FINAL REPORT
CAPSTONE PROJECT-PGP-DSBA

Supply Chain Project

By: Parthasarathi Behura

Page Index

Introduction

What did you wish to achieve while doing the project ?

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.

- The objective of this exercise is to build a model, using historical data that will determine an optimum weight of the product to be shipped each time to the warehouse.
- Also try to analysis the demand pattern in different pockets of the country so management can drive the advertisement campaign particular in those pockets.

Challenges

- Misalignment between demand and supply across warehouses.
- High inventory costs due to improper stock management.
- Inefficient distribution strategies.

Project Obj.

- Analyze demand patterns.
- Building predictive model to optimize shipment quantities.
- Support targeted marketing campaigns.

Key Constraints

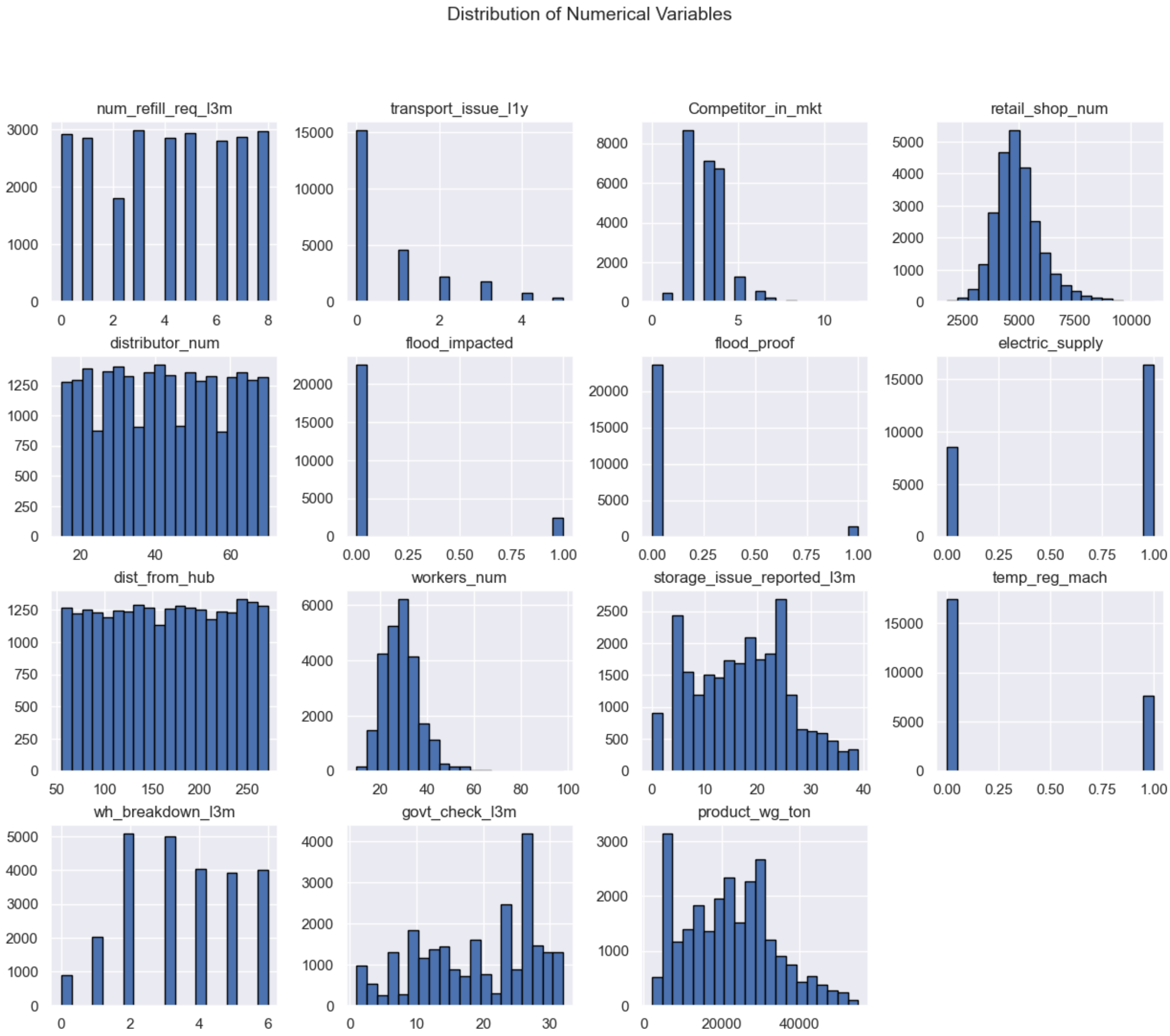
- Diverse warehouse locations (Rural vs. Urban)
- Varying warehouse capacities
- Regional distribution challenges

Understanding business/social opportunity:

- The objective of this exercise is to build a model, using historical data that will determine an optimum weight of the product to be shipped each time to the warehouse.
- Also try to analysis the demand pattern in different pockets of the country so management can drive the advertisement campaign particular in those pockets.

2. Exploratory Data Analysis: EDA - Univariate / Bivariate / Multivariate analysis to understand relationship between variables. - Both visual and non-visual understanding of the data.

Univariate Analysis:



Hist-plot of Numeric variables

Inferences:

Number of refill requests in last 3 months:

- ➢ Distribution is fairly uniform from 0 to 8.
- ➢ Warehouses are evenly spread in terms of refill frequency. No major skew, implying diverse operational needs.

Transport issues in last 1 year:

- ➢ Distribution is highly right-skewed.
- ➢ Most warehouses had 0–1 transport issues and very few faced repeated disruptions.

Number of competitors in market:

- Distribution is Slight right-skew.
- Most locations have 3–6 competitors; high competition could be a contributing factor in refill logistics.

retail_shop_num:

- Distribution is normally distributed around 6000.
- Most warehouses serve mid-size markets. Very few serve extremely large or small markets.

distributor_num:

This is uniform and a balanced penetration in the market.

flood_impacted:

Very few warehouses are flood-impacted. But those that are might pose high refill risk.

flood_proof:

Very few are flood-proof, increasing vulnerability in disaster-prone areas.

electric_supply:

Majority of warehouses have electricity, which is good for cold storage and operations.

Distance from distribution hub:

It is uniform. Warehouses are spread across various distances. Logistics must cater to both near and remote sites.

workers_num:

Very less number of stores are having larger workforce. Other wise there are a very small number of workers in majority of the stores.

storage_issue_reported_l3m:

Most warehouses reported 0–5 issues; very few had high storage problems, which can affect restocking.

Temperature regulation machine availability:

Cold storage is common, good for sensitive products. Few warehouses lack this, potentially increasing spoilage/refill needs.

Warehouse breakdowns in last 3 months:

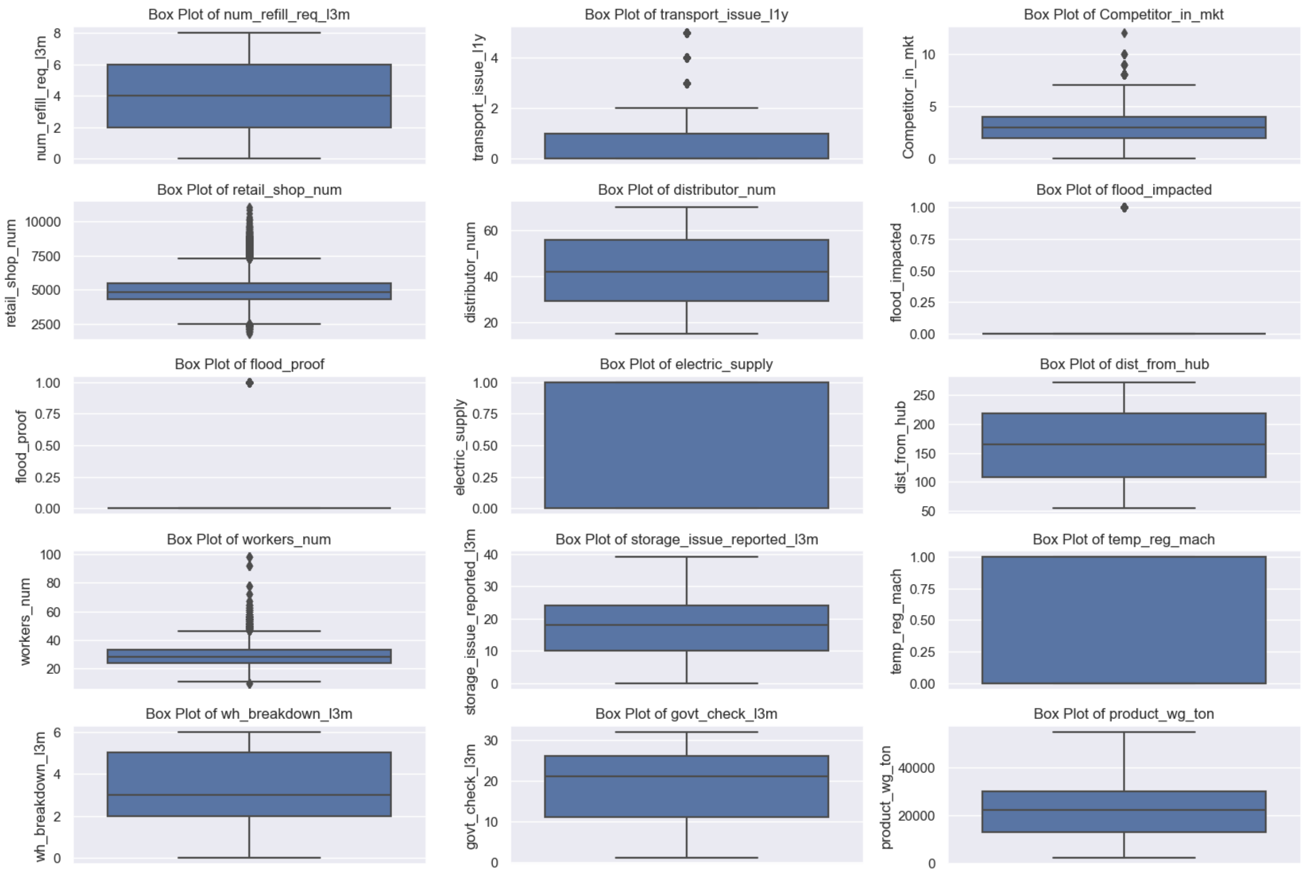
Very less stores have no breakdowns in last 3 months. But others have reported, which may directly impact refill urgency.

Number of government checks in 3 months:

Most warehouses faced few inspections; some had frequent ones, which may affect operational compliance or delays.

Product weight in tons:

Product load varies, most between 10,000–30,000 tons. High weight could indicate higher product turnover and refill need.



Box-plot of Numeric variables

Inference:

- There are some outliers present in 'transport_issue_l1y', 'Competitor_in_mkt', 'retail_shop_num', 'flood_impacted', 'flood_proof','workers_num', 'wh_breakdown_l3m'

Box-plot of Categorial variables

Location_type:

Rural : 22957
Urban: 2043

WH_capacity_size

Large :10169
Mid :10020
Small : 4811

zone

North 10278
West 7931
South 6362
East 429

WH_regional_zone

Zone 6: 8339
Zone 5: 4587
Zone 4: 4176
Zone 2: 2963
Zone 3: 2881
Zone 1: 2054

wh_owner_type

Company Owned: 13578
Rented : 11422

approved_wh_govt_certificate

C : 6409
B+: 4917
B : 4812
A : 4671
A+: 4191

Inferences:

Location_type

- Rural: 22,957
- Urban: 2,043
  Majority of warehouses are in rural areas which is 92%.
  Rural logistics dominate; hence, accessibility and connectivity play a critical role in refill performance.

WH_capacity_size

- Mid: 10,200
- Large: 10,169
- Small: 4,811
  Most warehouses are of mid or large capacity.
  Refill planning should be scalable, as large volume handling is common.

Zone (Geographical)

- North is 10,378 and West is 7,931 dominate.
- East has the least 429.
  Geographic distribution is skewed towards North and West zones.
  Refill logistics and resource deployment must be tailored to these high-density zones.

WH_regional_zone

- Most common: Zone 6 has 8,139 and Zone 5 has 4,587
- Least common: Zone 1 is having 2,383
  Regional zones are unequally distributed.

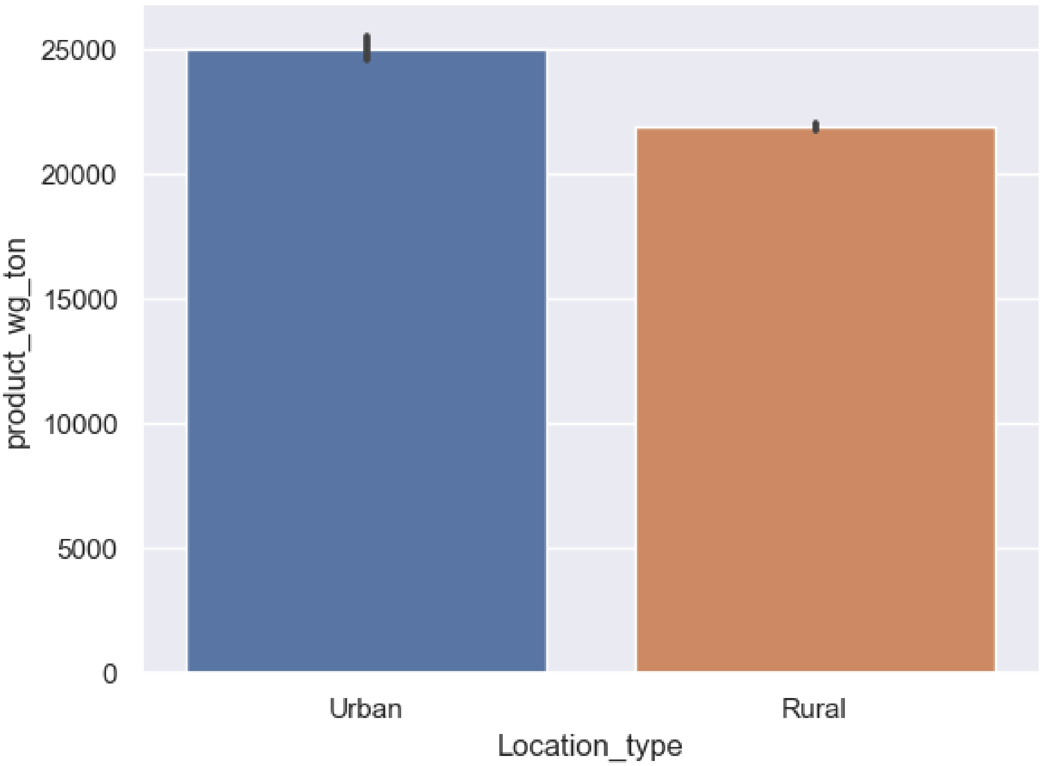
WH_owner_type

- Rented: 11,422
- Company Owned: 13,578
  Balanced mix of ownership, with a slight tilt towards company-owned warehouses.

approved_wh_govt_certificate

- Highest: C grade has 6,409, followed by B+ grade has 4,917
- Lowest: A+ grade has 4,191
  Most warehouses are certified at basic levels (C or B+).
  There may be a scope for quality improvement; higher certification could relate to better management and fewer refill issues.

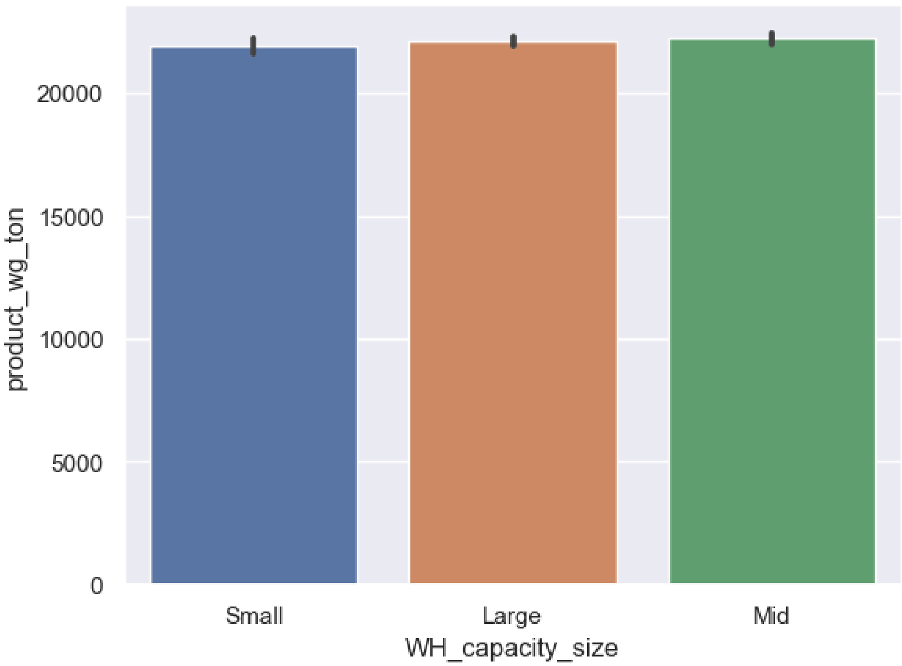
Bivariate analysis (relationship between target variable and other numericals):



Location type vs product_wg_ton

<u>Inferences:</u>

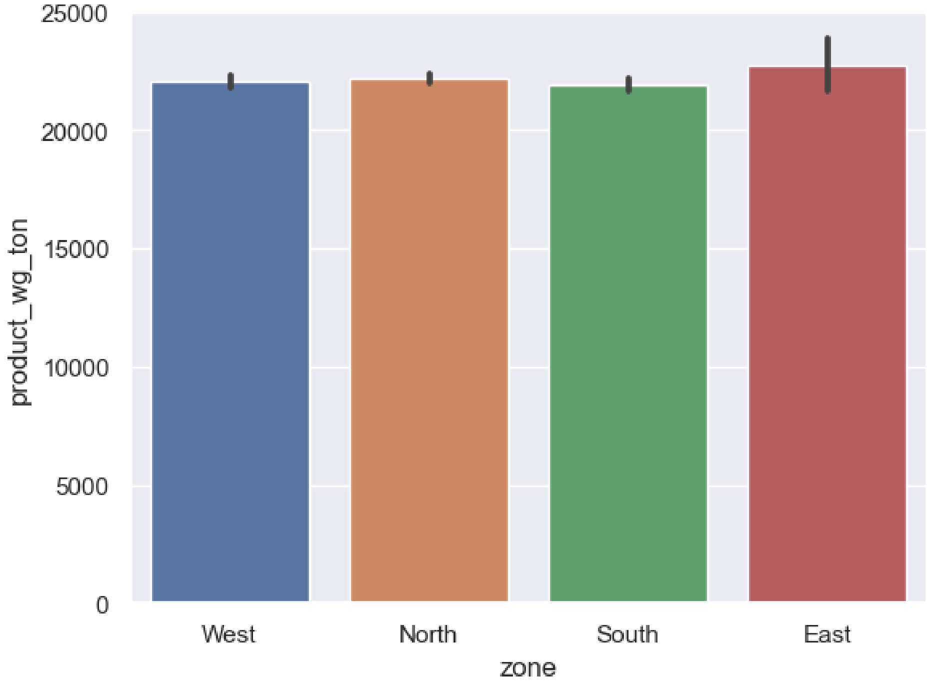
As per the storage in tons, the Urban warehouses have a total of highest storage, which is nearly 25,000 tons. And the Rural warehouses have a storage of nearly 22,000 tons.



