**Problem Statement**

Basic Design and deployment of a system for user click analysis.

The Solution can roughly be divided into three parts:

1. Generating Data
2. Modelling Data
3. Building a system deploying the Model

We will now go into details for each one of them

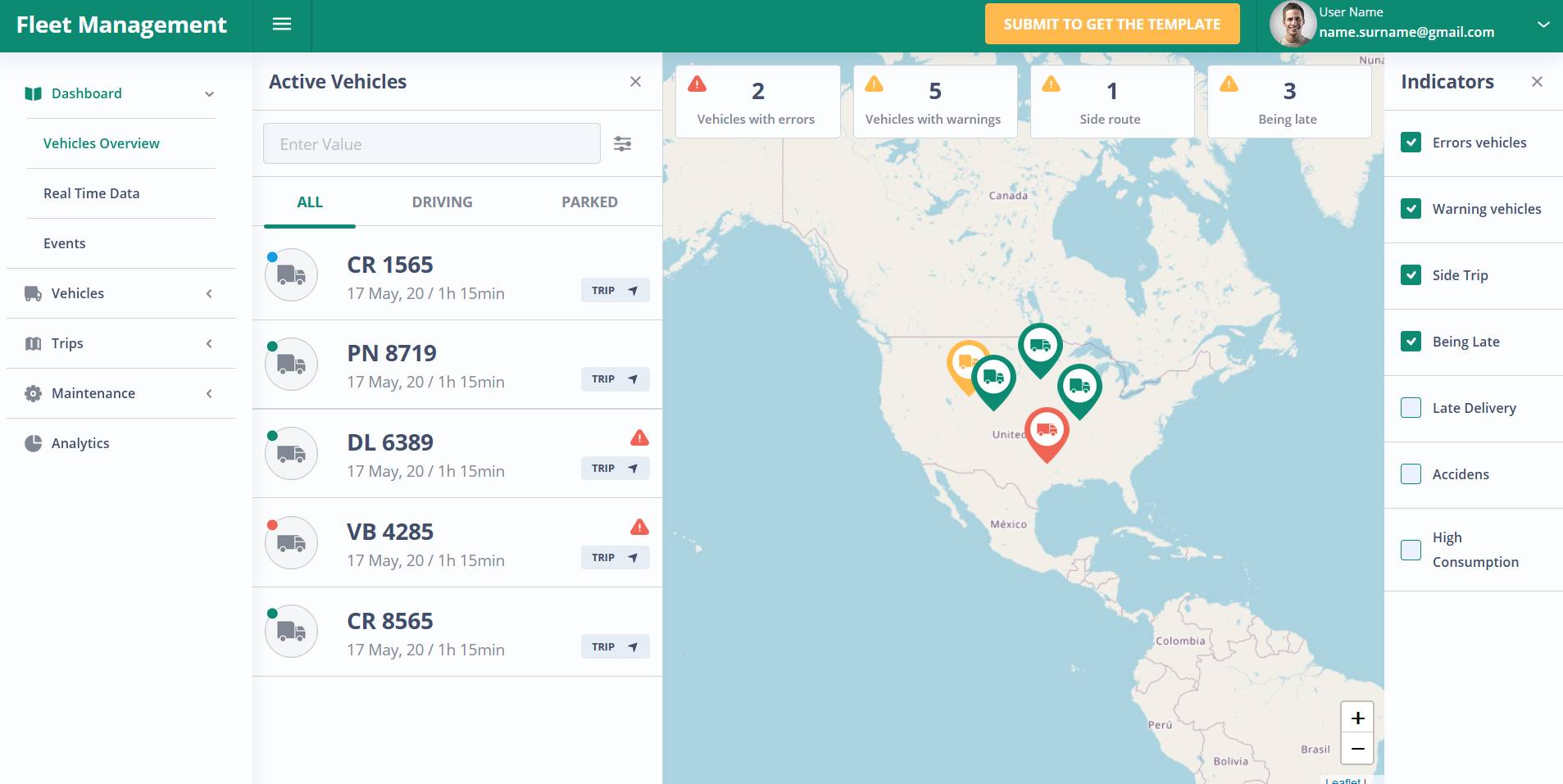
**Generating Data**

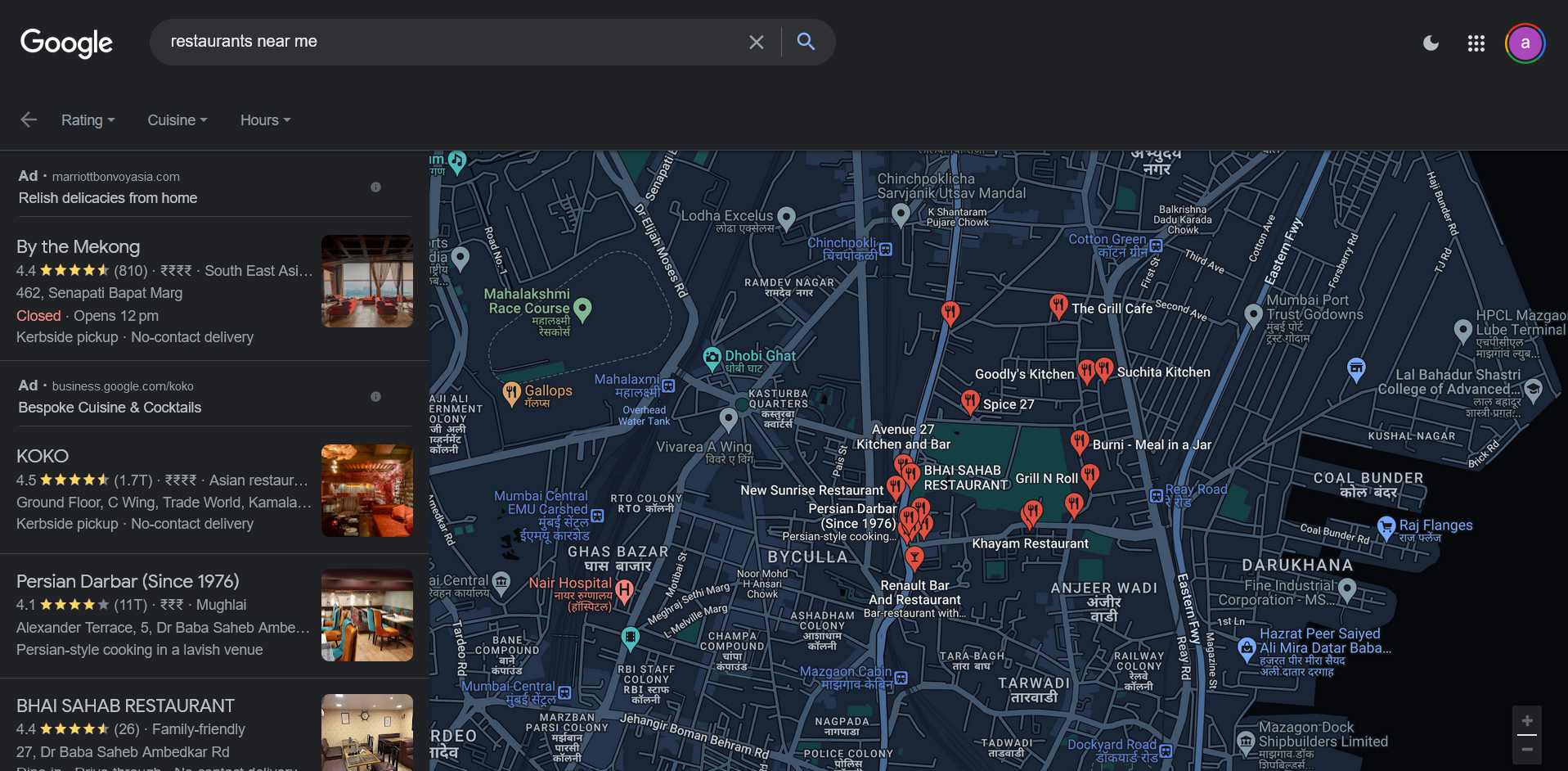
The problem suggests we have to cluster data which consists only of x,y coordinates.

