_id': '33064'}, {'_id': '33142'}, {'_id': '33157'}, {'_id': '33165'}, {'_id': '33311'}, {'_id': '37211'}, {'_id': '38109'}, {'_id': '38128'}, {'_id': '43055'}, {'_id': '43130'}, {'_id': '43302'}, {'_id': '43701'}, {'_id': '44035'}, {'_id': '44060'}, {'_id': '44102'}, {'_id': '44105'}, {'_id': '44107'}, {'_id': '44130'}, {'_id': '45601'}, {'_id': '46227'}, {'_id': '46360'}, {'_id': '46383'}, {'_id': '47130'}, {'_id': '47374'}, {'_id': '47905'}, {'_id': '47906'}, {'_id': '48043'}, {'_id': '48044'}, {'_id': '48060'}, {'_id': '48066'}, {'_id': '48093'}, {'_id': '48161'}, {'_id': '48180'}, {'_id': '48185'}, {'_id': '48192'}, {'_id': '48203'}, {'_id': '48205'}, {'_id': '48213'}, {'_id': '48219'}, {'_id': '48224'}, {'_id': '48227'}, {'_id': '48228'}, {'_id': '48235'}, {'_id': '48238'}, {'_id': '48601'}, {'_id': '48823'}, {'_id': '49017'}, {'_id': '49504'}, {'_id': '49505'}, {'_id': '49509'}, {'_id': '49684'}, {'_id': '50010'}, {'_id': '53209'}, {'_id': '54401'}, {'_id': '54901'}, {'_id': '55337'}, {'_id': '60004'}, {'_id': '60016'}, {'_id': '60056'}, {'_id': '60067'}, {'_id': '60085'}, {'_id': '60103'}, {'_id': '60148'}, {'_id': '60187'}, {'_id': '60402'}, {'_id': '60411'}, {'_id': '60435'}, {'_id': '60441'}, {'_id': '60453'}, {'_id': '60505'}, {'_id': '60608'}, {'_id': '60609'}, {'_id': '60614'}, {'_id': '60617'}, {'_id': '60618'}, {'_id': '60619'}, {'_id': '60620'}, {'_id': '60621'}, {'_id': '60622'}, {'_id': '60623'}, {'_id': '60624'}, {'_id': '60625'}, {'_id': '60626'}, {'_id': '60628'}, {'_id': '60629'}, {'_id': '60632'}, {'_id': '60634'}, {'_id': '60636'}, {'_id': '60637'}, {'_id': '60638'}, {'_id': '60639'}, {'_id': '60640'}, {'_id': '60641'}, {'_id': '60643'}, {'_id': '60644'}, {'_id': '60647'}, {'_id': '60649'}, {'_id': '60650'}, {'_id': '60651'}, {'_id': '60657'}, {'_id': '62301'}, {'_id': '63031'}, {'_id': '63136'}, {'_id': '66502'}, {'_id': '70065'}, {'_id': '70072'}, {'_id': '70117'}, {'_id': '70560'}, {'_id': '72401'}, {'_id': '75007'}, {'_id': '75051'}, {'_id': '75150'}, {'_id': '75211'}, {'_id': '75216'}, {'_id': '75217'}, {'_id': '75228'}, {'_id': '77036'}, {'_id': '77901'}, {'_id': '78207'}, {'_id': '78228'}, {'_id': '78501'}, {'_id': '78520'}, {'_id': '78521'}, {'_id': '78539'}, {'_id': '78572'}, {'_id': '79907'}, {'_id': '79924'}, {'_id': '79936'}, {'_id': '80123'}, {'_id': '80221'}, {'_id': '80631'}, {'_id': '84118'}, {'_id': '84120'}, {'_id': '85023'}, {'_id': '85032'}, {'_id': '85204'}, {'_id': '85224'}, {'_id': '85364'}, {'_id': '85705'}, {'_id': '85706'}, {'_id': '85710'}, {'_id': '87105'}, {'_id': '87501'}, {'_id': '88001'}, {'_id': '88201'}, {'_id': '89115'}, {'_id': '89121'},