WH_capacity_size vs product_wg_ton

Inferences:

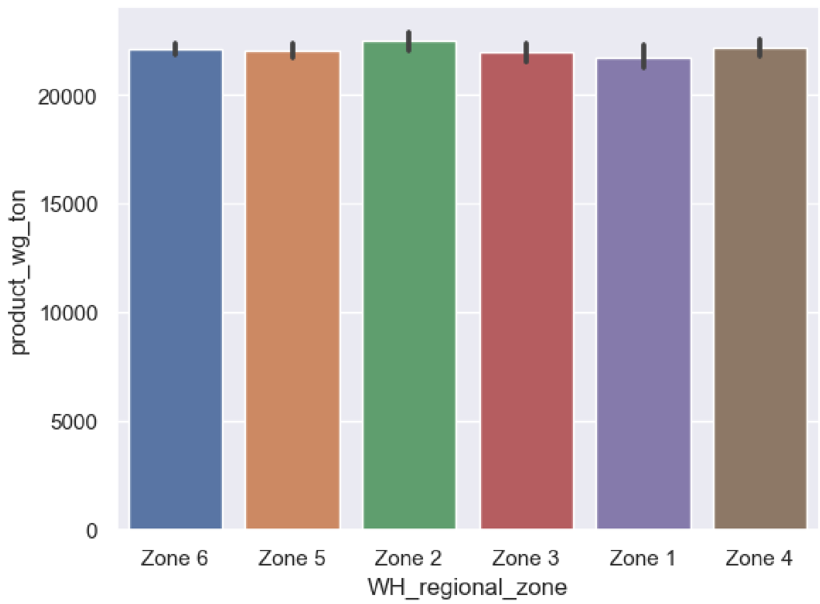
The Mid capacity warehouses are having the max storage in weight. Then the Large capacity warehouses come with the next larger weight in storage and the least weight in storages are in the small warehouses.



zone vs product_wg_ton

Inferences:

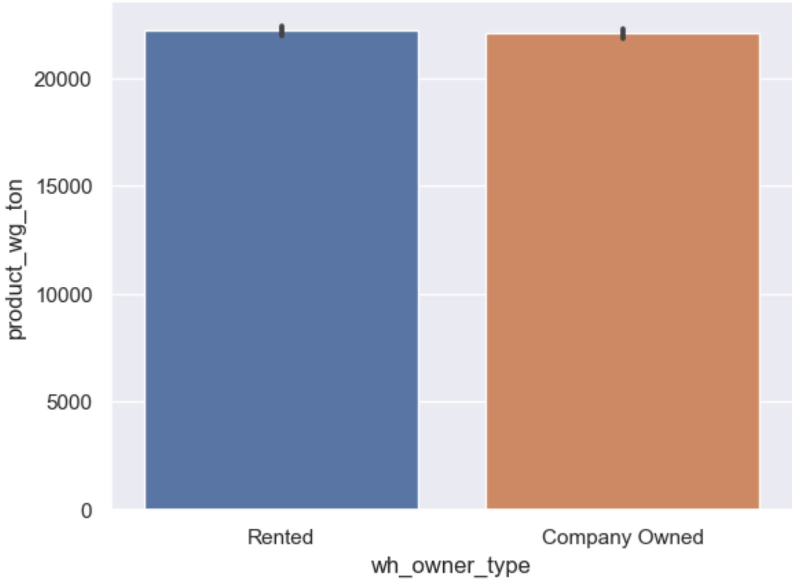
The East zone is having highest weight of storage. Then comes North zone and West zone. The least weight of storage is having the South zone.



WH_regional_zone vs product_wg_ton

Inferences:

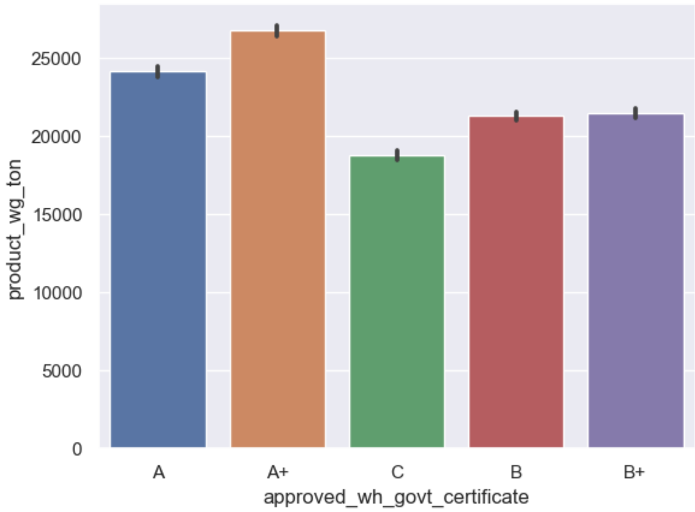
Among all the Zones, Zone-2 is having the highest weight of storage. And the Zone- 4, Zone-6, Zone-5, Zone-3 are following Zone-2. Zone-1 has the lowest weight of storage.



wh_owner_type vs product_wg_ton

Inferences:

The comparison between the weight of storage among warehouse type, the weight of storages in rented warehouses are larger than the company owned warehouses.



approved_wh_govt_certificate vs product_wg_ton

Inferences:

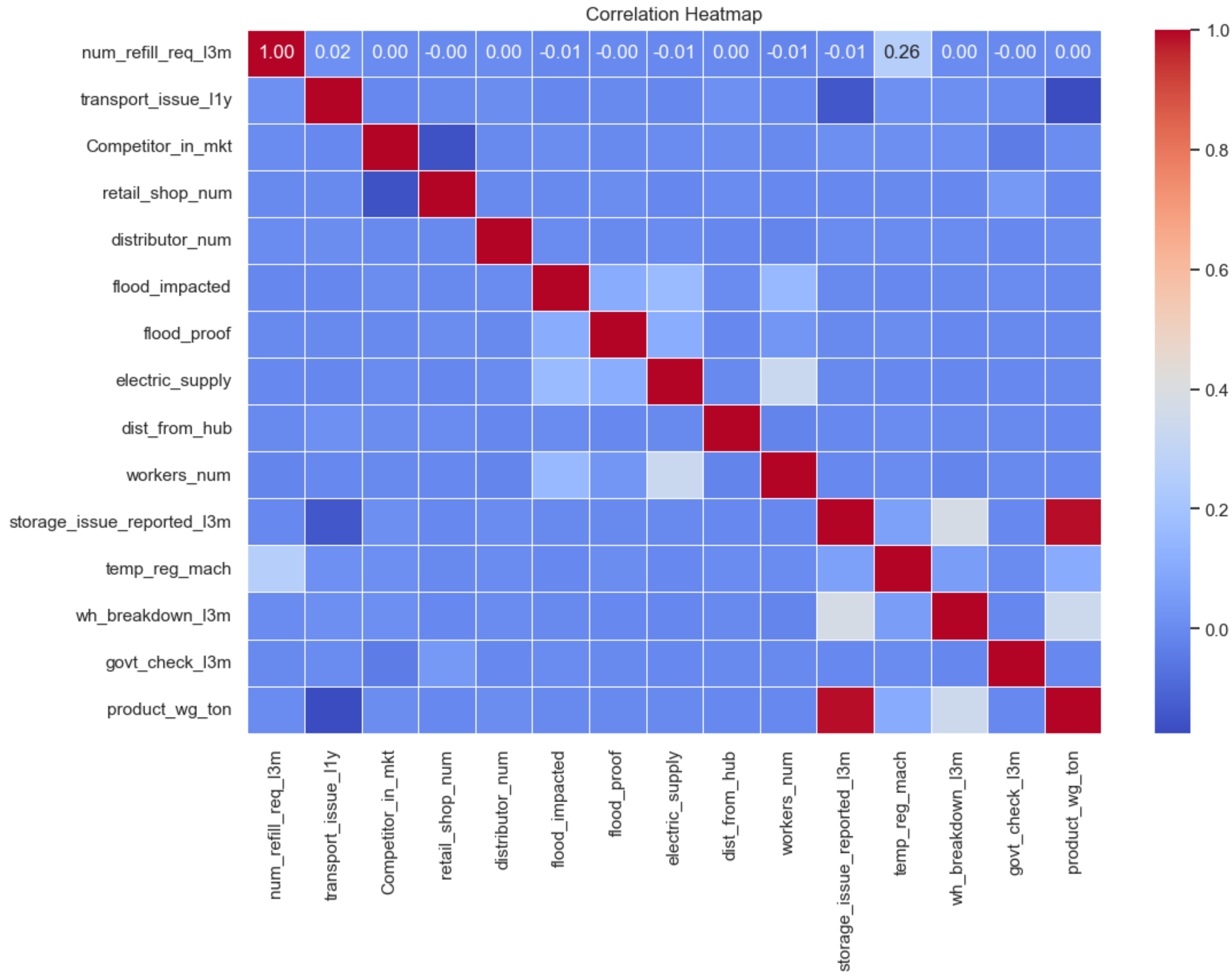
- Warehouses in urban areas did heavier amount of product storage than rural areas.
- East zone warehouses did the heaviest amount of product storage than North, South & West.
- Zone-2 & 4 did the heaviest amount of product storagecompared to other zones.
- A^+ & A govt certified warehouses did the heaviest amount of product storage.

Multi-variate analysis

Pair-plot of all numeric variables

Inference:

High correlation between retail_shop_num and distributor_num indicates market spread.



Correlation heat-map of numeric variables

Inferences:

- "storage_issue_reported_l3m" is highly positively correlated with target variable
- "wh_breakdown_l3m" has some correlation with target variable

3. Data Cleaning and Pre-processing –

Approach used for identifying and treating missing values and outlier treatment (and why) - Need for variable transformation (if any) - Variables removed or added and why (if any)

Missing Value:

```
wh_est_year                     11881
workers_num                       990
approved_wh_govt_certificate      908
dtype: int64
```

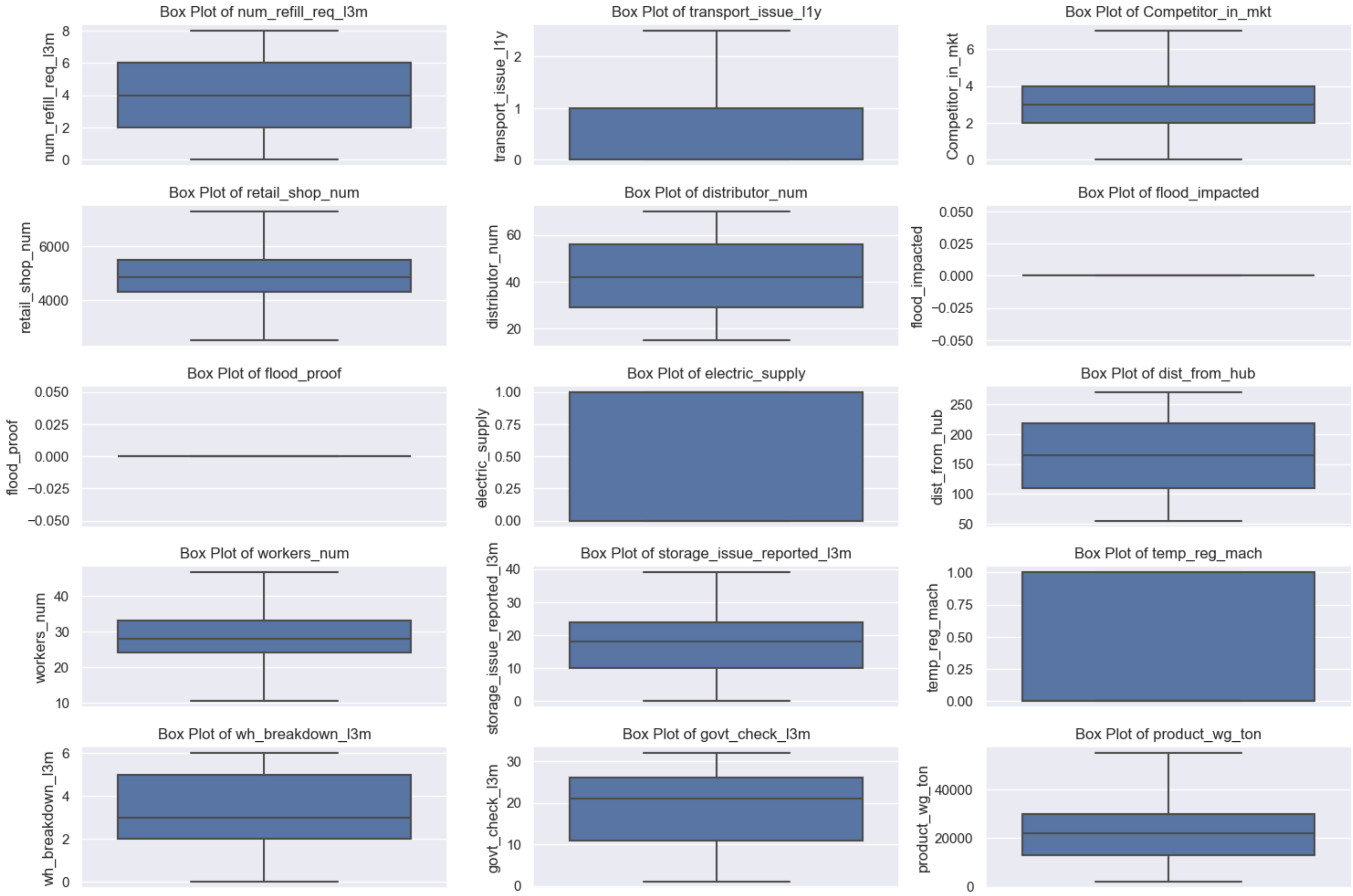
- The "wh_est_year" feature has high percentage of missing values and hence this feature would be dropped from analysis.
- Other two variables are imputed. The workers_num is imputed through median & approved_wh_govt_certificate is imputed through mode.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Location_type                 25000 non-null  object 
 1   WH_capacity_size              25000 non-null  object 
 2   zone                          25000 non-null  object 
 3   WH_regional_zone              25000 non-null  object 
 4   num_refill_req_l3m            25000 non-null  int64  
 5   transport_issue_l1y           25000 non-null  int64  
 6   Competitor_in_mkt             25000 non-null  int64  
 7   retail_shop_num               25000 non-null  int64  
 8   wh_owner_type                 25000 non-null  object 
 9   distributor_num               25000 non-null  int64  
 10  flood_impacted                25000 non-null  int64  
 11  flood_proof                   25000 non-null  int64  
 12  electric_supply               25000 non-null  int64  
 13  dist_from_hub                 25000 non-null  int64  
 14  workers_num                   25000 non-null  float64
 15  storage_issue_reported_l3m    25000 non-null  int64  
 16  temp_reg_mach                 25000 non-null  int64  
 17  approved_wh_govt_certificate  25000 non-null  object 
 18  wh_breakdown_l3m              25000 non-null  int64  
 19  govt_check_l3m                25000 non-null  int64  
 20  product_wg_ton                25000 non-null  int64  
dtypes: float64(1), int64(14), object(6)
memory usage: 4.0+ MB
```

After treating Missing Values

Outlier treatment:

All the outliers are treated by IQR method.



After outliers being treated

In the course of the EDA, we noted that the box plots of the numeric variables indicated a few values which could be classed as outliers. We identified and treated the outliers in the following manner:

“transport_issue_l1y”, “Competitor_in_mkt”, “retail_shop_num”, “flood_impacted”, “flood_proof” and “workers_num” have a few extreme values, which are true outliers. These variables have been capped at the Q3 + (1.5 * IQR) level.

However, variable transformation & addition of new variables are not required for this analysis right now. But, Log transformation applied on skewed variables like dist_from_hub. And standardization is used for numerical features.

Removal of unwanted variables:

- Ware_house_ID: ID of warehouse, it does not contribute to our solution
- WH_Manager_ID: ID of manager, it does not contribute to our solution
- flood_impacted: This feature has only one unique value, therefore it does not contribute to our solution
- flood_proof: This feature has only one unique value, therefore it does not contribute to our solution
- The "wh_est_year" feature has high percentage of missing values and hence this feature would be dropped from analysis.

Variable Transformation:

However, variable transformation & addition of new variables are not required for this analysis right now. But, Log transformation applied on skewed variables like dist_from_hub. And standardization is used for numerical features.

In the section, we will transform categorical variables into numeric values to make it ready for modeling:

- We converted the data type of the ordinal variable (Label Encoding): WH_capacity_size and approved_wh_govt_certificate
- We performed One Hot Encoding for the following categorical variables, with Drop First as True: Location_type, zone, WH_regional_zone and wh_owner_type.

Before we commence with modeling, we will further split the data into training and test datasets, and scale the data as a part of pre-processing. We will build and train our models using the training set, and validate it using the test set.

Data Split:

The dataset is split (80:20) into training and test data sets using scikit-learn's train_test_split function. The dataset dimensions after split are as follows:

Dimension of X_train: (20000, 24)
Dimension of X_test: (5000, 24)

Scaling:

- Scaling the numeric data: We used scikit-learn's **StandardScaler** library to standardise the numeric variables in the dataset.
- Note that we fit_transform the train data & only transformed the test data. Effectively we have used the means and standard deviations of the training data to transform the test dataset variables.

4. Model building –

Clear on why was a particular model(s) chosen. - Effort to improve model performance.

Preliminary Model Building:

A range of models (spanning parametric, non-parametric, kernel-based etc.) were built on the training data, to assess which model or model types are most suited for the data. The performance of these models was validated on the test data. And the performances were then tabulated and compared.

The principle metrics used to assess model performance are Mean Absolute Error (MAE), Mean Squared Error (MSE) and R-squared score in that particular order of importance. The choice of metrics and methods used for Validation have been discussed in the subsequent section.

Details of the models built, their performance metrics and insights have been provided in detail in the Appendix of the report. Here we summarise the performance and overall insights of Part 1 of the model building exercise:

MODEL	MAE ↓ (LOWER IS BETTER)	MSE ↓ (LOWER IS BETTER)	R^2 ↑ (HIGHER IS BETTER)
Linear Regression	1303.98	3,093,785.29	0.9769
Ridge Regression	1304.01	3,093,803.48	0.9769
Lasso Regression	1303.66	3,092,922.22	0.9769
Support Vector Regressor (SVR)	9201.02	124,884,627.79	0.0666
Random Forest Regressor	700.45	886,565.92	0.9934
Gradient Boosting Regressor	**689.44**	**836,425.33**	**0.9937**

Infference:

The Gradient Boosting Regressor is the best-performing model for this dataset, as it has the lowest error (MAE & MSE) and the highest R^2.For the best predictive performance, Gradient Boosting should be used.

Model Tuning, Boosting:

We also explored Ensemble modeling (Boosting or Tunning) to improve overall model performance. Details of the models built, their hyper-parameters, performance metrics and insights have been provided in detail in the Appendix of the report. Here we summarise the performance and overall insights of Part 2 of the model building exercise:

Model Performance Comparison:

MODEL	MAE ↓ (LOWER IS BETTER)	MSE ↓ (LOWER IS BETTER)	R^2 ↑ (HIGHER IS BETTER)
Linear Regression	1303.98	3,093,785.29	0.9769
Ridge Regression	1304.01	3,093,803.48	0.9769
Lasso Regression	1303.66	3,092,922.22	0.9769
Support Vector Regressor (SVR)	9201.02	124,884,627.79	0.0666
Random Forest Regressor	700.45	886,565.92	0.9934
Gradient Boosting Regressor	689.44	836,425.33	**0.9937**
Tuned Gradient Boosting (Best Parameters: learning_rate=0.1, max_depth=7, n_estimators=50, subsample=1.0)	**679.43**	**820,252.17**	**0.9939**
Tuned RandomForestRegressor(max_depth=30, min_samples_leaf=5)	691.32	851,405.36	0.9936

Based on the MAE, MSE and R-square scores, the top 2 models are:

- The tuned Gradient Boosting model outperforms all others, showing the lowest MAE (679.22), lowest MSE (819,545.85), and highest R^2 (0.9939).
- The tuned Random Forest (691.75 MAE, 855,671.29 MSE, 0.9936 R^2) remains close but is slightly inferior.

Modeling Summary:

Of all the models built and tuned, based on the MAE, MSE and R-square scores, the following 2 models have proved to be the best predictors of product_wg_ton:

- Gradient Boosting (tuned)
- RandomForest Regressor (tuned)

5. Model validation –

How was the model validated ? Just accuracy, or anything else too ?

Model	MAE ↓ (Lower is better)	MSE ↓ (Lower is better)	R^2 ↑ (Higher is better)
Random Forest Regressor	700.45	886,565.92	0.9934
Gradient Boosting Regressor	689.44	836,425.33	0.9937

This is the metric values captured for different metrics before tunning the models.
The Gradient Boosting Regressor and the RandomForest are the best-performing models for this dataset, as it has the lowest error (MAE & MSE) and the highest R^2. Hence for the best predictive models, Gradient Boosting and RandomForest are chosen to be hyper-tunned.

Model	**MAE ↓ (Lower is better)**	**MSE ↓ (Lower is better)**	**R^2 ↑ (Higher is better)**
Random Forest Regressor	700.45	886,565.92	0.9934
Gradient Boosting Regressor	689.44	836,425.33	0.9937
Tuned Gradient Boosting (Best Parameters: learning_rate=0.1, max_depth=7, n_estimators=50, subsample=1.0)	679.43	820,252.17	0.9939
Tuned RandomForestRegressor(max_depth=30, min_samples_leaf=5)	691.32	851,405.36	0.9936

- The tuned Gradient Boosting model outperforms all others, showing the lowest MAE (679.43), lowest MSE (820,252.17), and highest R^2 (0.9939).
- The tuned Random Forest (691.32 MAE, 851,405.36 MSE, 0.9936 R^2) remains close but is slightly inferior.
- Compared to the default Gradient Boosting model, the tuning improved all metrics, confirming that the optimization process worked well.

We hence conclude that the following 2 models have proved to be the best predictors of customer churn in this case:

1. Gradient Boosting (tuned)
2. Random Forest (tuned)

6. Final Interpretation and Recommendations

Optimized Gradient Boost Model:

- Provides accurate shipment predictions per warehouse.
- Helps optimize inventory levels, reducing overstocking costs.
- Aids logistics planning by predicting demand hotspots.

Business Recommendations

- Implement Gradient Boosting Model for Inventory Predictions
- Develop Region-Specific Distribution Strategies
- Focus on Rural Warehouse Optimization
- Enhance Logistics Planning
- Use data-driven insights to refine marketing strategies

Potential Impact

- Reduce Inventory Holding Costs
- Improve Supply Chain Efficiency
- Enable Data-Driven Decision Making
- Support Targeted Marketing Campaigns

Future Enhancements

- Real-Time Inventory Tracking
- Machine Learning Model Continuous Learning
- Integration with IoT and Predictive Analytics
- Develop Comprehensive Supply Chain Dashboard

Appendix:

All the raw codes and outputs are bellow for the reference:

