Adding support for spatial datasets in retrieverdash

By Kush Kothari, a passionate Web and Python developer

Personal and Contact Information:

Full Name: Kush Mukesh Kothari (he/him)
Primary Email: kushkothari2001@gmail.com
Secondary Email (via my university): kmkothari_b19@ce.vjti.ac.in
Github Profile: https://github.com/kkothari2001
My Resume/ CV: Drive link
LinkedIn Profile: https://www.linkedin.com/in/kush-kothari-ba013218b/
University: Veermata Jijabai Technological Institute, Mumbai
Ongoing Degree: Bachelor of Technology in Computer Engineering
Phone Number: +919833230036 (feel free to contact me via Whatsapp and Signal too!)

About Me:

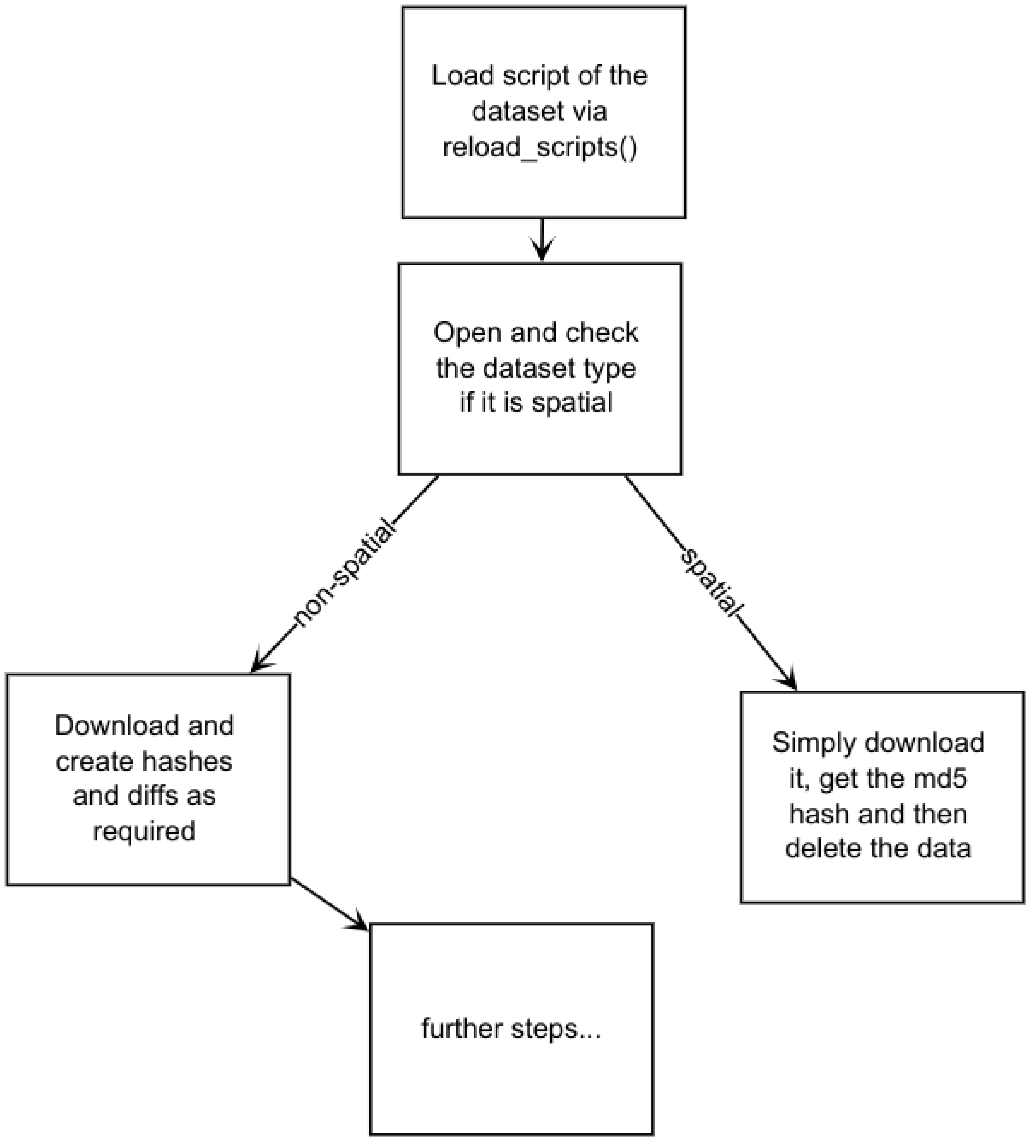
Hello there! I am Kush Kothari, a passionate web developer, a budding ML enthusiast and bubbly coder/designer. I am currently a sophomore at VJTI, Mumbai and am super excited to dive into the world of open source!