{'_id': '90001'}, {'_id': '90003'}, {'_id': '90004'}, {'_id': '90006'}, {'_id': '90011'}, {'_id': '90019'}, {'_id': '90022'}, {'_id': '90026'}, {'_id': '90027'}, {'_id': '90033'}, {'_id': '90034'}, {'_id': '90037'}, {'_id': '90042'}, {'_id': '90044'}, {'_id': '90063'}, {'_id': '90066'}, {'_id': '90201'}, {'_id': '90250'}, {'_id': '90255'}, {'_id': '90262'}, {'_id': '90274'}, {'_id': '90280'}, {'_id': '90631'}, {'_id': '90640'}, {'_id': '90650'}, {'_id': '90660'}, {'_id': '90701'}, {'_id': '90706'}, {'_id': '90731'}, {'_id': '90745'}, {'_id': '90805'}, {'_id': '90813'}, {'_id': '91331'}, {'_id': '91335'}, {'_id': '91342'}, {'_id': '91402'}, {'_id': '91605'}, {'_id': '91702'}, {'_id': '91706'}, {'_id': '91710'}, {'_id': '91720'}, {'_id': '91732'}, {'_id': '91744'}, {'_id': '91745'}, {'_id': '91754'}, {'_id': '91766'}, {'_id': '91770'}, {'_id': '91786'}, {'_id': '91801'}, {'_id': '91910'}, {'_id': '91911'}, {'_id': '91950'}, {'_id': '91977'}, {'_id': '92020'}, {'_id': '92021'}, {'_id': '92054'}, {'_id': '92071'}, {'_id': '92083'}, {'_id': '92105'}, {'_id': '92114'}, {'_id': '92115'}, {'_id': '92126'}, {'_id': '92154'}, {'_id': '92324'}, {'_id': '92335'}, {'_id': '92345'}, {'_id': '92376'}, {'_id': '92392'}, {'_id': '92404'}, {'_id': '92503'}, {'_id': '92509'}, {'_id': '92553'}, {'_id': '92627'}, {'_id': '92630'}, {'_id': '92646'}, {'_id': '92647'}, {'_id': '92680'}, {'_id': '92683'}, {'_id': '92701'}, {'_id': '92703'}, {'_id': '92704'}, {'_id': '92707'}, {'_id': '92708'}, {'_id': '92714'}, {'_id': '92804'}, {'_id': '92805'}, {'_id': '93030'}, {'_id': '93033'}, {'_id': '93065'}, {'_id': '93257'}, {'_id': '93277'}, {'_id': '93307'}, {'_id': '93309'}, {'_id': '93454'}, {'_id': '93550'}, {'_id': '93612'}, {'_id': '93727'}, {'_id': '94015'}, {'_id': '94080'}, {'_id': '94086'}, {'_id': '94110'}, {'_id': '94112'}, {'_id': '94122'}, {'_id': '94501'}, {'_id': '94509'}, {'_id': '94533'}, {'_id': '94536'}, {'_id': '94544'}, {'_id': '94550'}, {'_id': '94558'}, {'_id': '94565'}, {'_id': '94587'}, {'_id': '95035'}, {'_id': '95076'}, {'_id': '95122'}, {'_id': '95123'}, {'_id': '95127'}, {'_id': '95207'}, {'_id': '95336'}, {'_id': '95340'}, {'_id': '95350'}, {'_id': '95351'}, {'_id': '95380'}, {'_id': '95608'}, {'_id': '95616'}, {'_id': '95687'}, {'_id': '95823'}, {'_id': '95926'}, {'_id': '96734'}, {'_id': '96744'}, {'_id': '96797'}, {'_id': '96818'}, {'_id': '96819'}, {'_id': '98031'}]

1.10 10. Which city has the most zip codes?

