Chapter 1

Normalization of Company Names for Data Management Company

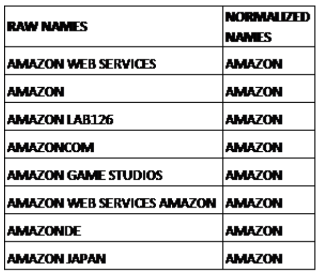
1.1 Problem statement

B2B organisations faces the common challenge of data normalization with respect to the names of the registered entities. For example, Service providers who have several customers and wants to determine their biggest customers and their related information, they often face a challenging task of normalization the business entity names and map them to their parent or legal entities.So the task is to get Standard Name given an unclean/messy list of Company Names.

1.2 Data Set Overview

The original raw data could have several challenges such as:

- Different/Non-standard legal entities in name, department names in addition to organization name.
- Organization names abbreviated
- Spelling errors
- Country/region names present in addition to organization name
- Email ids provided instead of organization name
- Non-English characters used.
- Subsidiary name which may not be mapped to parent organization name



RAW NAMES	NORMALIZED NAMES
AMAZON WEB SERVICES	AMAZON
AMAZON	AMAZON
AMAZON LAB126	AMAZON
AMAZONCOM	AMAZON
AMAZON GAME STUDIOS	AMAZON
AMAZON WEB SERVICES AMAZON	AMAZON
AMAZONDE	AMAZON
AMAZON JAPAN	AMAZON

Figure 1.1: Sample Input and Output

1.3 High Level Design/Architecture

A three step solution/approach was developed for this problem which can be further scaled to accommodate larger data requirements as and when required or requested. The Three Step Process:

- Cleaning the initial list of names by removing any special or non ASCII characters that exist in the names.
- The entire cleaned list of names is then passed to an information retrieval process to get a list of curated stop words.
- The list is again cleaned now, by removing the stop words and then passed on to the main algorithm to get standard name and match every clean name to its specific standard name.

All the steps will be discussed in detail in further sections of this chapter.

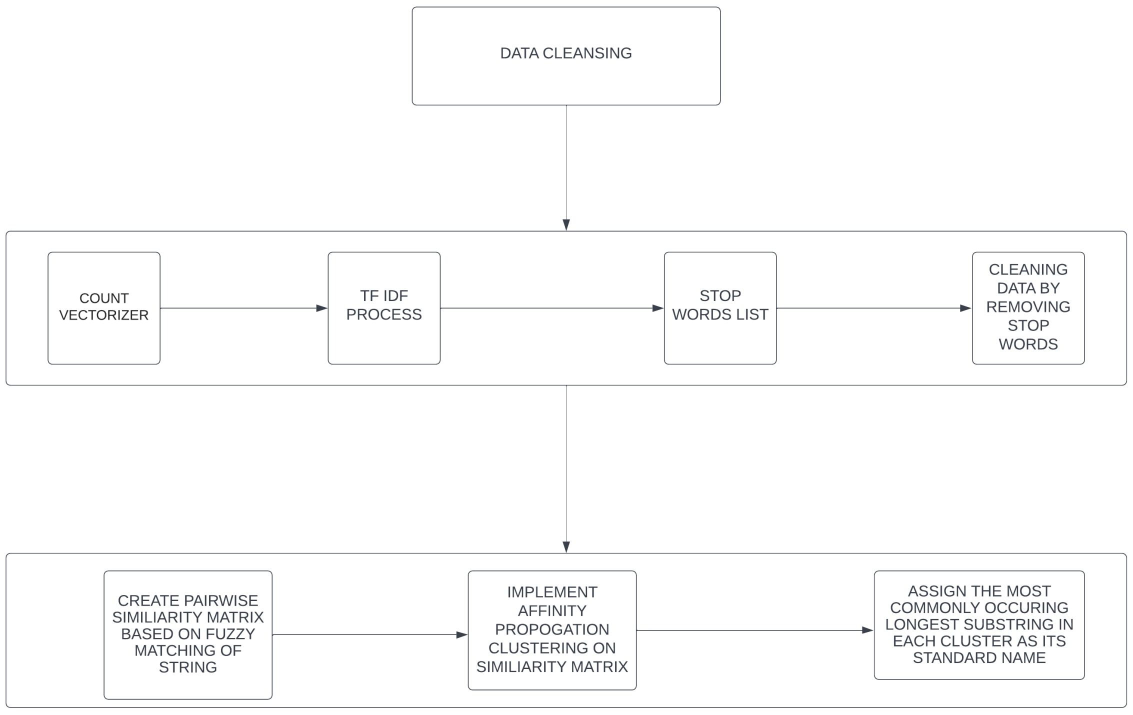
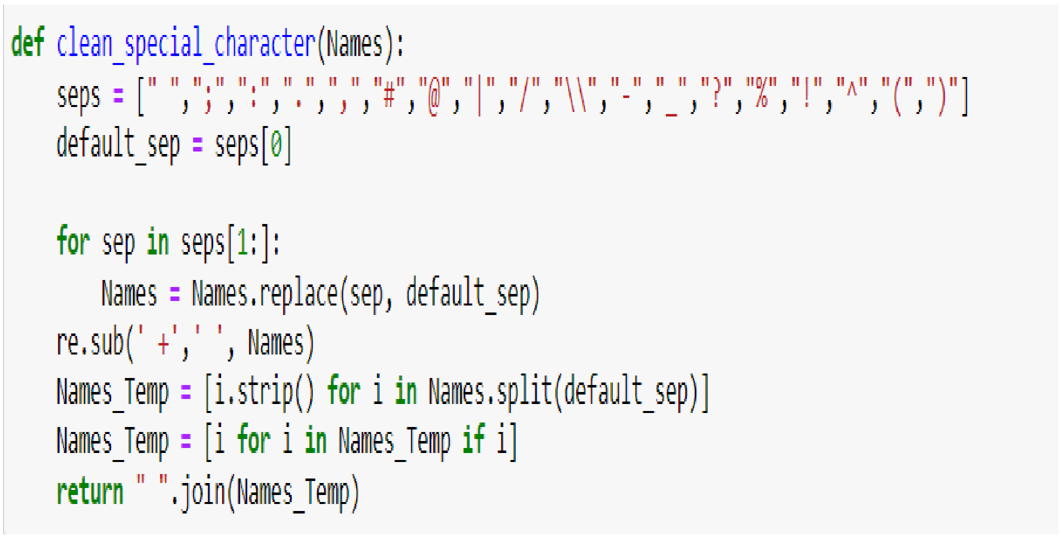