The assumptions made here are:

1. It is a single page application, no extra pages.
2. Only clock locations are captured which are normalised to be between 0 and 1.
3. Since most analysis is about the buttons/ regions the user clicks. The data generated reflects user clicks for a buttons and relevant regions (like a map region if it is for tracking logistics)
4. 5% clicks are unintentional (noise)

Here are a few examples of the frontend templates used.



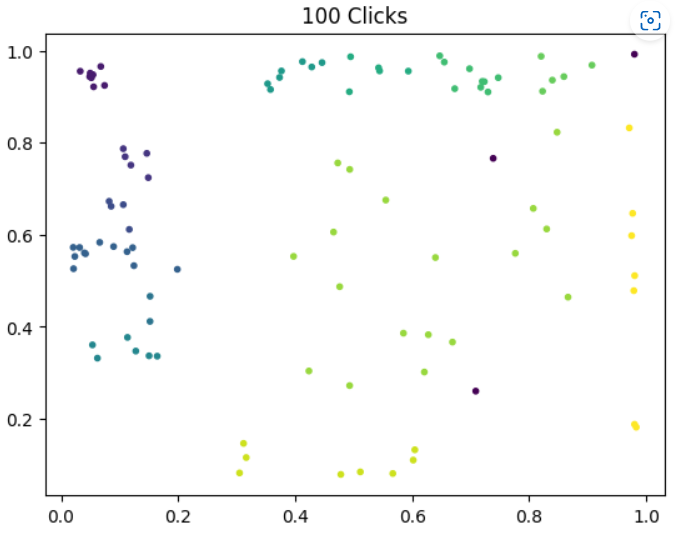


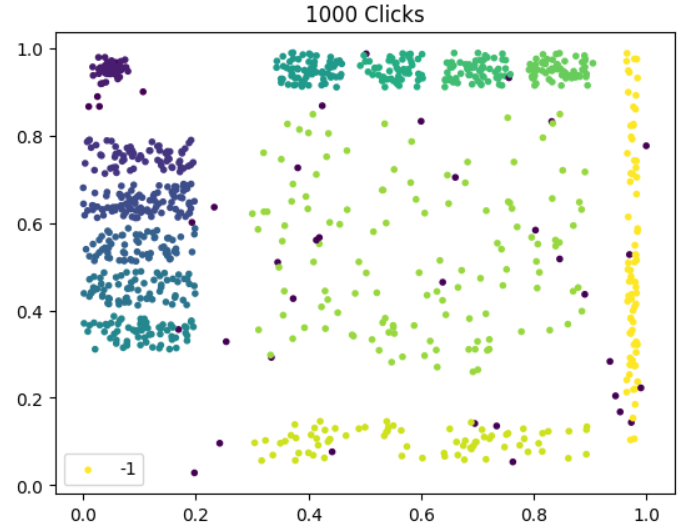
Built a ./data/generate\_data.py file which can be used to generate button clicks for a particular region.

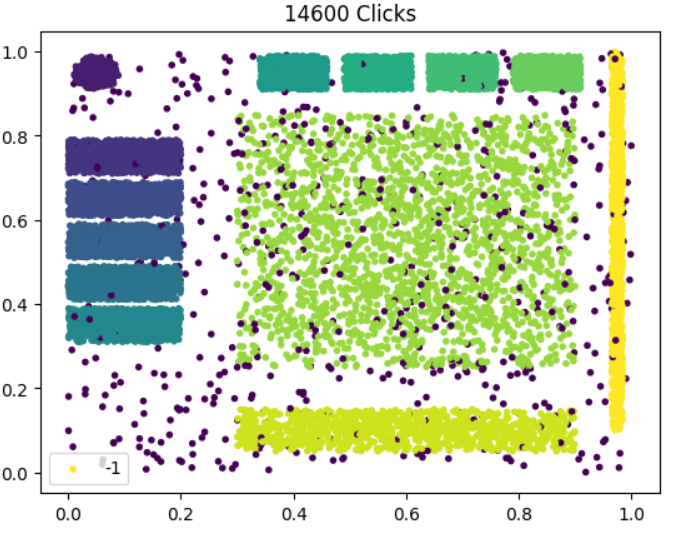
Generate\_data.py file can be used to emulate regions of clicks, rectangular or spherical along with the number and distribution (uniform of gaussian) of clicks for each region.

