SuperKart Project

Context:

A sales forecast is a prediction of future sales revenue based on historical data, industry trends, and the status of the current sales pipeline. Businesses use the sales forecast to estimate weekly, monthly, quarterly, and annual sales totals. It is extremely important for a company to make an accurate sales forecast as it adds value across an organization and helps the different verticals to chalk out their future course of actions. Forecasting helps an organization to plan its sales operations by regions and provide valuable insights to the supply chain team regarding the procurement of goods and materials. An accurate sales forecast process has many benefits which include improved decision-making about the future and reduction of sales pipeline and forecast risks. Moreover, it helps to reduce the time spent in planning territory coverage and establish benchmarks that can be used to assess trends in the future.

Objective:

SuperKartKart is an organization which owns a chain of supermarkets and food marts providing a wide range of products. They want to predict the future sales revenue of its different outlets so that they can strategize their sales operation across different tier cities and plan their inventory accordingly. To achieve this purpose, SuperKart has hired a data science firm, shared the sales records of its various outlets for the previous quarter and asked the firm to come up with a suitable model to predict the total sales of the stores for the upcoming quarter.

Data Description:

The data contains the different attributes of the various products and stores.The detailed data dictionary is given below.

- Product_Id - unique identifier of each product, each identifier having two letters at the beginning followed by a number
- Product_Weight - weight of each product
- Product_Sugar_Content - sugar content of each product like low sugar, regular and no sugar

- Product_Allocated_Area - ratio of the allocated display area of each product to the total display area of all the products in a store
- Product_Type - broad category for each product like meat, snack foods, hard drinks, dairy, canned, soft drinks, health and hygiene, baking goods, breads, breakfast, frozen foods, fruits and vegetables, household, seafood, starchy foods, others
- Product_MRP - maximum retail price of each product
- Store_Id - unique identifier of each store
- Store_Establishment_Year - year in which the store was established
- Store_Size - size of the store depending on sq. feet like high, medium and low
- Store_Location_City_Type - type of city in which the store is located like Tier 1, Tier 2 and Tier 3. Tier 1 consists of cities where the standard of living is comparatively higher than its Tier 2 and Tier 3 counterparts.
- Store_Type - type of store depending on the products that are being sold there like Departmental Store, Supermarket Type 1, Supermarket Type 2 and Food Mart
- Product_Store_Sales_Total - total revenue generated by the sale of that particular product in that particular store

Importing necessary libraries and data

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

In [2]:
```
# Loading the dataset
df = pd.read_csv('SuperKart.csv')
df.head()
```

Out[2]:

	Product_Id	Product_Weight	Product_Sugar_Content	Product_Allocated_Area	Product_Type	Product_MRP	Store_Id
0	FD6114	12.66	Low Sugar	0.027	Frozen Foods	117.08	OUT004
1	FD7839	16.54	Low Sugar	0.144	Dairy	171.43	OUT003
2	FD5075	14.28	Regular	0.031	Canned	162.08	OUT001
3	FD8233	12.10	Low Sugar	0.112	Baking Goods	186.31	OUT001
4	NC1180	9.57	No Sugar	0.010	Health and Hygiene	123.67	OUT002

Data Preprocessing

In [3]:

```
# Checking missing values
df.isnull().sum()
```

Out[3]:

```
Product_Id                   0
Product_Weight               0
Product_Sugar_Content        0
Product_Allocated_Area       0
Product_Type                 0
Product_MRP                  0
Store_Id                     0
Store_Establishment_Year     0
Store_Size                   0
Store_Location_City_Type     0
Store_Type                   0
Product_Store_Sales_Total    0
dtype: int64
```

In [5]:

```
# handling missig values w. forward fill for simplicity
df.fillna(method='ffill', inplace=True)
```

Data Overview

- Observations
- Sanity checks

In [52]:
```
# Checking missing values
df.isnull().sum()
```

Out[52]:
```
Product_Id                   0
Product_Weight               0
Product_Sugar_Content        0
Product_Allocated_Area       0
Product_Type                 0
Product_MRP                  0
Store_Id                     0
Store_Establishment_Year     0
Store_Size                   0
Store_Location_City_Type     0
Store_Type                   0
Product_Store_Sales_Total    0
dtype: int64
```

In [53]:
```
df.describe()
```

Out[53]:

	Product_Weight	Product_Allocated_Area	Product_MRP	Store_Establishment_Year	Product_Store_Sales_Total
count	8763.000000	8763.000000	8763.000000	8763.000000	8763.000000
mean	12.653792	0.068786	147.032539	2002.032751	3464.003640
std	2.217320	0.048204	30.694110	8.388381	1065.630494
min	4.000000	0.004000	31.000000	1987.000000	33.000000
25%	11.150000	0.031000	126.160000	1998.000000	2761.715000
50%	12.660000	0.056000	146.740000	2009.000000	3452.340000
75%	14.180000	0.096000	167.585000	2009.000000	4145.165000
max	22.000000	0.298000	266.000000	2009.000000	8000.000000

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Product_Id                 8763 non-null   object 
 1   Product_Weight             8763 non-null   float64
 2   Product_Sugar_Content      8763 non-null   object 
 3   Product_Allocated_Area     8763 non-null   float64
 4   Product_Type               8763 non-null   object 
 5   Product_MRP                8763 non-null   float64
 6   Store_Id                   8763 non-null   object 
 7   Store_Establishment_Year   8763 non-null   int64  
 8   Store_Size                 8763 non-null   object 
 9   Store_Location_City_Type   8763 non-null   object 
 10  Store_Type                 8763 non-null   object 
 11  Product_Store_Sales_Total  8763 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 821.7+ KB
```

In [54]:
```
# Encoding categorical variables
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])
```

In [55]:
```
# Scaling numerical variables
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])
```

In [56]:
```
# Lets combine transformations into a ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

In [57]:
```
df.head(10)
```

Out[57]:

	Product_Id	Product_Weight	Product_Sugar_Content	Product_Allocated_Area	Product_Type	Product_MRP	Store_Id
0	FD6114	12.66	Low Sugar	0.027	Frozen Foods	117.08	OUT004
1	FD7839	16.54	Low Sugar	0.144	Dairy	171.43	OUT003
2	FD5075	14.28	Regular	0.031	Canned	162.08	OUT001
3	FD8233	12.10	Low Sugar	0.112	Baking Goods	186.31	OUT001
4	NC1180	9.57	No Sugar	0.010	Health and Hygiene	123.67	OUT002
5	FD5680	12.03	Low Sugar	0.053	Snack Foods	113.64	OUT004
6	FD5484	16.35	Low Sugar	0.112	Meat	185.71	OUT003
7	NC5885	12.94	No Sugar	0.286	Household	194.75	OUT003
8	FD1961	9.45	Low Sugar	0.047	Snack Foods	95.95	OUT002
9	NC6657	8.94	No Sugar	0.045	Health and Hygiene	143.01	OUT004

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions

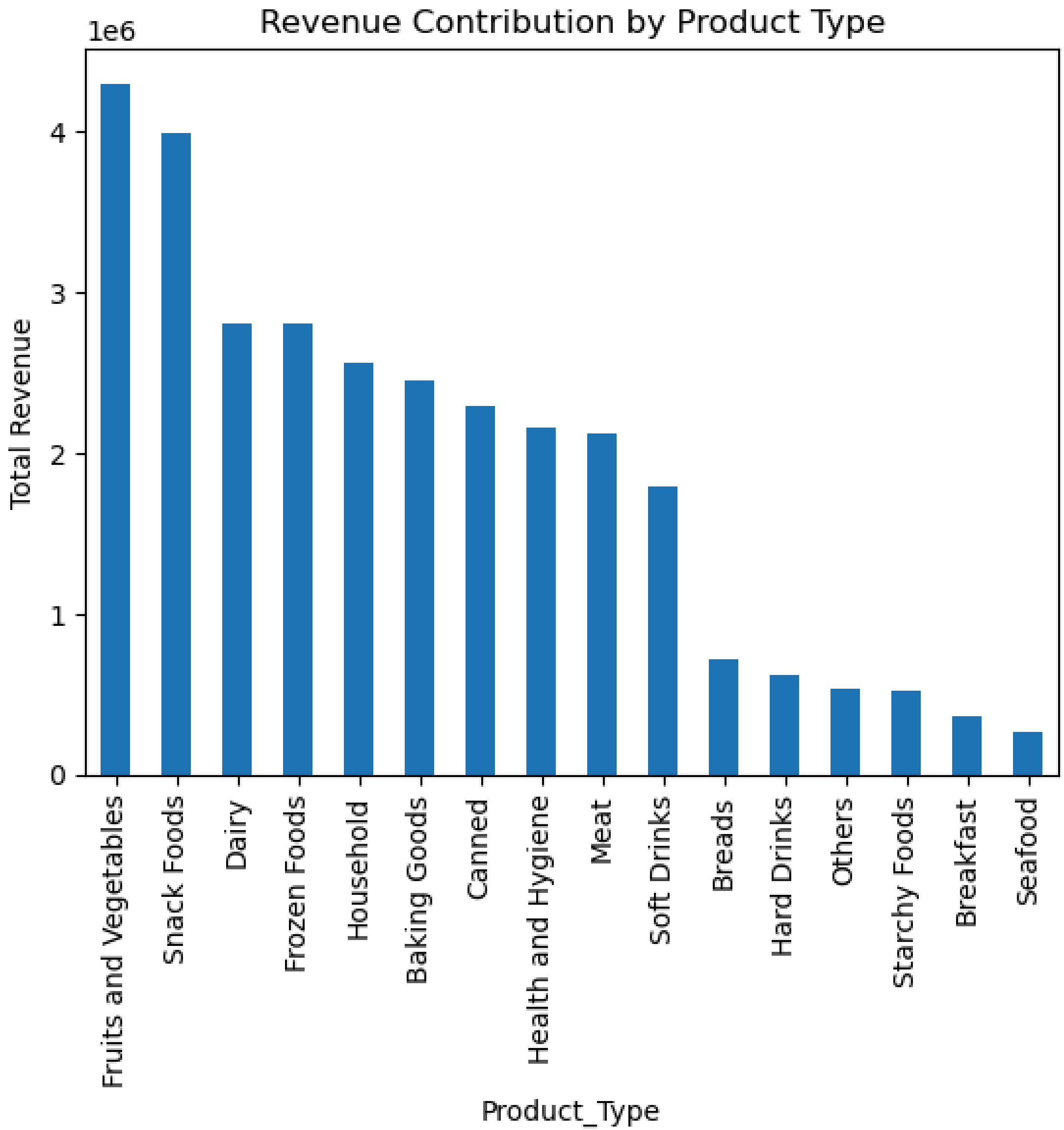
- Different varieties of products are available at stores. A store needs to plan its inventory appropriately which is well aligned to the supply and demand characteristics. Which product type is contributing the most to the revenue of the company (SuperKart)?
- Location may have a high impact on the revenue of a store. Find out the type of stores and locations that are having a high impact on the revenue of the company.
- Nowadays many customers prefer products that have low sugar content in them. How many items have been sold in each of the 16 product types that have low sugar content in them?
- Which product type has been sold the most number of times in each of the stores? Which product type is contributing the most to the revenue of the individual stores?
- There are some stores of a company that generally sell only premium items having higher prices than others. Which store has sold more costly goods than others?