1. Include the state in case the city name isn't unique across the states.
2. Be sure to handle the possibility of a tie.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use `collection.aggregate` with:

1. `$group` to group the documents by `city` and `state`, counting aggregated documents.
2. `$sort` with `"totalCount": -1` to sort by total count in descending order.
3. `$limit` with `1` to get the first document. This also handles ties
    (we only get the one that occurs first)

#### Full Command:
```js
  collection.aggregate([
```
```

```
    {{"$group": {{"_id": {{"city": "$city", "state": "$state"}}, "totalCount":␣
  ↪{{"$sum": 1}}}}}},
    {{"$sort": {{"totalCount": -1}}}},
    {{"$limit": 1}}
  ]).next()["_id"]
```

Results:

The city with the largest population is: \
{collection.aggregate([
  { "$group": { "_id": { "city": "$city", "state": "$state"}, "totalCount": {␣
  ↪"$sum": 1}}},
  { "$sort": { "totalCount": -1}},
  { "$limit": 1}
]).next()["_id"]}

"""))

Strategy We can use `collection.aggregate` with:

1. `$group` to group the documents by `city` and `state`, counting aggregated documents.
2. `$sort` with `"totalCount": -1` to sort by total count in descending order.
3. `$limit` with `1` to get the first document. This also handles ties (we only get the one that occurs first)

Full Command:

```
collection.aggregate([
  {"$group": {"_id": {"city": "$city", "state": "$state"}, "totalCount": {"$sum": 1}}},
  {"$sort": {"totalCount": -1}},
  {"$limit": 1}
]).next()["_id"]
```

Results: The city with the largest population is: {'city': 'HOUSTON', 'state': 'TX'}

---

2 Part 2 (20 Points)

2.1 1. Display the zip codes in Louisiana (LA) sorted by population from largest to smallest.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use `collection.aggregate` with:
```

```
1. `$match` to for documents in `LA`.
2. `$sort` with `"pop": -1` to sort by population in descending order.
3. `$project` with `_id: 1` to only get the `_id` field.

#### Full Command:
```js
  collection.aggregate([
    {{"$match": {{"state": "LA"}}}},
    {{"$sort": {{"pop": -1}}}},
    {{"$_id": 1}}
  ]).next()
```

Results:

The zip code with the largest population in LA is: \
{list(
  collection.aggregate([
  { "$match": { "state": "LA"}},
  { "$sort": { "pop": -1}},
  {"$project": {"_id": 1}}
])
)}

"""))
```

Strategy We can use `collection.aggregate` with:

1. `$match` to for documents in `LA`.
2. `$sort` with `"pop": -1` to sort by population in descending order.
3. `$project` with `_id: 1` to only get the `_id` field.

Full Command:

```
collection.aggregate([
  {"$match": {"state": "LA"}},
  {"$sort": {"pop": -1}},
  {"$_id": 1}
]).next()
```

Results: The zip code with the largest population in LA is: [{'_id': '70072'}, {'_id': '70117'}, {'_id': '70560'}, {'_id': '70065'}, {'_id': '70601'}, {'_id': '70119'}, {'_id': '70122'}, {'_id': '70570'}, {'_id': '70003'}, {'_id': '70126'}, {'_id': '70115'}, {'_id': '70605'}, {'_id': '70118'}, {'_id': '70001'}, {'_id': '70056'}, {'_id': '70301'}, {'_id': '71360'}, {'_id': '70058'}, {'_id': '71203'}, {'_id': '70094'}, {'_id': '70802'}, {'_id': '70726'}, {'_id': '70506'}, {'_id': '70816'}, {'_id': '71106'}, {'_id': '71202'}, {'_id': '70043'}, {'_id': '70501'}, {'_id': '70808'}, {'_id': '70805'}, {'_id': '70663'}, {'_id': '70114'}, {'_id': '70127'}, {'_id': '70582'}, {'_id': '71107'}, {'_id': '70458'}, {'_id': '71291'}, {'_id': '70815'}, {'_id': '71109'}, {'_id': '71459'}, {'_id':

