BMAN73701: COURSEWORK
Group 24
[Student ID: 11360204, 11361453, 11349153, 11356949]

Abstract

The retail industry has undergone substantial growth through digitalisation lately. However, navigating the current market scenario requires retailers to figure out complex challenges related to technology, sustainability, competition, and changing consumer expectations. This report focuses on a South America-based firm known as ABC, operating within the retail sector. The company is facing issues in handling the inventory levels of products to meet customer demands due to varied factors. For this, a machine learning approach has been designed to predict optimal levels of inventory required, in order to keep a subtle balance between demand and wastage. Based on exploratory analysis, historical data from July 2015 has been utilised for employing various supervised machine learning algorithms with cross validation and hyperparameter tuning. Of the 33 product classes, diverse sales trends were observed among them with minimal similarity, leading to the decision of creating individual models for each product. XGBoost Regressor showed a remarkable performance with cross validation with the lowest neg_root_mean_squared_error score of -46 amongst all other models and therefore was chosen as the final model. This report also discusses the challenges and suggestions for efficient management of inventory.

Introduction

Adapting to the modern trends is important for sustainability in a continuously developing retail environment which has evolved a lot from brick-and-mortar stores to present day online platforms. Globalisation has played a vital role in this major shift. Amidst growing competition and rising demand, retailers are bound to expand their business to new locations and introduce new products enabling them to enter new sectors and cater to global demand rather than serving locally. This in turn, might also help them in increased revenue generation. There is no specific solution to stay consistent amidst these changing dynamics however taking care of some factors in business might help the firms in the long run.

South America's prominent grocery retailer firm ABC, being familiar with the intricacies of changing consumer demands, wants to take this step and ensure the consistency of sales and product availability at all times. To achieve this, precise prediction of inventory is a major task as over prediction can lead to wastage while under prediction can lead to products being out of stock and eventually customer dissatisfaction.

They have a large dataset encompassing essential sales data for 33 distinct product types across 54 stores, spanning from 01/01/2013 to 15/08/2017 and wish to utilise it to identify customer purchase patterns and forecast sales using machine learning to make future decisions. Currently, they use subjective forecasting with limited automation techniques to achieve this without efficiently utilising the abundant historical data available.

The specific description of the dataset is as follows:

1) id: unique identifier of a record
2) date: the date of a record.
3) store_nbr: the store number.
4) product_type: the type of the product.
5) sales: the total sales for a specific product type at a specified store.
6) special_offer: the promotion effort, where a higher number signifies a more intense promotional effort.

id	date	store_nbr	product_type	sales	special_offer
0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	2013-01-01	1	BABY CARE	0.0	0
2	2013-01-01	1	BEAUTY	0.0	0
3	2013-01-01	1	BEVERAGES	0.0	0
4	2013-01-01	1	BOOKS	0.0	0

Table 1. First few records of the sales dataset.

Table 1 shows the first five records of the dataset.

A few observations by looking at the dataset and applying basic summary statistics are:

1) There are not many inconsistencies in the data apart from the fact that the firm remains closed on 25th December, Christmas every year. There are entries for all products and store combinations even if the products are not sold or are inactive with sales value as 0.
2) Since ABC is a grocery retailer, it makes sense that most of the products would be low quantity selling products which is also reflected in the data.
3) More than 75% of the products are sold without any promotional offers.

Exploratory Data Analysis

Having observed the basic characteristics of the dataset, we now delve into the distribution of sales and explore key patterns in the variables. We did some basic feature engineering to extract columns such as day, month, year, day of week in order to analyse trends on different frequencies of time.

In this section, we have leveraged a variety of visualisations to unravel patterns of sales across different dimensions – analysing overall trajectories, examining individual store performances, and delving into sales trends for each product.

We start by plotting sales for individual stores over time to look for any underlying patterns.

1. **Sales Pattern of Stores over time** *(Figure 1)*
    - For an overview, the graphs for a few stores have been included in the report.
    - The foremost observation that comes out is a general trend of increase in sales year on year for most of the stores which suggests an overall growth.
    - Overall, similar sales patterns were observed in groups of stores indicating shared dynamics and potentially suggesting a reason for store clustering based on sales.
    - Notably, Store 52 is the latest launched store (2017) since there is continuous zero sales before that. This will be difficult to predict since the dataset for training would be very small for this outlet-product combination.
    - Store 20, 21, 22, 29, 42, 29 commenced their operations from 2014 as the sales in the year was 0 for the whole of 2013. This also suggests that a number of new stores were launched in 2014 and there were business decisions being taken causing major changes in sales patterns.
    - Considering customised forecasting techniques or adequate preprocessing of data for stores with different opening dates can contribute to better predictions. Especially, for the case of recently opened stores.

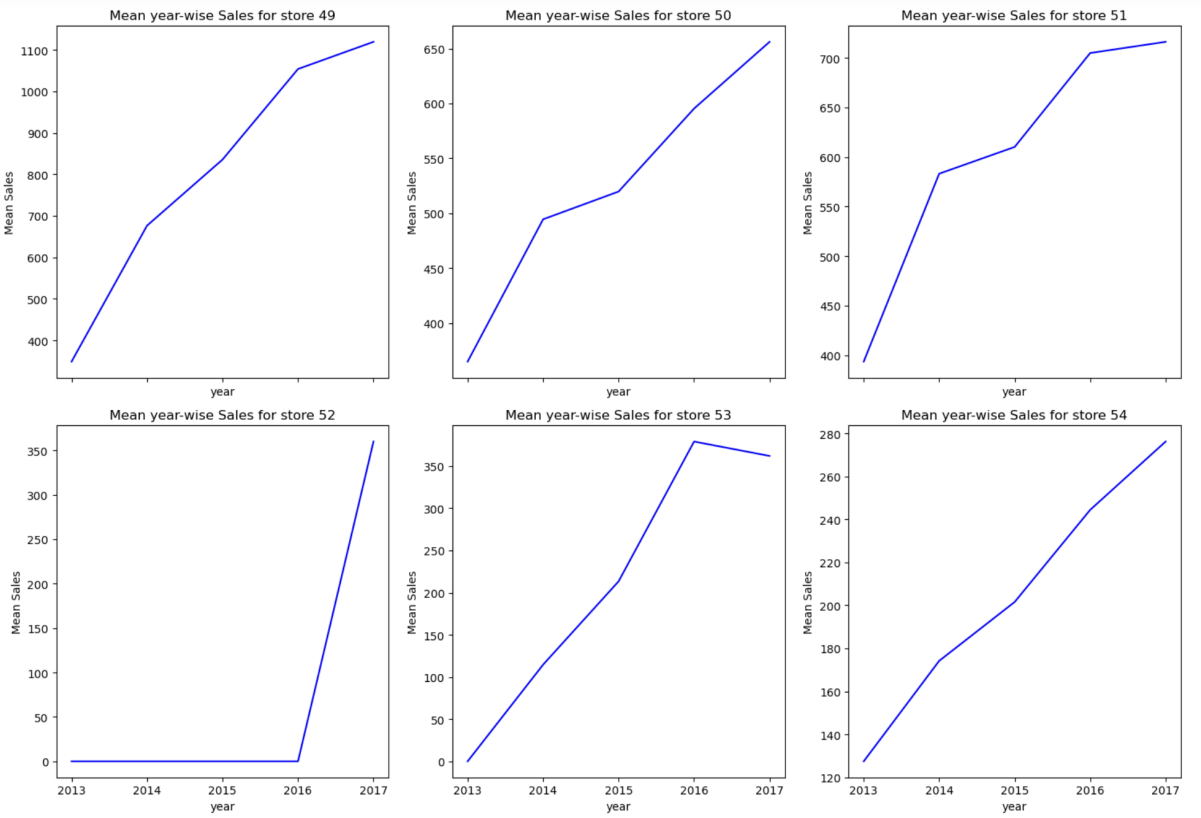


Figure 1. Yearly Mean sales over time for different stores

2. **Sales Pattern of Products over time** *(Figure 2)*
    - For an overview, the graphs for a few products have been included in the report.
    - Distinct sales patterns emerge for most products suggesting unique sales trends across the entire product catalogue. Tailored predictions for each product would be an efficient method. This aligns well with our aim of inventory management as with tailored predictions for each product, prediction could be even more precise.
    - A number of products sell in lower quantities such as Automotive, Beauty, Home Appliances, Home and Kitchen while higher selling products are daily usage product types such as grocery and dairy. The reason for low quantity sales in such products could be that these products are usually bought once by any customer and used for a long time. While the high selling products are mainly daily consumption or usage products which need restocking every now and then.
    - Another thing to note is that a few products began selling later than other products. Specifically, Books had zero sales up until 2014 i.e. new products launched.
    - A column start_date indicating the launch date of each product at each store was added to the dataset.

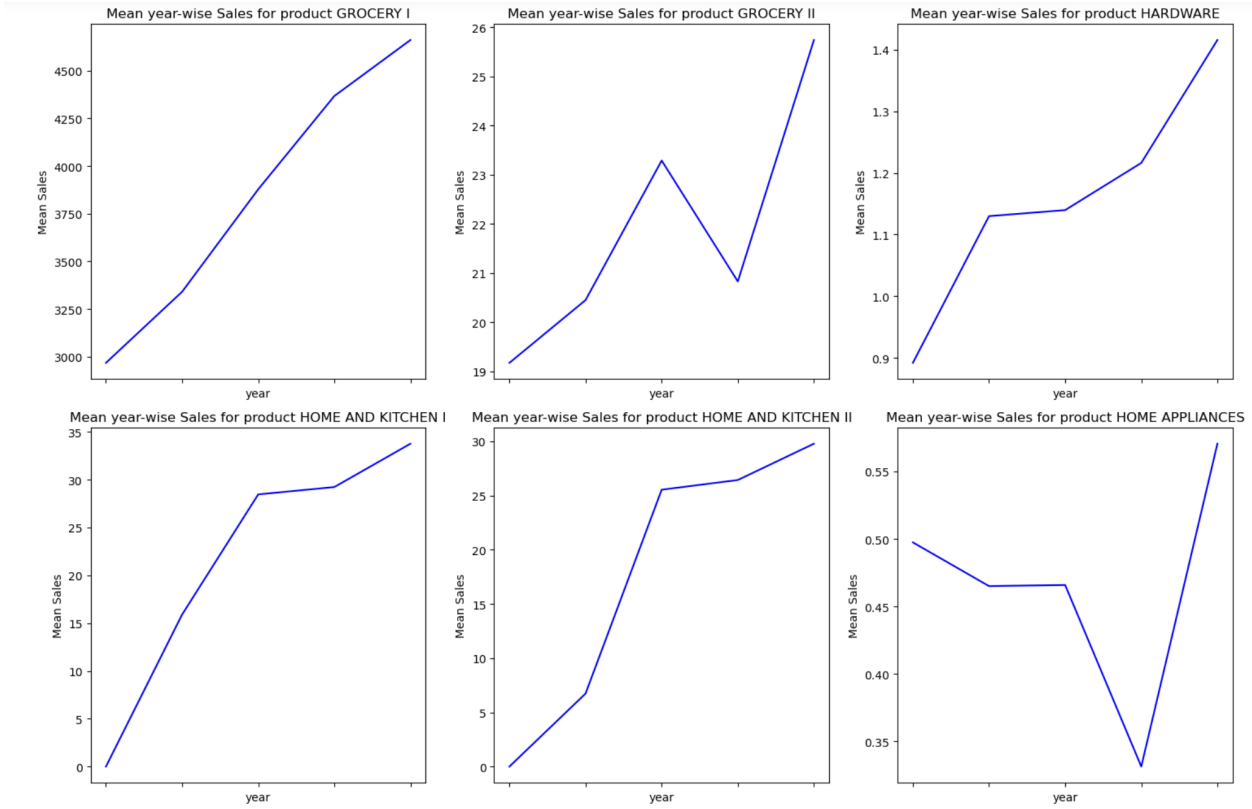


Figure 2. Yearly Mean sales over time for different products

3. **Overall Sales Patterns for each DOW over time + Anomaly Detection** *(Figure 3)*
    - An overall consistent year on year growth in sales is observed with comparatively lower sales in the year 2013 and high instability and considerable fluctuations in 2014 specifically. This could indicate a launching phase as we observed above that a number of stores and products were launched in the year 2014 potentially causing the deviations. The first half of 2015 observed a sudden dip (the only degrowth period) in sales which could be potentially attributed to some external influences affecting consumer purchasing patterns or market fluctuations. Post this, sales became quite stable suggesting a period of increased predictability of customer behaviour.
    - With the above information, we decided to trim our dataset and further explore the dataset post July 2015 as it would make more sense to further analyse trends on a stable dataset to conclude on patterns and eventual features for the predictive model.
    - Recognizing day-wise trends, weekends always marked the highest sales and Thursdays specifically emerged out as the lowest selling day of week.
    - High sales were observed during the start of the month, could be due to attainment of salaries.
    - Overall, highest sales were observed in the month of December followed by marginally low sales in January.

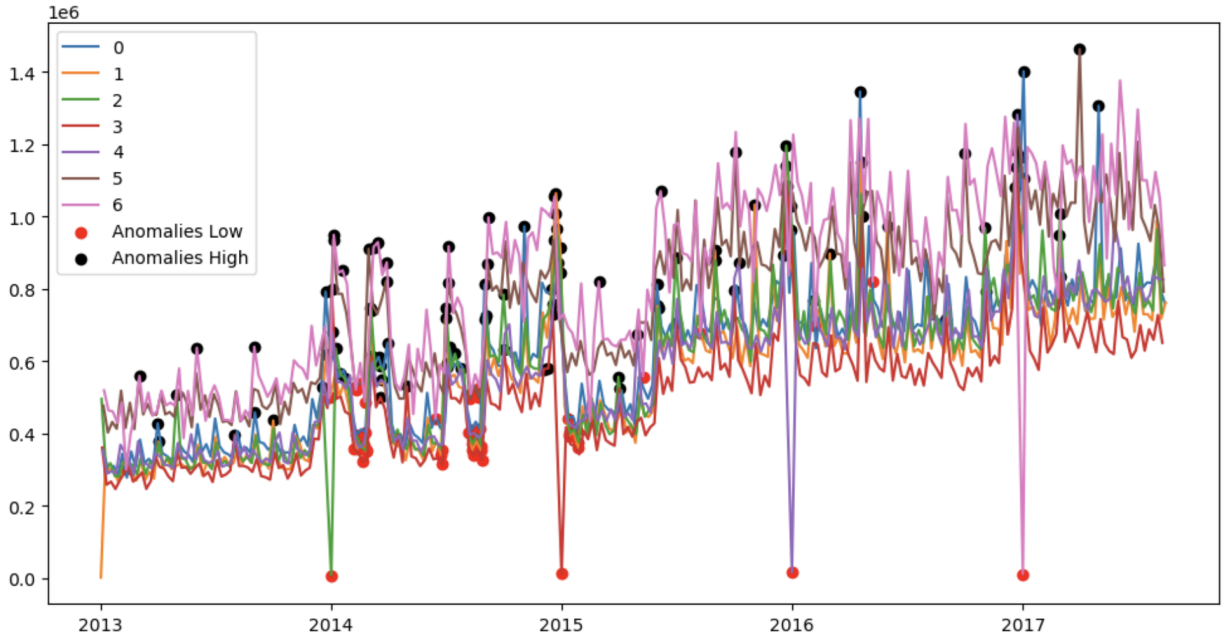


Figure 3. DOW-wise Overall Sales pattern over time

4. **Lag Features** *(Figure 4)*
    - Our dataset clearly follows a seasonal trend with repetition of trends at different frequency intervals which suggests that values at a certain point of time are related to the values at previous times. Lag features help the model understand and incorporate recent patterns and trends, making it better equipped to make accurate predictions.
    - As can be seen from *Figure 4*, lag feature lag_yoy_sales is somehow following the current actual values trend.
    - To include this feature, a feature called lag_yoy_sales representing the DOW's average from the previous year

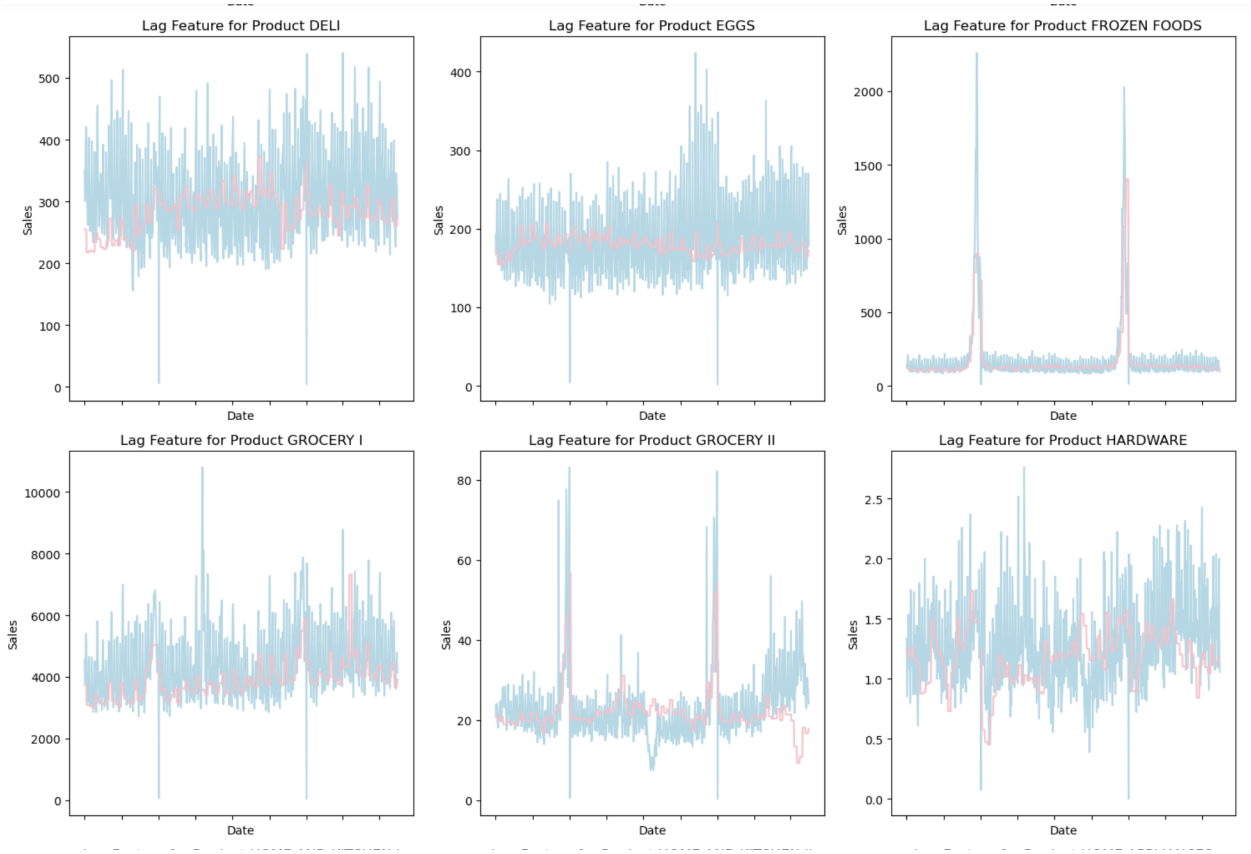


Figure 4. Understanding Lag Feature Trends

5. **Sales vs Special Offer**

   <u>Monthly Trend</u> *(Figure 5)*

   - A modest correlation is found between sales and special offers over months suggesting the impact of promotional offers on sales.
   - Peak offers are released around May and December suggesting festive offers for Easter and Christmas. Interestingly, even though the highest offers are released in May, December stands out with highest purchasing activity.

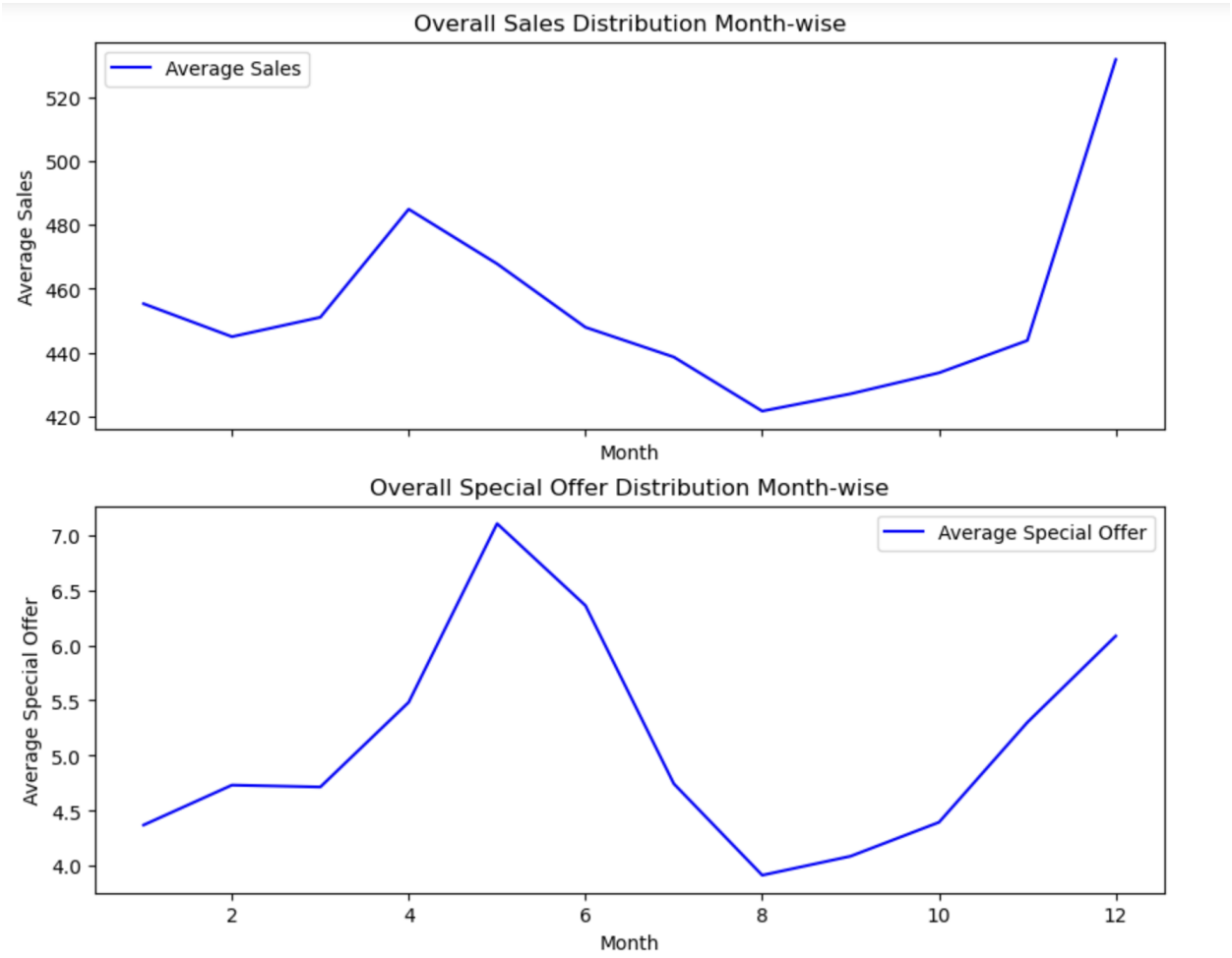


Figure 5. Monthly trends of average sales and special offers

DOW Trend *(Figure 6)*

- A highly consistent pattern for sales as well as promotions is observed each year with respect to DOW.
- Even though the offers on Wednesdays are the highest, sales are at its peak on weekdays suggesting the need for re-tailoring promotional strategies.
- Overall DOW exhibits a repetitive pattern for sales as well as special offers which can be an important feature for predictions.

