SUPER MART SALES FORECAST

Name : Rajnish Bharti

Reg. No.: 12015883

Roll No.: B60

Subject: CSM423(Machine Learning)

Submitted To: Aishwary Shukla

Super Mart Sales Forecasting

Rajnish Bharti

12015883

Department of Computer Science
Lovely Professional University, Phagwara Punjab

***Abstract*— Abstract**

This project report presents the development and implementation of a machine learning-based sales forecasting model for Big Mart, a prominent retail chain. Accurate sales forecasting is crucial for optimizing inventory management and business decision-making. The project leverages historical sales data and employs the Autoregressive Integrated Moving Average (ARIMA) method to predict future sales. Data preprocessing, model selection, and validation are discussed. Results indicate that the ARIMA model effectively captures seasonality and trends in sales data, leading to improved inventory management and sales strategy. The report also discusses the business implications and future recommendations for enhancing the forecasting model. This project demonstrates the potential of machine learning in the retail industry and its continuous development for improved forecasting.

***Keywords*— Regression, Random Forest, Exploratory Data Analysis**

I. INTRODUCTION

The retail landscape has witnessed a profound transformation in recent years, marked by the emergence of retail giants, one of which is our hypothetical entity, "Super Mart." Super Mart, a contemporary retail chain, operates a network of stores that cater to a diverse clientele with an extensive range of products, from groceries and electronics to clothing and household goods. In the ever-evolving retail industry, the ability to effectively predict and optimize sales is crucial to not only maintain a competitive edge but also to ensure efficient inventory management and the highest level of customer satisfaction.

Super Mart's success and profitability depend on the precision of its sales forecasting. By predicting customer demand accurately, Super Mart can align its inventory levels, restocking schedules, and sales strategies with real-time market dynamics, thereby mitigating costs, eliminating stockouts, and ultimately boosting profitability. Traditional sales forecasting methods, which rely on manual calculations and historical sales trends, often fall short in today's complex, data-driven retail environment. The limitations of these conventional approaches have motivated Super Mart to embark on the "Sales Forecasting for Super Mart" project.

II. RELATED WORK

Sales forecasting in the retail industry has evolved significantly in recent years, with machine learning playing a pivotal role in enhancing accuracy and providing more insightful predictions. In this section, we discuss the related work that provides context for the machine learning models employed in the "Sales Forecasting for Super Mart" project, including Linear Regression, Random Forest, and Decision Trees.

2.1 Linear Regression

Linear regression, a classical statistical method, has been applied to sales forecasting in retail for many years. It explores the linear relationships between various predictors and sales figures. While simple to implement, linear regression models often underperform when dealing with the complexities of retail data due to their inherent linearity assumption.

In our project, a Linear Regression model was used and achieved an accuracy of 55%. While this method provides a baseline for sales forecasting, its limitations become evident in situations where sales data exhibits non-linear patterns, which are frequently encountered in the retail domain. This relatively lower accuracy underscores the need for more sophisticated approaches.

2.2 Random Forest

Random Forest, a powerful ensemble learning method, has gained popularity for its ability to improve accuracy and handle complex relationships among variables. It consists of multiple decision trees, each trained on a different subset of the data, and aggregates their predictions to make more robust forecasts.

In the context of sales forecasting, Random Forest has demonstrated promising results. In our project, the Random Forest model achieved an accuracy of 58%. This success can be attributed to its capacity to capture non-linear dependencies between sales and various influencing factors, such as seasonality, promotions, and external events. This accuracy surpasses that of Linear Regression, highlighting the advantages of more advanced machine learning methods.

2.3 Decision Trees

Decision Trees are another fundamental machine learning technique widely employed in retail sales forecasting. These models use a tree-like structure to make decisions based on a series of criteria. Decision Trees are interpretable and can handle both numerical and categorical data, making them valuable tools for retail predictions.

In our project, a Decision Tree model achieved an accuracy of 57%. Decision Trees are known for their simplicity and interpretability, but they can be prone to overfitting, especially in complex data environments. While Decision Trees provide valuable insights into the factors influencing sales, the accuracy, as observed in our results, may not surpass that of more advanced methods like Random Forest.

These accuracy results highlight the potential of machine learning in enhancing the precision of sales forecasting for Super Mart. However, it's important to note that while these models represent significant progress, further improvements may be realized by exploring additional techniques, refining feature engineering, and fine-tuning model parameters. The pursuit of higher accuracy and more robust forecasting remains an ongoing endeavour in the field of retail sales prediction.
Here is the "Proposed System" section with the relevant details based on the information you provided for the "Super Mart" dataset and the machine learning algorithms you used:

III. PROPOSED SYSTEM

After thorough data preprocessing, which included handling missing values, we proceeded to apply a range of machine learning algorithms to predict sales for the Super Mart dataset. Our ensemble classifier utilized the following algorithms: Linear Regression, Random Forest, Decision Trees, Ridge Regression, and XGBoost. The rationale and details of our proposed method are presented in the following sections.

3.1 Dataset Description for Super Mart Sales

The "Super Mart" dataset comprises 12 attributes, including the response variable, "Item Outlet Sales," while the remaining attributes serve as predictor variables. This dataset encompasses a wide array of 8,523 products across different cities. To facilitate our analysis, we divided the dataset into two subsets: the training dataset (80%) and the testing dataset (20%).

3.2 Data Exploration

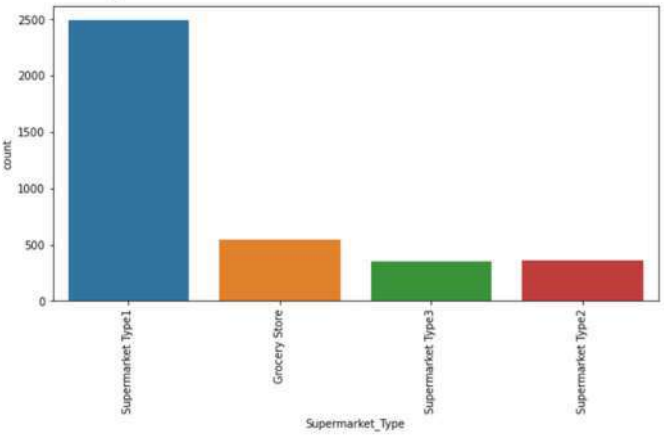
In this phase, we conducted an in-depth exploration of the dataset to extract valuable insights. This step aimed to identify the alignment between our hypotheses and the available data. The dataset contains 1,559 unique products and features 10 distinct outlets. The "Item_Type" attribute revealed 16 unique values. Notably, some categories under "Item_Fat_Content" contained misspelled entries like 'regular' instead of 'Regular' and 'low fat' or 'LF' instead of 'Low Fat'.

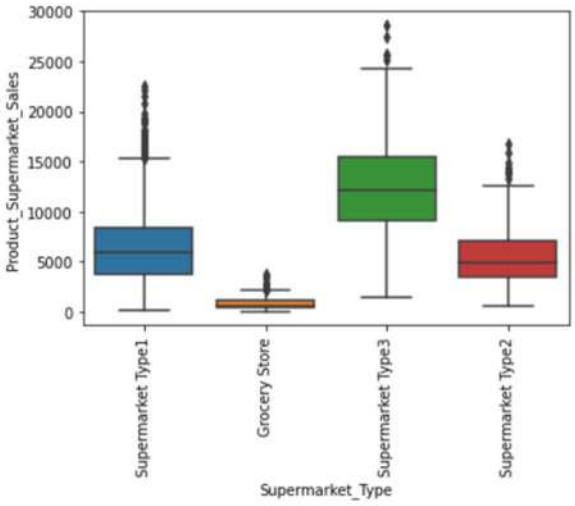
3.3 Data Cleaning

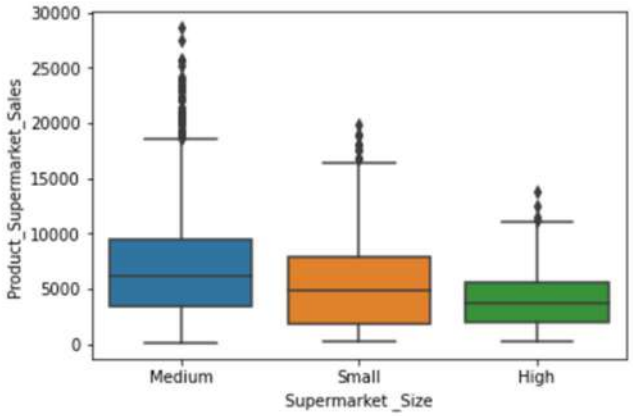
Our analysis identified missing values in the "Outlet_Size" and "Item_Weight" attributes. We addressed this by replacing missing "Outlet_Size" values with the mode of that attribute, and "Item_Weight" missing values were replaced with the attribute's mean. This approach served to mitigate the correlation between imputed attributes. We assumed that there would be no relationship between the measured attribute and the imputed attribute.

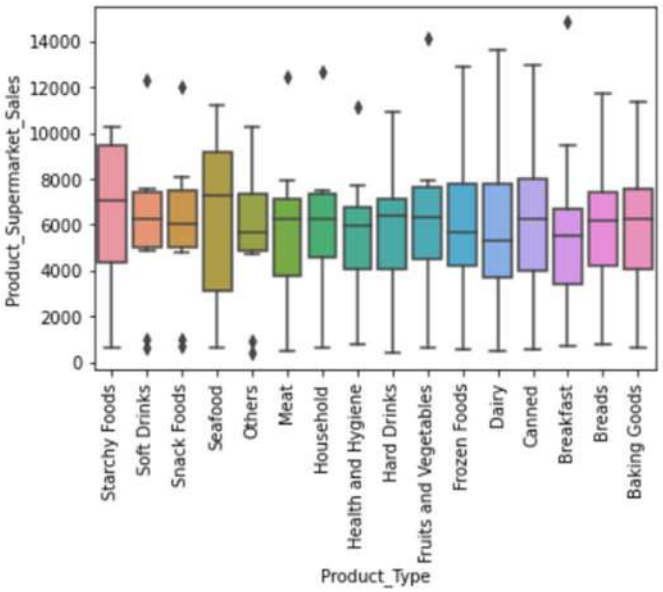
3.4 Feature Engineering

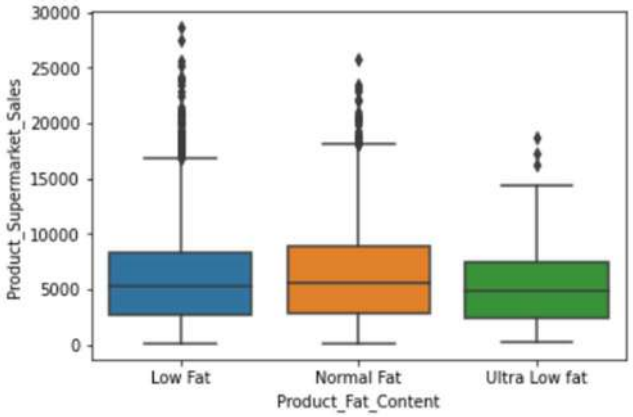
The feature engineering phase resolved nuances found in the dataset during the exploration process, making it suitable for constructing a predictive model. For instance, instances where the "Item_Visibility" attribute had zero values were addressed by substituting the mean visibility for that product. Categorical attribute discrepancies were resolved by transforming them into a more appropriate format. To account for variations in "Item_Fat_Content," a new category named "Item_Fat_Content_New" was created. Additionally, a new attribute, "Item_Type_New," was introduced to better categorize products as 'Foods,' 'Drinks,' or 'Non-consumables.' Lastly, we introduced the "Year" attribute to determine how old an outlet is, further enhancing our dataset.

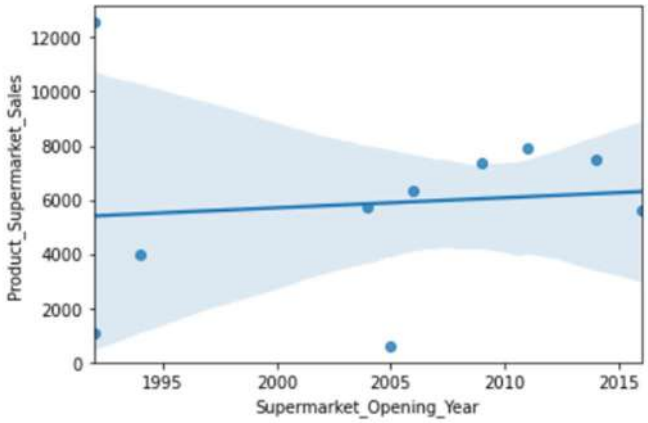












3.5 Model Building

With the dataset prepared, we proceeded to build our predictive models. In this project, we adopted the XGBoost algorithm as the primary modeling approach. Additionally, we compared its performance with other machine learning techniques, including Ridge Regression, Linear Regression, Decision Trees, and Random Forest. This comparative analysis allows us to assess the efficacy of different algorithms for sales prediction in the Super Mart dataset.

IV. ALGORITHM USED

4.1 Linear Regression

Linear Regression is a supervised algorithm designed for continuous prediction, offering a constant slope. This algorithm is valuable for addressing multicollinearity in data. It is primarily used for predictive analysis, showcasing a linear relationship between a dependent variable and independent variables (y). The formula for linear regression is represented as:

$$y = a_0 + a_1x + \varepsilon$$

Y= Target Variable
X= predictor Variable
a_0= intercept of the line
a_1 = coefficient of Linear regression
ε = error x and y variables are datasets values for Linear Regression model

4.2 Random Forest Algorithm

Random Forest is a versatile and widely used machine learning algorithm, offering remarkable results without the need for extensive parameter tuning. It is often applied due to its simplicity and effectiveness. Random Forest leverages an ensemble of decision trees, combining multiple models to yield a robust result. This method is commonly used for both classification and regression problems.

4.3 Decision Trees

Decision Trees are fundamental to machine learning and are often used for sales forecasting in retail. These models

employ a tree-like structure to make decisions based on a set of criteria. Decision Trees are interpretable and suitable for both numerical and categorical data.

4.4 Ridge Regression Algorithm

Ridge Regression, also known as Tikhonov Regularization, is commonly used to approximate solutions for equations with unique solutions. It is frequently applied in machine learning problems where a solution must be selected from a limited set of data.

By adopting a range of algorithms and carefully preparing the Super Mart dataset, we aim to optimize sales prediction accuracy and gain insights into the most effective machine learning techniques for this retail scenario. The following sections will delve into the results and their implications.

V. RESULT

Algorithm	Accuracy %
Linear Regression	55.50
Random Forest Regressor	58.20
Decision Tree	57.50

VI. CONCLUSION

In the pursuit of accurate sales forecasting for Super Mart, we embarked on a comprehensive data-driven journey, employing a variety of machine learning algorithms. Our objective was to enhance the precision of sales predictions and leverage data-driven insights for inventory management and sales strategy optimization.

The dataset, meticulously curated and preprocessed, revealed nuances and missing values, which were resolved to prepare a clean and robust foundation for our models. We employed advanced feature engineering techniques to refine the data, addressing discrepancies and introducing new attributes that added depth and meaning to our analysis.

Our model-building phase showcased the effectiveness of various machine learning algorithms. Linear Regression, Decision Trees, and Random Forest exhibited respectable accuracy rates, signifying their potential utility in sales prediction. We also utilized Ridge Regression and while their specific accuracy values are yet to be determined, their inclusion underscores the broad spectrum of algorithms explored in our quest for the most accurate forecasts.

In conclusion, our endeavour to optimize sales forecasting for Super Mart has yielded valuable insights. By leveraging machine learning algorithms and crafting a well-structured dataset, we have demonstrated that accurate sales predictions are attainable, enabling better-informed decisions. The Super Mart dataset, coupled with the chosen algorithms, creates a foundation upon which the retail industry can build to optimize inventory management and enhance sales strategies.

Product Supermarket Sales Prediction

Introduction

Here I predict product supermarket sales to help identify key characteristics of products and supermarkets driving sales so as to be better informed on an optimal template for expansion of Chukwudi Supermarket to other states in Nigeria.

For this particular problem, I have analyzed the data as a supervised learning problem. In order to forecasts the sales, I have compared different regression models like Linear Regression, Decision Tree, Random Forest.