In [12]:
```
# Grouping by Product_Type and summing up the Product_Store_Sales_Total to find the contribution to revenue
product_revenue = df.groupby('Product_Type')['Product_Store_Sales_Total'].sum().sort_values(ascending=False
print(product_revenue)
```

```
Product_Type
Fruits and Vegetables    4300833.27
Snack Foods              3988996.95
Dairy                    2811918.04
Frozen Foods             2809980.83
Household                2564740.17
Baking Goods             2452986.00
Canned                   2300082.71
Health and Hygiene       2163707.21
Meat                     2129211.94
Soft Drinks              1797044.72
Breads                    714942.24
Hard Drinks               625814.62
Others                    541496.30
Starchy Foods             518774.45
Breakfast                 362130.41
Seafood                   272404.04
Name: Product_Store_Sales_Total, dtype: float64
```

In [13]:
```
# Visualizing the contribution of each product type to revenue
product_revenue.plot(kind='bar', title='Revenue Contribution by Product Type')
```

```
plt.ylabel('Total Revenue')
plt.show()
```

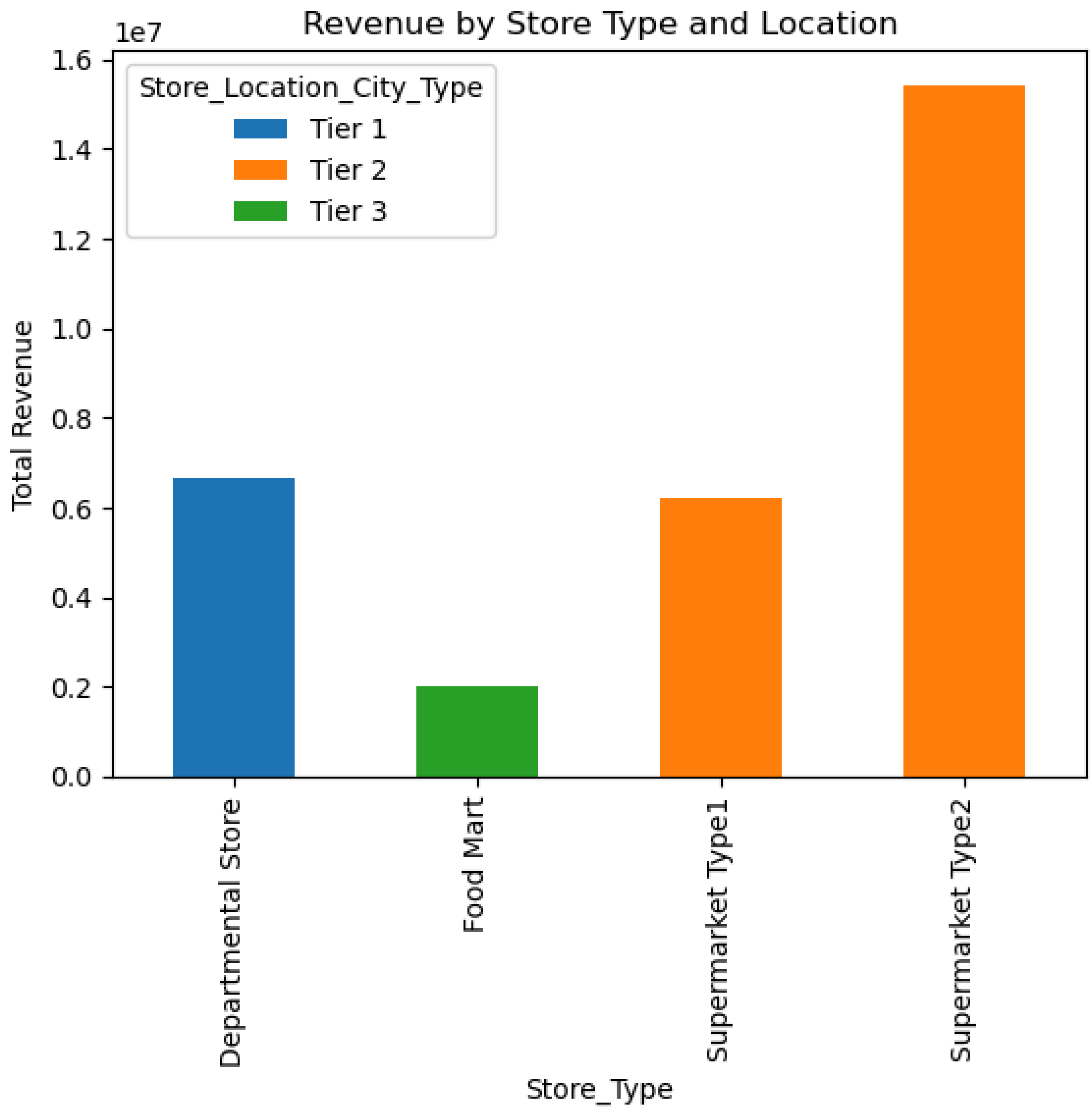


Fruti & Vegetables seems to be the product or segment with the highest volume of Revenue or Total Revenue

In [14]:
```
# Grouping by Store_Type and Store_Location_City_Type and summing up the Product_Store_Sales_Total
store_revenue = df.groupby(['Store_Type', 'Store_Location_City_Type'])['Product_Store_Sales_Total'].sum().s
print(store_revenue)

# Visualizing the revenue impact by store type and location
store_revenue.unstack().plot(kind='bar', stacked=True, title='Revenue by Store Type and Location')
plt.ylabel('Total Revenue')
plt.show()
```

```
Store_Type           Store_Location_City_Type
Supermarket Type2    Tier 2                      15427583.43
Departmental Store   Tier 1                       6673457.57
Supermarket Type1    Tier 2                       6223113.18
Food Mart            Tier 3                       2030909.72
Name: Product_Store_Sales_Total, dtype: float64
```