Abstract:

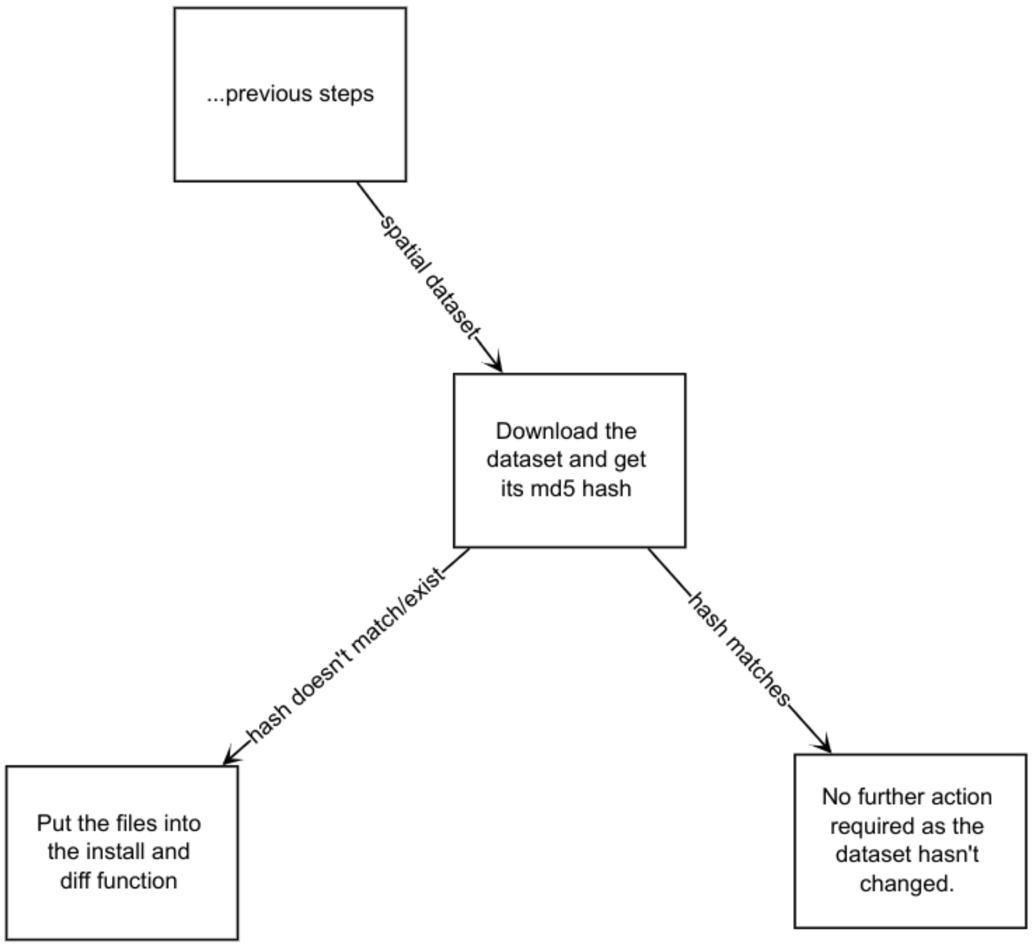
A major requirement for the current status of the dashboard is the implementation of a download and install pipeline for tables having spatial data in the form of vector images/data. According to the retriever docs, currently the retriever library uses the PostGIS extension for the same. Therefore, the install process to be used for the dashboard will make use of the postgres engine. The spatial datasets are already tested for only download and changes are detected using the md5 hash. This proposal will further that, into proper install of these datasets into Postgres and find out differences in the databases using PostGIS joins. The results obtained will then be converted into HTML diffs just like we do for non-spatial databases.