‘70131’}, {’_id’: ‘70005’}, {’_id’: ‘71111’}, {’_id’: ‘71220’}, {’_id’: ‘70433’}, {’_id’: ‘71270’}, {’_id’: ‘70068’}, {’_id’: ‘70806’}, {’_id’: ‘70364’}, {’_id’: ‘71112’}, {’_id’: ‘70503’}, {’_id’: ‘70123’}, {’_id’: ‘71301’}, {’_id’: ‘71457’}, {’_id’: ‘70363’}, {’_id’: ‘71118’}, {’_id’: ‘70807’}, {’_id’: ‘70124’}, {’_id’: ‘70125’}, {’_id’: ‘70448’}, {’_id’: ‘71201’}, {’_id’: ‘70810’}, {’_id’: ‘71446’}, {’_id’: ‘71055’}, {’_id’: ‘71303’}, {’_id’: ‘70737’}, {’_id’: ‘70535’}, {’_id’: ‘70510’}, {’_id’: ‘70817’}, {’_id’: ‘70508’}, {’_id’: ‘70062’}, {’_id’: ‘71108’}, {’_id’: ‘70714’}, {’_id’: ‘70002’}, {’_id’: ‘70634’}, {’_id’: ‘71105’}, {’_id’: ‘70427’}, {’_id’: ‘70128’}, {’_id’: ‘70791’}, {’_id’: ‘70360’}, {’_id’: ‘70460’}, {’_id’: ‘70526’}, {’_id’: ‘71292’}, {’_id’: ‘70342’}, {’_id’: ‘70461’}, {’_id’: ‘70053’}, {’_id’: ‘70006’}, {’_id’: ‘71302’}, {’_id’: ‘70764’}, {’_id’: ‘70116’}, {’_id’: ‘70438’}, {’_id’: ‘70401’}, {’_id’: ‘70454’}, {’_id’: ‘70578’}, {’_id’: ‘70809’}, {’_id’: ‘70130’}, {’_id’: ‘70538’}, {’_id’: ‘70403’}, {’_id’: ‘71295’}, {’_id’: ‘70586’}, {’_id’: ‘71104’}, {’_id’: ‘70544’}, {’_id’: ‘70129’}, {’_id’: ‘71037’}, {’_id’: ‘70811’}, {’_id’: ‘70392’}, {’_id’: ‘70769’}, {’_id’: ‘70345’}, {’_id’: ‘70814’}, {’_id’: ‘70121’}, {’_id’: ‘70394’}, {’_id’: ‘71103’}, {’_id’: ‘70767’}, {’_id’: ‘71129’}, {’_id’: ‘70785’}, {’_id’: ‘70611’}, {’_id’: ‘71282’}, {’_id’: ‘71052’}, {’_id’: ‘70520’}, {’_id’: ‘70113’}, {’_id’: ‘70507’}, {’_id’: ‘70422’}, {’_id’: ‘70546’}, {’_id’: ‘70070’}, {’_id’: ‘70812’}, {’_id’: ‘71269’}, {’_id’: ‘71101’}, {’_id’: ‘70346’}, {’_id’: ‘70444’}, {’_id’: ‘71351’}, {’_id’: ‘70525’}, {’_id’: ‘70669’}, {’_id’: ‘71251’}, {’_id’: ‘70374’}, {’_id’: ‘70820’}, {’_id’: ‘71119’}, {’_id’: ‘70092’}, {’_id’: ‘70047’}, {’_id’: ‘70037’}, {’_id’: ‘70517’}, {’_id’: ‘71463’}, {’_id’: ‘70548’}, {’_id’: ‘71019’}, {’_id’: ‘71483’}, {’_id’: ‘71334’}, {’_id’: ‘71373’}, {’_id’: ‘70452’}, {’_id’: ‘70583’}, {’_id’: ‘71263’}, {’_id’: ‘70032’}, {’_id’: ‘71115’}, {’_id’: ‘70446’}, {’_id’: ‘70818’}, {’_id’: ‘71047’}, {’_id’: ‘70633’}, {’_id’: ‘70518’}, {’_id’: ‘70085’}, {’_id’: ‘70739’}, {’_id’: ‘70445’}, {’_id’: ‘71038’}, {’_id’: ‘71254’}, {’_id’: ‘70390’}, {’_id’: ‘70760’}, {’_id’: ‘70748’}, {’_id’: ‘70533’}, {’_id’: ‘71006’}, {’_id’: ‘70075’}, {’_id’: ‘71241’}, {’_id’: ‘70529’}, {’_id’: ‘70084’}, {’_id’: ‘71040’}, {’_id’: ‘70648’}, {’_id’: ‘70592’}, {’_id’: ‘71232’}, {’_id’: ‘70090’}, {’_id’: ‘70554’}, {’_id’: ‘70668’}, {’_id’: ‘71449’}, {’_id’: ‘70041’}, {’_id’: ‘70344’}, {’_id’: ‘70083’}, {’_id’: ‘70087’}, {’_id’: ‘70426’}, {’_id’: ‘71075’}, {’_id’: ‘71322’}, {’_id’: ‘70774’}, {’_id’: ‘71328’}, {’_id’: ‘70112’}, {’_id’: ‘71429’}, {’_id’: ‘71082’}, {’_id’: ‘71342’}, {’_id’: ‘71245’}, {’_id’: ‘70788’}, {’_id’: ‘71367’}, {’_id’: ‘70631’}, {’_id’: ‘70512’}, {’_id’: ‘70775’}, {’_id’: ‘70462’}, {’_id’: ‘70591’}, {’_id’: ‘70466’}, {’_id’: ‘71417’}, {’_id’: ‘71343’}, {’_id’: ‘71227’}, {’_id’: ‘70712’}, {’_id’: ‘70819’}, {’_id’: ‘70380’}, {’_id’: ‘70543’}, {’_id’: ‘71486’}, {’_id’: ‘70343’}, {’_id’: ‘70770’}, {’_id’: ‘70339’}, {’_id’: ‘70549’}, {’_id’: ‘71403’}, {’_id’: ‘70577’}, {’_id’: ‘70079’}, {’_id’: ‘70647’}, {’_id’: ‘71064’}, {’_id’: ‘70341’}, {’_id’: ‘70437’}, {’_id’: ‘71418’}, {’_id’: ‘70395’}, {’_id’: ‘70359’}, {’_id’: ‘71409’}, {’_id’: ‘70754’}, {’_id’: ‘70514’}, {’_id’: ‘71071’}, {’_id’: ‘70559’}, {’_id’: ‘70777’}, {’_id’: ‘70711’}, {’_id’: ‘71068’}, {’_id’: ‘71001’}, {’_id’: ‘70443’}, {’_id’: ‘71467’}, {’_id’: ‘71433’}, {’_id’: ‘71049’}, {’_id’: ‘71378’}, {’_id’: ‘71225’}, {’_id’: ‘70357’}, {’_id’: ‘71441’}, {’_id’: ‘71234’}, {’_id’: ‘70071’}, {’_id’: ‘71411’}, {’_id’: ‘70397’}, {’_id’: ‘70730’}, {’_id’: ‘70431’}, {’_id’: ‘70377’}, {’_id’: ‘70542’}, {’_id’: ‘70441’}, {’_id’: ‘70589’}, {’_id’: ‘71260’}, {’_id’: ‘70049’}, {’_id’: ‘70354’}, {’_id’: ‘70555’}, {’_id’: ‘71371’}, {’_id’: ‘70734’}, {’_id’: ‘70656’}, {’_id’: ‘71110’}, {’_id’: ‘71222’}, {’_id’: ‘71259’}, {’_id’: ‘71465’}, {’_id’: ‘70719’}, {’_id’: ‘70030’}, {’_id’: ‘70653’}, {’_id’: ‘71350’}, {’_id’: ‘71357’}, {’_id’: ‘70776’}, {’_id’: ‘70051’}, {’_id’: ‘71327’}, {’_id’: ‘71033’}, {’_id’: ‘71369’}, {’_id’: ‘70733’}, {’_id’: ‘70655’}, {’_id’: ‘70447’}, {’_id’: ‘71023’}, {’_id’: ‘71341’}, {’_id’: ‘71346’}, {’_id’: ‘70722’}, {’_id’: ‘71078’}, {’_id’: ‘70372’}, {’_id’: ‘71060’}, {’_id’: ‘71235’}, {’_id’: ‘71277’}, {’_id’: ‘71407’}, {’_id’: ‘71016’}, {’_id’: ‘70528’}, {’_id’: ‘70581’}, {’_id’: ‘70052’}, {’_id’: ‘70639’}, {’_id’: ‘70515’}, {’_id’: ‘71353’}, {’_id’: ‘70755’}, {’_id’: ‘71468’}, {’_id’: ‘70516’}, {’_id’: ‘70420’}, {’_id’: ‘70763’}, {’_id’: ‘71268’}, {’_id’: ‘70657’}, {’_id’: ‘71362’}, {’_id’: ‘70086’}, {’_id’: ‘70449’}, {’_id’: ‘71275’}, {’_id’: ‘71404’}, {’_id’: ‘70661’}, {’_id’: ‘70039’}, {’_id’:

‘70744’}, {’_id’: ‘70660’}, {’_id’: ‘70710’}, {’_id’: ‘71340’}, {’_id’: ‘70040’}, {’_id’: ‘71238’}, {’_id’: ‘71454’}, {’_id’: ‘71028’}, {’_id’: ‘71051’}, {’_id’: ‘71422’}, {’_id’: ‘71261’}, {’_id’: ‘71477’}, {’_id’: ‘71354’}, {’_id’: ‘71366’}, {’_id’: ‘70757’}, {’_id’: ‘71456’}, {’_id’: ‘70456’}, {’_id’: ‘71018’}, {’_id’: ‘70723’}, {’_id’: ‘71423’}, {’_id’: ‘70532’}, {’_id’: ‘71485’}, {’_id’: ‘70783’}, {’_id’: ‘71067’}, {’_id’: ‘71355’}, {’_id’: ‘71061’}, {’_id’: ‘70715’}, {’_id’: ‘70761’}, {’_id’: ‘71430’}, {’_id’: ‘71323’}, {’_id’: ‘70658’}, {’_id’: ‘70353’}, {’_id’: ‘70645’}, {’_id’: ‘70717’}, {’_id’: ‘70356’}, {’_id’: ‘71027’}, {’_id’: ‘70637’}, {’_id’: ‘71419’}, {’_id’: ‘70753’}, {’_id’: ‘71237’}, {’_id’: ‘70450’}, {’_id’: ‘71368’}, {’_id’: ‘71325’}, {’_id’: ‘71219’}, {’_id’: ‘70358’}, {’_id’: ‘71266’}, {’_id’: ‘71039’}, {’_id’: ‘71031’}, {’_id’: ‘70630’}, {’_id’: ‘70749’}, {’_id’: ‘71424’}, {’_id’: ‘71073’}, {’_id’: ‘71439’}, {’_id’: ‘71065’}, {’_id’: ‘71003’}, {’_id’: ‘70652’}, {’_id’: ‘70778’}, {’_id’: ‘70031’}, {’_id’: ‘71479’}, {’_id’: ‘71226’}, {’_id’: ‘71375’}, {’_id’: ‘71030’}, {’_id’: ‘71070’}, {’_id’: ‘71229’}, {’_id’: ‘71024’}, {’_id’: ‘70721’}, {’_id’: ‘70725’}, {’_id’: ‘70759’}, {’_id’: ‘71326’}, {’_id’: ‘71046’}, {’_id’: ‘71286’}, {’_id’: ‘71032’}, {’_id’: ‘70632’}, {’_id’: ‘71432’}, {’_id’: ‘71069’}, {’_id’: ‘71063’}, {’_id’: ‘71223’}, {’_id’: ‘70740’}, {’_id’: ‘70080’}, {’_id’: ‘71469’}, {’_id’: ‘70750’}, {’_id’: ‘71450’}, {’_id’: ‘71333’}, {’_id’: ‘71264’}, {’_id’: ‘71276’}, {’_id’: ‘71438’}, {’_id’: ‘71004’}, {’_id’: ‘71462’}, {’_id’: ‘70752’}, {’_id’: ‘70772’}, {’_id’: ‘71416’}, {’_id’: ‘71336’}, {’_id’: ‘71318’}, {’_id’: ‘71043’}, {’_id’: ‘71447’}, {’_id’: ‘71072’}, {’_id’: ‘70057’}, {’_id’: ‘70762’}, {’_id’: ‘71044’}, {’_id’: ‘70643’}, {’_id’: ‘70340’}, {’_id’: ‘70584’}, {’_id’: ‘71358’}, {’_id’: ‘71406’}, {’_id’: ‘71473’}, {’_id’: ‘70455’}, {’_id’: ‘70736’}, {’_id’: ‘70773’}, {’_id’: ‘70720’}, {’_id’: ‘71339’}, {’_id’: ‘70375’}, {’_id’: ‘70654’}, {’_id’: ‘70537’}, {’_id’: ‘71472’}, {’_id’: ‘70650’}, {’_id’: ‘70091’}, {’_id’: ‘70355’}, {’_id’: ‘71331’}, {’_id’: ‘71008’}, {’_id’: ‘70453’}, {’_id’: ‘71455’}, {’_id’: ‘71256’}, {’_id’: ‘71412’}, {’_id’: ‘71048’}, {’_id’: ‘71007’}, {’_id’: ‘71356’}, {’_id’: ‘70780’}, {’_id’: ‘71250’}, {’_id’: ‘71045’}, {’_id’: ‘71451’}, {’_id’: ‘71426’}, {’_id’: ‘71029’}, {’_id’: ‘70531’}, {’_id’: ‘70729’}, {’_id’: ‘70036’}, {’_id’: ‘71435’}, {’_id’: ‘70662’}, {’_id’: ‘71427’}, {’_id’: ‘71444’}, {’_id’: ‘70792’}, {’_id’: ‘71239’}, {’_id’: ‘71461’}, {’_id’: ‘71466’}, {’_id’: ‘71034’}, {’_id’: ‘71059’}, {’_id’: ‘70436’}, {’_id’: ‘71316’}, {’_id’: ‘71425’}, {’_id’: ‘70552’}, {’_id’: ‘70789’}, {’_id’: ‘71243’}, {’_id’: ‘70801’}, {’_id’: ‘71401’}, {’_id’: ‘70781’}, {’_id’: ‘71079’}, {’_id’: ‘70467’}, {’_id’: ‘71280’}, {’_id’: ‘70756’}, {’_id’: ‘70067’}, {’_id’: ‘70732’}]

