

SUPPLY CHAIN OPTIMIZATION

Machine Learning
Project Review
Delhi Technological University

Harshit Jha – 2K19/CO/152



We have developed a solution to a basic logistic problem most MNC's face. How to establish plants and warehouses across the country to reduce the overall cost and meet the SLA of all the shops in our case. Further we also suggest the companies how the shops selling their products can be more efficient with the delivery system

This Power Bi Webpages basically advices the MNC, locations where they can place their warehouses by considering their already made network and advising changes to improve it . It also provides the company with customer delivery routes for every shop



What Is My Project About?

EVOLUTION OF THE PROJECT

Analyse input data which contains
the current supply chain

EXCEL

Python

Process the DATA using Kmeans,
Dijkstra to produce clustering
output and generate TSP results

Get the output data ready to
be visualized in POWER BI

EXCEL

POWER BI

Visualize the output as a
Power Bi report

Supply Chain Network



Plant

Manufacture product using the raw materials



Warehouse

Store product and perform deliveries to nearby shops



Shops

Sells the product within the city to customers

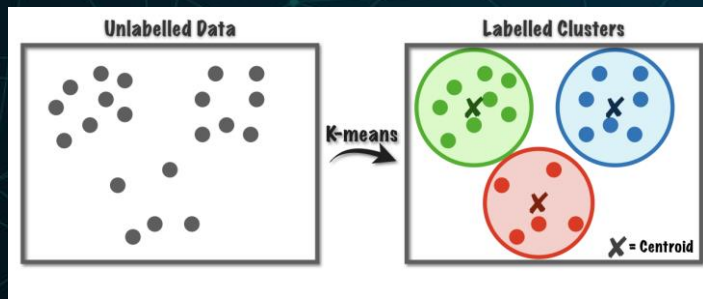


Raw Material

Raw materials are provided to the plant for manufacturing of the products by raw material providers

What is K-Means?

The K-means algorithm is an iterative technique that attempts to split a dataset into K separate non-overlapping subgroups (clusters), each of which contains only one data point. It attempts to make intra-cluster data points as comparable as possible while maintaining clusters as distinct (far) as possible. It distributes data points to clusters in such a way that the sum of the squared distances between them and the cluster's centroid (arithmetic mean of all the data points in that cluster) is as small as possible. Within clusters, the less variance there is, the more homogenous (similar) the data points are.



The distance metric that was used in the project in the execution of the K-Means algorithm was Great Circle Distance. The Great Circle distance is the smallest distance between any two places on the sphere's surface. It takes into consideration the curvature of the Earth's surface. As the project comprises of delivering products and finding optimal paths for the same, it was the optimal measure of distance to be used.

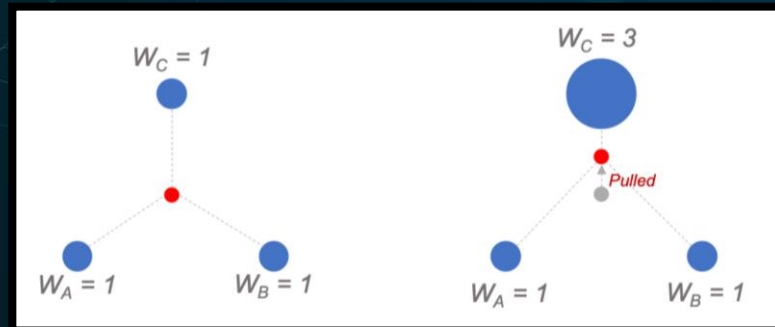
$$d = r \cos^{-1} [\cos \sigma_1 \cos \sigma_2 \cos (\Lambda_1 - \Lambda_2) + \sin \sigma_1 \sin \sigma_2]$$

WHY DO WE USE WEIGHTED K-MEANS?

Assume you're wanting to segment a retail store's customers based on their purchases of organic and local products. Some clients have a high spending propensity in both areas, whereas others have a mixed propensity.

Although a substantial percentage of customers spend heavily in these areas, their overall spending and visit frequency are low, making them less likely to respond to the ad. These customers' data points will result in skewed clusters and centroid values. In this case, weighted K-Means can be used to provide observational weight to each customer data point.

