**Important Files in project:**

**datamapper.py -** Code for functions related to canonical ID

**lambda\_handler.py –** Just a high-level logic on how a lambda can be created in aws for deployment and create an auto mapper capable of working with minimal human intervention.

**Reputation\_assignment.ipynb –** Notebook containing how the solution was built brick by brick, which can answer the questions of user who wants to use data mapper.

**Assignment problem statement** – Word document containing problem statement

**Assignment Solution walkthrough and layout –** Word document to explain work done

**Dataset.csv –** The main dataset used for the entire exercise

**Assignment Conclusions:**

The data was consolidated basis on various factors like:

1. Dropping duplicates,
2. Consolidate based on combination of keys as primary keys.
3. Data removed as part of bad data cleaning and preprocessing.

After certain level of consolidations, the amount of data came down by **72%**

**Data consolidation journey -> 30000 to 8342 distinct dealers** (can still can be refined over time though)

The data can be further consolidated basis on different primary key combinations, but this was what is obtained on best effort for a day.

The raw\_data is mapped to canonical\_data on basis of text similarity technique.

The technique used here is **Fuzzywuzzy**.. but we can also implement some interesting techniques like cosine similarity etc. The mapping comes out to be promising based on tests done in notebook.

**Further scope for the work:**

* There are few missing cases of address lines and zips which can also be extracted using an external API like google geocoding

Which has complete information based on zipcode, firmname, address line etc. Can be added as a further improvement.

* The data mapper functionality and scope can be further increased to layout some stats on mappings done, number of not applicable cases and so on.
* A dash board can be created on a ML web framework like streamlit with the stats obtained to monitor the data frequently.
* Reports about edits done for the problem can be achieved by using levinshtine-distance about number of edits done and
* Although the clean function handles most of the edits, it is currently not coded to generate reports but it can be done easily by adding a new parameter to save the values before edits and post it to csv.
* The data quality of a data row is as of now a bit broad metric for me atleast with an idea to proceed so Icould not understand clearly but if it is based on a certain metrics, we can add it as a function in datamapper.py to compute the data quality per data entry.

**Example : metric is based on Null values?**

**Logic = (total nulls in row / total columns in row) \* 100**

**Gives data quality which can be added as a new column to computed dataset.**

**Functional Advantages achieved:**

* Update functions are also defined in data mapper to remap **not applicable** values (values which could not be mapped either due to improper data or similarity less than certain threshold), which can significantly reduce time for remapping. As need not bother about already mapped values, **making the whole project scalable**
* The canonical list is always appended with **new data only with new IDs** without disturbing the old list, so it is easy to maintain the mappings and update in frequently there by creating a master data **which can be used as a central lookup table**.
* The datamapper needs minimal change to be made as a class with pre defined class variables or a module which is portable,

an object of such a feature can directly be pickled and can be used for deployment purposes without any hassle.

* The predefined doc strings and parameters provided to functions make the module more easily understandable.

**Operational Instructions:**

**In Page below:**

**How does the assignment work?**

The dataset given is used in notebook to clean, preprocess and feature\_engineer the data.

Use pandas to refine, remove duplicates, eliminate overlappings and consolidate repetitive data as much as possible based on different combination of keys to form a single primary key.

The consolidated data is given an ID called **canonical\_id** which will become the primary key of the master data.

A text similarity technique is implemented on the base meta\_data of canonical list and raw\_data which are nothing but combination of keys which are capable of distinctly differentiating one dealer with another and a

**confidence score is setup** to determine how similar the meta\_data compares to master\_df for mapping.

The raw data is then mapped with corresponding canonical\_list in a separate key called mapped\_canonical\_id,

A JOIN BETWEEN **canonical\_id** and **mapped\_canonical\_id** will point raw data to its corresponding part in master data.

The confidence of mapping is given in **confidence column** of **raw dataset.**

**#Operation types that can be achieved**

**---------------------------------------------------------**

* UI driven json based webservice can be spun up and hosted in a Heroku based PaaS service and can be added with additional features like JSON API response, database addition etc, based on requirement.
* The API calculates and displays the confidence and corresponding canonical\_id for a random data input.
* The UI driven service can also be provided with options to update the canonical list.
* The datamapper can also be converted into a module and hosted in a container as a microservice solution for mapping new data and **updating the canonical list which is stored in a common protected environment in cloud or database.**

**---------------------------------------------------------**

***For any queries, can contact yuvaraj@outlook.in***

**- Yuvaraj**