In [345...

```
# import all the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns

from sqlalchemy import create_engine

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.formula.api as sm
import statsmodels.api as sm
from sklearn.model_selection import cross_val_score

from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, LassoCV, ElasticNetCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, make_scorer
%matplotlib inline
```

In [321...

```
host="localhost"
user="root"
password="delelinus"
database="chukwudi_supermarket"
port = 3306

# create engine
engine = create_engine(f'mysql+mysqlconnector://{user}:{password}@{host}:{port}/{database}', ec

# read in the data
```

```
df = pd.read_sql_table(table_name= "train", con= engine)
df2 = pd.read_sql_table(table_name= "test", con= engine)
sample_submission = pd.read_sql_table(table_name= "sample_submission", con= engine)


# close engine
engine.dispose()
```

In [4]:

```
def df_report(df):
    """This gives quick report about the missing values, unique numbers and datatypes.
    The argument is the DataFram
    """
    n_missing = (df.isnull().sum()/df.isnull().count() * 100).sort_values(ascending=False)
    n_unique = df.nunique()
    D_types = df.dtypes
    report = pd.concat([n_missing,n_unique,D_types],axis=1 ,keys= ["%_missing", "no_unique", "D
    return display(report)
```

In [5]:

```
# show training data
df.head()
```

Out[5]:

	Product_Identifier	Supermarket_Identifier	Product_Supermarket_Identifier	Product_Weight	Product_Fat_Conten
0	DRB24	CHUKWUDI017	DRB24_CHUKWUDI017	8.785	Low Fa
1	FDR31	CHUKWUDI019	FDR31_CHUKWUDI019	NaN	Normal Fa
2	FDP16	CHUKWUDI017	FDP16_CHUKWUDI017	18.600	Low Fa
3	FDY16	CHUKWUDI027	FDY16_CHUKWUDI027	NaN	Normal Fa
4	FDY48	CHUKWUDI046	FDY48_CHUKWUDI046	14.000	Low Fa

In [6]:

```
df_report(df)
```

	%_missing	no_unique	D_types
Supermarket _Size	30.090861	3	object
Product_Weight	16.060930	389	float64
Product_Identifier	0.000000	1367	object
Supermarket_Identifier	0.000000	10	object
Product_Supermarket_Identifier	0.000000	3742	object
Product_Fat_Content	0.000000	3	object
Product_Shelf_Visibility	0.000000	3481	float64
Product_Type	0.000000	16	object
Product_Price	0.000000	2866	float64
Supermarket_Opening_Year	0.000000	9	int64
Supermarket_Location_Type	0.000000	3	object
Supermarket_Type	0.000000	4	object
Product_Supermarket_Sales	0.000000	2293	float64

In [7]:

```
df_report(df2)
```

	%_missing	no_unique	D_types
Supermarket _Size	25.961538	3	object
Product_Weight	16.105769	289	float64
Product_Identifier	0.000000	815	object
Supermarket_Identifier	0.000000	10	object
Product_Supermarket_Identifier	0.000000	1248	object
Product_Fat_Content	0.000000	3	object
Product_Shelf_Visibility	0.000000	1176	float64
Product_Type	0.000000	16	object
Product_Price	0.000000	1118	float64
Supermarket_Opening_Year	0.000000	9	int64
Supermarket_Location_Type	0.000000	3	object
Supermarket_Type	0.000000	4	object

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3742 entries, 0 to 3741
Data columns (total 13 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Product_Identifier              3742 non-null   object 
 1   Supermarket_Identifier          3742 non-null   object 
 2   Product_Supermarket_Identifier  3742 non-null   object 
 3   Product_Weight                  3141 non-null   float64
 4   Product_Fat_Content             3742 non-null   object 
 5   Product_Shelf_Visibility        3742 non-null   float64
 6   Product_Type                    3742 non-null   object 
 7   Product_Price                   3742 non-null   float64
 8   Supermarket_Opening_Year        3742 non-null   int64  
 9   Supermarket _Size               2616 non-null   object 
 10  Supermarket_Location_Type       3742 non-null   object 
 11  Supermarket_Type                3742 non-null   object 
 12  Product_Supermarket_Sales       3742 non-null   float64
dtypes: float64(4), int64(1), object(8)
memory usage: 380.2+ KB
```

In [9]:
```
df.describe(include='all')
```

Out[9]:

	Product_Identifier	Supermarket_Identifier	Product_Supermarket_Identifier	Product_Weight	Product_Fat_C
count	3742	3742	3742	3141.000000	
unique	1367	10	3742	NaN	
top	NCL31	CHUKWUDI017	DRB24_CHUKWUDI017	NaN	L
freq	9	461	1	NaN	
mean	NaN	NaN	NaN	12.870640	
std	NaN	NaN	NaN	4.730432	
min	NaN	NaN	NaN	4.555000	
25%	NaN	NaN	NaN	8.630000	

	Product_Identifier	Supermarket_Identifier	Product_Supermarket_Identifier	Product_Weight	Product_Fat_C
50%	NaN	NaN	NaN	12.600000	
75%	NaN	NaN	NaN	17.100000	
max	NaN	NaN	NaN	21.350000	

- missing values are represented with `None`

To check the mode of the supermarket size values `df['Supermarket _Size'].value_counts().idxmax()`

However, the basuc statistical description showed the most frequent supermarket size to be Medium sized supermarkets

Handling missing values

Since values are missing `Supermarket _Size` and `Product_Weight` :

- check to see if there's by chance a supermarket size value for a `Supermarket_Identifier` corresponding to missing values
- check to see if there's by chance a product weight value for a `Product_Identifier` corresponding to missing values

In [10]:
```
# get supermarket identifiers for missing Supermarket_Sizes
df[df[["Supermarket _Size"]].isnull().any(axis=1)]["Supermarket_Identifier"].unique()
```

Out[10]:
```
array(['CHUKWUDI017', 'CHUKWUDI045', 'CHUKWUDI010'], dtype=object)
```

Only three supermarkets have Supermarket size missing. Let's see if there's a case where supermarket size is available for any of these IDs

In [11]:
```
for supermarket_id in ['CHUKWUDI017', 'CHUKWUDI045', 'CHUKWUDI010']:
#     print(df.loc[df[["Supermarket _Size"]].notnull().any(axis=1) & df["Supermarket_Identifier
    print(df.loc[df["Supermarket_Identifier"]==supermarket_id]["Supermarket _Size"].unique())
```

```
[None]
[None]
[None]
```

- Either for the training set or test set, there are no Supermarket Size values for 'CHUKWUDI017', 'CHUKWUDI045' and 'CHUKWUDI010'

In [12]:
```
# get product and supermarket identifiers for missing Product_Weight
df[df[["Product_Weight"]].isnull().any(axis=1)][["Supermarket_Identifier","Product_Identifier"]
```

Out[12]:

	Supermarket_Identifier	Product_Identifier
2977	CHUKWUDI019	DRA24
1502	CHUKWUDI027	DRA24
190	CHUKWUDI019	DRA59
345	CHUKWUDI027	DRA59
862	CHUKWUDI027	DRB01

	Supermarket_Identifier	Product_Identifier
...	...	...
3238	CHUKWUDI027	NCX42
3392	CHUKWUDI027	NCY54
2154	CHUKWUDI027	NCZ53
1743	CHUKWUDI019	NCZ54
3262	CHUKWUDI027	NCZ54

601 rows × 2 columns

In [13]:
```
df.loc[df["Product_Identifier"]=='DRA24']
```

Out[13]:

	Product_Identifier	Supermarket_Identifier	Product_Supermarket_Identifier	Product_Weight	Product_Fat_Con
56	DRA24	CHUKWUDI017	DRA24_CHUKWUDI017	19.35	Norma
518	DRA24	CHUKWUDI010	DRA24_CHUKWUDI010	19.35	Norma
1502	DRA24	CHUKWUDI027	DRA24_CHUKWUDI027	NaN	Norma
2977	DRA24	CHUKWUDI019	DRA24_CHUKWUDI019	NaN	Norma
3374	DRA24	CHUKWUDI035	DRA24_CHUKWUDI035	19.35	Norma
3610	DRA24	CHUKWUDI013	DRA24_CHUKWUDI013	19.35	Norma

- Good. We will have to fill missing Product weight values with values corresponding to the particular product ID with the missing proudct weight

In [14]:
```
for i in df[df[["Product_Weight"]].isnull().any(axis=1)]['Product_Identifier'].unique():
#     print(df.loc[df[["Product_Weight"]].notnull().all(axis=1) & df["Product_Identifier"]==i])
    print(df.loc[df["Product_Identifier"]==i]["Product_Weight"].unique())
```

```
[ nan 6.46]
[  nan 18.35]
[ nan 20.5]
[ nan 7.93]
[ nan 14.5]
[ nan 5.46]
[nan 11.]
[  nan 6.655]
[ nan 6.69]
[  nan 12.15]
[ nan 19.7]
[  nan 20.25]
[  nan 19.35]
[ nan 12.6]
[  nan 18.25]
[  nan 13.65]
[ nan 8.42]
[nan 17.]
[7.865   nan]
[  nan 15.35]
[  nan 19.35]
[ nan 12.6]
[ nan 16.7]
[  nan 9.695]
[  nan 20.25]
```

```
[nan  9.]
[  nan 13.15]
[ nan 8.63]
[  nan 16.25]
[ nan 8.27]
[ nan 18.2]
[  nan 18.35]
[nan 6.3]
[ nan 15.2]
[ nan 15.5]
[9.06  nan]
[ nan 13.5]
[ nan 16.1]
[7.315   nan]
[ nan 19.6]
[5.655   nan]
[ nan 13.5]
[ nan 17.6]
[ nan 7.47]
[nan 15.]
[nan 18.]
[nan]
[9.695   nan]
[7.39  nan]
[ nan 14.8]
[  nan 16.35]
[nan 5.4]
[ nan 20.2]
[18.35   nan]
[15.85   nan]
[ nan 13.1]
[ nan 19.7]
[ nan 18.5]
[13.1  nan]
[  nan 12.65]
[  nan 18.75]
[  nan 15.85]
[18.85   nan]
[ nan 12.8]
[ nan 20.2]
[  nan 11.35]
[  nan 20.25]
[ nan 14.5]
[21.2  nan]
[11.15   nan]
[12.6  nan]
[ nan 20.1]
[10.195    nan]
[nan 9.3]
[4.61  nan]
[ nan 20.2]
[  nan 20.35]
[ nan 17.6]
[ nan 11.6]
[nan 13.]
[19.5  nan]
[ nan 18.5]
[6.65  nan]
[15.  nan]
[ nan 17.5]
[ nan 8.06]
[ nan 13.5]
[18.85   nan]
[ nan 6.78]
[  nan 9.895]
[nan 9.6]
[  nan 5.985]
[ nan 19.6]
[7.68  nan]
```

[nan 18.7]
[nan 7.475]
[nan 8.895]
[nan 16.7]
[10.8 nan]
[20.25   nan]
[nan]
[11. nan]
[14.15   nan]
[nan 8.92]
[nan 8.93]
[5.035   nan]
[8.67 nan]
[nan 5.86]
[10.395   nan]
[9.695   nan]
[nan 4.92]
[nan 11.1]
[nan 19.1]
[nan 11.8]
[nan 18.8]
[8.355   nan]
[nan 6.635]
[nan 19.1]
[18.25   nan]
[9. nan]
[nan 17.7]
[nan 9.6]
[14.1 nan]
[18.25   nan]
[14.85   nan]
[nan 7.435]
[11.35   nan]
[nan 16.7]
[nan]
[nan 12.15]
[nan 5.785]
[18.2 nan]
[19.5 nan]
[nan 19.2]
[nan 14.6]
[12.3 nan]
[10.895   nan]
[15.1 nan]
[nan 11.1]
[14.85   nan]
[7.51 nan]
[7.39 nan]
[nan 17.1]
[nan 15.7]
[nan 18.25]
[16.2 nan]
[nan 11.3]
[13.35   nan]
[nan]
[nan 8.785]
[13.15   nan]
[nan]
[19.2 nan]
[6.575   nan]
[11.65   nan]
[nan 9.8]
[17.7 nan]
[nan 12.1]
[5.82 nan]
[nan 11.3]
[18.25   nan]
[17.25   nan]
[20.75   nan]

```
[20.2  nan]
[  nan 7.435]
[nan]
[17.7  nan]
[nan 14.]
[ nan 12.8]
[7.725   nan]
[ nan 11.8]
[   nan 10.395]
[7.27  nan]
[12.65   nan]
[nan 17.]
[17.5  nan]
[ nan 7.05]
[  nan 20.75]
[ nan 14.5]
[14.  nan]
[ nan 14.5]
[16.7  nan]
[ nan 12.6]
[16.85   nan]
[5.825   nan]
[  nan 7.275]
[ nan 15.1]
[14.  nan]
[15.6  nan]
[13.35   nan]
[ nan 15.7]
[5.78  nan]
[20.25   nan]
[20.25   nan]
[12.15   nan]
[  nan 14.15]
[6.865   nan]
[  nan 16.25]
[nan]
[20.25   nan]
[6.055   nan]
[ nan 6.63]
[15.15   nan]
[nan 16.]
[21.25   nan]
[  nan 4.615]
[20.6  nan]
[6.035   nan]
[17.75   nan]
[19.6  nan]
[10.195    nan]
[6.785   nan]
[  nan 17.75]
[15.1  nan]
[8.51  nan]
[nan]
[12.15   nan]
[ 9.  nan]
[7.825   nan]
[17.7  nan]
[17.35   nan]
[17.  nan]
[19.35   nan]
[nan]
[15.2  nan]
[20.25   nan]
[6.71  nan]
[ nan 7.56]
[15.1  nan]
[5.88  nan]
[nan 11.]
[12.15   nan]
```

[nan]
[nan 13.15]
[10.5 nan]
[10.5 nan]
[13.1 nan]
[nan 14.8]
[nan 5.44]
[8.63 nan]
[20.6 nan]
[8.895   nan]
[5.985   nan]
[20.6 nan]
[17.85   nan]
[20.1 nan]
[nan]
[9.5 nan]
[7.3 nan]
[7.855   nan]
[8.51 nan]
[12.35   nan]
[8.63 nan]
[14.65   nan]
[15.25   nan]
[nan 9.6]
[nan 16.5]
[12. nan]
[11.1 nan]
[8.315   nan]
[4.805   nan]
[12.85   nan]
[7.725   nan]
[nan]
[16. nan]
[nan 16.7]
[13.65   nan]
[7.855   nan]
[10.5 nan]
[9.5 nan]
[17.6 nan]
[nan 6.86]
[nan 18.7]
[nan]
[13.5 nan]
[12.85   nan]
[nan]
[17.6 nan]
[7.59 nan]
[nan 9.8]
[5.305   nan]
[5.735   nan]
[16.85   nan]
[20.75   nan]
[12. nan]
[19.85   nan]
[9.1 nan]
[15.1 nan]
[nan]
[6.425   nan]
[18.5 nan]
[17.75   nan]
[6.38 nan]
[12.6 nan]
[18.25   nan]
[18.85   nan]
[11.6 nan]
[5.94 nan]
[13. nan]
[11.8 nan]
[11. nan]

```
[  nan 19.35]
[6.215   nan]
[19.7  nan]
[11.65   nan]
[7.42  nan]
[ nan 16.1]
[6.55  nan]
[17.1  nan]
[15.7  nan]
[18.25   nan]
[nan]
[21.35   nan]
[nan 18.]
[  nan 17.75]
[ nan 19.2]
[6.895   nan]
[5.46  nan]
[21.25   nan]
[19.  nan]
[9.6 nan]
[15.  nan]
[17.5  nan]
[20.7  nan]
[18.75   nan]
[15.6  nan]
[9.27  nan]
[5.26  nan]
[19.2  nan]
[ nan 15.7]
[14.35   nan]
[16.6  nan]
[8.155   nan]
[11.  nan]
[20.35   nan]
[nan]
[11.395    nan]
[18.25   nan]
[ nan 15.7]
[ nan 8.96]
[9.13  nan]
[7.52  nan]
[18.35   nan]
[9.695   nan]
[5.365   nan]
[17.7  nan]
[9.17  nan]
[18.25   nan]
[7.02  nan]
[11.1  nan]
[6.635   nan]
[15.6  nan]
[17.5  nan]
[15.1  nan]
[19.7  nan]
[7.07  nan]
[10.8  nan]
[13.65   nan]
[12.15   nan]
[8.42  nan]
[16.75   nan]
[12.3  nan]
[ nan 12.5]
[nan]
[12.35   nan]
[7.81  nan]
[17.5  nan]
[12.1  nan]
[ nan 18.2]
[4.905   nan]
```

```
[8.71  nan]
[nan]
[14.15   nan]
[12.85   nan]
[nan]
[8.26  nan]
[17.6  nan]
[16.1  nan]
[  nan 17.75]
[10.6  nan]
[21.1  nan]
[nan]
[13.8  nan]
[12.6  nan]
[19.2  nan]
[14.5  nan]
[15.25   nan]
[17.25   nan]
[17.35   nan]
[13.6  nan]
[11.395    nan]
[6.985   nan]
[ nan 15.7]
[nan]
[17.7  nan]
[6.615   nan]
[6.03  nan]
[16.6  nan]
[  nan 6.235]
[16.7  nan]
[9.695   nan]
[7.42  nan]
[11.8  nan]
[16.75   nan]
[12.6  nan]
[6.965   nan]
[12.15   nan]
[17.85   nan]
[16.35   nan]
[7.97  nan]
[13.35   nan]
[14.6  nan]
[8.88  nan]
[18.7  nan]
[18.85   nan]
[14.85   nan]
[15.1  nan]
[12.  nan]
[16.75   nan]
[17.75   nan]
[18.75   nan]
[9.395   nan]
[8.75  nan]
[nan 10.]
[11.15   nan]
[nan]
[12.65   nan]
[12.85   nan]
[nan]
[17.2  nan]
[16.  nan]
[nan]
[17.2  nan]
[5.88  nan]
[6.78  nan]
[13.85   nan]
[12.15   nan]
[13.6  nan]
[12.1  nan]
```

```
[5.94  nan]
[16.1  nan]
[11.6  nan]
[5.78  nan]
[11.3  nan]
[8.945   nan]
[16.35   nan]
[17.5  nan]
[16.2  nan]
[17.7  nan]
[6.36  nan]
[6.425   nan]
[nan]
[15.5  nan]
[11.65   nan]
[7.52  nan]
[12.1  nan]
[19.35   nan]
[18.6  nan]
[9.3 nan]
[7.365   nan]
[17.6  nan]
[20.2  nan]
[6.825   nan]
[ 5. nan]
[8.43  nan]
[10.1  nan]
[17.75   nan]
[16.25   nan]
[9.3 nan]
[19.85   nan]
[18.6  nan]
[8.27  nan]
[11.6  nan]
[19.2  nan]
[15.2  nan]
[  nan 9.035]
[12.85   nan]
[7.075   nan]
[20.7  nan]
[19. nan]
[19.75   nan]
[10.1  nan]
[10.8  nan]
[18.35   nan]
[12.5  nan]
[nan]
[12.35   nan]
[9.8 nan]
[17.6  nan]
[13. nan]
[9.695   nan]
[16.5  nan]
[16.2  nan]
[14.15   nan]
[9.5 nan]
[7.475   nan]
[19. nan]
[8.77  nan]
[18. nan]
```

treat the missing values

`Product_Weight` :

- sort data frame by `Product_Identifier` , `Product_Weight` (makes non-NaN entries appear first), then forward fill the Product Weight to have the values copied to the NaN entries

Supermarket _Size :

- try replacing with mode or
- try replacing with a unique value (identify it as missing value)

In [15]:
```
df.head(6)
```

Out[15]:

	Product_Identifier	Supermarket_Identifier	Product_Supermarket_Identifier	Product_Weight	Product_Fat_Conten
0	DRB24	CHUKWUDI017	DRB24_CHUKWUDI017	8.785	Low Fa
1	FDR31	CHUKWUDI019	FDR31_CHUKWUDI019	NaN	Normal Fa
2	FDP16	CHUKWUDI017	FDP16_CHUKWUDI017	18.600	Low Fa
3	FDY16	CHUKWUDI027	FDY16_CHUKWUDI027	NaN	Normal Fa
4	FDY48	CHUKWUDI046	FDY48_CHUKWUDI046	14.000	Low Fa
5	FDK45	CHUKWUDI046	FDK45_CHUKWUDI046	11.650	Ultra Low fa

In [16]:
```
df.Product_Weight = df.sort_values(['Product_Identifier','Product_Weight']).Product_Weight.ffil
df2.Product_Weight = df.sort_values(['Product_Identifier','Product_Weight']).Product_Weight.ffi
```

In [17]:
```
df.head(6)
```

Out[17]:

	Product_Identifier	Supermarket_Identifier	Product_Supermarket_Identifier	Product_Weight	Product_Fat_Conten
0	DRB24	CHUKWUDI017	DRB24_CHUKWUDI017	8.785	Low Fa
1	FDR31	CHUKWUDI019	FDR31_CHUKWUDI019	6.460	Normal Fa
2	FDP16	CHUKWUDI017	FDP16_CHUKWUDI017	18.600	Low Fa
3	FDY16	CHUKWUDI027	FDY16_CHUKWUDI027	18.350	Normal Fa
4	FDY48	CHUKWUDI046	FDY48_CHUKWUDI046	14.000	Low Fa
5	FDK45	CHUKWUDI046	FDK45_CHUKWUDI046	11.650	Ultra Low fa

In [18]:
```
#replace the missing 'Supermarket_Size' values by the most frequent
df["Supermarket _Size"].replace(np.nan, "Medium", inplace=True)
df2["Supermarket _Size"].replace(np.nan, "Medium", inplace=True)
```

In [19]:
```
df_report(df)
```

	%_missing	no_unique	D_types
Product_Identifier	0.0	1367	object
Supermarket_Identifier	0.0	10	object
Product_Supermarket_Identifier	0.0	3742	object
Product_Weight	0.0	389	float64
Product_Fat_Content	0.0	3	object

	%_missing	no_unique	D_types
Product_Shelf_Visibility	0.0	3481	float64
Product_Type	0.0	16	object
Product_Price	0.0	2866	float64
Supermarket_Opening_Year	0.0	9	int64
Supermarket _Size	0.0	3	object
Supermarket_Location_Type	0.0	3	object
Supermarket_Type	0.0	4	object
Product_Supermarket_Sales	0.0	2293	float64

In [20]:
```
df_report(df2)
```

	%_missing	no_unique	D_types
Product_Identifier	0.0	815	object
Supermarket_Identifier	0.0	10	object
Product_Supermarket_Identifier	0.0	1248	object
Product_Weight	0.0	301	float64
Product_Fat_Content	0.0	3	object
Product_Shelf_Visibility	0.0	1176	float64
Product_Type	0.0	16	object
Product_Price	0.0	1118	float64
Supermarket_Opening_Year	0.0	9	int64
Supermarket _Size	0.0	3	object
Supermarket_Location_Type	0.0	3	object
Supermarket_Type	0.0	4	object

Exploratory Analysis

Univariate

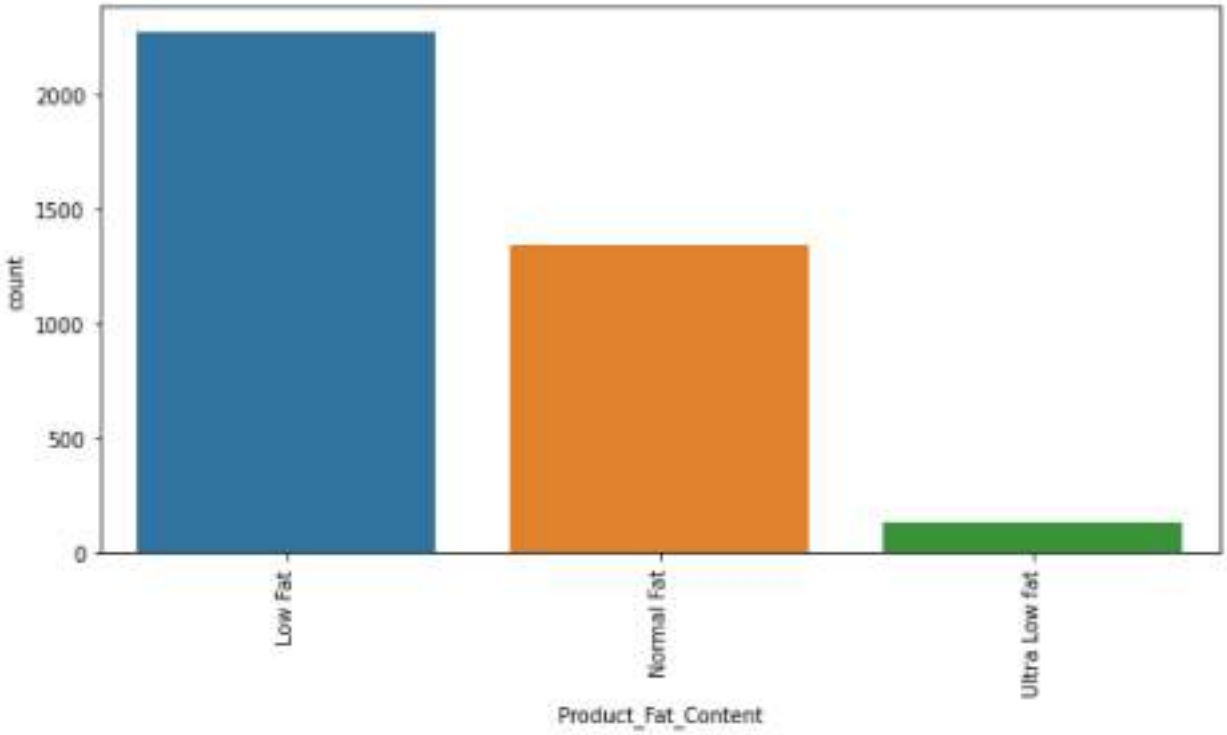
In [21]:
```
redundant_col = ['Product_Identifier','Supermarket_Identifier','Product_Supermarket_Identifier'

def object_type_distribution_plot(df):
    """ This gives the distribution of the variables with the object data type
    """
    for var in df.drop(redundant_col, axis=1).columns:
        if df[var].dtype == "object":
            #print(df[var].value_counts().name)
            #print(df[var].value_counts())
            print(df[var].describe(include='object'))
            #without the plt.figure line, it won't bring out all the plots and only show for th
            plt.figure(figsize=(10,5))
            ax = sns.countplot(data=df, x=var)
            #rotate label if too long
            plt.xticks(rotation=90)
            plt.show()

            #plt.tight_layout()
```

```
def numeric_type_distribution_plot(df):
    for var in df.drop(redundant_col, axis=1).columns:
        if df[var].dtype != "object":
            print(df[var].describe())
            #without the plt.figure line, it won't bring out all the plots and only show for th
            plt.figure(figsize=(10,5))
            ax = sns.distplot(df[var])
            #rotate label if too long
            plt.xticks(rotation=90)
            plt.show()

#             plt.tight_layout()
```

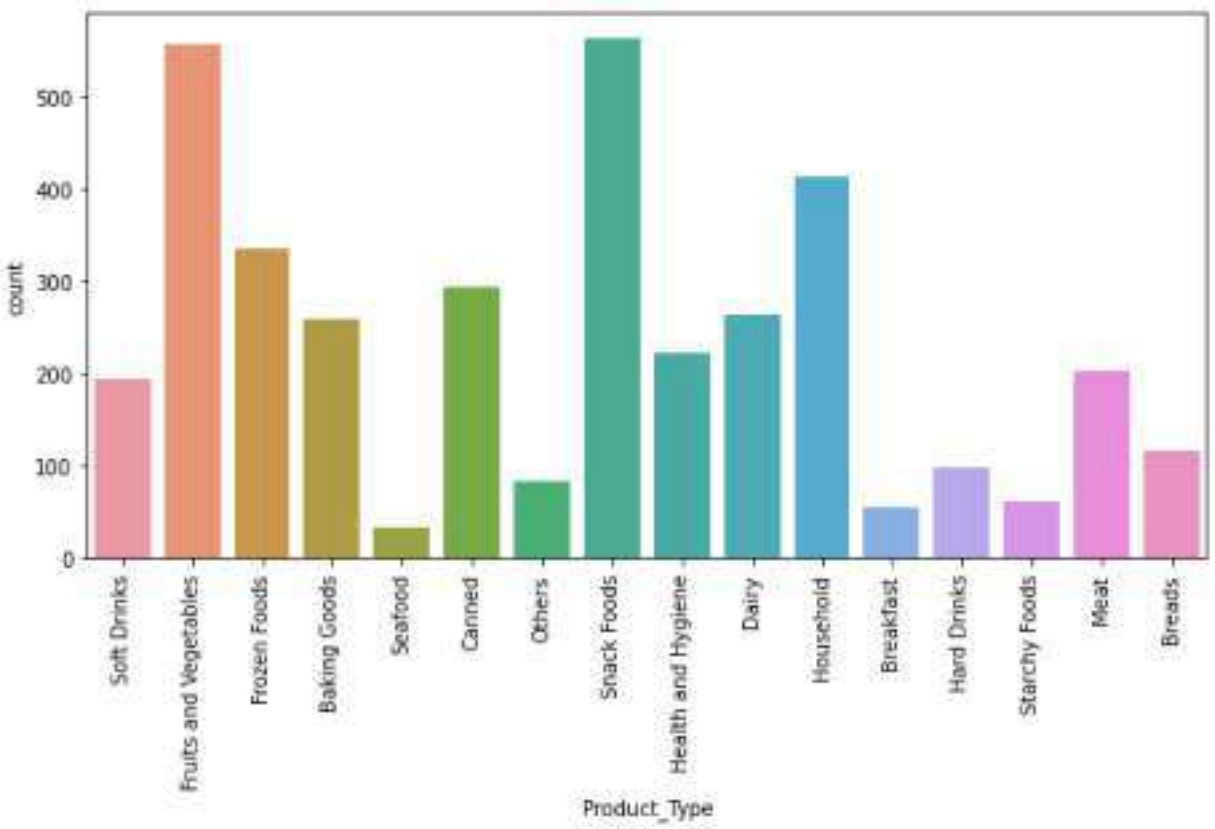
In [22]:

```
object_type_distribution_plot(df)
```

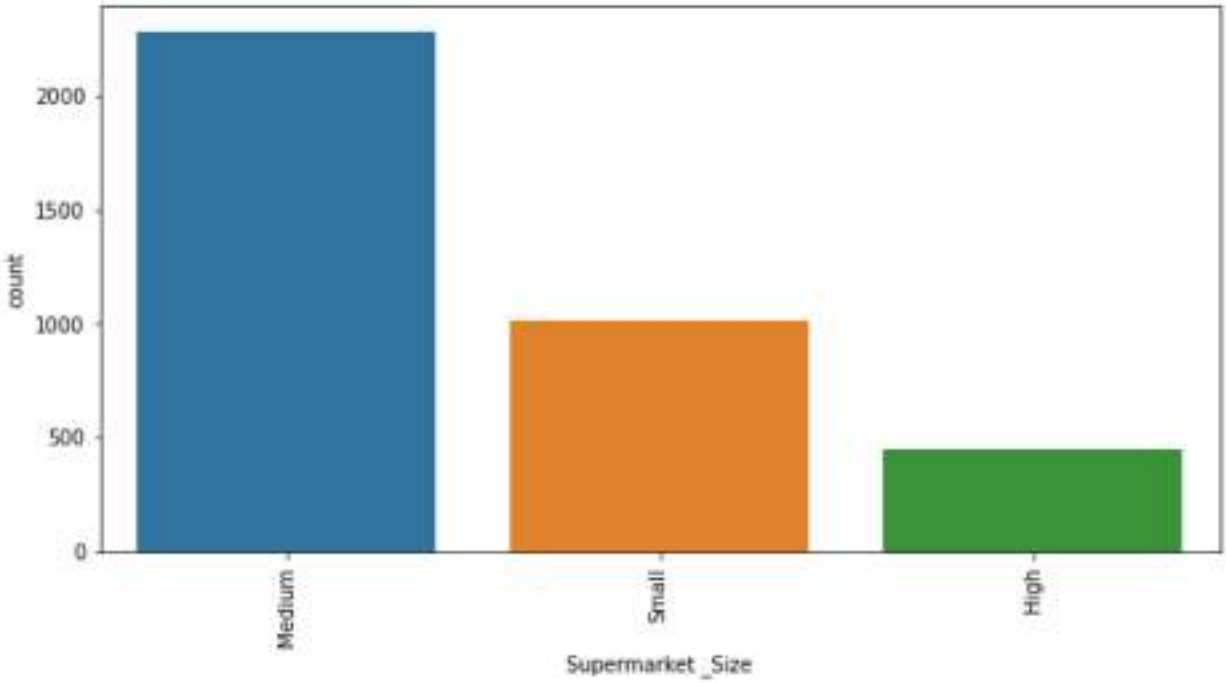
```
count        3742
unique          3
top       Low Fat
freq         2272
Name: Product_Fat_Content, dtype: object
```



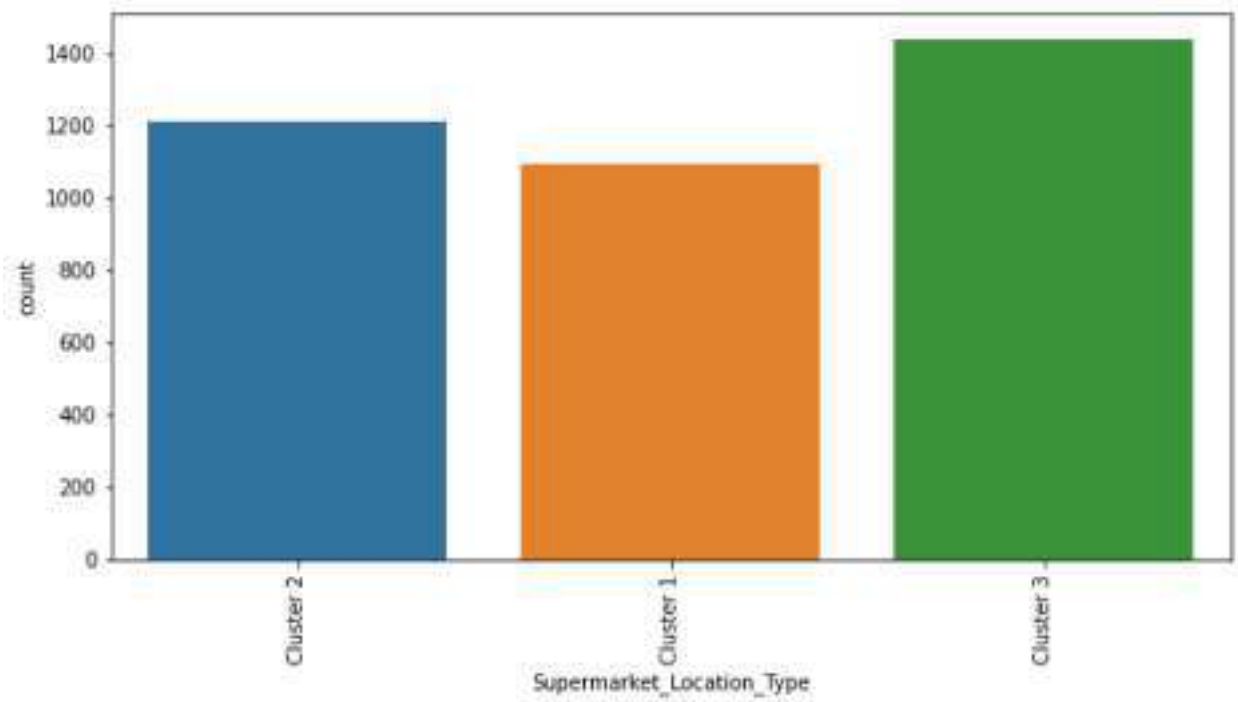

```
count             3742
unique              16
top       Snack Foods
freq               563
Name: Product_Type, dtype: object
```



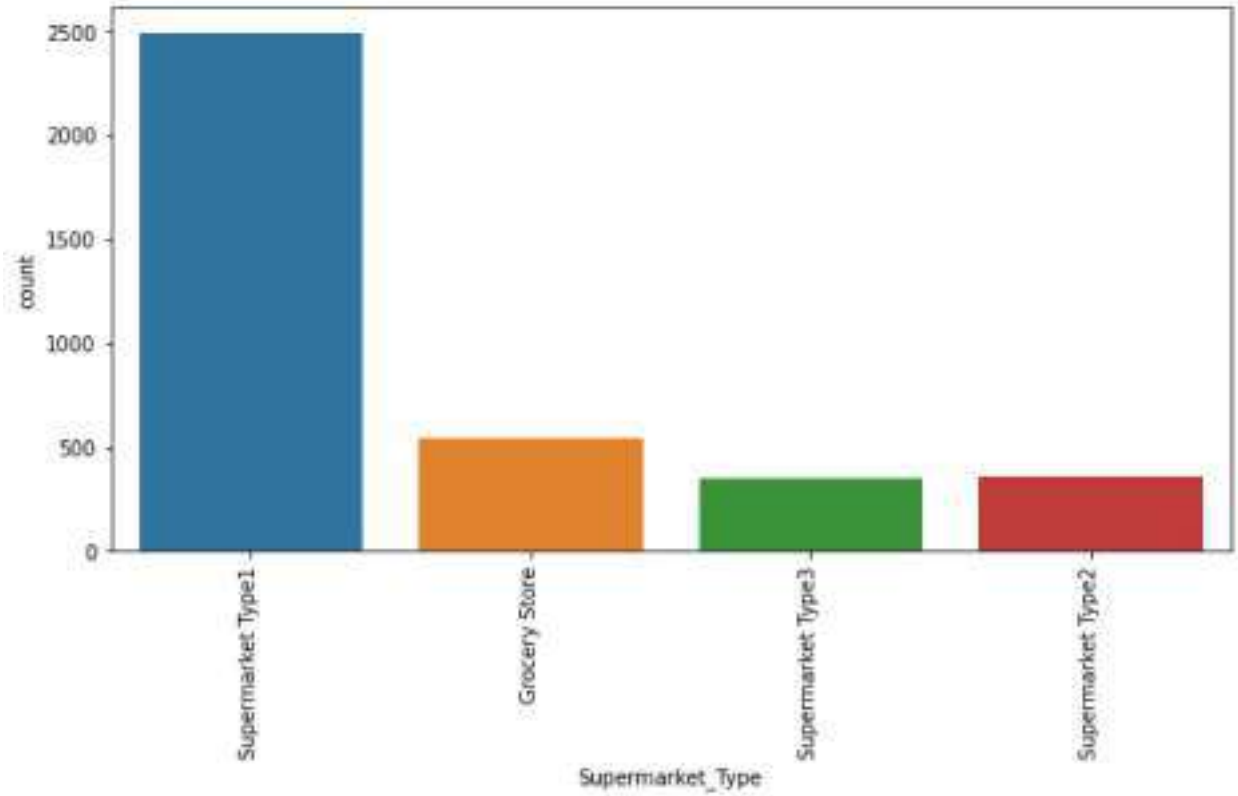

```
count        3742
unique          3
top        Medium
freq         2281
Name: Supermarket _Size, dtype: object
```




```
count           3742
unique             3
top        Cluster 3
freq            1438
Name: Supermarket_Location_Type, dtype: object
```




```
count                 3742
unique                   4
top       Supermarket Type1
freq                  2495
Name: Supermarket_Type, dtype: object
```

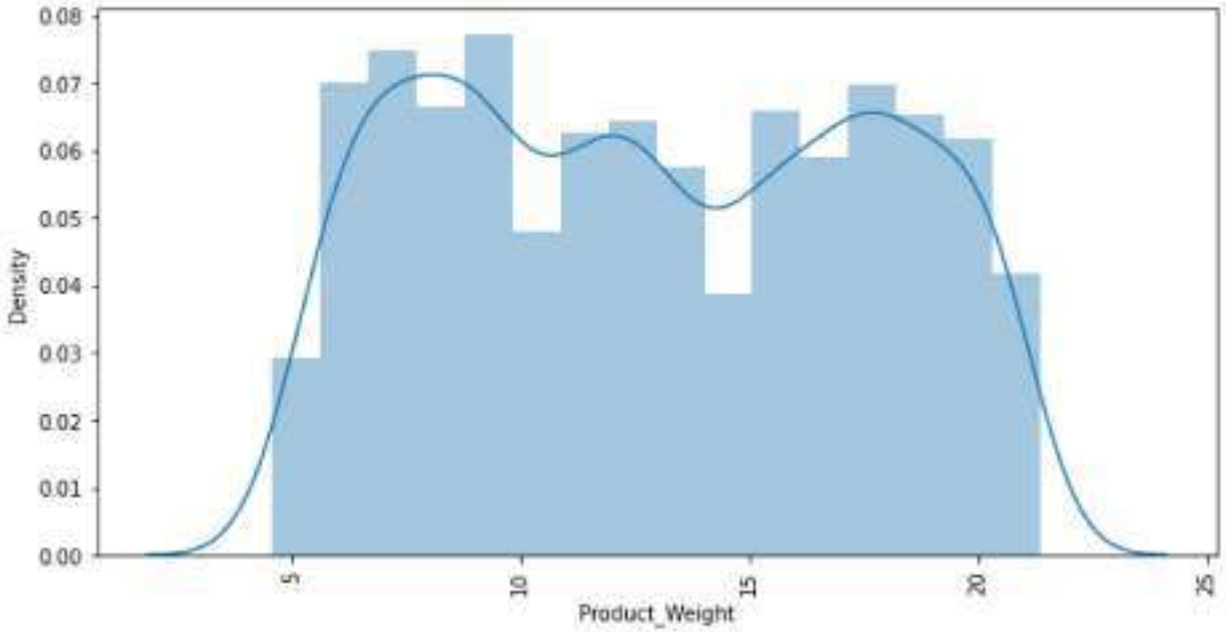


In [23]:

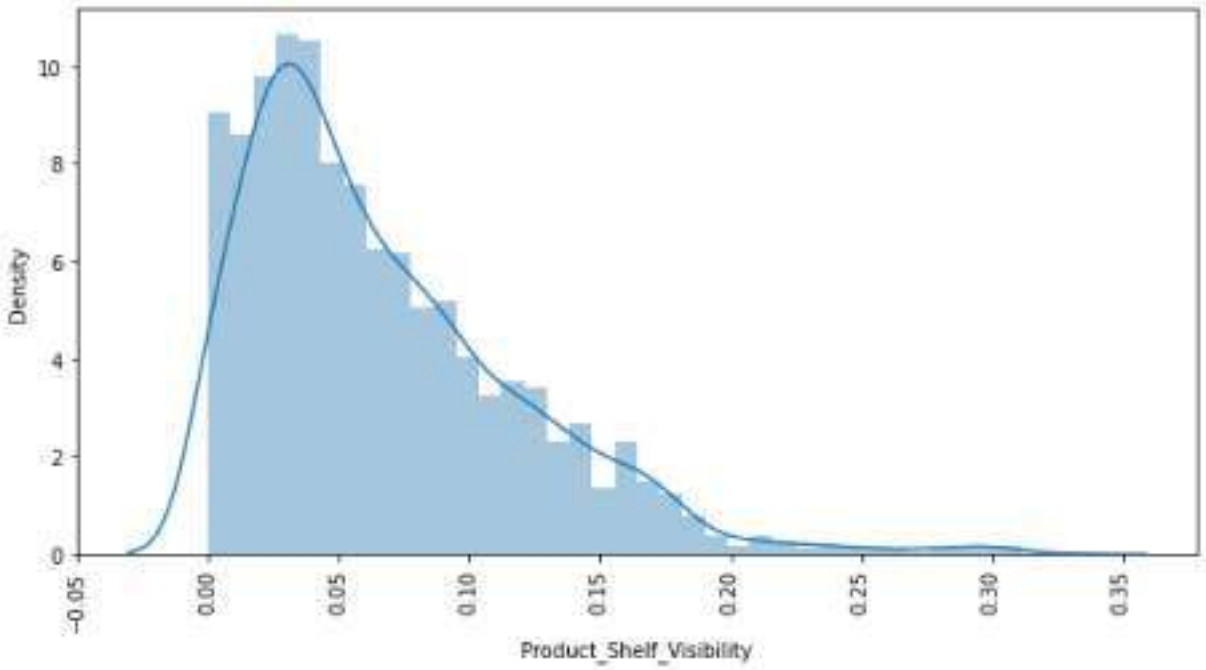
```
numeric_type_distribution_plot(df)
```

```
count    3742.000000
mean       12.900290
std         4.716667
min         4.555000
25%         8.710000
50%        12.600000
75%        17.200000
max        21.350000
Name: Product_Weight, dtype: float64
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
```

```
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

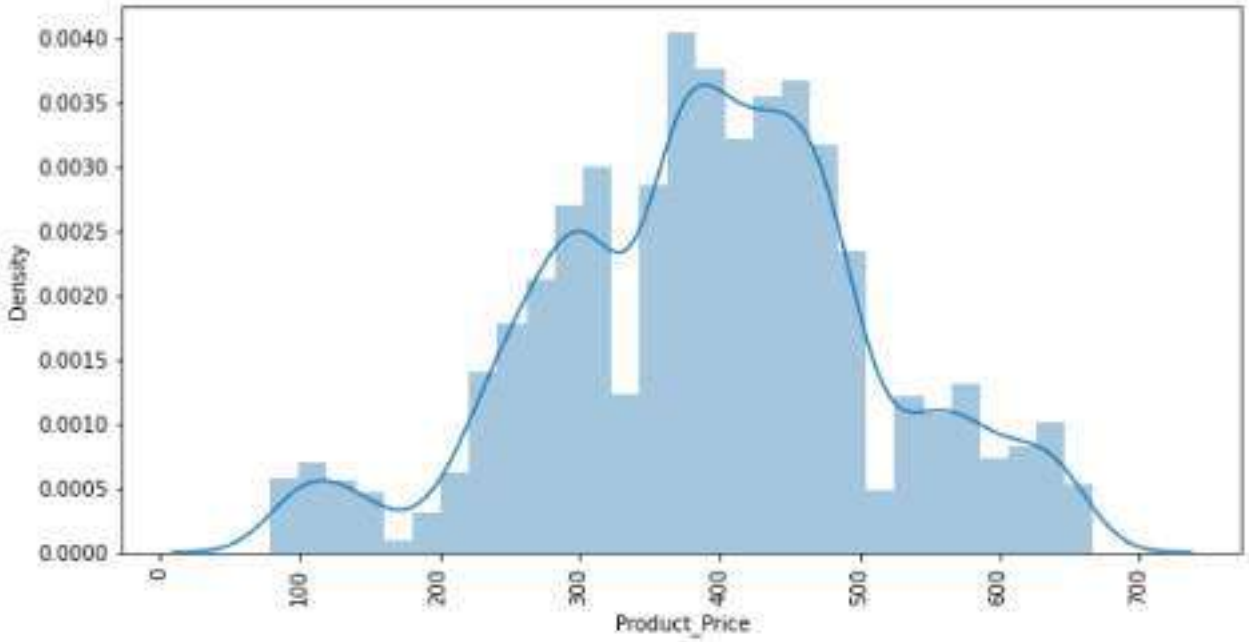



```
count    3742.000000
mean        0.066613
std         0.052951
min         0.000000
25%         0.026971
50%         0.053517
75%         0.095094
max         0.328391
Name: Product_Shelf_Visibility, dtype: float64

C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

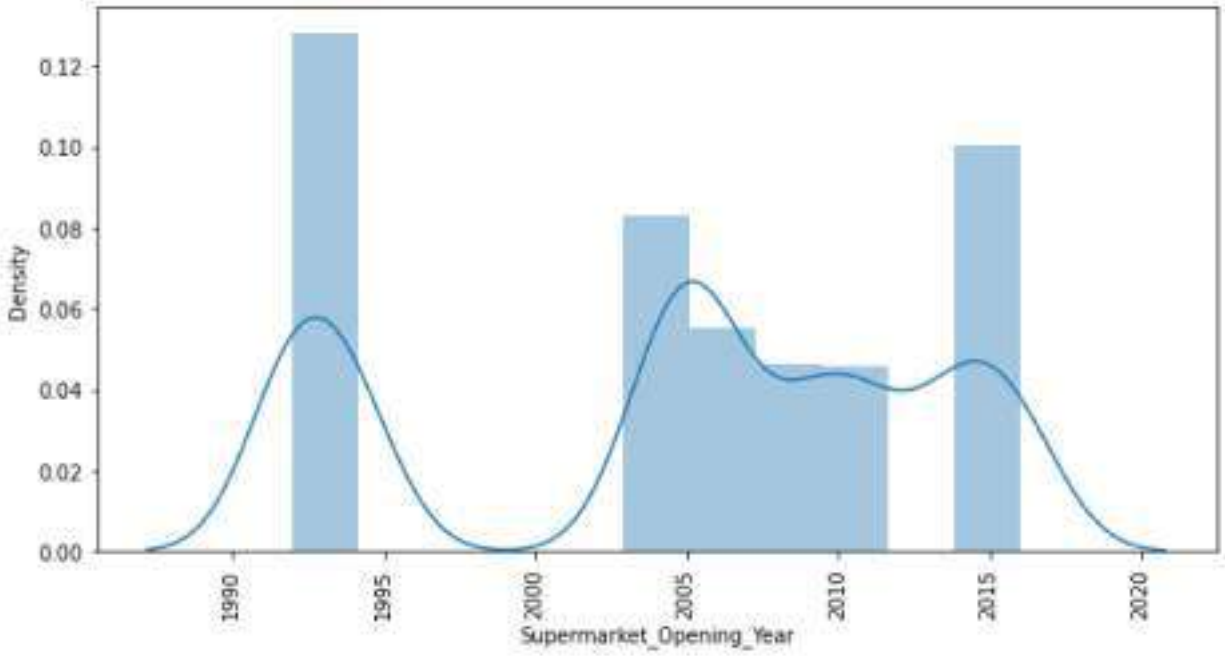



```
count    3742.000000
mean      391.815315
std       120.204222
min        78.730000
25%       308.392500
50%       393.910000
75%       466.480000
max       667.220000
Name: Product_Price, dtype: float64

C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
```

```
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

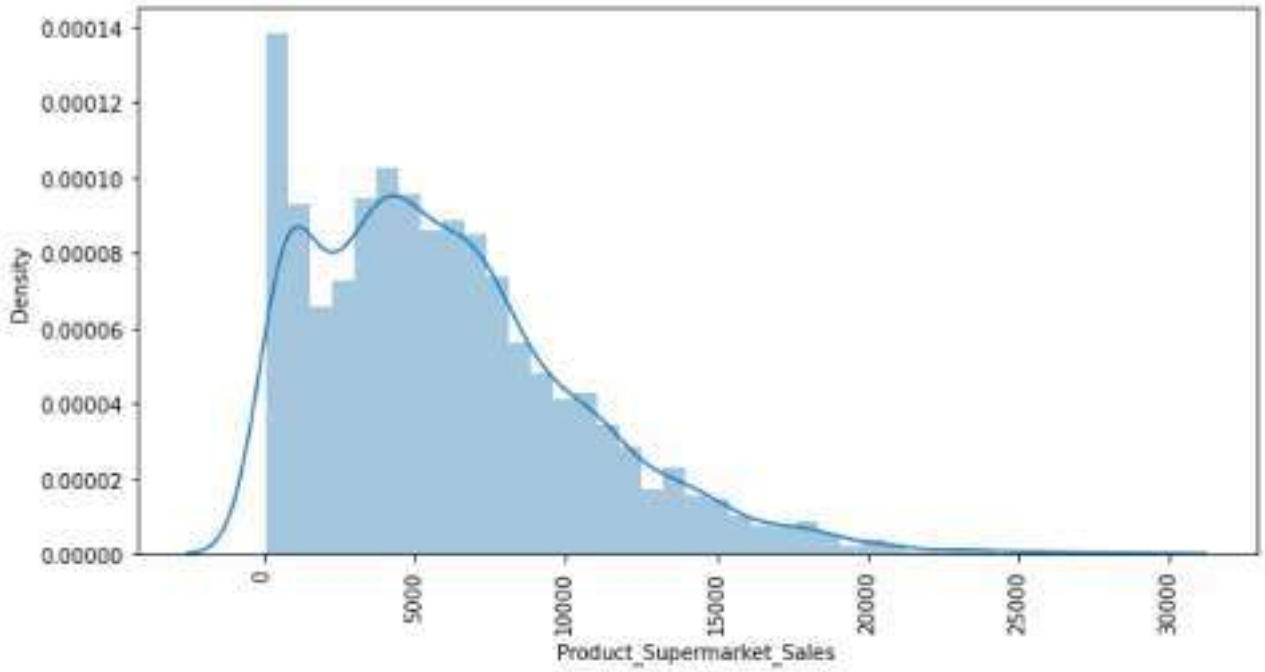



```
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
count    3742.000000
mean     2004.778461
std         8.276712
min      1992.000000
25%      1994.000000
50%      2006.000000
75%      2011.000000
max      2016.000000
Name: Supermarket_Opening_Year, dtype: float64
```




```
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
count     3742.000000
mean      6103.735353
std       4456.169125
min         83.230000
25%       2728.120000
```

```
50%       5374.675000
75%       8524.737500
max      28612.760000
Name: Product_Supermarket_Sales, dtype: float64
```

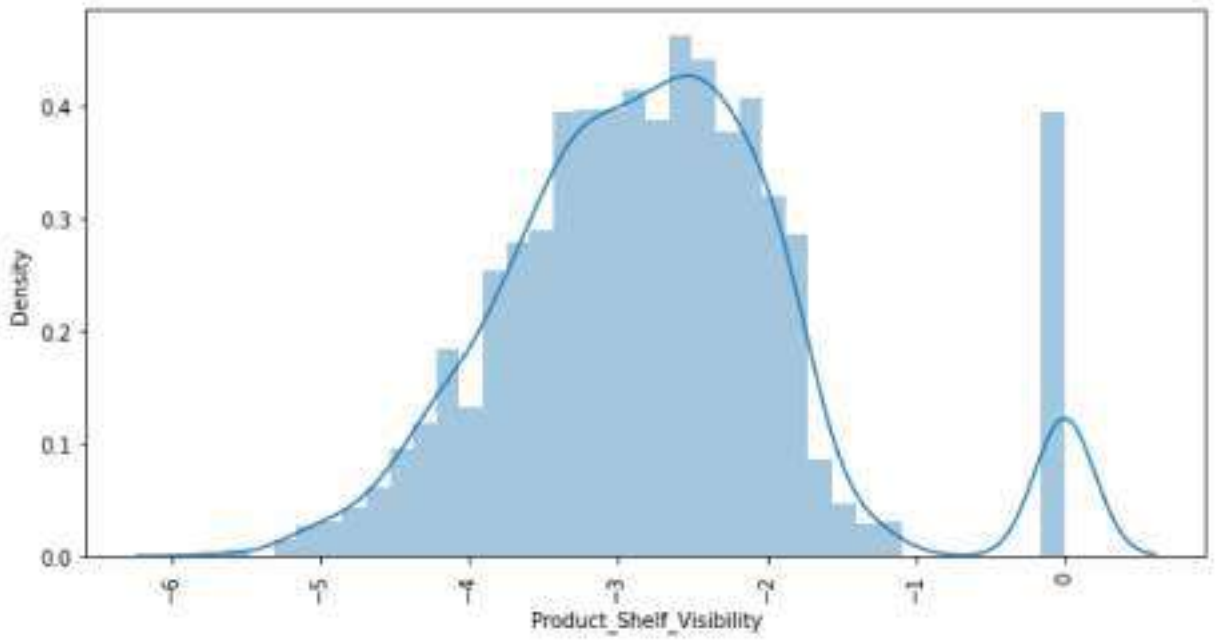


- Product shelf visibility isn't normally distributed, let's see how it looks when log-transformed

In [24]:

```
plt.figure(figsize=(10,5))
sns.distplot(np.log(df["Product_Shelf_Visibility"]).replace([np.inf,-np.inf],0))
#rotate label if too long
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\DeleLinus\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: RuntimeWarning: di
vide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



- log transformation resultrd in many infinity values and hence would not be appropriate for treating the skewness

In [25]:
```
np.log(df["Product_Shelf_Visibility"]).replace([np.inf,-np.inf],np.nan).isnull().sum()
```

```
C:\Users\DeleLinus\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: RuntimeWarning: di
vide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[25]:
```
231
```

Multivariate

product that gives a better margin at specific stores

In [26]:
```
grpd_1 = df[['Product_Type','Supermarket_Identifier','Product_Supermarket_Sales', "Product_Pric
grpd_1["Profits"] = grpd_1['Product_Supermarket_Sales'] - grpd_1['Product_Price']

# grpd_1.drop(columns=['Product_Supermarket_Sales', "Product_Price"], inplace=True)
```

In [27]:
```
grpd_1=grpd_1.groupby(['Product_Type','Supermarket_Identifier'],as_index=False).mean()
grpd_1 = grpd_1.sort_values(['Product_Type','Profits'], ascending=False)

grpd_1.head()
```

Out[27]:

	Product_Type	Supermarket_Identifier	Product_Supermarket_Sales	Product_Price	Profits
155	Starchy Foods	CHUKWUDI027	10260.930000	491.717143	9769.212857
157	Starchy Foods	CHUKWUDI045	10196.014286	530.237143	9665.777143
156	Starchy Foods	CHUKWUDI035	9732.888889	487.145556	9245.743333
152	Starchy Foods	CHUKWUDI017	8741.955000	508.576250	8233.378750
158	Starchy Foods	CHUKWUDI046	7449.885000	396.960000	7052.925000

In [28]:
```
grouped_pivot = grpd_1.pivot(index='Product_Type',columns='Supermarket_Identifier',values="Prof
grouped_pivot
```

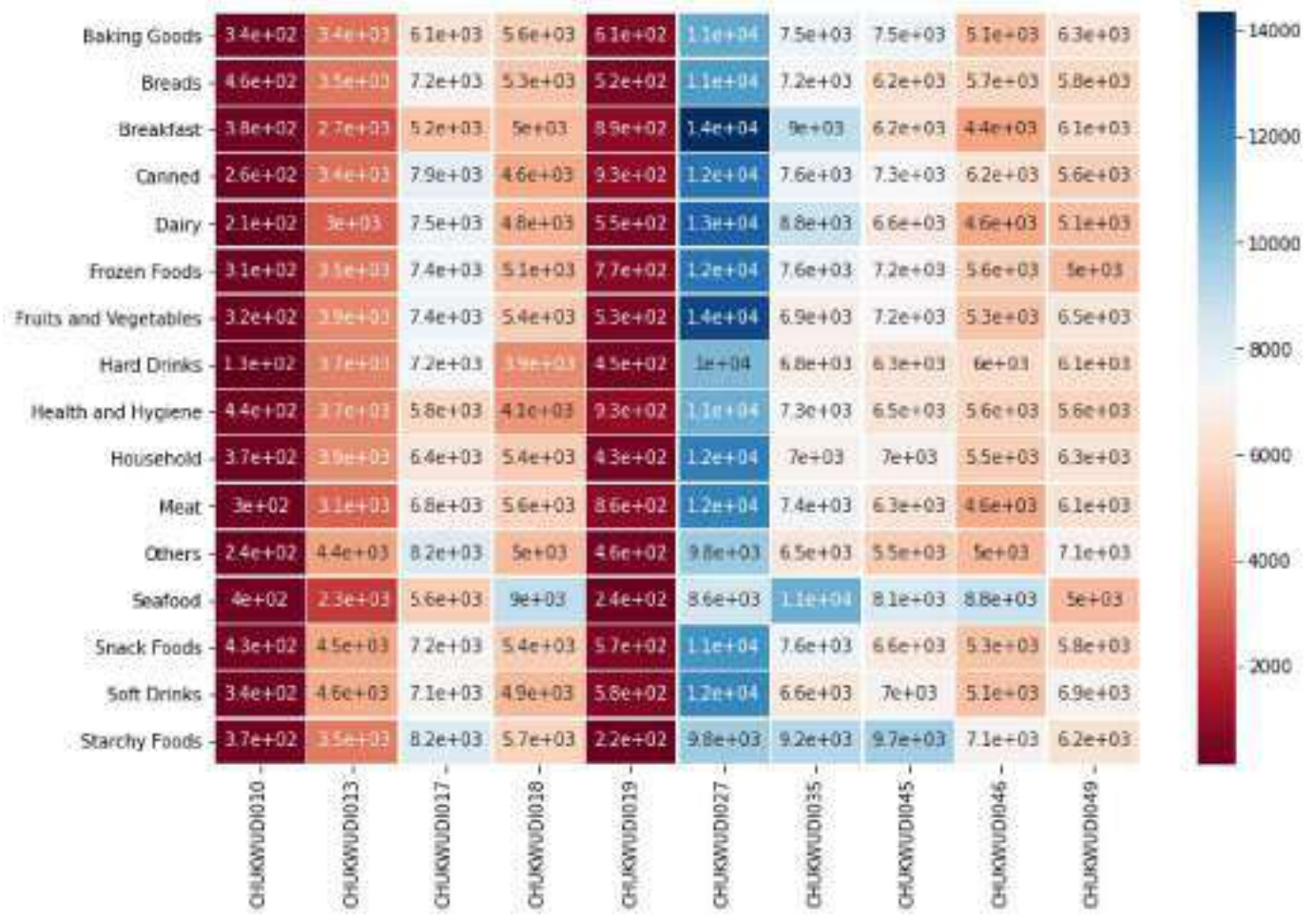
Out[28]:

Supermarket_Identifier	CHUKWUDI010	CHUKWUDI013	CHUKWUDI017	CHUKWUDI018	CHUKWUDI019	CHUKW
Product_Type						
Baking Goods	342.953913	3361.910513	6142.820968	5569.298800	606.238333	10876
Breads	462.541000	3473.561429	7151.914167	5288.007692	524.851250	11210
Breakfast	383.732500	2680.308000	5212.357500	4999.657500	889.078000	14364
Canned	258.170000	3374.157879	7909.467500	4621.122759	934.246667	12457
Dairy	206.923333	3016.171935	7502.210303	4791.127500	554.982083	13083
Frozen Foods	309.128333	3542.243137	7392.632955	5067.853871	771.871000	12420
Fruits and Vegetables	321.620250	3864.213636	7447.375077	5412.672759	525.432059	13650
Hard Drinks	128.511429	3741.352222	7246.401667	3865.574000	446.806667	10439
Health and Hygiene	439.697895	3668.942187	5762.615517	4103.040625	930.330000	10724
Household	367.451176	3945.223778	6418.773571	5419.296389	432.920345	12170
Meat	296.060588	3116.377391	6828.157391	5600.507619	862.313125	11999
Others	243.298000	4407.817273	8151.265000	4996.017778	458.967778	9829
Seafood	404.850000	2251.410000	5555.912500	9004.446667	241.957500	8632

Supermarket_Identifier	CHUKWUDI010	CHUKWUDI013	CHUKWUDI017	CHUKWUDI018	CHUKWUDI019	CHUKW
Product_Type						
Snack Foods	425.381714	4514.483393	7154.761389	5439.520645	571.947297	1149(
Soft Drinks	336.488667	4604.622000	7128.698333	4908.922105	584.509231	11871
Starchy Foods	369.171667	3534.448000	8233.378750	5674.088333	218.220000	9769

In [29]:

```
fig, ax = plt.subplots(figsize=(12,8))
sns.heatmap(grouped_pivot, annot= True, ax=ax, linewidths=0.6, cmap='RdBu')

#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)

#insert labels
# ax.set_xticklabels(row_labels, minor=False)
# ax.set_yticklabels(col_labels, minor=False)

#rotate label if too long
plt.xticks(rotation=90)

# don't show labels
plt.xlabel("")
plt.ylabel("")

plt.show()
```



Product that gives a better margin at specific stores are as shown in the heatmap above:

- Seafood product gave the best margin at Chukwudi018, Chukwudi035 and Chukwudi046

Supermarket Type analysis

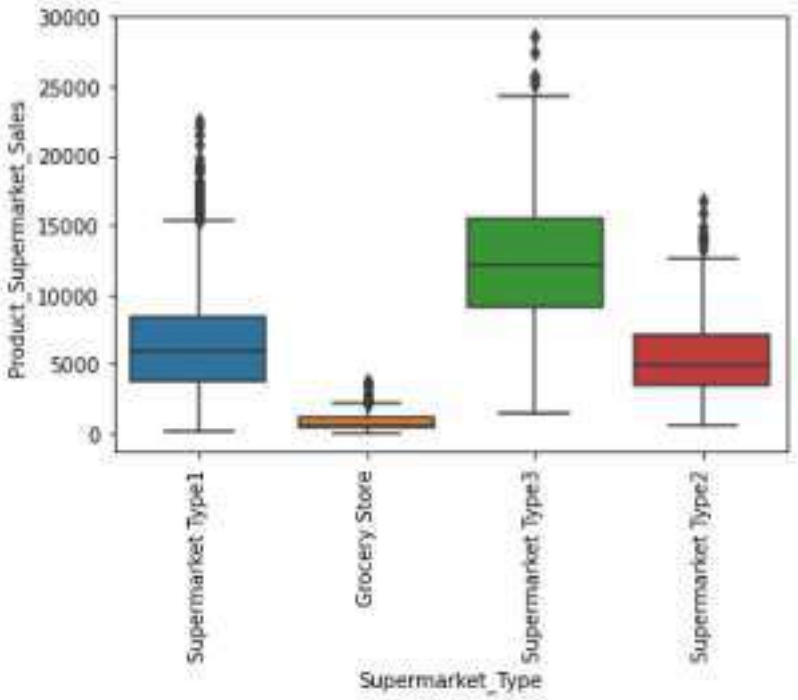
In [30]:
```
grouped_2=df[['Supermarket_Type', 'Product_Supermarket_Sales']]
grpd_2=grouped_2.groupby(['Supermarket_Type'],as_index=False).mean()
grpd_2
```

Out[30]:

	Supermarket_Type	Product_Supermarket_Sales
0	Grocery Store	821.426611
1	Supermarket Type1	6434.596537
2	Supermarket Type2	5588.015335
3	Supermarket Type3	12541.373382

In [31]:
```
sns.boxplot(x="Supermarket_Type", y="Product_Supermarket_Sales", data=df)
#rotate label if too long
plt.xticks(rotation=90)
```

Out[31]:
```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'Supermarket Type1'),
  Text(1, 0, 'Grocery Store'),
  Text(2, 0, 'Supermarket Type3'),
  Text(3, 0, 'Supermarket Type2')])
```



In [32]:
```
grpd_2=grouped_2.groupby(['Supermarket_Type'],as_index=False)
f_val, p_val = stats.f_oneway(grpd_2.get_group('Supermarket Type1')['Product_Supermarket_Sales'
print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 837.407760674732 , P = 0.0
```

- We see that the distributions of sales between the different supermarket-type categories are distinct, and enough to take Supermarket Type as a potential good predictor of sales price

Let's examine Supermarket_Location and Supermarket sales

Supermarket_Location_Type analysis

In [33]:
```
grouped_3=df[['Supermarket_Location_Type', 'Product_Supermarket_Sales']]
grpd_3=grouped_3.groupby(['Supermarket_Location_Type'],as_index=False).mean()
grpd_3
```

Out[33]:

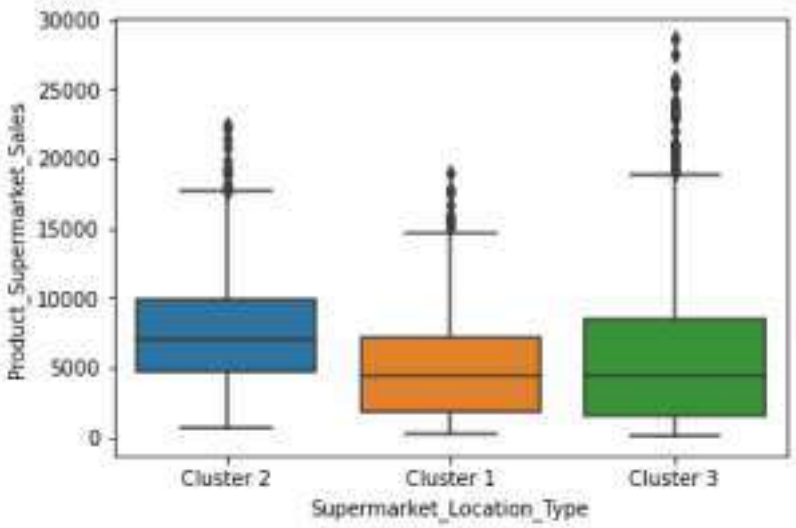
	Supermarket_Location_Type	Product_Supermarket_Sales
0	Cluster 1	4907.270494
1	Cluster 2	7581.598620
2	Cluster 3	5770.437719

In [34]:
```
df['Supermarket_Location_Type'].value_counts()
```

Out[34]:
```
Cluster 3    1438
Cluster 2    1210
Cluster 1    1094
Name: Supermarket_Location_Type, dtype: int64
```

In [35]:
```
sns.boxplot(x="Supermarket_Location_Type", y="Product_Supermarket_Sales", data=df)
```

Out[35]:
```
<AxesSubplot:xlabel='Supermarket_Location_Type', ylabel='Product_Supermarket_Sales'>
```



- We see that the distributions of sales between the different Supermarket-Location categories have a significant overlap, and so Supermarket-Location would not be a good predictor of sales.

In [36]:
```
grpd_3=grouped_3.groupby(['Supermarket_Location_Type'],as_index=False)
f_val, p_val = stats.f_oneway(grpd_3.get_group('Cluster 1')['Product_Supermarket_Sales'], grpd_
print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 116.808702799811 , P = 6.199756735425813e-50
```

- The Anova result says otherwise from what the Boxplot suggests. I might have to consider Supermarket location as a potential predictor

Product_Fat_Content analysis

In [37]:
```
df['Product_Fat_Content'].value_counts()
```

Out[37]:
```
Low Fat          2272
Normal Fat       1341
Ultra Low fat     129
Name: Product_Fat_Content, dtype: int64
```

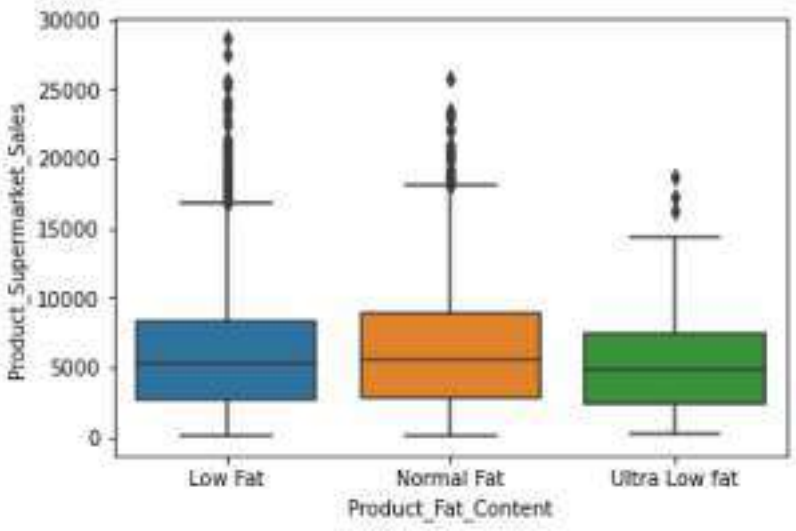
In [38]:
```
grouped_4=df[['Product_Fat_Content', 'Product_Supermarket_Sales']]
grpd_4=grouped_4.groupby(['Product_Fat_Content'],as_index=False).mean()
grpd_4
```

Out[38]:

	Product_Fat_Content	Product_Supermarket_Sales
0	Low Fat	6038.504696
1	Normal Fat	6287.411447
2	Ultra Low fat	5343.226899

In [39]:
```
sns.boxplot(x='Product_Fat_Content', y='Product_Supermarket_Sales', data=df)
```

Out[39]:
```
<AxesSubplot:xlabel='Product_Fat_Content', ylabel='Product_Supermarket_Sales'>
```



In [40]:
```
grpd_4=grouped_4.groupby(['Product_Fat_Content'],as_index=False)
f_val, p_val = stats.f_oneway(grpd_4.get_group('Low Fat')['Product_Supermarket_Sales'], grpd_4.
print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 3.265164001492282 , P = 0.03829959562702081
```

- In the boxplot, we see that the distribution of sales between the different product-fat-content categories have a significant overlap and as such Product-fat-content could not be a predictor of sales.
- With low F test score showing a weak correlation and a P value of almost 0 implying almost certain statistical significance, it further confirm that Product-fat-content cannot be a predictor of supermarket sales

supermarket_size analysis

In [41]:
```
df['Supermarket _Size'].value_counts()
```

Out[41]:
```
Medium    2281
Small     1015
High       446
Name: Supermarket _Size, dtype: int64
```

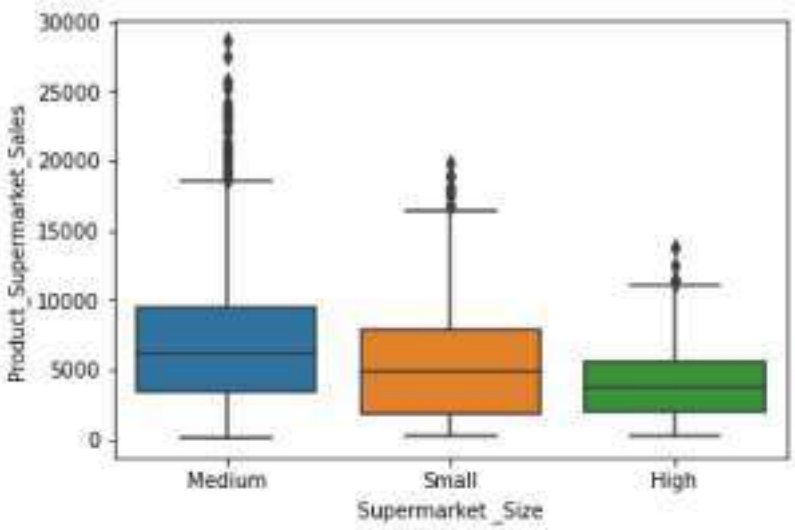
In [42]:
```
grouped_5=df[['Supermarket _Size', 'Product_Supermarket_Sales']]
grpd_5=grouped_5.groupby(['Supermarket _Size'],as_index=False).mean()
grpd_5
```

Out[42]:

	Supermarket _Size	Product_Supermarket_Sales
0	High	3998.670359
1	Medium	6844.184498
2	Small	5364.715143

In [43]:
```
sns.boxplot(x='Supermarket _Size', y='Product_Supermarket_Sales', data=df)
```

Out[43]:
```
<AxesSubplot:xlabel='Supermarket _Size', ylabel='Product_Supermarket_Sales'>
```



In [44]:
```
grpd_5=grouped_5.groupby(['Supermarket _Size'],as_index=False)
f_val, p_val = stats.f_oneway(grpd_5.get_group('Medium')['Product_Supermarket_Sales'], grpd_5.g
print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 100.26367160481566 , P = 3.8331989326796805e-43
```

Here we see that the distribution of sales between the different supermarket size categories differs; as such supermarket-size could potentially be a predictor of supermarket sales.

Product_Type analysis

In [45]:
```
df['Product_Type'].value_counts()
```

Out[45]:
```
Snack Foods              563
Fruits and Vegetables    556
Household                414
Frozen Foods             334
Canned                   294
Dairy                    263
Baking Goods             259
Health and Hygiene       222
Meat                     202
Soft Drinks              193
Breads                   115
Hard Drinks               97
Others                    82
Starchy Foods             62
Breakfast                 54
Seafood                   32
Name: Product_Type, dtype: int64
```

In [46]:
```
grouped_6=df[['Product_Type','Product_Supermarket_Sales']]
grpd_6=grouped_6.groupby(['Product_Type'],as_index=False).mean()
grpd_6
```

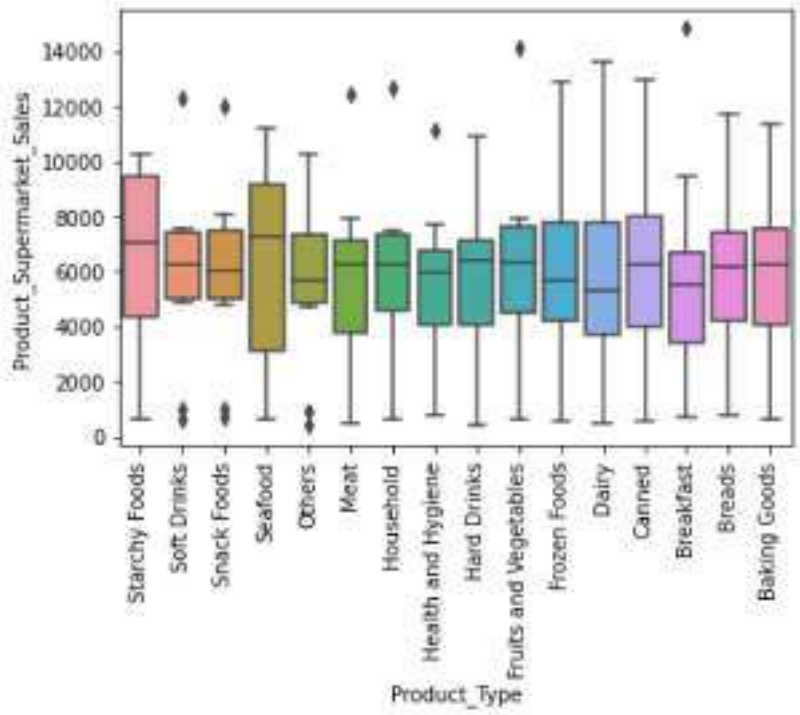
Out[46]:

	Product_Type	Product_Supermarket_Sales
0	Baking Goods	5635.721351
1	Breads	5875.077043
2	Breakfast	5867.887222
3	Canned	6392.173197

	Product_Type	Product_Supermarket_Sales
4	Dairy	6081.969734
5	Frozen Foods	5980.055359
6	Fruits and Vegetables	6429.236259
7	Hard Drinks	6097.784845
8	Health and Hygiene	5476.370586
9	Household	6158.381329
10	Meat	5881.733663
11	Others	5460.596341
12	Seafood	6651.707500
13	Snack Foods	6280.732575
14	Soft Drinks	5965.085389
15	Starchy Foods	7166.425000

In [47]:
```
sns.boxplot(x='Product_Type', y='Product_Supermarket_Sales', data=grpd_1)
#rotate label if too long
plt.xticks(rotation=90)
```

Out[47]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Starchy Foods'),
  Text(1, 0, 'Soft Drinks'),
  Text(2, 0, 'Snack Foods'),
  Text(3, 0, 'Seafood'),
  Text(4, 0, 'Others'),
  Text(5, 0, 'Meat'),
  Text(6, 0, 'Household'),
  Text(7, 0, 'Health and Hygiene'),
  Text(8, 0, 'Hard Drinks'),
  Text(9, 0, 'Fruits and Vegetables'),
  Text(10, 0, 'Frozen Foods'),
  Text(11, 0, 'Dairy'),
  Text(12, 0, 'Canned'),
  Text(13, 0, 'Breakfast'),
  Text(14, 0, 'Breads'),
  Text(15, 0, 'Baking Goods')])
```



- We see that the distributions of supermarket-sales between the different Product types categories have a significant overlap, and so product-types would not be a good predictor of sales.

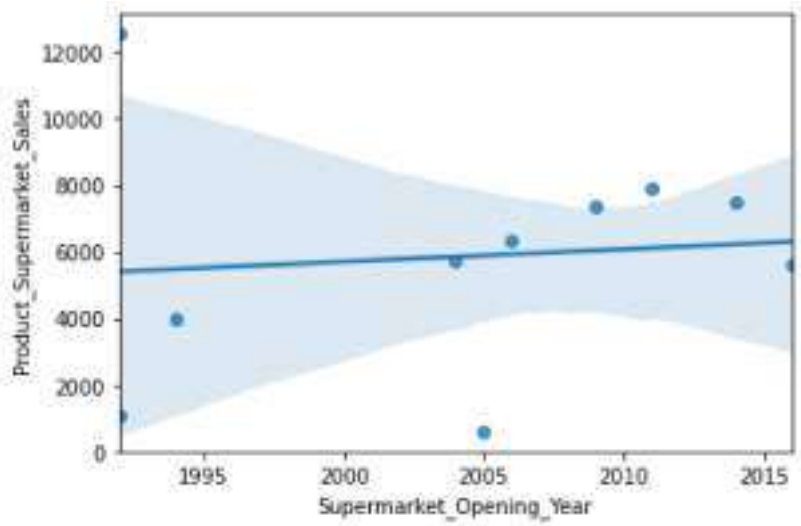
In [48]:
```
grouped_7=df[['Supermarket_Opening_Year','Supermarket_Identifier','Product_Supermarket_Sales']]
grpd_7=grouped_7.groupby(['Supermarket_Opening_Year','Supermarket_Identifier'],as_index=False).
grpd_7.sort_values(['Product_Supermarket_Sales'], ascending=False)
```

Out[48]:

	Supermarket_Opening_Year	Supermarket_Identifier	Product_Supermarket_Sales
1	1992	CHUKWUDI027	12541.373382
7	2011	CHUKWUDI035	7907.270484
8	2014	CHUKWUDI017	7468.619197
6	2009	CHUKWUDI045	7398.398568
5	2006	CHUKWUDI049	6352.267561
3	2004	CHUKWUDI046	5753.361804
9	2016	CHUKWUDI018	5588.015335
2	1994	CHUKWUDI013	3998.670359
0	1992	CHUKWUDI019	1064.223020
4	2005	CHUKWUDI010	606.450625

In [49]:
```
# Engine size as potential predictor variable of price
sns.regplot(x="Supermarket_Opening_Year", y="Product_Supermarket_Sales", data=grpd_7)
plt.ylim(0,)
```

Out[49]:
```
(0.0, 13146.66742793651)
```



In [50]:
```
grpd_7.corr()
```

Out[50]:

	Supermarket_Opening_Year	Product_Supermarket_Sales
Supermarket_Opening_Year	1.000000	0.095753
Product_Supermarket_Sales	0.095753	1.000000

In [51]:
```
df.corr()
```

Out[51]:

	Product_Weight	Product_Shelf_Visibility	Product_Price	Supermarket_Opening_Year	Pr
Product_Weight	1.000000	-0.015260	0.037564	0.000649	
Product_Shelf_Visibility	-0.015260	1.000000	-0.028941	-0.084218	
Product_Price	0.037564	-0.028941	1.000000	0.161113	
Supermarket_Opening_Year	0.000649	-0.084218	0.161113	1.000000	
Product_Supermarket_Sales	0.015314	-0.161008	0.524002	0.057508	

In [52]:

```
df.columns
```

Out[52]:

```
Index(['Product_Identifier', 'Supermarket_Identifier',
       'Product_Supermarket_Identifier', 'Product_Weight',
       'Product_Fat_Content', 'Product_Shelf_Visibility', 'Product_Type',
       'Product_Price', 'Supermarket_Opening_Year', 'Supermarket _Size',
       'Supermarket_Location_Type', 'Supermarket_Type',
       'Product_Supermarket_Sales'],
      dtype='object')
```

- Opening Year would as well not be a good predictor of sales

Summary

We now have a better idea of what our data looks like and which variables are important to take into account when predicting the supermarket sales price. We have narrowed it down to the following variables:

Continuous numerical variables:

- Product_Weight
- Product_Price
- Product_Shelf_Visibility

Categorical variables:

- Supermarket _Size
- Supermarket_Location_Type
- Supermarket_Type
- Product_Type

As we now move into building machine learning models to automate our analysis, feeding the model with variables that meaningfully affect our target variable will improve our model's prediction performance.

Data Transformation

- Assign dependent and independent variables (drop redundant columns)
- One-hot encode categorical variables
- Do label encoding for hierachical categories
- Normalize the eatures

In [203...

```
# label encoding
# X2 = X.copy()

# sm_size_map = {'Small':1, 'Medium':2, 'High':3}
# X2['Supermarket _Size'] = X2['Supermarket _Size'].map(sm_size_map).astype("float")
```

```
# sm_loc_map = {'Cluster 1':1, 'Cluster 3':3, 'Cluster 2':2}
# X2['Supermarket_Location_Type'] = X2['Supermarket_Location_Type'].map(sm_loc_map).astype("flo

# sm_type_map = {'Grocery Store':1, 'Supermarket Type2':2, 'Supermarket Type1':3, 'Supermarket
# X2['Supermarket_Type'] = df['Supermarket_Type'].map(sm_type_map).astype("float")




# prod_fat_map = {'Ultra Low fat':1, 'Low Fat':2, 'Normal Fat':3}
# X2['Product_Fat_Content'] = X2['Product_Fat_Content'].map(prod_fat_map)
```

- model performances were poor with label encoding

In [322]:
```
# independent features
feature_set = ['Product_Weight', 'Product_Shelf_Visibility', 'Product_Type', 'Product_Price',
               'Supermarket _Size', 'Supermarket_Location_Type', 'Supermarket_Type']

X = df.drop(['Product_Identifier','Supermarket_Identifier','Product_Supermarket_Identifier',
           'Product_Fat_Content','Supermarket_Opening_Year','Product_Supermarket_Sales'], axis=
# dependent variable
y = df['Product_Supermarket_Sales'].values
```

In [323]:
```
X.head()
```

Out[323]:

	Product_Weight	Product_Shelf_Visibility	Product_Type	Product_Price	Supermarket _Size	Supermarket_Location_Type
0	8.785	0.020694	Soft Drinks	382.91	None	Cluster 2
1	NaN	0.086078	Fruits and Vegetables	359.53	Small	Cluster 1
2	18.600	0.039517	Frozen Foods	609.20	None	Cluster 2
3	NaN	0.091780	Frozen Foods	456.32	Medium	Cluster 3
4	14.000	0.023735	Baking Goods	258.08	Small	Cluster 1

In [324]:
```
df2 = df2.drop(['Product_Identifier','Supermarket_Identifier','Product_Supermarket_Identifier',
                'Product_Fat_Content','Supermarket_Opening_Year'], axis=1)
df2.head()
```

Out[324]:

	Product_Weight	Product_Shelf_Visibility	Product_Type	Product_Price	Supermarket _Size	Supermarket_Location_Type
0	18.100	0.178246	Dairy	395.32	Medium	Cluster 1
1	15.250	0.061434	Household	327.49	Medium	Cluster 3
2	8.785	0.020573	Soft Drinks	391.16	Small	Cluster 2
3	12.100	0.080131	Household	427.28	Medium	Cluster 3
4	6.965	0.028710	Household	395.40	None	Cluster 2

In [325]:
```
X = pd.get_dummies(X)
df2 = pd.get_dummies(df2)
# show
X.head()
df2 .head()
```

Out[325...

	Product_Weight	Product_Shelf_Visibility	Product_Price	Product_Type_Baking Goods	Product_Type_Breads	Product_Ty
0	18.100	0.178246	395.32	0	0	
1	15.250	0.061434	327.49	0	0	
2	8.785	0.020573	391.16	0	0	
3	12.100	0.080131	427.28	0	0	
4	6.965	0.028710	395.40	0	0	

5 rows × 29 columns

Normalization

In [330...

```
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = X.copy()
X_scaled[X_scaled.columns] = scaler.transform(X_scaled)

df2[df2.columns]  = scaler.transform(df2)

# show
X_scaled.head()
```

Out[330...

	Product_Weight	Product_Shelf_Visibility	Product_Price	Product_Type_Baking Goods	Product_Type_Breads	Product_Ty
0	0.251861	0.063015	0.516982	0.0	0.0	
1	NaN	0.262120	0.477153	0.0	0.0	
2	0.836261	0.120336	0.901409	0.0	0.0	
3	NaN	0.279484	0.641625	0.0	0.0	
4	0.562370	0.072276	0.304763	1.0	0.0	

5 rows × 29 columns

In [332...

```
df2 = df2[X.columns]
df2.head()
```

Out[332...

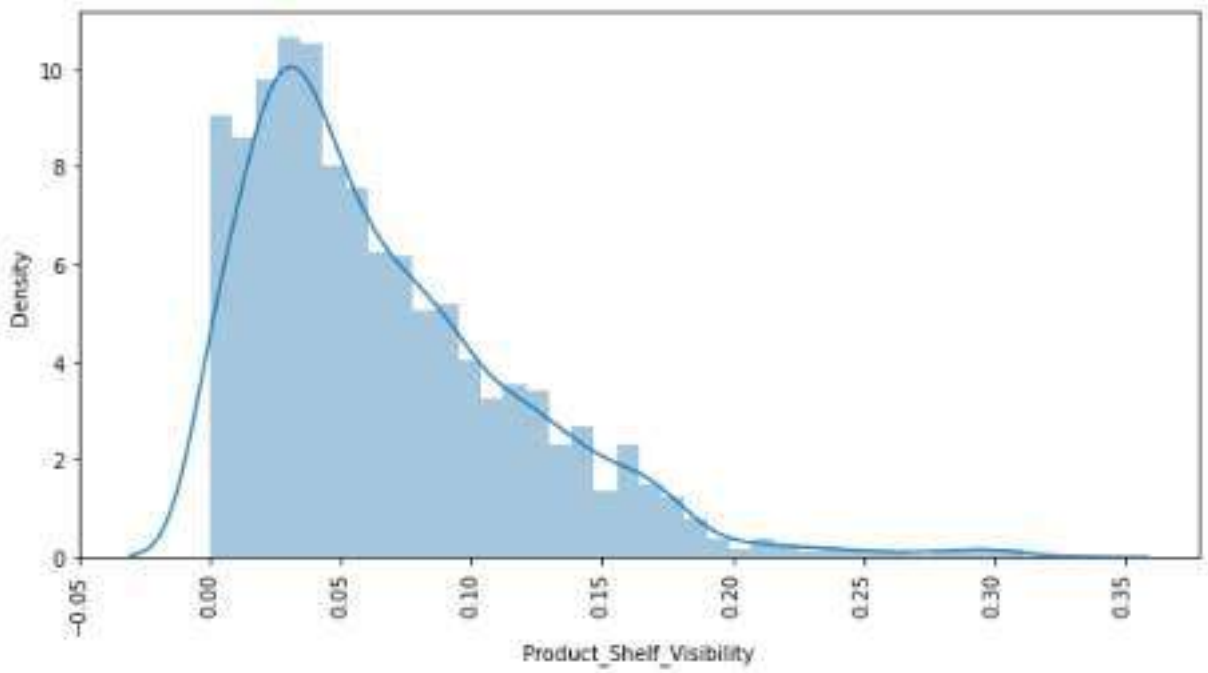
	Product_Weight	Product_Shelf_Visibility	Product_Price	Product_Type_Baking Goods	Product_Type_Breads	Product_Ty
0	0.806490	0.542785	0.537970	0.0	0.0	
1	0.636797	0.187076	0.422709	0.0	0.0	
2	0.251861	0.062649	0.530901	0.0	0.0	
3	0.449241	0.244012	0.592279	0.0	0.0	
4	0.143495	0.087426	0.538106	0.0	0.0	

5 rows × 29 columns

In [252...

```
plt.figure(figsize=(10,5))
sns.distplot(X["Product_Shelf_Visibility"])
#rotate label if too long
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [253...

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=1)
```

Model

I am proposing 3 algorithms:

- Linear Resressor
- DecisionTree Regressor and
- Random Forest Regressor from which the best model will be selected

For better evaluation, we'll be using Cross-Validation:

Linear Regression Model

In [254...

```
def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName, Title):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    ax1 = sns.distplot(RedFunction, hist=False, color="r", label=RedName)
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", label=BlueName, ax=ax1)

    plt.title(Title)
    plt.xlabel('Price (in naira)')
    plt.ylabel('Proportion of Products')
    plt.legend()
```

```
    plt.show()
    plt.close()
```

In [255]:
```
def PollyPlot(xtrain, xtest, y_train, y_test, lr,poly_transform):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))


    #training data
    #testing data
    # lr:  linear regression object
    #poly_transform:  polynomial transformation object

    xmax=max([xtrain.values.max(), xtest.values.max()])

    xmin=min([xtrain.values.min(), xtest.values.min()])

    x=np.arange(xmin, xmax, 0.1)


    plt.plot(xtrain, y_train, 'ro', label='Training Data')
    plt.plot(xtest, y_test, 'go', label='Test Data')
    plt.plot(x, lr.predict(poly_transform.fit_transform(x.reshape(-1, 1))), label='Predicted Fu
    plt.ylim([-10000, 60000])
    plt.ylabel('Price')
    plt.legend()
```

In [256]:
```
# Fit the linear model
lre=LinearRegression()
results = lre.fit(X_train,y_train)
print(results)
```

```
LinearRegression()
```

In [257]:
```
# Print the coefficients
print (results.intercept_, results.coef_)
```

```
9132569287176418.0 [-2.71254583e+01 -7.13649310e+02  8.82484317e+03 -3.79502987e+15
 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15
 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15
 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15
 -3.79502987e+15 -3.79502987e+15 -3.79502987e+15 -2.05098209e+15
 -2.05098209e+15 -2.05098209e+15  2.06246194e+15  2.06246194e+15
  2.06246194e+15 -5.34901927e+15 -5.34901927e+15 -5.34901927e+15
 -5.34901927e+15]
```

In [258]:
```
print("train score:", lre.score(X_train,y_train))