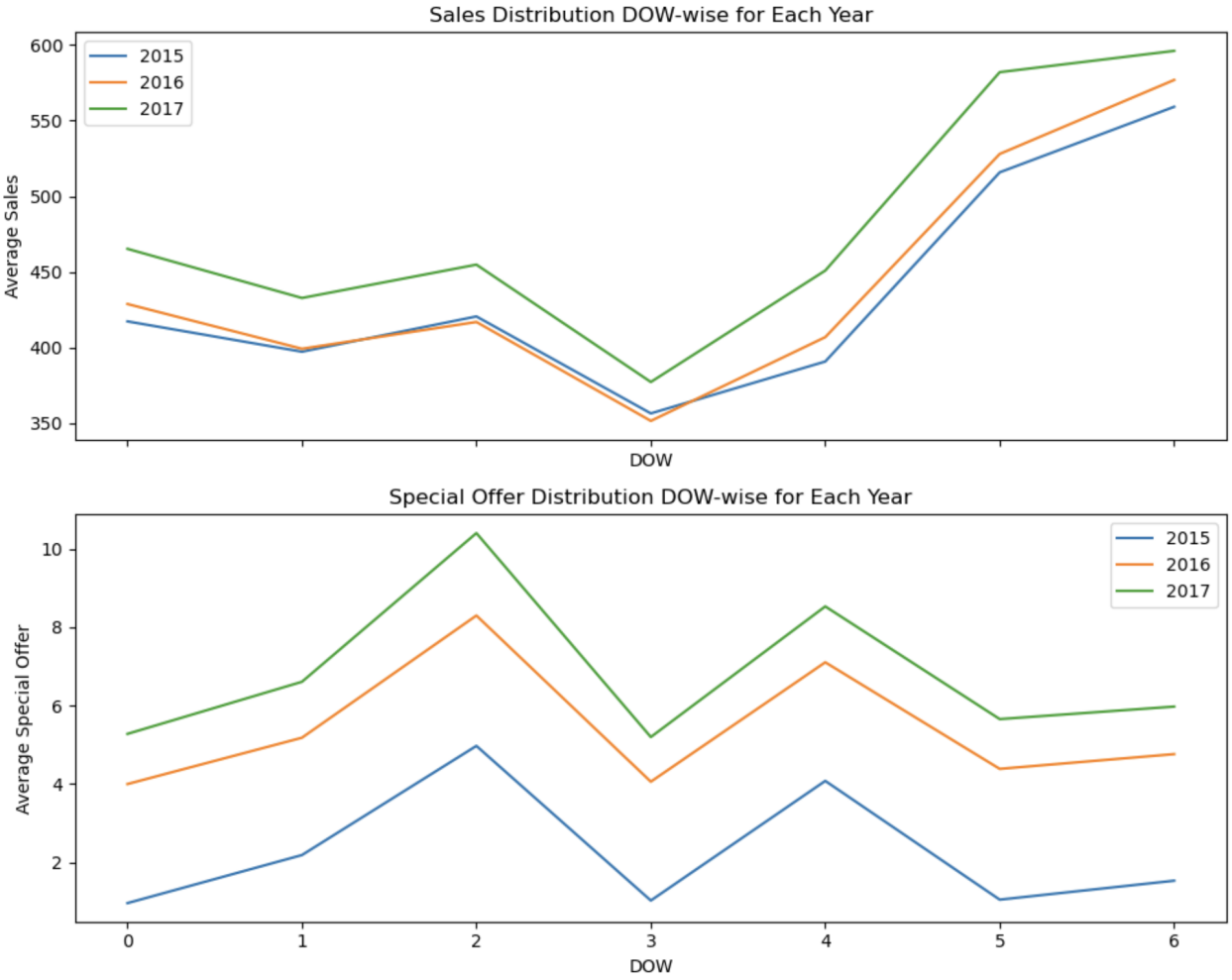


Figure 6. DOW wise trends of average sales and special offers for each year

With a broader understanding of sales patterns over time, we move to a more granular exploration on each product.

Product Specific Patterns + Clustering Possibility

1. **Sales Distribution Clusters**
    - For an overview, the graphs for a few products have been included in the report.
    - From initial exploration, it was observed that distinct products exhibited different sales ranges and patterns amongst them due to which we tried a clustering approach to categorise products into similar sales groups.
    - Three Clusters with values as 1 (Low), 2 (Medium) and 3 (High) were formed to represent their monthly average sales segmentation for each product as shown in *Figure 7*.
    - Similarly, a daily average sales cluster was also formed representing daily average sales groups for each product as shown in *Figure 8*.

-

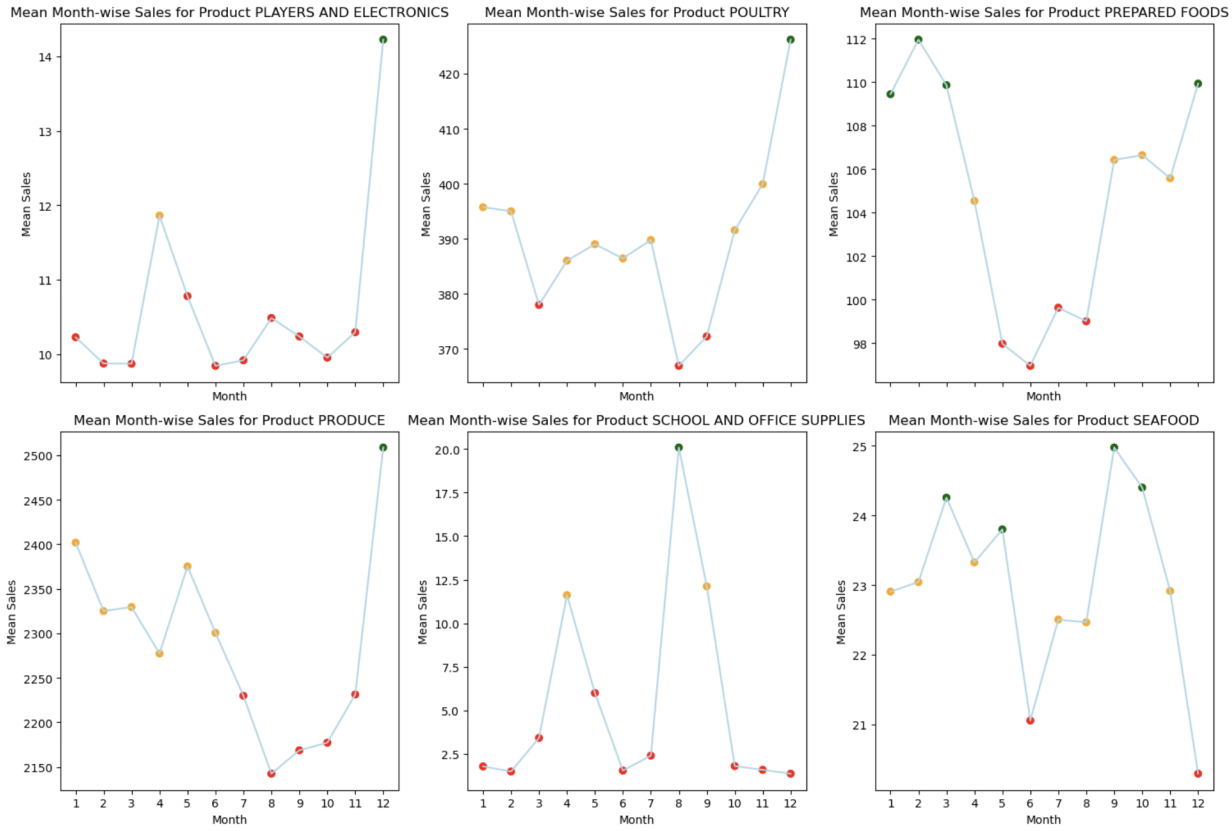


Figure 7. Monthly Average Sales clustering for each product

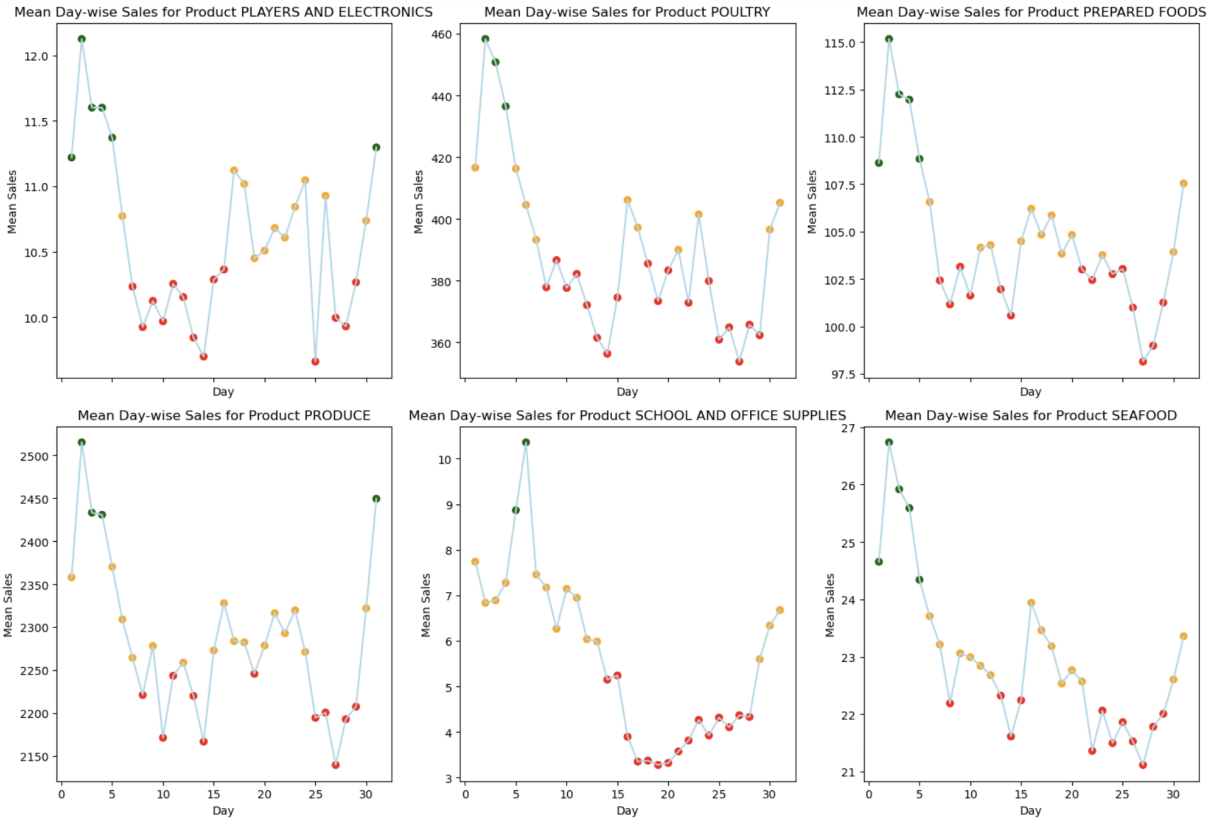


Figure 8. Daily Average Sales clustering for each product

2. **Most Selling Products** *(Figure 9)*
   - Upon plotting the yearly distribution of products as shown in *Figure 8*, approximately 80% of the sales came from just 5 products namely Grocery I, Beverages, Produce, Cleaning and Dairy. Among these, Grocery I & Beverages alone account for approximately 50% of the total sales each year.
   - Given their sales contribution, evaluating model performance for these products would be of paramount importance.

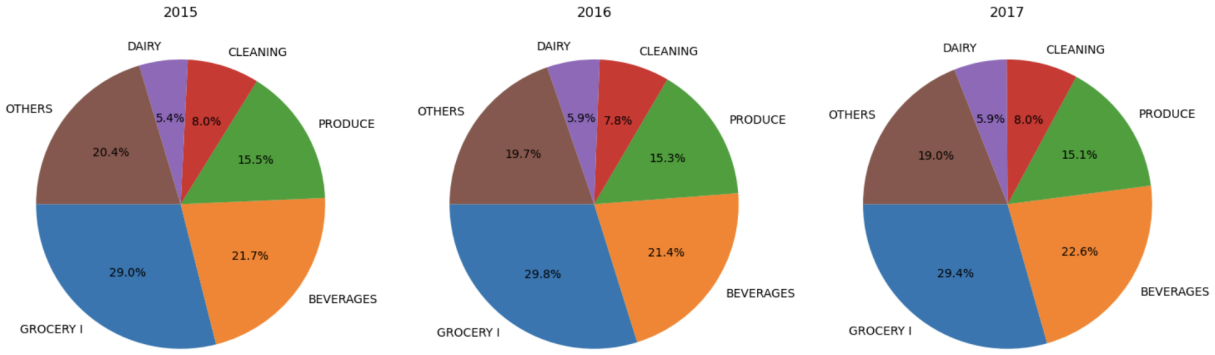


Figure 9. Sales Contribution percentage of different products for each year

3. **Correlation Matrix b/w special offer on products and sales** *(Figure 10)*

- Special Offers don't work equally effectively for all products and are generally less effective in increasing sales.
- School and Office supplies showed a standout correlation between sales & special offers each year underlining the feature importance of promotional offers only for selected products.
- Table 2 shows products with more than 20% correlation amongst sales & offers.

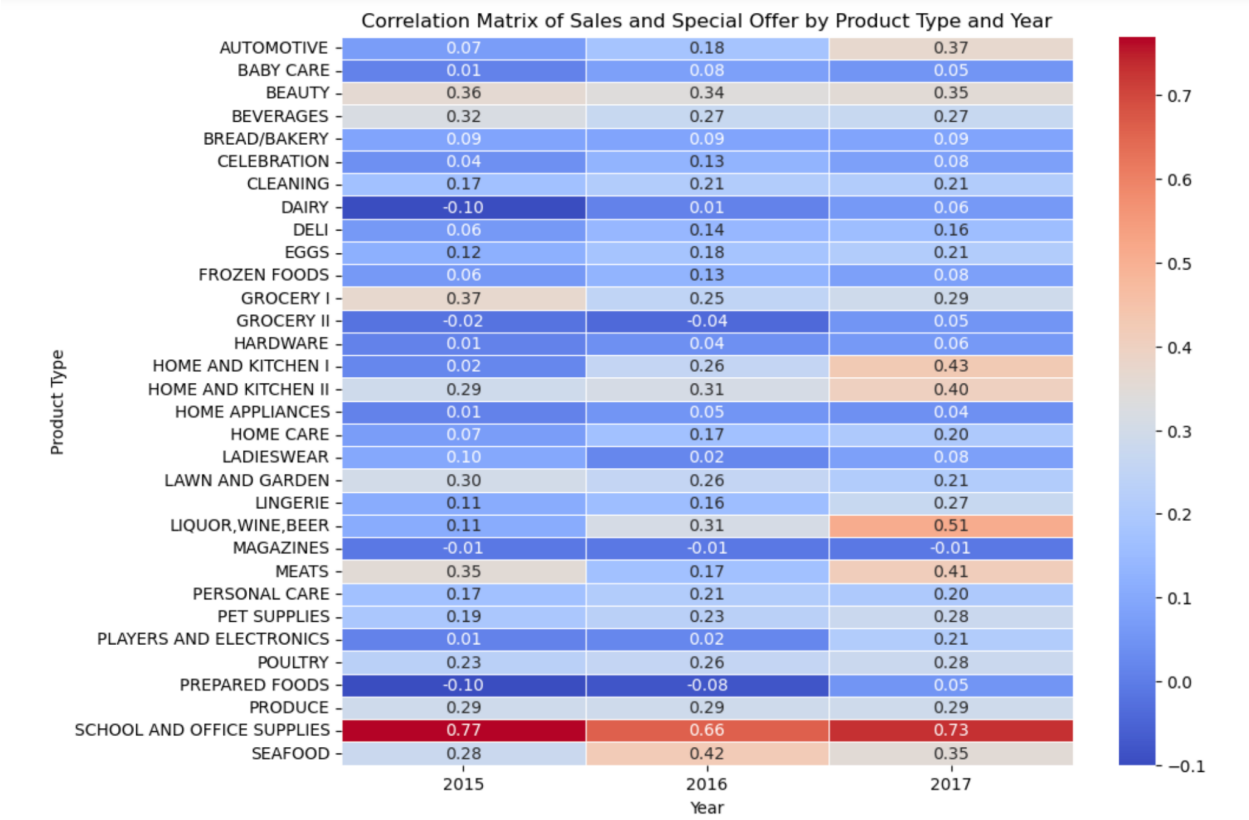


Figure 10. Correlation Matrix between sales and special offers by product for each year

year	2015	2016	2017
product_type			
BEAUTY	0.36	0.34	0.35
BEVERAGES	0.32	0.27	0.27
GROCERY I	0.37	0.25	0.29
HOME AND KITCHEN II	0.29	0.31	0.40
LAWN AND GARDEN	0.30	0.26	0.21
POULTRY	0.23	0.26	0.28
PRODUCE	0.29	0.29	0.29
SCHOOL AND OFFICE SUPPLIES	0.77	0.66	0.73
SEAFOOD	0.28	0.42	0.35

Table 2. Products with more than 20% correlation amongst sales & offers

4. **Special Offer Clusters** *(Figure 11)*
    - We formed monthly special offer clusters for products in Table n following the same trend of segments as for others with values as 1 (Low), 2 (Medium) and 3 (High) representing their monthly average offer segmentation for each month.

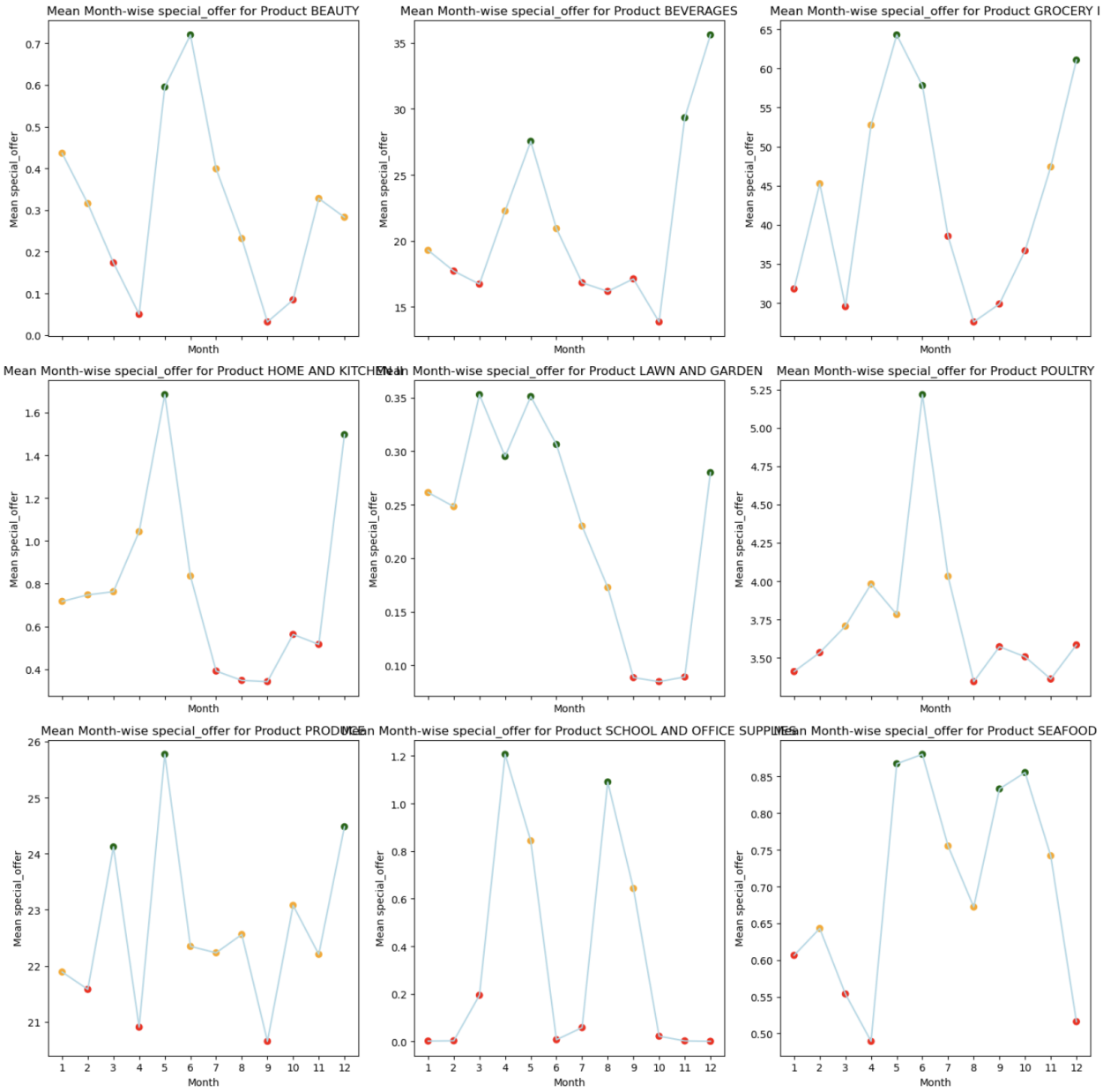


Figure 11. Special offer clusters for products with higher correlation between sales and offers

Store Specific Patterns + Clustering Possibility

1. **Correlation amongst stores** *(Figure 12)*
    - A notable correlation amongst all stores was found highlighting the importance of considering store clustering based on sales. Store 52 specifically had a low correlation with any of the stores since it opened in 2017 which needs to be considered while modelling.

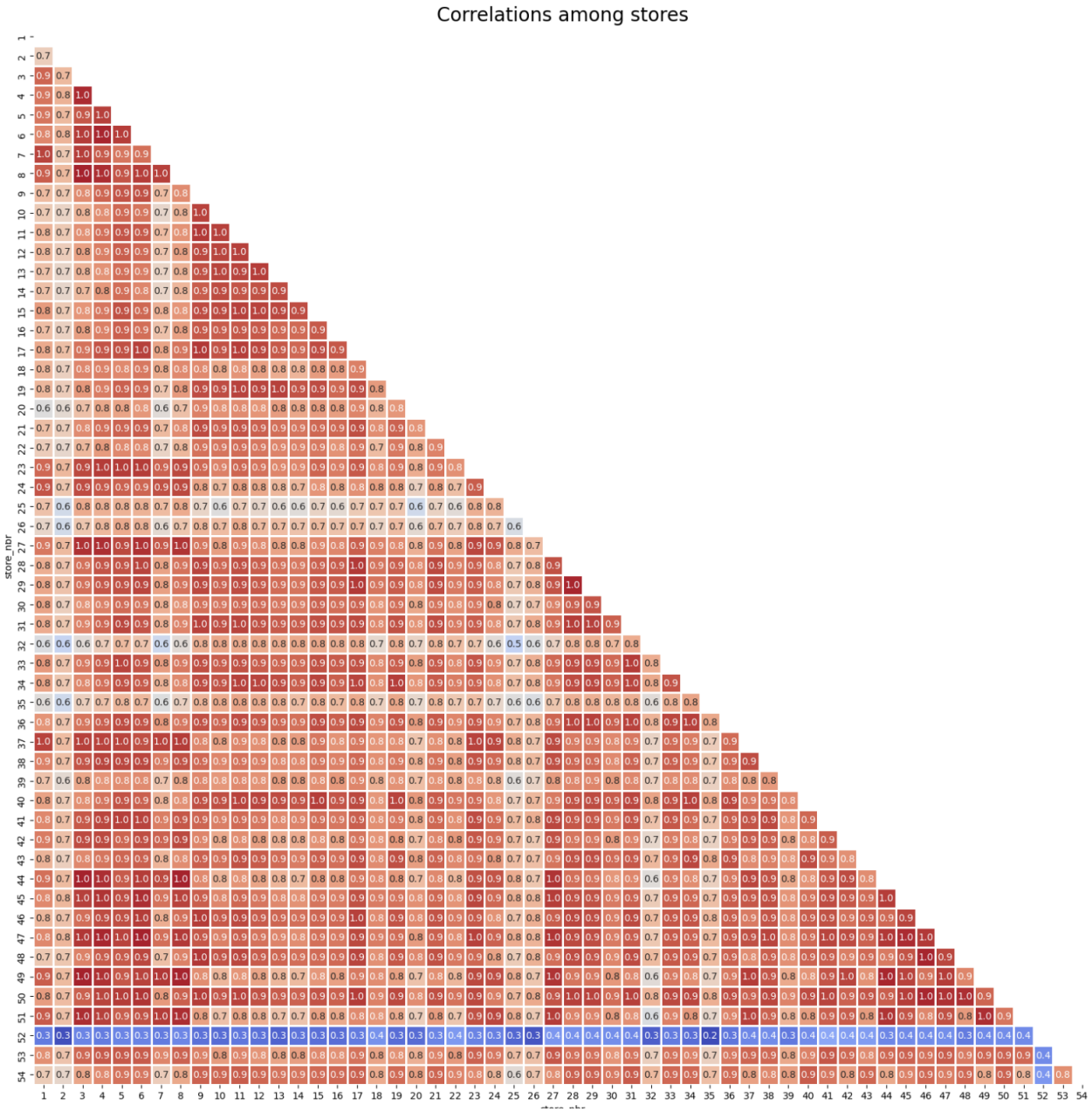


Figure 12. Correlation Matrix for sales of stores

- Recognizing the distinct sales patterns for different products and the aim being optimal inventory management, it seemed wiser to explore further store specific trends for each product individually.
- As shown in *Figure 13*, We created average monthly sales store clusters for each product with ordinal positioning i.e. 1 being the lowest selling segment. Around 7-12 clusters of stores were formed for each product. Plots for only 2 products have been included here for illustration.