Then I wrote ./data/template\_data to build sample clicks for an application. Here is the sample template for clicks.

The region has 14 different regions, including a scroll, two large central regions, a spherical button, a top bar and a sidebar. It also contains 5% noisy clicks. Also note some buttons (eg: sidebar) are deliberately chosen close to one another as that emulates a real application layout. The clicks are chosen to be distributed normally in the region. Here are outputs for 100, 1000 and 14600 clicks:







Note: the visualisation can be found in both ./data/visualisation.in pub and ./data/modelling.ipynb notebooks

**Modelling Data**

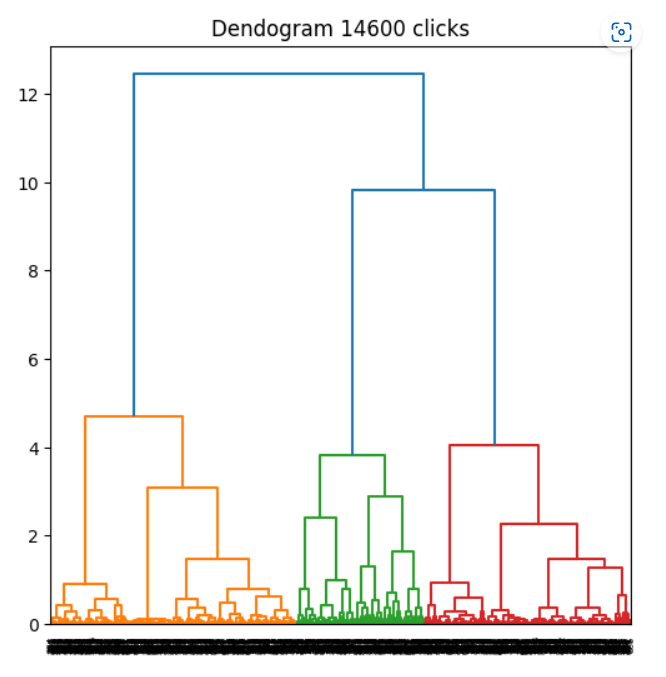
The next challenge is modelling data. The problem statement suggests clustering, so we will try a few different approaches.

Note: the approach should not just model the data well, but be suitable for different data sizes as well; from 100 clicks to 10000.

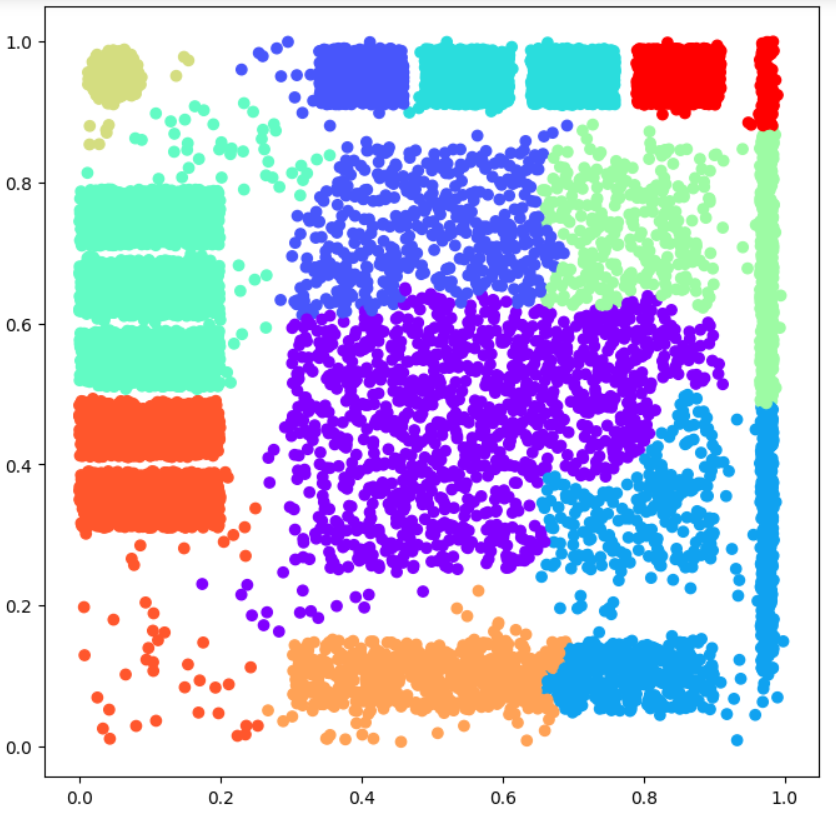
Distance metric is not ideal as the distribution of regions (and hence clicks) is sporadic, ruling out methods such as k-means.