Imagine we're now working with data points of various sizes, and the weights have an impact on the algorithm (represented by their weights). We can also see that the size of the data point is related to its gravitational force. As a result, the larger the weight, the more a data point pulls the centroid toward it. In conclusion, the most significant modification from the usual approach is in the centroid computation, which now use weighted averages rather than standard means.



Cleaning data before K-Means can be applied

Removing records with missing data

27	East	BHUBANESHWAR	20.2961	85.8245	Visakhapatnam	17.74585	83.2426122	14457	72
28	East	BHUBANESHWAR	20.2961	85.8245	Vizianagaram	18.111529	83.395852	7180	96
29	East	GUWAHATI	26.1445	91.7362	Agartala	23.7928483	91.2782839	23938	120
30	East	GUWAHATI	26.1445	91.7362	Rangia	26.4397687		12225	48
31	East	GUWAHATI	26.1445	91.7362	Aizwal	26.1603552	91.7747472	28565	144
32	East	GUWAHATI	26.1445	91.7362	Barpeta	26.1230191	91.6846747	12853	72
33	East	GUWAHATI	26.1445	91.7362	Bongaigaon	26.4902033	90.5512525	14691	72
34	East	GUWAHATI	26.1445	91.7362	Dhubri	26.0206082	89.0742462	5021	96

Removing records which are outliers

78	East	PATNA	25.5941	85.1376	Bhabua	25.6032982	85.137024	8441	72
79	East	PATNA	25.5941	85.1376	Bhagalpur	25.5845965	85.1991676	55567	72
80	East	PATNA	25.5941	85.1376	Biharsharif	25.2049738	85.517437	21002	48
81	East	PATNA	25.5941	85.1376	Raxaul	23.8103	90.4125	4640	96
82	East	PATNA	25.5941	85.1376	Buxar	25.6150013	85.144441	13355	72
83	East	PATNA	25.5941	85.1376	Chapra	25.7421253	84.9770735	5776	72
84	East	PATNA	25.5941	85.1376	Darbhanga	25.6216476	85.1643816	17685	48
85	East	PATNA	25.5941	85.1376	Forbishganj	26.2986049	87.268613	6587	96
86	East	PATNA	25.5941	85.1376	Gaya	25.57103	85.1255618	37855	48

Removing redundant columns

Lon	Demand	SLA	Delivery Date	Customer Satisfactor
85.1510818	25499	48	09-01-2021	1
85.8191119	24387	48	06-06-2021	8
86.737824	12916	72	06-03-2021	4
84.8275064	54704	48	08-02-2021	9
85.8249476	16649	48	22-08-2021	10
84.582481	5282	48	10-03-2021	4
85.8245398	18859	24	25-02-2021	8
85.8245398	9928	72	10-04-2021	10
85.8829895	33605	24	06-04-2021	9
82.8128265	6588	72	26-09-2021	8

Initial Data that we have after the cleaning process

Warehouses

Region	City	Lat	Lon	Level
East	BHUBANESHWAR	20.2961	85.8245	3
East	GUWAHATI	26.1445	91.7362	3
East	KOLKATA	22.5726	88.3639	1
East	PATNA	25.5941	85.1376	3
East	RANCHI DEPOT	23.3441	85.3096	3
North	JAIPUR	26.9124	75.7873	3
North	LUCKNOW	26.8467	80.9462	3
North	NOIDA	28.5355	77.391	3
North	Banur	30.5596	76.6982	1
South	BANGALORE	12.9716	77.5946	1
South	COIMBATORE DEPOT	11.0168	76.9558	3
South	HYDERABAD	17.385	78.4867	3
West	AHMEDABAD	23.01451	72.59176	3
West	INDORE	22.7196	75.8577	3
West	MUMBAI	19.076	72.8777	1
West	NAGPUR	21.1458	79.0882	3
West	PUNE	18.5204	73.8567	3
West	RAIPUR	21.2514	81.6296	3
North	ZIRAKPUR DEPOT	30.6425	76.8173	3