DATA CLEANSING
COUNT VECTORIZER
TF IDF PROCESS
STOP WORDS LIST
CLEANING DATA BY REMOVING STOP WORDS
CREATE PAIRWISE SIMILIARITY MATRIX BASED ON FUZZY MATCHING OF STRING
IMPLEMENT AFFINITY PROPOGATION CLUSTERING ON SIMILIARITY MATRIX
ASSIGN THE MOST COMMONLY OCCURING LONGEST SUBSTRING IN EACH CLUSTER AS ITS STANDARD NAME

Figure 1.2: Architecture Design

1.4 Step 1: Data Cleansing

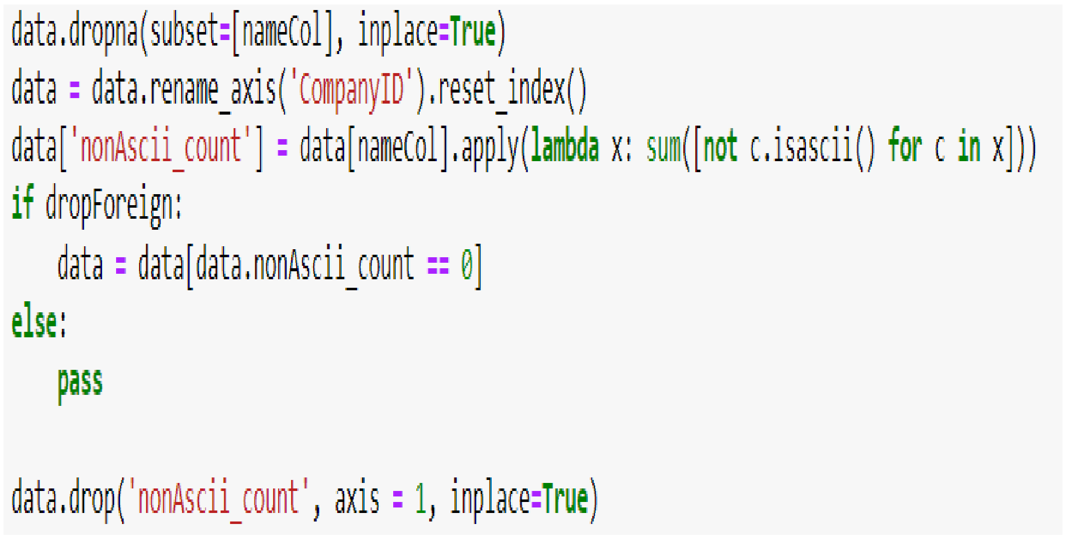
This is a basic pre-processing that includes removing special characters, extra white-spaces, strings containing Non-English characters, and converting all text to lower case.



```python
def clean_special_character(Names):
    seps = [" ",";",":",".",",","#","@","|","/","\\","-","_","?","%","!","^","(",")"]
    default_sep = seps[0]

    for sep in seps[1:]:
        Names = Names.replace(sep, default_sep)
    re.sub(' +',' ', Names)
    Names_Temp = [i.strip() for i in Names.split(default_sep)]
    Names_Temp = [i for i in Names_Temp if i]
    return " ".join(Names_Temp)
```

Figure 1.3: Code Snippet 1



```python
data.dropna(subset=[nameCol], inplace=True)
data = data.rename_axis('CompanyID').reset_index()
data['nonAscii_count'] = data[nameCol].apply(lambda x: sum([not c.isascii() for c in x]))
if dropForeign:
    data = data[data.nonAscii_count == 0]
else:
    pass

data.drop('nonAscii_count', axis = 1, inplace=True)
```

