Capstone Project 2 Final Report

Current Challenges for Data Enrichment

There are two challenges for current Data Enrichment workflow: (1). How to develop a supervised ML algorithm that is computationally efficient and highly accurate, so third-party data records can be accurately linked with internal data. Without an accurate algorithm, the business value and enterprise analytics built on top of the enriched data is questionable, and therefore cannot provide useful guidance for decision makers. (2). How to design a robust and cost-effective architecture to keep the process running on a fixed schedule, so the business information provided by various data sources are up-to-date and valuable.

In terms of challenge 1, customer data, no matter how detailed, is fundamentally a snapshot in time. Business reputation, service quality and ownership can alter over time. Accurate detection of such information will provide valuable insights for supply chain management and customer services. Given the possibility of all these changes, data enrichment processes need to run on a continuous basis with a quantitative supervised ML algorithm. There are several cloud database services already providing some primitive data enrichment services, e.g. AWS FindMatches, MongoDB Single View. MongoDB Single View is based on query language to filter and match records. While AWS FindMatches has the ML capability to develop a supervised ML model after rounds of human data labeling. However due to the size of the different databases need to be matched for some large business entities (where a single database may contain records up to 10 million), the computational time for such algorithm can be lengthy, and the relatively more expensive AWS Findmatches service is not the most cost-effective solution for such computational intensive task. This paper describes an in-house highly accurate supervised ML model that can be effectively deployed on the more cost-effective virtual machine (AWS EC2 or docker). More specifically, a two stage ML model is used for the first time for third-party data enrichment. After software prototyping and ML algorithm accuracy benchmark, the software application is saved as artifact files in AWS S3 bucket. AWS Code Pipeline is initiated to trigger the ML algorithm with a fixed schedule to generate up-to-date business information at a minimal cost. The final output is QC tested and delivered with AWS glue crawler. The final output of the application is directly served with Redshift Spectrum for further downstream consumption.

For ML algorithm, a two-stage cascade model is adopted to improve the match percentage rate between datasets, while saving unnecessary computational efforts by narrowing down the potential match pairs. The first stage model targets on records with same Zipcode and address number (not the entirely address including both address street and number). This assumption greatly reduced the number of potential match pairs being compared, and almost 85% of match records were generated from this stage. This approach significantly reduces the memory and computation costs associated with other matching algorithm, e.g. fuzzy matching with O(n) running cost. The second stage process including all unmatched records left from the first stage. Comparisons for similar records are performed for these only with same Zipcode(but different address number) at this stage, since the assumption is that records may contain unexpected human errors on address or business might recently move to a new physical location. The number of pairs for comparison significantly increased at this stage. However, this stage is necessary for matching any records that may contain human errors, since it improved the match percentage by another 15%. The XGBoost model is used for both stages of pairing processes. The model parameters and weights are adjusted for the best Roc-Auc accuracy at each stage. Key features extracted from data for XGBoost classifier model are: similarities between Name, address, address

number, telephone number and customer type classification. For string type features, Jarowrinkler distance is measured to assess similarities between these short phrases. For numerical features, gaussian distribution based algorithm is used to determine the similarities between these numbers.

The second challenge of capable of serving data in fast turnaround is addressed by fully utilizing the AWS Glue service to directly ingest the JSON documents produced from the ML algorithm as Parquet format, and then using dynamic schema interference of this service to iterate on the self-defined schema without the rigid definition process. This fast turnaround capability is vital for such data enrichment application. Because the downstream data analytics may iterate on the first schema frequently, depends on the analytic topic. Various third-party databases with different focus can be introduced for matching, and each comes with different key and data types. How to adapt the output schema in quick turnaround so that it can satisfy needs of different user case is always challenging. The data post processing and pipeline solution proposed in this can evolve the document-based output according to different business needs, and therefore significantly accelerate the data delivery speed for further analysis. Since this task is relatively less computationally intensive, more mature cloud service can be utilized to accelerate the data sharing process without incurring large cost. When different third-party data are used for data augmentation, the matching algorithm is paralleled to run on individual EC2 machine to accelerate computation. The in-house built algorithm goes through minimal variation for data ingestion from various data sources and can be easily scaled up for parallelization.

Description of the dataset

The dataset used for this project are from Kaggle website: One for Safeagraph, and one for Yelp. Some basic data cleansing is performed on both internal data and external data. First, lowercasing all text data, remove special characters from strings to reduce sparsity of embedded words, and strip empty spaces before and after strings. Extensive noise removal is also applied, including punctuation removal, domain specific keyword removal. Then NLP standardization methods are applied next, including stemming and Lemmatization. For this record comparison task, text information comes as short phrase, often in abbreviation format, so there is no need to remove stopping words. Text normalization is applied to address features, since many address data are noisy, and vary a lot based on peoples' spelling habits. A dictionary type mapping is used for cleansing non-standard spelling.

The unique difference between this task and other NLP taxonomy task is that the similarity scores will be calculated for both text information, and numerical features, such as zip-code, phone, address number. Without a comprehensive assessment with both text and numerical features, the accuracy of record matching will be sacrificed, due to limited information. The cleansing methods for numerical features are: special character removal, extract the first five digit of zip code for standardization, remove country code from phone number and reformat the phone number to normalize data. Remove domain specific information in front of address number to improve consistency.

Another practical observation is that there is very limited benefit to adopt text augmentation and embedding that are prevalent for deep learning NLP approach in this record matching task. This is because text information for records are noun types, and the string length associated with records is short.

After hand-crafted these features for each record, the similarities of different records can be calculated for numerical and string features respectively.

Data Cleansing

Steps of data cleansing of such NLP data are:

1. NLP data type specification: Choose string data type for string similarity calculation, numerical data type for numerical value similarity calculation.
2. Data normalization technique: including remove capital letters, remove unique characters, and remove country code for phone number. Strip the space before and after customer name, city, state and country strings.
3. For zip code, only compare the first 5 digits. Since the detailed last 4 digits associated with possible human errors, so the last 4 digits were removed.

Data Cleansing Rule

Description	Rule
Number of OPCOS for the source	Distinct count of the OpCo's
Number of OPCOS we have data available for	Total OpCo's – distinct count of OpCo's
Total number of unique customers for the available OPCOS	Distinct count of the customer and OpCo
Total Number of Logically Active Customers	Distinct count of the customer and OpCo for a customer who has had a sales transaction with Sysco in the last 12
Total Number of Invalid Address = a+b+c+d+e Total Valid Address = Total Address – Invalid Count	1. Replace all the special characters with an empty string (gsub("[]\\?!\"\'#$%&()\{\}+*/:;,._`|~\\[<=>@\\^-]", "", x) and convert everything to uppercase. 2. Count the number of records which have an empty or null address_1, address_2, state, city, zip and remove these records before the next step **[Count: a]** 3. Count the number of address_1, address_1 fields which are empty or null and remove these records before the next step **[Count: b]** 4. Count the number of records which have the state, city, and zip empty but have an address in the address fields and remove these records before the next step. **[Count: c]** 5. Count the number of records for which the address fields have only numbers and remove these records before the next step. **[Count: d]** 6. Count the records which have the following strings in them: ('CLOSED','INACTIVE','NOT ACTIVE','DO NOT','DONT USE', 'DONT USE', 'BAD DEBT', 'BAD DEPT', 'RESETUP OF ACCT', 'NOT VALID', 'INVALID', 'GROWTH PROGR', 'WRITE OFF', 'NEED PHONE','TEST', 'TEST ', 'PLACEHOLDER', 'UNKNOWN', 'DISCONTINUED', 'DUPLICATED', 'USE OTHER', 'EMPLOYEE', 'RETURN TO', 'SYSCO') **[Count: e]**
Total Number of Invalid Names = a+b+c+d Total Valid Names = Total Names – Invalid Count	1. Replace all the special characters with an empty string (gsub("[]\\?!\"\'#$%&()\{\}+*/:;,._`|~\\[<=>@\\^-]", "", x) and convert everything to uppercase. 2. Count the number of records which have an empty or null address_ and remove these records before the next step **[Count: a]** 3. Count the number of records for which the names are only numbers and remove these records before the next step. **[Count: b]** 4. Count the records which have the following strings in them: ('CLOSED','INACTIVE','NOT ACTIVE','DO NOT','DONT USE', 'DONT USE', 'BAD DEBT', 'BAD DEPT', 'RESETUP OF ACCT',

	'NOT VALID', 'INVALID', 'GROWTH PROGR', 'WRITE OFF', 'NEED PHONE','TEST', 'TEST ', 'PLACEHOLDER', 'UNKNOWN', 'DISCONTINUED', 'DUPLICATED', 'USE OTHER', 'EMPLOYEE', 'RETURN TO', 'SYSCO') **[Count: c]** 5. For **SSMG**, records which have an acc_typ_cd = EMPLOYEE and cust_id like 'e%' **[Count: d]**
Total Number of Valid Records for Match	Count the number of records after the intersection of all the records which have both valid names and valid address.
Logically Active	The customers who have had a transcation with Sysco in the past 12 months.

Data Exploration:

The proposed software architecture is shown in the figure 1:

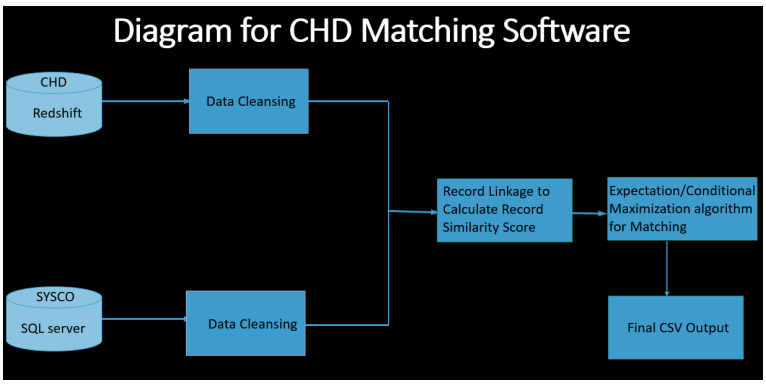


Figure 1: SVOC Software architecture

After ingested data from two relational databases, the data are not labeled with matched or unmatched. To develop a data science model with supervised ML algorithm and rigorous threshold, the first step is to use Unsupervised learning to generate labeled data. To accelerate this process, a unsupervised ML algorithm is used in this step. For this step, the Expectation/conditional maximization algorithm is used. **Expectation-Maximization algorithm** can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.

The procedures of the EM algorithm are:

1. Given a set of incomplete data, consider a set of starting parameters.
2. Expectation step (E – step): Using the observed available data of the dataset, estimate (guess) the values of the missing data.
3. Maximization step (M – step): Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.

	CHD	SYSCO
No. of Records	1,910,120	307,574
Ratio Comparison	6.3	1

- CHD is a much bigger dataset
- Other external datasets can be a lot bigger than CHD

Fig. 2: Initial Data size exploration

Data size comparison showed CHD data is a lot bigger than the internal dataset, and therefore it is important to parallel the data science model and use the memory efficiently.

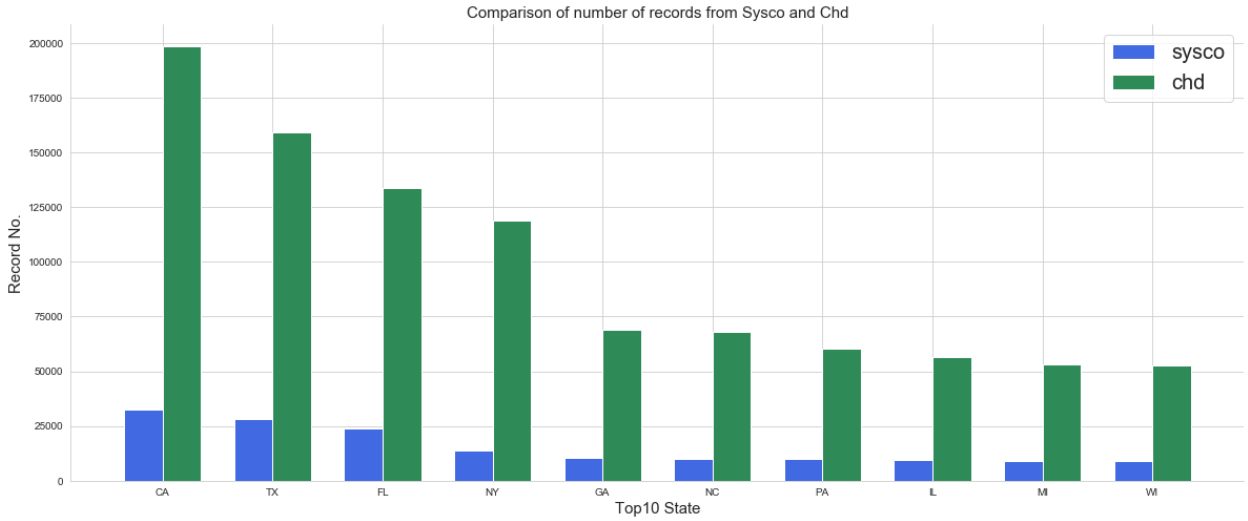


Fig. 3: Initial Data count.

Figure 3 showed that the data distribution in different states. It can be seen that the large cosmopolitan contains 5 times more data compared with less populated states.

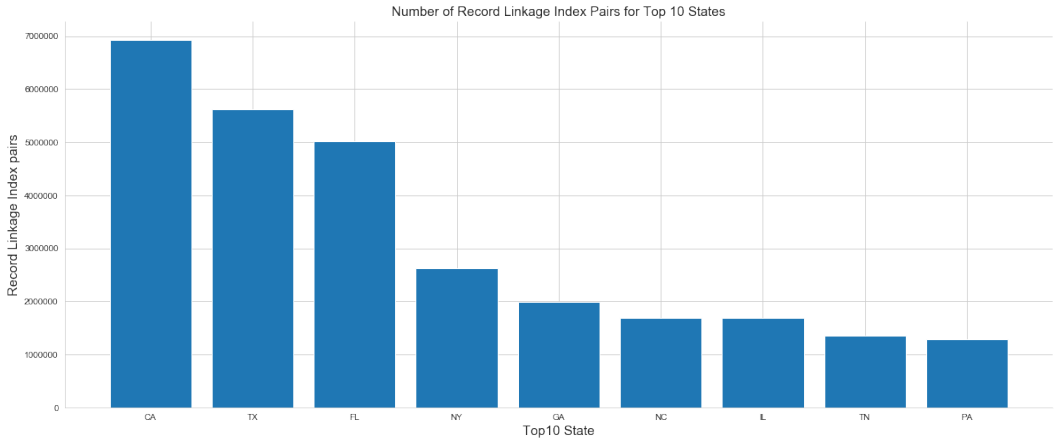


Figure 4: Total Number of Index Pairs generated from RecordLinkage for Calculation

Figure 4 shows total number of index pairs needs to be calculated from all 51 states: 53,541,044, and top 10 states are counted for 60% of the index pairs: 29,561,880. This amount of index pairs are generated by using blocking conditions: Only records with the same city and zip code will be paired for similarity comparison. Even with this blocking condition, there are large number of index pairs need to be calculated. This presents a significant challenge for computational efficiency.

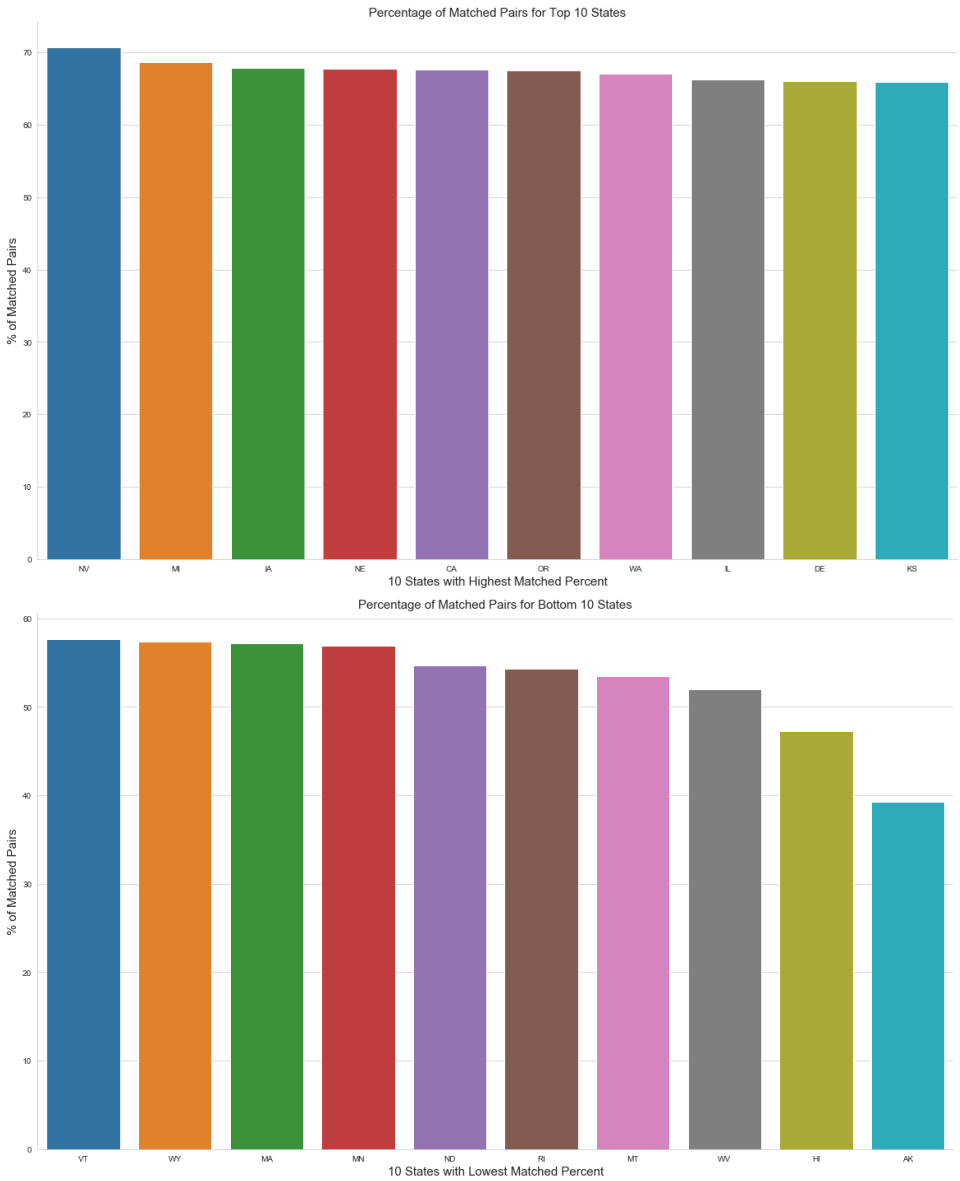
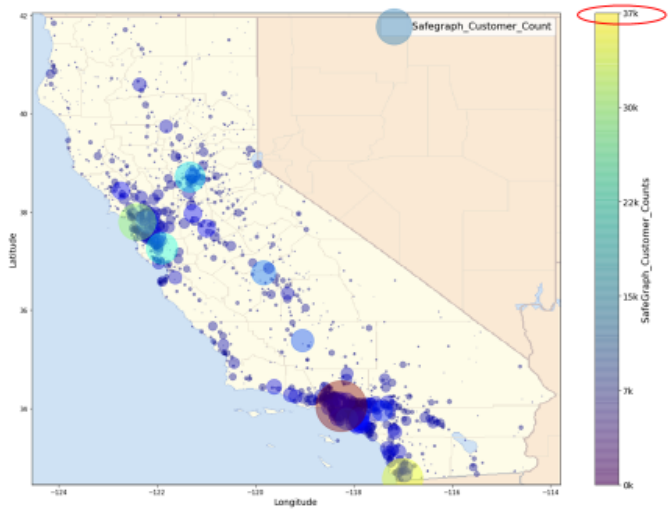


Figure 5: Matched percentage counts for top 10 states and bottom 10 states.

After applying the unsupervised algorithm, the number of matched percentage counts were compared. Filling rate from RecordLinkage is good, top ten states have match rates bigger than 65%. Bottom ten states have match rates bigger than 50%, except AK.

Safegraph Record Number Counts Distribution in CA



Summary:

Algorithm Analysis Performed by using python version of RecordLinkage.

The results by using unsupervised ML algorithm are impressive, and either library can be used for benchmark purpose due to the unsupervised learning nature.

Initial research was performed to implement the RecordLinkage on Spark platform: Strategic move for future external data consumption.

To improve the performance, the data need to be labeled as true match or false match, and apply supervised learning algorithm to make the algorithm more accurate.

Current Unsupervised Algorithm is good for providing fast initial result

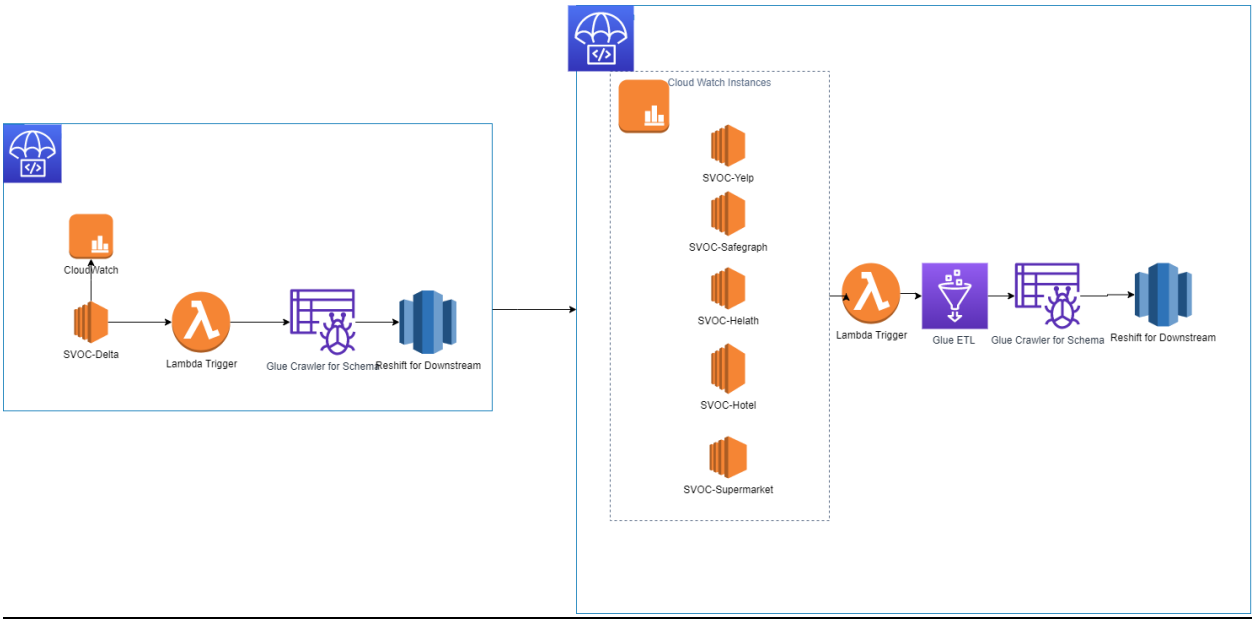
However, there is a major pitfall for this method: Lack of Rigoristic Measurement for Accuracy.

Because Unsupervised learning does not have labeled targets

This create obstacles for further algorithm optimization (No way to compare effectiveness of algorithms)

Initial study showed that machine learning is not one size for all: optimum thresholds are different for different states (e.g. NY and LV need higher thresholds, because many shops sharing same address in one plaza) .

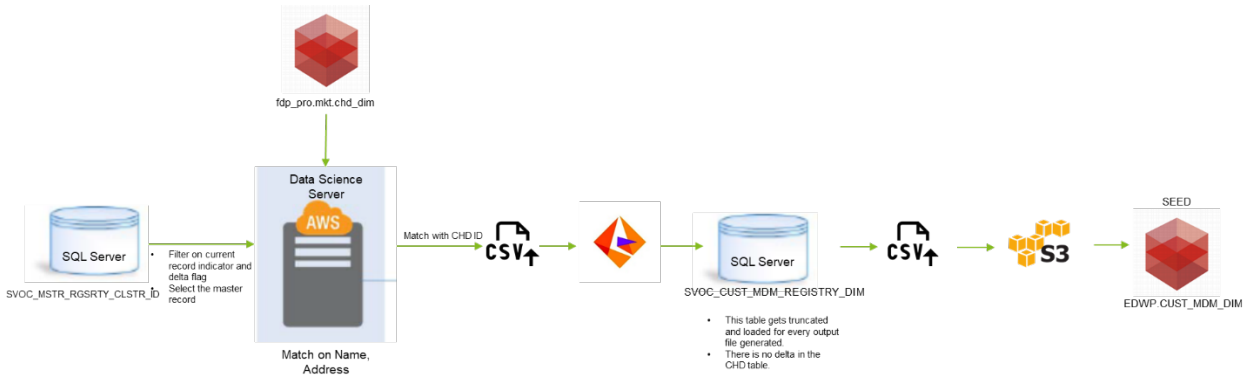
Flow Diagram for Third-party Enrichment Process:



The above flow has the following steps:

1. Sourcing & cleansing of USBL, SSMG, and Fresh Point data for logically active customers (which were either created or had a transaction in last 12 months). The cleansing rules have been attached in the cleansing section.
2. Handling delta for the following 4 scenarios:
    a. New customer
    b. Ownership change (Change in name and not in address)
    c. Location change (Change in address and not in name)
    d. Logically Inactive records
3. Perform in file deduplication clustering for all the records in the first load and then for the delta records in the subsequent loads based on Name and Address fields
4. Load the output to a staging table and identify the master record for each cluster following the below rule:
    a. Sort based on source in the following order USBL, Fresh Point, SSMG
    b. Sort on customer names within each source
    c. Sort on concat Id within each sorted name in a source
5. Use the master record of each cluster to perform cross match with all third party data, including safegraph, CHD, and Yelp. To accelerate the calculation, each match is run on an individual EC2 instance.
6. Populate the relevant third party enrichment data for the respective cluster IDs after the CHD match
7. Load the SQL and SEED table with the final data.

<u>Third party data Matching & Master Registry</u>



Once the view is populated along with the Record Rank from the SVOC process, we trigger the third party matching algorithm with a scheduled AWS lambda function. To achieve parallel computing, the same matching data science model is loaded on 4 individual EC2 instances. Each individual EC2 instance will ingest data from different databases (Safegraph, Yelp, CHD and Retail store). Once the four third party match are completed, the output file is placed in a temporary folder in server. This location is mounted on Informatica Server and we use the file as a source for our ETL process to generate Master Registry table in SQL Server.

Matching with Python Algorithm:

The matchingMainCHD.py uses Record Linkage (BigDataLinkage) method from RecordLingake library to identify if the master record from the view matches with the CHD data provided by a 3rd party within each state. Within each state and city, it searches for similar records based on the string match for customer name, customer street address, customer segmentation and phone number. The algorithm matches either the internal data or third party data with an appropriate third party record based on the threshold value in the configuration file. For all the records which did not get populated with a match data, the match columns are populated as NULL. The match data is used to enhance the internal data for customer insights. There are different user cases for the matching output, e.g. lead to prospect potential, customer basic metric, etc. The final output of the algorithm is a csv file (SVOC_MSTR_RGSRTY_CLSTR_ID_CHD.csv.) in the ADM server.

To improve the accuracy of the original fuzzy-match based algorithm, I implemented Supervised Machine Learning Model instead of Fuzzy Matching: Consistent measures of matching probability. After the unsupervised labeling process as described in the first report, human supervised data label was performed. This process removed obvious mistakes of data records mismatch. After this ground-truth label process, different data science models are applied to determine the best model for this dataset. As shown in Figure 1, ensemble method, XGBoost achieved the best AUC accuracy among other methods, and therefore is chosen as the final model.

To accelerate the computational efficiency, a Two Stage Cascade Model is proposed in this project: The first stage is matching blocked records with same zip code and address

number. The ML on these records have a more relaxed division point threshold for tree based models. (XGBoost). Second Stage are for all records having the same zip code. More feature engineering for this stage.

One development for feature engineering: Break down Record Linkage library, and fine tune parameters from Record linkage to generate more accurate similarity comparisons for "String" type features(NLP language), and "Numerics" features. The new data science model also contains extensive Feature Engineering for Attributes-based matching: Comprehensive consideration of various types of evidence for matching. Besides the original attributes, customer name, address, other features are included in the final DS model. These features, e.g. customer segmentation and phone number are also added. From the feature selection chart shown in Figure 2, these two features also shown great importance in the final model.

To effectively accelerate data model development speed, the third party library is deployed for the index-pair generation between Internal and External data: Solved the memory and computation efforts of Fuzzy Match. It is worth to notice that only the index-pair part of the record linkage library is used in this data science model. Other parts of the unsupervised classifiers from RecordLinkage library are not used. The reason for this is that the unsupervised classifier cannot establish a rigorous threshold, and will result in poor record matching results.

Finally, there are other possible products on the market, however such products also require many rounds of data labeling, and will result in high cost for building a customaries machine learning model. In house software that can be run economically on a serverless virtual machine: AWS FindMatches is expensive and use the similar algorithm.

A detailed comparison of the original unsupervised ML algorithm and supervised ML algorithm is shown in Figure 3. It can be seen that the matching percentage has been improved by 7% after implemented the supervised ML algorithm.

Use Record Linkage to calculate String/Numeric Similarity scores within blocked indices between two databases:

Index Pairs	State	City	Zip	Address Name	Address N
(Record 1, Record 2)	1	1	1	0.9	1
(Record 1, Record 3)	1	1	1	0.8	0.4
(Record 2, Record 3)	1	1	1	0.85	0.4

- Total number of index pairs needs to be calculated from all 51 states: 53,541,044, and top 10 states are counted for 60% of the index pairs: 29,561,880

- ❑ To reduce the amount of computation, we confine the comparison for records with the same State and City.
- ❑ After String similarities are calculated, unsupervised classification is performed to cluster results accordingly.

Different machine learning models are applied to the data, and their AUC scores are used to benchmark their performanaces:

	Method	Best_Score
0	XGBoost	0.996654
4	SGDClassifier	0.996460
5	ExtraTreesClassifier	0.996448
6	SVC	0.996434
2	LogisticRegression	0.996395
3	AdaBoostClassifier	0.996209
7	KNeighborsClassifier	0.992302
1	RandomForest	0.980872

Figure 1: Comparison of different DS models.

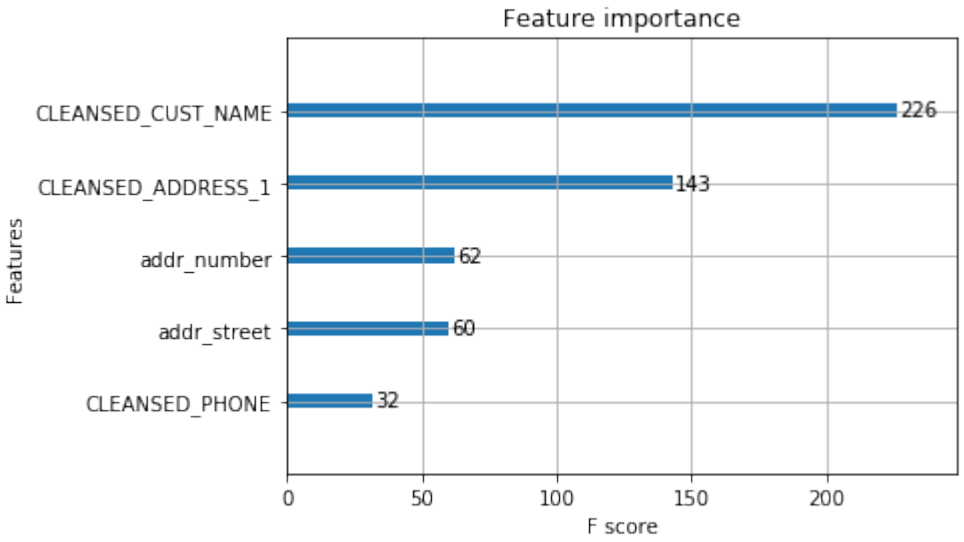


Figure 2: Feature importance comparison.

```
blockList = ['CLEANSED_STATE','CLEANSED_CITY','CLEANSED_ZIP']
```

```
### **************************Start for RecordLinkage Matching*****************************************************
#create block linkage index to confine the search
indexer = recordlinkage.Index()
indexer.block(blockList)

candidate_links = indexer.index(Data1_df, Data2_df)
```

```
candidate_links
```

```
MultiIndex([(    0,    580),
            (    0,   2301),
            (    0,   2703),
            (    0,   5052),
            (    0,   9687),
            (    0,  10255),
            (    0,  10798),
            (    0,  13603),
            (    0,  17795),
            (    0,  23898),
            ...
            (32996, 184220),
            (32996, 209503),
            (32996, 239173),
            (32996, 264150),
            (32996, 268233),
            (32996, 276337),
            (32996, 328345),
            (32996, 346545),
            (32996, 368245),
            (32996, 403567)],
           length=19811663)
```

	ML Algorithm	Unsupervised Algorithm	Difference
Total Matched Count	242011	223747	18264
Total Count Pct for Sysco Data	70%	64%	6%

Figure 3: A comparison of Supervised and Unsupervised ML algorithms.

Moving data to SEED

Linux based shell script that uses the 'AWS CONFIGURE' command to connect to the S3 bucket where the SVOC file (REDSHIFT_SVOC_MSTR_RGSRTY_CLSTR_ID_CHD.txt) will be placed. Once the file has been moved from the INFA source location (/mnt/csv2/), it will then be moved to the S3 bucket (//OpCo Data/MDM/). Upon landing to the S3 bucket, the Linux script will connect to the Redshift server using the SVC_MDM account and truncate the existing CUST_MDM_DIM table. Once the table has been truncated, a AWS COPY command will be executed loading the file from S3:

copy edwp.cust_mdm_dim_bkp (mdm_src_sys_cd, cust_concat_id, co_skey, cust_skey, co_nbr, cust_nbr, cust_shipto_nbr, src_cust_nm, src_addr_line_1_txt, src_addr_line_2_txt, src_addr_line_3_txt, src_cty_nm, src_stt_nm, src_zip_cd, src_cntry_nm, sysco_phone, cln_cust_nm, cln_addr_line_1_txt, cln_addr_line_2_txt, cln_addr_line_3_txt, cln_cty_nm, cln_stt_cd, cln_zip_cd, cln_cntry_cd, mdm_cust_clstr_id, clstr_or_isl_ind, clstr_typ_cd,clstr_src_cnt, clstr_lbl_nm, clstr_cnfdnc_scor_rng, chd_id, chain_id, location_pid, owner_pid, chd_name, chd_address, chd_city, chd_state, chd_zip, chd_menu_type, chd_operation_type, chd_market_sgmnt, chd_customer_type, src_sys_cd, mstr_rec_rank) from 's3:// /OpcoData/MDM/REDSHIFT_SVOC_MSTR_RGSRTY_CLSTR_ID_CHD.txt'

Marketing Analytics

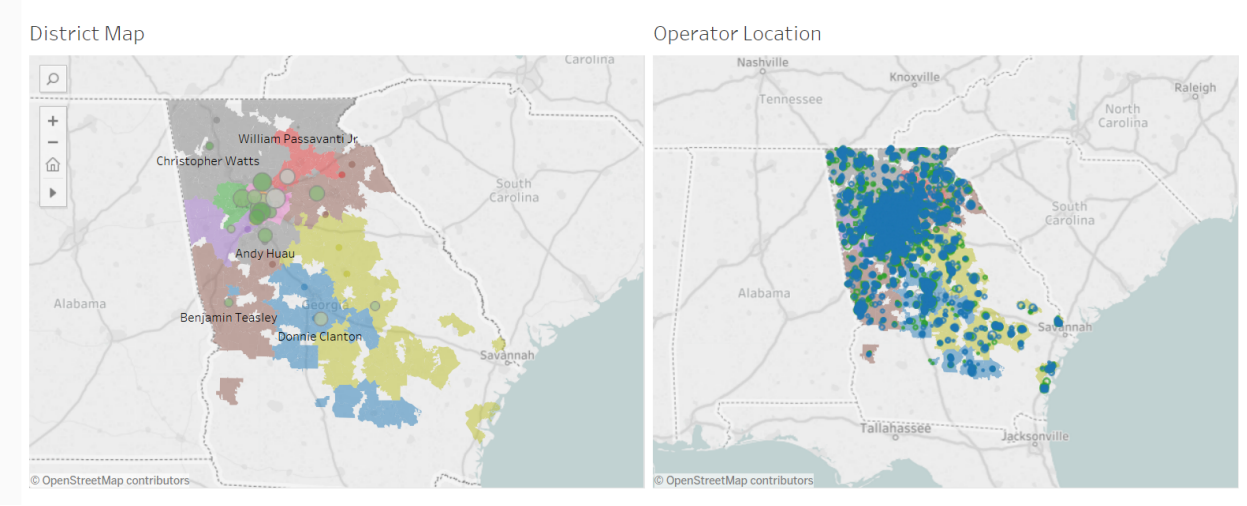
The Marketing Analytics Team is the consumer of the data in SEED. Marketing Analytics uses this data to identify the market potential of the customers Sysco is dealing with and identifies the market potential of the customer Sysco needs to engage in a business with.

The inputs to the marketing analytics team are the customer master data, which has the clustered data across the 3 lines of business, and sales transactional data, which helps identify the market potential for business.

SQL Query used for aggregating the sales data:

MAQuery.txt

Screenshot of the Marketing Analytics Dashboard:



AWS Data Pipeline and Lambda

BI SEED team requires that all processes be schedule via a AWS Lambda using CRON job to kick off the process every week at 0100 hrs. on Thursday nights. The AWS Lambda is written in Python 2.7 and looks to identify the SVOC Data Pipeline and the status of the Data Pipeline and whether any previous Data Pipelines are still running. If the previous SVOC pipeline has finished and the Pre-Day close has finished, the Lambda will call the SVOC_Pipeline and write a record to the Aurora Tracking table that the pipeline has started

The SVOC data pipeline once started via the Lambda, calls the workflow () and waits for the workflow to complete. Once the workflow is complete, the data pipeline will update the Aurora Tracking table is complete. If there is an error in completing the workflow, a log with update showing the workflow and pipeline failed.

1. Upload JSON file for Pipeline
    a. Please name the pipeline “SVOC_MDM_DIM”
    b.

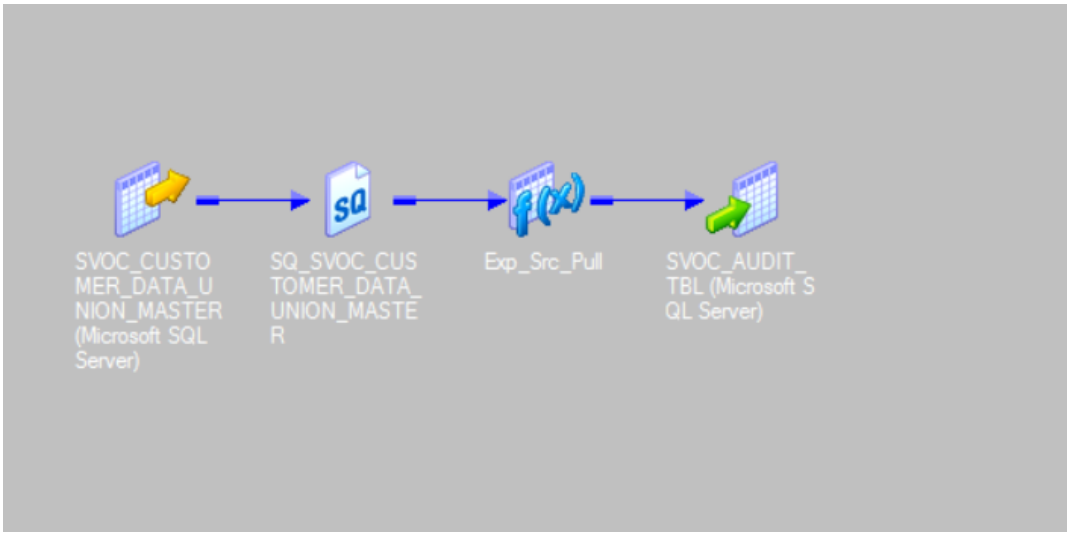
2. Upload Lambda Zip File
    a. Please name the pipeline “SVOC_MDM_DIM”
    b. 

3. Create CloudWatch Event
    a. Add CRON job for Bi-weekly at 8:00 AM CST (13:00 GST)
    b. 0 0 13 * * WED

SVOC Process Statistics Report

A mapping has been created to pull all the statistics and load in the audit table, details below. This process will run at the end of SVOC workflow and populate the statistics in the Audit Table. A date field has been added to enable any trend analysis in future.

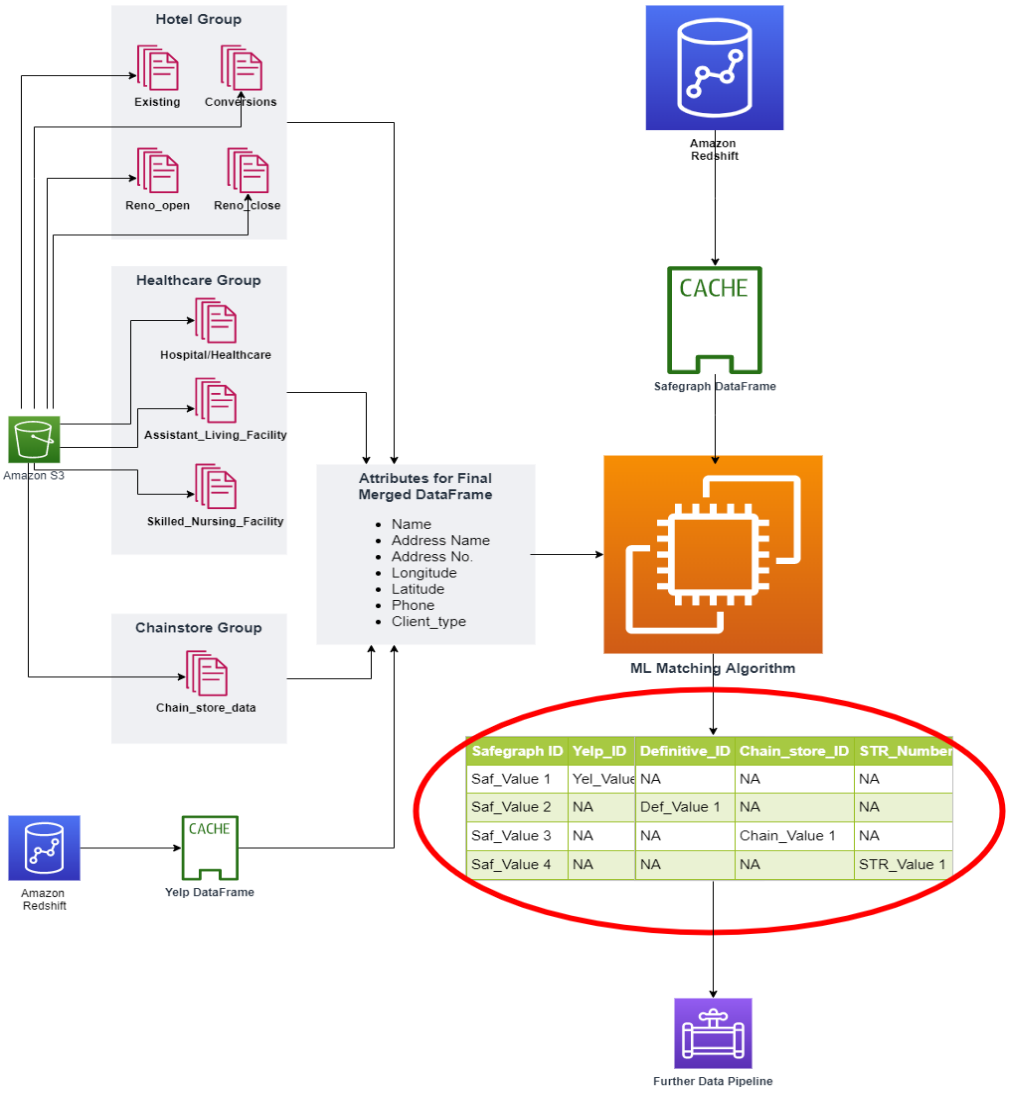
Mapping Screenshot:



Screenshot of the output in Audit Table:

Third-party data ETL methodology:

1. Concat all four different types of operators into one dataset. This operation is based on the mutual independent characteristic of these operator types.
2. Extract common features from both the concatenated dataset and Safegraph data for machine learning. These features are: (1)Customer name, (2) address name, (3) address number, (4) longitude, (5) latitude, (6) phone, (7) customer category.
3. Output a final table where each safegraph ID is matched with one unique record from the concatenated dataset. The output table key structure is like this (Such key structure is designed for easy linking different types of operators):



AWS Glue will play a major role in creating ETL job after the ML algorithm, fast Data Catalog, and schema refinement.

Why AWS Glue:

- Fast Data Catalog especially suitable for ad-hoc table creation and big query: Glue Grawler has classifer logic to infer the schema, format and datatype.
- Further ETL job is authored to refine table structure by using the metadata generated from Cralwer, e.g. columns structures, datatypes. ETL jobs are flexible to process multiple third-party data, table add-on or remove.
- Data stay in Sync with ML in a cost-effective way
- On top of AWS Glue, our architect will use AWS Cloudwatch as major event triggers (Glue job schedule, monitor job logs and retries) and monitoring tool.

- Cost Analysis with GloudWatch: Gathers runtime metrics to monitor and optimize the Glue job cost.

Summary and Future direction:

- Algorithm Analysis Performed respectively on Python and R version of RecordLinkage
- Initial research was performed to implement the RecordLinkage on Spark platform: Strategic move for future external data consumption
- To improve the performance, apply supervised learning algorithm
- **Other improvements:** (1) ML algorithm provides more accurate probability statistics for matching pairs, and therefore only the record with highest probability is chosen. This effectively solved the multiple similar/repetitive third party records matching to one Sysco record.
- (2) Matching statistics is included, and will aid Data Steward for further check.

Future Plan: Considering the high accuracy of the ML model (99%), and the consistent matching percentage across different state (all between 60 to 70%). It is difficult to improve the matching pct by simply using a different ML algorithm. It is most likely due to data quality.

(1) It is important to look into the unmatched records to find out possible reasons for unmatching. Very likely due to wrong address: such as “key drop” or no address at all, etc.

(2) A few key word cleansing can be implemented for customer name: ARA, SODEXO, etc.