```
import warnings
warnings.filterwarnings(action='ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt

sns.set()
sns.set_palette(palette='deep')
import folium
from folium.plugins import FastMarkerCluster

# importing dataset
df = pd.read_csv('Data.csv')
```

Understanding Data

```
df.head()
```

```
  Ware_house_ID WH_Manager_ID Location_type WH_capacity_size   zone  \
0     WH_100000     EID_50000         Urban            Small   West
1     WH_100001     EID_50001         Rural            Large  North
2     WH_100002     EID_50002         Rural              Mid  South
3     WH_100003     EID_50003         Rural              Mid  North
4     WH_100004     EID_50004         Rural            Large  North

  WH_regional_zone  num_refill_req_l3m  transport_issue_l1y  \
0           Zone 6                   3                    1
1           Zone 5                   0                    0
2           Zone 2                   1                    0
3           Zone 3                   7                    4
4           Zone 5                   3                    1

   Competitor_in_mkt  retail_shop_num  wh_owner_type  distributor_num  \
0                  2             4651         Rented               24
1                  4             6217  Company Owned               47
2                  4             4306  Company Owned               64
3                  2             6000         Rented               50
4                  2             4740  Company Owned               42

   flood_impacted  flood_proof  electric_supply  dist_from_hub  workers_num  \
0               0            1                1             91         29.0
1               0            0                1            210         31.0
2               0            0                0            161         37.0
3               0            0                0            103         21.0
4               1            0                1            112         25.0

   wh_est_year  storage_issue_reported_l3m  temp_reg_mach  \
0          NaN                          13              0
1          NaN                           4              0
2          NaN                          17              0
3          NaN                          17              1
4       2009.0                          18              0
```

```
  approved_wh_govt_certificate  wh_breakdown_l3m  govt_check_l3m  \
0                            A                 5              15   
1                            A                 3              17   
2                            A                 6              22   
3                           A+                 3              27   
4                            C                 6              24   

   product_wg_ton  
0           17115  
1            5074  
2           23137  
3           22115  
4           24071  
```

```
pd.options.display.max_columns = None
```

```
df.head()
```

```
  Ware_house_ID WH_Manager_ID Location_type WH_capacity_size   zone  \
0     WH_100000     EID_50000         Urban            Small   West   
1     WH_100001     EID_50001         Rural            Large  North   
2     WH_100002     EID_50002         Rural              Mid  South   
3     WH_100003     EID_50003         Rural              Mid  North   
4     WH_100004     EID_50004         Rural            Large  North   

  WH_regional_zone  num_refill_req_l3m  transport_issue_l1y  \
0           Zone 6                   3                    1   
1           Zone 5                   0                    0   
2           Zone 2                   1                    0   
3           Zone 3                   7                    4   
4           Zone 5                   3                    1   

   Competitor_in_mkt  retail_shop_num  wh_owner_type  distributor_num  \
0                  2             4651         Rented               24   
1                  4             6217  Company Owned               47   
2                  4             4306  Company Owned               64   
3                  2             6000         Rented               50   
4                  2             4740  Company Owned               42   

   flood_impacted  flood_proof  electric_supply  dist_from_hub  workers_num  \
0               0            1                1             91         29.0   
1               0            0                1            210         31.0   
2               0            0                0            161         37.0   
3               0            0                0            103         21.0   
4               1            0                1            112         25.0   

   wh_est_year  storage_issue_reported_l3m  temp_reg_mach  \
0          NaN                          13              0   
1          NaN                           4              0   
2          NaN                          17              0   
3          NaN                          17              1   
4       2009.0                          18              0   

  approved_wh_govt_certificate  wh_breakdown_l3m  govt_check_l3m  \
0                            A                 5              15   
1                            A                 3              17   
2                            A                 6              22   
3                           A+                 3              27   
4                            C                 6              24   
```

```
   product_wg_ton
0           17115
1            5074
2           23137
3           22115
4           24071
```

```
df.shape
```

```
(25000, 24)
```

```
df.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
```

Data Cleaning & Preprocessing

```
### Examine missing values
df_na = df.isna().sum()
df_na[df_na.values > 0].sort_values(ascending=False) # Find out all variables that contain missing values
```

```
wh_est_year                     11881
workers_num                       990
approved_wh_govt_certificate      908
dtype: int64
```

The "wh_est_year" feature has high percentage of missing values and hence this feature would be dropped from analysis. We are dropping this from our dataset to make sure that other valid observations do not get eliminated when we remove or impute the 'na' values.

```
df.drop(['wh_est_year'],axis='columns', inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 23 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  storage_issue_reported_l3m    25000 non-null  int64
 18  temp_reg_mach                 25000 non-null  int64
 19  approved_wh_govt_certificate  24092 non-null  object
 20  wh_breakdown_l3m              25000 non-null  int64
 21  govt_check_l3m                25000 non-null  int64
 22  product_wg_ton                25000 non-null  int64
dtypes: float64(1), int64(14), object(8)
memory usage: 4.4+ MB
```

```
df.describe()
```

```
       num_refill_req_l3m  transport_issue_l1y  Competitor_in_mkt  \
count        25000.000000         25000.000000       25000.000000
mean             4.089040             0.773680           3.104200
std              2.606612             1.199449           1.141663
min              0.000000             0.000000           0.000000
25%              2.000000             0.000000           2.000000
50%              4.000000             0.000000           3.000000
75%              6.000000             1.000000           4.000000
max              8.000000             5.000000          12.000000

       retail_shop_num  distributor_num  flood_impacted   flood_proof  \
count     25000.000000     25000.000000    25000.000000  25000.000000
mean       4985.711560        42.418120        0.098160      0.054640
std        1052.825252        16.064329        0.297537      0.227281
min        1821.000000        15.000000        0.000000      0.000000
25%        4313.000000        29.000000        0.000000      0.000000
50%        4859.000000        42.000000        0.000000      0.000000
75%        5500.000000        56.000000        0.000000      0.000000
max       11008.000000        70.000000        1.000000      1.000000

       electric_supply  dist_from_hub   workers_num  \
count     25000.000000   25000.000000  24010.000000
mean          0.656880     163.537320     28.944398
std           0.474761      62.718609      7.872534
```

```
min           0.000000     55.000000     10.000000
25%           0.000000    109.000000     24.000000
50%           1.000000    164.000000     28.000000
75%           1.000000    218.000000     33.000000
max           1.000000    271.000000     98.000000

       storage_issue_reported_l3m  temp_reg_mach  wh_breakdown_l3m  \
count                25000.000000   25000.000000      25000.000000
mean                    17.130440       0.303280          3.482040
std                      9.161108       0.459684          1.690335
min                      0.000000       0.000000          0.000000
25%                     10.000000       0.000000          2.000000
50%                     18.000000       0.000000          3.000000
75%                     24.000000       1.000000          5.000000
max                     39.000000       1.000000          6.000000

       govt_check_l3m  product_wg_ton
count    25000.000000    25000.000000
mean        18.812280    22102.632920
std          8.632382    11607.755077
min          1.000000     2065.000000
25%         11.000000    13059.000000
50%         21.000000    22101.000000
75%         26.000000    30103.000000
max         32.000000    55151.000000
```

```
### Let's examine the target column which is product_wg_ton

df.describe(include="all")["product_wg_ton"]
```

```
count     25000.000000
unique             NaN
top                NaN
freq               NaN
mean      22102.632920
std       11607.755077
min        2065.000000
25%       13059.000000
50%       22101.000000
75%       30103.000000
max       55151.000000
Name: product_wg_ton, dtype: float64
```

```
# Filling missing numerical values with median
df['workers_num'].fillna(df['workers_num'].median(), inplace=True)

# Filling missing categorical values with mode
df['approved_wh_govt_certificate'].fillna(df['approved_wh_govt_certificate'].mode()[0],
inplace=True)

# Verifying that there are no null values in any column in the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 23 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                   25000 non-null  float64
 17  storage_issue_reported_l3m    25000 non-null  int64
 18  temp_reg_mach                 25000 non-null  int64
 19  approved_wh_govt_certificate  25000 non-null  object
 20  wh_breakdown_l3m              25000 non-null  int64
 21  govt_check_l3m                25000 non-null  int64
 22  product_wg_ton                25000 non-null  int64
dtypes: float64(1), int64(14), object(8)
memory usage: 4.4+ MB
```

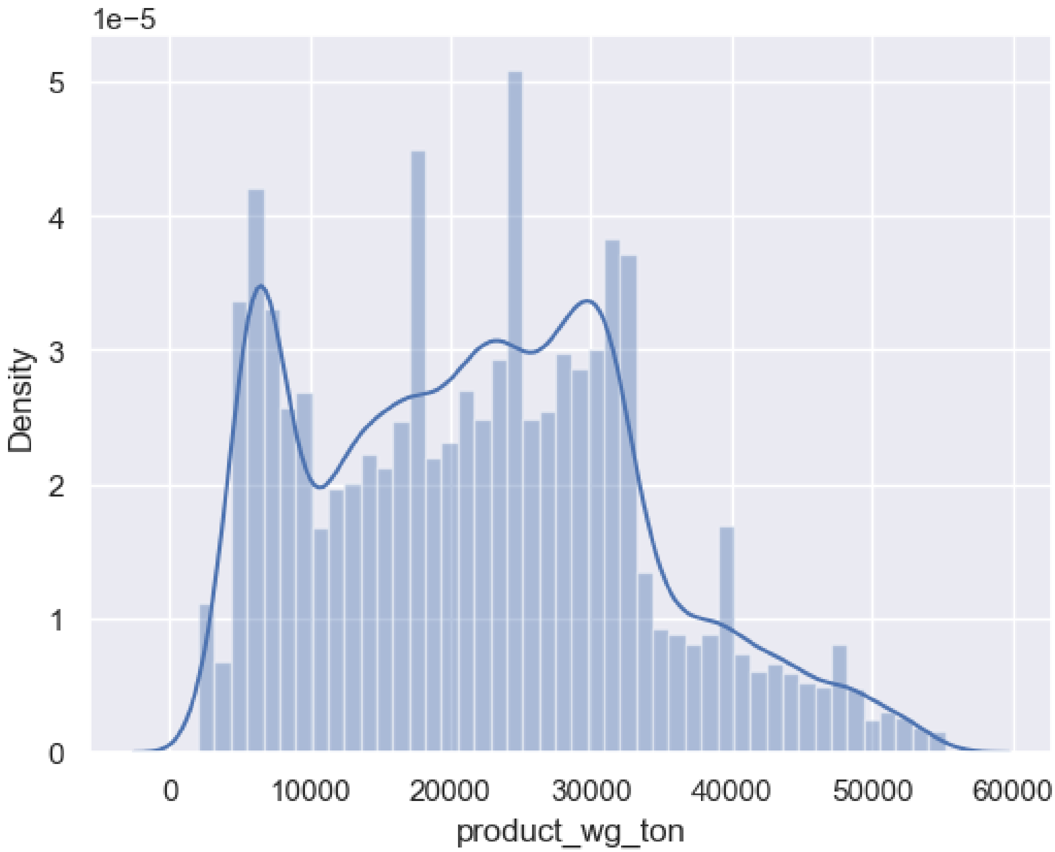
Removing Unwanted Variables

```
# Dropping irrelevant columns (IDs are not useful for prediction)
df.drop(columns=['Ware_house_ID', 'WH_Manager_ID'], inplace=True)
```

Analyzing Target variable

```
sns.distplot(df['product_wg_ton'])
```

```
<Axes: xlabel='product_wg_ton', ylabel='Density'>
```




```
df['product_wg_ton'].plot(kind='box')
```

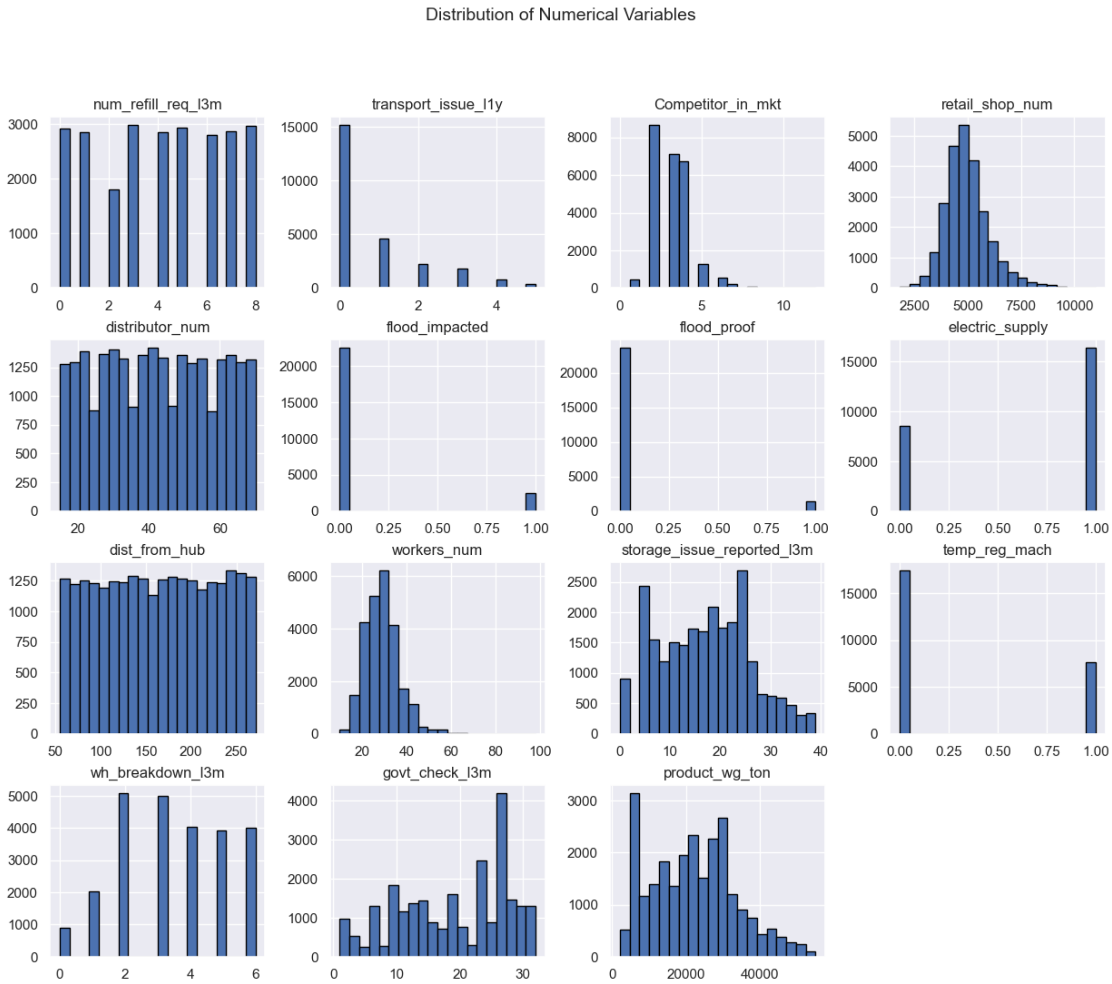
```
<Axes: >
```

Target varaible is nicely distributed and does not contain any outliers

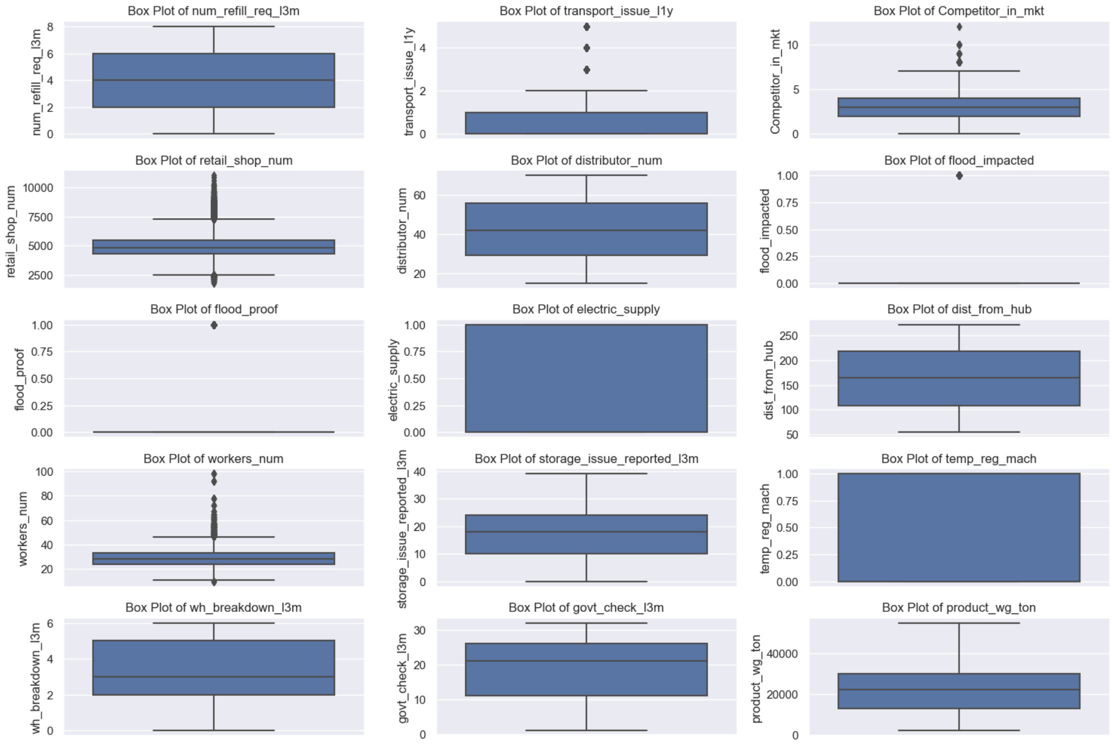
Exploratory Data Analysis (EDA)

Univariate Analysis

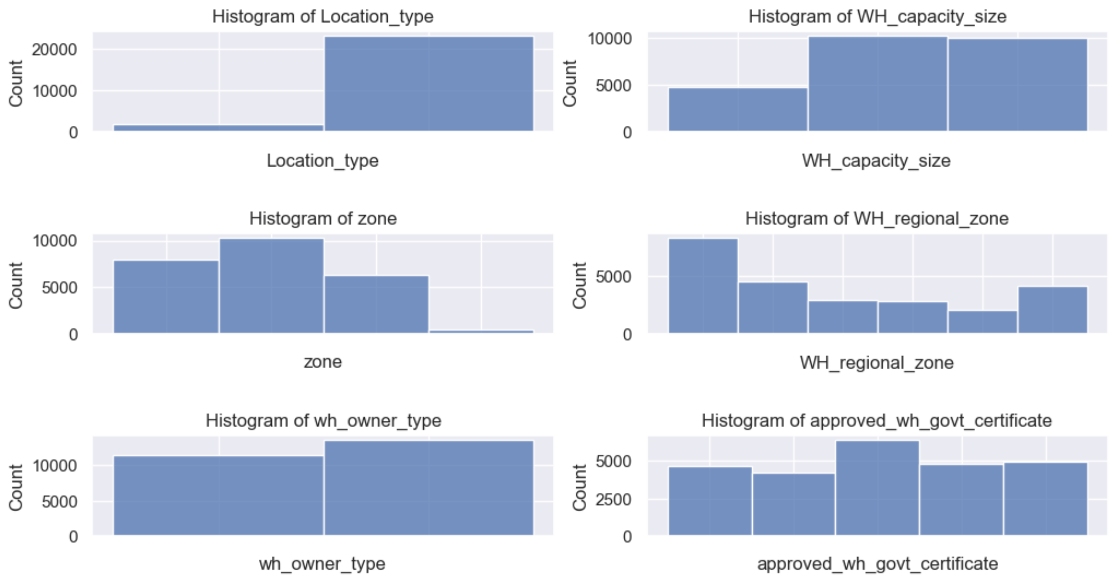
```
# Plot distribution of numerical variables
df.hist(figsize=(15, 12), bins=20, edgecolor="black")
plt.suptitle("Distribution of Numerical Variables", fontsize=14)
plt.show()
```