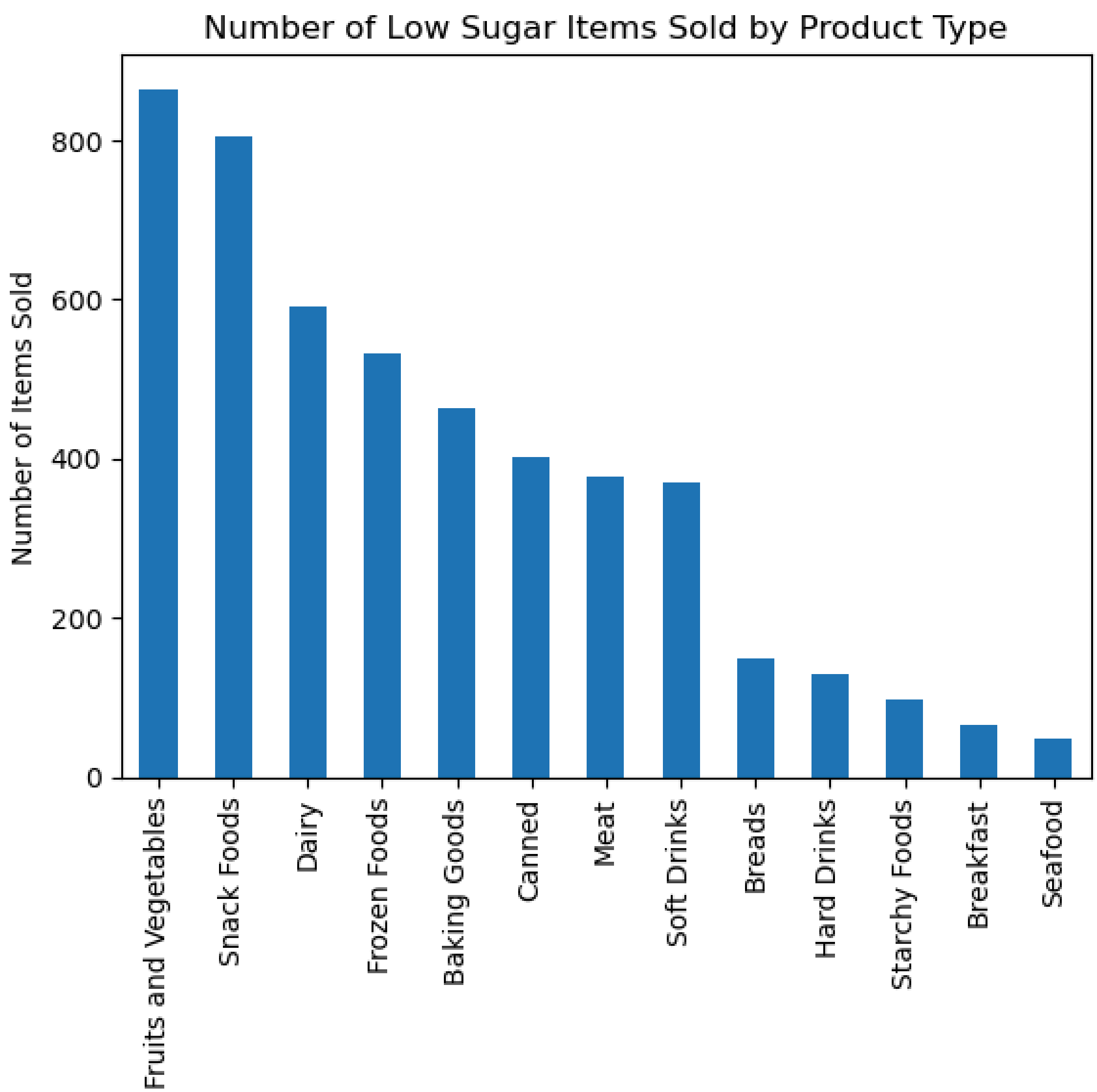
Supermarkets Type1 in Tier2 cities generate the highest revenues. Department Stores in Tier 1 and Food Marts in Tier 3 generate the least revenue. It seems like the stores location and type significantly impacts the reveue amount.

In [15]:
```
# Filtering products with low sugar content
low_sugar_products = df[df['Product_Sugar_Content'] == 'Low Sugar']

# Counting the number of items sold for each of the 16 product types with low sugar
low_sugar_sales = low_sugar_products['Product_Type'].value_counts()
print(low_sugar_sales)
```

```
Fruits and Vegetables    864
Snack Foods              804
Dairy                    590
Frozen Foods             531
Baking Goods             462
Canned                   402
Meat                     377
Soft Drinks              370
Breads                   148
Hard Drinks              128
Starchy Foods             97
Breakfast                 65
Seafood                   47
Name: Product_Type, dtype: int64
```

In [16]:
```
# Visualizing the number of low sugar items sold by product type
low_sugar_sales.plot(kind='bar', title='Number of Low Sugar Items Sold by Product Type')
plt.ylabel('Number of Items Sold')
plt.show()
```