Region	City	Lat	Lon	Demand
North	Akkanwali	29.8683	75.4214	177637
West	Aurangabad	19.17833	74.7679	82502
North	Baddi	30.95783	76.79136	863191
North	Bahadurgarh	28.6914	76.9314	117083
North	Baddi	30.95783	76.79136	415545
North	Bhiwadi	28.20141	76.82755	137146
West	Chennai	13.08268	80.27072	0
North	Chhat Rampur	26.3584	89.6405	753402
West	Daman	20.39737	72.8328	231555
North	Dehradun	30.31649	78.03219	13320
North	Faridabad	28.40891	77.31779	49417
North	Gagret Una	31.4685	76.2708	11012
North	Haridwar	29.94569	78.16425	636019
West	Hyderabad	17.38504	78.48667	422323
North	Jhilmil Colony, New Delhi	28.6715	77.3066	1430
North	Jodhpur	26.23895	73.02431	10747
West	Kaikalampalayam	11.0745	77.055	25400

Region	Branch	Branch L	Branch L	City	Lat	Lon	Demand	SLA
East	BHUBANESHWAR	20.2961	85.8245	Angul	20.8444033	85.1510818	25499	48
East	BHUBANESHWAR	20.2961	85.8245	Balasore	20.3098259	85.8191119	24387	48
East	BHUBANESHWAR	20.2961	85.8245	Baripada	21.923201	86.737824	12916	72
East	BHUBANESHWAR	20.2961	85.8245	Berhampur	19.3105216	84.8275064	54704	48
East	BHUBANESHWAR	20.2961	85.8245	Bhadrak	20.2449129	85.8249476	16649	48
East	BHUBANESHWAR	20.2961	85.8245	Bhanjanagar	19.9358071	84.582481	5282	48
East	BHUBANESHWAR	20.2961	85.8245	Bhubaneswar	20.2960587	85.8245398	18859	24
East	BHUBANESHWAR	20.2961	85.8245	Bolangir	20.2960587	85.8245398	9928	72
East	BHUBANESHWAR	20.2961	85.8245	Cuttack	20.462521	85.8829895	33605	24
East	BHUBANESHWAR	20.2961	85.8245	Gunupur	19.0754648	83.8128265	6589	72
East	BHUBANESHWAR	20.2961	85.8245	Jajpur Road	20.9550552	86.1271048	31237	48
East	BHUBANESHWAR	20.2961	85.8245	Jeypore	18.8963119	82.553821	16659	96
East	BHUBANESHWAR	20.2961	85.8245	Jharsuguda	21.8829086	84.0278719	26334	48
East	BHUBANESHWAR	20.2961	85.8245	Keonjhar	21.628933	85.5816847	14630	72
East	BHUBANESHWAR	20.2961	85.8245	Kesinga	20.1850477	83.2104426	9038	96
East	BHUBANESHWAR	20.2961	85.8245	Khariar Road, Dist Nuapada	20.2866522	82.762337	14489	96
East	BHUBANESHWAR	20.2961	85.8245	Nayagarh	20.305678	85.85378	13870	48

Shops

Raw Material provider

Calculating new plant locations to suggest a new supply network



Python Code to calculate plant locations

Scaling
points

```
df_sh = df_shops.copy()[['Lat', 'Lon', 'Demand']] # new dataframe with 3 columns from vendor dataframe
df_sh['Type'] = 0
df_ven = df_vendors.copy()[['Lat', 'Lon', 'Demand']]
df_ven['Type'] = 1
df_sh['Demand'] = df_sh['Demand'] / 100
df = pd.concat([df_ven, df_sh])
```

```
km = KMeans(n_clusters=start_point, random_state=rand) # kmeans object number of centroid, rand
km.fit(X=df[["Lat", "Lon"]], sample_weight=df["Demand"]) # df passes is dataset to work on, wei
df['Cluster'] = km.labels_ # label of each pt
centers = pd.DataFrame(km.cluster_centers_, columns=["Lat", "Lon"]) # lat and long of clusters
```

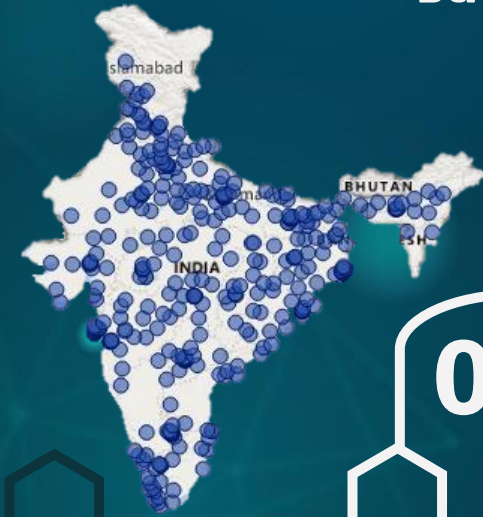
Fitting data in
K-Means
object

Output

	Lat	Lon	Cluster	Region	City	Demand	Level
0	29.308025	78.503308	0	North	Dhampur	6612227.0	1.0
1	19.082698	74.735461	1	West	Ahmednagar	1534817.0	1.0