```python
numeric_cols = df.select_dtypes(include=["number"]).columns

plt.figure(figsize=(15, 12))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(len(numeric_cols) // 3 + 1, 3, i)  # Adjust layout for better visualization
    sns.boxplot(y=df[col])
    plt.title(f"Box Plot of {col}")

plt.tight_layout()
plt.show()
```




```
columns_to_plot_categorical = [
    'Location_type',
    'WH_capacity_size',
    'zone',
    'WH_regional_zone',
    'wh_owner_type',
    'approved_wh_govt_certificate']

# Create a single large subplot to display all histograms
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(12, 6))
fig.subplots_adjust(hspace=1)

# For loop to draw Seaborn histogram graphs
for i, column in enumerate(columns_to_plot_categorical):
    row, col = divmod(i, 2)

    # Draw Seaborn boxplot graph
    sns.histplot(x=df[column], ax=axes[row, col])
    axes[row, col].set_title(f'Histogram of {column}')
    axes[row, col].set_xticklabels([])  # Remove labels on x-axis
    axes[row, col].tick_params(axis='x', rotation=45)

plt.show()
```



Looking more into categorical variables

```
def check_value_count_for_categorical_data(column):
    print("value_count for '" ,column, "':\n", df[column].value_counts(), "\n\n---------------
----------------------\n\n" )

for col in columns_to_plot_categorical:
    check_value_count_for_categorical_data(col)
```

```
value_count for ' Location_type ':
 Location_type
Rural     22957
Urban      2043
Name: count, dtype: int64

-------------------------------------



value_count for ' WH_capacity_size ':
 WH_capacity_size
Large     10169
Mid       10020
Small      4811
Name: count, dtype: int64

-------------------------------------



value_count for ' zone ':
 zone
North     10278
West       7931
South      6362
East        429
Name: count, dtype: int64

-------------------------------------



value_count for ' WH_regional_zone ':
 WH_regional_zone
Zone 6     8339
```

```
Zone 5    4587
Zone 4    4176
Zone 2    2963
Zone 3    2881
Zone 1    2054
Name: count, dtype: int64

-------------------------------------



value_count for ' wh_owner_type ':
 wh_owner_type
Company Owned    13578
Rented           11422
Name: count, dtype: int64

-------------------------------------



value_count for ' approved_wh_govt_certificate ':
 approved_wh_govt_certificate
C     6409
B+    4917
B     4812
A     4671
A+    4191
Name: count, dtype: int64

-------------------------------------
```

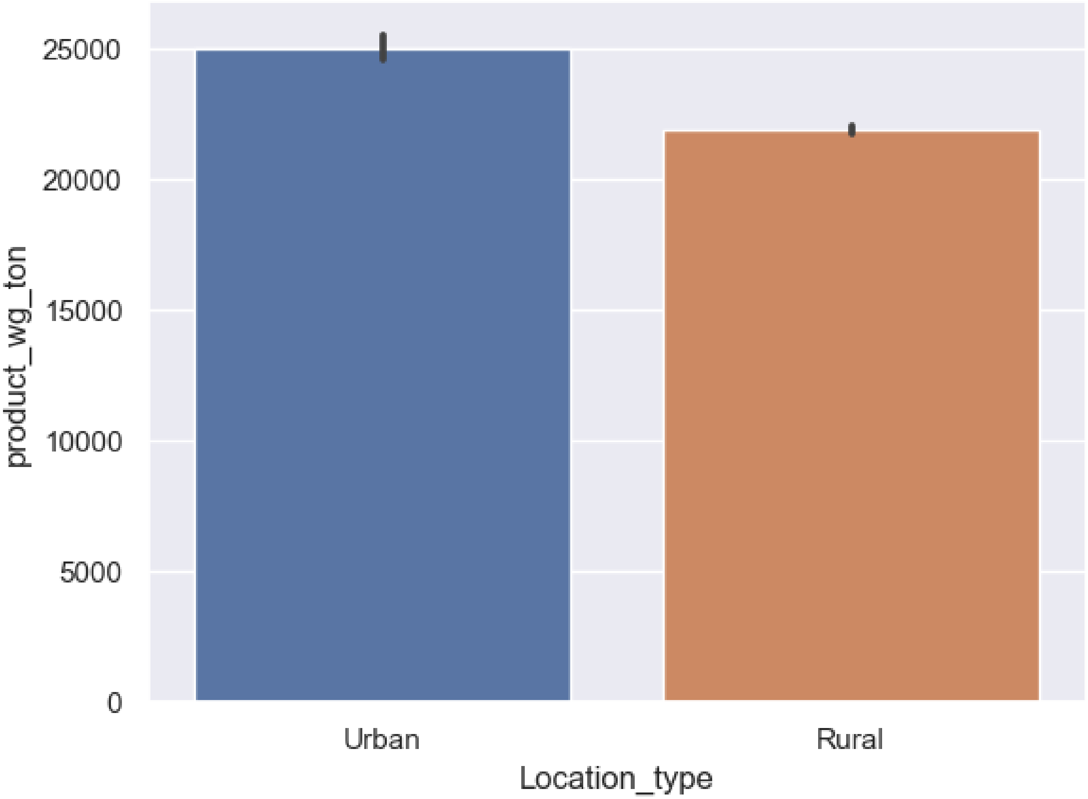
Observation from Univariate Analysiss

Many columns have ouliers present and distribution is also skewed for many features

Bi-variate Analysis

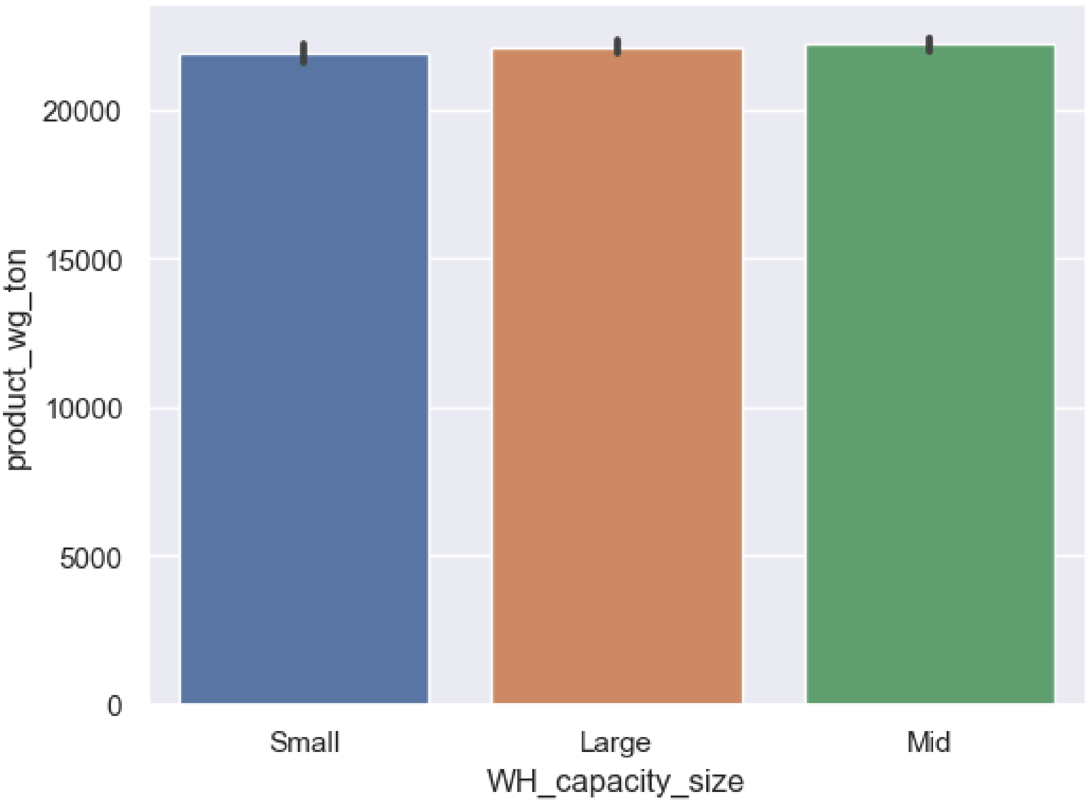
```
sns.barplot(x='Location_type',y='product_wg_ton',data=df)
```

```
<Axes: xlabel='Location_type', ylabel='product_wg_ton'>
```



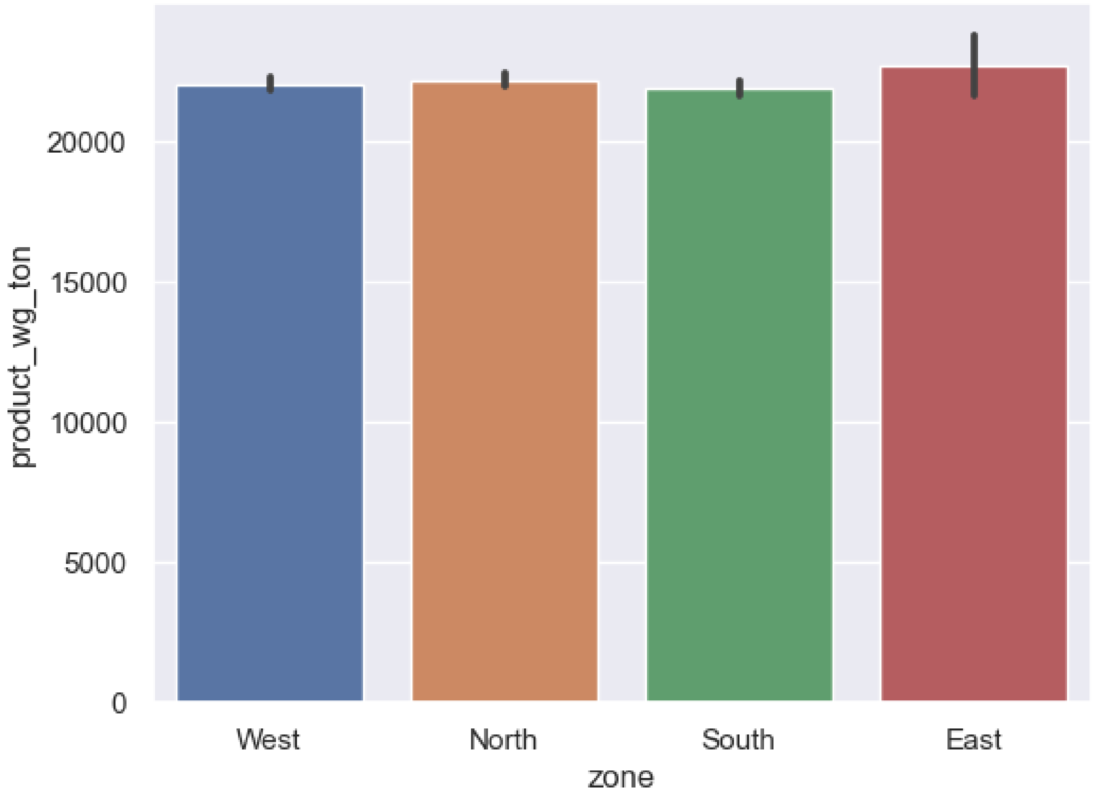

```
sns.barplot(x='WH_capacity_size',y='product_wg_ton',data=df)
```

```
<Axes: xlabel='WH_capacity_size', ylabel='product_wg_ton'>
```



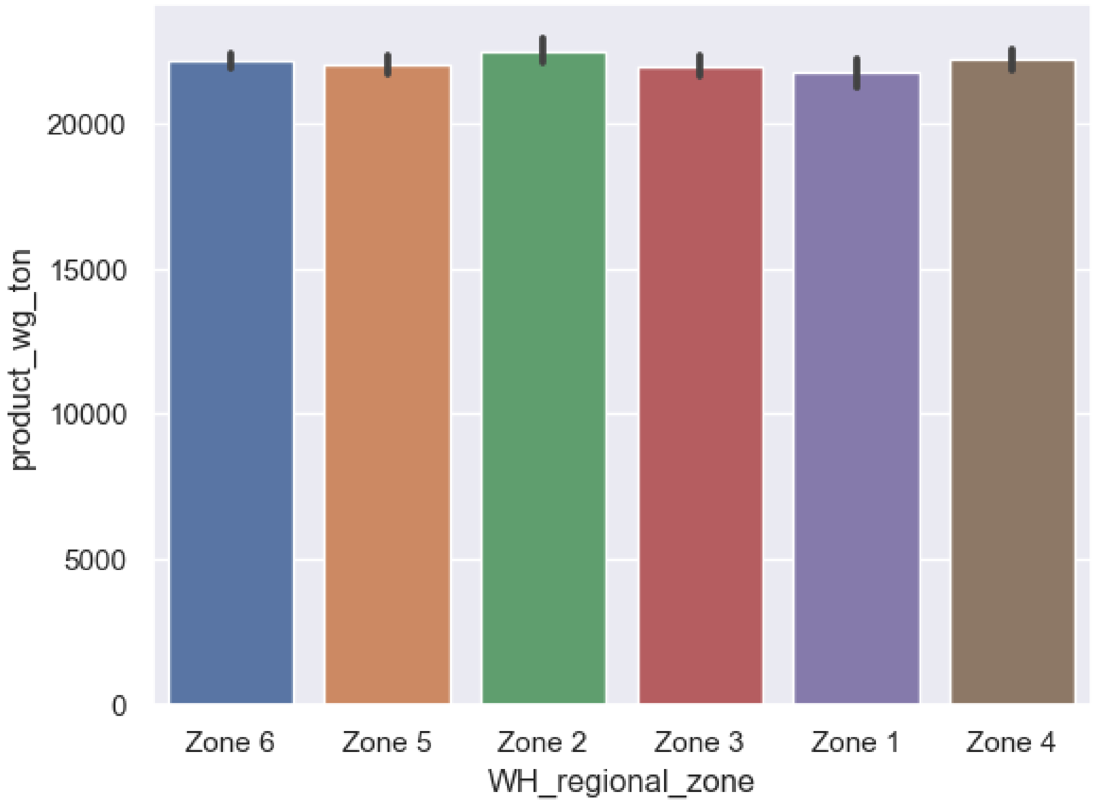

```
sns.barplot(x='zone',y='product_wg_ton',data=df)
```

```
<Axes: xlabel='zone', ylabel='product_wg_ton'>
```



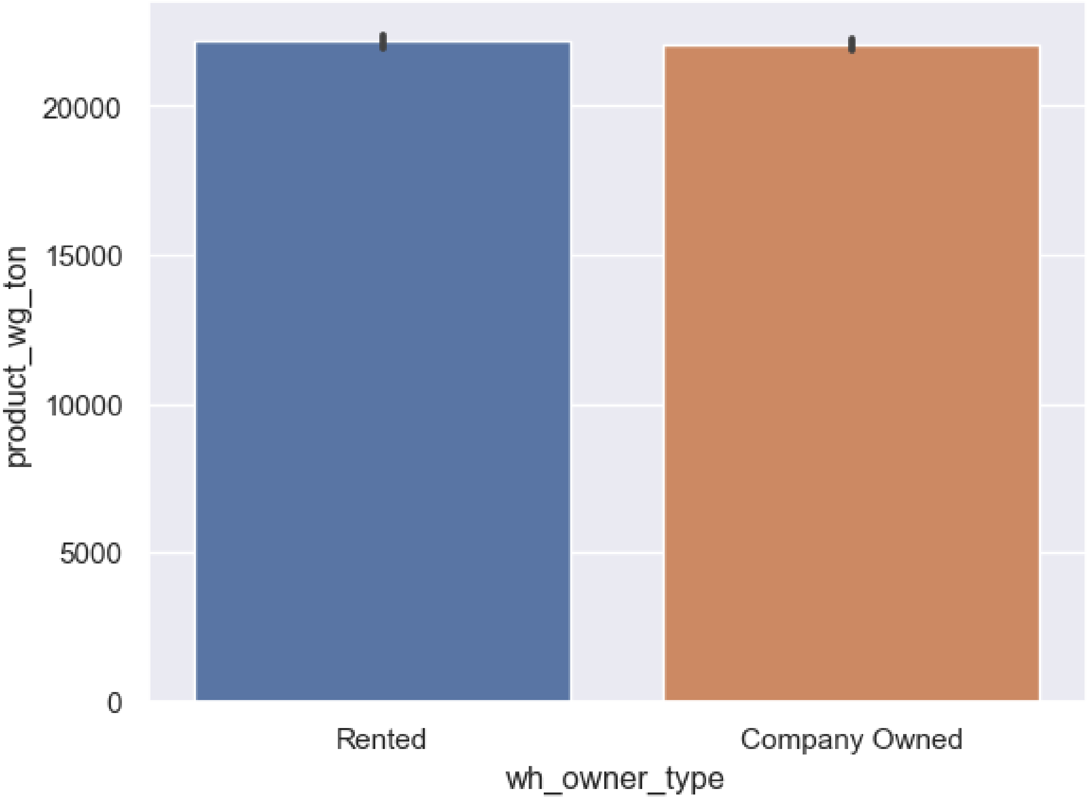