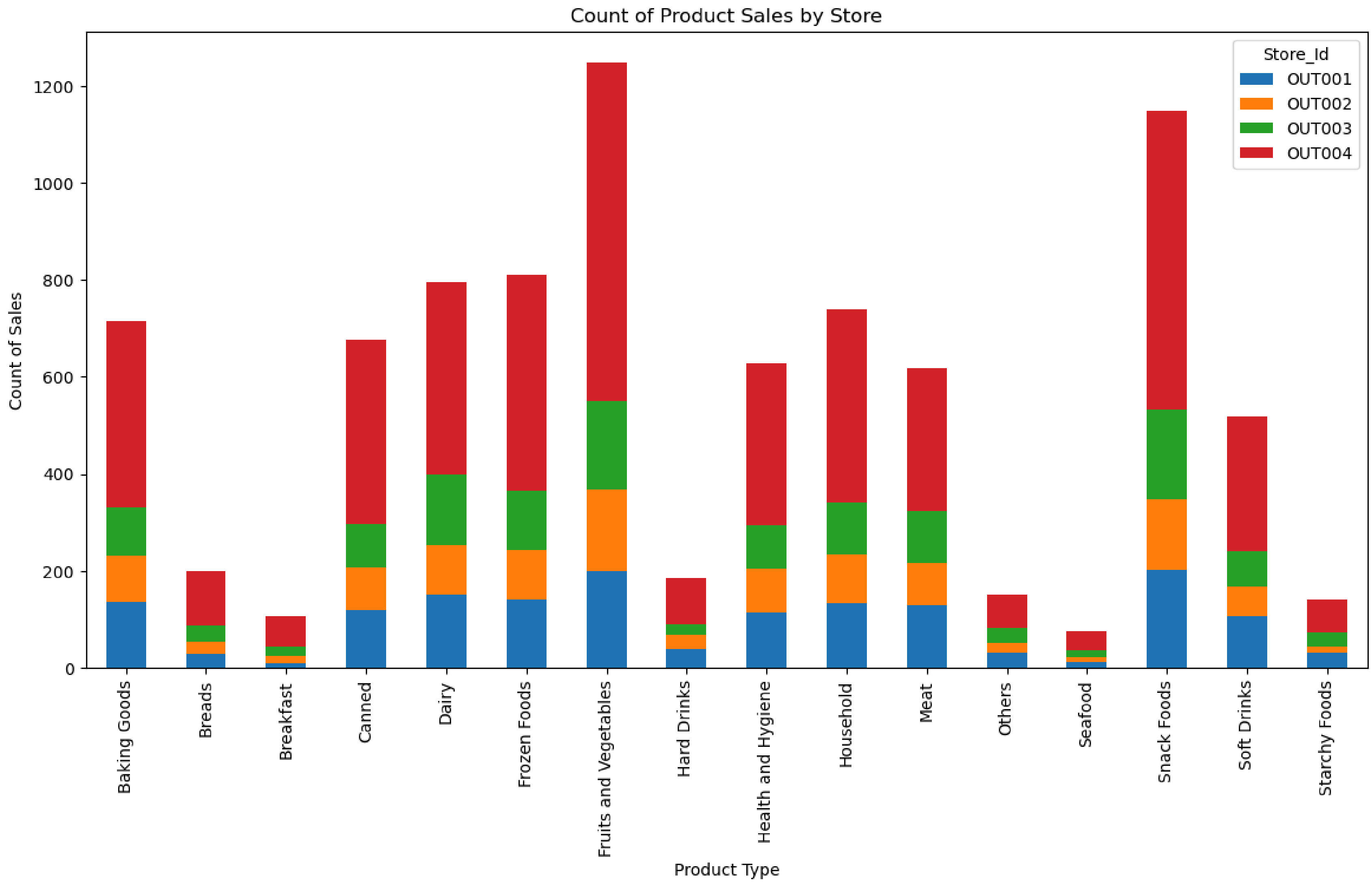


In [17]:
```
# Grouping by Store_Id and Product_Type and counting the occurrences
product_sales_count = df.groupby(['Store_Id', 'Product_Type'])['Product_Id'].count().reset_index().sort_va

# Pivot table to prepare data for a stacked bar chart for count of sales
sales_count_pivot = product_sales_count.pivot(index='Product_Type', columns='Store_Id', values='Product_Id

# Visualizing the count of sales by Store and Product Type
```

```
sales_count_pivot.plot(kind='bar', stacked=True, figsize=(14, 7), title='Count of Product Sales by Store')
plt.ylabel('Count of Sales')
plt.xlabel('Product Type')
plt.show()
```

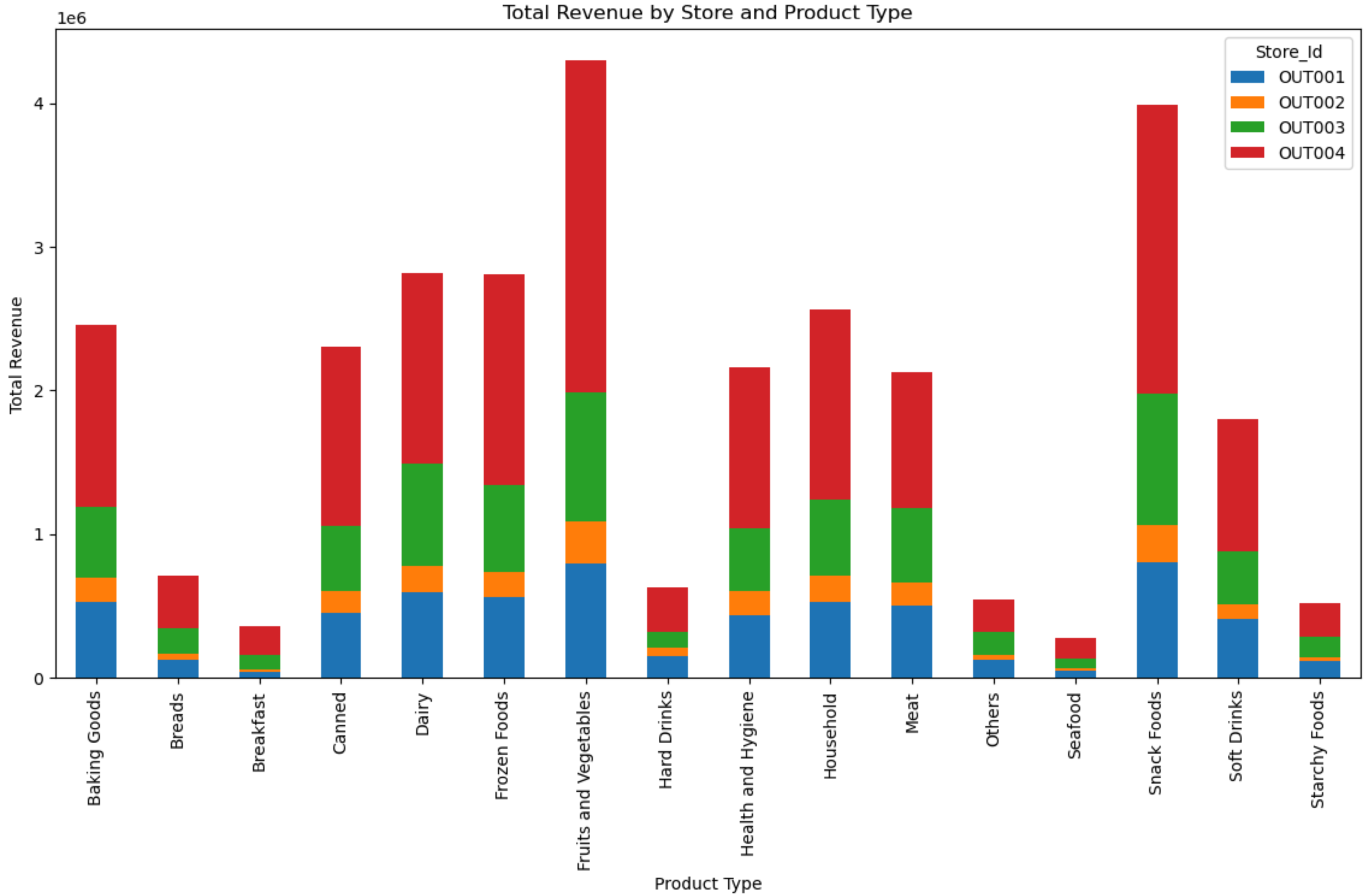


In [ ]:

In [18]:
```
# Grouping by Store_Id and Product_Type and summing up the revenue
product_sales_revenue = df.groupby(['Store_Id', 'Product_Type'])['Product_Store_Sales_Total'].sum().reset_i

# Pivot table to prepare data for a stacked bar chart for revenue
sales_revenue_pivot = product_sales_revenue.pivot(index='Product_Type', columns='Store_Id', values='Produc

# Visualizing the revenue by Store and Product Type
sales_revenue_pivot.plot(kind='bar', stacked=True, figsize=(14, 7), title='Total Revenue by Store and Produ
plt.ylabel('Total Revenue')
plt.xlabel('Product Type')
plt.show()
```

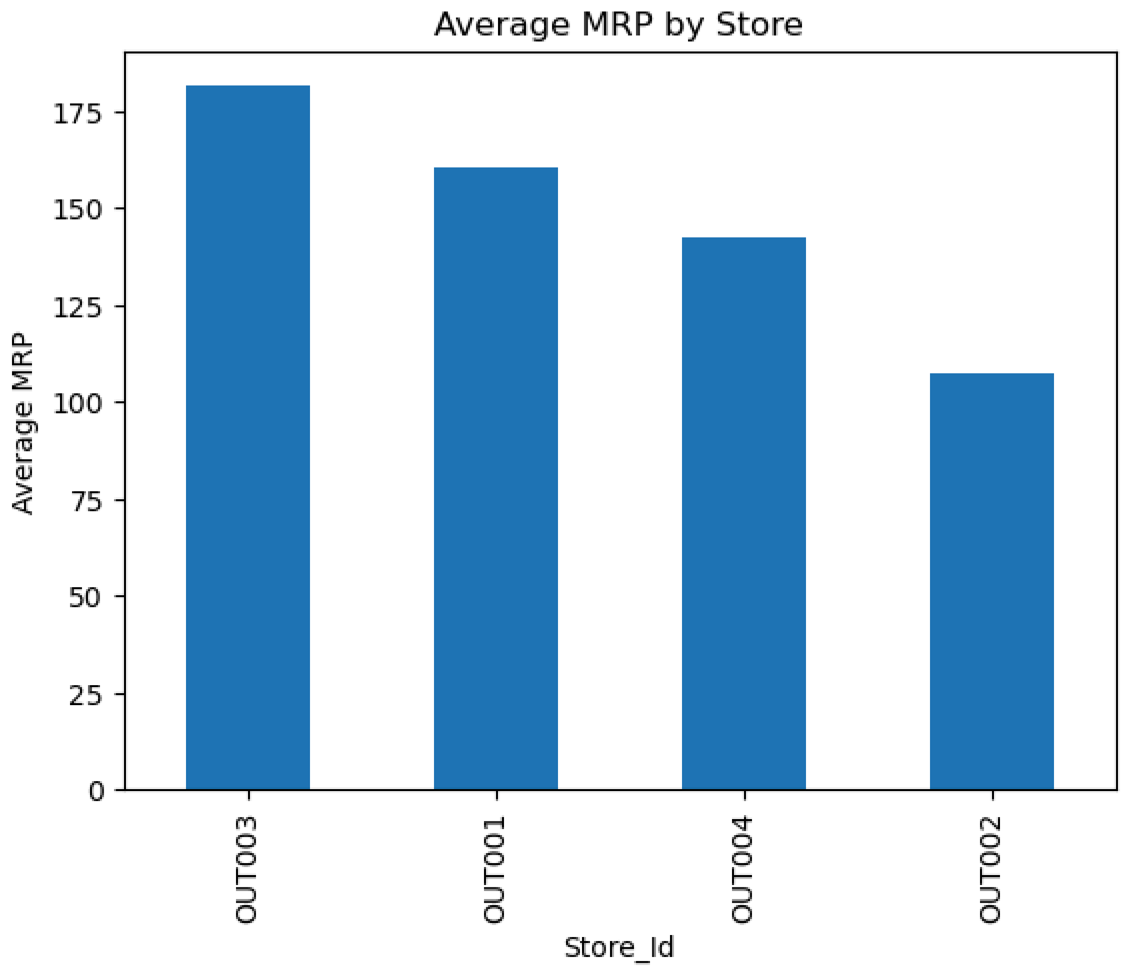


Store with ID OUT004 has the highest volume of sales.

In [19]:
```
# Grouping by Store_Id and calculating the average MRP of products sold
average_mrp_by_store = df.groupby('Store_Id')['Product_MRP'].mean().sort_values(ascending=False)
print(average_mrp_by_store)
```

```
Store_Id
OUT003    181.358725
OUT001    160.514054
OUT004    142.399709
OUT002    107.080634
Name: Product_MRP, dtype: float64
```

In [20]:
```
# Visualizing which store sells more costly goods on average
average_mrp_by_store.plot(kind='bar', title='Average MRP by Store')
plt.ylabel('Average MRP')
plt.show()
```



Store ID OUT003 has the highest avg MRP.

Data Preprocessing

(Please see everything below this markdown cell)

1. Missing Value Treatment (not needed)

In [21]:
```
# Check for missing values
missing_values = df.isnull().sum()

# missing values treatment ( replacement w. mean)
df['Product_Weight'].fillna(df['Product_Weight'].mean(), inplace=True)
```

2. Outlierr Detection and treatment

In [22]:
```
# Using IQR for outlier detection on the 'Product_Store_Sales_Total' column
Q1 = df['Product_Store_Sales_Total'].quantile(0.25)
Q3 = df['Product_Store_Sales_Total'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Removing outliers
df = df[(df['Product_Store_Sales_Total'] >= lower_bound) & (df['Product_Store_Sales_Total'] <= upper_bound
```

Preparing Data for Modeling

In [23]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Defining the columns that need encoding and scaling
categorical_cols = ['Product_Sugar_Content', 'Product_Type', 'Store_Id', 'Store_Size', 'Store_Location_City
numerical_cols = ['Product_Weight', 'Product_Allocated_Area', 'Product_MRP', 'Store_Establishment_Year']
```

```
# Creating a ColumnTransformer to apply the transformations
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Removing the target variable and identifier columns for features
X = df.drop(['Product_Store_Sales_Total', 'Product_Id'], axis=1)
y = df['Product_Store_Sales_Total']