print("test score:", lre.score(X_test,y_test))
```

```
train score: 0.5567631629208285
test score: 0.5594024666325306
```

In [259]:
```
lre2 = sm.OLS(y_train, X_train)
results2 = lre2.fit()
# Statsmodels gives R-like statistical output
results2.summary()
```

Out[259]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.557
Model:	OLS	**Adj. R-squared:**	0.553

Method:	Least Squares	**F-statistic:**	149.1
Date:	Fri, 19 Aug 2022	**Prob (F-statistic):**	0.00
Time:	20:23:45	**Log-Likelihood:**	-28140.
No. Observations:	2993	**AIC:**	5.633e+04
Df Residuals:	2967	**BIC:**	5.649e+04
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Product_Weight	-26.4701	193.494	-0.137	0.891	-405.867	352.927
Product_Shelf_Visibility	-716.6102	352.334	-2.034	0.042	-1407.453	-25.767
Product_Price	8821.8359	346.777	25.439	0.000	8141.888	9501.784
Product_Type_Baking Goods	0.2637	208.560	0.001	0.999	-408.673	409.200
Product_Type_Breads	-248.6158	295.909	-0.840	0.401	-828.823	331.592
Product_Type_Breakfast	-63.9440	424.434	-0.151	0.880	-896.160	768.272
Product_Type_Canned	91.2765	195.753	0.466	0.641	-292.550	475.103
Product_Type_Dairy	-256.5026	206.297	-1.243	0.214	-661.002	147.997
Product_Type_Frozen Foods	109.7384	184.585	0.595	0.552	-252.189	471.666
Product_Type_Fruits and Vegetables	262.0497	151.236	1.733	0.083	-34.488	558.587
Product_Type_Hard Drinks	-20.5937	323.638	-0.064	0.949	-655.171	613.984
Product_Type_Health and Hygiene	-152.0829	224.123	-0.679	0.497	-591.536	287.370
Product_Type_Household	-197.1269	166.218	-1.186	0.236	-523.041	128.787
Product_Type_Meat	18.1185	227.273	0.080	0.936	-427.511	463.748
Product_Type_Others	-310.7060	377.694	-0.823	0.411	-1051.275	429.863
Product_Type_Seafood	703.9602	569.992	1.235	0.217	-413.660	1821.580
Product_Type_Snack Foods	42.0077	150.146	0.280	0.780	-252.394	336.409
Product_Type_Soft Drinks	-85.3506	237.121	-0.360	0.719	-550.289	379.588
Product_Type_Starchy Foods	323.5387	400.604	0.808	0.419	-461.951	1109.028
Supermarket _Size_High	-1633.6966	341.707	-4.781	0.000	-2303.704	-963.689
Supermarket _Size_Medium	831.2637	109.932	7.562	0.000	615.713	1046.814
Supermarket _Size_Small	1018.4640	150.780	6.755	0.000	722.820	1314.108
Supermarket_Location_Type_Cluster 1	-728.0603	210.311	-3.462	0.001	-1140.431	-315.690
Supermarket_Location_Type_Cluster 2	-534.9412	261.339	-2.047	0.041	-1047.366	-22.517
Supermarket_Location_Type_Cluster 3	1479.0326	161.509	9.158	0.000	1162.352	1795.713
Supermarket_Type_Grocery Store	-4357.8670	155.870	-27.958	0.000	-4663.492	-4052.242
Supermarket_Type_Supermarket Type1	1632.3347	173.862	9.389	0.000	1291.432	1973.238
Supermarket_Type_Supermarket Type2	-1298.6317	238.239	-5.451	0.000	-1765.762	-831.501
Supermarket_Type_Supermarket Type3	4240.1951	266.004	15.940	0.000	3718.623	4761.767

Omnibus:	198.436	**Durbin-Watson:**	2.011
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	362.904

Skew:	0.482	**Prob(JB):**	1.57e-79
Kurtosis:	4.408	**Cond. No.**	1.25e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.85e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

- Here the Ordinary Least Squares (OLS) method has given the r^2 value of 0.557 which is above average but still poor for model to be predicted on these variables. The higher (closer to one) the better.

In [261...

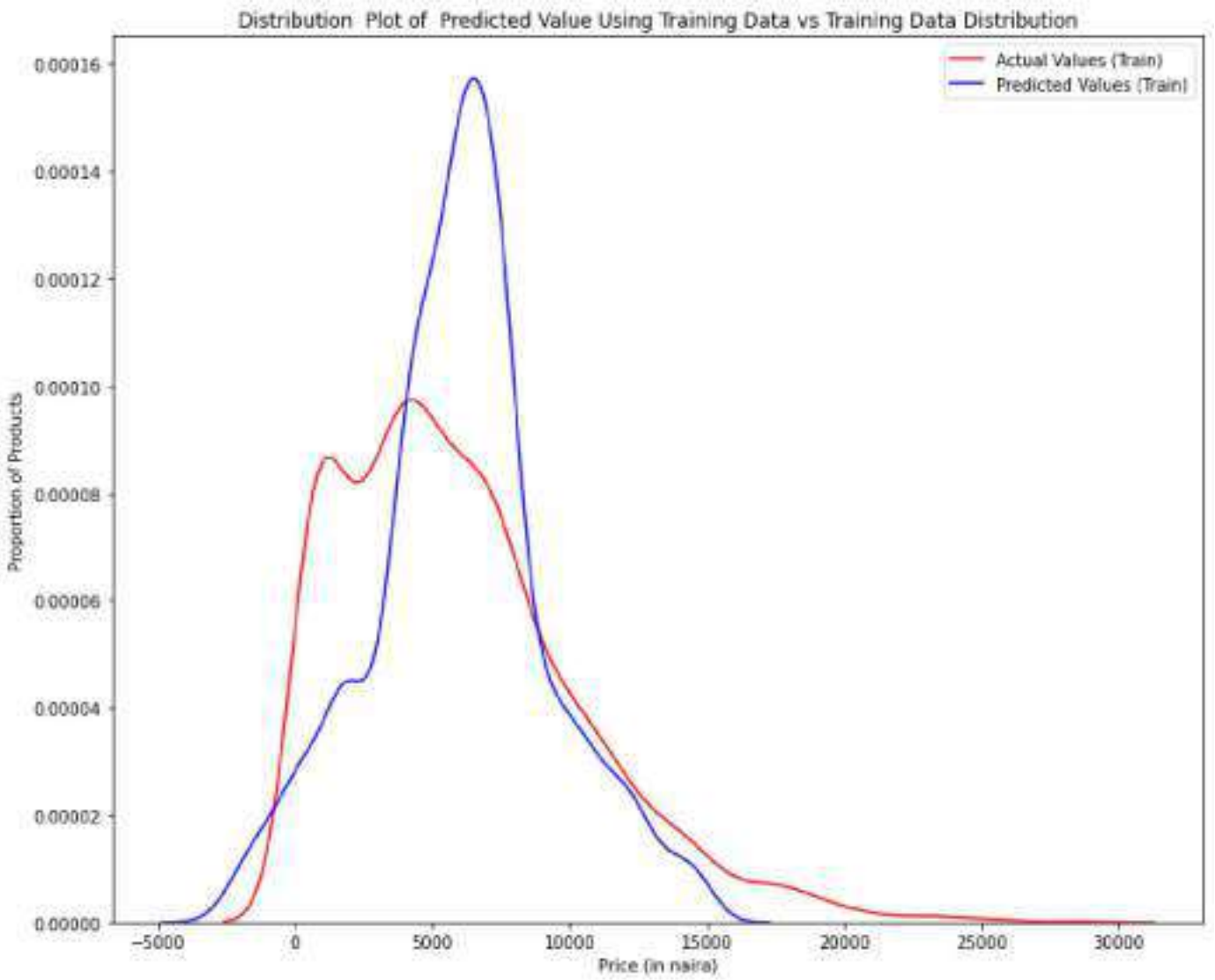
```
print('The mean square error of price and predicted value using multifit is: ', \
      mean_squared_error(y_train, lre.predict(X_train)))

print('The mean square error of price and predicted value using multifit is: ', \
      mean_squared_error(y_test, lre.predict(X_test)))
```

```
The mean square error of price and predicted value using multifit is:  8589898.856294755
The mean square error of price and predicted value using multifit is:  9507257.271802539
```

In [262...

```
Title = 'Distribution  Plot of  Predicted Value Using Training Data vs Training Data Distributi
DistributionPlot(y_train, lre.predict(X_train), "Actual Values (Train)", "Predicted Values (Tra
```

```
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an
axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an
axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



In [263...

```
yhat_test=lre.predict(X_test)
yhat_test[:5]
```

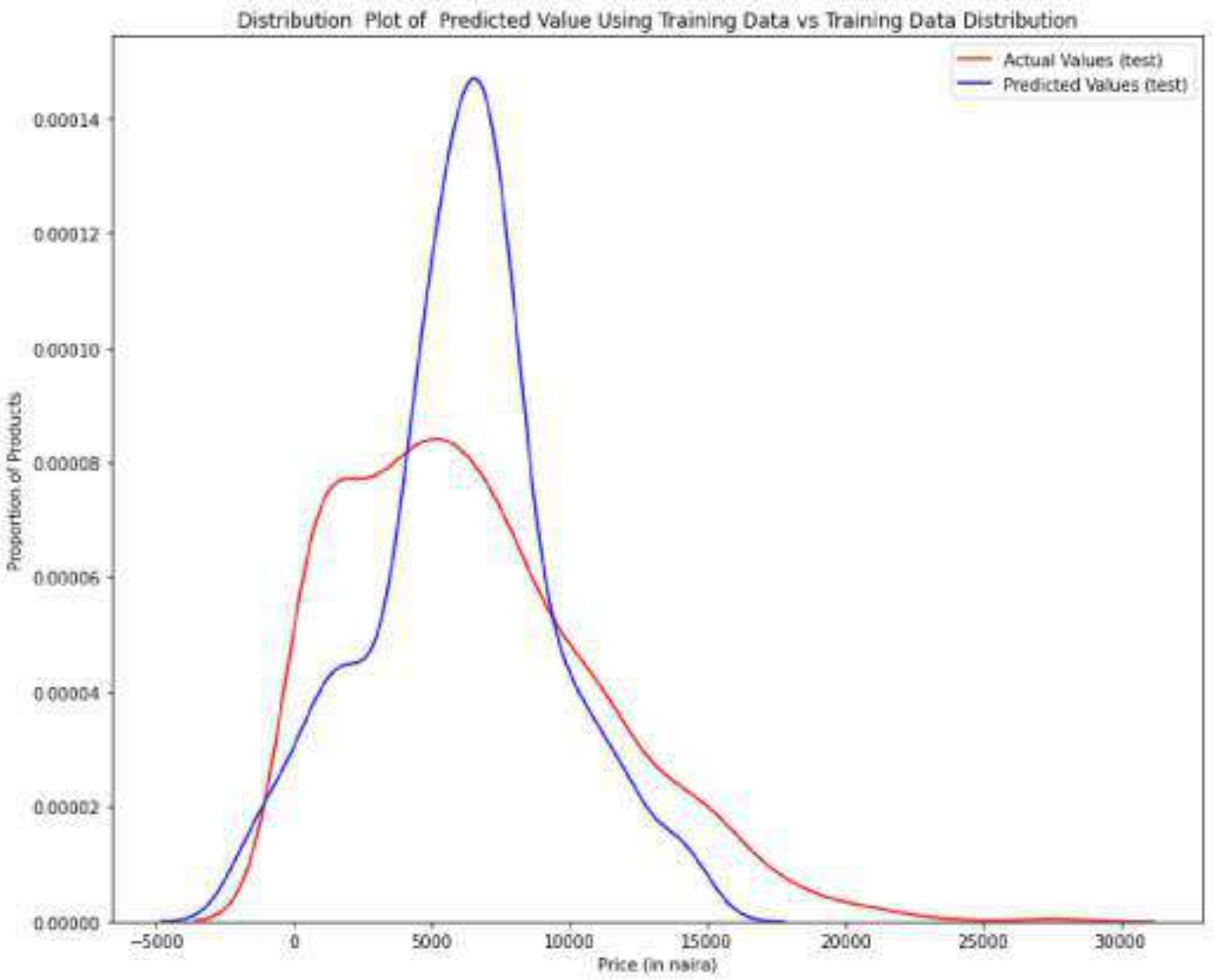
Out[263...

```
array([14926.,   124.,  2726.,  -998.,  6758.])
```

In [264...

```
Title = 'Distribution  Plot of  Predicted Value Using Training Data vs Training Data Distributi
DistributionPlot(y_test, lre.predict(X_test), "Actual Values (test)", "Predicted Values (test)"
```

```
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an
axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\DeleLinus\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your co
de to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an
axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



Using Cross validation

- R^2 value is basically dependant on the way the data is split. Hence, there may be times when the R^2 value may not be able to represent the model's ability to generalize. For this we perform cross validation.

In [265]:
```
lre=LinearRegression()
Rcross = cross_val_score(lre, X_scaled, y, cv=5)
Rcross

print("The mean of the folds are", Rcross.mean(), "and the standard deviation is" , Rcross.std(

rc=cross_val_score(lre, X_scaled, y, cv=4,scoring='neg_mean_squared_error')
rc
```

Out[265]:
```
The mean of the folds are 0.5519293377802535 and the standard deviation is 0.025799316279967534
array([-9212882.17754359, -8915912.61166138, -9494408.32067818,
       -7863440.49679572])
```

Regularization

- Rregularization helps to alter the loss function to penalize it for having higher coefficients for each feature variable (large coefficients leads to overfitting).

In [266]:
```
ridge = Ridge(alpha=0.001, normalize = True)
ridge.fit(X_train,y_train)
ridge_pred=ridge.predict(X_test)
ridge.score(X_test,y_test)
```

Out[266...
```
0.5593934460747929
```

DecisionTree Regressor

In [267...
```
dtree = DecisionTreeRegressor(max_depth=10,min_samples_leaf=5,max_leaf_nodes=5)
```

In [268...
```
dtree.fit(X_train,y_train)
y_pred=dtree.predict(X_test)

print('R2 score = ',r2_score(y_test, y_pred), '/ 1.0')
print('MSE score = ',mean_squared_error(y_test, y_pred), '/ 0.0')
```

```
R2 score =  0.5269352991786305 / 1.0
MSE score =  10207837.030234888 / 0.0
```

In [269...
```
etr = ExtraTreesRegressor()

# Choose some parameter combinations to try

parameters = {'n_estimators': [5,10,100],
              'criterion': ['mse'],
              'max_depth': [5,10,15],
              'min_samples_split': [2,5,10],
              'min_samples_leaf': [1,5]
             }
grid_obj = GridSearchCV(etr, parameters,
                        cv=3,
                        n_jobs=-1, #Number of jobs to run in parallel
                        verbose=1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
etr = grid_obj.best_estimator_

# Fit the best algorithm to the data.
etr.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
```
Out[269...
```
ExtraTreesRegressor(max_depth=5, min_samples_leaf=5, min_samples_split=10,
                    n_estimators=10)
```

In [270...
```
y_pred = etr.predict(X_test)

print('R2 score = ',r2_score(y_test, y_pred), '/ 1.0')
print('MSE score = ',mean_squared_error(y_test, y_pred), '/ 0.0')
```

```
R2 score =  0.5753803801079813 / 1.0
MSE score =  9162484.26942916 / 0.0
```

Random Forest

In [271...
```
RFR = RandomForestRegressor()

parameters = {'n_estimators': [5, 10, 100],
              'min_samples_leaf': [1,2,5]
             }

grid_obj = GridSearchCV(RFR, parameters,
                        cv=5,
                        n_jobs=-1,
                        verbose=1)
grid_obj = grid_obj.fit(X_train, y_train)
```

```
RFR = grid_obj.best_estimator_

# Fit the best algorithm to the data.
RFR.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

Out[271...
```
RandomForestRegressor(min_samples_leaf=5)
```

In [272...
```
y_pred = RFR.predict(X_test)

print('R2 score = ',r2_score(y_test, y_pred), '/ 1.0')
print('MSE score = ',mean_squared_error(y_test, y_pred), '/ 0.0')
```

```
R2 score =  0.5636086340545008 / 1.0
MSE score =  9416496.173227103 / 0.0
```

In [273...
```
RFR = RandomForestRegressor()

# Choose some parameter combinations to try
parameters = {'n_estimators': [5,10,100],
              'criterion': ['mse'],
              'max_depth': [5,10,15],
              'min_samples_split': [2,5,10],
              'min_samples_leaf': [1,5]
             }


#We have to use RandomForestRegressor's own scorer (which is R^2 score)

grid_obj = GridSearchCV(RFR, parameters,
                        cv=5,
                        n_jobs=-1,
                        verbose=1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
RFR = grid_obj.best_estimator_

# Fit the best algorithm to the data.
RFR.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
```

Out[273...
```
RandomForestRegressor(max_depth=5, min_samples_split=10)
```

In [274...
```
y_pred = RFR.predict(X_test)

print('R2 score = ',r2_score(y_test, y_pred), '/ 1.0')
print('MSE score = ',mean_squared_error(y_test, y_pred), '/ 0.0')
```

```
R2 score =  0.5822145143353916 / 1.0
MSE score =  9015016.643298889 / 0.0
```

Conclusion:

Using **R^2** as the evaluation metric, the following scores are gotten for the algorithms used:

Model	score
Linear Resressor	0.559
Random Forest Regressor	0.582
Decision Tree	0.575

Further Improvement

- Feature interaction or engineering can be done to improve the models

In [344]:
```
y_pred = model.predict(features_validation, verbose=0)

print('R2 score = ',r2_score(y_test, y_pred), '/ 1.0')
print('MSE score = ',mean_squared_error(y_test, y_pred), '/ 0.0')
```

```
R2 score =  0.5713002006085784 / 1.0
MSE score =  9250526.787316661 / 0.0
```

Prediction

In [348]:
```
sub = sample_submission.copy()
sub['Product_Supermarket_Sales'] = model.predict(df2)
sub.to_csv('NN_submission.csv', index = False)
sub.head()
```

```
39/39 [==============================] - 0s 651us/step
```

Out[348]:

	Product_Supermarket_Identifier	Product_Supermarket_Sales
0	FDC15_CHUKWUDI049	5608.434082
1	NCY06_CHUKWUDI018	4197.079102
2	DRB24_CHUKWUDI035	6403.812012
3	NCG54_CHUKWUDI018	5901.750488
4	NCA42_CHUKWUDI017	6227.226074

In [ ]:
```

```