Process finished with exit code -1

Calculating new warehouse locations to suggest a new supply network

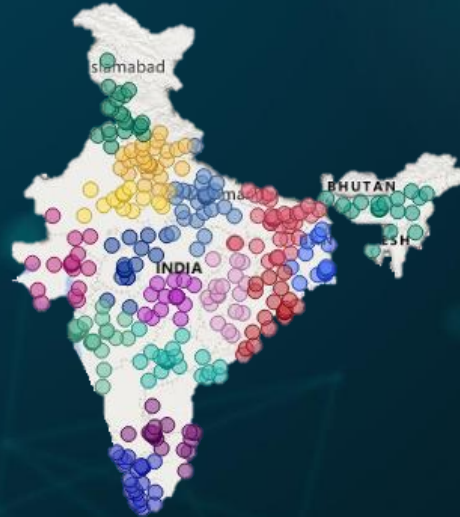


01

Location of
Company Shops all
across India

Segregating shops into clusters
after applying K-Means

02



03

Making the cluster
centroids as new
warehouses

Python Code to calculate new warehouses


```
df = df_shops.copy()

km = KMeans(n_clusters=start_point, random_state=rand) # no of centers is k (10 to 30)
km.fit(X=df[["Lat", "Lon"]], sample_weight=df["TotalDemand"]) # dataset is all customers

df["Cluster"] = km.labels_
centers = pd.DataFrame(km.cluster_centers_, columns=["Cluster Lat", "Cluster Lon"]) # k clustered warehouse locations
centers["Cluster"] = centers.index
```

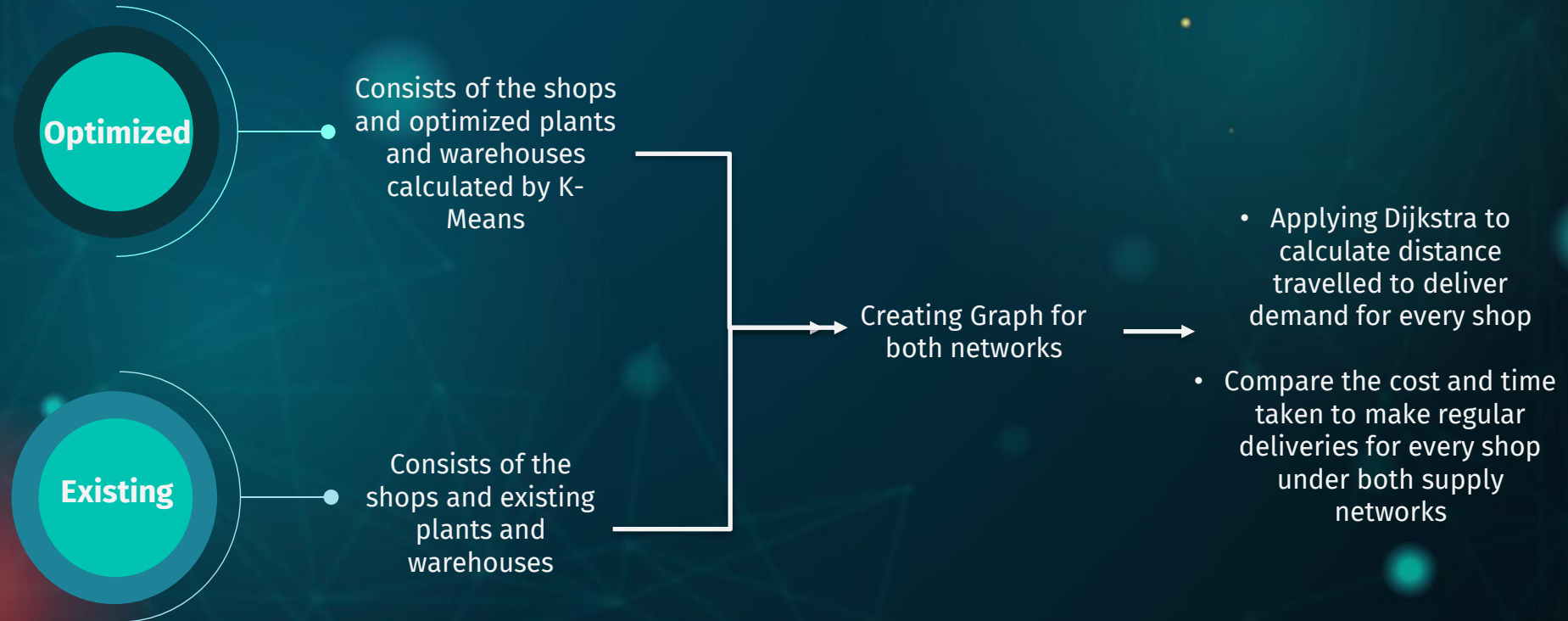
Fitting data in
K-Means
object

Output



	Cluster Lat	Cluster Lon	Cluster	Cluster City
0	26.160355	91.774747	0	Aizwal
1	17.522534	78.522522	1	Karimnagar
2	29.308025	78.503308	2	Dhampur
3	21.251384	81.629641	3	Baradwar
4	13.435498	77.731534	4	CHICKBALLAPUR
5	22.778804	73.614279	5	Godhra
6	21.923201	86.737824	6	Baripada
7	24.289552	87.255183	7	Dumka
8	26.781491	80.884538	8	Kanpur
9	19.082698	74.735461	9	Ahmednagar

Comparing Cost, SLA level and Other parameters between Optimized and Existing Supply chain Network



Python code to make graphs and calculate distance

Graph Creation

```
g = nx.Graph()
g.add_nodes_from(source)

for index, v in vendors.iterrows():
    for windex, w in warehouse[(warehouse.Level == 1)].iterrows():
        g.add_edge(v['Vendor_Key'], w['City'], weight=distance(v['Lat'], v['Lon'], w['Lat'], w['Lon']))

for index, m in warehouse[(warehouse.Level < 3)].iterrows():
    for rindex, r in warehouse.iterrows():
        g.add_edge(m['City'], r['City'], weight=distance(m['Lat'], m['Lon'], r['Lat'], r['Lon']))

for index, m in warehouse[(warehouse.Level == 1)].iterrows():
    for rindex, r in warehouse[(warehouse.Level == 2)].iterrows():
        g.add_edge(m['City'], r['City'], weight=distance(m['Lat'], m['Lon'], r['Lat'], r['Lon']))

if existing:
    for cindex, cu in customers.iterrows():
        g.add_edge(cu[branch_colname], 'customer_' + cu['City'],
                    weight=distance(cu['Lat'], cu['Lon'], cu[branch_colname + ' Lat'], cu[branch_colname + ' Lon']))
else:
    for cindex, cu in customers.iterrows():
        g.add_edge(cu['Cluster City'], 'customer_' + cu['City'],
                    weight=distance(cu['Lat'], cu['Lon'], cu[branch_colname + ' Lat'], cu[branch_colname + ' Lon']))

return g
```

```
['Dhampur', 'Baradwar', 'customer_Visakhapatnam'] Distance = 1942.880534351929
['Dhampur', 'Baradwar', 'customer_Vizianagaram'] Distance = 1931.044506801948
['Ahmednagar', 'Baradwar', 'customer_Balaghat'] Distance = 896.8465906893256
['Ahmednagar', 'Baradwar', 'customer_Chhindwara'] Distance = 876.9516248774969
['Ahmednagar', 'Baradwar', 'customer_Seoni'] Distance = 876.3177279279955
['Ahmednagar', 'Baradwar', 'customer_Shahdol'] Distance = 872.592725103609
['Ahmednagar', 'Baradwar', 'customer_Wani'] Distance = 1226.9488553294177
['Ahmednagar', 'Baradwar', 'customer_Akola'] Distance = 999.5875449495728
```

Output

```
for index, c in customer.iterrows():
    mw = get_mother_warehouse(warehouse, c.Region)
    target1 = "customer_" + c.City
    m = nx.shortest_path(network_graph, mw, target1, weight='distance')
    totaldis = 0
    for ran in range(len(m) - 1):
        totaldis = nx.dijkstra_path_length(network_graph, m[0], 'customer_' + c.City, weight='weight')
    customer.at[index, 'end_to_end_existing_distance'] = totaldis + average_inward_distance
```

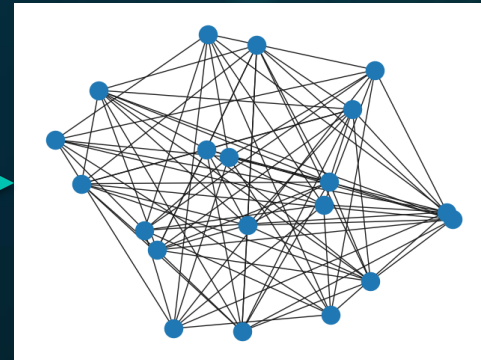
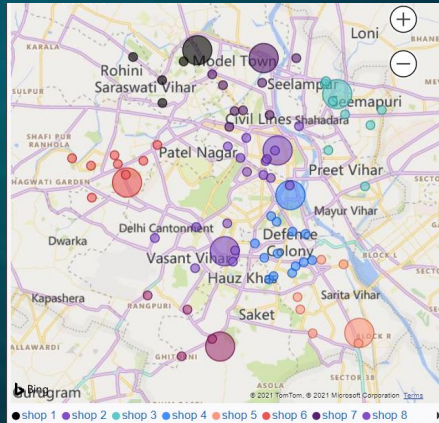
Applying Dijkstra for
shortest path

Calculating inter-city delivery clusters for shops and their customers and creating their disjoint graphs

We choose a sample city like Delhi with 8 shopkeepers and 73 customers (consumers of the product). After the product is distributed amongst the shopkeepers throughout India, customers like us buy these products online and offline as well. Products bought offline is not considered here, rather we focus on the products bought online. The online customers expect an efficient delivery from the nearby shops so using K-means we cluster the customers into X clusters where X is the number of shops in the area. Then we assign each cluster to the shop nearest to the centroid. Efficiently delivering the products by saving fuel and time satisfies the customer and saves the shopkeeper money which converts into profit for the company as well.

8 disjoint graphs are created to connect shopkeepers with their respective customers which bought the product from that shop. All of these 8 graphs are disjoint but each of these graphs are complete graphs

Clustering
result



Example of
the graph for
shop 8

Applying K-Means to create delivery clusters

```
km = KMeans(n_clusters=count_auth, random_state=101)
km.fit(X=df_cit[["Lat", "Long"]])
centers = pd.DataFrame(km.cluster_centers_, columns=["Center Lat", "Center Long"])
centers["Cluster"] = centers.index
df_cit["Cluster"] = km.labels_
```

Clustering
code

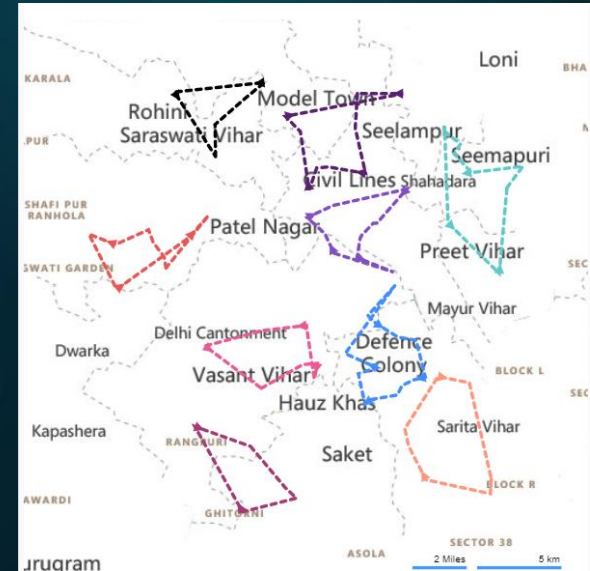
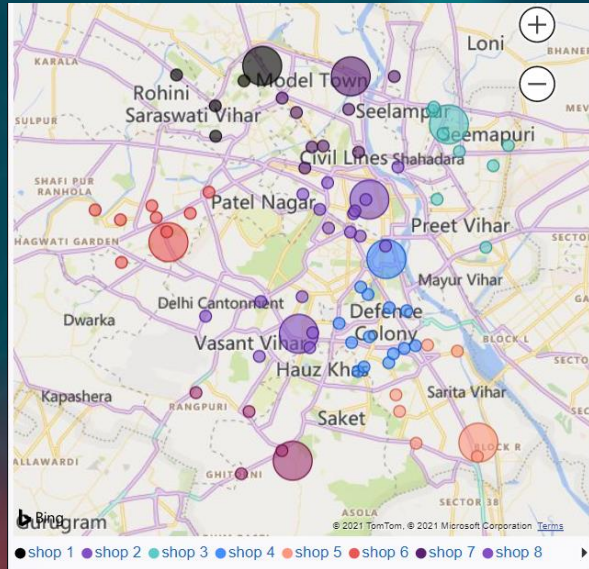


Clustering
result

Lat	Long	identity	level	Cluster	Center Lat	Center Long	shop
28.61419	77.07154	customer 1	1	3	28.6256914	77.10194107	shop 6
28.67979	77.19491	customer 8	1	5	28.720002	77.220003	shop 7
28.71745	77.15087	customer 7	1	8	28.7259717	77.162658	shop 1
28.65952	77.20501	customer 28	1	2	28.6499765	77.2320588	shop 2
28.60014	77.22649	customer 36	1	6	28.61609	77.243048	shop 4
28.66916	77.31227	customer 41	1	4	28.6926703	77.28354435	shop 3
28.57416	77.19537	customer 64	1	1	28.5735343	77.1863593	shop 8
28.56366	77.28905	customer 53	1	0	28.5115704	77.3026233	shop 5
28.54001	77.11978	customer 63	1	7	28.5012304	77.1823908	shop 9

Applying TSP and calculating delivery paths for each shop

To make maximum deliveries in one run for a particular shop, it is profitable, and time saving for the delivery personnel to visit every customer of that particular shop and return back after completing all the deliveries in one circuit. Hence, we apply TSP algorithm to calculate such a path. We apply TSP (heuristics algorithm) for every separate graph and formulate the TSP paths for the delivery personnel to follow. All these paths are displayed using a webpage made by PowerBI using data generated by python.



Applying TSP to on graphs to find optimal paths

```
def find_path(g, gsource, source, path, path_calc, totdis=0):
    if len(path_calc) == 1:
        path.append(path_calc[0])
        path.append(gsource)
        totdis = totdis + nx.single_source_dijkstra(g, gsource, path_calc[0])[0]
        return totdis, path
    closest_node = path_calc[0]
    dis = nx.single_source_dijkstra(g, source, closest_node)[0]
    for node in path_calc:
        tempdis = nx.single_source_dijkstra(g, source, node)[0]
        if tempdis < dis:
            closest_node = node
            dis = tempdis
    path.append(closest_node)
    path_calc.remove(closest_node)
    totdis = totdis + dis
    totdis, path = find_path(g, gsource, closest_node, path, path_calc, totdis)
    return totdis, path
```

Applying TSP

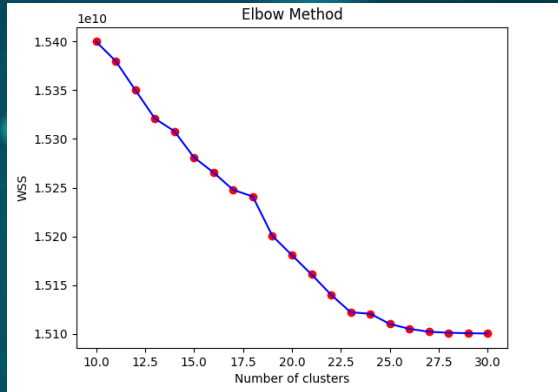


TSP paths
generated

```
['shop 8', 'customer 64', 'customer 54', 'customer 35', 'customer 31', 'customer 61', 'customer 68', 'customer 66', 'customer 60',
 'shop 7', 'customer 18', 'customer 12', 'customer 13', 'customer 8', 'customer 17', 'customer 16', 'customer 27', 'customer 9', '
 'shop 6', 'customer 69', 'customer 71', 'customer 14', 'customer 73', 'customer 72', 'customer 1', 'customer 70', 'customer 67',
 'customer 41', 'customer 20', 'customer 21', 'customer 42', 'customer 19', 'shop 3', 'customer 24', 'customer 38', 'customer 22',
 'customer 55', 'customer 50', 'customer 48', 'customer 46', 'customer 57', 'customer 49', 'customer 51', 'customer 53', 'customer
 'shop 2', 'customer 11', 'customer 10', 'customer 25', 'customer 29', 'customer 32', 'customer 30', 'customer 37', 'customer 36',
 'shop 1', 'customer 7', 'customer 4', 'customer 5', 'customer 6', 'customer 3', 'shop 1']
['customer 65', 'customer 44', 'shop 9', 'customer 62', 'customer 63', 'customer 65']
['shop 5', 'customer 59', 'customer 58', 'shop 5']
```


USING THE ELBOW METHOD TO CALCULATE THE OPTIMAL K FOR K-MEANS

We used the WSS technique to try and identify the best K for K means. Even though we provided the whole model results for K from 13 to 30 for the company to be able to see that how the cost and service level shift increasing the credibility of the model, still we computed WSS's best K.

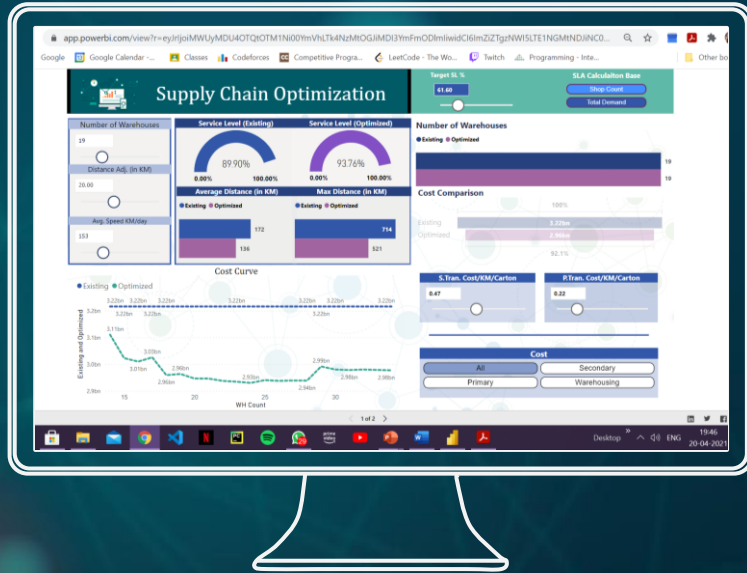


The elbow method suggest that the knee point is at $k = 23$ and hence 23 are the optimal number of warehouses (clusters) that we need to calculate



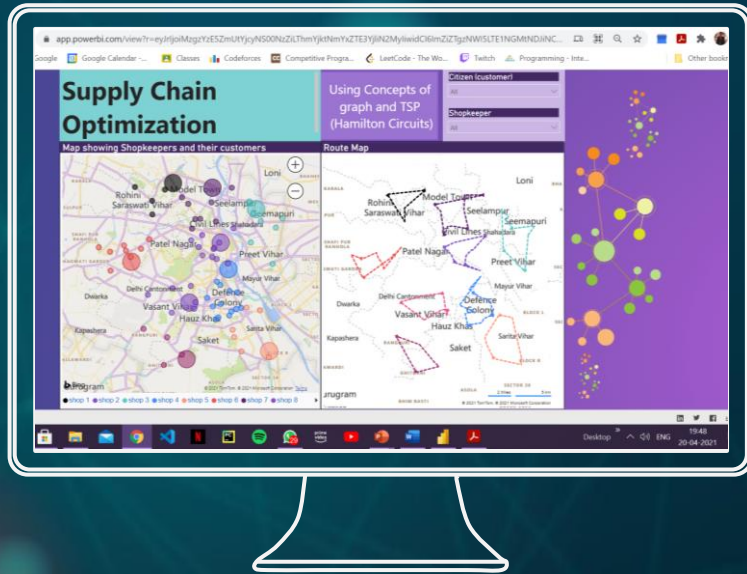
The final cost curve also suggests that $K = 23$ is the optimal number of warehouses that we need.

POWER BI WEB SOLUTION FOR SUPPLY NETWORK COMPARISON



Using Microsoft Power BI We have created a web solution to demonstrate how this project accurately tends to optimize cost and achieve SLA

POWER BI WEB SOLUTION FOR INTERCITY DELIVERY SYSTEM



Using Microsoft Power BI We have created a web solution to demonstrate how shops can efficiently deliver products to their customers