2.2 2. Provide a python or javascript code snippet that will determine which state has the fewest zip codes.

It is not necessary to provide a complete working program.

[ ]:
```
display(Markdown(f"""
#### Strategy

We can use `collection.aggregate` with:

1. `$group` to group the documents by `state`, counting aggregated documents.
2. `$sort` with `"totalCount": 1` to sort by total count in ascending order.
3. `$limit` with `1` to get the first document. This also handles ties.
4. `$project` with `_id: 1` to only get the state name field.

#### Full Command:
```

```
```js
  collection.aggregate([
    {{"$group": {{"_id": "$state", "totalCount": {{"$sum": 1}}}}}},
    {{"$sort": {{"totalCount": 1}}}},
    {{"$limit": 1}},
    {{"$project": {{"_id": 1}}}}
  ]).next()
```

Results:

The state with the fewest zip codes is: \
{collection.aggregate([
  { "$group": { "_id": "$state", "totalCount": { "$sum": 1}}},
  { "$sort": { "totalCount": 1}},
  { "$limit": 1},
  { "$project": { "_id": 1}}
]).next()}
"""))
```

Strategy We can use `collection.aggregate` with:

1. `$group` to group the documents by `state`, counting aggregated documents.
2. `$sort` with `"totalCount": 1` to sort by total count in ascending order.
3. `$limit` with `1` to get the first document. This also handles ties.
4. `$project` with `_id: 1` to only get the state name field.