```
sns.barplot(x='WH_regional_zone',y='product_wg_ton',data=df)
```

```
<Axes: xlabel='WH_regional_zone', ylabel='product_wg_ton'>
```



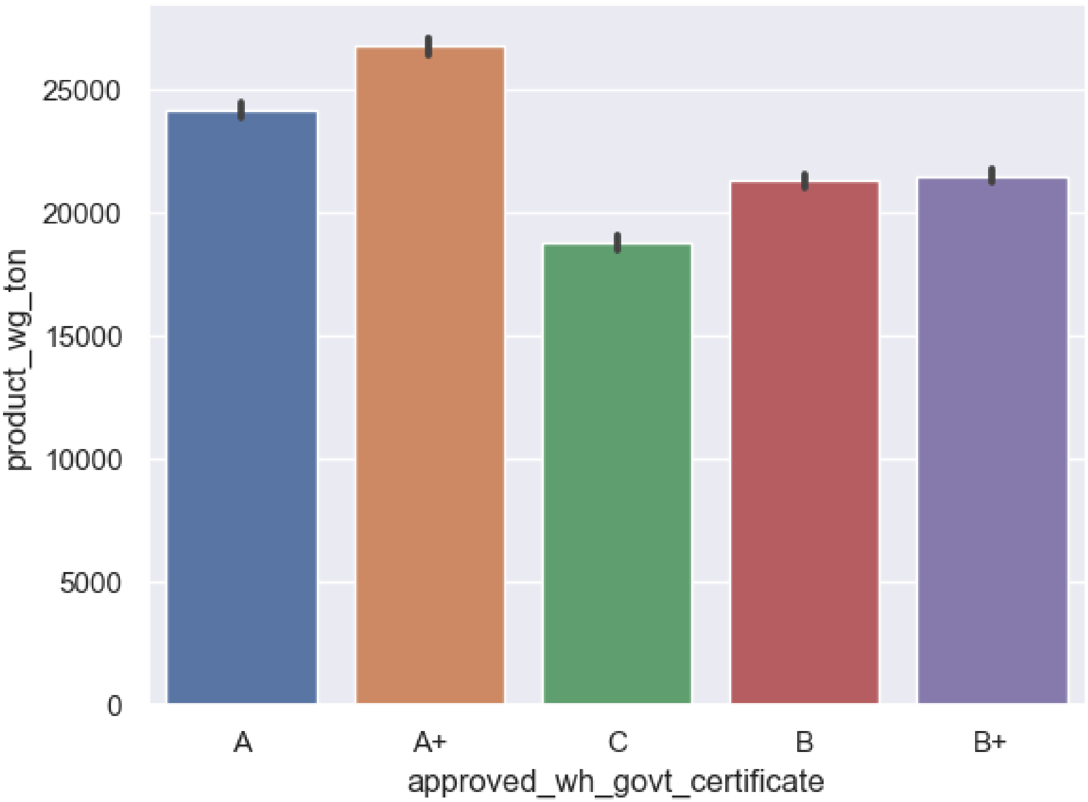

```
sns.barplot(x='wh_owner_type',y='product_wg_ton',data=df)
```

```
<Axes: xlabel='wh_owner_type', ylabel='product_wg_ton'>
```



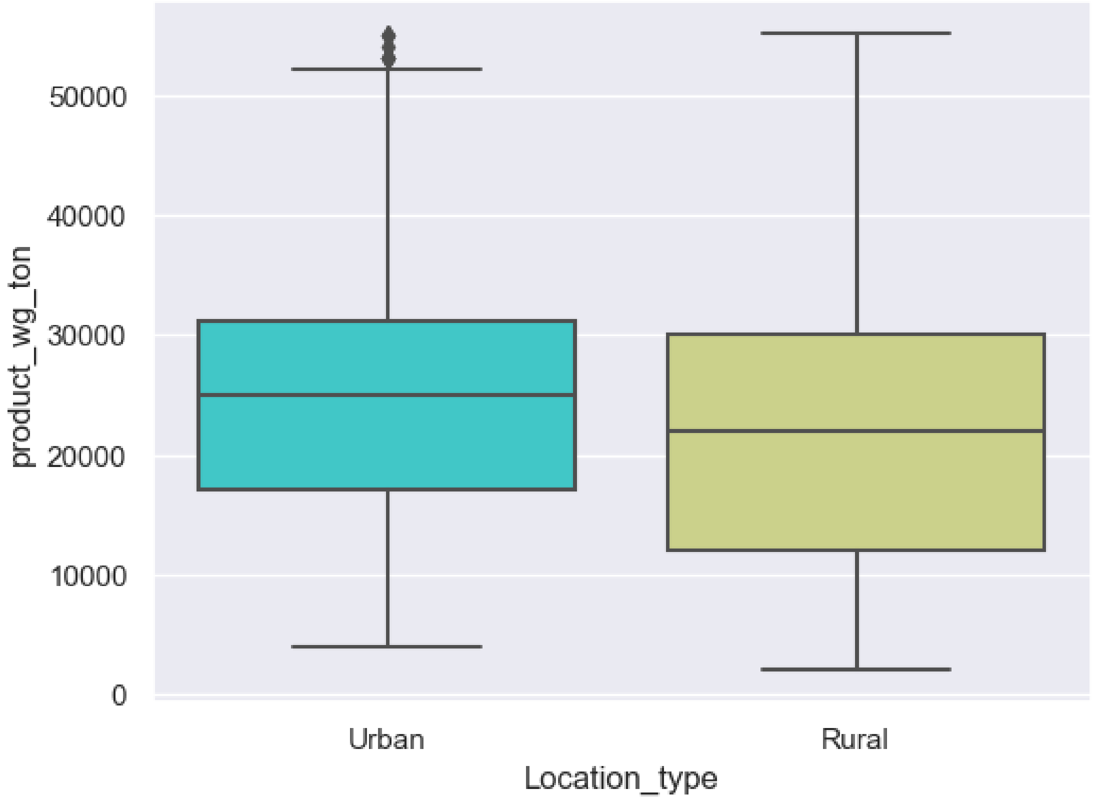

```
sns.barplot(x='approved_wh_govt_certificate',y='product_wg_ton',data=df)
```

```
<Axes: xlabel='approved_wh_govt_certificate', ylabel='product_wg_ton'>
```



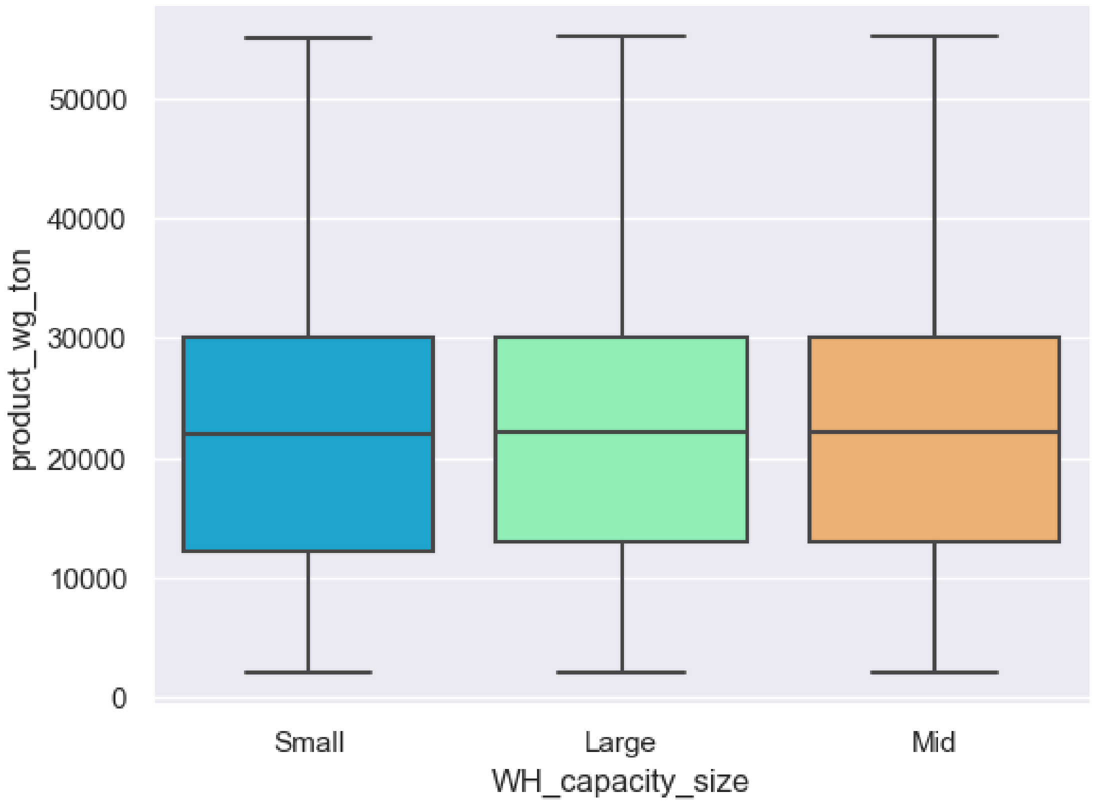

```
sns.boxplot(x='Location_type',y='product_wg_ton',data=df,palette='rainbow')
```

```
<Axes: xlabel='Location_type', ylabel='product_wg_ton'>
```



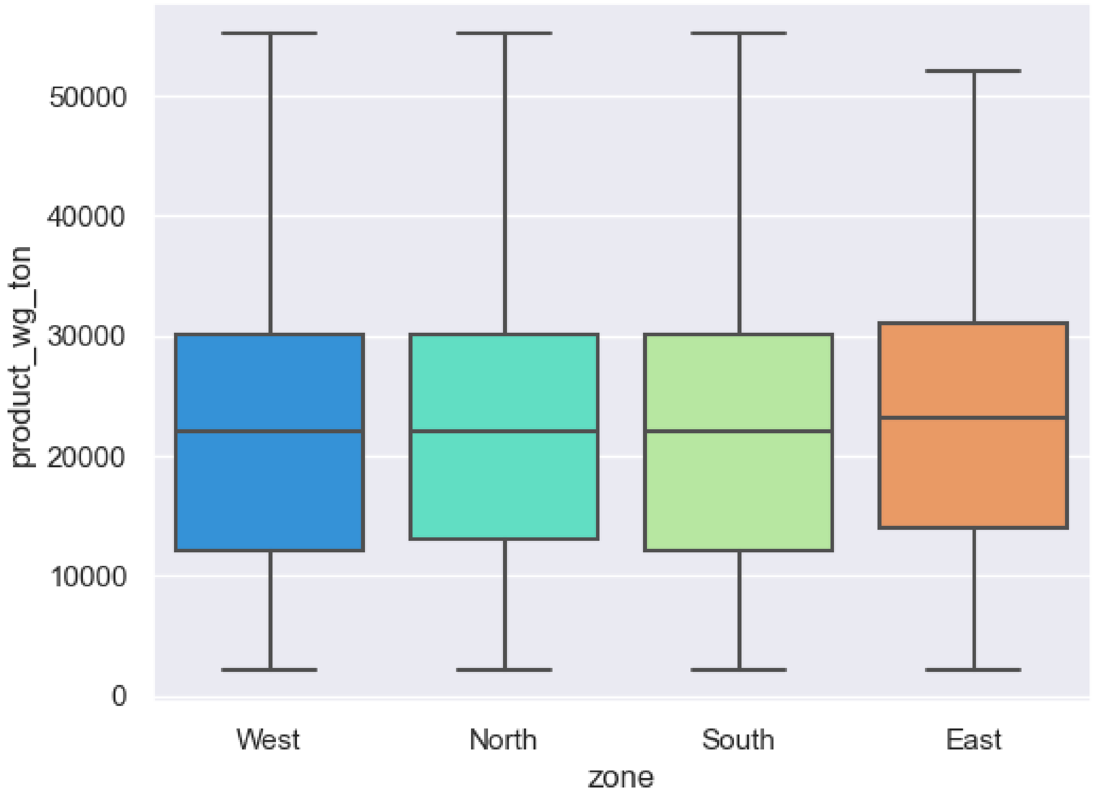

```
sns.boxplot(x='WH_capacity_size',y='product_wg_ton',data=df,palette='rainbow')
```

```
<Axes: xlabel='WH_capacity_size', ylabel='product_wg_ton'>
```



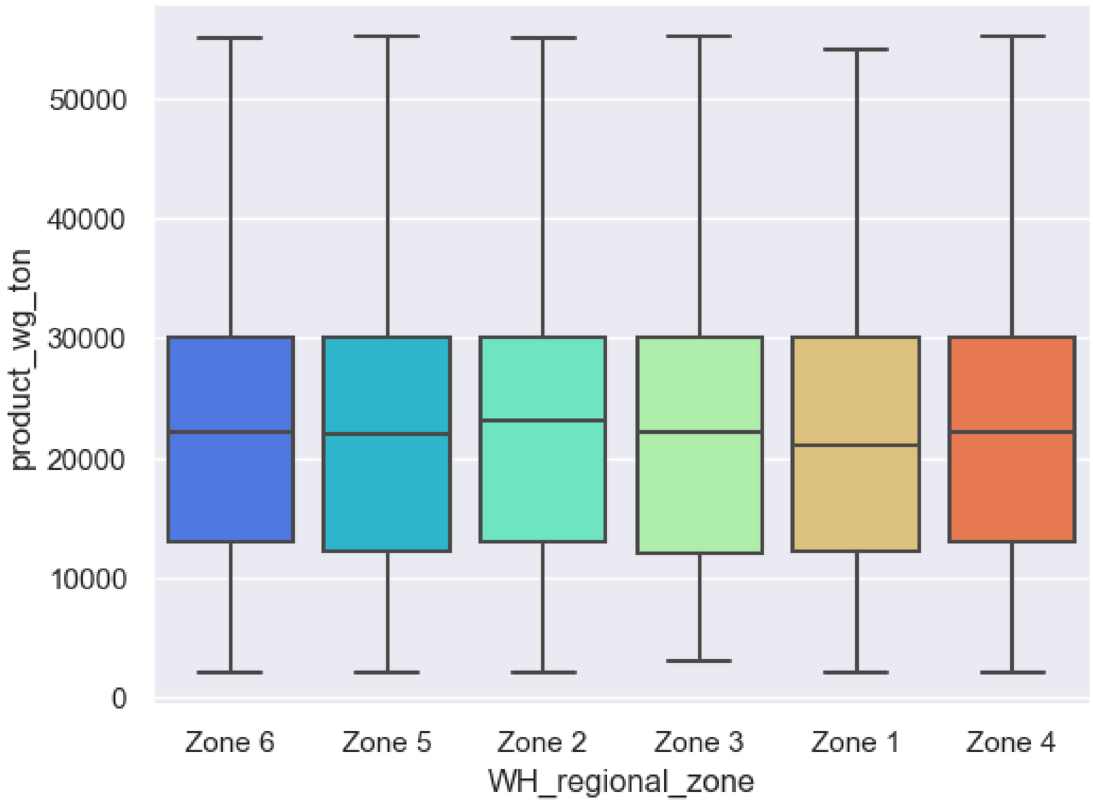

```
sns.boxplot(x='zone',y='product_wg_ton',data=df,palette='rainbow')
```

```
<Axes: xlabel='zone', ylabel='product_wg_ton'>
```



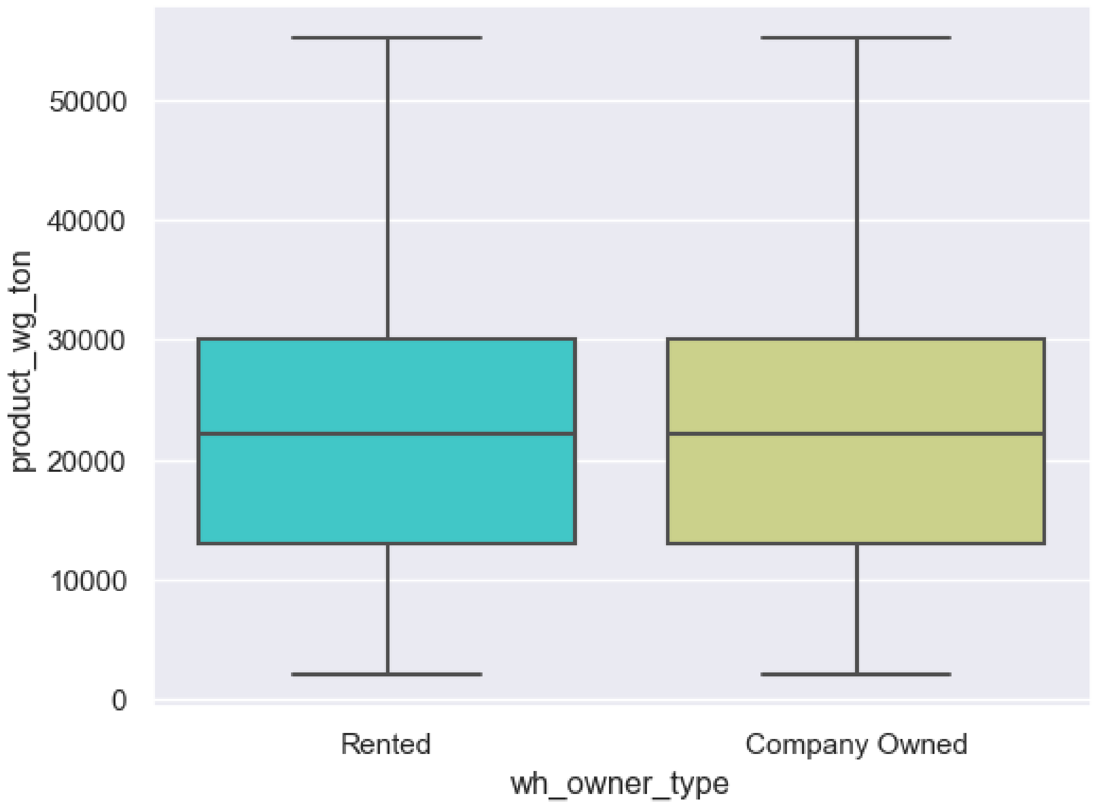

```
sns.boxplot(x='WH_regional_zone',y='product_wg_ton',data=df,palette='rainbow')
```

```
<Axes: xlabel='WH_regional_zone', ylabel='product_wg_ton'>
```



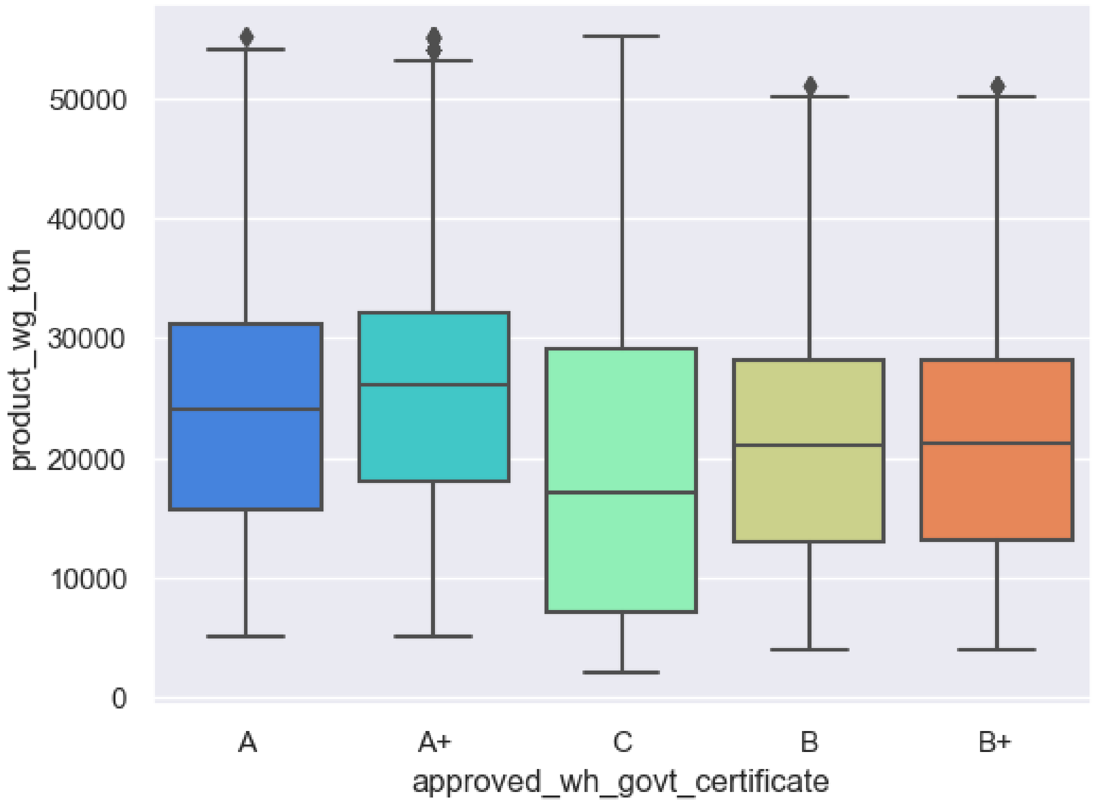

```
sns.boxplot(x='wh_owner_type',y='product_wg_ton',data=df,palette='rainbow')
```

```
<Axes: xlabel='wh_owner_type', ylabel='product_wg_ton'>
```




```
sns.boxplot(x='approved_wh_govt_certificate',y='product_wg_ton',data=df,palette='rainbow')
```

```
<Axes: xlabel='approved_wh_govt_certificate', ylabel='product_wg_ton'>
```



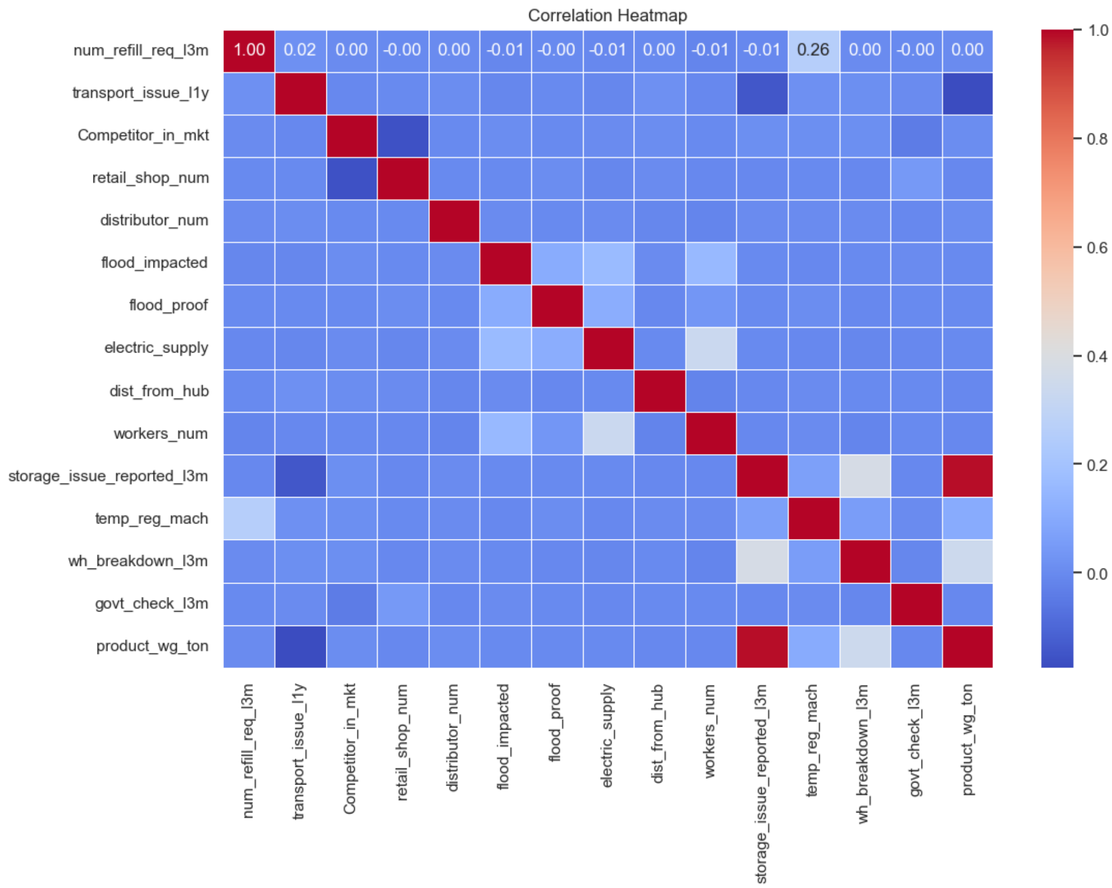
Not much be can be interpreted for categorical variables with respect to target variable as all of these are fairly distibuted

Multivariate analysis

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x205a6e44c50>
```

```
# Select only numeric columns
df_numeric = df.select_dtypes(include=["number"])

plt.figure(figsize=(12, 8))
sns.heatmap(df_numeric.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```



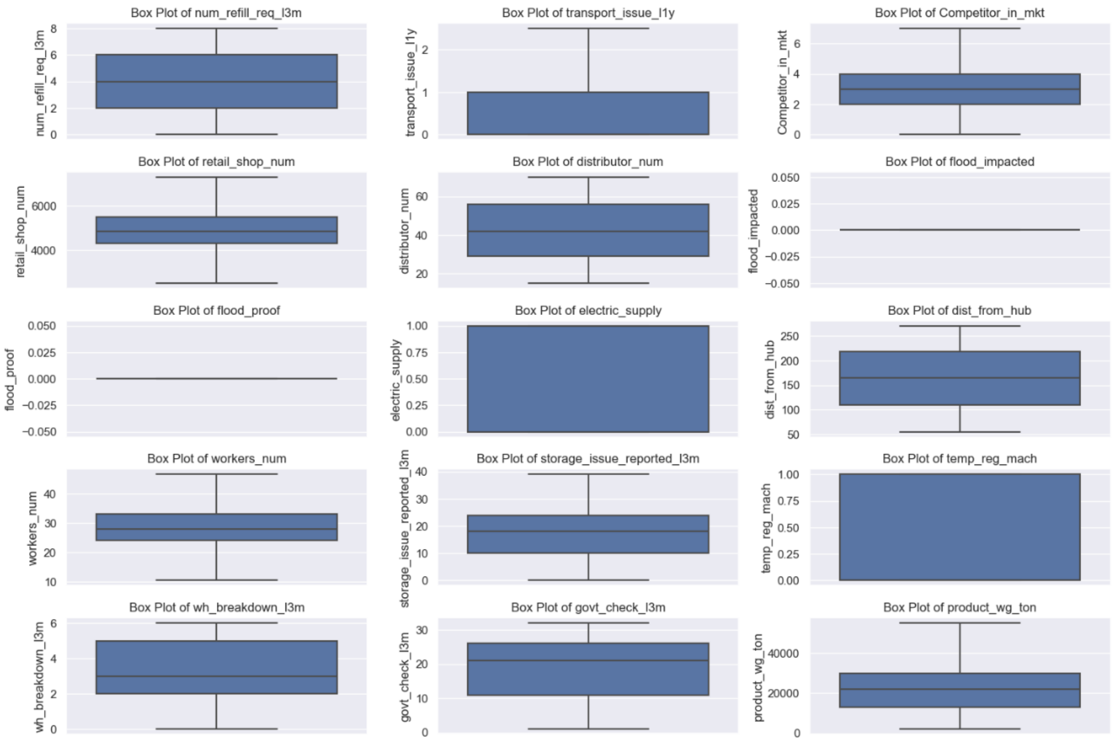
Inference from Multivariate Analysis

- "storage_issue_reported_l3m" is highly positively correlated with target variable
- "wh_breakdown_l3m" has some correlation with target variable

Preprocessing

Outlier Treatment

```
def treat_outlier_for_numerical_features(column):
    Q1=np.nanpercentile(df[column],25)
    Q2=np.nanpercentile(df[column],50)
    Q3=np.nanpercentile(df[column],75)
    IQR=Q3-Q1
    lower_limit=Q1-1.5*IQR
    upper_limit=Q3+1.5*IQR
    df[column] = np.where(df[column] > upper_limit, upper_limit,df[column])
    df[column] = np.where(df[column] < lower_limit, lower_limit,df[column])

for col in df_numeric:
    treat_outlier_for_numerical_features(col)

# Again check the distribution of numerical features after treatment of outliers
numeric_cols = df.select_dtypes(include=["number"]).columns

plt.figure(figsize=(15, 12))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(len(numeric_cols) // 3 + 1, 3, i)  # Adjust layout for better visualization
    sns.boxplot(y=df[col])
    plt.title(f"Box Plot of {col}")

plt.tight_layout()
plt.show()
```



The outliers have been succesfully treated and there are no more outliers exist.

```
df.nunique()
```

```
Location_type                    2
WH_capacity_size                 3
zone                             4
WH_regional_zone                 6
num_refill_req_l3m               9
transport_issue_l1y              4
Competitor_in_mkt                8
retail_shop_num               4151
wh_owner_type                    2
distributor_num                 56
flood_impacted                   1
flood_proof                      1
electric_supply                  2
dist_from_hub                  217
workers_num                     38
storage_issue_reported_l3m      37
temp_reg_mach                    2
approved_wh_govt_certificate     5
wh_breakdown_l3m                 7
govt_check_l3m                  32
product_wg_ton                4561
dtype: int64
```

Creating a list of features that are not needed

```
removed_features = []
```

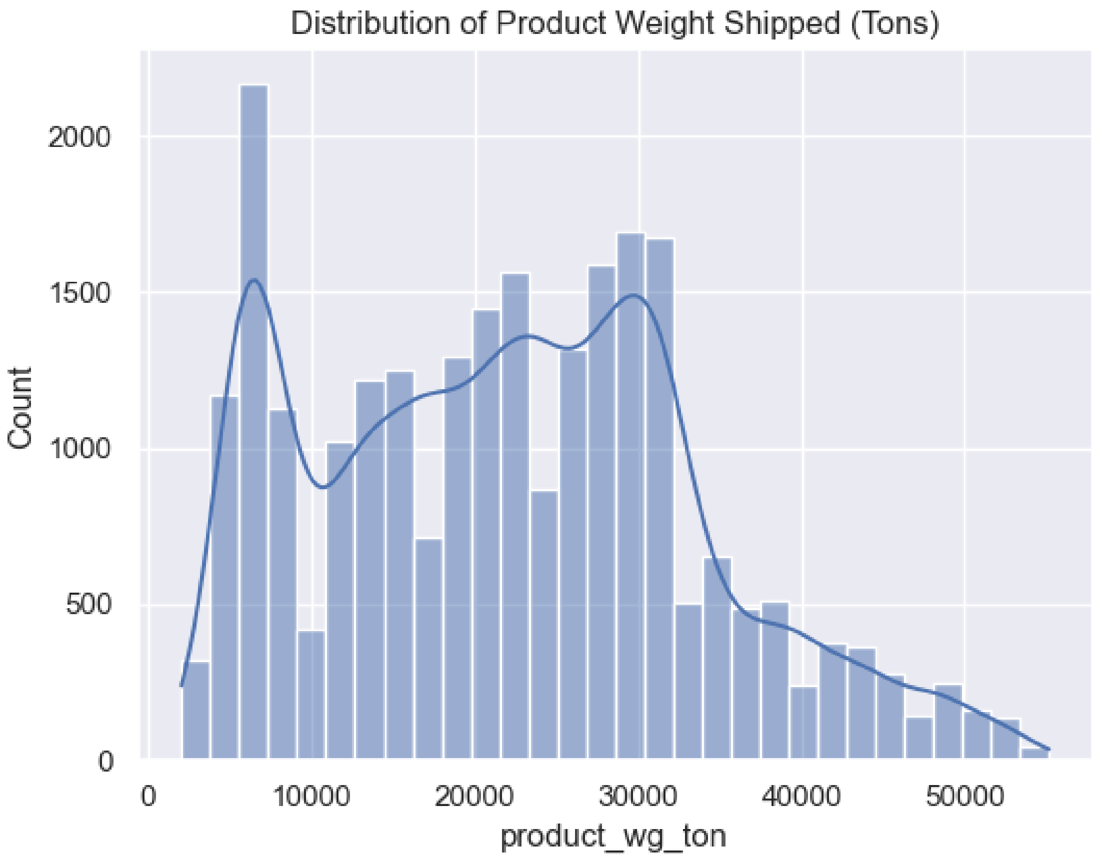
Adding below features to list removed_features:

- flood_impacted: This feature has only one unique value, therefore it does not contribute to our solution- flood_proof: This feature has only one unique value, therefore it does not contribute to our solution

```
df.drop(columns=['flood_impacted', 'flood_proof'], inplace=True)
```

Check for Data Imbalance

```
sns.histplot(df['product_wg_ton'], kde=True, bins=30)
plt.title("Distribution of Product Weight Shipped (Tons)")
plt.show()
```



Preprocessing Continues...

Encoding for Categorical Variables

categorical columns = [

- 'Location_type',
- 'WH_capacity_size',
- 'zone',
- 'WH_regional_zone',
- 'wh_owner_type',
- 'approved_wh_govt_certificate']

```
for col in columns_to_plot_categorical:
    check_value_count_for_categorical_data(col)
```

```
value_count for ' Location_type ':
 Location_type
Rural    22957
Urban     2043
Name: count, dtype: int64

-------------------------------------


value_count for ' WH_capacity_size ':
 WH_capacity_size
Large    10169
```

```
Mid       10020
Small      4811
Name: count, dtype: int64

-------------------------------------


value_count for ' zone ':
 zone
North    10278
West      7931
South     6362
East       429
Name: count, dtype: int64

-------------------------------------


value_count for ' WH_regional_zone ':
 WH_regional_zone
Zone 6    8339
Zone 5    4587
Zone 4    4176
Zone 2    2963
Zone 3    2881
Zone 1    2054
Name: count, dtype: int64

-------------------------------------


value_count for ' wh_owner_type ':
 wh_owner_type
Company Owned    13578
Rented           11422
Name: count, dtype: int64

-------------------------------------


value_count for ' approved_wh_govt_certificate ':
 approved_wh_govt_certificate
C     6409
B+    4917
B     4812
A     4671
A+    4191
Name: count, dtype: int64

-------------------------------------
```

One-Hot Encoding

Following variables can be encoded using one-hot encoding as they do not have any ordinal nature:

- Location_type
- zone

- WH_regional_zone
- wh_owner_type

```
data=pd.get_dummies(df, columns=['Location_type', 'zone', 'WH_regional_zone',
'wh_owner_type'],drop_first=True)
```

Label Encoding

Following variables can be encoded using label encoding as they have ordinal nature:

- WH_capacity_size
- approved_wh_govt_certificate

```
from sklearn import preprocessing
def label_encoding(column):
    label_encoder = preprocessing.LabelEncoder()
    data[column] = label_encoder.fit_transform(data[column])

for col in ['WH_capacity_size', 'approved_wh_govt_certificate']:
    label_encoding(col)
```

Checking the data again after preprocessing and encoding

```
print(data.columns.tolist())
data.head()
```

```
['WH_capacity_size', 'num_refill_req_l3m', 'transport_issue_l1y', 'Competitor_in_mkt',
'retail_shop_num', 'distributor_num', 'electric_supply', 'dist_from_hub', 'workers_num',
'storage_issue_reported_l3m', 'temp_reg_mach', 'approved_wh_govt_certificate',
'wh_breakdown_l3m', 'govt_check_l3m', 'product_wg_ton', 'Location_type_Urban', 'zone_North',
'zone_South', 'zone_West', 'WH_regional_zone_Zone 2', 'WH_regional_zone_Zone 3',
'WH_regional_zone_Zone 4', 'WH_regional_zone_Zone 5', 'WH_regional_zone_Zone 6',
'wh_owner_type_Rented']
```

```
   WH_capacity_size  num_refill_req_l3m  transport_issue_l1y  \
0                 2                 3.0                  1.0
1                 0                 0.0                  0.0
2                 1                 1.0                  0.0
3                 1                 7.0                  2.5
4                 0                 3.0                  1.0

   Competitor_in_mkt  retail_shop_num  distributor_num  electric_supply  \
0                2.0           4651.0             24.0              1.0
1                4.0           6217.0             47.0              1.0
2                4.0           4306.0             64.0              0.0
3                2.0           6000.0             50.0              0.0
4                2.0           4740.0             42.0              1.0

   dist_from_hub  workers_num  storage_issue_reported_l3m  temp_reg_mach  \
0           91.0         29.0                        13.0            0.0
1          210.0         31.0                         4.0            0.0
2          161.0         37.0                        17.0            0.0
3          103.0         21.0                        17.0            1.0
4          112.0         25.0                        18.0            0.0

   approved_wh_govt_certificate  wh_breakdown_l3m  govt_check_l3m  \
0                             0               5.0            15.0
1                             0               3.0            17.0
2                             0               6.0            22.0
3                             1               3.0            27.0
4                             4               6.0            24.0

   product_wg_ton  Location_type_Urban  zone_North  zone_South  zone_West  \
```

```
0        17115.0                  True        False        False        True
1         5074.0                 False         True        False       False
2        23137.0                 False        False         True       False
3        22115.0                 False         True        False       False
4        24071.0                 False         True        False       False

   WH_regional_zone_Zone 2  WH_regional_zone_Zone 3  WH_regional_zone_Zone 4  \
0                    False                    False                    False
1                    False                    False                    False
2                     True                    False                    False
3                    False                     True                    False
4                    False                    False                    False

   WH_regional_zone_Zone 5  WH_regional_zone_Zone 6  wh_owner_type_Rented
0                    False                     True                  True
1                     True                    False                 False
2                    False                    False                 False
3                    False                    False                  True
4                     True                    False                 False
```

Train-Test Split

Remove target col from feature dataframe

```
X = data.drop(['product_wg_ton'], axis=1)
y = data['product_wg_ton']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)
print("Dimension of X_train:", X_train.shape)
print("Dimension of X_test:", X_test.shape)

Dimension of X_train: (20000, 24)
Dimension of X_test: (5000, 24)
```

Standardisation after Train Test Split

```
from sklearn.preprocessing import StandardScaler
non_categorical_columns = [
    'num_refill_req_l3m',
    'transport_issue_l1y',
    'Competitor_in_mkt',
    'retail_shop_num',
    'distributor_num',
    'electric_supply',
    'dist_from_hub',
    'workers_num',
    'storage_issue_reported_l3m',
    'temp_reg_mach',
    'wh_breakdown_l3m',
    'govt_check_l3m']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=42)

scaler = StandardScaler()
X_train[non_categorical_columns] = scaler.fit_transform(X_train[non_categorical_columns])

# using same fit params of train data for the test data to avoid data leakage
X_test[non_categorical_columns] = scaler.transform(X_test[non_categorical_columns])

X_train.head()
```

```
       WH_capacity_size  num_refill_req_l3m  transport_issue_l1y  \
23311                 0           -1.188838             0.372996   
23623                 1            1.496848            -0.715249   
1020                  0            1.496848            -0.715249   
12645                 0            1.496848             0.372996   
1533                  0           -0.037830            -0.715249   

       Competitor_in_mkt  retail_shop_num  distributor_num  electric_supply  \
23311           0.815374        -0.908895         1.033787         0.719661   
23623          -0.083555         1.155974         0.659390         0.719661   
1020            0.815374        -0.891326         0.284994         0.719661   
12645          -0.982485        -0.517210         0.596991         0.719661   
1533            0.815374        -0.145162         0.596991        -1.389544   

       dist_from_hub  workers_num  storage_issue_reported_l3m  temp_reg_mach  \
23311       1.193945    -0.949013                    1.184447      -0.657149   
23623       0.954705     0.171874                    2.165712      -0.657149   
1020       -0.018206    -0.388570                    0.748329       1.521726   
12645      -0.624282     1.572983                    0.966388      -0.657149   
1533       -0.177700    -1.509457                    0.203182      -0.657149   

       approved_wh_govt_certificate  wh_breakdown_l3m  govt_check_l3m  \
23311                             3          0.899010       -1.490176   
23623                             0          0.306289        0.946117   
1020                              1         -0.286433        0.482061   
12645                             2         -0.286433        0.714089   
1533                              2          0.899010       -1.490176   

       Location_type_Urban  zone_North  zone_South  zone_West  \
23311                False       False       False       True   
23623                False        True       False      False   
1020                 False        True       False      False   
12645                False       False        True      False   
1533                 False        True       False      False   

       WH_regional_zone_Zone 2  WH_regional_zone_Zone 3  \
23311                    False                    False   
23623                    False                     True   
1020                     False                    False   
12645                    False                    False   
1533                     False                    False   

       WH_regional_zone_Zone 4  WH_regional_zone_Zone 5  \
23311                    False                    False   
23623                    False                    False   
1020                     False                    False   
12645                    False                     True   
1533                     False                    False   

       WH_regional_zone_Zone 6  wh_owner_type_Rented  
23311                     True                 False  
23623                    False                  True  
1020                      True                 False  
12645                    False                 False  
1533                      True                  True  
```

Building different ML Models

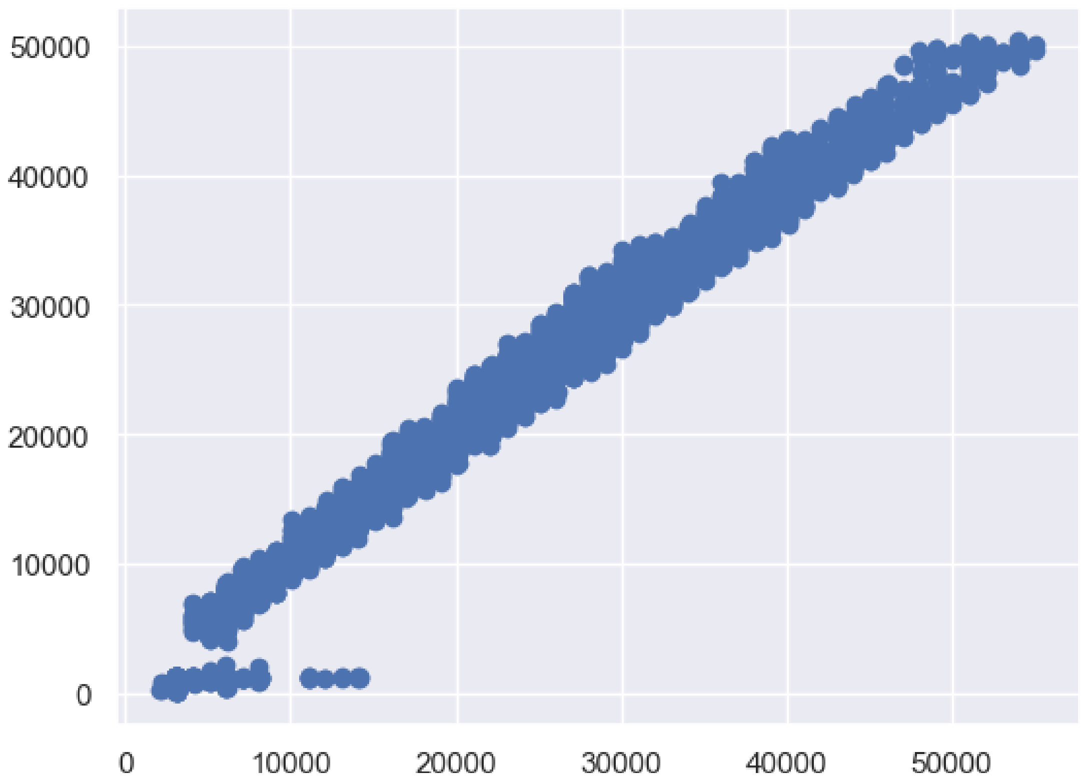
```python
def create_and_evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
```

```
    pred = model.predict(X_test)
    print(model, ' MAE:', metrics.mean_absolute_error(y_test, pred))
    print('--'*50)
    print(model, ' MSE:', metrics.mean_squared_error(y_test, pred))
    print('--'*50)
    print(model, ' R-Squared:', metrics.r2_score(y_test, pred))
    print('--'*50)
    plt.scatter(y_test,pred)
```

Linear Regression Model

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
create_and_evaluate_model(LinearRegression(), X_train, y_train, X_test, y_test)
```

```
LinearRegression()  MAE: 1303.975230024748
----------------------------------------------------------------------------------------------------
LinearRegression()  MSE: 3093785.2900887085
----------------------------------------------------------------------------------------------------
LinearRegression()  R-Squared: 0.9768775144228907
----------------------------------------------------------------------------------------------------
```

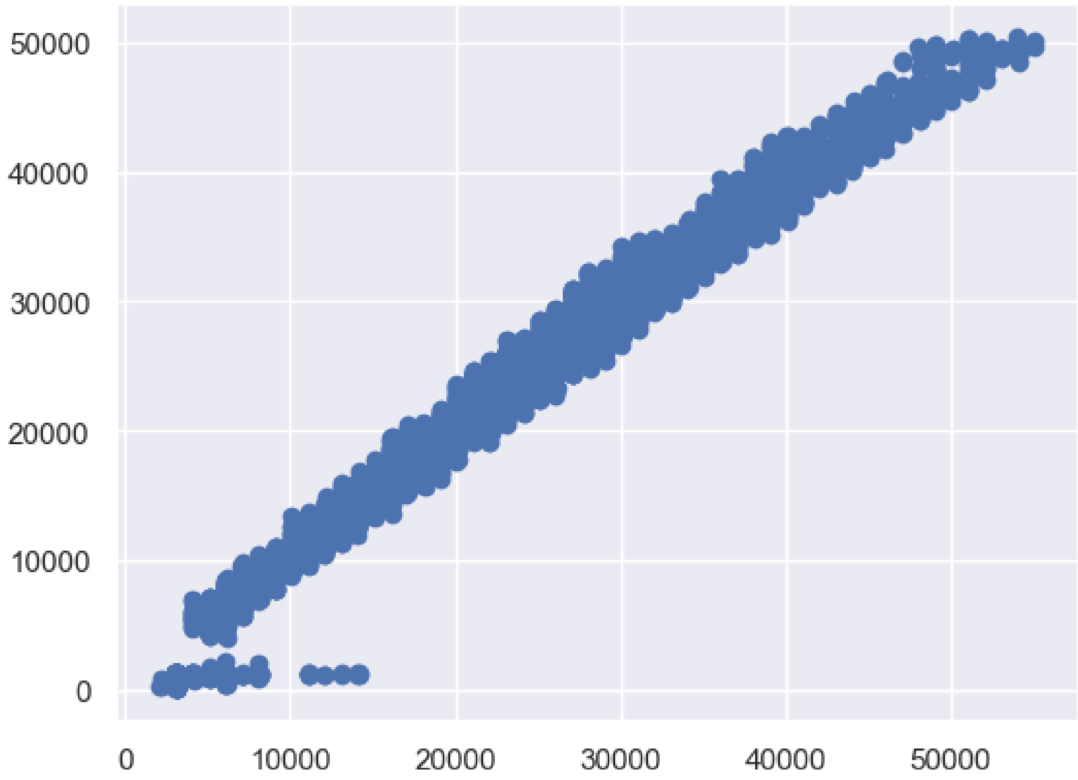


Ridge Regression

```
from sklearn.linear_model import Ridge
create_and_evaluate_model(Ridge(), X_train, y_train, X_test, y_test)
```

```
Ridge()  MAE: 1304.012858037786
----------------------------------------------------------------------------------------------------
Ridge()  MSE: 3093803.482199039
----------------------------------------------------------------------------------------------------
Ridge()  R-Squared: 0.9768773784577964
```

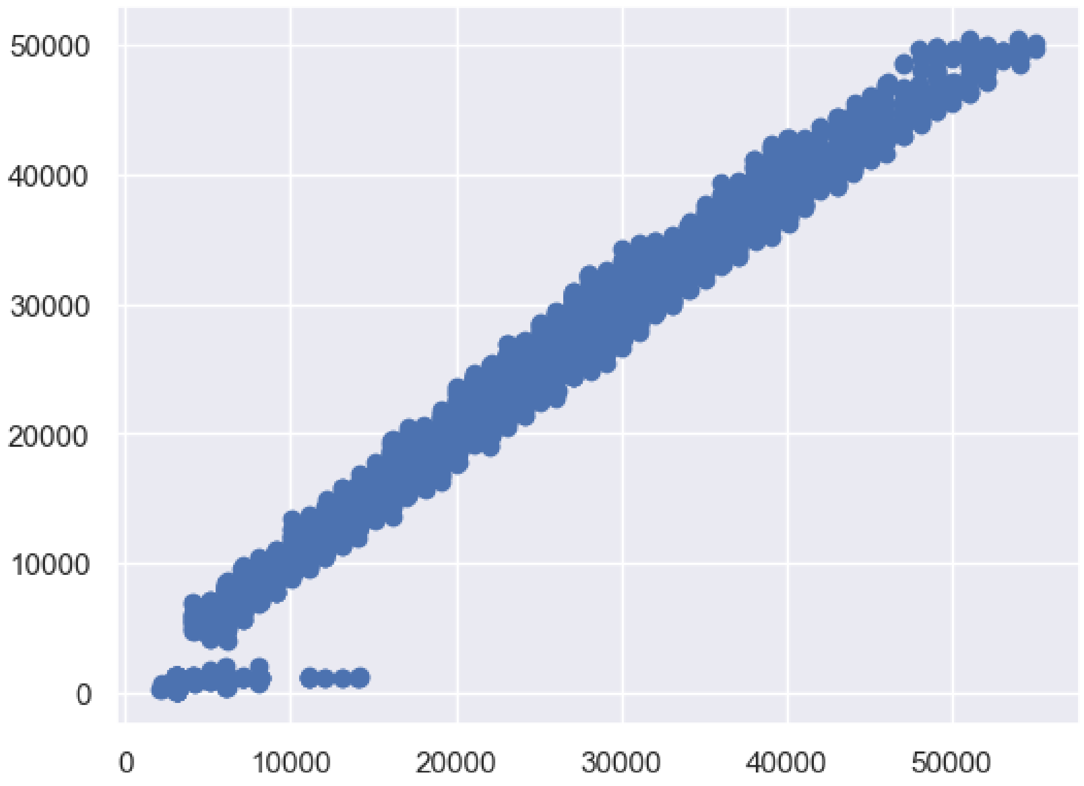
```
----------------------------------------------------------------------------------------------------
------
```



Lasso Regression

```
from sklearn.linear_model import Lasso
from sklearn import linear_model
create_and_evaluate_model(linear_model.Lasso(), X_train, y_train, X_test, y_test)
```

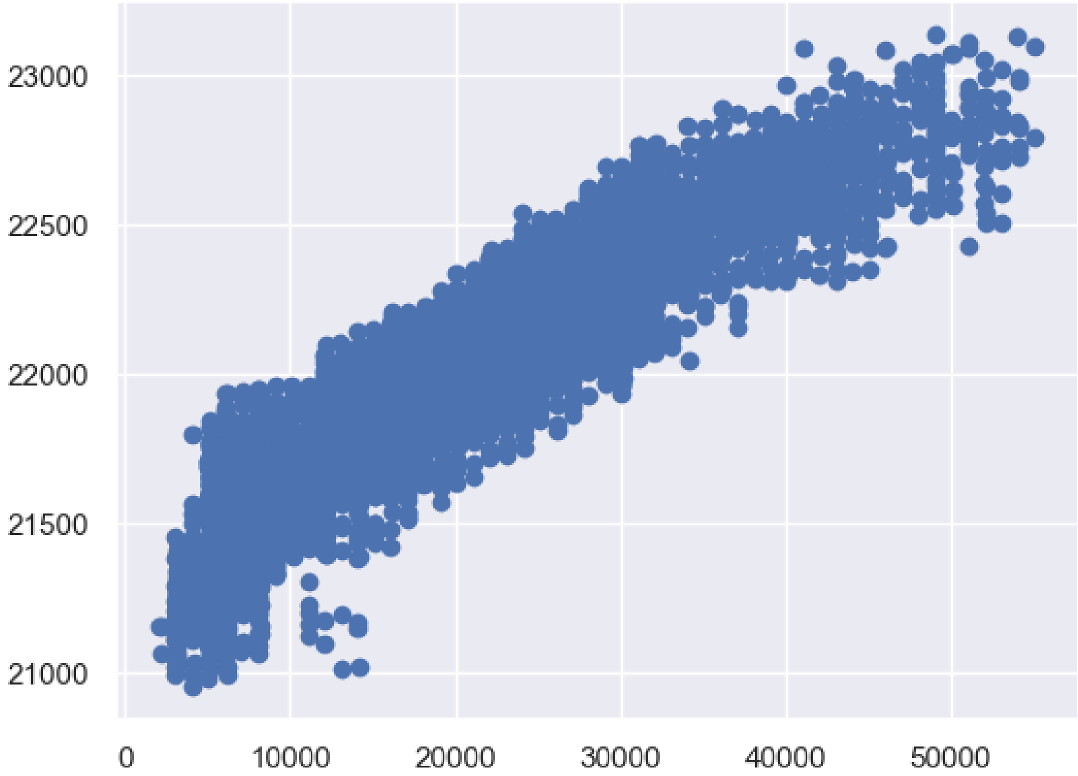
```
Lasso()  MAE: 1303.6572451266954
----------------------------------------------------------------------------------------------------
------
Lasso()  MSE: 3092922.220914573
----------------------------------------------------------------------------------------------------
------
Lasso()  R-Squared: 0.9768839648719878
----------------------------------------------------------------------------------------------------
------
```



SVR

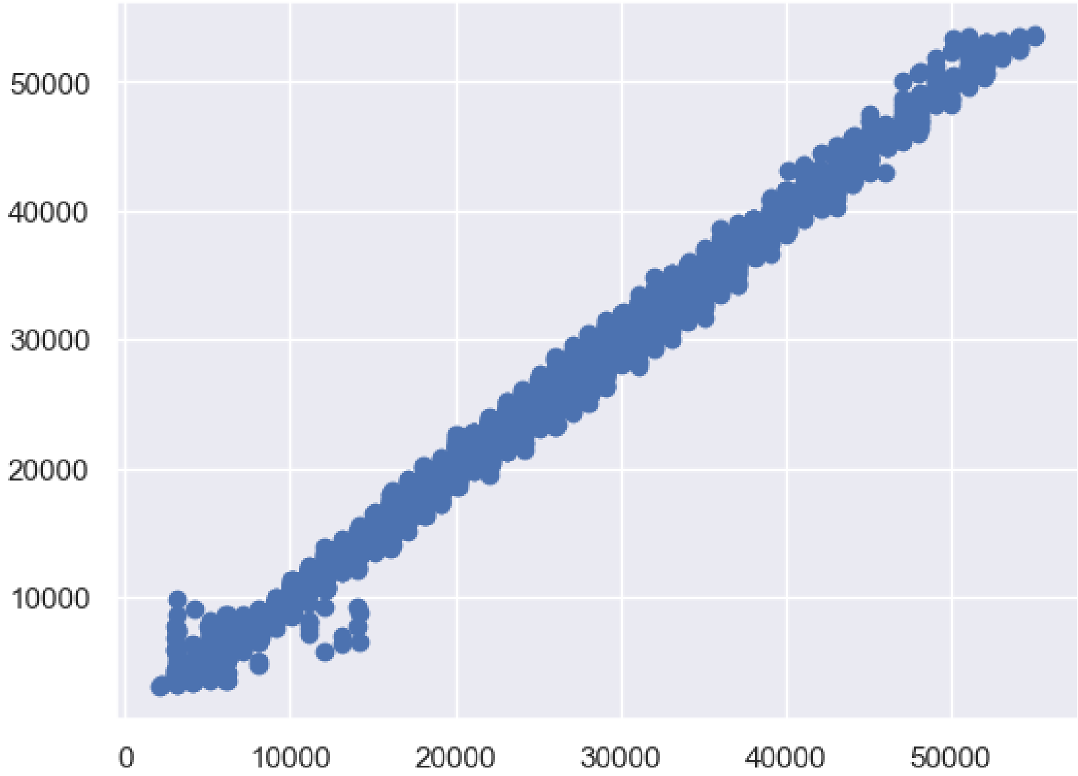
```
from sklearn.svm import SVR
create_and_evaluate_model(SVR(), X_train, y_train, X_test, y_test)
```

```
SVR()  MAE: 9201.023627557543
----------------------------------------------------------------------------------------------------
------
SVR()  MSE: 124884627.78575395
----------------------------------------------------------------------------------------------------
------
SVR()  R-Squared: 0.06663108974315357
----------------------------------------------------------------------------------------------------
------
```



RandomForest Regresson

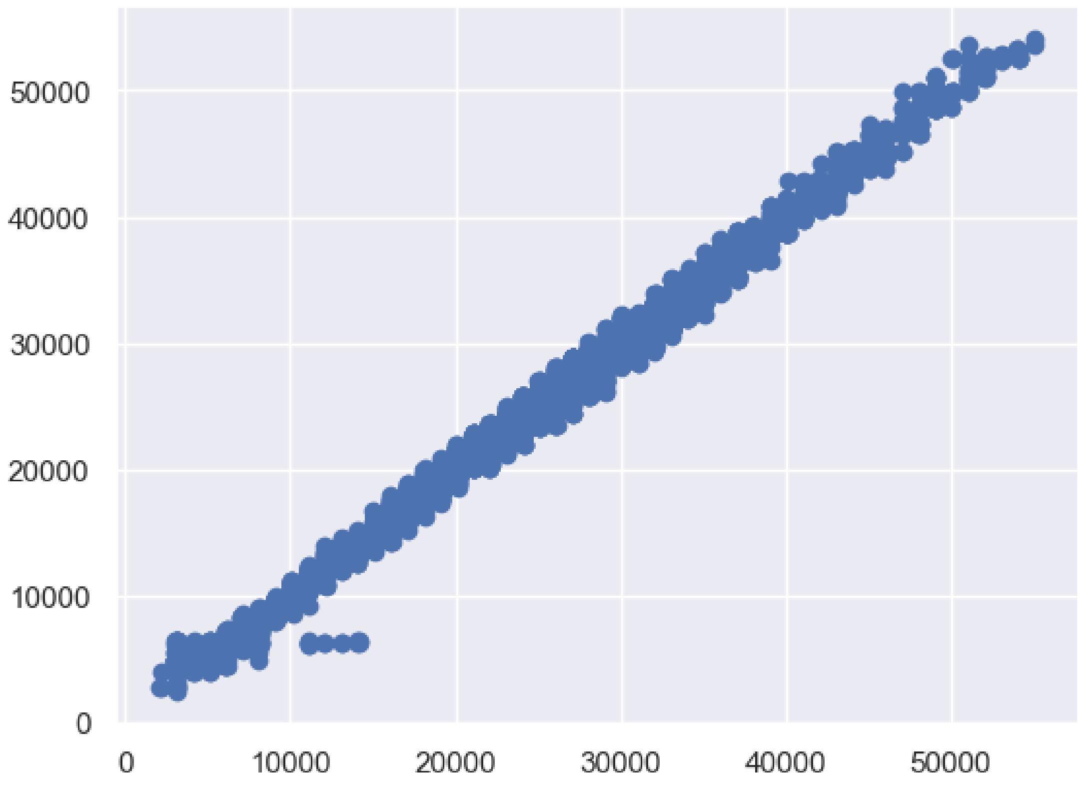
```
from sklearn.ensemble import RandomForestRegressor
create_and_evaluate_model(RandomForestRegressor(), X_train, y_train, X_test, y_test)

RandomForestRegressor()  MAE: 700.459422
------------------------------------------------------------------------------------------------
------
RandomForestRegressor()  MSE: 886565.92105138
------------------------------------------------------------------------------------------------
------
RandomForestRegressor()  R-Squared: 0.9933739397532402
------------------------------------------------------------------------------------------------
------
```



Gradient Boosting

```
from sklearn import ensemble
create_and_evaluate_model(ensemble.GradientBoostingRegressor(), X_train, y_train, X_test,
y_test)
```

```
GradientBoostingRegressor()  MAE: 689.4401832979134
----------------------------------------------------------------------------------------------
------
GradientBoostingRegressor()  MSE: 836425.3304306811
----------------------------------------------------------------------------------------------
------
GradientBoostingRegressor()  R-Squared: 0.9937486829803054
----------------------------------------------------------------------------------------------
------
```



Infference

The Gradient Boosting Regressor is the best-performing model for this dataset, as it has the lowest error (MAE & MSE) and the highest R^2.For the best predictive performance, Gradient Boosting should be used.

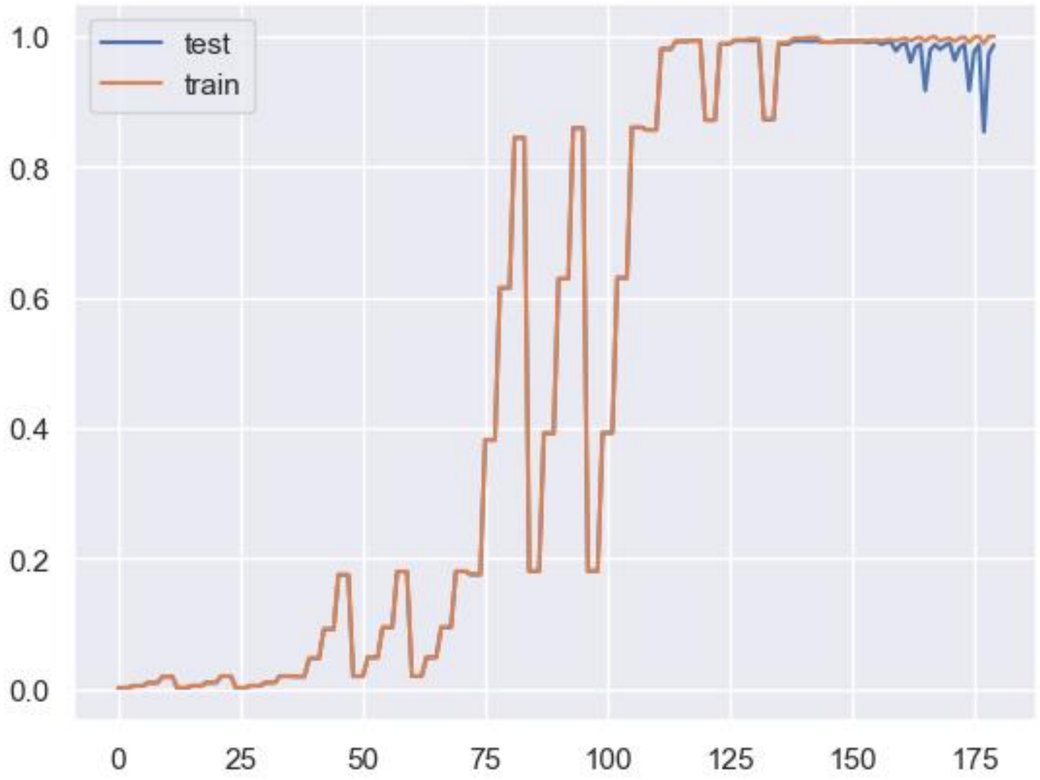
Hyperparameter Tuning

- Best top two performing models are Gradient boosting & Random forest & we'll try to optimize them.

Gradient Boost

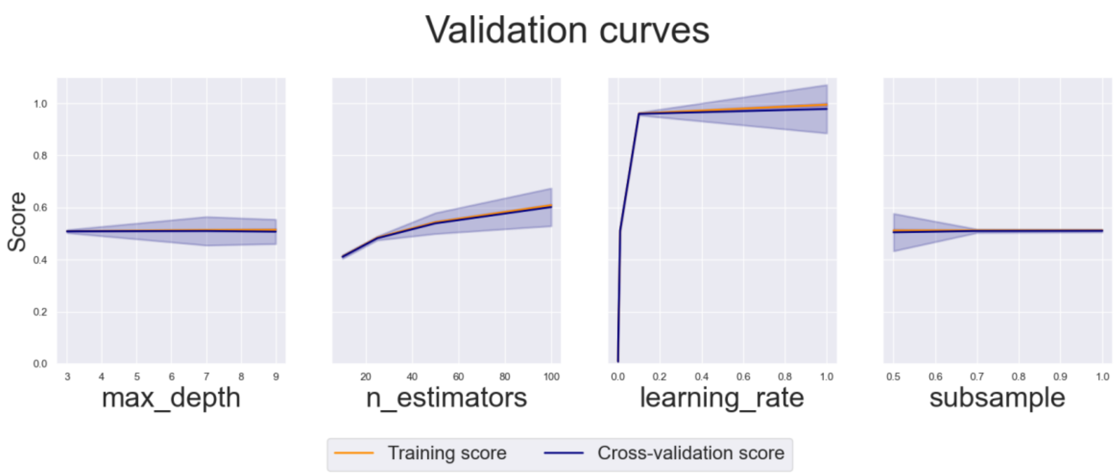
```
grid_params = {

    'max_depth': [3, 7, 9],
    'n_estimators': [10, 25, 50, 100],
    'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0],
    'subsample': [0.5, 0.7, 1.0]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = ensemble.GradientBoostingRegressor(),
                           param_grid = grid_params,
                           cv = 10,
                           n_jobs = -1,
                           verbose = 1,
                           return_train_score=True)

# Fit the model
grid_search.fit(X_train, y_train)
print("Best score: ", grid_search.best_score_)
print("Best param: ", grid_search.best_params_)

Fitting 10 folds for each of 180 candidates, totalling 1800 fits
Best score:  0.9936400690380257
Best param:  {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 50, 'subsample': 1.0}
```

```
test_scores = grid_search.cv_results_['mean_test_score']
train_scores = grid_search.cv_results_['mean_train_score']

plt.plot(test_scores, label='test')
plt.plot(train_scores, label='train')
plt.legend(loc='best')
plt.show()
```




```
df = pd.DataFrame(grid_search.cv_results_)
results = ['mean_test_score',
           'mean_train_score',
           'std_test_score',
           'std_train_score']


def pooled_var(stds):
    n = 10 # size of each group
    return np.sqrt(sum((n-1)*(stds**2))/ len(stds)*(n-1))


fig, axes = plt.subplots(1, len(grid_params),
                         figsize = (5*len(grid_params), 7),
                         sharey='row')
axes[0].set_ylabel("Score", fontsize=25)
lw = 2

for idx, (param_name, param_range) in enumerate(grid_params.items()):
    grouped_df = df.groupby(f'param_{param_name}')[results]\
        .agg({'mean_train_score': 'mean',
              'mean_test_score': 'mean',
              'std_train_score': pooled_var,
              'std_test_score': pooled_var})

    previous_group = df.groupby(f'param_{param_name}')[results]
    axes[idx].set_xlabel(param_name, fontsize=30)
```

```
    axes[idx].set_ylim(0.0, 1.1)
    axes[idx].plot(param_range,
                   grouped_df['mean_train_score'],
                   label="Training score",
                   color="darkorange",
                   lw=lw)
    axes[idx].fill_between(param_range,
                   grouped_df['mean_train_score'] - grouped_df['std_train_score'],
                   grouped_df['mean_train_score'] + grouped_df['std_train_score'],
                   alpha=0.2,
                   color="indigo",
                   lw=lw)
    axes[idx].plot(param_range,
                   grouped_df['mean_test_score'],
                   label="Cross-validation score",
                   color="navy",
                   lw=lw)
    axes[idx].fill_between(param_range,
                       grouped_df['mean_test_score'] - grouped_df['std_test_score'],
                       grouped_df['mean_test_score'] + grouped_df['std_test_score'],
                       alpha=0.2,
                       color="navy",
                       lw=lw)

handles, labels = axes[0].get_legend_handles_labels()
fig.suptitle('Validation curves', fontsize=40)
fig.legend(handles, labels, loc=8, ncol=2, fontsize=20)

fig.subplots_adjust(bottom=0.25, top=0.85)
plt.show()
```



applying best params of Gradientboost

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
def evaluate_gradient_boosting_with_params(X_train, y_train, X_test, y_test):
    """
    Trains and evaluates a Gradient Boosting Regressor model with the provided best
parameters,
    and returns MAE, MSE, and R-squared.

    Args:
        X_train: Training features.
        y_train: Training target.
        X_test: Testing features.
        y_test: Testing target.
```

```
    Returns:
        A dictionary containing MAE, MSE, and R-squared.
    """
    best_params = {
        'learning_rate': 0.1,
        'max_depth': 7,
        'n_estimators': 50,
        'subsample': 1.0,
    }

    model = GradientBoostingRegressor(**best_params)

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return {
        "mae": mae,
        "mse": mse,
        "r2": r2,
    }

results = evaluate_gradient_boosting_with_params(X_train, y_train, X_test, y_test)
print("MAE:", results["mae"])
print("MSE:", results["mse"])
print("R-squared:", results["r2"])
```

```
MAE: 679.4282484982145
MSE: 820252.1652334589
R-squared: 0.9938695587825813
```

Random Forest

```
grid_params = {

    'max_depth': [2, 5, 10, 20, 30],
    'min_samples_leaf': [5, 10, 20, 50],
    'max_features': ['sqrt', 'log2'],
    'n_estimators': [10, 25, 50, 100, 200]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = RandomForestRegressor(),
                           param_grid = grid_params,
                           cv = 10,
                           n_jobs = -1,
                           verbose = 1,
                           return_train_score=True)

# Fit the model
grid_search.fit(X_train, y_train)
print("Best score: ", grid_search.best_score_)
print("Best param: ", grid_search.best_params_)
```

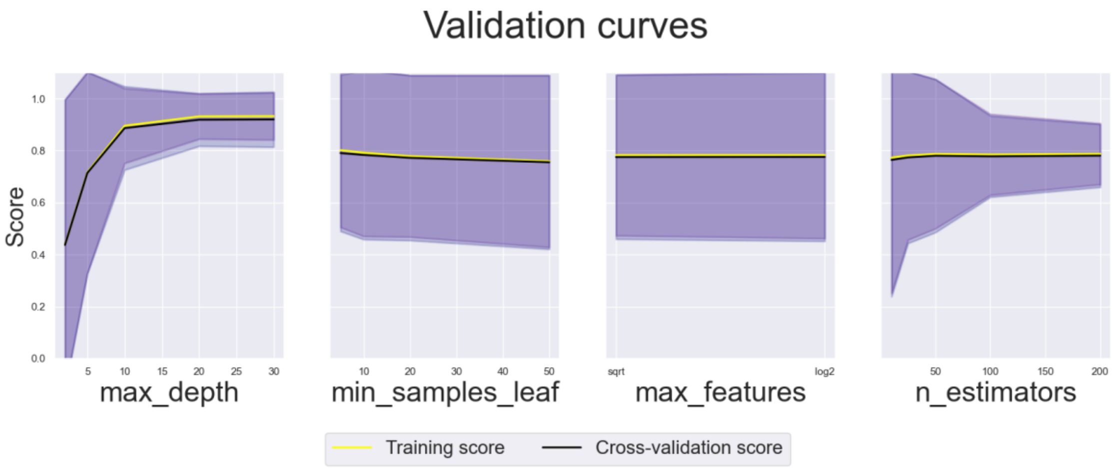
```
Fitting 10 folds for each of 200 candidates, totalling 2000 fits
Best score:  0.9517636918726804
Best param:  {'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 5, 'n_estimators':
100}
```

```
test_scores = grid_search.cv_results_['mean_test_score']
train_scores = grid_search.cv_results_['mean_train_score']

plt.plot(test_scores, label='test')
plt.plot(train_scores, label='train')
plt.legend(loc='best')
plt.show()
```

- Both train and test scores are overlapping, therefore there is no overfitting

```
df = pd.DataFrame(grid_search.cv_results_)
results = ['mean_test_score',
           'mean_train_score',
           'std_test_score',
           'std_train_score']


def pooled_var(stds):
    n = 10 # size of each group
    return np.sqrt(sum((n-1)*(stds**2))/ len(stds)*(n-1))


fig, axes = plt.subplots(1, len(grid_params),
                         figsize = (5*len(grid_params), 7),
                         sharey='row')
axes[0].set_ylabel("Score", fontsize=25)
lw = 2

for idx, (param_name, param_range) in enumerate(grid_params.items()):
    grouped_df = df.groupby(f'param_{param_name}')[results]\
        .agg({'mean_train_score': 'mean',
              'mean_test_score': 'mean',
              'std_train_score': pooled_var,
              'std_test_score': pooled_var})

    previous_group = df.groupby(f'param_{param_name}')[results]
```

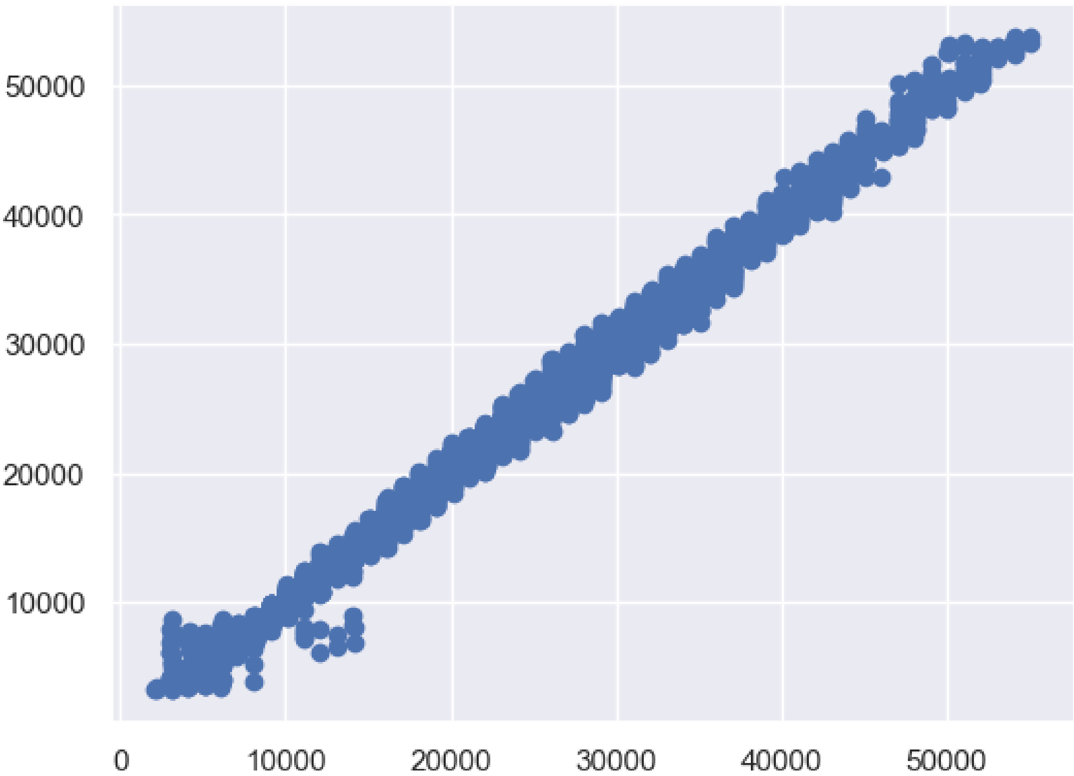
```
    axes[idx].set_xlabel(param_name, fontsize=30)
    axes[idx].set_ylim(0.0, 1.1)
    axes[idx].plot(param_range,
                grouped_df['mean_train_score'],
                label="Training score",
                color="yellow",
                lw=lw)
    axes[idx].fill_between(param_range,
                grouped_df['mean_train_score'] - grouped_df['std_train_score'],
                grouped_df['mean_train_score'] + grouped_df['std_train_score'],
                alpha=0.2,
                color="indigo",
                lw=lw)
    axes[idx].plot(param_range,
                grouped_df['mean_test_score'],
                label="Cross-validation score",
                color="black",
                lw=lw)
    axes[idx].fill_between(param_range,
                    grouped_df['mean_test_score'] - grouped_df['std_test_score'],
                    grouped_df['mean_test_score'] + grouped_df['std_test_score'],
                    alpha=0.2,
                    color="navy",
                    lw=lw)

handles, labels = axes[0].get_legend_handles_labels()
fig.suptitle('Validation curves', fontsize=40)
fig.legend(handles, labels, loc=8, ncol=2, fontsize=20)

fig.subplots_adjust(bottom=0.25, top=0.85)
plt.show()
```



applying best params of RandomForest

```
create_and_evaluate_model(RandomForestRegressor(bootstrap=True,
                                max_depth=grid_search.best_params_['max_depth'],
                                min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
                                n_estimators=grid_search.best_params_['n_estimators']), X_train,
y_train, X_test, y_test)
```

```
RandomForestRegressor(max_depth=30, min_samples_leaf=5)  MAE: 691.3156382494558
--------------------------------------------------------------------------------------------------
------
RandomForestRegressor(max_depth=30, min_samples_leaf=5)  MSE: 851405.3583082906
--------------------------------------------------------------------------------------------------
------
RandomForestRegressor(max_depth=30, min_samples_leaf=5)  R-Squared: 0.9936367245067635
```


50000
40000
30000
20000
10000
0
10000
20000
30000
40000
50000