Here we try two approaches DBSCAN and hierarchical.

For Hierarchical Clustering we plot the dendrogram with ward linkage and we find the ideal number of clusters as 10.

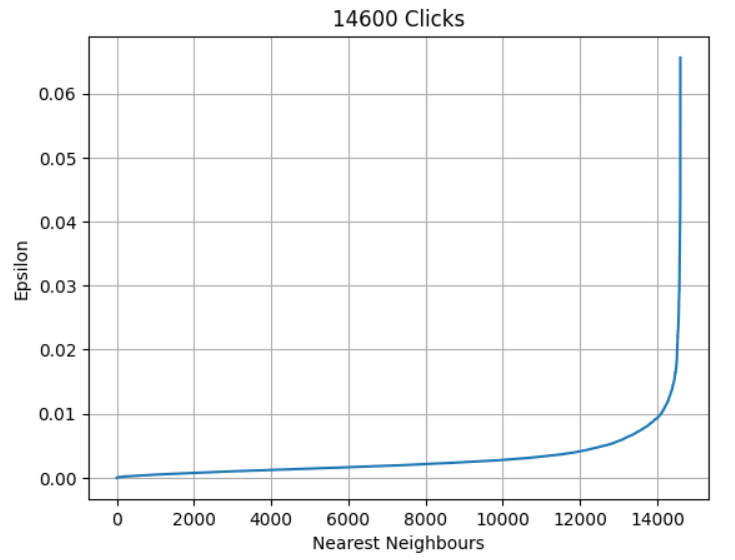


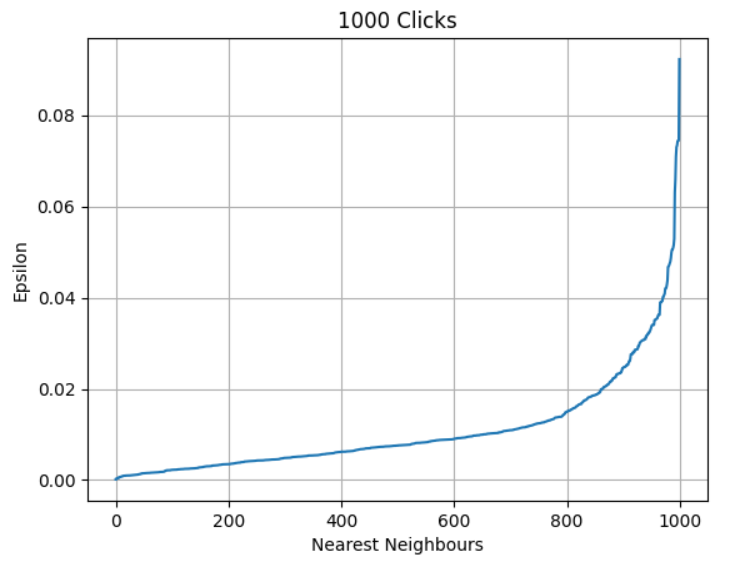
Building the model, we get the following results.

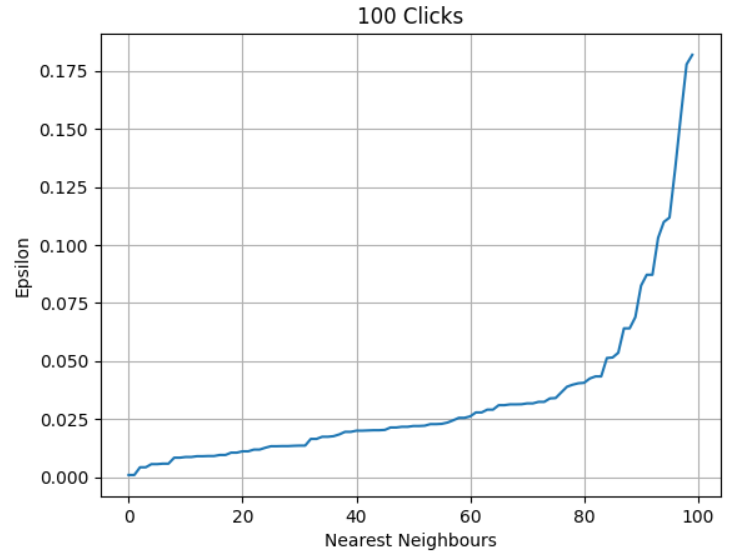


As we can see, the model performs poorly on the larger regions and the scroll, splitting them into multiple parts and merging them with other regions. This is the biggest issue with a global distance metric, as they are unable to handle non-normal structures that are close to one another.

For DBSCAN, choosing min\_neighbors to be 3 (<100 points) and 5 (>100 points). And plotting distance with nearest neighbours graphs (DBSCAN elbow method) for different data sizes. We get the following plots:



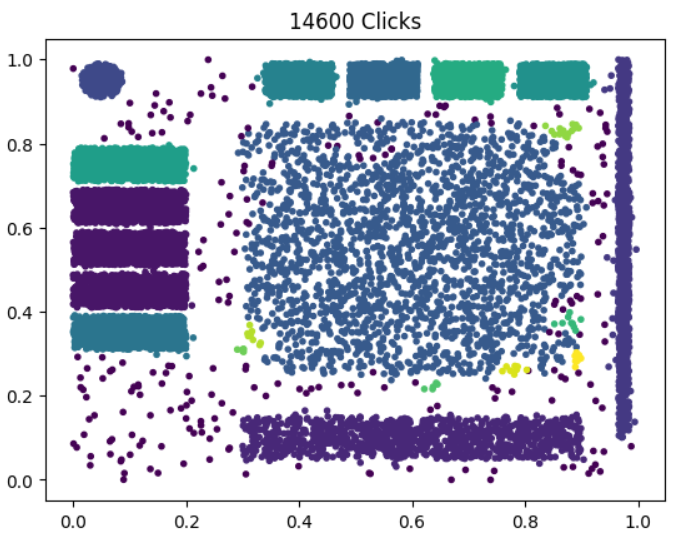


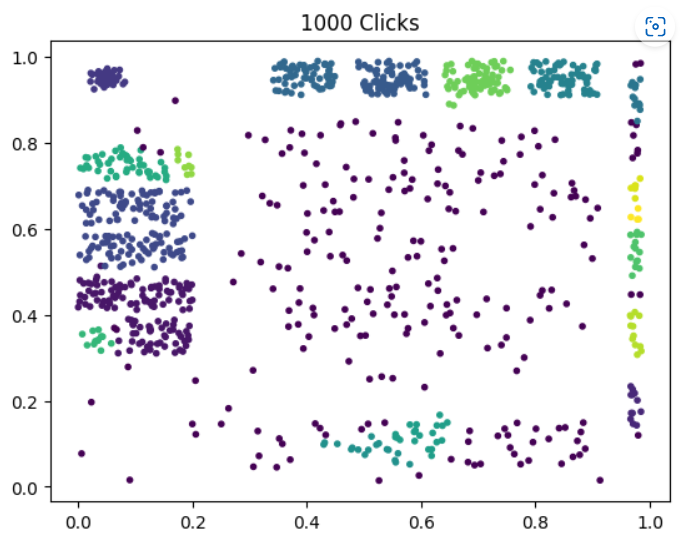


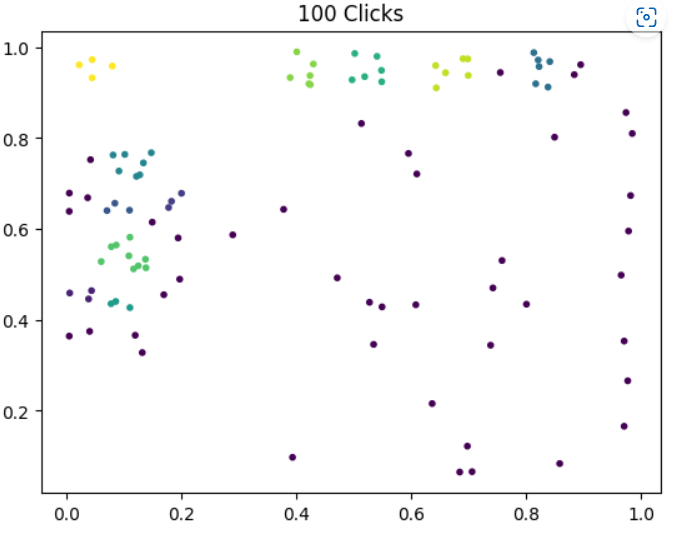
Based on this and check for different, here were the final parameters

| Number of Clicks | Epsilon | Number of Neighbours |
| --- | --- | --- |
| 100 | 0.04 | 3 |
| 1000 | 0.028 | 5 |
| 14600 | 0.019 | 5 |

The cluster outputs are:







As you can see, the model performs much better with sparse and dense data and with different region shapes and sizes.

The same set of heuristics will be used for deployment:

No classification below 40 points

# eps=0.4 for points between 40-500

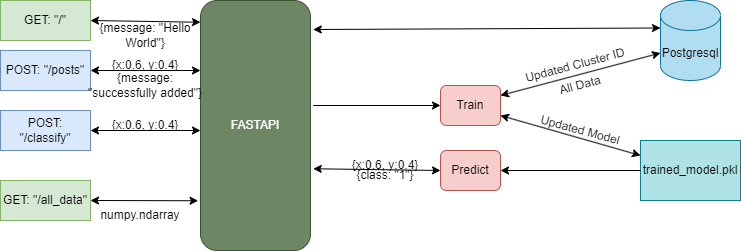
# eps=0.28 for points between 500-5000

# eps=0.19 for points between 5000-

# Train every time number of points double

**Deploying the Model**

Here is the block diagram for the system:



Here is the tachstack:

RestAPIs and Backend: Python FastAPI

Database: Postgresql, Sqlalchemy

Model training: Sklearn

Model Storage: Pickle

Though noSQL databases like mongodb and cassandra would be better for the system (given the changing schema, clicks now, events later, multi page applications, etc) and companies the provide application analytics (like clevertap) using noSQL databases like Cassandra, I have chosen Postgresql due to my limited experience with noSQL databases.

Also note the get “/” noot cal is only to check if the server is up and working and has no other utility.

Whenever POST “/posts”is called, two things happen:

1. Predict is called to get the clusterID
2. (x,y, clusterID) tuple is stored in postgresql
3. Train is called. In train:
   1. A check if done if data is less than 40 and if more, how much is the increase in data since last model training
   2. Since the method is DBSCAN, you need an increment in spatial density of data for the model to be retrained and tweaked. If data has doubled the model is retrained with suitable parameters, all new cluster\_ids are updates in postgresql and model is stored in a pickle file

You can run tests by running:

- python ./data/apptest.py

You can visualise appdata by running the Visualize\_current\_model.ipynb notebook