Full Command:

```
  collection.aggregate([
    {"$group": {"_id": "$state", "totalCount": {"$sum": 1}}},
    {"$sort": {"totalCount": 1}},
    {"$limit": 1},
    {"$project": {"_id": 1}}
  ]).next()
```

Results: The state with the fewest zip codes is: {'_id': 'DC'}

2.3 3. Write a javascript code snippet suitable for running in the mongo shell that will generate a new database collection of 100 random credit card charge amounts with the following schema.

It is not necessary to provide a complete working program.

```
{
  cardNo: 12 character string,
  expDate: {mm: integer, yy: integer},
```

```
  amount: float with 2 decimal places,
  secCode: integer
}
```

[ ]:
```
/* Assuming we are already in a connection to a database
   and that we have a collection called "charges",
   then: */

db.charges.drop();

for (let chargeId = 1; chargeId <= 100; chargeId++) {
  db.charges.insert({
    "_id": chargeId,
    "cardNo": "1234567890123456",
    "expDate": [
      Math.floor(Math.random() * 12),
      Math.floor(Math.random() * 22) ],
    "amount": Math.random() * 1000,
    "secCode": Math.floor(Math.random() * 999)
  });
}
```

2.4 4. What MongoDB statement(s) would you use to add a four-digit integer field named "plusFour" initialized to "0000" to each entry of the zipcode collection?

We can use the `updateMany()` function with the empty selector to match all documents in the collection. We then use `$set` to set the `plusFour` field to `0000` for each document.

Command:

```
db.zipcodes.updateMany({}, {$set: {plusFour: "0000"});
```

---

3 Extra Credit (5 Points)

Provide python program that will use the zips MongoDB database (and no other data or source) to determine which zipcode is nearest the center of the continental United States (that is, considering only zipcodes in the continental United States (i.e., all states other than AK and HI). Looking up the answer and simply hardcoding it in your code is unacceptable.

Strategy

1. Find the center of the United States.

- Find the average longitude and latitude of all zip codes.

- Use `collection.aggregate` with: `-$project` to get the longitude and latitude fields (I know I can index directly but it was somewhat messy).
  - `$group` to group ALL the documents, with average longitudes, latitudes.
  - `$project` to only pick the average longitude and latitude.

2. Find the zip code closest to the center of the United States.

- Use `collection.aggregate` with:
  - `$geoNear` to find the nearest zip code to the center of the United States.

[ ]:
```
# Find Center

center = collection.aggregate([
  { "$project": {
      "_id": 1,
      "longitude": { "$arrayElemAt": ["$loc", 0] },
      "latitude" : { "$arrayElemAt": ["$loc", 1] }
    }
  }, {
    "$group": {
      "_id": "null",
      "avgLong": { "$avg": "$longitude" },
      "avgLat": { "$avg": "$latitude" }
    }
  }, {
    "$project": {
      "_id": 0,
      "avgLong": 1,
      "avgLat": 1
    }
  }]).next()

center
```

[ ]:
```
{'avgLong': -90.89122853061015, 'avgLat': 39.01513788491806}
```

[ ]:
```
# Find Closest Zip Code

collection.create_index([("loc", "2d")])

closest = collection.aggregate([
  { "$geoNear": {
      "near": {
        "type": "Point",
        "coordinates": [center["avgLong"], center["avgLat"]]
      },
      "distanceField": "distance",
      "spherical": False,
```

```
      "key": "loc"
    }
  }
]).next()

closest
```

[ ]:
```
{'_id': '63379',
 'city': 'TROY',
 'loc': [-90.962449, 39.001212],
 'pop': 7636,
 'state': 'MO',
 'distance': 6352.689353294055}
```