Further Technical Details:

The current download flow for the dataset is as follows:

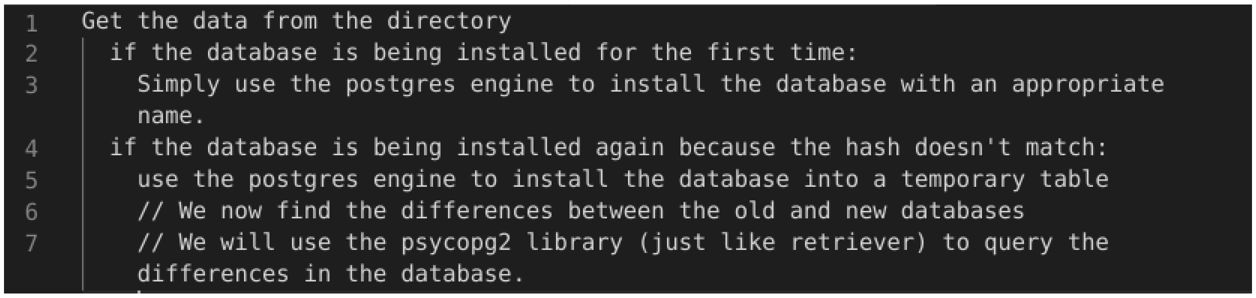

Load script of the dataset via reload_scripts()
Open and check the dataset type if it is spatial
non-spatial
spatial
Download and create hashes and diffs as required
Simply download it, get the md5 hash and then delete the data
further steps...

Planned extension of the algorithm:


...previous steps
spatial dataset
Download the dataset and get its md5 hash
hash doesn't match/exist
hash matches
Put the files into the install and diff function
No further action required as the dataset hasn't changed.

The install and diff function will work as given below:

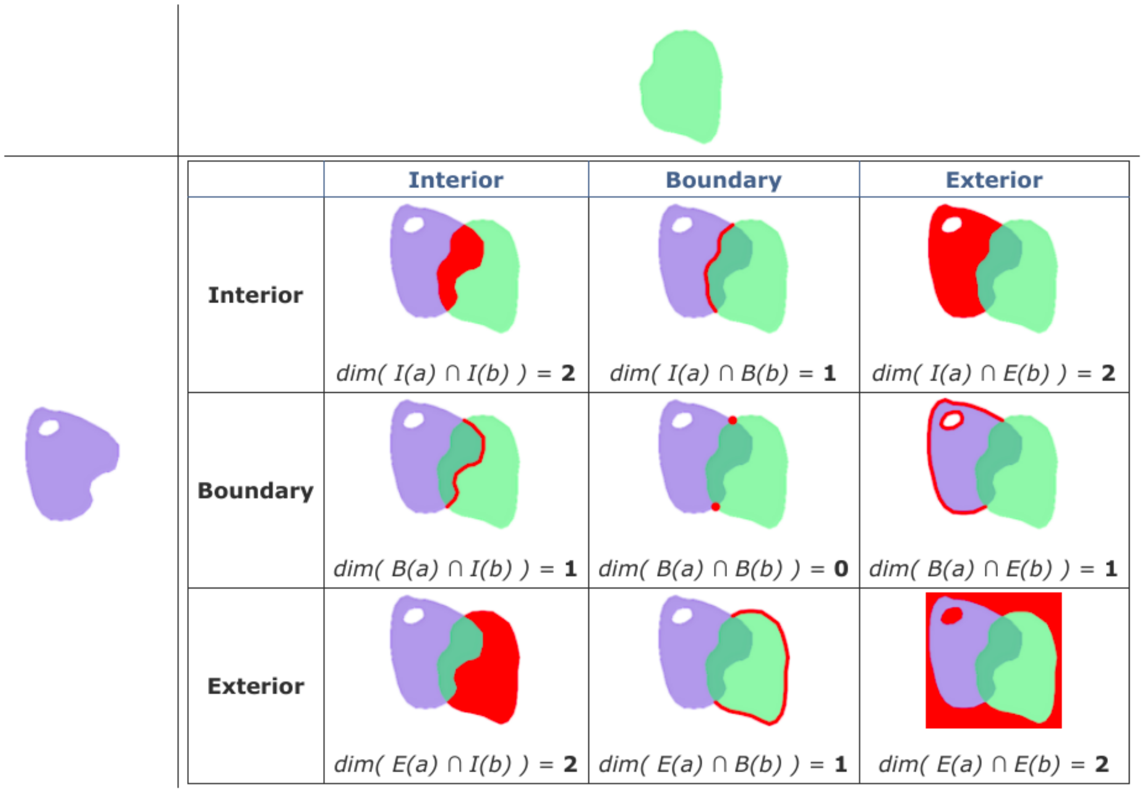
Pseudocode:



```
Get the data from the directory
  if the database is being installed for the first time:
    Simply use the postgres engine to install the database with an appropriate
    name.
  if the database is being installed again because the hash doesn't match:
    use the postgres engine to install the database into a temporary table
    // We now find the differences between the old and new databases
    // We will use the psycopg2 library (just like retriever) to query the
    differences in the database.
```