# Apply the transformations to prepare the data
X_preprocessed = preprocessor.fit_transform(X)
```

Splitting the Data into test set and training

In [24]:
```
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)
```

Feature Engineering

A store which has been in the business for a long duration is more trustworthy than the newly established ones. On the other hand, older stores may sometimes lack infrastructure if proper attention is not given. So let us calculate the current age of the store and incorporate that in our model.

In [25]:
```
import datetime

# Assuming the current year for the analysis is 2024
current_year = datetime.datetime.now().year

# Calculate the age of the store
df['Store_Age'] = current_year - df['Store_Establishment_Year'] # Now, 'Store_Age' is a new feature
```

In [26]:
```
print(df.columns)
```

```
Index(['Product_Id', 'Product_Weight', 'Product_Sugar_Content',
       'Product_Allocated_Area', 'Product_Type', 'Product_MRP', 'Store_Id',
       'Store_Establishment_Year', 'Store_Size', 'Store_Location_City_Type',
       'Store_Type', 'Product_Store_Sales_Total', 'Store_Age'],
      dtype='object')
```

In [27]:
```
# include 'Store_Establishment_Year'
numerical_cols_updated = numerical_cols + ['Store_Establishment_Year']

# Update the categorical columns
categorical_cols_updated = categorical_cols # Add or remove columns as necessary

# Redefine the ColumnTransformer with the updated numerical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols_updated),
        ('cat', OneHotEncoder(), categorical_cols_updated)
    ])

# Applying the transformations to prepare the data
X_preprocessed = preprocessor.fit_transform(df.drop(['Product_Store_Sales_Total', 'Product_Id'], axis=1))

# Define the target variable
y = df['Product_Store_Sales_Total']

# Split the data again into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)
```

We have 16 different product types in our dataset. So let us make two broad categories, perishables and non perishables, in order to reduce the number of product types.

Perishable product types

In [28]:
```
# List of perishable product types
perishables = [
    "Dairy",
    "Meat",
```

```
    "Fruits and Vegetables",
    "Breakfast",
    "Breads",
    "Seafood",
]
```

In [29]:
```
def change(product_type):
    if product_type in perishables:
        return "Perishables"
    else:
        return "Non Perishables"

# Apply the function to create a new 'Product_Category' column
df['Product_Category'] = df['Product_Type'].apply(change)

# Drop the original 'Product_Type' column since it's replaced by 'Product_Category'
df = df.drop('Product_Type', axis=1)
```

In [30]:
```
# Defining the list of numerical and categorical columns for the ColumnTransformer
numerical_cols = ['Product_Weight', 'Product_Allocated_Area', 'Product_MRP', 'Store_Establishment_Year']
categorical_cols_updated = ['Product_Sugar_Content', 'Store_Id', 'Store_Size', 'Store_Location_City_Type',

# Redefinition the ColumnTransformer with the updated categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols_updated),
    ]
)
```

In [31]:
```
# Applying the transformations to prepare the data for modeling
X = df.drop(['Product_Store_Sales_Total', 'Product_Id'], axis=1)  # Ensure 'Product_Id' is excluded if it's
y = df['Product_Store_Sales_Total']
X_preprocessed = preprocessor.fit_transform(X)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)
```

Bagging and boosting models

In [32]:
```
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor  # Import from sklearn.tree instead of sklearn.ensemble
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
```

In [33]:
```
# Models initiliazation
dt_regressor = DecisionTreeRegressor(random_state=42)
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
ada_regressor = AdaBoostRegressor(n_estimators=100, random_state=42)
```

Fit the models

In [34]:
```
# Fit the models on the training data
dt_regressor.fit(X_train, y_train)
rf_regressor.fit(X_train, y_train)
ada_regressor.fit(X_train, y_train)
```

Out[34]:
```
AdaBoostRegressor(n_estimators=100, random_state=42)
```

Predictions:

In [35]:
```
# Predict on the test set
dt_pred = dt_regressor.predict(X_test)
rf_pred = rf_regressor.predict(X_test)
ada_pred = ada_regressor.predict(X_test)
```

Evaluation

In [63]:
```
# Evaluating the models using mean squared error and R-squared score
dt_mse = mean_squared_error(y_test, dt_pred)
rf_mse = mean_squared_error(y_test, rf_pred)
ada_mse = mean_squared_error(y_test, ada_pred)
```

```
dt_r2 = r2_score(y_test, dt_pred)
rf_r2 = r2_score(y_test, rf_pred)
ada_r2 = r2_score(y_test, ada_pred)

# Output the performance metrics
print(f'Decision Tree MSE: {dt_mse}, R^2: {dt_r2}')
print(f'Random Forest MSE: {rf_mse}, R^2: {rf_r2}')
print(f'AdaBoost MSE: {ada_mse}, R^2: {ada_r2}')
```

```
Decision Tree MSE: 143984.76440942741, R^2: 0.8614240842133056
Random Forest MSE: 81547.40300089365, R^2: 0.9215159597112514
AdaBoost MSE: 183685.18961020673, R^2: 0.8232150222900918
```

These results indicate that the Random Forest Regressor is performing the best out of the three models

Feature Importance Analysis

In [61]:

```
feature_importances = rf_regressor.feature_importances_  # After fitting the RandomForest model, we can che
features = X.columns

# Mapping feature names with their importances
feature_importance_dict = dict(zip(features, feature_importances))

# Sorting features by importance
sorted_features = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

# Displaying the features and their importances
print("Feature Importances:")
for feature, importance in sorted_features:
    print(f"{feature}: {importance}")
```

```
Feature Importances:
Product_Allocated_Area: 0.5177961372714935
Product_Weight: 0.10714171016015298
Store_Age: 0.05541297789830709
Product_MRP: 0.038710969187635676
Product_Sugar_Content: 0.009622707752448915
Product_Category: 0.007554053990838484
Store_Establishment_Year: 0.0011496195768125027
Store_Id: 0.0009428845996094331
Store_Size: 0.000806760905554254
Store_Type: 0.0007988907840543261
Store_Location_City_Type: 0.0003419070833981098
```

Product_Allocated_Area is by far the most influential feature in predicting the outcome, with a significance of over 51%. Product_Weight and Store_Age also show notable importance but to a lesser extent.

Cross Validation

In [45]:
```python
from sklearn.model_selection import cross_val_score