Figure 1.4: Code Snippet 2

1.5 Step 2: Information Retrieval(Stop Word List)

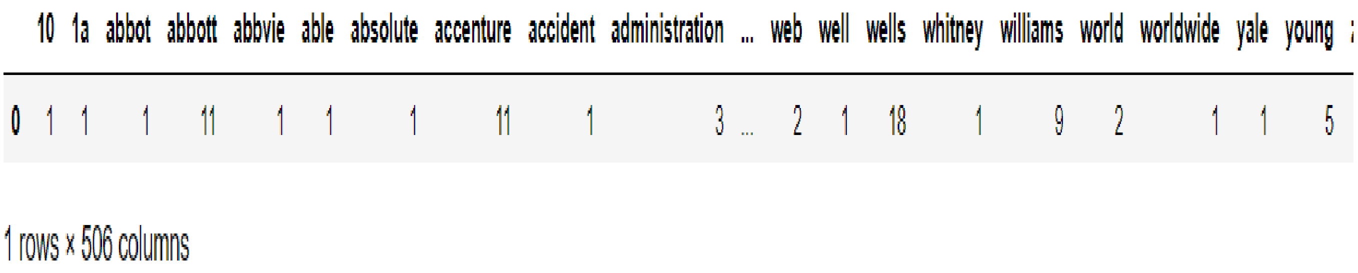
Generally the step of selecting Stop Word List is done manually by going through the entire data-set. To ease out the process we use a concept of tf–idf(or term frequency–inverse document frequency), it is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

It is ften used s weighting ftr in serhes f infrmtin retrievl, text mining, nd user mdeling. The tf–idf vlue inreses rrtinlly t the number f times wrd ers in the dument nd is ffset by the number f duments in the rus tht ntin the wrd, whih hels t djust fr the ft tht sme wrds er mre frequently in generl. tf–idf is ne f the mst ulr term-weighting shemes tdy.

tf–idf can be successfully used for stop-words filtering in various subject fields, including text summarization and classification.

1.5.1 Step 2.1: Count Vectorizer

Converts a collection of text documents to a matrix of counts.This means that for each item in the list, it will produce a list of words in that item, and the count of each words in it.



	10	1a	abbot	abbott	abbvie	able	absolute	accenture	accident	administration	...	web	well	wells	whitney	williams	world	worldwide	yale	young
0	1	1	1	11	1	1	1	11	1	3	...	2	1	18	1	9	2	1	1	5

1 rows × 506 columns

Figure 1.5: Code Snippet 3

1.5.2 Step 2.2: tf-idf Processing

This process will transform the above matrix to a normalized tf or tf-idf representation.

The main aim of using tf-idf instead of raw frequencies of occurrence of a token(or words) in a given document is to scale down the impact of tokens that occur very frequently in a given document and are less informative than those occurring less frequently.

Formula to calculate tf-idf term for a term t in a document d in a document set is defined as:

tf-idf(t,d) = tf(t,d) * idf(t)

Where idf(inverse document frequency) is calculated as:

idf(t) = log[n/df(t)] + 1

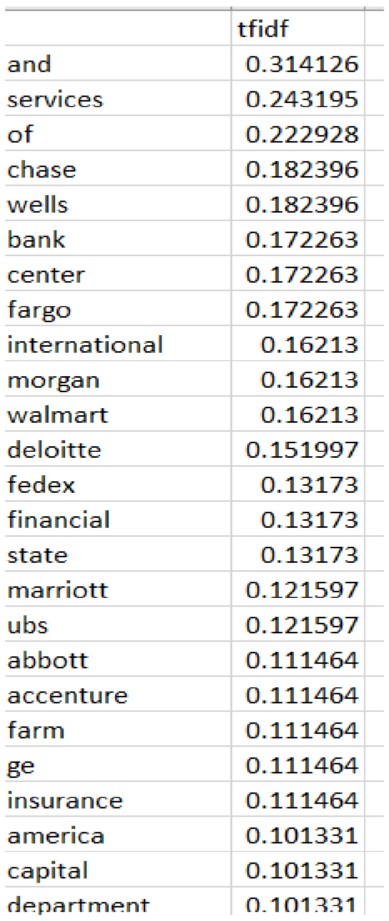
Where n is the total number of documents in document set and df(t) is the document frequency of t, i.e. the total number of documents in the document set that contains the term t.
Here the formula used for the calculation of **idf** is a bit different from the textbook formula i.e.

idf(t) = log[n/df(t) + 1]

In the formula used 1 is added to in the idf so that terms(or words) with zero idf, i.e. words that occur in almost all document are not completely ignored.
Here we also use Smooth idf process where a constant 1 is added both in the numerator and denominator making the effective formula:

idf(t) = log[(1+n)/(1+df(t))] + 1

Here the constant 1 practically signifies that there is an extra document which contains all the terms(or words) in the collection exactly one, this prevents zero division.



	tfidf
and	0.314126
services	0.243195
of	0.222928
chase	0.182396
wells	0.182396
bank	0.172263
center	0.172263
fargo	0.172263
international	0.16213
morgan	0.16213
walmart	0.16213
deloitte	0.151997
fedex	0.13173
financial	0.13173
state	0.13173
marriott	0.121597
ubs	0.121597
abbott	0.111464
accenture	0.111464
farm	0.111464
ge	0.111464
insurance	0.111464
america	0.101331
capital	0.101331
department	0.101331

Figure 1.6: Some Values For tf-idf

Despite of the tf-idf filtering for this particular data-set some manual filtering was required to create a stop word list.

1.6 Step 3: Name Standardization

In this step the company names which are cleaned of special character and stop words are processed to produce the standard names of the company.

This step mainly has **Three** processes:

- Creation of Pairwise Similarity Matrix using fuzzy matching of string(or company names).
- Applying the Affinity Propagation Clustering Algorithm on the computed Similarity Matrix.
- Finding the longest occurring sub-string in a cluster and assigning it as the Standard Company Name.

1.6.1 Step 3.1: Similarity Matrix and Fuzzy Matching

Using the cleansed name from the step above we construct a similarity matrix **S** of size **nXn** where n is the number of company in our data-set i.e. 502 X 502 in this case.The Sij th element in the matrix is the score that will quantify the similarity between ith and jth company name.

Here the similarity score is calculated as the harmonic mean of **partial ratio** and **token set ratio** metrics to get the pairwise similarity metric for two strings(or company name).Using the partial ratio in the metric takes care of the partial string matches.

Partial Ratio:-
Partial ratio helps us to quantify partial matching between two strings(or company names) by taking the shortest string and comparing it with all the sub-string.For Example let X and Y be two strings and let X be the shorter one with length **s**. The metric calculates similarity ratio between the shorter string and every sub-string of length **s** of the longer string and returns the maximum out of those measure.It is an integral score between [0,100].

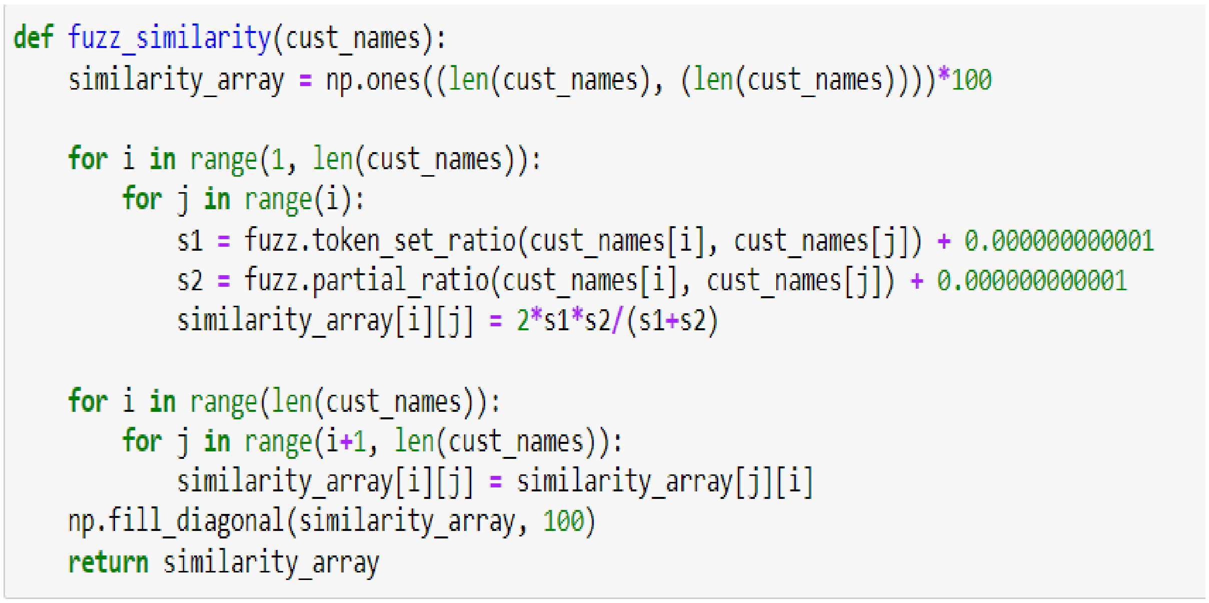
Token Set Ratio:-
In this the strings(or company names) are tokenized, converted to lower case any punctuation if remaining are removed and then they are pasted back together.After this a set operation is applied to remove the common terms(or intersection of two names is taken) and then pairwise similarity is calculated using the fuzz.ratio().

Note:- The Fuzzy Ratio Function is Common in both the ratio and based on the Levenshtein Distance between two strings(or two company names in our case)

Levenshtein Distance:-
Levenshtein distance is a series metric to measure the difference between two strings. Informally, the Levenshtein distance between two words is the minimum number of single character modifications (i.e. insertions, deletions, or substitutions) required to change one word into another.

For Example in our data-set if there are two names JPMORGAN and JPMRGN the Levenshtein distance is 1 as with one edit one name can tranform to other.

Matrix Generation Algorithm and Code Snippet:-



```python
def fuzz_similarity(cust_names):
    similarity_array = np.ones((len(cust_names), (len(cust_names))))*100

    for i in range(1, len(cust_names)):
        for j in range(i):
            s1 = fuzz.token_set_ratio(cust_names[i], cust_names[j]) + 0.000000000001
            s2 = fuzz.partial_ratio(cust_names[i], cust_names[j]) + 0.000000000001
            similarity_array[i][j] = 2*s1*s2/(s1+s2)

    for i in range(len(cust_names)):
        for j in range(i+1, len(cust_names)):
            similarity_array[i][j] = similarity_array[j][i]
    np.fill_diagonal(similarity_array, 100)
    return similarity_array
```

Figure 1.7: Code Snippet For Similarity Matrix Generation

Algorithm 1 An algorithm for creating Similarity Matrix

```
$SimilarityMatrix \leftarrow len(CustomerNames) X len(CustomerNames)$
for $i$ in range(1,len(CustomerNames)) do
    for $j$ in range(i) do
        $S_1$ = TokenSetRatio(CustomerNames[i],CustomerNames[j])+0.000000000001
        $S_2$ = PartialRatio(CustomerNames[i],CustomerNames[j])+0.000000000001
        $SimilarityMatrix[i][j]$ = 2*$S_1$*$S_2$/($S_1$+$S_2$)
for $i$ in range(len(CustomerNames)) do
    for $j$ in range(i+1,len(CustomerNames)) do
        $SimilarityMatrix[i][j] = SimilarityMatrix[j][i]$
$SimilarityMatrix[i][i] \leftarrow 100$
```

1.6.2 Step 3.2: Applying Affinity Propagation Clustering

After creating the similarity matrix in the above step we use the Affinity Propagation Clustering to create cluster containing the names of one company i.e. a cluster of Accenture can contains its name in many types like Accenture Pvt. Ltd.

The main reasons behind choosing Affinity Propagation Clustering are:

- Unlike K-Means algorithm here we do not need to provide it the number of cluster required to be generated it determines it on its own.
- It is very suitable for a pre computed similarity matrix as in our case.

Basic Working Of **Affinity Propagation Clustering**:

For a list of Company Names i.e. $C_1, C_2, C_3.......C_N$.We created a Similarity Matrix **S**, the value of an element S_{ij} in **S** is the harmonic mean of partial ratio and token ratio of i^{th} *and* j^{th} company name.

“The diagonal of S, i.e. S(i, i) is particularly important, as it represents the input preference, i.e. the probability that a particular input will become an instance. When set to the same value for all inputs, it controls the number of classes generated by the algorithm. A value close to the minimum similarity can produce fewer classes, while a value close to or greater than the maximum similarity can produce more classes. It is usually initialized as the median similarity of all input company names in a pairwise manner.”

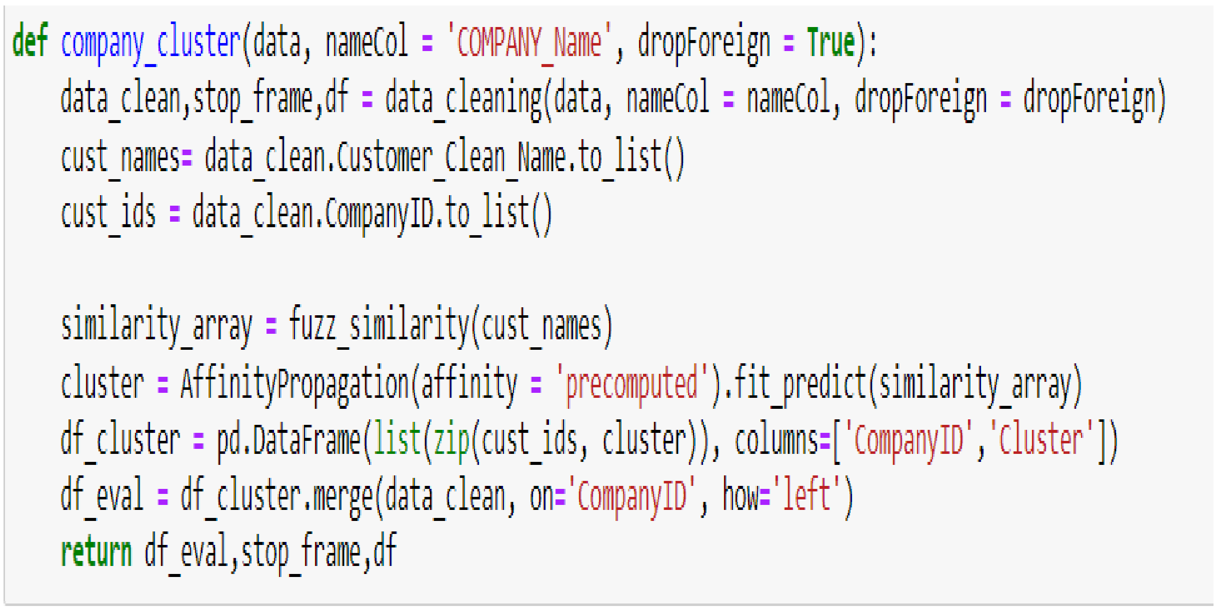
The algorithm proceeds by interleaving two message passes, to update the two matrices:

- The “responsibility” matrix R has r(i, k) values that quantify the likelihood of x_k as an example for x_i, compared with other candidate examples for x_i.
- The “availability” matrix A contains values a(i, k) representing how “appropriate” is for x_i when choosing x_k as an example, taking into account the preference of other points for x_k as an example.

“The iterations are performed until the cluster boundary remains unchanged for a certain number of iterations or after a predetermined number of iterations. The examples extracted from the final matrices are matrices where ”responsibility + availability” to themselves is positive

$$(i.e.(r(i,i) + a(i,i))) > 0)$$

.”



```python
def company_cluster(data, nameCol = 'COMPANY_Name', dropForeign = True):
    data_clean,stop_frame,df = data_cleaning(data, nameCol = nameCol, dropForeign = dropForeign)
    cust_names= data_clean.Customer_Clean_Name.to_list()
    cust_ids = data_clean.CompanyID.to_list()

    similarity_array = fuzz_similarity(cust_names)
    cluster = AffinityPropagation(affinity = 'precomputed').fit_predict(similarity_array)
    df_cluster = pd.DataFrame(list(zip(cust_ids, cluster)), columns=['CompanyID','Cluster'])
    df_eval = df_cluster.merge(data_clean, on='CompanyID', how='left')
    return df_eval,stop_frame,df
```

Figure 1.8: Code Snippet For Clustering

For this data-set there were a total of 66 clusters created i.e. practically indicating there might be 66 companies with variation in their names.

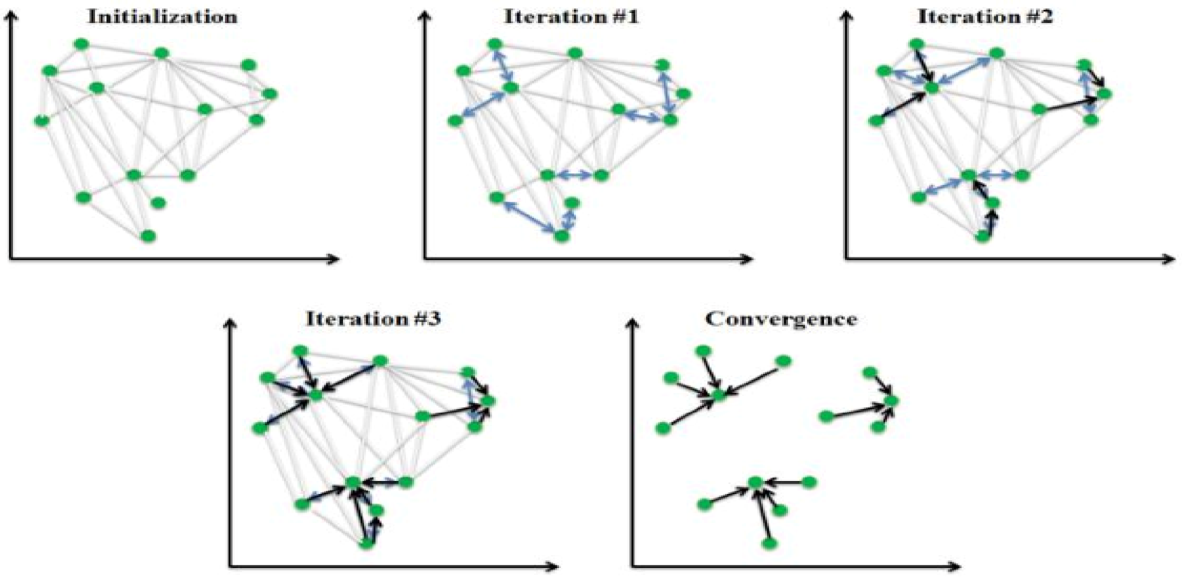


Figure 1.9: Clustering By Iteration

1.6.3 Step 3.3: Finding Standard Name From Each Cluster and Scoring Metrics

After grouping all the names into cluster using Affinity Propagation, we have to assign a common name to each cluster created, this will be our Standard Company Name.

To extract a Standard Name from the each cluster we will use the process called **Longest Common Sub-string**.

In this process we use the *Sequence Matcher* form **difflib** in python.We iterate through all the unique Clusters and extract list of names for that particular clusters.

If the list of names is 1 then we assign the name as standard name. Otherwise we iterate through the list of names in a nested for loop and use the Sequence Matcher to get the longest sub-string.

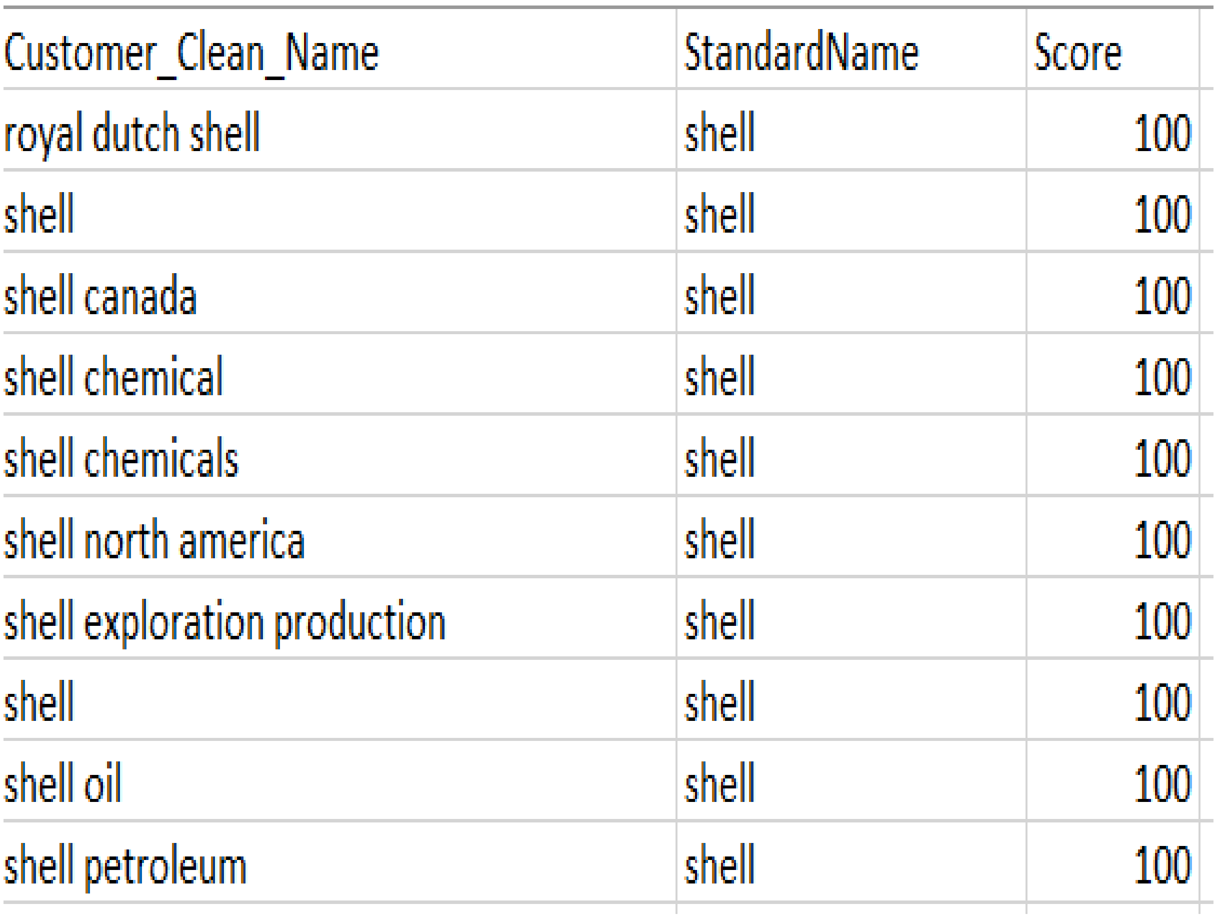
From the list of sub-strings for a cluster, we take the one with the highest occurrence (mode), which is considered as the Standard name to be assigned to the current cluster. The exercise is then repeated for all clusters. It is possible to get multiple modes for a list; in which case, all the modes are returned.

"First we initialize SequenceMatcher object with two input string str_1 and str_2, $find\_longest\_match()$."

Need 4 parameters a_{Low}, b_{Low} is the starting index of the first and second clean company name respectively and a_{High}, b_High is the length of the first and second clean company name respectively. $find\_longest\_match()$ returns a tuple named (i, j, k) such that a[i: i + k] equals b[j: j + k], if no blocks match, it returns (a_{Low}, b_{Low}, 0).

The **Confidence Scoring** is the score metric that will be used to measure whether the standard name provided to the Company Name is correct.

Confidence score quantifies the confidence with which we can say that the standard name we identified truly represents the company name for the raw string(or company name). For cases where multiple standard names were identified, string matching is done with each and mean of all values is taken. The *token_set_ratio* function of the FuzzyWuzzy library is used again for this purpose.



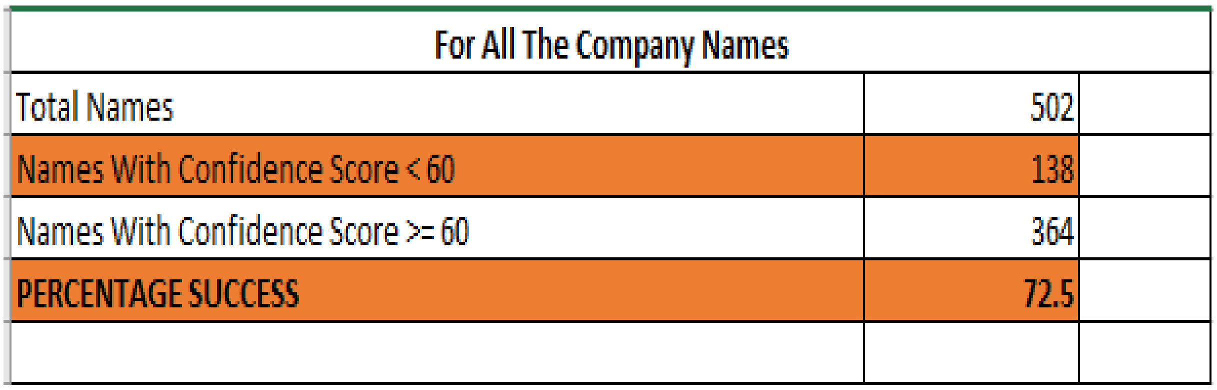
Customer_Clean_Name	StandardName	Score
royal dutch shell	shell	100
shell	shell	100
shell canada	shell	100
shell chemical	shell	100
shell chemicals	shell	100
shell north america	shell	100
shell exploration production	shell	100
shell	shell	100
shell oil	shell	100
shell petroleum	shell	100

Figure 1.10: Confidence Score For Shell Companies

1.7 Results For Sample Data-Set

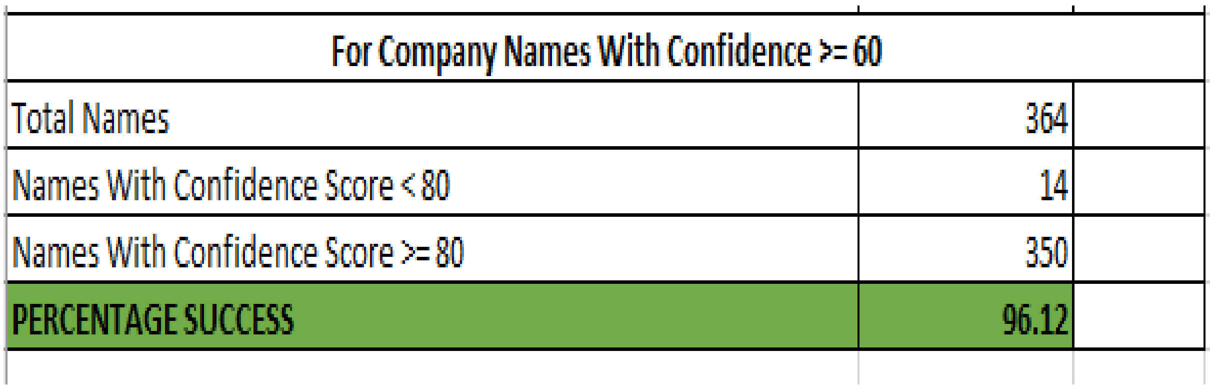
From requirement analysis all the names with **Confidence Score** greater than or equal to 60 are to be considered to be by the client to approach from their marketing campaign.

In the list from the above step, names with score less than 80 are to be consider for manual inspection.The statistics for the result generated are shared below in form of excel tables.



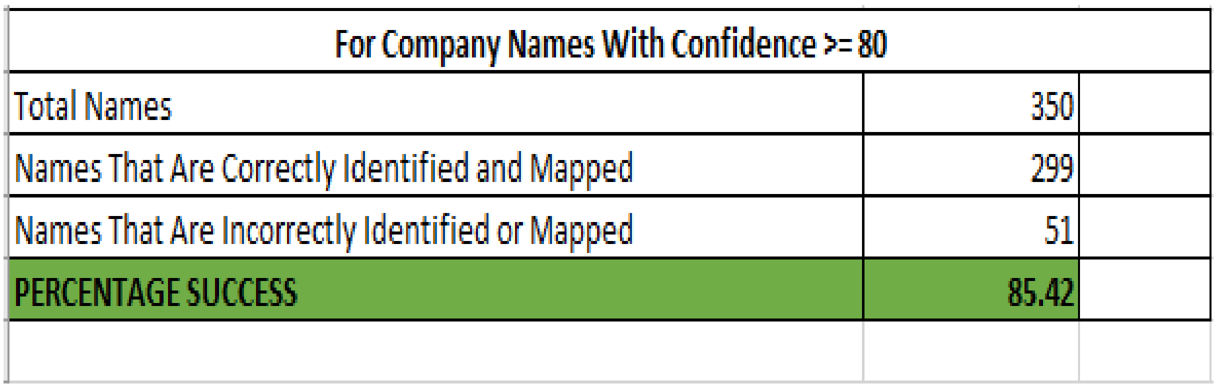
For All The Company Names		
Total Names	502	
Names With Confidence Score < 60	138	
Names With Confidence Score >= 60	364	
PERCENTAGE SUCCESS	72.5	

Figure 1.11: Result Analysis 1



For Company Names With Confidence >= 60		
Total Names	364	
Names With Confidence Score < 80	14	
Names With Confidence Score >= 80	350	
PERCENTAGE SUCCESS	96.12	

Figure 1.12: Result Analysis 2



For Company Names With Confidence >= 80		
Total Names	350	
Names That Are Correctly Identified and Mapped	299	
Names That Are Incorrectly Identified or Mapped	51	
PERCENTAGE SUCCESS	85.42	

Figure 1.13: Result Analysis 3

1.8 Result Discussion

The final results show that 72.5% of the names had a score greater than 60. Out of these names only 14 or 3.88% of names had to be checked manually and of the remaining names only 14.58% of names were either incorrectly identified or mapped to a different standard name.

Thus the **Accuracy** for the standard name assignment can be said to be 85.42% for properly identified clusters and clean name.