How the diff queries are made:
Since the data is spatial in nature we will need to use some of the JOINS provided by PostGIS in order to get the diff.

Source: https://postgis.net/docs/postgis_usage.html#using_postgis_query



The above is a representation of the DE-9IM topological model (used by PostGIS) , and we can use these to find the exact differences between the 2 databases.

The exact query function call to find the diff will be ST_Relate (this uses the 9 dimensional approach as explained above):
Source: https://postgis.net/docs/ST_Relate.html

If required we can also use these functions for the diff queries(but ST_Relate seemed most appropriate to me):

- ST_Contains
- ST_ContainsProperly
- ST_Covers
- ST_CoveredBy
- ST_Crosses
- ST_Disjoint
- ST_Equals
- ST_Intersects
- ST_Overlaps

- ST_Touches
- ST_Within

Note: The above functions are just wrapper functions for the ST_Relate command which uses the 9 dimensional matrix given above to simultaneously give us all sorts of custom spatial data joins.

Since we are using psycopg2, the output from PostGIS can be easily saved to a text file using python file handling. This text file thus stores the diff in the two values.

Summary of the install section and diff section:

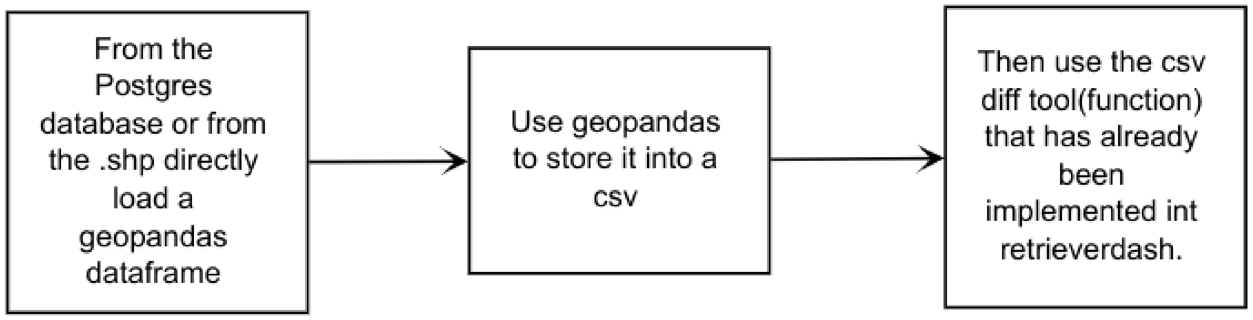
I*nstall the dataset using the postgres engine*

Use the JOINS provided by PostGIS to find the diffs

Store the diffs in a text file which will be later accessed by Django

Alternative Diff Finding Algorithm:

This second method doesn't give very accurate diffs but it is useful for quick glances into the differences. Some datasets might not be supported by the geopandas library and larger databases might be more difficult to load into python.


From the Postgres database or from the .shp directly load a geopandas dataframe
Use geopandas to store it into a csv
Then use the csv diff tool(function) that has already been implemented int retrieverdash.

I believe the second is faster and easier to implement but the first one gives a more thorough testing about whether the database was installed properly. Also like it was mentioned earlier, loading entire tables into python might prove difficult.

While this is an optional feature to have, it provides a very useful way to actually find out whether the databases have been significantly altered.

Schedule of Deliverables and some implementation details:

Application Review Period:

- Go through the PostGIS docs in detail
- Learn more about the datasets I will be working with from the scripts section.
- Add PRs to retriever and retriever dash too and see if there are any issues I can fixor code I can refactor.
- I have a few college exams coming up in this period so I may be slightly busy with that. A few subjects from my curriculum overlap too (DBMS and Web Technologies). So I guess they would be excellent ways to revise some concepts before beginning work.

Community Bonding Period:

I understand that Numfocus expects us to maintain a blog while working on the projects. So I will be regularly writing blog posts and updates on my medium page linked below:
https://medium.com/@kushkothari2001

I believe that the community bonding period will be the best time for me to explore the existing spatial datasets. I can go about manually downloading and installing a few of them via self-contained python scripts. This is to easily foresee and catch some errors I might face early in the development period.

For this year's GSoC, I believe I can easily dedicate at least 25hrs a week including my ongoing college too.

Weeks 1 & 2:

1. I believe it will be best to spend the first day to quickly get a nice grounding on the code to be changed.
2. The first week will be spent in designing and coding functions responsible for downloading the data from the url, and hashing each file associated with the database.
3. I believe the schema of dataset_details.json (in dashboard_script) might also have to be altered in order to add support for certain datasets
4. The second week will be spent in integrating the postgres engine to automatically download the spatial datasets.
5. I believe that for this part I will need to refactor some existing code since the current code only handles the making of the csv files.
6. This week, the code required for installing the databases into Postgres will also be completed.

Week 3:

1. This week will mainly focus on prototyping a very basic diff generator so as to quickly find diffs for smaller spatial datasets.
2. This will make use of the alternative diff finding algorithm explained above.
3. This will be tested on all datasets to see if it provides the necessary speed and accuracy.

Weeks 4 & 5:

1. These weeks, I will be working on making a better diff generator that makes use of the Postgres prompt.
2. The library that will be used to enter these commands will be psycopg2.
3. Like mentioned above, special PostGIS joins will be used to create the diff and differentiate between insertions and deletions.
4. The week will have a lot of testing of commands in the CLI in order to find the best join for the required data.
5. In week 5, this join will be used and a database will be created using psycopg2 and then return the output in a text file/json file.
6. This text/json file can later be parsed by Django to create the diff view just like for non-spatial databases

Week 6:

1. This week, we will now modify the existing Django app to now parse these text/json files and create the diff view.
2. Separate views and templates will have to be made and used for the same.
3. Extra code might need to be refactored too.

Week 7:

This week will be a buffer period to incorporate any new errors we find and solve them.
I can also work on previously pending things.

Weeks 8 & 9:

1. During these weeks I will work on the documentation of the project and thus improve it to not only include the new functions added by me, but also add some missing docs.
2. Issue #47 reports some missing documentation and that can be improved too.

Development Experience:

I have been a web and python developer for quite some time. While I am majorly excited about ML and data-related projects, I have experience in a variety of projects and fields. Feel free to browse my github and ask me questions about the repos there!

Some of the best python projects I have worked on include:

twAIn: As the punny name suggests, we created an AI in python that can write awesome stories after being trained on many modern classics! For this project I created the Web UI, and worked on the GPT-2 model that produces the stories. A demo video can be found at this drive link.

Gesture Controlled Mouse using Opencv: OpenCV (Python) was used to create a model that detects the hand, either through movement, colored tape or through interest detection algorithms. A nice demo is available here.

Turn-the-Tables: A Data Extraction Web App that converts Images to Tables. The Exported Table could be downloaded in xlsx or csv format. The backend was done in OpenCV, Tesseract and Flask.

Image-Processing Algorithms: A variety of image processing algorithms were implemented from scratch by me in libraries such as numpy and Pillow. This is done in an attempt to recreate famous algorithms like Sobel Edge Detection and ROI mapping.

A few Django projects under my name:
To-Do App: App that can be used to maintain a list of tasks that can be dynamically added and deleted once completed. Deployment at: http://djangogetitdone.herokuapp.com/

URL Shortener: A bitly clone that stores URLs in a database as a hash and redirects when the URL is clicked. Deployment at: https://djangourlshortner.herokuapp.com/