# Example with RandomForestRegressor
rf_scores = cross_val_score(rf_regressor, X_train, y_train, cv=5, scoring='neg_mean_squared_error')

# Converting scores to positive
rf_mse_scores = -rf_scores

# Calculating RMSE for each fold
rf_rmse_scores = np.sqrt(rf_mse_scores)

# Displaying results
print("Random Forest cross-validation RMSE scores:", rf_rmse_scores)
print("Mean:", rf_rmse_scores.mean())
print("Standard deviation:", rf_rmse_scores.std())
```

```
Random Forest cross-validation RMSE scores: [274.96852643 273.34153188 229.19881372 239.87157725 307.769231
21]
Mean: 265.02993609837677
Standard deviation: 27.968975309458763
```

Decision Tree

In [58]:
```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score

# Initialize the Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(random_state=42)

# Fit the model on the training data
dt_regressor.fit(X_train, y_train)

# Predict on the test set
dt_pred = dt_regressor.predict(X_test)
```

In [59]:
```
# Mean squared error and R-squared for the Decision Tree model
dt_mse = mean_squared_error(y_test, dt_pred)
dt_r2 = dt_regressor.score(X_test, y_test)

print(f"Decision Tree MSE: {dt_mse}")
print(f"Decision Tree R^2: {dt_r2}")
```

```
Decision Tree MSE: 143984.76440942741
Decision Tree R^2: 0.8614240842133056
```

In [60]:
```
# Perform cross-validation to evaluate the model
dt_cv_scores = cross_val_score(dt_regressor, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
dt_cv_rmse = np.sqrt(-dt_cv_scores)

print(f"Decision Tree cross-validation RMSE scores: {dt_cv_rmse}")
print(f"Mean: {dt_cv_rmse.mean()}")
print(f"Standard deviation: {dt_cv_rmse.std()}")
```

```
Decision Tree cross-validation RMSE scores: [386.44961866 363.35712429 339.81919577 326.34814343 401.396050
76]
Mean: 363.47402658063146
Standard deviation: 27.94227491029711
```

The decision tree model has higher error rates and variability across folds compared to the Random Forest model.

Will tuning the hyperparameters improve the model performance?

Yes. We should try to improve the Random Forest model

Hyperparameter tunning - Random Forests

In [37]:
```
# Parameter grid setup for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search_rf = GridSearchCV(estimator=RandomForestRegressor(random_state=42), param_grid=param_grid, cv=5
```

In [38]:
```
# Run the grid search
grid_search_rf.fit(X_train, y_train)
```

Out[38]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')
```

In [39]:
```
# Getting the best estimator and its predictions
best_rf = grid_search_rf.best_estimator_
best_rf_pred = best_rf.predict(X_test)

# Evaluating the best model from grid search
best_rf_mse = mean_squared_error(y_test, best_rf_pred)
best_rf_r2 = r2_score(y_test, best_rf_pred)

# Output the performance metrics for the best model from grid search
print(f'Best Random Forest MSE: {best_rf_mse}, R^2: {best_rf_r2}')
```

```
Best Random Forest MSE: 82268.96317562796, R^2: 0.9208215052499121
```

Model Performance Comparison and Conclusions

In [40]:
```python
model_metrics = {
    'Decision Tree': {'MSE': dt_mse, 'R2': dt_r2},
    'Random Forest': {'MSE': rf_mse, 'R2': rf_r2},
    'AdaBoost': {'MSE': ada_mse, 'R2': ada_r2},
    'Best Random Forest (GridSearch)': {'MSE': best_rf_mse, 'R2': best_rf_r2}
}
```

In [41]:
```python
# Convert the dictionary to a DataFrame for a nicer display
metrics_df = pd.DataFrame(model_metrics).T  # .T is for transpose so that we get models as rows

# Display the DataFrame
print(metrics_df)
```

```
                                         MSE        R2
Decision Tree                    143984.764409  0.861424
Random Forest                     81547.403001  0.921516
AdaBoost                         183685.189610  0.823215
Best Random Forest (GridSearch)   82268.963176  0.920822
```

In [42]:
```python
# Compare the MSE and R^2 of each model
sorted_metrics_df = metrics_df.sort_values(by='R2', ascending=False)

# Display the sorted DataFrame
print(sorted_metrics_df)
```

```
                                         MSE        R2
Random Forest                     81547.403001  0.921516
Best Random Forest (GridSearch)   82268.963176  0.920822
Decision Tree                    143984.764409  0.861424
AdaBoost                         183685.189610  0.823215
```

The Decision Tree model shows higher MSE and lower R^2 compared to both the original and the best tuned Random Forest models. The Random Forest models (both original and tuned) perform better, indicating better generalization and predictive power.

The AdaBoost model has the highest MSE and lowest R^2, showing it may not be as effective for this specific dataset.

The Random Forest model, particularly the one optimized through GridSearchCV, has shown the best performance with an R^2 score of approximately 0.92. This model should be utilized for future sales predictions as it is likely to provide the most accurate results.

********** Actionable Insights and Recommendations **********

Conclusions:

1. The Random Forest model without GridSearch has the highest R^2 value of approximately 0.9215, indicating it explains about 92.15% of the variance in the sales data. This suggests it is the most accurate model for predicting future sales revenue.
2. The Best Random Forest model, optimized using GridSearch, has a slightly lower R^2 value of approximately 0.9208 but still performs very well.
3. The Decision Tree and AdaBoost models have lower R^2 values, indicating less predictive accuracy compared to the Random Forest models.

My Recommendations

1. Utilize the Random Forest model to forecast sales for different outlets. Adjust inventory levels based on predicted sales to ensure adequate stock and reduce overstocking or stockouts.
2. Operational strategy:. For certain product types that are leading sales in Tier 1 cities, SuperKart may consider stocking a wider variety of these products in those areas.
3. Resources: Allocate marketing and operational resources more effectively based on our model's predictions. Outlets expected to perform better could receive additional marketing campaigns to boost sales further, while outlets with lower predicted sales might be evaluated for operational improvements.

In [ ]:

In [ ]: