SUPPLY CHAIN MANAGEMENT

Prepared
By
M.ABINAYA

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROBLEM : PRODUCT WEIGHT SHIPMENT

A FMCG company has entered into the instant noodles business two years back. Their higher management has notices that there is a miss match in the demand and supply. Where the demand is high, supply is pretty low and where the demand is low, supply is pretty high. In both the ways it is an inventory cost loss to the company; hence, the higher management wants to optimize the supply quantity in each and every warehouse in entire country.

1.1 INTRODUCTION :

The management of the FMCG company provide few datas to look after the ways to reduce inventory loss to the company because there is a mismatch between Demand and Supply. And also they wanted to optimize the supply quantity in each and every warehouse in the entire country.

- **PROJECT IMPORTANCE :**

The product shipment project helps the company to analysis the demand pattern in different pockets of the country so management can drive the advertisement campaign particular in those pockets to increase its sales.

- **PROJECT OBJECTIVE :**

To build a model, using historical data that will determine an optimum weight of the product to be shipped each time to the warehouse.

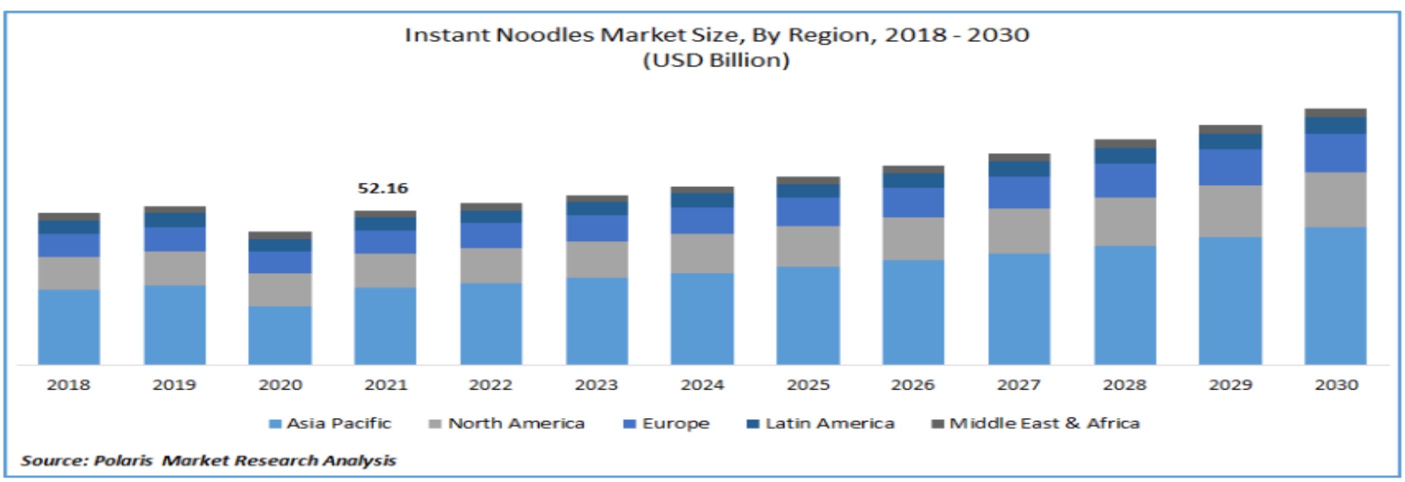
- **INSTANT NOODLES MARKET - GROWTH & ITS SIZE :**

  - The global instant noodles market was valued at USD 52.16 billion in 2021 and is expected to grow at a CAGR of 5.91% during the forecast period (2022 - 2023).

  - The outbreak of COVID-19 and the resultant measures imposed by the government have resulted in the strict closure of market to implement social distancing. Consumers preferred home cooked food and instant foods.

- It is essential to understand the trends and opportunities that lie ahead for the Food industry. The increased demand for quick-to-prepare foods like instant noodles is fueled by increasing urbanization, a growing middle class, and an increase in working women.

- The expansion of the organized food retail sector is anticipated to boost sales of instant noodles because there are more department stores, hypermarkets, supermarkets everywhere. During the predicted period, there will likely be a noticeable increase in the demand for instant noodles worldwide.



1.1 Instant noodles market size

1.2 EDA :

❖ DATA COLLECTION :

- The company has entered into the instant noodles business two years back andso they shared only limited information.

- The data given by the company is Location based whether the warehouse present in Rural or Urban areas.

- We have to find out the mismatch between Demand and Supply in various parts of the country and then focus on product weight in each warehouse to increase the supply.

❖ DATA SUMMARY :

- The given dataset has 25000 entries with 24 variables.
- Initially It has 8 Categorical and 16 Numerical variables.
- Product_wg_ton – Target variable.

HEAD :

	Ware_house_ID	WH_Manager_ID	Location_type	WH_capacity_size	zone	WH_regional_zone	num_refill_req_l3m	transport_issue_l1y	Competitor_in_mkt	ret
0	WH_100000	EID_50000	Urban	Small	West	Zone 6	3	1	2	
1	WH_100001	EID_50001	Rural	Large	North	Zone 5	0	0	4	
2	WH_100002	EID_50002	Rural	Mid	South	Zone 2	1	0	4	
3	WH_100003	EID_50003	Rural	Mid	North	Zone 3	7	4	2	
4	WH_100004	EID_50004	Rural	Large	North	Zone 5	3	1	2	

5 rows × 24 columns

TAIL :

	Ware_house_ID	WH_Manager_ID	Location_type	WH_capacity_size	zone	WH_regional_zone	num_refill_req_l3m	transport_issue_l1y	Competitor_in_mkt
24995	WH_124995	EID_74995	Rural	Small	North	Zone 1	3	0	4
24996	WH_124996	EID_74996	Rural	Mid	West	Zone 2	6	0	4
24997	WH_124997	EID_74997	Urban	Large	South	Zone 5	7	0	2
24998	WH_124998	EID_74998	Rural	Small	North	Zone 1	1	0	2
24999	WH_124999	EID_74999	Rural	Mid	West	Zone 4	8	2	4

5 rows × 24 columns

Data Dictionary :

Variable	Business Definition
Ware_house_ID	Product warehouse ID
WH_Manager_ID	Employee ID of warehouse manager
Location_type	Location of warehouse like in city or village
WH_capacity_size	Storage capacity size of the warehouse
zone	Zone of the warehouse
WH_regional_zone	Regional zone of the warehouse under each zone
num_refill_req_l3m	Number of times refilling has been done in last 3 months
transport_issue_l1y	Any transport issue like accident or goods stolen reported in last one year
Competitor_in_mkt	Number of instant noodles competitor in the market
retail_shop_num	Number of retails shop who sell the product under the warehouse area
wh_owner_type	Company is owning the warehouse or they have get the warehouse on rent
distributor_num	Number of distributer works in between warehouse and retail shops
flood_impacted	Warehouse is in the Flood impacted area indicator
flood_proof	Warehouse is flood proof indicators. Like storage is at some height not directly on the ground
electric_supply	Warehouse have electric back up like generator, so they can run the warehouse in load shedding
dist_from_hub	Distance between warehouse to the production hub in Kms
workers_num	Number of workers working in the warehouse
wh_est_year	Warehouse established year
storage_issue_reported_l3m	Warehouse reported storage issue to corporate office in last 3 months. Like rat, fungus because of moisture etc.
temp_reg_mach	Warehouse have temperature regulating machine indicator
approved_wh_govt_certificate	What kind of standard certificate has been issued to the warehouse from government regulatory body
wh_breakdown_l3m	Number of time warehouse face a breakdown in last 3 months. Like strike from worker, flood, or electrical failure
govt_check_l3m	Number of time government Officers have been visited the warehouse to check the quality and expire of stored food in last 3 months
product_wg_ton	Product has been shipped in last 3 months. Weight is in tons

SHAPE : (25000, 24)

INFO:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Ware_house_ID                 25000 non-null  object 
 1   WH_Manager_ID                 25000 non-null  object 
 2   Location_type                 25000 non-null  object 
 3   WH_capacity_size              25000 non-null  object 
 4   zone                          25000 non-null  object 
 5   WH_regional_zone              25000 non-null  object 
 6   num_refill_req_l3m            25000 non-null  int64  
 7   transport_issue_l1y           25000 non-null  int64  
 8   Competitor_in_mkt             25000 non-null  int64  
 9   retail_shop_num               25000 non-null  int64  
 10  wh_owner_type                 25000 non-null  object 
 11  distributor_num               25000 non-null  int64  
 12  flood_impacted                25000 non-null  int64  
 13  flood_proof                   25000 non-null  int64  
 14  electric_supply               25000 non-null  int64  
 15  dist_from_hub                 25000 non-null  int64  
 16  workers_num                   24010 non-null  float64
 17  wh_est_year                   13119 non-null  float64
 18  storage_issue_reported_l3m    25000 non-null  int64  
 19  temp_reg_mach                 25000 non-null  int64  
 20  approved_wh_govt_certificate  24092 non-null  object 
 21  wh_breakdown_l3m              25000 non-null  int64  
 22  govt_check_l3m                25000 non-null  int64  
 23  product_wg_ton                25000 non-null  int64  
dtypes: float64(2), int64(14), object(8)
memory usage: 4.6+ MB
```

- 8 variables can be converted from Numerical into Categorical (since they are predefined) so finally we got 16 Categorical and only 8 Numerical variables.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype   
---  ------                        --------------  -----   
 0   Ware_house_ID                 25000 non-null  object  
 1   WH_Manager_ID                 25000 non-null  object  
 2   Location_type                 25000 non-null  object  
 3   WH_capacity_size              25000 non-null  object  
 4   zone                          25000 non-null  object  
 5   WH_regional_zone              25000 non-null  object  
 6   num_refill_req_l3m            25000 non-null  category
 7   transport_issue_l1y           25000 non-null  category
 8   Competitor_in_mkt             25000 non-null  category
 9   retail_shop_num               25000 non-null  int64   
 10  wh_owner_type                 25000 non-null  object  
 11  distributor_num               25000 non-null  int64   
 12  flood_impacted                25000 non-null  category
 13  flood_proof                   25000 non-null  category
 14  electric_supply               25000 non-null  category
 15  dist_from_hub                 25000 non-null  int64   
 16  workers_num                   24010 non-null  float64 
 17  wh_est_year                   13119 non-null  float64 
 18  storage_issue_reported_l3m    25000 non-null  int64   
 19  temp_reg_mach                 25000 non-null  category
 20  approved_wh_govt_certificate  24092 non-null  object  
 21  wh_breakdown_l3m              25000 non-null  category
 22  govt_check_l3m                25000 non-null  int64   
 23  product_wg_ton                25000 non-null  int64   
dtypes: category(8), float64(2), int64(6), object(8)
memory usage: 3.2+ MB
```

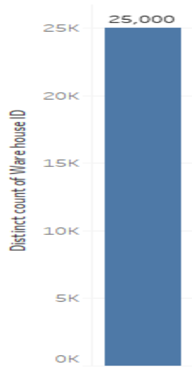
No Duplicates present in the given dataset.

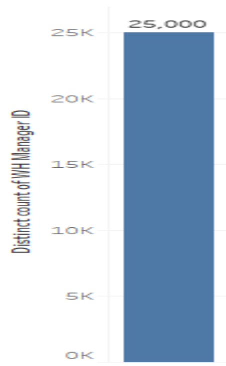
DATA DESCRIPTION :

	retail_shop_num	distributor_num	dist_from_hub	workers_num	wh_est_year	storage_issue_reported_l3m	govt_check_l3m	product_wg_ton
count	25000.000000	25000.000000	25000.000000	24010.000000	13119.000000	25000.000000	25000.000000	25000.000000
mean	4985.711560	42.418120	163.537320	28.944398	2009.383185	17.130440	18.812280	22102.632920
std	1052.825252	16.064329	62.718609	7.872534	7.528230	9.161108	8.632382	11607.755077
min	1821.000000	15.000000	55.000000	10.000000	1996.000000	0.000000	1.000000	2065.000000
25%	4313.000000	29.000000	109.000000	24.000000	2003.000000	10.000000	11.000000	13059.000000
50%	4859.000000	42.000000	164.000000	28.000000	2009.000000	18.000000	21.000000	22101.000000
75%	5500.000000	56.000000	218.000000	33.000000	2016.000000	24.000000	26.000000	30103.000000
max	11008.000000	70.000000	271.000000	98.000000	2023.000000	39.000000	32.000000	55151.000000

UNIVARIATE ANALYSIS :

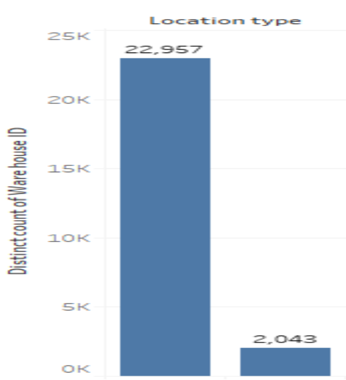
Univariate Analysis done using TABLEAU Tool.

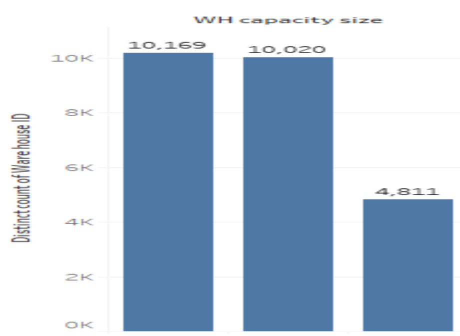




1.2 Unique WH and Manager ID count

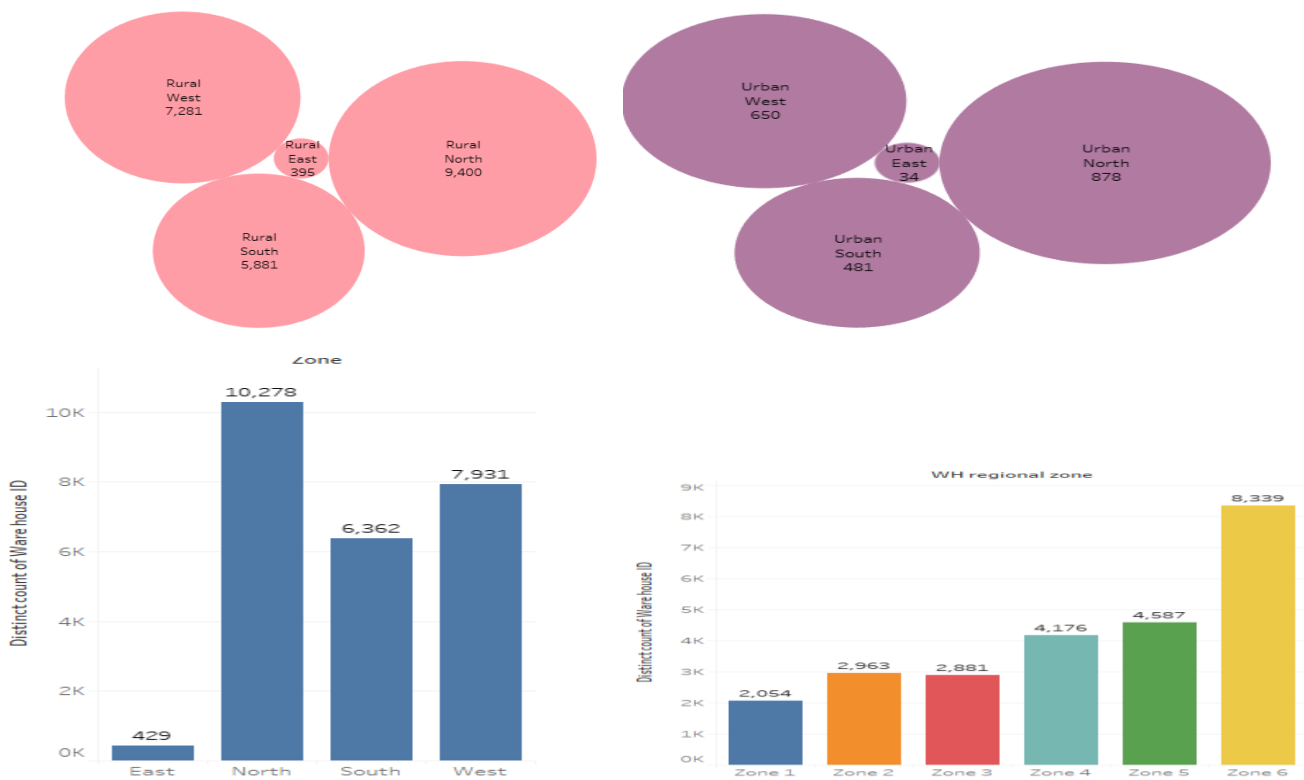
There are 25000 Unique Warehouse and 25000 Unique Warehouse Manager ID given in the dataset.





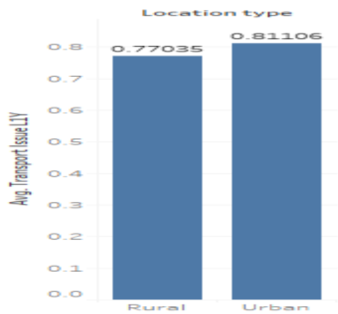
1.3 Location and capacity of WH

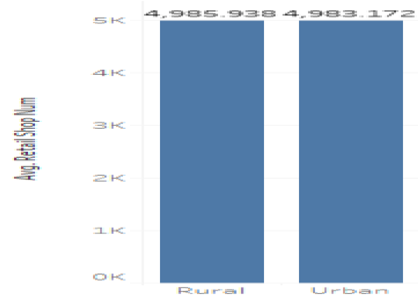
- Presence of 22957 Warehouse in Rural areas(91.8%) and 2043 Warehouse in Urban areas(8.2%).
- So we can say that majority of warehouses are present in Rural areas.
- North zone in both Rural and Urban areas has 41% and 43% of warehouses respectively.
- Large sized warehouse are more followed by Mid and Small.

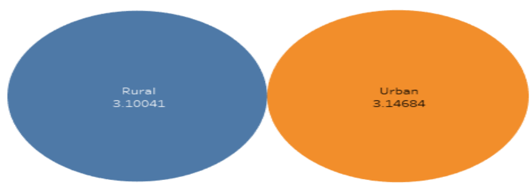


1.4 Region & Zone wise count of WH

- Presence of more Warehouses in North zone followed by South , West and East.
- Also more number of warehouse present in Zone 6 and least in Zone 1.

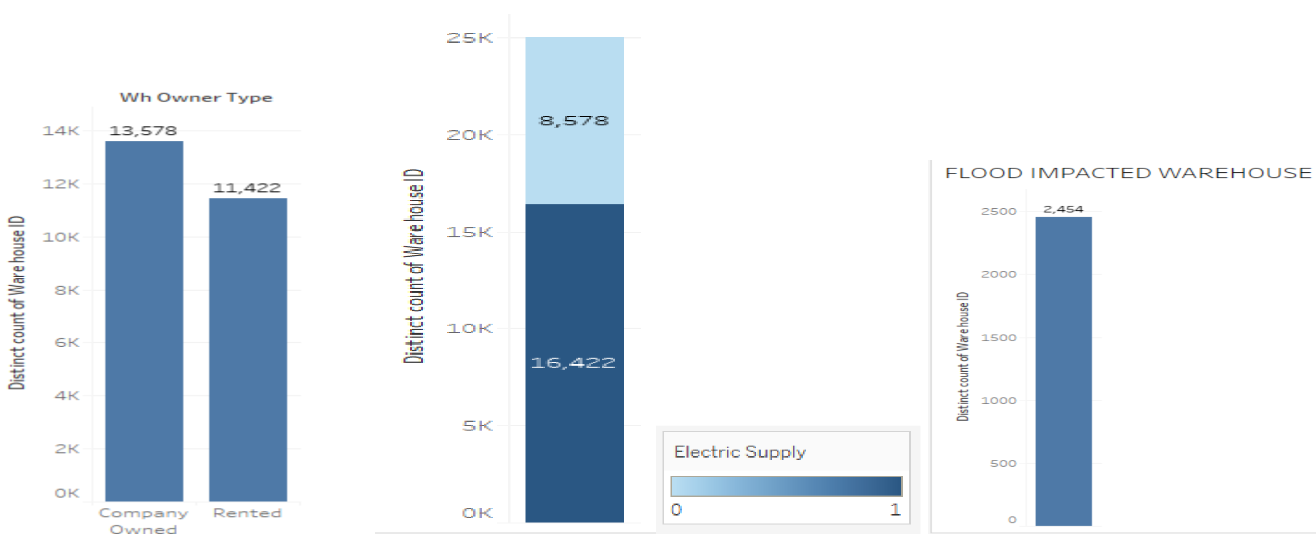






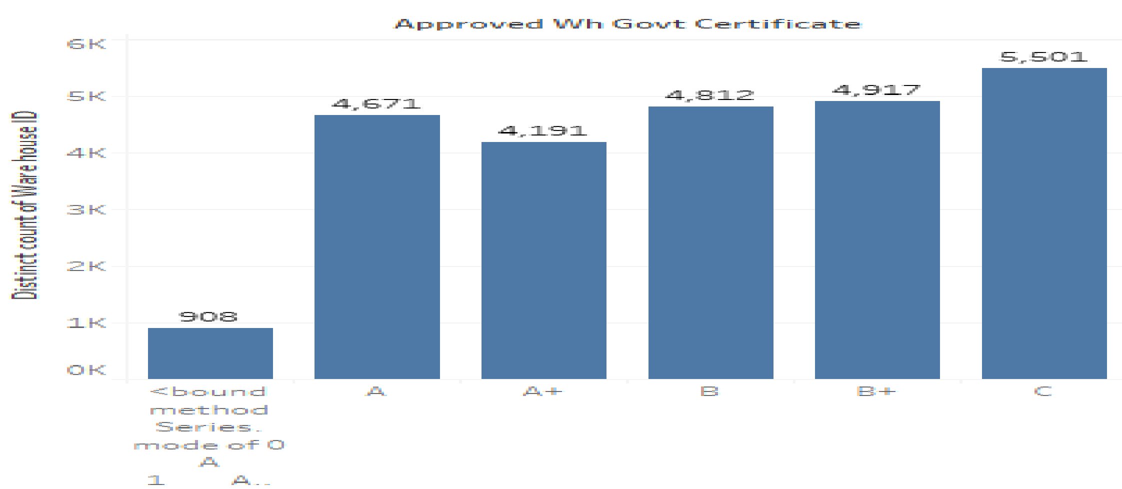
1.5 Avg transport issues & No. of Retail shops

- Average number of Transport issues reported more in Urban than Rural areas in last 1 year.
- On an average , Rural area warehouses supplied more to the retail shops nearby.
- More number of competitors available for Urban warehouses than Rural Warehouses.



1.6 Electric supply & Flood impacted WH

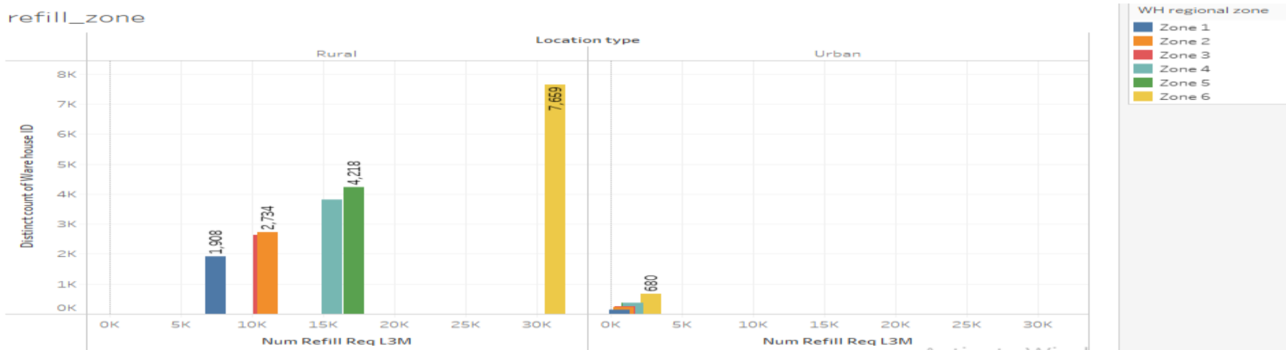
- Among the 25000 warehouse, only 13578 was maintained by Company itself.
- Only 16422 warehouses having alternate electric supply.
- 2454 Warehouses are Flood impacted.(9.8%)
- Only 7582 warehouses have Temparature Regulatory Mechanism.(3.03%)
- No storage issues reported only in 908 Warehouses and highest number of storage issues reported in 156 Warehouses.



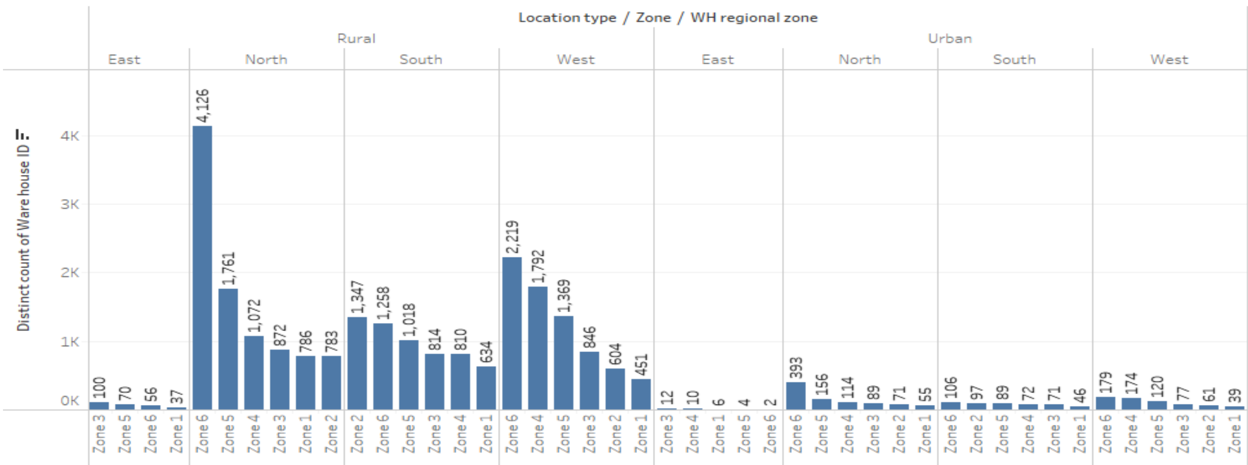
More number of warehouses got C Govt certificate (22%)

BIVARIATE ANALYSIS :

Done using TABLEAU Tool.

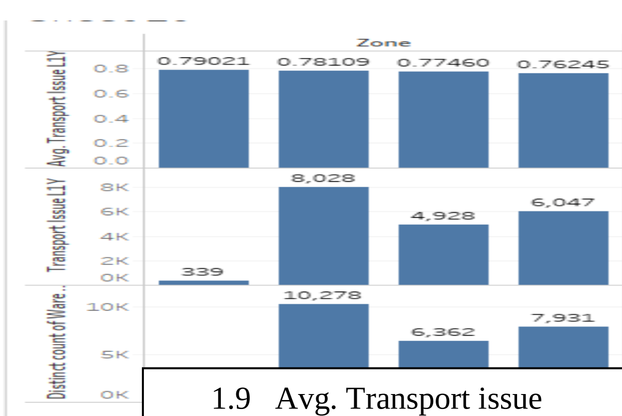


1.7 Refill zones

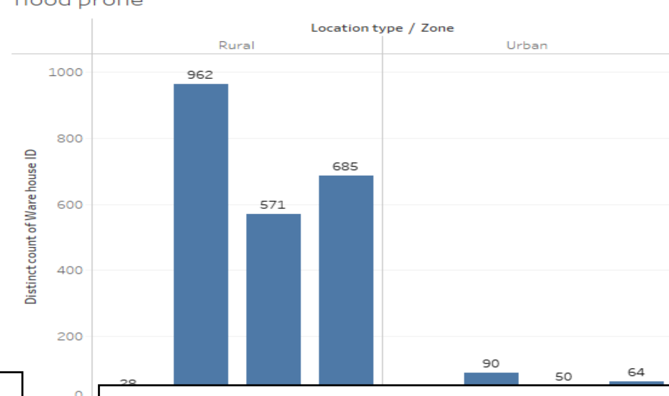


1.8 No of warehouse in each Regional zone

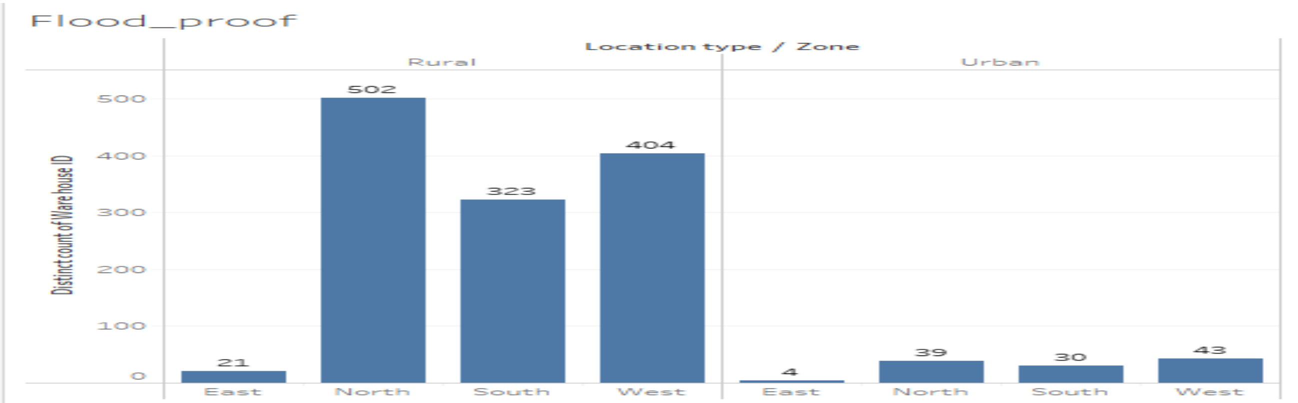
flood prone



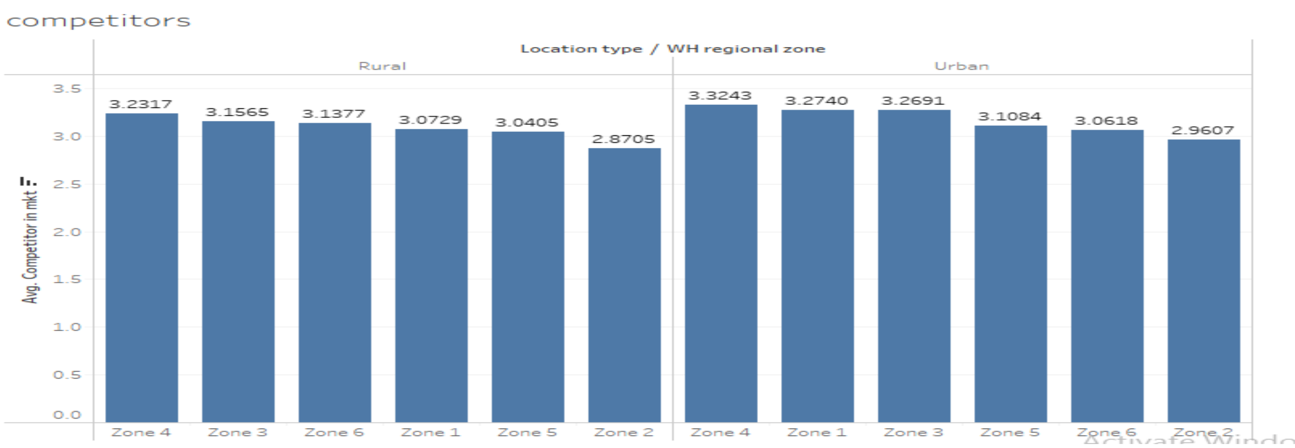
1.9 Avg. Transport issue



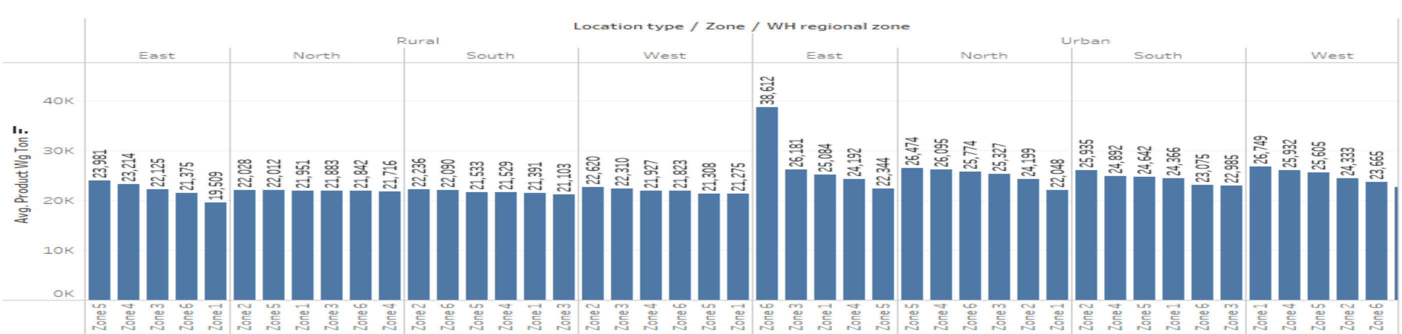
1.10 No. of warehouse in rural and urban



1.11 Flood proof zones



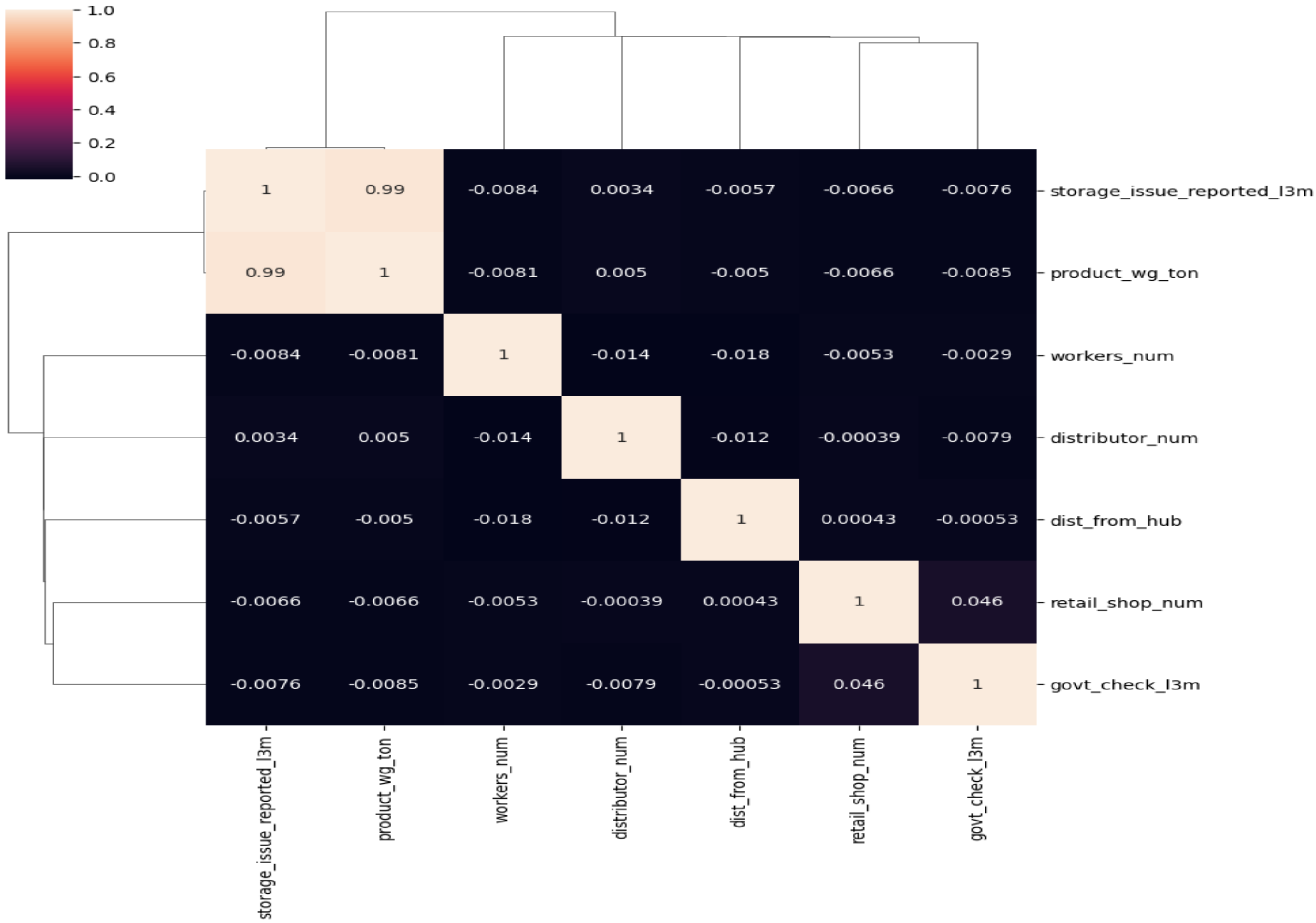
1.12 Competitors in mkt



- Most number of times i.e) 8 times refill occurred in Zone 6 of Rural areas and Urban areas.
- Most number of warehouses present in Zone 6 of North in both Rural (4126) and Urban areas (393).
- On an Average, more number of Transport issues reported in East zone for past 1 year.
- On an avg, more number of competitors present in zone 4 in both Rural and Urban areas.
- More number of Warehouses in North zone (10.25%)in Rural and East zone (11.76%) in Urban are flood impacted zones.
- Maximum of West zone (5.55%) in Rural and East zone (11.76%) in Urban are flood proof zones.
- On an avarage , more amount of products sold in Rural East zone 4 and Urban East zone 6.
- Warehouse having more storage issues ,have frequent shipping of products in last 3 months due of lack of proper storage facilities.This causes unwanted loss of products and increase transport charges.

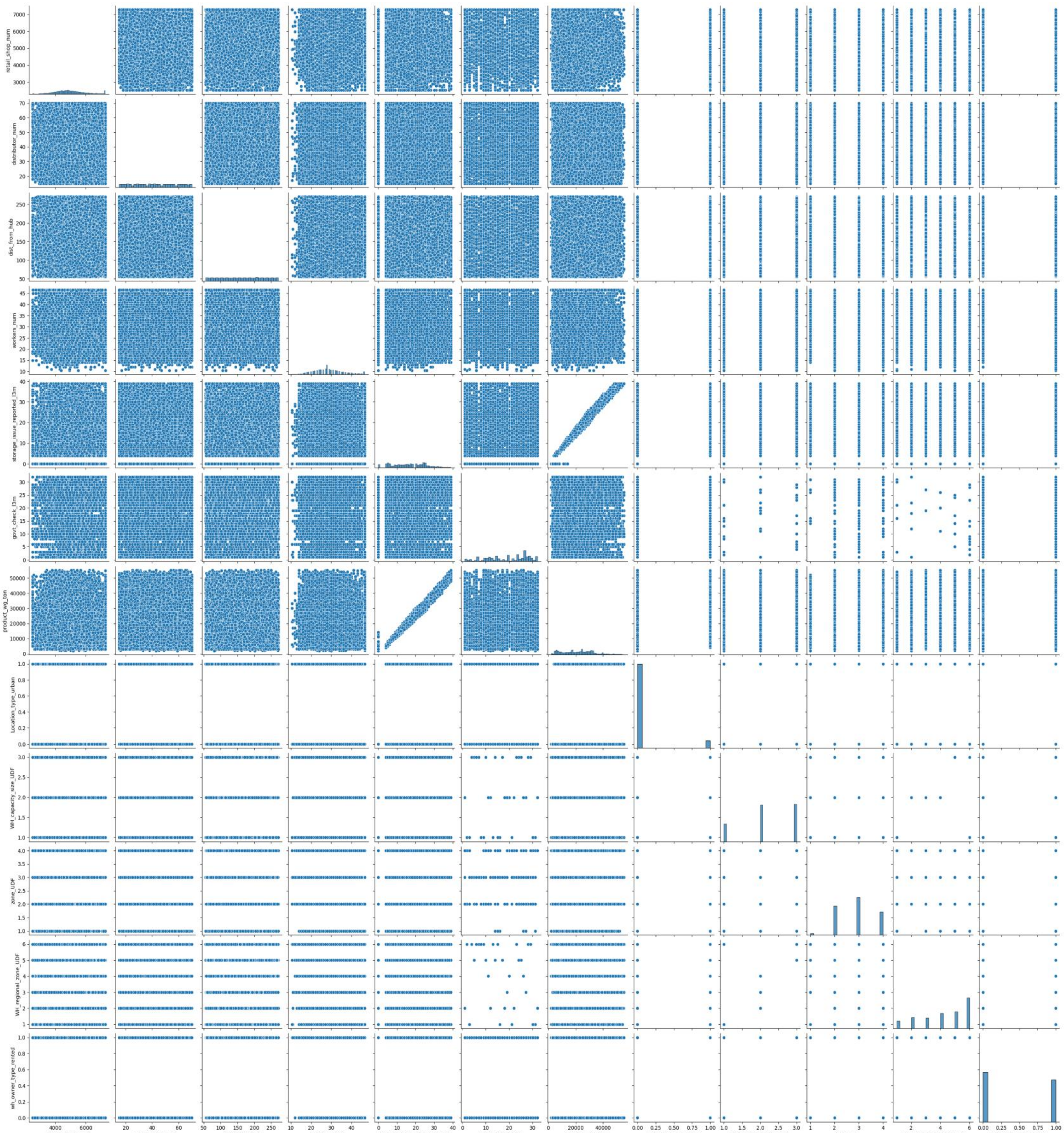
MULTI-VARIATE ANALYSIS :

CORELATION :



1.14 Cluster map

PAIRPLOT :



1.15 Pairplot

ANALYSIS :

- Warehouse having more storage issues ,have frequent shipping of products in last 3 months due of lack of proper storage facilities.This causes unwanted loss of products and increase transportation charges.
- More number of products sold in Rural East zone 4 and Urban East zone 6
- Warehouse WH_101568 present in zone 6 of East zone in Rural areas had highest number (12) of competitors in the market and also it had a maximum of 8 times refilled in the last 3 months. But it is a small sized warehouse so it is better if we have to change it to Large sized to avoid transportation charges.
- Urban East zone more prone to flood
- On an avg, product_wg_ton supplied more in Zone 6 of East region in Urban areas.
- If number of storage issues increased , product_wg_ton of the respective warehouse also increases.
- From the map we can say that the wh_capacity_size and wh_regional_zone are correlated with each other.

    - Zone 1 has only SMALL sized warehouse
    - Zone 2 , zone 3, zone 4 has MID sized warehouse

    - Zone 5 has LARGE sized warehouse
    - Zone 6 has both LARGE and SMALL sized warehouse.

- There is no zone 2 in the East region.
- On an average , Urban WH are highly prone to flood especially East zone(11.7%).
- Zone 4 in both Rural & Urban has more competitors.
- 15095 (65.75%) of Rural WH and 1327 (64.95%) of Urban WH have alternate electric supply.
- More number of Storage issues reported from Urban areas.
- Maximum of WH are C type Government certified.

- 2970 warehouse had a maximum of 8 times refilled in the last 3 months.
    - Rural : 2701 warehouse (11.7%)
    - Urban : 269 warehouse (13.16%)

  Among the 2970, 544 WH are Small and 2426 WH are Mid so we can convert them to Large sized to reduce transportation cost.
- 2912 warehouse had 0 times refilled.

  Among them
    - Rural – 533 WH
    - Urban – 41 WH are small sized but 0 refill in last 3 months so we can reduce the quantity of products shipped there.

1.3 DATA CLEANING & DATA PRE-PROCESSING :

MISSING VALUE TREATMENT :

- Presence of missing values in 2 columns
  - Workers_num : 990 (3.9%)
  - Approved_wh_govt_certificate : 908 (3.6%)
  - wh_est_year :11881 (47.5%)
- Due to negligible amount of missing values, we can impute them.
- wh_est_year having 47.5 % missing value , so we have to drop the column.
- Workers_num is a Numerical column so we have to replace missing values using MEDIAN.
- Approved_wh_govt_certificate is a Categorical column so we have to replace missing values using MODE.

Before treatment

```
Ware_house_ID                     0
WH_Manager_ID                     0
Location_type                     0
WH_capacity_size                  0
zone                              0
WH_regional_zone                  0
num_refill_req_l3m                0
transport_issue_l1y               0
Competitor_in_mkt                 0
retail_shop_num                   0
wh_owner_type                     0
distributor_num                   0
flood_impacted                    0
flood_proof                       0
electric_supply                   0
dist_from_hub                     0
workers_num                     990
storage_issue_reported_l3m        0
temp_reg_mach                     0
approved_wh_govt_certificate    908
wh_breakdown_l3m                  0
govt_check_l3m                    0
product_wg_ton                    0
dtype: int64
```

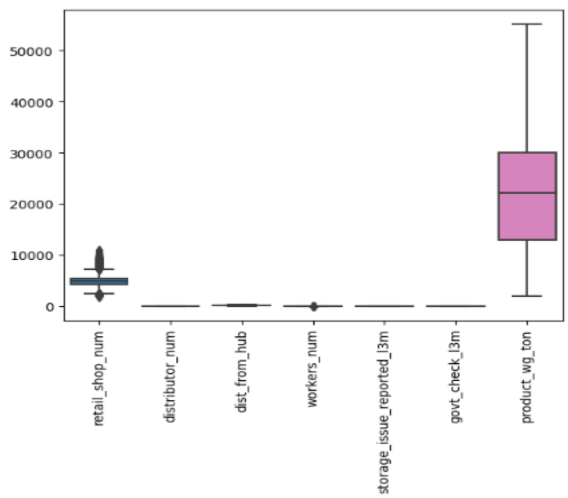
After treatment

```
Ware_house_ID                   0
WH_Manager_ID                   0
Location_type                   0
WH_capacity_size                0
zone                            0
WH_regional_zone                0
num_refill_req_l3m              0
transport_issue_l1y             0
Competitor_in_mkt               0
retail_shop_num                 0
wh_owner_type                   0
distributor_num                 0
flood_impacted                  0
flood_proof                     0
electric_supply                 0
dist_from_hub                   0
workers_num                     0
storage_issue_reported_l3m      0
temp_reg_mach                   0
approved_wh_govt_certificate    0
wh_breakdown_l3m                0
govt_check_l3m                  0
product_wg_ton                  0
dtype: int64
```

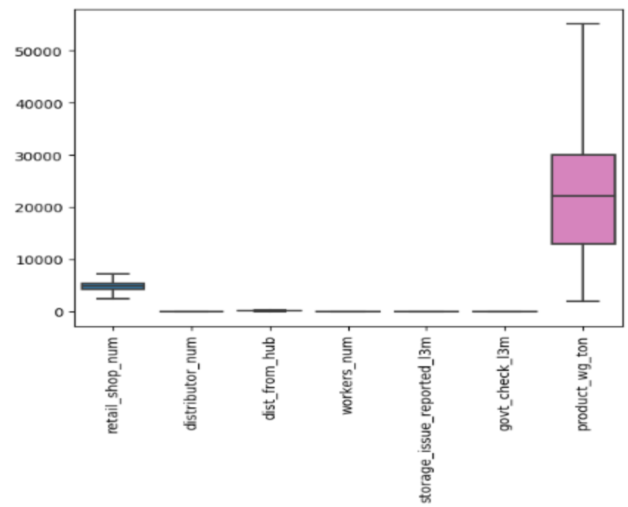
OUTLIER TREATMENT :

- Presence of outliers in 2 columns
  - Retail_shop_num
  - Workers_num
- To improve accuracy and better visualization, We have to remove them using BOXPLOT Method.

Before treatment



After treatment



VARIABLE TRANSFORMATION :

- 3 variables can be converted from Numerical into Categorical (since they are predefined) so finally we got 11 Categorical and only 13 Numerical variables.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 24 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Ware_house_ID                 25000 non-null  object
 1   WH_Manager_ID                 25000 non-null  object
 2   Location_type                 25000 non-null  object
 3   WH_capacity_size              25000 non-null  object
 4   zone                          25000 non-null  object
 5   WH_regional_zone              25000 non-null  object
 6   num_refill_req_l3m            25000 non-null  int64
 7   transport_issue_l1y           25000 non-null  int64
 8   Competitor_in_mkt             25000 non-null  int64
 9   retail_shop_num               25000 non-null  int64
 10  wh_owner_type                 25000 non-null  object
 11  distributor_num               25000 non-null  int64
 12  flood_impacted                25000 non-null  category
 13  flood_proof                   25000 non-null  category
 14  electric_supply               25000 non-null  category
 15  dist_from_hub                 25000 non-null  int64
 16  workers_num                   24010 non-null  float64
 17  wh_est_year                   13119 non-null  float64
 18  storage_issue_reported_l3m    25000 non-null  int64
 19  temp_reg_mach                 25000 non-null  int64
 20  approved_wh_govt_certificate  24092 non-null  object
 21  wh_breakdown_l3m              25000 non-null  int64
 22  govt_check_l3m                25000 non-null  int64
 23  product_wg_ton                25000 non-null  int64
dtypes: category(3), float64(2), int64(11), object(8)
memory usage: 4.1+ MB
```

- Skewness and kurtosis :

```
scm.skew()
```

```
retail_shop_num               0.435217
distributor_num               0.015213
dist_from_hub                -0.005999
workers_num                   0.453899
storage_issue_reported_l3m    0.113345
govt_check_l3m               -0.363262
product_wg_ton                0.331631
dtype: float64
```

ALL THE COLUMN HAVING SKEW VALUE RANGES FROM -1 TO +1. SO NO NEED TO DO TRANSFORMATION

```
scm.kurtosis()
```

```
retail_shop_num               0.073453
distributor_num              -1.187564
dist_from_hub                -1.200682
workers_num                  -0.065361
storage_issue_reported_l3m   -0.680142
govt_check_l3m               -1.057342
product_wg_ton               -0.502022
dtype: float64
```

REMOVAL OF UNWANTED VARIABLES :

- Presence of 11881 missing values in **wh_est_year** i.e) 47.5% so imputation doesnot give perfect results so we can remove the column using drop() function.
- Finally we have only 25000 rows and 23 columns to proceed the model building.

	Ware_house_ID	WH_Manager_ID	Location_type	WH_capacity_size	zone	WH_regional_zone	num_refill_req_l3m	transport_issue_l1y	Competitor_in_mkt
0	WH_100000	EID_50000	Urban	Small	West	Zone 6	3	1	2
1	WH_100001	EID_50001	Rural	Large	North	Zone 5	0	0	4
2	WH_100002	EID_50002	Rural	Mid	South	Zone 2	1	0	4
3	WH_100003	EID_50003	Rural	Mid	North	Zone 3	7	4	2
4	WH_100004	EID_50004	Rural	Large	North	Zone 5	3	1	2
...	...	...	...	...	...	...	...	...	...
24995	WH_124995	EID_74995	Rural	Small	North	Zone 1	3	0	4
24996	WH_124996	EID_74996	Rural	Mid	West	Zone 2	6	0	4
24997	WH_124997	EID_74997	Urban	Large	South	Zone 5	7	0	2
24998	WH_124998	EID_74998	Rural	Small	North	Zone 1	1	0	2
24999	WH_124999	EID_74999	Rural	Mid	West	Zone 4	8	2	4

25000 rows × 23 columns

ADDITION OF NEW VARIABLES :

- Encoding removes redundancies from data, size of the files will be a lot smaller. This results in faster input speed when data is saved.
- Since encoded data is smaller in size, we should be able to save space on storage devices. This is ideal if you have large amounts of data that need to be archived.
- Here I used USER DEFINED ENCODING .

- Addition of new variables such as
  - WH_capacity_size_UDF

    Large':3,'Mid':2,'Small':1

  - zone_UDF

    East':1,'West':2,'North':3,'South':4

  - WH_regional_zone_UDF

    Zone 1':1,'Zone 2':2,'Zone 3':3,

    'Zone 4':4,'Zone 5':5,'Zone 6':6

  - wh_owner_type_rent Company Owned':0,'Rented':1
  - approved_wh_govt_certificate_UDF

    '<bound method Series.mode

    of 01':0,'A+':1,'A':2,'B+':3,'B':4,'C':5

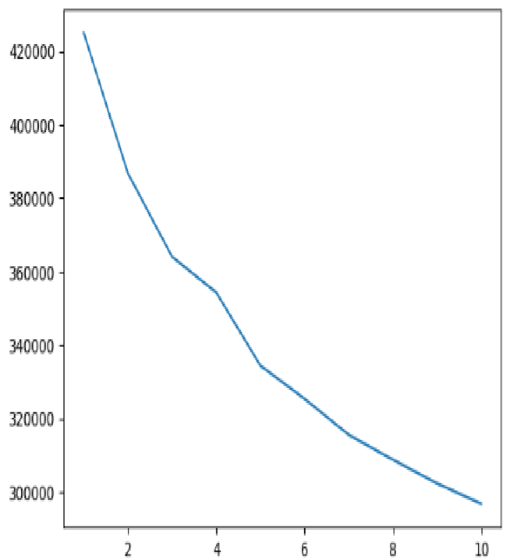
  - Location_type_urban Urban':1,'Rural':0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 29 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Ware_house_ID                     25000 non-null  object
 1   WH_Manager_ID                     25000 non-null  object
 2   Location_type                     25000 non-null  object
 3   WH_capacity_size                  25000 non-null  object
 4   zone                              25000 non-null  object
 5   WH_regional_zone                  25000 non-null  object
 6   num_refill_req_l3m                25000 non-null  category
 7   transport_issue_l1y               25000 non-null  category
 8   Competitor_in_mkt                 25000 non-null  category
 9   retail_shop_num                   25000 non-null  float64
 10  wh_owner_type                     25000 non-null  object
 11  distributor_num                   25000 non-null  int64
 12  flood_impacted                    25000 non-null  category
 13  flood_proof                       25000 non-null  category
 14  electric_supply                   25000 non-null  category
 15  dist_from_hub                     25000 non-null  int64
 16  workers_num                       25000 non-null  float64
 17  storage_issue_reported_l3m        25000 non-null  int64
 18  temp_reg_mach                     25000 non-null  category
 19  approved_wh_govt_certificate      25000 non-null  object
 20  wh_breakdown_l3m                  25000 non-null  category
 21  govt_check_l3m                    25000 non-null  int64
 22  product_wg_ton                    25000 non-null  int64
 23  Location_type_urban               25000 non-null  int64
 24  WH_capacity_size_UDF              25000 non-null  int64
 25  zone_UDF                          25000 non-null  int64
 26  WH_regional_zone_UDF              25000 non-null  int64
 27  wh_owner_type_rented              25000 non-null  int64
 28  approved_wh_govt_certificate_UDF  25000 non-null  object
dtypes: category(8), float64(2), int64(10), object(9)
memory usage: 4.2+ MB
```

Finally we got 29 columns with 25000 rows.

CLUSTERING :

- Here we have to use K-Means clustering. K-means clustering is now widely used in machine learning to partition data points into K clusters based on their similarity.
- The goal is to minimize the sum of squared distances between the data points and their corresponding cluster centroids, resulting in clusters that are internally homogeneous and distinct from each other.



Here we have to take k=7 and adding a column as Demand labels. At the output , we have 7 clusters.

Taking median of product wg_ton for respective Demand labels.

Demand_labels	Qty_needed
0	22093.0
1	30092.0
2	30095.0
3	7146.0
4	21114.0
5	17061.0
6	25076.0

_ton	Location_type_urban	WH_capacity_size_UDF	zone_UDF	WH_regional_zone_UDF	wh_owner_type_rented	approved_wh_govt_certificate_UDF	Demand_labels
7115	1	1	2	6	1	2	6
5074	0	3	3	5	0	2	3
3137	0	2	4	2	0	2	4
2115	0	2	3	3	1	1	5
4071	0	3	3	5	0	5	2

TABLEAU LINK FOR EDA :

https://public.tableau.com/app/profile/abinaya.m8348/viz/capstone_1_16965341987370/Sheet16

MODEL BUILDING

BUILD VARIOUS MODELS :

```
num_refill_req_l3m            int64
transport_issue_l1y           int64
Competitor_in_mkt             int64
retail_shop_num             float64
distributor_num               int64
dist_from_hub                 int64
workers_num                 float64
storage_issue_reported_l3m    int64
temp_reg_mach                 int64
wh_breakdown_l3m              int64
govt_check_l3m                int64
product_wg_ton                int64
Location_type_urban           int64
WH_capacity_size_UDF          int64
zone_UDF                      int64
WH_regional_zone_UDF          int64
wh_owner_type_rented          int64
Demand_labels                 int32
dtype: object
```

Test Train split :

We have to split the train and test dataset in 70:30 ratio.

```
x_train,x_test , y_train, y_test = train_test_split(x,y,test_size = .30 ,random_state = 0)
```

This is a multi classification model so we can use Naïve Bayes , KNN , Decision Tree, SVC Models.

✓ **DECISION TREE CLASSIFIER :**

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
]: print(dTree.score(x_train, y_train))
   print(dTree.score(x_test, y_test))

1.0
0.9192
```

The decision tree score will be overfitting so here we have to choose Gini criterion , maximum depth = 7 to limit the branches to get good accuracy.

TRAIN DATASET

```
[[1473  110  276  156  134   20    0]
 [  61 2396    0  110   26   13    0]
 [ 232    0 3038   63   73   17    0]
 [ 135   26  180 2914   70   14    0]
 [  34   97  240  202 2000   86    0]
 [  70   32  125   47   25 1634    0]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       0.73      0.68      0.71      2169
           1       0.90      0.92      0.91      2606
           2       0.79      0.89      0.83      3423
           3       0.83      0.87      0.85      3339
           4       0.86      0.75      0.80      2659
           5       0.92      0.85      0.88      1933
           6       1.00      1.00      1.00      1371

    accuracy                           0.85     17500
   macro avg       0.86      0.85      0.85     17500
weighted avg       0.85      0.85      0.85     17500
```

TEST DATASET

```
[[ 660   47   98   57   39   16    1]
 [  25 1033    0   59   14    4    0]
 [  92    0 1296   39   38    3    0]
 [  66   13   75 1228   27    7    0]
 [  14   50  117   87  804   32    0]
 [  38   16   47   20   10  664    1]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       0.74      0.72      0.73       918
           1       0.89      0.91      0.90      1135
           2       0.79      0.88      0.84      1468
           3       0.82      0.87      0.85      1416
           4       0.86      0.73      0.79      1104
           5       0.91      0.83      0.87       796
           6       1.00      1.00      1.00       663

    accuracy                           0.85      7500
   macro avg       0.86      0.85      0.85      7500
weighted avg       0.85      0.85      0.85      7500
```

```
ROC-AUC SCORE FOR TRAIN: 0.9708318196049958

ROC-AUC SCORE FOR Test: 0.9694609072078721
```

Here we got 85 % accuracy for both Test and Train dataset while using Decision Tree Classifier and we got good F1 score (above 70%).

NAÏVE BAYES :

```
nb=GaussianNB()
nb.fit(x_train,y_train)
```

```
GaussianNB()
```

Confusion Matrix & Classification report :

TRAIN DATA

```
[[  21  354 1697   79   11    3    4]
 [   0 2606    0    0    0    0    0]
 [   0    0 3423    0    0    0    0]
 [   0  689 1278 1366    6    0    0]
 [   0  538 1679   44  398    0    0]
 [   0  590  733  136  140  333    1]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       1.00      0.01      0.02      2169
           1       0.55      1.00      0.71      2606
           2       0.39      1.00      0.56      3423
           3       0.84      0.41      0.55      3339
           4       0.72      0.15      0.25      2659
           5       0.99      0.17      0.29      1933
           6       1.00      1.00      1.00      1371

    accuracy                           0.54     17500
   macro avg       0.78      0.53      0.48     17500
weighted avg       0.74      0.54      0.47     17500
```

TEST DATA

```
[[  14  143  715   41    3    1    1]
 [   0 1135    0    0    0    0    0]
 [   0    0 1468    0    0    0    0]
 [   0  308  543  564    1    0    0]
 [   0  221  705   19  159    0    0]
 [   0  241  270   59   53  170    3]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       1.00      0.02      0.03       918
           1       0.55      1.00      0.71      1135
           2       0.40      1.00      0.57      1468
           3       0.83      0.40      0.54      1416
           4       0.74      0.14      0.24      1104
           5       0.99      0.21      0.35       796
           6       0.99      1.00      1.00       663

    accuracy                           0.56      7500
   macro avg       0.79      0.54      0.49      7500
weighted avg       0.74      0.56      0.49      7500
```

```
ROC-AUC SCORE FOR TRAIN: 0.9609613010552591
```

```
ROC-AUC SCORE FOR Test: 0.9610786949372148
```

Here the accuracy will be 54 % for train & 56% test data Roc- Auc will be 0.9609 and 0.9610 for train and test data.

✓ **KNN Model :**

```
knn = KNeighborsClassifier(n_neighbors = 7)
knn = knn.fit(x_train,y_train)
```

Confusion Matrix & Classification report :

TRAIN DATA

```
[[1575  141  210  100  109   30    4]
 [  14 2533    0   16   32   11    0]
 [ 177    0 3132   17   85   12    0]
 [  69   68  104 3025   62   11    0]
 [  32   61   74   59 2423   10    0]
 [  50   77   59  120   78 1548    1]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       0.82      0.73      0.77      2169
           1       0.88      0.97      0.92      2606
           2       0.88      0.91      0.89      3423
           3       0.91      0.91      0.91      3339
           4       0.87      0.91      0.89      2659
           5       0.95      0.80      0.87      1933
           6       1.00      1.00      1.00      1371

    accuracy                           0.89     17500
   macro avg       0.90      0.89      0.89     17500
weighted avg       0.89      0.89      0.89     17500
```

TEST DATA

```
[[ 587   70  128   66   49   17    1]
 [  13 1097    0    9    9    7    0]
 [  93    0 1313   18   37    7    0]
 [  51   38   61 1225   28   13    0]
 [  19   41   48   40  945   11    0]
 [  36   47   43   69   32  566    3]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       0.73      0.64      0.68       918
           1       0.85      0.97      0.90      1135
           2       0.82      0.89      0.86      1468
           3       0.86      0.87      0.86      1416
           4       0.86      0.86      0.86      1104
           5       0.91      0.71      0.80       796
           6       0.99      1.00      1.00       663

    accuracy                           0.85      7500
   macro avg       0.86      0.85      0.85      7500
weighted avg       0.85      0.85      0.85      7500
```

```
ROC-AUC SCORE FOR TRAIN: 0.992233716668591
```

```
ROC-AUC SCORE FOR Test: 0.9788979685106006
```

Here the accuracy will be 89 % for train and 85 % for test data and Roc-Auc score will be 0.9922 and 0.9788 for train and test dataset.It has good accuracy and F1 score so we can say this would be a better model.

✓ **SVC :**

```
SVC
SVC(kernel='linear', random_state=0)
```

Confusion Matrix & Classification report :

TRAIN DATA

```
[[2101   12   19   15    9    9    4]
 [   7 2582    0    7    7    3    0]
 [  17    0 3385    8    5    8    0]
 [   5    2   16 3299   10    7    0]
 [   9    5    9    2 2631    3    0]
 [   1    6    2    0    5 1918    1]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      2169
           1       0.99      0.99      0.99      2606
           2       0.99      0.99      0.99      3423
           3       0.99      0.99      0.99      3339
           4       0.99      0.99      0.99      2659
           5       0.98      0.99      0.99      1933
           6       1.00      1.00      1.00      1371

    accuracy                           0.99     17500
   macro avg       0.99      0.99      0.99     17500
weighted avg       0.99      0.99      0.99     17500
```

TEST DATA

```
[[ 882    7   10    7    5    6    1]
 [   4 1117    0    7    1    6    0]
 [   6    0 1455    2    0    5    0]
 [   2    4    3 1393    7    7    0]
 [   5    3    8    2 1081    5    0]
 [   1    3    2    4    5  778    3]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       918
           1       0.99      0.98      0.98      1135
           2       0.98      0.99      0.99      1468
           3       0.98      0.98      0.98      1416
           4       0.98      0.98      0.98      1104
           5       0.96      0.98      0.97       796
           6       0.99      1.00      1.00       663

    accuracy                           0.98      7500
   macro avg       0.98      0.98      0.98      7500
weighted avg       0.98      0.98      0.98      7500
```

```
ROC-AUC SCORE FOR TRAIN: 0.990992878113669
```

```
ROC-AUC SCORE FOR Test: 0.9904773277776889
```

The SVC model gave accuracy of train and test data will be 99 % and 98 %.The Roc – Auc score will be 0.99 and 0.99 for both the dataset.This would be a overfitting model.

INTERPRETATION :

From the above discussed model, we can say that **KNN model** will give better accuracy and F1 score.

MODEL TUNNING :

✓ ENSEMBLE TECHNIQUE :

❖ RANDOM FOREST CASSIFIER :

```
rfcl = RandomForestClassifier(n_estimators = 100, random_state=0,max_features=12)
rfcl = rfcl.fit(x_train, y_train)
```

Confusion Matrix & Classification report :

TRAIN DATA

```
[[2169    0    0    0    0    0    0]
 [   0 2606    0    0    0    0    0]
 [   0    0 3423    0    0    0    0]
 [   0    0    0 3339    0    0    0]
 [   0    0    0    0 2659    0    0]
 [   0    0    0    0    0 1933    0]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2169
           1       1.00      1.00      1.00      2606
           2       1.00      1.00      1.00      3423
           3       1.00      1.00      1.00      3339
           4       1.00      1.00      1.00      2659
           5       1.00      1.00      1.00      1933
           6       1.00      1.00      1.00      1371

    accuracy                           1.00     17500
   macro avg       1.00      1.00      1.00     17500
weighted avg       1.00      1.00      1.00     17500
```

TEST DATA

```
[[ 795   20   26   42   13   21    1]
 [   8 1097    0   13    8    9    0]
 [  22    0 1412    7   20    7    0]
 [  13   10   20 1354   13    6    0]
 [  11   16   28   16 1025    8    0]
 [  11    5   12    8   10  747    3]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       0.92      0.87      0.89       918
           1       0.96      0.97      0.96      1135
           2       0.94      0.96      0.95      1468
           3       0.94      0.96      0.95      1416
           4       0.94      0.93      0.93      1104
           5       0.94      0.94      0.94       796
           6       0.99      1.00      1.00       663

    accuracy                           0.95      7500
   macro avg       0.95      0.95      0.95      7500
weighted avg       0.95      0.95      0.95      7500
```

```
ROC-AUC SCORE FOR TRAIN: 1.0
```

```
ROC-AUC SCORE FOR Test: 0.997831071803252
```

Because of 100% and 95% accuracy, we can say this model is overfiting. So we have to neglect it.

❖ **BAGGING :**

```
bgcl = BaggingClassifier(base_estimator=dtR, n_estimators=100,random_state=0)
bgcl = bgcl.fit(x_train, y_train)
```

Confusion Matrix & Classification report :

TRAIN DATA

```
[[2168    0    1    0    0    0    0]
 [   0 2606    0    0    0    0    0]
 [   0    0 3423    0    0    0    0]
 [   0    0    0 3339    0    0    0]
 [   1    0    0    0 2658    0    0]
 [   0    0    0    0    0 1933    0]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2169
           1       1.00      1.00      1.00      2606
           2       1.00      1.00      1.00      3423
           3       1.00      1.00      1.00      3339
           4       1.00      1.00      1.00      2659
           5       1.00      1.00      1.00      1933
           6       1.00      1.00      1.00      1371

    accuracy                           1.00     17500
   macro avg       1.00      1.00      1.00     17500
weighted avg       1.00      1.00      1.00     17500
```

TEST DATA

```
[[ 790   26   28   39   13   21    1]
 [   8 1103    0   11   10    3    0]
 [  36    0 1397    6   21    8    0]
 [  23   10   14 1344   20    5    0]
 [  12   24   36   16 1008    8    0]
 [  15    9   10    8   11  741    2]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       0.89      0.86      0.88       918
           1       0.94      0.97      0.96      1135
           2       0.94      0.95      0.95      1468
           3       0.94      0.95      0.95      1416
           4       0.93      0.91      0.92      1104
           5       0.94      0.93      0.94       796
           6       1.00      1.00      1.00       663

    accuracy                           0.94      7500
   macro avg       0.94      0.94      0.94      7500
weighted avg       0.94      0.94      0.94      7500
```

```
ROC-AUC SCORE FOR TRAIN: 0.9999999847265554
```

```
ROC-AUC SCORE FOR Test: 0.9957198851316391
```

The accuracy for Train and Test will be 100% and 94% for bagging model and so we considered this as Overfitting model.So we neglect it.

❖ GRADIENT BOOSTING :

```
gbcl = GradientBoostingClassifier(n_estimators = 50,random_state=0)
gbcl = gbcl.fit(x_train, y_train)
```

Confusion Matrix & Classification report :

TRAIN DATA

```
[[2039   18   54   25   18   15    0]
 [   1 2586    0    7    9    3    0]
 [  17    0 3367    7   23    9    0]
 [  11    4   25 3263   17   19    0]
 [   3   22   44   27 2546   17    0]
 [   8    5    6   12    7 1895    0]
 [   0    0    0    0    0    0 1371]]
              precision    recall  f1-score   support

           0       0.98      0.94      0.96      2169
           1       0.98      0.99      0.99      2606
           2       0.96      0.98      0.97      3423
           3       0.98      0.98      0.98      3339
           4       0.97      0.96      0.96      2659
           5       0.97      0.98      0.97      1933
           6       1.00      1.00      1.00      1371

    accuracy                           0.98     17500
   macro avg       0.98      0.98      0.98     17500
weighted avg       0.98      0.98      0.98     17500
```

TEST DATA

```
[[ 841   12   28   18    5   13    1]
 [   5 1111    0    8    6    5    0]
 [   7    0 1436    4   16    5    0]
 [   9    2   14 1375   11    5    0]
 [   2   11   27   13 1041   10    0]
 [   9    4    4    5    4  767    3]
 [   0    0    0    0    0    0  663]]
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       918
           1       0.97      0.98      0.98      1135
           2       0.95      0.98      0.96      1468
           3       0.97      0.97      0.97      1416
           4       0.96      0.94      0.95      1104
           5       0.95      0.96      0.96       796
           6       0.99      1.00      1.00       663

    accuracy                           0.96      7500
   macro avg       0.97      0.96      0.97      7500
weighted avg       0.96      0.96      0.96      7500
```

```
ROC-AUC SCORE FOR TRAIN: 0.9995696405247659
```

```
ROC-AUC SCORE FOR Test: 0.9990860784133605
```

In Model Tuning , the Gradient boosting gives better accuracy i.e) 98% Train and 96 % Test Accuracy and better F1 score.

MODELS USED :

MODELS	TRAIN DATA		TEST DATA	
	ACCURACY	ROC-AUC	ACCURACY	ROC-AUC
DECISION TREE	0.85	0.97	0.85	0.96
SVC	0.99	0.99	0.98	0.99
KNN	0.89	0.99	0.85	0.98
NB	0.54	0.96	0.56	0.96
RFC	1.00	1.00	0.95	0.997
BG	1.00	0.99	0.94	0.995
GB	0.98	0.995	0.96	0.999

1 .1 Performance metrics

From the above model , we have to choose Gradient Boosting because it has good accuracy (96%) and good Roc –Auc score (99%).

EFFORTS TO IMPROVE MODEL PERFORMANCE :

- Perform Scaling (Min-Max scaling) and remove outliers.
- Taken only Numerical variables and find out multi-collinearity among them.
- Perform User defined encoding to make categorical into numerical variable.
- Using Ensemble technique (Gradient Boosting) to improve accuracy.

MODEL VALIDATION :

- Initially KNN model will be a better one because it has 85% accuracy on Test data.
- To improve its performance we have to use Model tuning (Ensemble Technique).
- Among Ensemble technique , **Gradient Boosting** will be a better one. It provides 96 % Accuracy and good Roc-Auc score.
- Gradient boosting :
  - From confusion matrics , it shows 7234 out of 7500 True positive results.
  - Accuracy – 96 %
  - F1 score – above 94%
  - Recall – above 92%
  - Precision – above 95%

So we choose this would be a better model when compared to others.

RECOMMENDATIONS :

- Less number of warehouse present in zone 6 of East region in Urban areas (2) but sales is higher in those region so we can open few warehouse there to increase sales and make them as flood proof.
- 2912 warehouse had 0 times refilled.

  Among them

  Rural – 533 WH

  Urban – 41 WH are small sized but 0 refill in last 3 months so we can reduce the quantity of products shipped there.

  To increase sales in those region we can provide offers to attract people.
- 2970 warehouse had a maximum of 8 times refilled in the last 3 months.

  Rural : 2701 warehouse (11.7%)

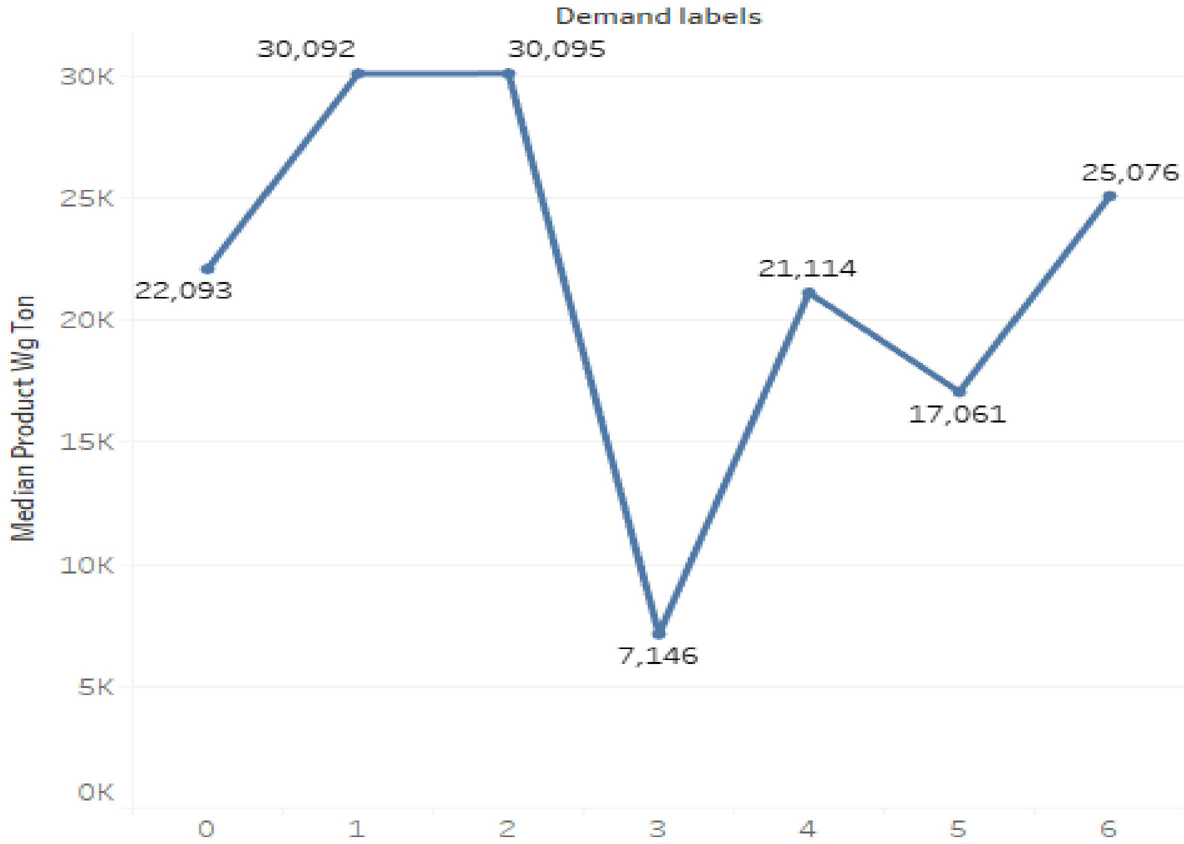
  Urban : 269 warehouse (13.16%)

  Among the 2970, 544 WH are Small and 2426 WH are Mid so we can convert them into Large sized to reduce transportation cost.
- WH_101568 present in Rural East zone 6 has more competitors and 8 times refill but of small sized so we can convert them into Large or Mid sized to reduce transportation cost.

- More amount of loss during storage and transportation occurred in Urban areas , this cause unwanted loss of products. So we have to reduce them to reduce inventory loss.
- More number of warehouses in Urban areas are prone to flood so we can make it flood proof to reduce wastage of products.
- More sales occurred in Urban areas so we can open warehouses there to increase sales
- Maximum breakdown occurred in Urban west zone so this can be reduced.

Demand label	**Location**	**Capacity**	**Zone**	**Regional zone**	**Qty needed (in thousands)**
0	Rural,Urban	Large,Mid,Small	N,E,W,S	1 to 6	22093
1	Rural	Large,Mid,Small	N,E,W,S	1 to 6	30092
2	Rural	Large,Mid,Small	N,E,W,S	1 to 6	30095
3	Rural	Large,Mid,Small	N,E,W,S	1 to 6	7145
4	Rural	Mid,Small	N,E,W,S	1 to 4,6	21114
5	Rural,Urban	Large,Mid,Small	N,E,W,S	1 to 6	17061
6	Urban	Large,Mid,Small	N,E,W,S	1 to 6	25076

1.2 Warehouse clusters



1.17 Warehouse clusters

THANK YOU