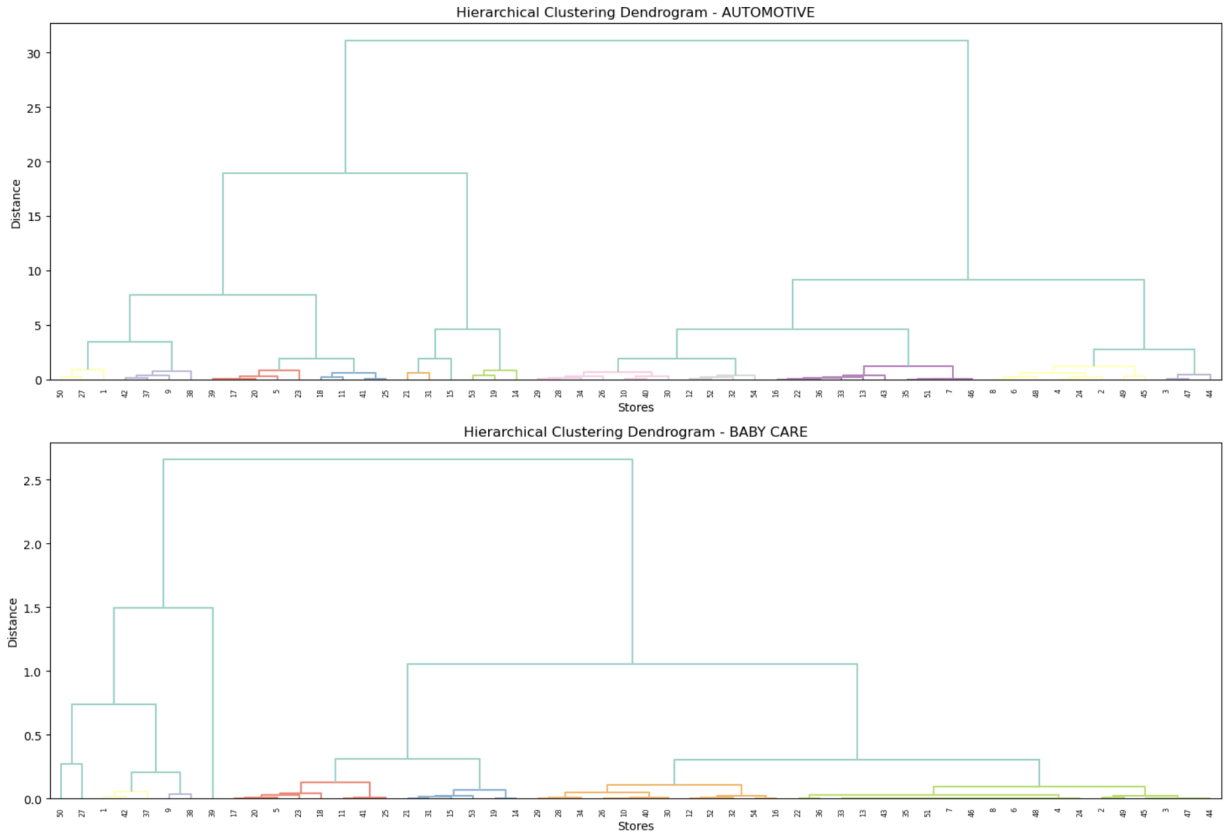


Figure 13. Clustering stores on the basis of monthly average for each product

2. **Products Unavailability in stores**
    - *Figure 14* shows a pictorial representation of which products are not sold in any particular stores.
    - Books are only sold in a limited number of stores while most of the stores sell all varieties of products.

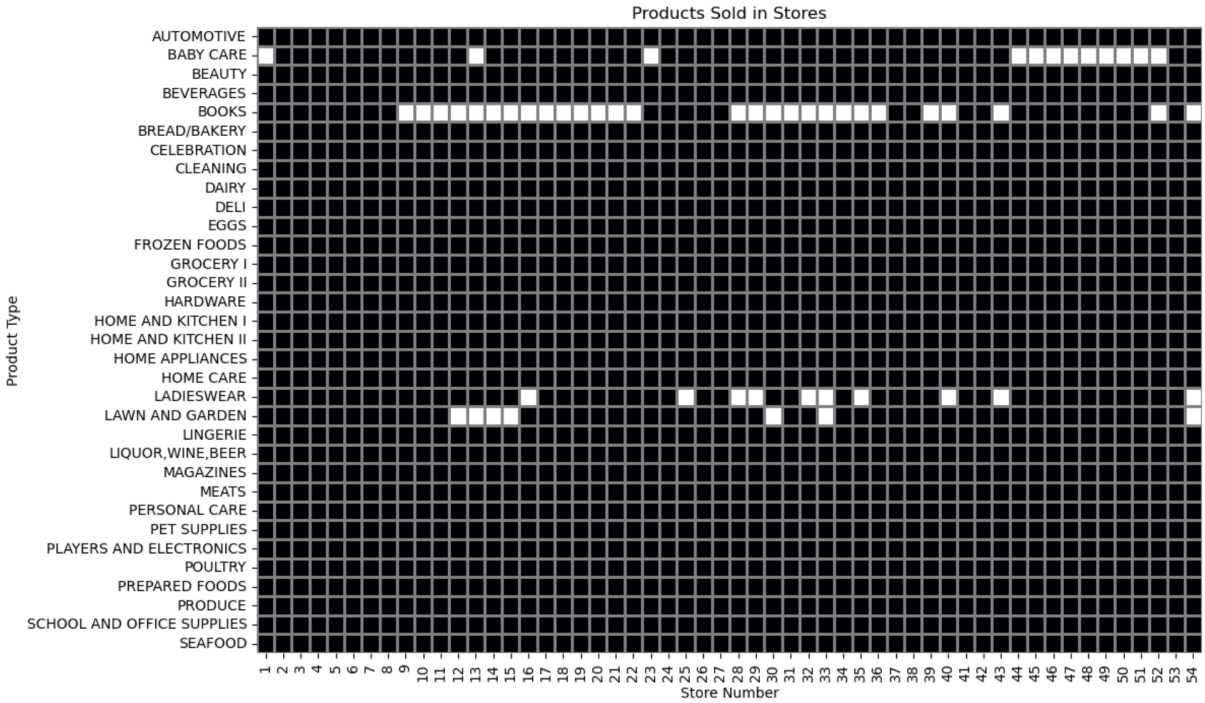


Figure 14. Heatmap illustrating product-store availability (white dots represent unavailability)

Outliers *(Figure 15)*

- Outliers were identified by finding unusual hike or dip in sales without any major offers on a particular day for each product, which can also be seen from the red dots in figure n. They were selected such that the sales on a particular day was 15 times the average sales of the particular product in that month and the sales jump or downfall was not attributed to promotions.
- Looking at School & Office Supplies, it had unusual highs and lows in its sales but minimal outliers due to the fact that they had promotional effect for its unusual sales.
- On the other hand, Home and Kitchen showed a lot of random behaviour which if not handled well, will make it difficult to predict its sales accurately.

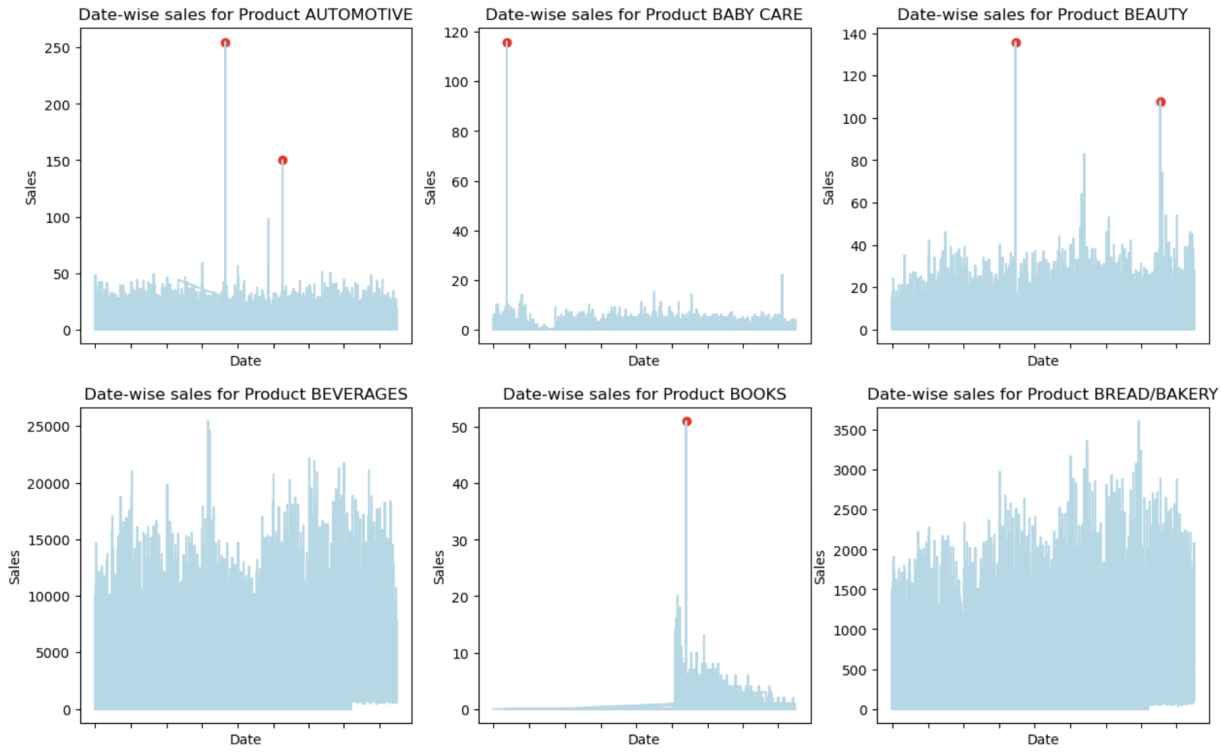


Figure 15. Highlighting sales outlier values (red points) for each product

To summarise, the comprehensive Exploratory Data Analysis conducted has provided key insights into the complexities of the intricate dynamics of this dataset. In light of these findings, we have laid the groundwork for subsequent modelling with the key takeaways being:

1. Dataset Trimming to contain data post July 2015 was strategically implemented to account for stable data with minimal random fluctuations for subsequent modelling.
2. Minimal similarity was observed between sales trajectories for different products suggesting to create different models for each product for better predictions.
3. Weekends showed a higher sales pattern in comparison to weekdays with a definite repetitive trend for each dow for sales as well as special offers.
4. High sales during the start of months which was probably due to salary releases and customers overall spending patterns.
5. Major hike in sales was observed in festive seasons around May and December in comparison to other months.
6. Many stores (store 52, opened in 2017) started functioning at a later date compared to other product and store combinations which needs to be accounted for, while modelling.
7. 5 products contributed to over 80% of the total sales and these products are daily usage products.
8. Special Offers are only effective for certain products and therefore any features related to special offer should only be used for selective products for which sales and offers are correlated.

9. Outliers tend to deviate from actual trends and therefore need to be removed from our dataset.

Basic Feature Engineering

To enhance the robustness and applicability of our future modelling efforts, we trimmed the dataset and refined the dataset to include the following possible additional features (as shown in *Table 3*) for the model feature set, some of which including dow, month and year were created during EDA. While the remaining were to be created during modelling as they could only be extracted from training dataset in order to avoid data leakage.

1. Dow: day of week
2. Month
3. Year
4. Start_date : date at which a product was launched at a store
5. Relative_monthly_sales_cluster: Cluster number in which the monthly average sales lies for that month (created ordinally, 1 being the lowest valued)
6. Relative_daily_sales_cluster: Cluster number in which the daily average sales lies for that day of month (created ordinally, 1 being the lowest valued)
7. Relative_monthly_offer_cluster: Cluster number in which the monthly average offer lies for that month (created ordinally, 1 being the lowest valued)
8. Relative_store_cluster: Cluster number in which the average sales of that store lies (created ordinally, 1 being the lowest valued)

	id	date	store_nbr	product_type	sales	special_offer	year	month	day	dow	is_weekend	start_date
1619838	1619838	2015-07-01	1	AUTOMOTIVE	5.0	0	2015	7	1	2	0	2013-01-02
1619839	1619839	2015-07-01	1	BABY CARE	0.0	0	2015	7	1	2	0	NaT
1619840	1619840	2015-07-01	1	BEAUTY	5.0	1	2015	7	1	2	0	2013-01-02
1619841	1619841	2015-07-01	1	BEVERAGES	2638.0	2	2015	7	1	2	0	2013-01-02
1619842	1619842	2015-07-01	1	BOOKS	0.0	0	2015	7	1	2	0	2016-10-13

Table 3. First 5 rows of the dataset after EDA & basic feature engineering

Modelling

In this section, we delve into the detailed explanation of the comprehensive modelling approach implemented to predict sales for the 16 days date range from 01/08/2017 to 16/08/2017. The framework involves execution of supervised machine learning algorithms including Random Forest, X Gradient Boost, and LightGBM to find the best working model for sales forecasting using cross validation and hyperparameter tuning. The structured pipeline includes data preprocessing to address the key issues identified during EDA, followed by feature engineering, hyperparameter tuning, and ultimately, the application of the selected model. Overall, 33 models were created for each of the unique product types to predict sales.

Note: Even though we deployed different models for each product, cross validation and hyperparameter tuning was only done on most critical products (highest selling products) to avoid the complexity of implementing different machine learning algorithms for test dataset for each product.

The key procedural steps undertaken during this phase include:

Data Preprocessing

The first step includes conditioning the dataset well enough for the model to identify trends and predict accurately. For this, a class 'ModelPreprocessing' was created, encompassing a suite of methods for dropping outliers, feature engineering to identify clusters and assessing inactive products. No encoding was required since we did not use any categorical columns in our feature set. Additionally, all numerical columns represented ordinal values. Regarding scaling all 3 employed models can work without scaling, for that reason scaling was also skipped. The functions in the class were namely:

1. drop_outliers: to drop outliers in the dataset
2. get_freq_group_cluster : to get relative daily sales cluster number & relative monthly sales cluster number
3. get_store_cluster: to get store cluster number
4. is_special_offer_effective: to assess whether the product is promotion sensitive. If yes, relative monthly offer cluster number
5. is_product_active: to assess whether the product has been active (available for sale) recently or not
6. set_lag_features: to include a lag feature representing the moving average of the DOW's average from the previous year

- Clustering includes techniques such as KMeans Clustering and Hierarchical Clustering to form clusters for sales and stores for each product.
- To avoid deviations in predicted sales, we have excluded the recent inactive products from the test dataset and we manually append their predicted sales as 0 with the model's predicted values.

Custom Grid Search for Hyperparameter Tuning

The heart of our modelling strategy lies in the custom class 'CustomGridSearch' created for applying grid search cross validation to orchestrate hyperparameter tuning. The need for a customised grid search method arose from the fact that our dataset was based on time and the sequence of time series was of utmost importance. Traditional validation sets are random subsets of train test splits which would not be possible to consider here. The dataset is firstly preprocessed and then the following methods are applied:

1. **Train-Test Split**

The number of folds and prediction days are passed to the method and accordingly sets of training and validation sets are created with prediction_days parameter as the number of days in validation sets and remaining data as the train set.

For illustration, to create 2 validation sets with 16 days prediction window:

The date ranges of validation set and train set would be:

Set 1:
train set: 01/07/2015 to 29/06/2017
validation set : 30/06/2017-15/07/2017

Set 2:
train set: 01/07/2015 to 14/07/2017
validation set : 15/07/2017-31/07/2017

2. GridSearchCV()

For selecting the best hyperparameters and cross validation, GridSearchCV from scikit-learn is deployed. The scoring method used for this was neg_root_mean_squared_error and finally the best hyperparameters and score obtained for each model was as shown in table .

3. Model Selection

A comparative graph for different models indicating the best scores for all 3 regressors can be seen from Figure 16. Detailed values can be referred from table 4. The lowest score indicating least deviation (nearest to zero) and best params were found from XGBoost Regressor with the same set of params for 4 out of 5 products_types passed for cross validation.

Best Params:

model	best_params	
XGBoost	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}	4

product	model	best_score	best_params
BEVERAGES	RandomForest	-379.945847	{'RF__max_depth': 10, 'RF__n_estimators': 10}
BEVERAGES	XGBoost	-60.765024	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
BEVERAGES	LightGBM	-340.247965	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}
CLEANING	RandomForest	-262.544810	{'RF__max_depth': 10, 'RF__n_estimators': 10}
CLEANING	XGBoost	-38.880899	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
CLEANING	LightGBM	-267.913110	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}
DAIRY	RandomForest	-104.502614	{'RF__max_depth': 10, 'RF__n_estimators': 10}
DAIRY	XGBoost	-14.985900	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
DAIRY	LightGBM	-105.765573	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}
GROCERY I	RandomForest	-820.853197	{'RF__max_depth': 10, 'RF__n_estimators': 10}
GROCERY I	XGBoost	-109.588523	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
GROCERY I	LightGBM	-795.383109	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}
PRODUCE	RandomForest	-0.438669	{'RF__max_depth': 10, 'RF__n_estimators': 10}
PRODUCE	XGBoost	-6.321448	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
PRODUCE	LightGBM	-110.948429	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}

Table 4. Best scores and params from Cross Validation for all 3 models for most selling products

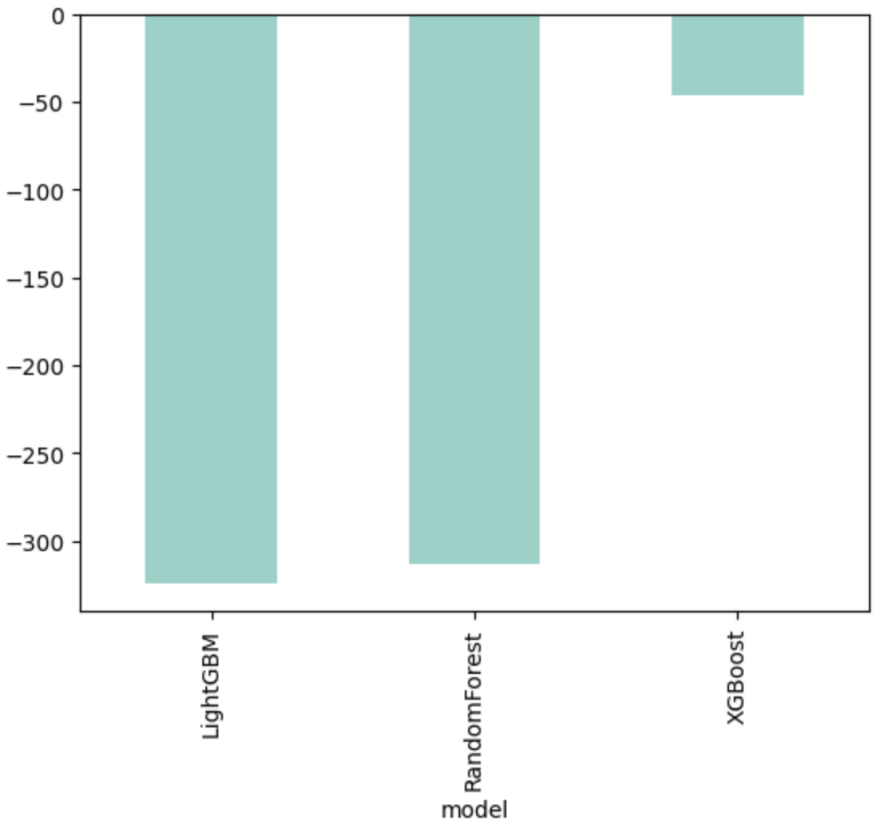


Figure 16. Model Score Comparisons

Model Application

Based on the score of cross validation, we selected the XGBoostRegressor since the neg_RMSE was the lowest (-46) out of all other models. A different model was employed for each of the products and individual predictions were made for them.

The Modelling for prediction on final test set is structured into several stages:

1. **Data Splitting**
   The dataset is divided into two subsets: a training set, comprising data until July 2017, and a test set, consisting of data for the next 16 days from 01/08/2017 to 16/07/2017. The pipeline has been crafted carefully to avoid data leakage and overfitting.

2. **Data Preprocessing**
   Similar to the cross validation pipeline, data is preprocessed to drop outliers, identify inactive products on store level and create clustering columns.

3. **Feature Selection**
   Apart from special offer clusters, all transformed and created features are passed to the model. The Features used for training the model include dow, year, month, lag_yoy_sales, relative_monthly_sales_cluster, relative_daily_sales_cluster, relative_store_cluster. Depending upon special offer effectiveness for a product, the column relative_monthly_offer_cluster is used as a feature.

4. **Model Training**
   An XGBoostRegressor model is trained on the preprocessed training set and fine-tuned with the pre-defined set of hyperparameters obtained from Grid Search cross validation to optimise predictive performance.

5. **Model Prediction**
   The trained XGBoostRegressor is applied to the preprocessed test set for sales prediction. The predictions are evaluated against the actual sales data, providing insights into the model's accuracy.

Performance Evaluation: Unveiling Insights into Sales Predictions

After putting our model to test, we now evaluate how well our model performed. In this comprehensive analysis, we explore various metrics and visualisations to dissect our model performance through multiple lenses. The Model takes between 1-2 minutes to run overall for all products with an average 1.2 seconds spent for each product’s prediction.

1. **Sales Comparison on product level**

   A bar chart is plotted for the products constituting majority of sales and remaining product types clubbed as others to illustrate a side by side comparison of cumulative actual sales vs predicted sales for the 16 day period. This visual (*Figure 17* and *Table 5*)allows us to identify patterns and discrepancies at a glance and we can see that the overall predictions are very close to the actual sales demonstrating a commendable performance.

product_type	avg_sales	avg_predicted_sales	mae	rmse
BEVERAGES	3469.02	3576.01	751.68	1084.93
CLEANING	1204.76	1301.95	314.43	527.65
DAIRY	832.77	884.40	118.71	182.63
GROCERY I	4580.49	4569.45	677.03	1027.26
PRODUCE	2290.86	2254.22	313.27	476.31
OTHERS	107.01	107.38	24.57	44.98

Table 5. Numerical Values for Sales v/s Predicted Sales

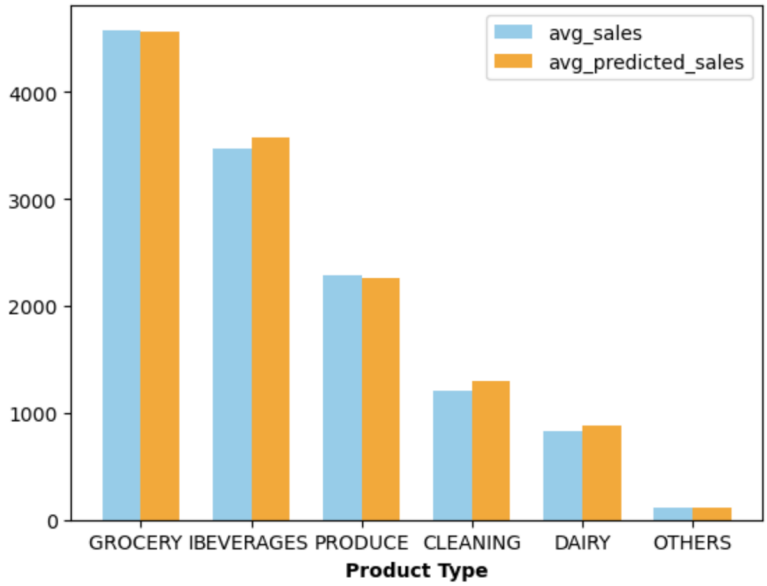
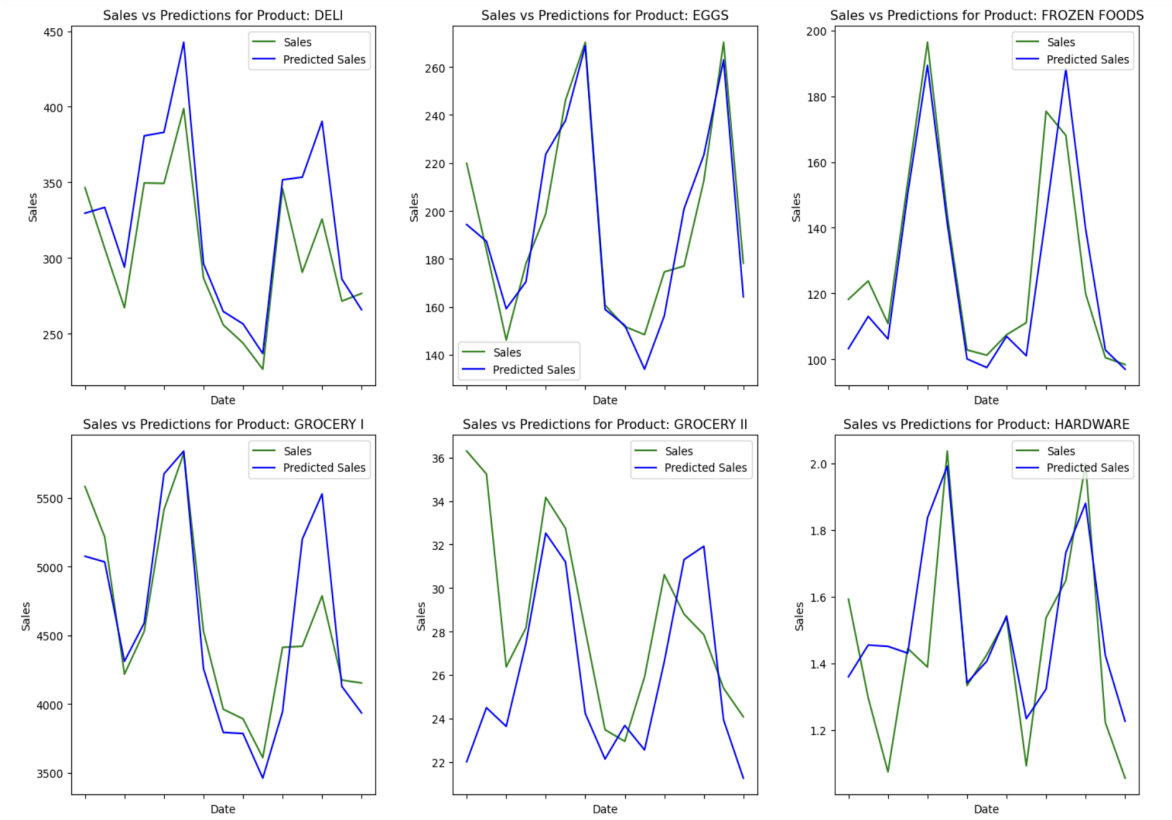


Figure 17. Sales v/s Predicted Sales comparison

2. **Line Plot analysis for actual vs predicted sales**

   To evaluate the model on a granular level, we plot a line chart to visualise trends and deviations between actual sales and predicted sales. Graphs for a few products are shown below for illustration in *Figure 18*. It can be observed from the plots that the model has identified sales patterns for most of the products and has worked really well.

However, the challenge specifically lies in sales forecasting for School and Office Supplies and Books with significant deviation from actual sales. This is attributed to sparse data and erratic sales patterns as observed during EDA.



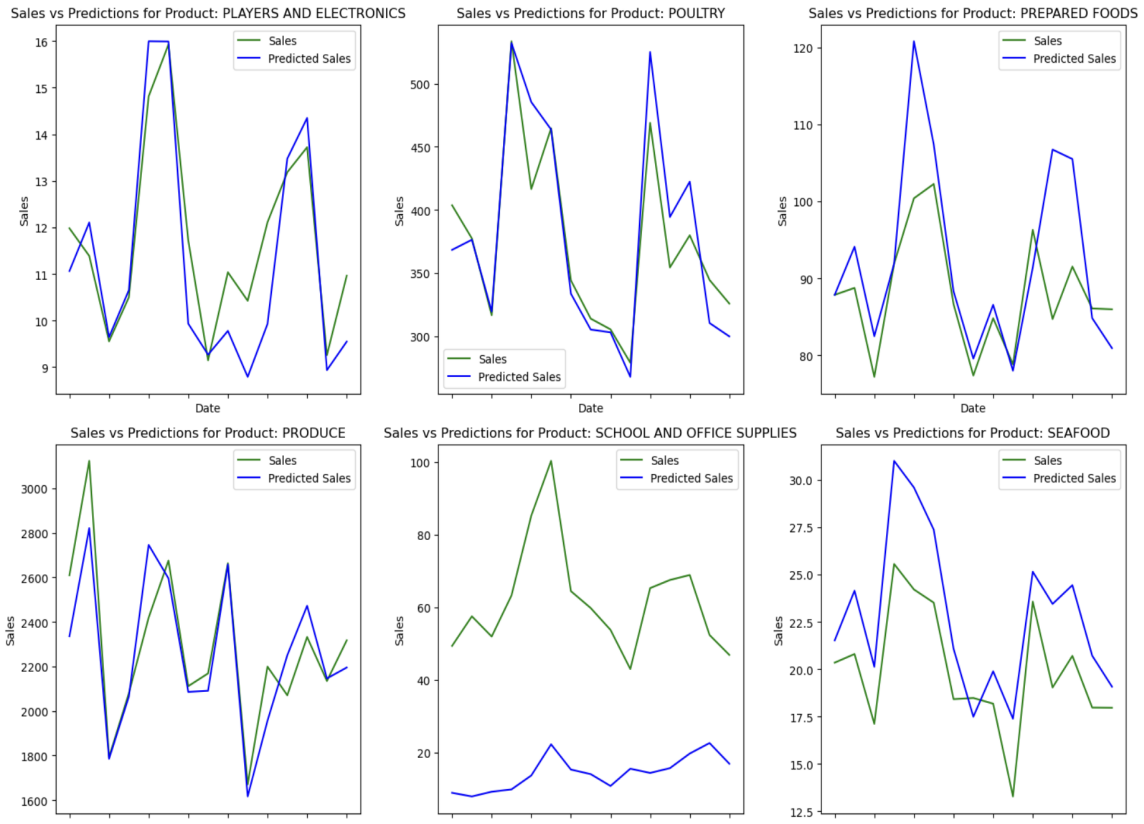


Figure 18. Date level actual vs predicted sales trends

3. **<u>Day-wise Percentage Deviation for all products</u>**

   Figure 19 shows day-wise percentage deviation of predicted sales from actual sales for the most selling products providing a quantitative measure of the model's accuracy. Of these, the model predicts well within the deviation of 20% which is an acceptable industry standard deviation for 4 products. Notably, Dairy products exhibit a higher deviation which is not a good sign and might need some more re tuning and in-depth analysis.

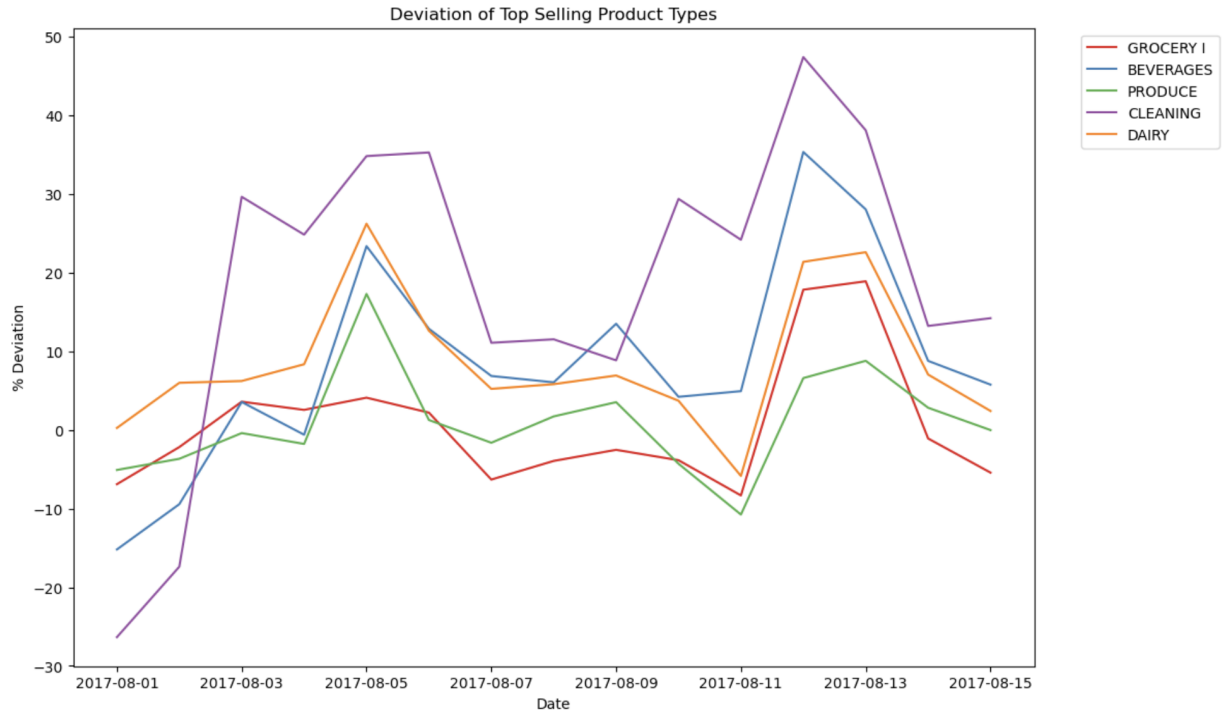


Figure 19. Date level actual vs predicted sales deviation percentages for most selling products

4. **<u>Shelf Life Sensitivity & Predictions</u>**

   A crucial aspect of maintaining stocks is understanding the shelf life of products. The model's performance is also scrutinised by assessing the predictions for low shelf life products. Based on domain knowledge, a list of products as shown in *Figure 20* was selected as sensitive products with low shelf life. The findings indicated a thumbs up for the model's working by aligning with the deviation under 20% for most of the products. However, Seafood emerged as an outlier, indicating a need for specialised attention to enhance predictions for this product category.

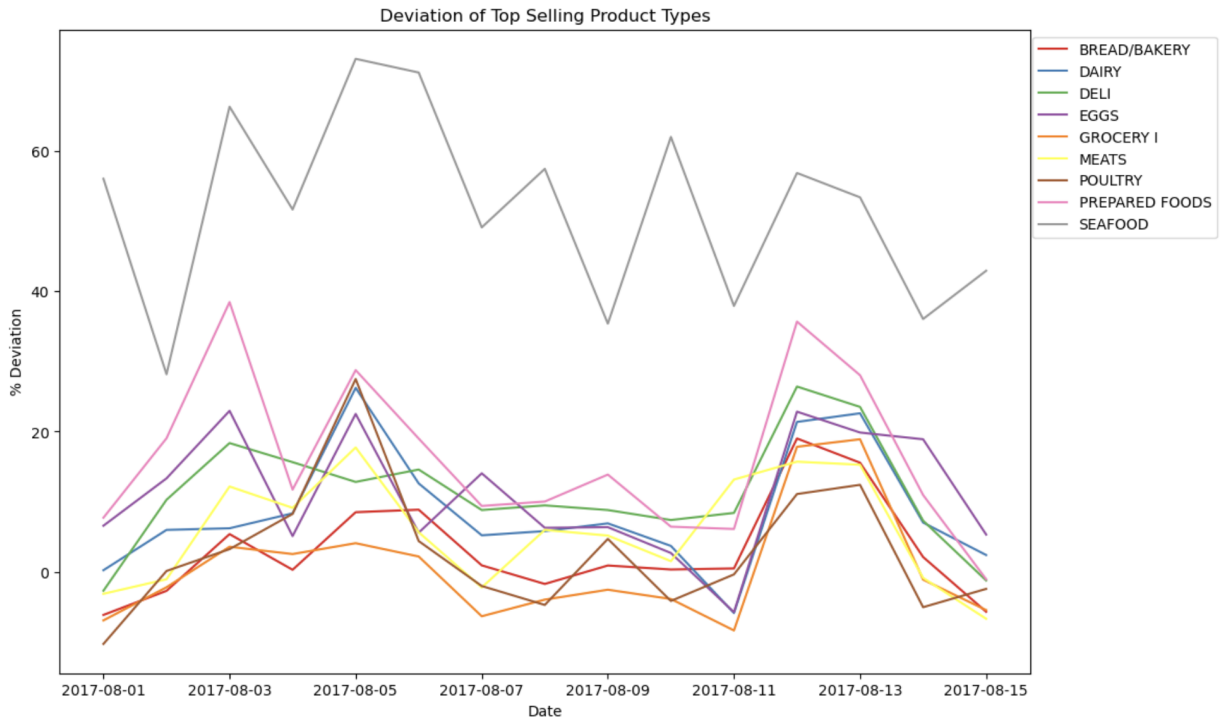


Figure 20. Date level actual vs predicted sales deviation percentages for low shelf life products

5. **<u>Day-wise Out of Stock Percentage Predictions for each product</u>**
   Preventing out of stock scenarios is paramount to prevent sales loss and customer dissatisfaction in the retail business. The *Table 6* shows Out of Stock (OOS) percentages for all products across all days. The data shows our model tends to predict sales lesser than the actual sales most of the times which needs to be taken care of. However, there is minimal wastage based on the predictions since predicted values are generally lower than the actual sales.

oos_flag

date product_type	2017-08-01	2017-08-02	2017-08-03	2017-08-04	2017-08-05	2017-08-06	2017-08-07	2017-08-08	2017-08-09	2017-08-10	2017-08-11	2017-08-12	2017-08-13	2017-08-14	2017-08-15
AUTOMOTIVE	50.00	48.15	40.74	59.26	33.33	61.11	59.26	53.70	46.30	40.74	64.81	31.48	46.30	37.04	62.96
BABY CARE	14.81	14.81	7.41	14.81	9.26	14.81	18.52	16.67	14.81	9.26	22.22	0.00	12.96	9.26	12.96
BEAUTY	48.15	38.89	44.44	48.15	62.96	53.70	61.11	64.81	59.26	57.41	61.11	51.85	48.15	51.85	61.11
BEVERAGES	79.63	64.81	46.30	44.44	24.07	48.15	37.04	37.04	37.04	44.44	42.59	14.81	27.78	31.48	31.48
BOOKS	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.85	0.00	0.00	0.00	0.00	0.00	0.00
BREAD/BAKERY	57.41	61.11	37.04	51.85	35.19	46.30	50.00	66.67	50.00	46.30	59.26	22.22	38.89	40.74	51.85
CELEBRATION	57.41	33.33	35.19	42.59	40.74	40.74	35.19	37.04	33.33	62.96	42.59	24.07	44.44	57.41	37.04
CLEANING	94.44	83.33	40.74	24.07	14.81	16.67	29.63	35.19	35.19	12.96	20.37	11.11	24.07	33.33	40.74
DAIRY	59.26	35.19	40.74	33.33	11.11	37.04	37.04	38.89	29.63	35.19	68.52	20.37	29.63	29.63	46.30
DELI	57.41	33.33	25.93	31.48	31.48	31.48	35.19	25.93	35.19	46.30	42.59	7.41	25.93	37.04	51.85
EGGS	53.70	48.15	31.48	62.96	31.48	53.70	37.04	44.44	48.15	57.41	70.37	22.22	50.00	42.59	61.11
FROZEN FOODS	70.37	64.81	51.85	50.00	59.26	51.85	50.00	51.85	50.00	59.26	75.93	31.48	25.93	46.30	53.70
GROCERY I	64.81	57.41	42.59	40.74	40.74	51.85	70.37	50.00	66.67	53.70	75.93	29.63	25.93	44.44	61.11
GROCERY II	79.63	68.52	57.41	59.26	46.30	57.41	64.81	61.11	38.89	55.56	51.85	42.59	40.74	46.30	51.85
HARDWARE	48.15	33.33	25.93	40.74	33.33	42.59	42.59	38.89	38.89	38.89	42.59	40.74	37.04	35.19	33.33
HOME AND KITCHEN I	59.26	37.04	35.19	33.33	42.59	44.44	46.30	37.04	46.30	55.56	68.52	35.19	37.04	59.26	57.41
HOME AND KITCHEN II	59.26	51.85	38.89	46.30	40.74	50.00	42.59	51.85	42.59	38.89	55.56	27.78	37.04	57.41	53.70
HOME APPLIANCES	18.52	16.67	12.96	11.11	20.37	16.67	16.67	14.81	5.56	5.56	16.67	7.41	1.85	5.56	5.56
HOME CARE	77.78	48.15	33.33	11.11	9.26	25.93	29.63	25.93	20.37	29.63	38.89	9.26	14.81	20.37	31.48
LADIESWEAR	38.89	33.33	33.33	22.22	16.67	22.22	25.93	37.04	29.63	14.81	33.33	20.37	16.67	31.48	37.04
LAWN AND GARDEN	37.04	35.19	24.07	24.07	35.19	29.63	25.93	33.33	27.78	37.04	38.89	27.78	25.93	31.48	33.33
LINGERIE	38.89	35.19	42.59	37.04	42.59	37.04	31.48	40.74	37.04	51.85	57.41	38.89	35.19	44.44	44.44
LIQUOR,WINE,BEER	31.48	62.96	48.15	29.63	85.19	70.37	9.26	25.93	48.15	64.81	74.07	64.81	51.85	14.81	38.89
MAGAZINES	61.11	46.30	31.48	40.74	57.41	61.11	57.41	53.70	46.30	51.85	46.30	40.74	50.00	72.22	66.67
MEATS	62.96	62.96	44.44	50.00	24.07	55.56	66.67	44.44	50.00	48.15	46.30	37.04	35.19	53.70	74.07
PERSONAL CARE	70.37	57.41	29.63	50.00	35.19	40.74	61.11	50.00	51.85	38.89	75.93	22.22	18.52	40.74	57.41
PET SUPPLIES	44.44	38.89	42.59	35.19	44.44	44.44	44.44	44.44	44.44	33.33	61.11	27.78	31.48	37.04	40.74
PLAYERS AND ELECTRONICS	53.70	31.48	40.74	48.15	37.04	44.44	55.56	37.04	57.41	55.56	51.85	33.33	50.00	48.15	51.85
POULTRY	72.22	59.26	57.41	42.59	14.81	50.00	64.81	61.11	50.00	62.96	66.67	27.78	35.19	61.11	61.11
PREPARED FOODS	44.44	35.19	29.63	42.59	24.07	37.04	44.44	40.74	42.59	50.00	55.56	29.63	38.89	46.30	50.00
PRODUCE	68.52	62.96	46.30	48.15	16.67	64.81	59.26	59.26	38.89	62.96	70.37	35.19	38.89	44.44	64.81
SCHOOL AND OFFICE SUPPLIES	40.74	37.04	38.89	38.89	29.63	22.22	27.78	42.59	35.19	33.33	55.56	50.00	37.04	38.89	38.89
SEAFOOD	37.04	29.63	29.63	20.37	22.22	31.48	29.63	40.74	29.63	20.37	33.33	22.22	25.93	31.48	35.19

Table 6. Day level Out of Stock percentages for all products

6. Data Leakage & Overfit Examination

Clear distinction between training and test datasets ensuring that the training set precedes the test set as well as the validation sets chronologically has been maintained to avoid data leakage.

Conclusion: Towards Model Refinement and Optimization

In a nutshell, it can be concluded that the model has worked exceptionally well for some products while there still are areas of improvement for others. The model performance evaluation comprehensively underlines the areas of strengths and marks for improvement providing an aligned direction for specific products.

Roadmap for future model refinements

1. Dashboards could be created for continuous monitoring of the metrics evaluated during model performance evaluation to measure improvement over time.
2. In the critical facet of the retail industry, a subtle balance between OOS and wastage is of utmost importance, More so, it should be initially preferred to have a higher stock of inventory at all times to build a stronger customer base. To handle this, the inventory created could be a multiple of the predicted sales for a few months until the model starts to grasp the trends automatically.
3. Special attention to identify underlying trends to improve predictions for a few products such as School & Office Supplies, Seafood and books can be done.
4. Moving forward, a continuous feedback loop between model evaluation and refinement is recommended, fostering an iterative process that ensures the model evolves with changing data patterns and business dynamics.
5. Inter-store dispatching of products can also be done by creating a dashboard to manage the system with automation in order to efficiently utilise excess inventory.

Challenges Faced

One major challenge that still remains and incorporating which might improve the predictions are introduction of more lag features. We could not include recency based lag features such as values for a day before as the model was predicting values for n days in advance and for using the predicted values as lag values, we would have to predict one day at a time. Due to time constraints, we could not implement that completely and have removed that implementation from this report but we believe inclusion of recent lag features would significantly improve the sales prediction.

REFERENCES

1. Feature Engineering, https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10
2. K-Means clustering method, https://scikit-learn.org/stable/modules/clustering.html#k-means
3. Hierarchical Clustering, https://www.w3schools.com/python/python_ml_hierarchial_clustering.asp
4. X Gradient Boost Technique, https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/
5. Light GBM modelling technique, https://lightgbm.readthedocs.io/en/latest/Python-Intro.html

bman73701-group24-code-1

December 8, 2023

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
import numpy as np
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import IsolationForest
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import warnings
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
import joblib
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression
from datetime import timedelta,date
import lightgbm as lgb
from itertools import product
import numpy as np
import time

# from fbprophet import Prophet
warnings.filterwarnings("ignore")
```

[2]:
```
data = pd.read_csv('/Users/rahulsen/Downloads/Products_Information.csv')
data.head()
```

[2]:

	id	date	store_nbr	product_type	sales	special_offer
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	1	2013-01-01	1	BABY CARE	0.0	0
2	2	2013-01-01	1	BEAUTY	0.0	0
3	3	2013-01-01	1	BEVERAGES	0.0	0
4	4	2013-01-01	1	BOOKS	0.0	0

0.1 Summary Statistics:

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000888 entries, 0 to 3000887
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   id             int64
 1   date           object
 2   store_nbr      int64
 3   product_type   object
 4   sales          float64
 5   special_offer  int64
dtypes: float64(1), int64(3), object(2)
memory usage: 137.4+ MB
```

[4]:
```
data['date'] = pd.to_datetime(data['date'], format='%Y-%m-%d')
data.describe().T

# date was object time, hence converted to datetime
# total 30L records
# sales for >25% products is zero
# almost half of products are low quantity selling products
```

[4]:

	count	mean	std	min	25%	50%	75%	max
id	3000888.0	1.500444e+06	866281.891642	0.0	750221.75	1500443.5	2.250665e+06	3000887.0
store_nbr	3000888.0	2.750000e+01	15.585787	1.0	14.00	27.5	4.100000e+01	54.0
sales	3000888.0	3.577757e+02	1101.997721	0.0	0.00	11.0	1.958473e+02	124717.0
special_offer	3000888.0	2.602770e+00	12.218882	0.0	0.00	0.0	0.000000e+00	741.0

[5]:
```
data['store_nbr'].nunique(), data['product_type'].nunique()
# total stores 54
```

```
# total products 33
```

[5]:
```
(54, 33)
```

0.2 Data Consistency Check:

[6]:
```
data_check = data.groupby(['store_nbr','product_type','date'])['id'].count().
 ↪reset_index()
data_check[data_check['id']>1]

# no duplicates
```

[6]:
```
Empty DataFrame
Columns: [store_nbr, product_type, date, id]
Index: []
```

0.3 Missing Data Check:

[7]:
```
data.isna().sum(axis=0)
# no nulls
```

[7]:
```
id              0
date            0
store_nbr       0
product_type    0
sales           0
special_offer   0
dtype: int64
```

[8]:
```
store_occurences = data['store_nbr'].value_counts().unique()
print (store_occurences)

# data present for each product's sales for each day at each store even when␣
 ↪the outlet doesn't sell that product
if len(store_occurences)==1:
    print (int(store_occurences[0]/(data['product_type'].nunique())), 'days')
    print (data['date'].max()-data['date'].min()+timedelta(days=1))
# four days missing
```

```
[55572]
1684 days
1688 days 00:00:00
```

0.3.1 25th Dec: Stores Closed

[9]:
```
complete_date_range = pd.date_range(start=data['date'].min(), end=data['date'].
↪max(), freq='D')

missing_dates = set(complete_date_range) - set(data['date'])

missing_dates_df = pd.DataFrame({'missing_dates': sorted(list(missing_dates))})

print(missing_dates_df)
# all outlets remain closed on Christmas
```

```
  missing_dates
0    2013-12-25
1    2014-12-25
2    2015-12-25
3    2016-12-25
```

0.4 Feature Engineering (Basic):

[10]:
```
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
data['dow'] = data['date'].dt.dayofweek
data['week_num'] = data['date'].dt.isocalendar().week
data.head()
```

[10]:

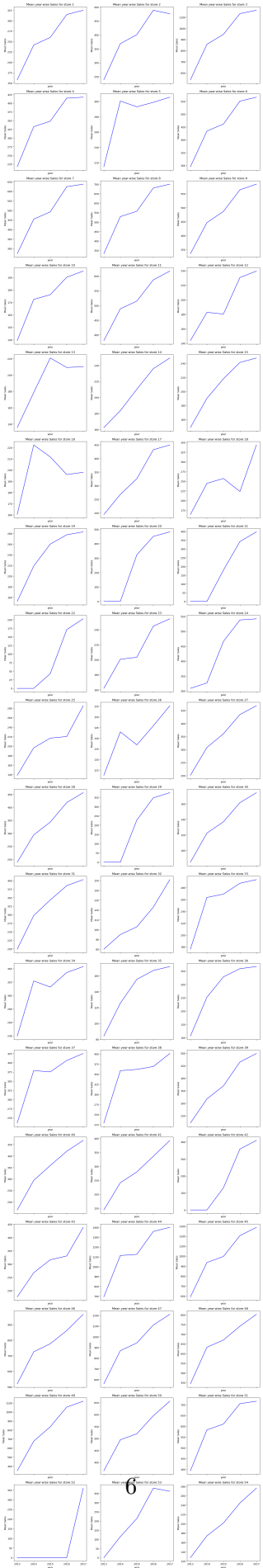
	id	date	store_nbr	product_type	sales	special_offer	year	month	\
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	2013	1	
1	1	2013-01-01	1	BABY CARE	0.0	0	2013	1	
2	2	2013-01-01	1	BEAUTY	0.0	0	2013	1	
3	3	2013-01-01	1	BEVERAGES	0.0	0	2013	1	
4	4	2013-01-01	1	BOOKS	0.0	0	2013	1	

	day	dow	week_num
0	1	1	1
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1

1 EDA : Sales Pattern

1.1 Analysis 1 (wrt Stores):

[11]:
```
df = data.copy()
unique_stores = df['store_nbr'].unique()

num_rows = math.ceil(len(unique_stores) / 3)
num_cols = min(len(unique_stores), 3)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)


for i, store in enumerate(sorted(unique_stores.tolist())):
    row_idx = i // 3
    col_idx = i % 3

    store_data = df[df['store_nbr'] == store].groupby(['year'])['sales'].mean()
    store_data = store_data.reset_index()

    axes[row_idx, col_idx].plot(store_data['year'], store_data['sales'],␣
 ↪label=f'Mean Sales - {store}', c = 'blue')
    axes[row_idx, col_idx].set_xticks(store_data['year'])
    axes[row_idx, col_idx].set_xlabel('year')
    axes[row_idx, col_idx].set_ylabel('Mean Sales')
    axes[row_idx, col_idx].set_title(f'Mean year-wise Sales for store {store}')

plt.tight_layout()
plt.show()

# Similar patterns of sales observed between some stores, hence can be clustered
# 52: opened at 2017 (no prev sales)
# 42, 29, 22, 21, 20: started in 2014
# Generally increasing sales year to year
```


Mean year-wise Sales for store 1
Mean year-wise Sales for store 2
Mean year-wise Sales for store 3
Mean year-wise Sales for store 4
Mean year-wise Sales for store 5
Mean year-wise Sales for store 6
Mean year-wise Sales for store 7
Mean year-wise Sales for store 8
Mean year-wise Sales for store 9
Mean year-wise Sales for store 10
Mean year-wise Sales for store 11
Mean year-wise Sales for store 12
Mean year-wise Sales for store 13
Mean year-wise Sales for store 14
Mean year-wise Sales for store 15
Mean year-wise Sales for store 16
Mean year-wise Sales for store 17
Mean year-wise Sales for store 18
Mean year-wise Sales for store 19
Mean year-wise Sales for store 20
Mean year-wise Sales for store 21
Mean year-wise Sales for store 22
Mean year-wise Sales for store 23
Mean year-wise Sales for store 24
Mean year-wise Sales for store 25
Mean year-wise Sales for store 26
Mean year-wise Sales for store 27
Mean year-wise Sales for store 28
Mean year-wise Sales for store 29
Mean year-wise Sales for store 30
Mean year-wise Sales for store 31
Mean year-wise Sales for store 32
Mean year-wise Sales for store 33
Mean year-wise Sales for store 34
Mean year-wise Sales for store 35
Mean year-wise Sales for store 36
Mean year-wise Sales for store 37
Mean year-wise Sales for store 38
Mean year-wise Sales for store 39
Mean year-wise Sales for store 40
Mean year-wise Sales for store 41
Mean year-wise Sales for store 42
Mean year-wise Sales for store 43
Mean year-wise Sales for store 44
Mean year-wise Sales for store 45
Mean year-wise Sales for store 46
Mean year-wise Sales for store 47
Mean year-wise Sales for store 48
Mean year-wise Sales for store 49
Mean year-wise Sales for store 50
Mean year-wise Sales for store 51
Mean year-wise Sales for store 52
Mean year-wise Sales for store 53
Mean year-wise Sales for store 54
year
Mean Sales

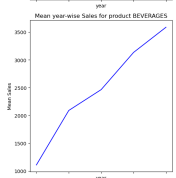
1.2 Analysis 2 (wrt Product):

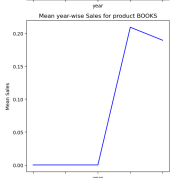
[12]:
```
df = data.copy()
unique_products = df['product_type'].unique()

num_rows = math.ceil(len(unique_products) / 3)
num_cols = min(len(unique_products), 3)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)


for i, product in enumerate(sorted(unique_products.tolist())):
    row_idx = i // 3
    col_idx = i % 3

    product_data = df[df['product_type'] == product].groupby(['year'])['sales'].
 ↪mean()
    product_data = product_data.reset_index()

    axes[row_idx, col_idx].plot(product_data['year'], product_data['sales'],␣
 ↪label=f'Mean Sales - {product}', c = 'blue')
    axes[row_idx, col_idx].set_xticks(product_data['year'])
    axes[row_idx, col_idx].set_xlabel('year')
    axes[row_idx, col_idx].set_ylabel('Mean Sales')
    axes[row_idx, col_idx].set_title(f'Mean year-wise Sales for product␣
 ↪{product}')

plt.tight_layout()
plt.show()

# Most products have different sales pattern, hence clustering not applied
# Also, since the aim is inventory management, it makes sense to not create␣
 ↪clusters of products
# Books started selling from 2015
# Many products sell in lower quantity
# High quantity selling products are mostly daily usage products such as dairy,␣
 ↪grocery I etc., also these products have lower shelf life compared to others␣
 ↪(major focus on their prediction)
```


Mean year-wise Sales for product AUTOMOTIVE
Mean Sales
year

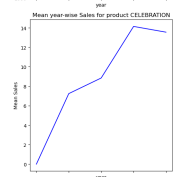
Mean year-wise Sales for product BABY CARE
Mean Sales
year

Mean year-wise Sales for product BEAUTY
Mean Sales
year

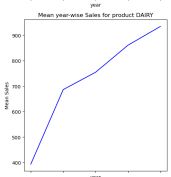

Mean year-wise Sales for product BEVERAGES
Mean Sales
year

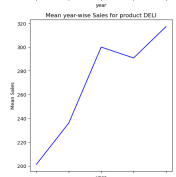

Mean year-wise Sales for product BOOKS
Mean Sales
year

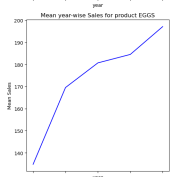

Mean year-wise Sales for product BREAD/BAKERY
Mean Sales
year


Mean year-wise Sales for product CELEBRATION
Mean Sales
year

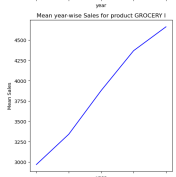

Mean year-wise Sales for product CLEANING
Mean Sales
year

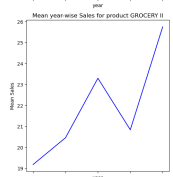

Mean year-wise Sales for product DAIRY
Mean Sales
year


Mean year-wise Sales for product DELI
Mean Sales
year

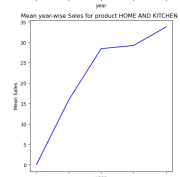

Mean year-wise Sales for product EGGS
Mean Sales
year

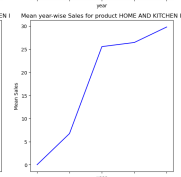

Mean year-wise Sales for product FROZEN FOODS
Mean Sales
year

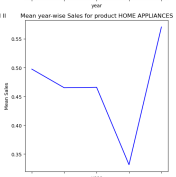

Mean year-wise Sales for product GROCERY I
Mean Sales
year

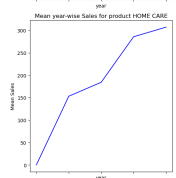

Mean year-wise Sales for product GROCERY II
Mean Sales
year


Mean year-wise Sales for product HARDWARE
Mean Sales
year

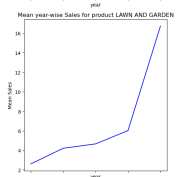

Mean year-wise Sales for product HOME AND KITCHEN I
Mean Sales
year

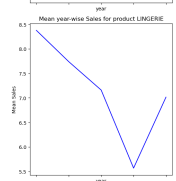

Mean year-wise Sales for product HOME AND KITCHEN II
Mean Sales
year

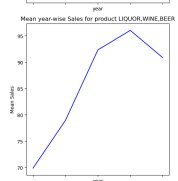

Mean year-wise Sales for product HOME APPLIANCES
Mean Sales
year

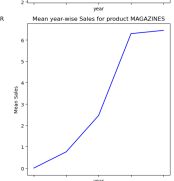

Mean year-wise Sales for product HOME CARE
Mean Sales
year

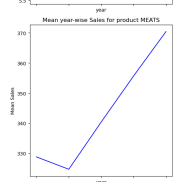

Mean year-wise Sales for product LADIESWEAR
Mean Sales
year

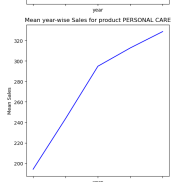

Mean year-wise Sales for product LAWN AND GARDEN
Mean Sales
year

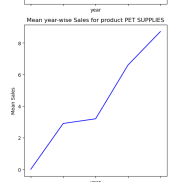

Mean year-wise Sales for product LINGERIE
Mean Sales
year

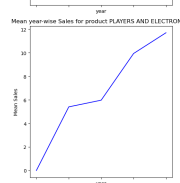

Mean year-wise Sales for product LIQUOR,WINE,BEER
Mean Sales
year

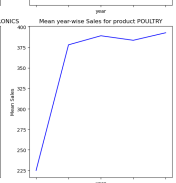

Mean year-wise Sales for product MAGAZINES
Mean Sales
year

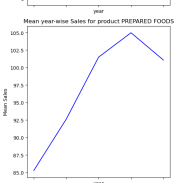

Mean year-wise Sales for product MEATS
Mean Sales
year

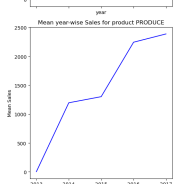

Mean year-wise Sales for product PERSONAL CARE
Mean Sales
year


Mean year-wise Sales for product PET SUPPLIES
Mean Sales
year

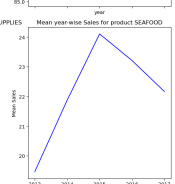

Mean year-wise Sales for product PLAYERS AND ELECTRONICS
Mean Sales
year


Mean year-wise Sales for product POULTRY
Mean Sales
year


Mean year-wise Sales for product PREPARED FOODS
Mean Sales
year


Mean year-wise Sales for product PRODUCE
Mean Sales
year


Mean year-wise Sales for product SCHOOL AND OFFICE SUPPLIES
Mean Sales
year


Mean year-wise Sales for product SEAFOOD
Mean Sales
year

1.2.1 Adding start_date column

[13]:
```
df = data.copy()
df = df[df['sales']>0]
df = df.groupby(['product_type','store_nbr'])['date'].min().reset_index()
df = df.rename(columns = {'date':'start_date'})
data = pd.merge(data, df, how='left', on=['product_type','store_nbr'])
data.head()
# some products started selling at a later point in time
# some stores still don't sell some products
```

[13]:

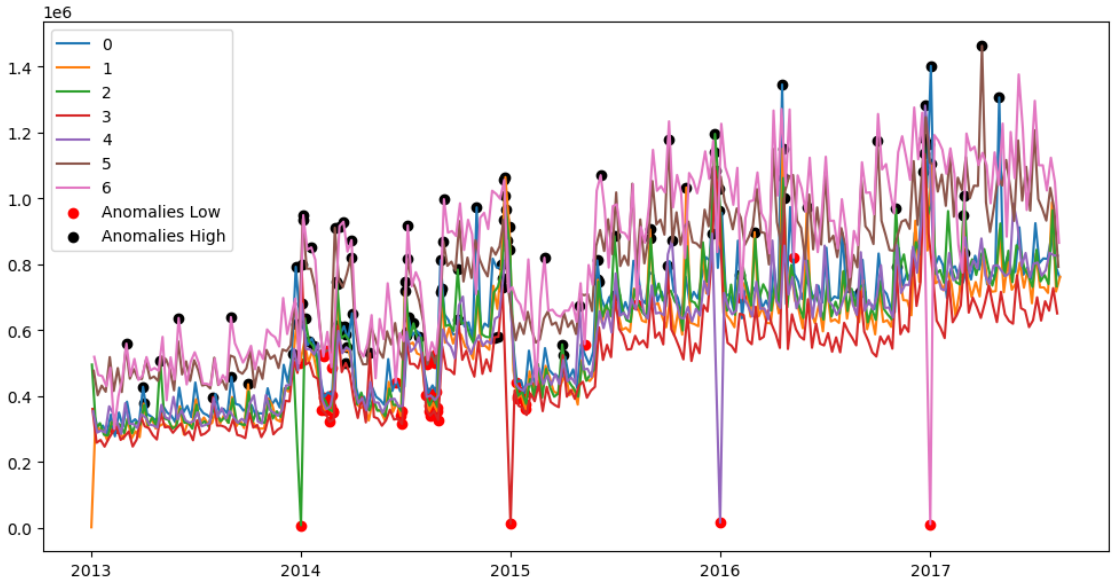
	id	date	store_nbr	product_type	sales	special_offer	year	month	day	dow	week_num	start_date
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	2013	1	1	1	1	2013-01-02
1	1	2013-01-01	1	BABY CARE	0.0	0	2013	1	1	1	1	NaT
2	2	2013-01-01	1	BEAUTY	0.0	0	2013	1	1	1	1	2013-01-02
3	3	2013-01-01	1	BEVERAGES	0.0	0	2013	1	1	1	1	2013-01-02
4	4	2013-01-01	1	BOOKS	0.0	0	2013	1	1	1	1	2016-10-13

1.3 Analysis 3 (wrt Day of Week + Anomaly Detection):

[14]:
```
daily_sales = data.copy()
daily_sales = daily_sales.groupby(['date','dow'])['sales'].sum().reset_index()

## rolling period
n= 5

daily_sales.sort_values(by = 'date', inplace=True)

#####
rolling_avg_prev = daily_sales.groupby('dow')['sales'].rolling(n, min_periods =␣
 ↪n-1, closed = 'left').mean().reset_index()
rolling_avg_prev.set_index('level_1', inplace=True)
rolling_avg_prev.rename(columns = {'sales': 'sales_prev'},inplace=True)
rolling_avg_prev = rolling_avg_prev[['sales_prev']]

daily_sales.sort_values(by = 'date', ascending=False, inplace=True)
```

```
rolling_avg_post = daily_sales.groupby('dow')['sales'].rolling(n, min_periods =␣
 ↪n-1, closed = 'left').mean().reset_index()
rolling_avg_post.set_index('level_1', inplace=True)
rolling_avg_post.rename(columns = {'sales': 'sales_post'},inplace=True)
rolling_avg_post = rolling_avg_post[['sales_post']]

daily_sales = pd.concat([daily_sales, rolling_avg_prev,␣
 ↪rolling_avg_post],axis=1)
daily_sales.sort_values(by = ['dow','date'], inplace=True)
####


daily_sales['sales_avg'] = (daily_sales['sales_prev'] +␣
 ↪daily_sales['sales_post'])/2
daily_sales['sales_factor'] = (daily_sales['sales'] - daily_sales['sales_avg'])/
 ↪daily_sales['sales_avg']
daily_sales['sales_pattern'] = np.where(daily_sales['sales_factor'] > 0.25,␣
 ↪'High', np.where(daily_sales['sales_factor'] < -0.25, 'Low', 'Normal'))

anomalies = daily_sales[daily_sales['sales_pattern'].isin(['High','Low'])]


daily_sales = pd.concat([daily_sales, rolling_avg_prev,␣
 ↪rolling_avg_post],axis=1)
daily_sales.sort_values(by = ['dow','date'], inplace=True)

plt.figure(figsize=(12, 6))

for day, row in daily_sales.groupby('dow'):
    plt.plot(row['date'], row['sales'], label=day)

plt.scatter(daily_sales[daily_sales['sales_pattern']=='Low']['date'],␣
 ↪daily_sales[daily_sales['sales_pattern']=='Low']['sales'], color='red',␣
 ↪label='Anomalies Low')
plt.scatter(daily_sales[daily_sales['sales_pattern']=='High']['date'],␣
 ↪daily_sales[daily_sales['sales_pattern']=='High']['sales'], color='black',␣
 ↪label='Anomalies High')
plt.legend()
plt.show()

# Inference:
# YoY increase in sales
# 2013 quite low sales
# 2014 lot of fluctuation in sales (unstable)
# 2015 first half sudden dip in sales
```

```
# starting from 2nd half 2015 sales became quite stable with minimal random␣
 ↪jumps
# saturday and sunday highest sales throughout
# lowest sales mostly on thursdays
```



1.3.1 Identifying Repetitive Sales Anomalies:

[15]:
```
anomalies['month'] = anomalies['date'].dt.month
anomalies['day'] = anomalies['date'].dt.day
anomalies.sort_values(by = 'date', inplace=True)

# Identify repeated anomalies
repeated_dates = anomalies.groupby(['month','day','sales_pattern']).size()

repeated_dates = repeated_dates[repeated_dates >= 3].reset_index()

repeated_dates[['month','day','sales_pattern']]

# High sales were observed at the start of the month in most months (maybe due␣
 ↪to salary)
# Low repetative sales on 1st Jan
# December faces a lot of rise in sales
```

[15]:

	month	day	sales_pattern
0	1	1	Low
1	3	1	High
2	4	1	High

```
3        5    1          High
4        9    1          High
5        9    2          High
6       10    1          High
7       11    3          High
8       12   18          High
9       12   21          High
10      12   22          High
11      12   23          High
12      12   24          High
13      12   30          High
```

1.3.2 Data Reduction Based on Analysis 1 & 3:

[16]:
```
data_backup = data.copy()
```

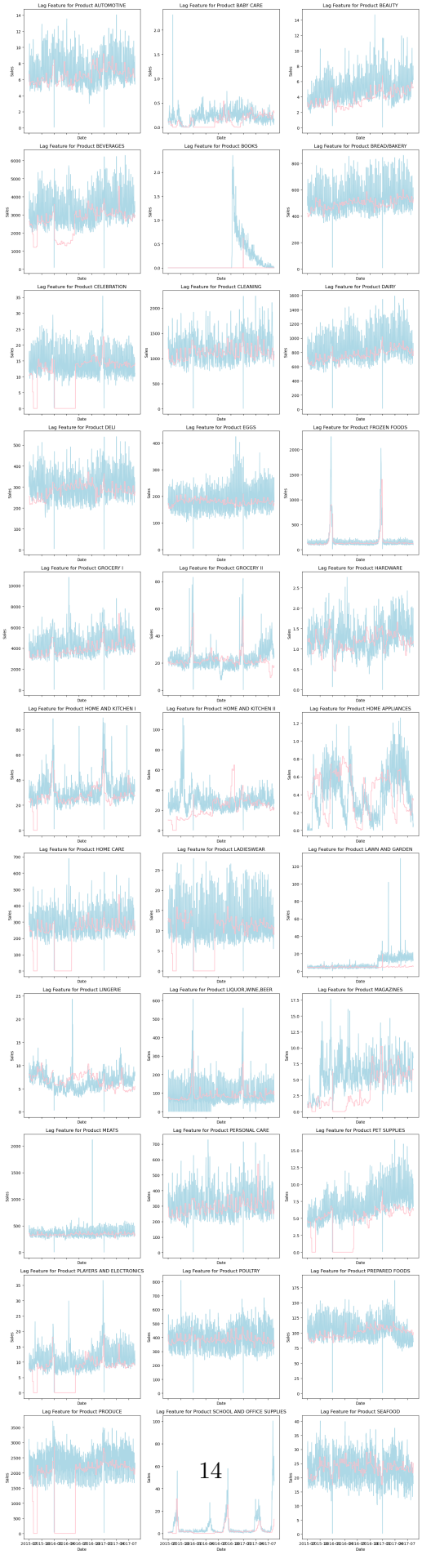
[17]:
```
# stable sales pattern observed after july 2015
# many stores didn't open till 2015
data = data[((data['year']==2015) & (data['month']>=7)) | (data['year']>=2016)]
data.head()
```

[17]:
```
               id       date  store_nbr product_type   sales  special_offer  \
1619838  1619838 2015-07-01          1   AUTOMOTIVE     5.0              0
1619839  1619839 2015-07-01          1    BABY CARE     0.0              0
1619840  1619840 2015-07-01          1       BEAUTY     5.0              1
1619841  1619841 2015-07-01          1    BEVERAGES  2638.0              2
1619842  1619842 2015-07-01          1        BOOKS     0.0              0

         year  month  day  dow  week_num start_date
1619838  2015      7    1    2        27 2013-01-02
1619839  2015      7    1    2        27        NaT
1619840  2015      7    1    2        27 2013-01-02
1619841  2015      7    1    2        27 2013-01-02
1619842  2015      7    1    2        27 2016-10-13
```

1.3.3 Understanding Lag Features

[18]:
```
df = data_backup.copy()
lag_data = df.groupby(['product_type','store_nbr','year','week_num'])['sales'].
 ↪mean().reset_index(name='lag_yoy_sales')
lag_data['year'] = lag_data['year']+1
df = pd.merge(df, lag_data, how='left', on =␣
 ↪['product_type','store_nbr','year','week_num'])
df = df[df['date']>='2015-07-01']

unique_products = df['product_type'].unique()
```

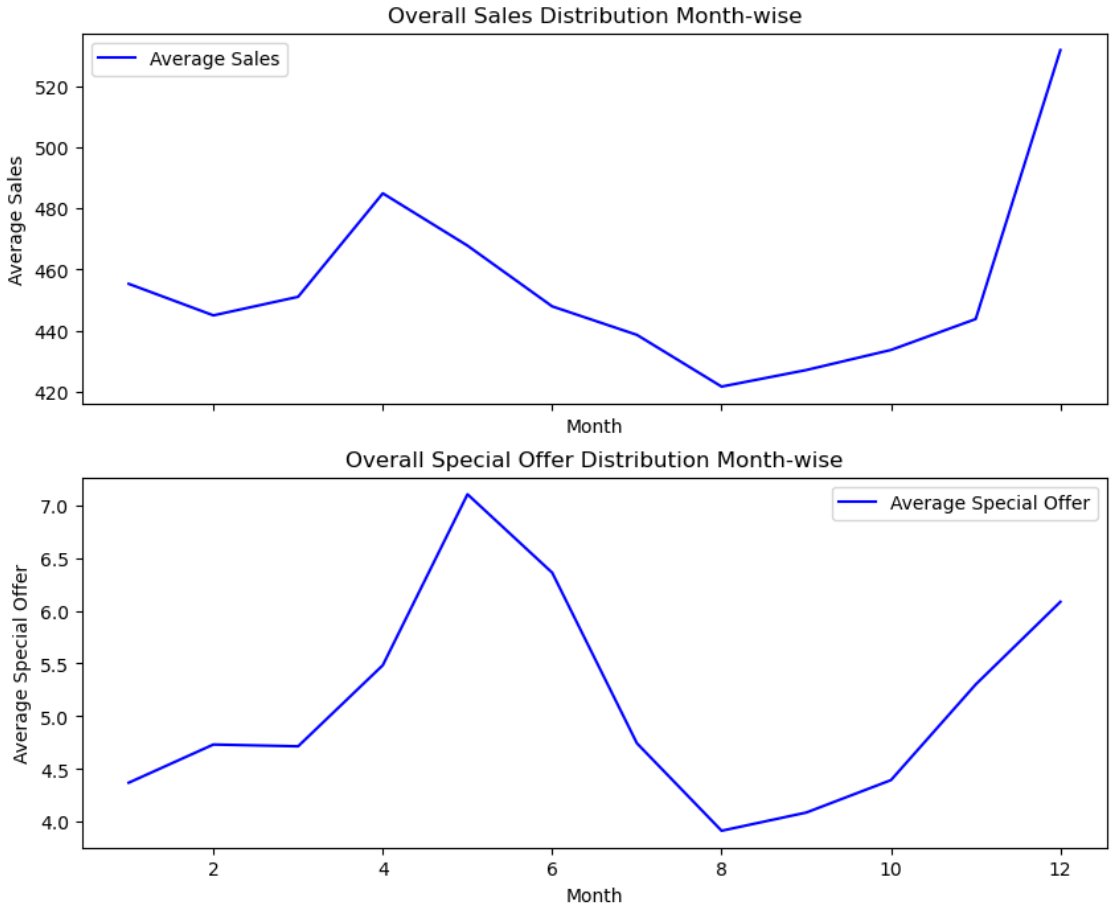
```
num_rows = math.ceil(len(unique_products) / 3)
num_cols = min(len(unique_products), 3)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)

for i, product in enumerate(unique_products):
    row_idx = i // 3
    col_idx = i % 3

    product_data = df[df['product_type'] == product].
 ↪groupby(['date'])[['sales','lag_yoy_sales']].mean()
    product_data = product_data.reset_index()


    axes[row_idx, col_idx].plot(product_data['date'], product_data['sales'],␣
 ↪label=f'Mean Sales - {product}', c = 'lightblue')
    axes[row_idx, col_idx].plot(product_data['date'],␣
 ↪product_data['lag_yoy_sales'], label=f'Previous Year Same Week No. Mean␣
 ↪Sales - {product}', c = 'pink')
    axes[row_idx, col_idx].set_xlabel('Date')
    axes[row_idx, col_idx].set_ylabel('Sales')
    axes[row_idx, col_idx].set_title(f'Lag Feature for Product {product}')

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


Lag Feature for Product AUTOMOTIVE
Lag Feature for Product BABY CARE
Lag Feature for Product BEAUTY
Lag Feature for Product BEVERAGES
Lag Feature for Product BOOKS
Lag Feature for Product BREAD/BAKERY
Lag Feature for Product CELEBRATION
Lag Feature for Product CLEANING
Lag Feature for Product DAIRY
Lag Feature for Product DELI
Lag Feature for Product EGGS
Lag Feature for Product FROZEN FOODS
Lag Feature for Product GROCERY I
Lag Feature for Product GROCERY II
Lag Feature for Product HARDWARE
Lag Feature for Product HOME AND KITCHEN I
Lag Feature for Product HOME AND KITCHEN II
Lag Feature for Product HOME APPLIANCES
Lag Feature for Product HOME CARE
Lag Feature for Product LADIESWEAR
Lag Feature for Product LAWN AND GARDEN
Lag Feature for Product LINGERIE
Lag Feature for Product LIQUOR,WINE,BEER
Lag Feature for Product MAGAZINES
Lag Feature for Product MEATS
Lag Feature for Product PERSONAL CARE
Lag Feature for Product PET SUPPLIES
Lag Feature for Product PLAYERS AND ELECTRONICS
Lag Feature for Product POULTRY
Lag Feature for Product PREPARED FOODS
Lag Feature for Product PRODUCE
Lag Feature for Product SCHOOL AND OFFICE SUPPLIES
Lag Feature for Product SEAFOOD
Sales
Date

1.4 Analysis 4 (wrt Special Offer):

1.4.1 Monthly Sales vs Special Offer Distribution

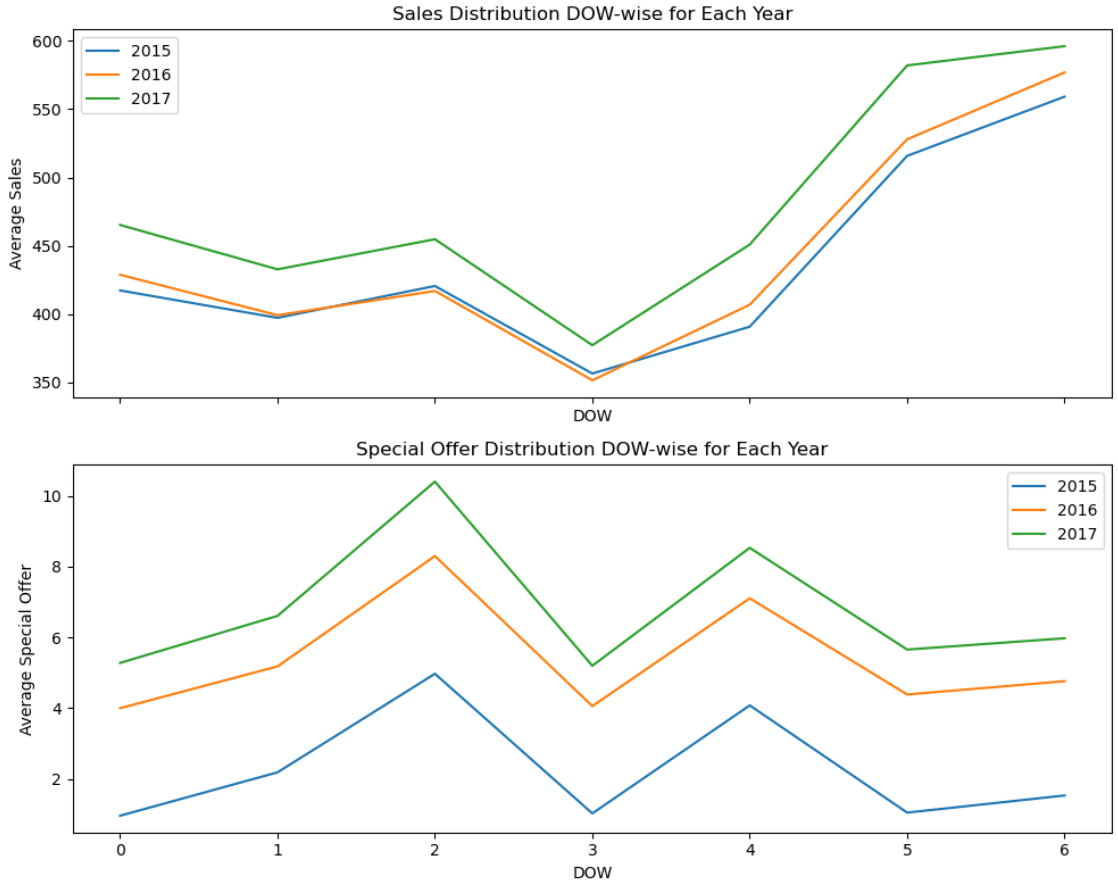
[20]:
```
df = data.copy()
monthly_data = df.groupby(['month']).agg({'sales': 'mean', 'special_offer':␣
 ↪'mean'}).reset_index()

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=True)

ax1.plot(monthly_data['month'], monthly_data['sales'], label='Average Sales',␣
 ↪color='blue')
ax1.set_xlabel('Month')
ax1.set_ylabel('Average Sales')
ax1.set_title('Overall Sales Distribution Month-wise')
ax1.legend()

ax2.plot(monthly_data['month'], monthly_data['special_offer'], label='Average␣
 ↪Special Offer', color='blue')
ax2.set_xlabel('Month')
ax2.set_ylabel('Average Special Offer')
ax2.set_title('Overall Special Offer Distribution Month-wise')
ax2.legend()

plt.show()

# little similarity observed b/w sales and special offer over months
# jan,feb, and aug have a major sales drop
# sales hike in may and december
# high offers during april, may, june, november, and december
# best offers offered around festival christmas and easter
```



1.4.2 DOW-wise Sales vs Special Offer distribution for each year

[21]:
```
df = data.copy()
dow_data = df.groupby(['year', 'dow']).agg({'sales': 'mean', 'special_offer':␣
↪'mean'}).reset_index()

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=True)


for year in dow_data['year'].unique():
    year_data = dow_data[dow_data['year'] == year]
    ax1.plot(year_data['dow'], year_data['sales'], label=str(year))


ax1.set_xlabel('DOW')
ax1.set_ylabel('Average Sales')
ax1.set_title('Sales Distribution DOW-wise for Each Year')
ax1.legend()
```

```
for year in dow_data['year'].unique():
    year_data = dow_data[dow_data['year'] == year]
    ax2.plot(year_data['dow'], year_data['special_offer'], label=str(year))


ax2.set_xlabel('DOW')
ax2.set_ylabel('Average Special Offer')
ax2.set_title('Special Offer Distribution DOW-wise for Each Year')
ax2.legend()

plt.tight_layout()
plt.show()

# very similar pattern observed for dow
# weekend has higher sales compared to weekdays
# wednesdays have highest offers
# special offer not much effective, dow is much more effective wrt sales
```

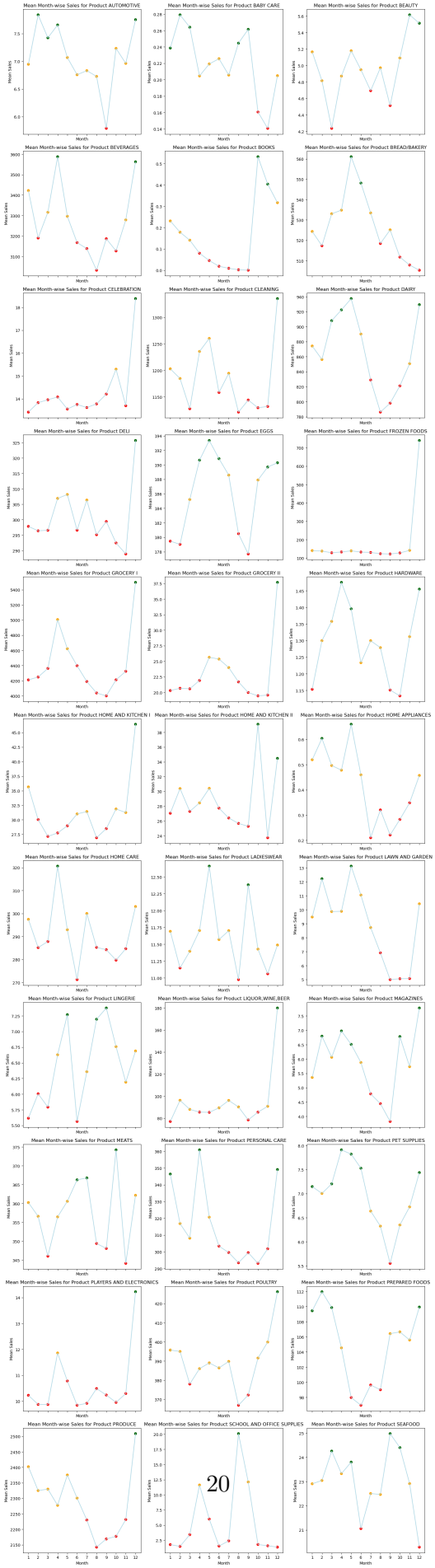


2 EDA : Product Specific Pattern + Clustering Possibility

2.1 Analysis 1 (Product-wise Monthly Sales Distribution):

[22]:
```
df = data.copy()
unique_products = df['product_type'].unique()

num_rows = math.ceil(len(unique_products) / 3)
num_cols = min(len(unique_products), 3)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)


for i, product in enumerate(unique_products):
    row_idx = i // 3
    col_idx = i % 3

    product_data = df[df['product_type'] == product].
 ↪groupby(['month','product_type'])['sales'].mean()
    product_data = product_data.reset_index()

    X = product_data['sales'].values

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X.reshape(-1, 1))

    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    y_pred = kmeans.fit_predict(X_scaled)

    ordered_labels = np.argsort(kmeans.cluster_centers_.sum(axis=1))
    label_mapping = {old_label: new_label for new_label, old_label in␣
 ↪enumerate(ordered_labels)}
    y_pred = pd.Series(y_pred).map(label_mapping)



    color_map = {0:'red', 1:'orange', 2:'darkgreen'}
    color_label = pd.Series(y_pred).map(color_map)

    axes[row_idx, col_idx].plot(product_data['month'], product_data['sales'],␣
 ↪label=f'Mean Sales - {product}', c = 'lightblue')
    axes[row_idx, col_idx].scatter(product_data['month'],␣
 ↪product_data['sales'], label=f'Mean Sales - {product}', c = color_label)
    axes[row_idx, col_idx].set_xticks(product_data['month'])
    axes[row_idx, col_idx].set_xlabel('Month')
```

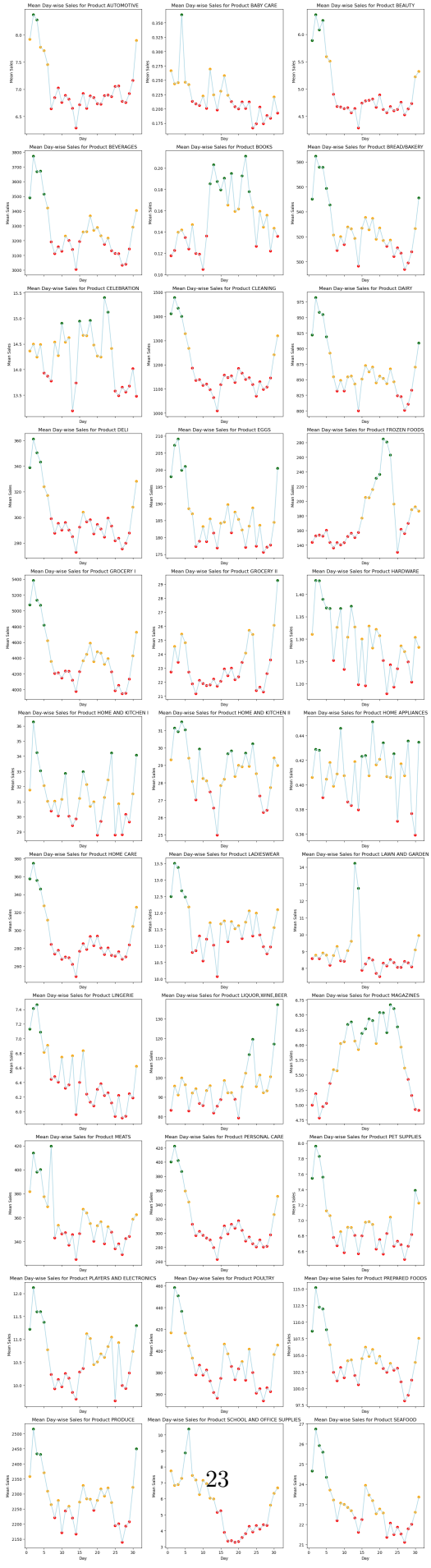
```
    axes[row_idx, col_idx].set_ylabel('Mean Sales')
    axes[row_idx, col_idx].set_title(f'Mean Month-wise Sales for Product␣
 ↪{product}')

plt.tight_layout()
plt.show()

# Different products have different sales range as observed in EDA 1
# Clustering of products based on their monthly avg sales
# Months with similar avg sales grouped together
# 3 clusters have been created overall (High, Medium, and Low sales category)
# Can be used for modelling
```


Mean Month-wise Sales for Product AUTOMOTIVE
Mean Month-wise Sales for Product BABY CARE
Mean Month-wise Sales for Product BEAUTY
Mean Month-wise Sales for Product BEVERAGES
Mean Month-wise Sales for Product BOOKS
Mean Month-wise Sales for Product BREAD/BAKERY
Mean Month-wise Sales for Product CELEBRATION
Mean Month-wise Sales for Product CLEANING
Mean Month-wise Sales for Product DAIRY
Mean Month-wise Sales for Product DELI
Mean Month-wise Sales for Product EGGS
Mean Month-wise Sales for Product FROZEN FOODS
Mean Month-wise Sales for Product GROCERY I
Mean Month-wise Sales for Product GROCERY II
Mean Month-wise Sales for Product HARDWARE
Mean Month-wise Sales for Product HOME AND KITCHEN I
Mean Month-wise Sales for Product HOME AND KITCHEN II
Mean Month-wise Sales for Product HOME APPLIANCES
Mean Month-wise Sales for Product HOME CARE
Mean Month-wise Sales for Product LADIESWEAR
Mean Month-wise Sales for Product LAWN AND GARDEN
Mean Month-wise Sales for Product LINGERIE
Mean Month-wise Sales for Product LIQUOR,WINE,BEER
Mean Month-wise Sales for Product MAGAZINES
Mean Month-wise Sales for Product MEATS
Mean Month-wise Sales for Product PERSONAL CARE
Mean Month-wise Sales for Product PET SUPPLIES
Mean Month-wise Sales for Product PLAYERS AND ELECTRONICS
Mean Month-wise Sales for Product POULTRY
Mean Month-wise Sales for Product PREPARED FOODS
Mean Month-wise Sales for Product PRODUCE
Mean Month-wise Sales for Product SCHOOL AND OFFICE SUPPLIES
Mean Month-wise Sales for Product SEAFOOD
Mean Sales
Month

2.2 Analysis 2 (Product-wise Day of Month Sales Distribution):

[23]:
```
df = data.copy()
unique_products = df['product_type'].unique()

num_rows = math.ceil(len(unique_products) / 3)
num_cols = min(len(unique_products), 3)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)

for i, product in enumerate(unique_products):
    row_idx = i // 3
    col_idx = i % 3

    product_data = df[df['product_type'] == product].
 ↪groupby(['day','product_type'])['sales'].mean()
    product_data = product_data.reset_index()

    X = product_data['sales'].values

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X.reshape(-1, 1))

    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    y_pred = kmeans.fit_predict(X_scaled)

    ordered_labels = np.argsort(kmeans.cluster_centers_.sum(axis=1))
    label_mapping = {old_label: new_label for new_label, old_label in␣
 ↪enumerate(ordered_labels)}
    y_pred = pd.Series(y_pred).map(label_mapping)

    color_map = {0:'red', 1:'orange', 2:'darkgreen'}
    color_label = pd.Series(y_pred).map(color_map)

    axes[row_idx, col_idx].plot(product_data['day'], product_data['sales'],␣
 ↪label=f'Mean Sales - {product}', c = 'lightblue')
    axes[row_idx, col_idx].scatter(product_data['day'], product_data['sales'],␣
 ↪label=f'Mean Sales - {product}', c = color_label)
    axes[row_idx, col_idx].set_xlabel('Day')
    axes[row_idx, col_idx].set_ylabel('Mean Sales')
    axes[row_idx, col_idx].set_title(f'Mean Day-wise Sales for Product␣
 ↪{product}')
```

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

# Clustering of products based on their daily avg sales as observed earlier during EDA 1
# Days with similar avg sales grouped together
# Most products have highest sales at the start of the month
# Some have well defined clusters (can be used for modelling) while some do not
# 3 clusters have been created overall (High, Medium, and Low sales category)
```


Mean Day-wise Sales for Product AUTOMOTIVE
Mean Day-wise Sales for Product BABY CARE
Mean Day-wise Sales for Product BEAUTY
Mean Day-wise Sales for Product BEVERAGES
Mean Day-wise Sales for Product BOOKS
Mean Day-wise Sales for Product BREAD/BAKERY
Mean Day-wise Sales for Product CELEBRATION
Mean Day-wise Sales for Product CLEANING
Mean Day-wise Sales for Product DAIRY
Mean Day-wise Sales for Product DELI
Mean Day-wise Sales for Product EGGS
Mean Day-wise Sales for Product FROZEN FOODS
Mean Day-wise Sales for Product GROCERY I
Mean Day-wise Sales for Product GROCERY II
Mean Day-wise Sales for Product HARDWARE
Mean Day-wise Sales for Product HOME AND KITCHEN I
Mean Day-wise Sales for Product HOME AND KITCHEN II
Mean Day-wise Sales for Product HOME APPLIANCES
Mean Day-wise Sales for Product HOME CARE
Mean Day-wise Sales for Product LADIESWEAR
Mean Day-wise Sales for Product LAWN AND GARDEN
Mean Day-wise Sales for Product LINGERIE
Mean Day-wise Sales for Product LIQUOR,WINE,BEER
Mean Day-wise Sales for Product MAGAZINES
Mean Day-wise Sales for Product MEATS
Mean Day-wise Sales for Product PERSONAL CARE
Mean Day-wise Sales for Product PET SUPPLIES
Mean Day-wise Sales for Product PLAYERS AND ELECTRONICS
Mean Day-wise Sales for Product POULTRY
Mean Day-wise Sales for Product PREPARED FOODS
Mean Day-wise Sales for Product PRODUCE
Mean Day-wise Sales for Product SCHOOL AND OFFICE SUPPLIES
Mean Day-wise Sales for Product SEAFOOD
Mean Sales
Day

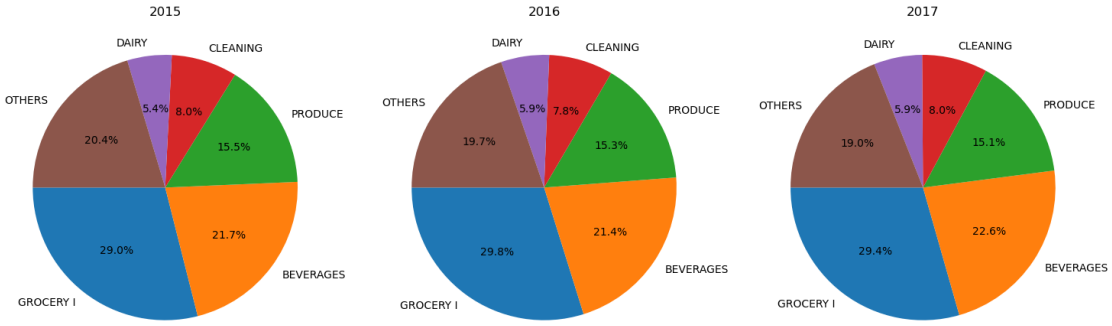
2.3 Analysis 3 (Most Selling Products (MSP)):

[24]:
```
fig = px.bar(data.groupby(['year', 'product_type']).sales.sum().
 ↪sort_values(ascending=False).reset_index(),
             x='product_type', y='sales', color='product_type',␣
 ↪facet_row='year',
             title="Yearly Contribution of Each Product Type to Total Sales",
             labels={'sales': 'Mean Sales', 'product_type': 'Product Type'},
             category_orders={'year': sorted(data['year'].unique())})

fig.update_layout(height=1400)

fig.show()

# HOMECARE, BOOKS,  was not present in 2013 and were introduced in 2014
# Majority sales contribution from Grocery & Beverages
# We need to predict correctly the products that majorly contribute to sales
```

2.3.1 MSP: Products with more than 5% contribution to total yearly sales

[25]:
```
grouped_data = data.groupby(['year', 'product_type']).sales.mean().reset_index()

grouped_data['percentage_contribution'] = grouped_data.groupby('year')['sales'].
 ↪transform(lambda x: (x / x.sum()) * 100)
msp = pd.pivot_table(grouped_data, values='percentage_contribution',␣
 ↪index='product_type', columns='year')
msp = msp.sort_values(by=2017, ascending = False)
msp = msp[(msp[2015]>5) | (msp[2016]>5) | (msp[2017]>5)]

msp_list = list(msp.index)

other_products = pd.DataFrame(100-msp.sum()).T
other_products['product_type'] = 'OTHERS'
other_products.set_index('product_type', inplace=True, drop=True)
msp = pd.concat([msp,other_products])


# plot pie charts
def pie_chart(data, title, axs):
    sns.set_palette("Set3")

    axes[axs].pie(data, labels=data.index, autopct='%1.1f%%', startangle=180)
    axes[axs].set_title(title)
```

```
ig, axes = plt.subplots(1, 3, figsize=(15, 5))
pie_chart(msp[2015],'2015',0)
pie_chart(msp[2016],'2016',1)
pie_chart(msp[2017],'2017',2)
plt.tight_layout()
plt.show()

# Approx. 80% contribution from these 5 products and 50% only from grocery I␣
 ↪and beverages
```

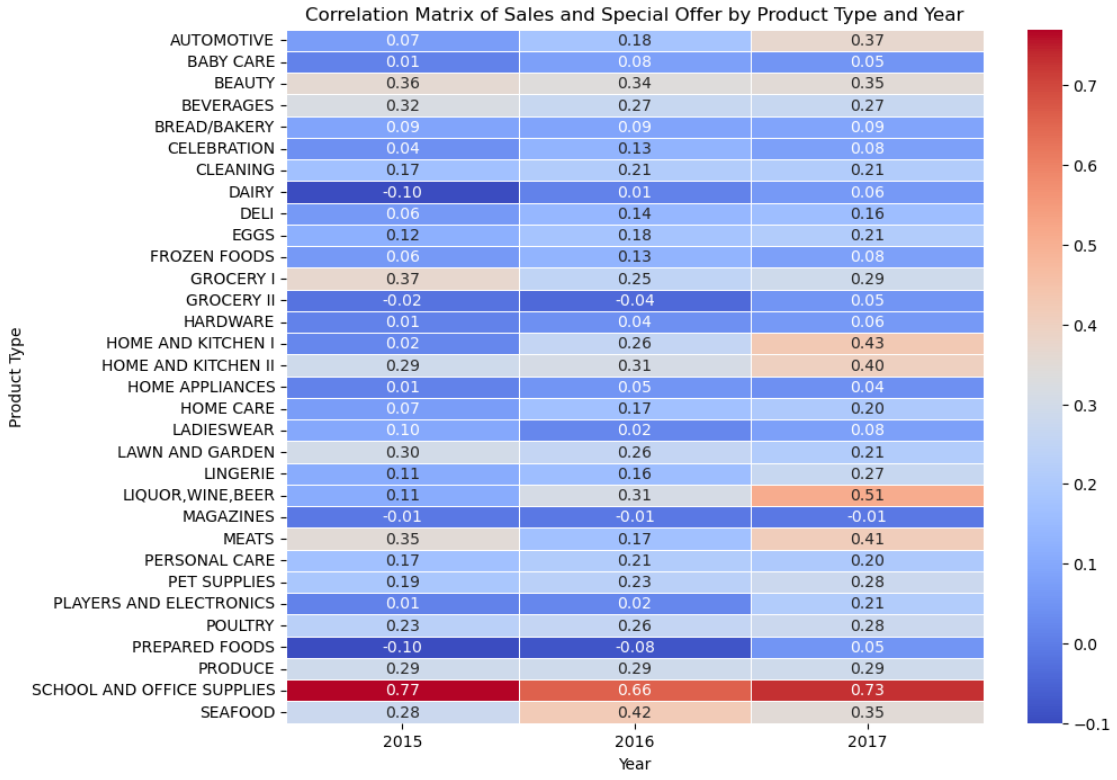


2.4 Analysis 4 (Product-wise Special Offers Distribution):

2.4.1 Correlation Matrix b/w special offer on products and sales

[26]:
```
correlation_data = []

for product_type in data['product_type'].unique():
    for year in data['year'].unique():
        subset_data = data[(data['product_type'] == product_type) &␣
 ↪(data['year'] == year)]
        correlation_value = subset_data[['special_offer', 'sales']].corr().
 ↪iloc[0, 1]


        correlation_data.append({
            'product_type': product_type,
            'year': year,
            'correlation': correlation_value
        })


correlation_df =  pd.DataFrame(correlation_data)
```

```
table_data = pd.pivot_table(correlation_df, values='correlation',␣
 ↪index='product_type', columns='year')

table_data = table_data.round(2)


plt.figure(figsize=(10, 8))
sns.heatmap(table_data, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix of Sales and Special Offer by Product Type and␣
 ↪Year')
plt.xlabel('Year')
plt.ylabel('Product Type')
plt.show()

# Not all offers on products lead to increase in sales
# special offer feature can be used for modelling only for few products that␣
 ↪have dependencies on special offer for sales hike
# school and office supplies have a remarkable correlation
```



2.4.2 Products with > 20 % correlation of sales & special offer

[27]:
```
table_data = table_data[ (table_data[2015]>0.2) & (table_data[2016]>0.2) &␣
 ↪(table_data[2017]>0.2)]
special_offer_products = table_data.index.tolist()
table_data
# Special offer feature will be used only for these products
```

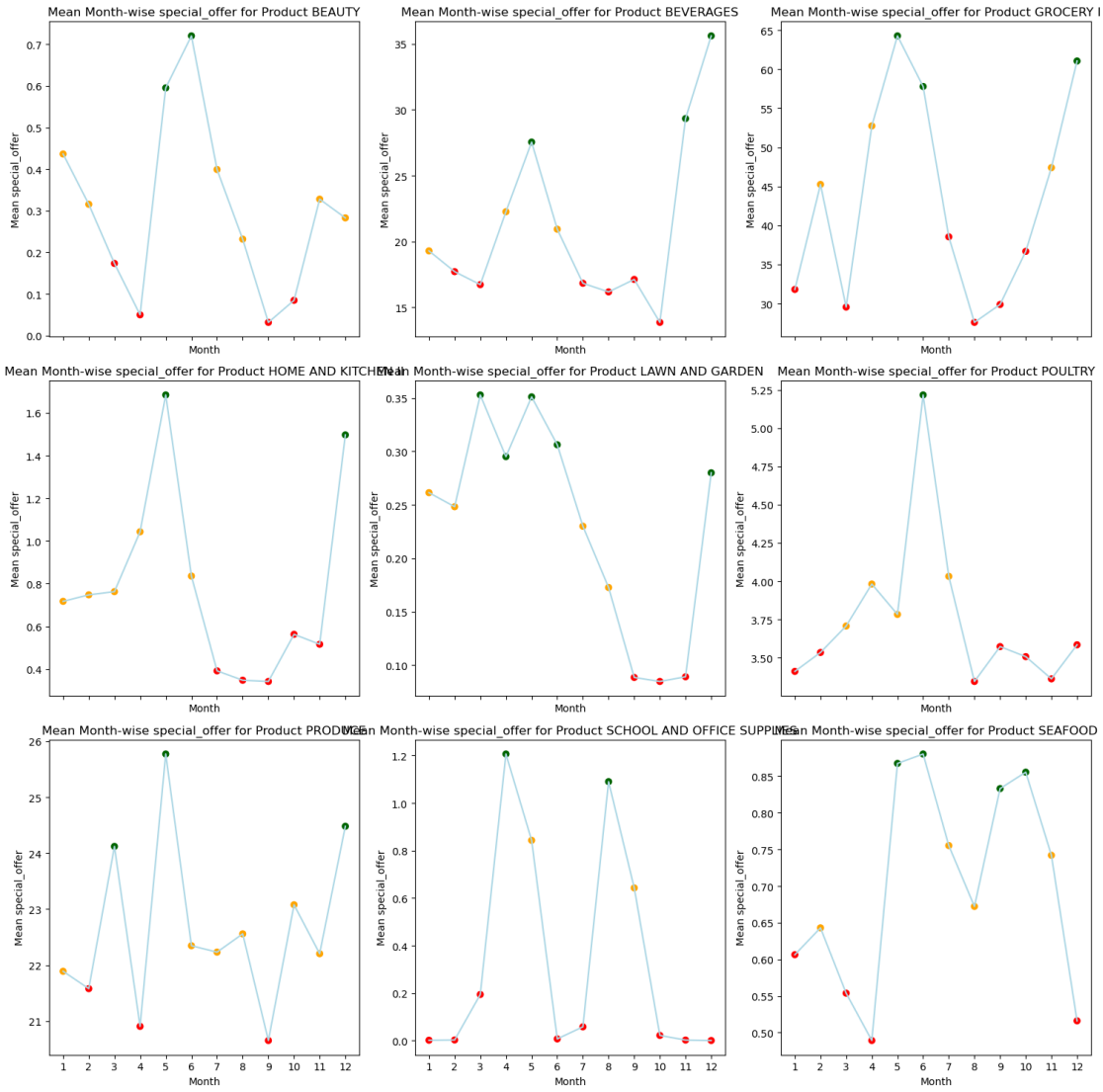
[27]:

year	2015	2016	2017
product_type			
BEAUTY	0.36	0.34	0.35
BEVERAGES	0.32	0.27	0.27
GROCERY I	0.37	0.25	0.29
HOME AND KITCHEN II	0.29	0.31	0.40
LAWN AND GARDEN	0.30	0.26	0.21
POULTRY	0.23	0.26	0.28
PRODUCE	0.29	0.29	0.29
SCHOOL AND OFFICE SUPPLIES	0.77	0.66	0.73
SEAFOOD	0.28	0.42	0.35

2.4.3 Product-wise Monthly Special Offers Distribution Clusters:

[28]:
```
df = data.copy()
df = df[df['product_type'].isin(special_offer_products)]
unique_products = df['product_type'].unique()

num_rows = math.ceil(len(unique_products) / 3)
num_cols = min(len(unique_products), 3)


fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)

product_final_data = pd.DataFrame()

for i, product in enumerate(unique_products):
    row_idx = i // 3
    col_idx = i % 3

    product_data = df[df['product_type'] == product].
 ↪groupby(['month','product_type'])['special_offer'].mean()
    product_data = product_data.reset_index()

    X = product_data['special_offer'].values

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X.reshape(-1, 1))
```

```
    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    y_pred = kmeans.fit_predict(X_scaled)

    ordered_labels = np.argsort(kmeans.cluster_centers_.sum(axis=1))
    label_mapping = {old_label: new_label for new_label, old_label in␣
 ↪enumerate(ordered_labels)}
    y_pred = pd.Series(y_pred).map(label_mapping)


    color_map = {0:'red', 1:'orange', 2:'darkgreen'}
    color_label = pd.Series(y_pred).map(color_map)

    axes[row_idx, col_idx].plot(product_data['month'],␣
 ↪product_data['special_offer'], label=f'Mean special_offer - {product}', c =␣
 ↪'lightblue')
    axes[row_idx, col_idx].scatter(product_data['month'],␣
 ↪product_data['special_offer'], label=f'Mean special_offer - {product}', c =␣
 ↪color_label)
    axes[row_idx, col_idx].set_xticks(product_data['month'])
    axes[row_idx, col_idx].set_xlabel('Month')
    axes[row_idx, col_idx].set_ylabel('Mean special_offer')
    axes[row_idx, col_idx].set_title(f'Mean Month-wise special_offer for␣
 ↪Product {product}')

plt.tight_layout()
plt.show()

# Books have no offer
# High offers on most products around april-may as observed in EDA 1
```

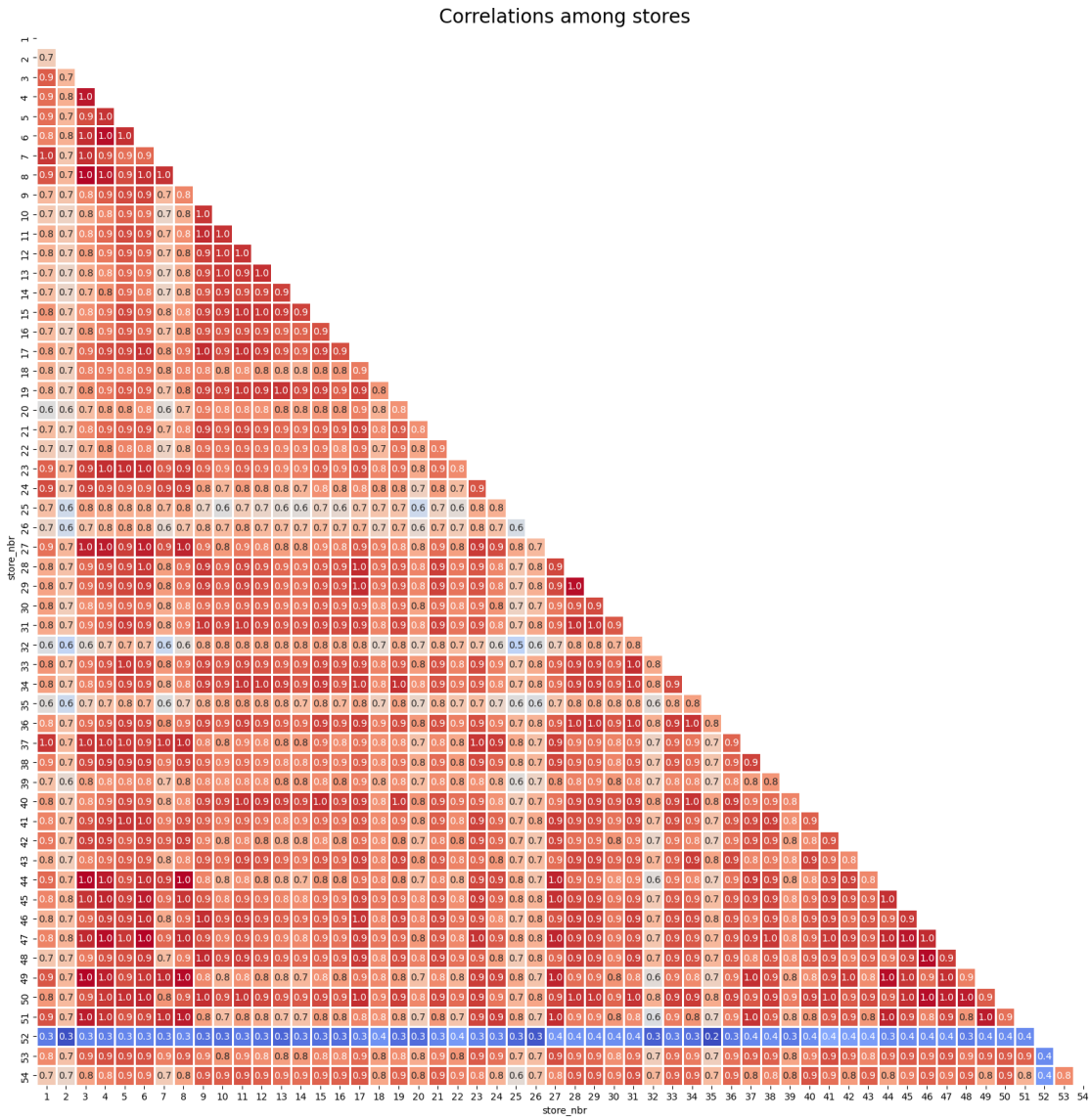


3 EDA : Store Specific Pattern + Clustering Possibility

3.1 Analysis 1 (Correlation b/w Sales of Stores):

[29]:
```
data_cpy = data.copy()
data_cpy['group_index'] = 1
data_cpy['group_index'] = data_cpy.groupby('store_nbr')['group_index'].cumsum().
 ↪values
data_cpy = pd.pivot(data_cpy, index='group_index', columns='store_nbr',␣
 ↪values='sales')
data_cpy = data_cpy.corr()
```

```
plt.figure(figsize=(20, 20))
sns.heatmap(
    data_cpy,
    annot=True,
    fmt=".1f",
    cmap="coolwarm",
    square=True,
    mask=np.triu(data_cpy),
    linewidths=1,
    cbar=False,
)
plt.title("Correlations among stores", fontsize=20)
plt.show()

# Most of the stores are correlated, hence can be clustered
# outlet 52 has the least correlation as it was opened in 2017 (need to be␣
 ↪careful while modelling)
# Since, different products follow different sales ranges and trends, and also␣
 ↪our aim is inventory management, if would be a wiser decision to cluster␣
 ↪stores for each product separately
```

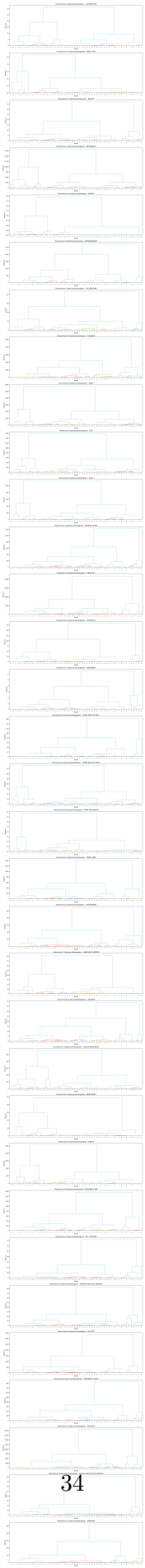


3.1.1 Product-wise Store Clustering

[30]:
```
df = data.copy()
unique_products = df['product_type'].unique()
num_rows = len(unique_products)

cluster_info_df = pd.DataFrame(columns=['Product', 'Cluster_Count'])

fig, axes = plt.subplots(num_rows, 1, figsize=(15, 5*num_rows), sharex=True)


for i, product in enumerate(unique_products):
```

```
    product_data = df[df['product_type'] == product]

    pivot_df = pd.pivot_table(product_data, values='sales', index='store_nbr',␣
↪fill_value=0)

    distances = pdist(pivot_df)
    linkage_matrix = hierarchy.linkage(distances, method='ward')


    height_threshold = 0.05 * np.max(linkage_matrix[:, 2])
    clusters = hierarchy.fcluster(linkage_matrix, height_threshold,␣
↪criterion='distance')


    store_clusters = dict(zip(pivot_df.index, clusters))

    product_data.loc[product_data['product_type'] ==␣
↪product,'product_store_cluster'] = product_data.
↪loc[product_data['product_type'] == product,'store_nbr'].map(store_clusters)

    avg_sales_per_cluster = product_data.
↪groupby('product_store_cluster')['sales'].mean()
    sorted_clusters = avg_sales_per_cluster.sort_values().index

    cluster_order_mapping = {cluster: new_cluster_num for new_cluster_num,␣
↪cluster in enumerate(sorted_clusters, start=1)}

    store_clusters = {store: cluster_order_mapping[store_clusters[store]] for␣
↪store in store_clusters}

    num_clusters = len(set(clusters))
    store_count_per_cluster = pd.Series(clusters).value_counts()

    cluster_info = pd.DataFrame({
        'Product': [product],
        'Cluster_Count': [num_clusters]
    })


    cluster_info_df = pd.concat([cluster_info_df, cluster_info],␣
↪ignore_index=True)

    hierarchy.dendrogram(linkage_matrix, labels=pivot_df.index,␣
↪orientation='top', distance_sort='descending',␣
↪color_threshold=height_threshold, ax=axes[i])
    axes[i].set_title(f'Hierarchical Clustering Dendrogram - {product}')
```

```
    axes[i].set_xlabel('Stores')
    axes[i].set_ylabel('Distance')

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



3.1.2 Stores Clusters Info

[31]:
```
cluster_info_df
```

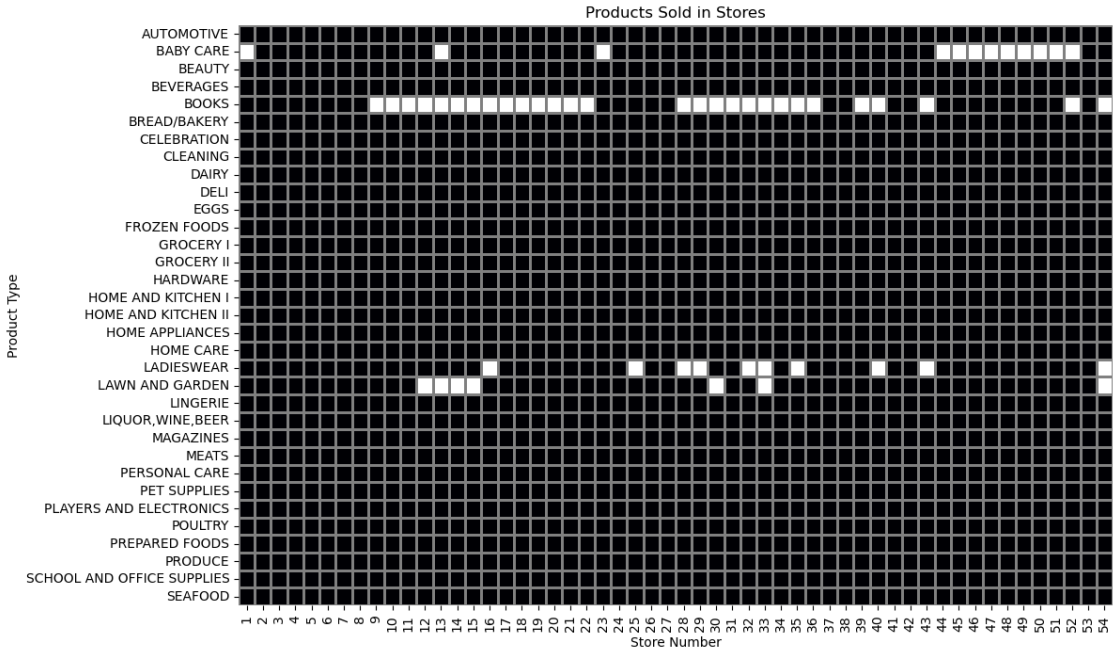
[31]:

	Product	Cluster_Count
0	AUTOMOTIVE	12
1	BABY CARE	9
2	BEAUTY	11
3	BEVERAGES	12
4	BOOKS	8
5	BREAD/BAKERY	11
6	CELEBRATION	11
7	CLEANING	11
8	DAIRY	10
9	DELI	11
10	EGGS	11
11	FROZEN FOODS	12
12	GROCERY I	10
13	GROCERY II	10
14	HARDWARE	12
15	HOME AND KITCHEN I	11
16	HOME AND KITCHEN II	11
17	HOME APPLIANCES	9
18	HOME CARE	11
19	LADIESWEAR	10
20	LAWN AND GARDEN	8
21	LINGERIE	10
22	LIQUOR,WINE,BEER	11
23	MAGAZINES	10
24	MEATS	10
25	PERSONAL CARE	10
26	PET SUPPLIES	9
27	PLAYERS AND ELECTRONICS	10
28	POULTRY	9
29	PREPARED FOODS	10
30	PRODUCE	9
31	SCHOOL AND OFFICE SUPPLIES	8
32	SEAFOOD	7

3.2 Analysis 2 (Product unavailability wrt stores):

3.2.1 Products never sold in a store:

[32]:
```
sales_data = data.groupby(['store_nbr', 'product_type'])['sales'].sum().
 ↪reset_index()

heatmap_data = sales_data.pivot_table(index='product_type',␣
 ↪columns='store_nbr', values='sales', fill_value=0)

zero_sales_mask = heatmap_data == 0

plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data.notnull(), cmap='magma', cbar=False, annot=False,␣
 ↪vmin=0, vmax=heatmap_data.max().max(), linewidths=1, linecolor='gray',␣
 ↪mask=zero_sales_mask)
plt.title('Products Sold in Stores')
plt.xlabel('Store Number')
plt.ylabel('Product Type')
plt.show()

# Many stores do not sell books
# Most of the stores sell all products
```



3.2.2 Temporary inactive products on store level (based on recent sales):

[33]:
```
df = data.copy()

recent_sales = df[(df['date'] > df['date'].max() - pd.DateOffset(days=15))]
recent_sales = recent_sales.drop(columns = ['date'])

zero_sales = recent_sales.groupby(['store_nbr', 'product_type'])['sales'].sum().
 ↪reset_index()
zero_sales = zero_sales[zero_sales['sales']==0][['product_type','store_nbr']]
zero_sales

# Needs to be handled during modelling
# Prediction for unavailable products will be set to 0
```

[33]:

	product_type	store_nbr
1	BABY CARE	1
4	BOOKS	1
17	HOME APPLIANCES	1
31	SCHOOL AND OFFICE SUPPLIES	1
34	BABY CARE	2
…	…	…
1753	BOOKS	54
1768	LADIESWEAR	54
1769	LAWN AND GARDEN	54
1775	PET SUPPLIES	54
1780	SCHOOL AND OFFICE SUPPLIES	54

[135 rows x 2 columns]

4 EDA : Random/Special Events (Highlighting Outliers)

[34]:
```
# Based on basic anomaly detection, we reduced the dataset to contain data from␣
 ↪july 2015
# Now, we will be delving deeper into remaining anomalies (post july 2015)
# This study will be product based as our model will be on product level
```

4.1 Analysis 1: Unusual sales jump/dip

[35]:
```
data_cpy = data.copy()
df = data_cpy[data_cpy['sales']!=0]
df = df.groupby(['product_type','month'])['sales'].mean()
df = df.reset_index().rename(columns = {'sales':'mean_sales'})

data_cpy = pd.merge(data_cpy, df, on = ['product_type','month'])
```

```
data_cpy['sales_outlier'] = (abs((data_cpy['sales']-data_cpy['mean_sales'])/
 ↪data_cpy['mean_sales'])>15).astype(int)

print('Percent outliers in terms of sales:',␣
 ↪round(len(data_cpy[data_cpy['sales_outlier']==1])/len(data_cpy)*100,2))

# outliers to be calculated and handled in the training dataset
```

```
Percent outliers in terms of sales: 0.01
```

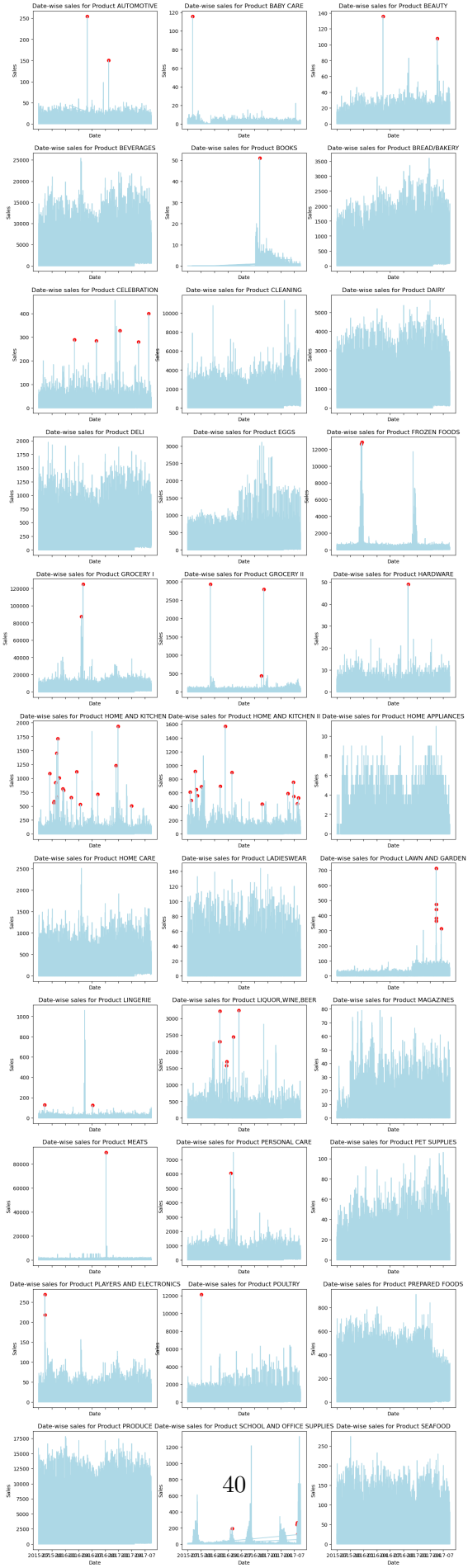
[36]:
```
data_c = data.copy()
# df = data_c[data_c['sales']!=0]
df = data_c.groupby(['product_type','month'])['special_offer'].mean()
df = df.reset_index().rename(columns = {'special_offer':'mean_special_offer'})

data_c = pd.merge(data_c, df, on = ['product_type','month'])
data_c['offer_outlier'] =␣
 ↪(abs((data_c['special_offer']-data_c['mean_special_offer'])/
 ↪data_c['mean_special_offer'])>2).astype(int)

print('Percent outliers in terms of special offers:',␣
 ↪round(len(data_c[data_c['offer_outlier']==1])/len(data_c)*100,2))
# outliers to be calculated and handled in the training dataset
```

```
Percent outliers in terms of special offers: 6.23
```

4.1.1 Visualisation for outliers

[37]:
```
data_cpy = pd.merge(data_cpy, data_c[['id','offer_outlier']], on=['id'])
unique_products = data_cpy['product_type'].unique()

num_rows = math.ceil(len(unique_products) / 3)
num_cols = min(len(unique_products), 3)


fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)


for i, product in enumerate(unique_products):
    row_idx = i // 3
    col_idx = i % 3

    product_data = data_cpy[data_cpy['product_type'] == product]
    product_data_outlier =␣
 ↪product_data[(product_data['sales_outlier']==1)&(product_data['offer_outlier']!
 ↪=1)]
```

```
    product_date = product_data.
 ↪sort_values(by=['product_type','store_nbr','date'])
    product_data_outlier = product_data_outlier.sort_values(by='date')
    axes[row_idx, col_idx].plot(product_data['date'], product_data['sales'],␣
 ↪label=f'Sales - {product}', c = 'lightblue')
    axes[row_idx, col_idx].scatter(product_data_outlier['date'],␣
 ↪product_data_outlier['sales'], c = 'red')
    axes[row_idx, col_idx].set_xlabel('Date')
    axes[row_idx, col_idx].set_ylabel('Sales')
    axes[row_idx, col_idx].set_title(f'Date-wise sales for Product {product}')

# plt.tight_layout()
plt.show()

# outliers are selected such that there was a hike/dip in sales which was 15␣
 ↪times the avg sales of that product for that month
# out of these identified outliers, such points will be dropped where the sales␣
 ↪hike is contributed due to high offers
# for eg. office supplies has random high sales but minimal outliers due to␣
 ↪high offers
# on the other hand, home and kitchen has a lot of random behaviour which will␣
 ↪make it difficult to predict accurately
```


Date-wise sales for Product AUTOMOTIVE
Date-wise sales for Product BABY CARE
Date-wise sales for Product BEAUTY
Date-wise sales for Product BEVERAGES
Date-wise sales for Product BOOKS
Date-wise sales for Product BREAD/BAKERY
Date-wise sales for Product CELEBRATION
Date-wise sales for Product CLEANING
Date-wise sales for Product DAIRY
Date-wise sales for Product DELI
Date-wise sales for Product EGGS
Date-wise sales for Product FROZEN FOODS
Date-wise sales for Product GROCERY I
Date-wise sales for Product GROCERY II
Date-wise sales for Product HARDWARE
Date-wise sales for Product HOME AND KITCHEN
Date-wise sales for Product HOME AND KITCHEN II
Date-wise sales for Product HOME APPLIANCES
Date-wise sales for Product HOME CARE
Date-wise sales for Product LADIESWEAR
Date-wise sales for Product LAWN AND GARDEN
Date-wise sales for Product LINGERIE
Date-wise sales for Product LIQUOR,WINE,BEER
Date-wise sales for Product MAGAZINES
Date-wise sales for Product MEATS
Date-wise sales for Product PERSONAL CARE
Date-wise sales for Product PET SUPPLIES
Date-wise sales for Product PLAYERS AND ELECTRONICS
Date-wise sales for Product POULTRY
Date-wise sales for Product PREPARED FOODS
Date-wise sales for Product PRODUCE
Date-wise sales for Product SCHOOL AND OFFICE SUPPLIES
Date-wise sales for Product SEAFOOD
Sales
Date

[38]:
```
data.head()
```

[38]:

	id	date	store_nbr	product_type	sales	special_offer	year	month	day	dow	week_num	start_date
1619838	1619838	2015-07-01	1	AUTOMOTIVE	5.0	0	2015	7	1	2	27	2013-01-02
1619839	1619839	2015-07-01	1	BABY CARE	0.0	0	2015	7	1	2	27	NaT
1619840	1619840	2015-07-01	1	BEAUTY	5.0	1	2015	7	1	2	27	2013-01-02
1619841	1619841	2015-07-01	1	BEVERAGES	2638.0	2	2015	7	1	2	27	2013-01-02
1619842	1619842	2015-07-01	1	BOOKS	0.0	0	2015	7	1	2	27	2016-10-13

5 Modelling:

5.1 Preprocessing

[39]:
```
class ModelPreprocessing:

    from sklearn.cluster import KMeans
    from sklearn.preprocessing import StandardScaler

    def __init__(self, dataset, level):
        self.dataset = dataset
        self.level = level

    # Drop outlier wrt target variable
    def drop_outliers(self, target_variable, comparison_variable,␣
↪frequency_level = 'month', unique_identifier='id' ,outlier_factor=15,␣
↪comparison_factor=2):
        # select only after start date
        self.dataset[(self.dataset['date']>=self.dataset['start_date'])]

        data_sales = self.dataset.copy()

        df = data_sales[data_sales[target_variable]!=0]
        df = df.groupby([self.level,frequency_level])[target_variable].mean()

        df = df.reset_index().rename(columns = {target_variable:
↪'mean_target_var'})
```

```
        data_sales = pd.merge(data_sales, df, on = [self.level,frequency_level])
        data_sales['target_outlier'] =␣
↪(abs((data_sales[target_variable]-data_sales['mean_target_var'])/
↪data_sales['mean_target_var'])>outlier_factor).astype(int)


        data_offer = self.dataset.copy()
        df = data_offer.groupby([self.
↪level,frequency_level])[comparison_variable].mean()
        df = df.reset_index().rename(columns = {comparison_variable:
↪'mean_comparison_var'})

        data_offer = pd.merge(data_offer, df, on = [self.level,frequency_level])
        data_offer['comparison_outlier'] =␣
↪(abs((data_offer[comparison_variable]-data_offer['mean_comparison_var'])/
↪data_offer['mean_comparison_var'])>comparison_factor).astype(int)


        data_sales = pd.merge(data_sales,␣
↪data_offer[[unique_identifier,'comparison_outlier']], on=[unique_identifier])
        data_sales =␣
↪data_sales[(data_sales['target_outlier']==1)&(data_sales['comparison_outlier']!
↪=1)]

        self.dataset = pd.merge(self.dataset, data_sales[[unique_identifier]],␣
↪on = unique_identifier, how='left', indicator=True).
↪query('_merge=="left_only"').drop('_merge', axis=1)

        return self.dataset


    # Functions that create strong relative clusters as discussed in EDA
    def get_freq_group_cluster(self, new_col, frequency, grouping_col, n=3):

        data_type = self.dataset['type'].unique().tolist()

        freq_data = self.dataset.groupby([frequency,self.level])[grouping_col].
↪mean()
        freq_data = freq_data.reset_index()

        X = freq_data[grouping_col].values

        # Creating cluster using KMeans
        pipeline = Pipeline([
                        ('scaler', StandardScaler()),
```

```
                    ('cluster', KMeans(n_clusters=n, random_state=42,␣
↪n_init=10))
                ])

        X_train_scaled = pipeline.named_steps['scaler'].fit_transform(X.
↪reshape(-1, 1))
        y_pred = pipeline.named_steps['cluster'].fit_predict(X_train_scaled)

        # Ordering cluster values
        ordered_labels = np.argsort(pipeline.named_steps['cluster'].
↪cluster_centers_.sum(axis=1))
        label_mapping = {old_label: new_label for new_label, old_label in␣
↪enumerate(ordered_labels)}
        y_pred = pd.Series(y_pred).map(label_mapping)

        freq_data[new_col] = y_pred

        return freq_data[[frequency,self.level,new_col]]

    def set_freq_group_cluster(self, freq_data, new_col, frequency):
        self.dataset = self.dataset.drop(new_col,axis=1)
        self.dataset = pd.merge(self.dataset, freq_data, how ='left',␣
↪on=[frequency,self.level])

        return self.dataset


    def get_store_cluster(self, new_col, cluster_col, grouping_col):

        data_type = self.dataset['type'].unique().tolist()

        pivot_df = pd.pivot_table(self.dataset, values=grouping_col,␣
↪index=[cluster_col,self.level], fill_value=0)

        # Hierarchical Clustering
        distances = pdist(pivot_df)
        linkage_matrix = hierarchy.linkage(distances, method='ward')

        # Cutting the Dendrogram to Form Clusters
        height_threshold = 0.05 * np.max(linkage_matrix[:, 2])
        clusters = hierarchy.fcluster(linkage_matrix, height_threshold,␣
↪criterion='distance')
        pivot_df.loc[:, new_col] = clusters

        pivot_df = pivot_df.sort_values(by=grouping_col).reset_index()
        cluster_rank = pd.Series(pivot_df[new_col].unique())
```

```
        cluster_rank = {v:k for k,v in cluster_rank.to_dict().items()}
        pivot_df[new_col] = pivot_df[new_col].map(cluster_rank)+1

        return pivot_df[[cluster_col,self.level,new_col]]

    def set_store_cluster(self, pivot_df, new_col, cluster_col):
        self.dataset = self.dataset.drop(new_col,axis=1)
        self.dataset = pd.merge(self.dataset, pivot_df, how ='left',␣
↪on=[cluster_col,self.level])

        return self.dataset


    def is_special_offer_effective(self, drop_col, correlated_cols):
        correlation_value = self.dataset[correlated_cols].corr().iloc[0, 1]

        if correlation_value<0.2:
            self.dataset = self.dataset.drop(drop_col, axis=1)
            return False
        return True


    def is_product_active(self, grouping_col, date_col='date',␣
↪target_variable='sales', check_days=15):

        df = self.dataset.copy()
        try:
            recent_sales = df[(df[date_col] > df[date_col].max() - check_days)]
        except:
            recent_sales = df[(df[date_col] > df[date_col].max() - pd.
↪DateOffset(days=check_days))]
        recent_sales = recent_sales.drop(columns = [date_col])

        zero_sales = recent_sales.groupby(['store_nbr', self.
↪level])[target_variable].sum().reset_index()
        zero_sales = zero_sales[zero_sales[target_variable]==0][[self.
↪level,'store_nbr']]

        zero_sales_dataset = pd.merge(self.dataset, zero_sales, how='inner',␣
↪on=[self.level,'store_nbr'])
        self.dataset = pd.merge(self.dataset, zero_sales, how='left', on=[self.
↪level,'store_nbr'], indicator=True).query('_merge=="left_only"').
↪drop('_merge', axis=1)


        return zero_sales_dataset
```

```
    def set_lag_feature(self, source, group_col, target_variable, new_col,␣
↪method='yearly'):
        try:
            self.dataset = self.dataset.drop(new_col, axis=1)
        except:
            pass

        lag_data = source.groupby(group_col)[target_variable].mean().
↪reset_index(name=new_col)
        if method=='yearly':
            lag_data['year'] = lag_data['year']+1
            self.dataset = pd.merge(self.dataset, lag_data, how='left', on =␣
↪group_col)


        return self.dataset

    def set_range(self, month_year, date_col):

        self.dataset=self.dataset[self.dataset[date_col]>=pd.
↪to_datetime(month_year).toordinal()]

        return self.dataset
```

5.2 Cross Validation + Hyperparameter Tuning

5.2.1 Custom Grid Search

[40]:
```
# Creating this grid search to cross validate models for time series forecasting
class CustomGridSearch(ModelPreprocessing):

    def __init__(self, pipeline, params, dataset, preprocessing_params,␣
↪postprocessing_params, cv=3, prediction_days = 30, method='yearly'):
        self.pipeline = pipeline
        self.params = params
        self.dataset = dataset
        self.preprocessing_params = preprocessing_params
        self.cv = cv
        self.method = method
        self.prediction_days = prediction_days
        self.level = preprocessing_params['level']

    def preprocessing(self):


        # adding lag features
```

```
        super().set_lag_feature(self.
↪train_dataset,preprocessing_params['lag_feature'][0],preprocessing_params['lag_feature'][1]
↪preprocessing_params['lag_feature'][2], method='yearly')
        self.train_dataset, self.dataset = self.dataset.copy(), self.
↪validation_dataset.copy()
        super().set_lag_feature(self.
↪train_dataset,preprocessing_params['lag_feature'][0],preprocessing_params['lag_feature'][1]
↪preprocessing_params['lag_feature'][2], method='yearly')
        self.validation_dataset, self.dataset = self.dataset.copy(), self.
↪train_dataset.copy()


        # selecting range
        super().set_range(preprocessing_params['set_range'][0],␣
↪preprocessing_params['set_range'][1])
        self.train_dataset = self.dataset.copy()


        # remove outliers
        super().drop_outliers(preprocessing_params['drop_outliers'][0],␣
↪preprocessing_params['drop_outliers'][1],␣
↪preprocessing_params['drop_outliers'][2])

        # feature engineering
        # cluster columns creation
        for row in preprocessing_params['clusters']:
            cluster = super().get_freq_group_cluster(row[0], row[1], row[2])
            super().set_freq_group_cluster(cluster, row[0], row[1])
            self.train_dataset, self.dataset = self.dataset.copy(), self.
↪validation_dataset.copy()
            super().set_freq_group_cluster(cluster, row[0], row[1])
            self.validation_dataset, self.dataset = self.dataset.copy(), self.
↪train_dataset.copy()

        # store clustering
        store_cluster = super().
↪get_store_cluster(preprocessing_params['store_cluster'][0],preprocessing_params['store_clus
        super().set_store_cluster(store_cluster,␣
↪preprocessing_params['store_cluster'][0],preprocessing_params['store_cluster'][1])
        self.train_dataset, self.dataset = self.dataset.copy(), self.
↪validation_dataset.copy()
        super().set_store_cluster(store_cluster,␣
↪preprocessing_params['store_cluster'][0],preprocessing_params['store_cluster'][1])
        self.validation_dataset, self.dataset = self.dataset.copy(), self.
↪train_dataset.copy()
```

```
        # drop special offer cluster if not effective
        is_offer_effective = super().
↪is_special_offer_effective(preprocessing_params['special_offer'][0],preprocessing_params['s

        if is_offer_effective==False:
            self.validation_dataset.
↪drop(preprocessing_params['special_offer'][0], axis=1, inplace=True)


        self.train_dataset, self.dataset = self.dataset.copy(), self.
↪validation_dataset.copy()
        # remove inactive products for test set
        super().is_product_active(preprocessing_params['product_availability'])
        self.validation_dataset = self.dataset.copy()


    def train_validation_split(self):

        for fold in range(self.cv):

            split_date = self.dataset['date'].max() - (fold+1)*self.
↪prediction_days

            self.train_dataset = self.dataset[self.dataset['date']<= split_date]
            self.validation_dataset = self.dataset[(self.dataset['date']>␣
↪split_date) & (self.dataset['date']<= split_date+self.prediction_days)]


            # Data Preprocessing
            self.dummy_dataset = self.dataset.copy()
            self.dataset = self.train_dataset.copy()

            self.preprocessing()

            self.dataset = self.dummy_dataset.copy()

            # Select indices
            self.train_indices = self.train_dataset.index
            self.validation_indices = self.validation_dataset.index

            yield self.train_indices, self.validation_indices



    def hyperparameter_tuning(self, target_variable):
        from sklearn.model_selection import GridSearchCV
```

```
        splitter = self.train_validation_split()

        gridcv = GridSearchCV(self.pipeline, param_grid = self.params, cv =␣
↪splitter, scoring = "neg_root_mean_squared_error")

        best_model = gridcv.fit(self.dataset.
↪drop(postprocessing_params['drop_columns'],axis=1),self.
↪dataset[target_variable])

        return best_model
```

[41]:
```
data.columns
```

[41]:
```
Index(['id', 'date', 'store_nbr', 'product_type', 'sales', 'special_offer',
       'year', 'month', 'day', 'dow', 'week_num', 'start_date'],
      dtype='object')
```

5.2.2 Applying Cross-Validation

[42]:
```
df = data_backup.copy()

# Create dummy columns before preprocessing
df[['relative_monthly_sales_cluster','relative_daily_sales_cluster','relative_monthly_offer_cl
↪= 0

df = df[df['product_type'].isin(msp_list)]

unique_products = df['product_type'].unique()

preprocessing_params = {'lag_feature':␣
↪[['product_type','store_nbr','week_num','year'], 'sales', 'lag_yoy_sales'],
                        'set_range': ['2015-07', 'date'],
                        'drop_outliers': ['sales', 'special_offer', 'month'],
                        'clusters': [['relative_monthly_sales_cluster',␣
↪'month', 'sales'],
                                     ['relative_daily_sales_cluster', 'day',␣
↪'sales'],
                                     ['relative_monthly_offer_cluster',␣
↪'month', 'special_offer']],
                        'store_cluster':␣
↪['relative_store_cluster','store_nbr','sales'],
                        'special_offer':␣
↪['relative_monthly_offer_cluster',['special_offer', 'sales']],
                        'product_availability': 'store_nbr',
                        'level': 'product_type'}
```

```
postprocessing_params = {'drop_columns':␣
 ↪['id','sales','product_type','special_offer','year','type','start_date']}

grid_params_lr = {'LR__fit_intercept' : [True]}
grid_params_rf = {'RF__n_estimators' : [10,5], 'RF__max_depth':[5,10]}
grid_params_xgb = {'XGB__n_estimators' : [10,100], 'XGB__max_depth':[5,10]}
grid_params_lgb = {'LGB__n_estimators' : [10,100], 'LGB__max_depth':[3,5]}

result_data = []

for product_type in unique_products:

    product_data = df.loc[(df['product_type'] == product_type)]
    product_data = product_data.reset_index(drop=True)
    product_data['date'] = product_data['date'].apply(lambda x: x.toordinal())
    product_data['start_date'] = product_data['start_date'].apply(lambda x: x.
 ↪toordinal())


    # Cross Validation + Hyperparameter Tuning for RandomForest

    pipeline_rf = Pipeline([
        ('RF',RandomForestRegressor())
    ])
    grid_test_rf = CustomGridSearch(pipeline_rf, grid_params_rf, product_data,␣
 ↪preprocessing_params, postprocessing_params, cv=3, prediction_days=16)
    score_rf = grid_test_rf.hyperparameter_tuning('sales')

    result_data.append({'product': product_type, 'model': 'RandomForest',␣
 ↪'best_score': score_rf.best_score_, 'best_params': score_rf.best_params_})

    # Cross Validation + Hyperparameter Tuning for XGBoost
    pipeline_xgb = Pipeline([
        ('XGB',XGBRegressor())
    ])
    grid_test_xgb = CustomGridSearch(pipeline_xgb, grid_params_xgb,␣
 ↪product_data, preprocessing_params, postprocessing_params, cv=3,␣
 ↪prediction_days=16)
    score_xgb = grid_test_xgb.hyperparameter_tuning('sales')

    result_data.append({'product': product_type, 'model': 'XGBoost',␣
 ↪'best_score': score_xgb.best_score_, 'best_params': score_xgb.best_params_})

    # Cross Validation + Hyperparameter Tuning for LightGBM
    pipeline_lgb = Pipeline([
```

```
        ('LGB',lgb.LGBMRegressor(random_state=0, verbose=-1))
    ])
    grid_test_lgb = CustomGridSearch(pipeline_lgb, grid_params_lgb,␣
↪product_data, preprocessing_params, postprocessing_params, cv=3,␣
↪prediction_days=16)
    score_lgb = grid_test_lgb.hyperparameter_tuning('sales')

    result_data.append({'product': product_type, 'model': 'LightGBM',␣
↪'best_score': score_lgb.best_score_, 'best_params': score_lgb.best_params_})

    print ('CV Done for Product:', product_type)

result_data = pd.DataFrame(result_data)
result_data
```

```
CV Done for Product: BEVERAGES
CV Done for Product: CLEANING
CV Done for Product: DAIRY
CV Done for Product: GROCERY I
CV Done for Product: PRODUCE
```

[42]:

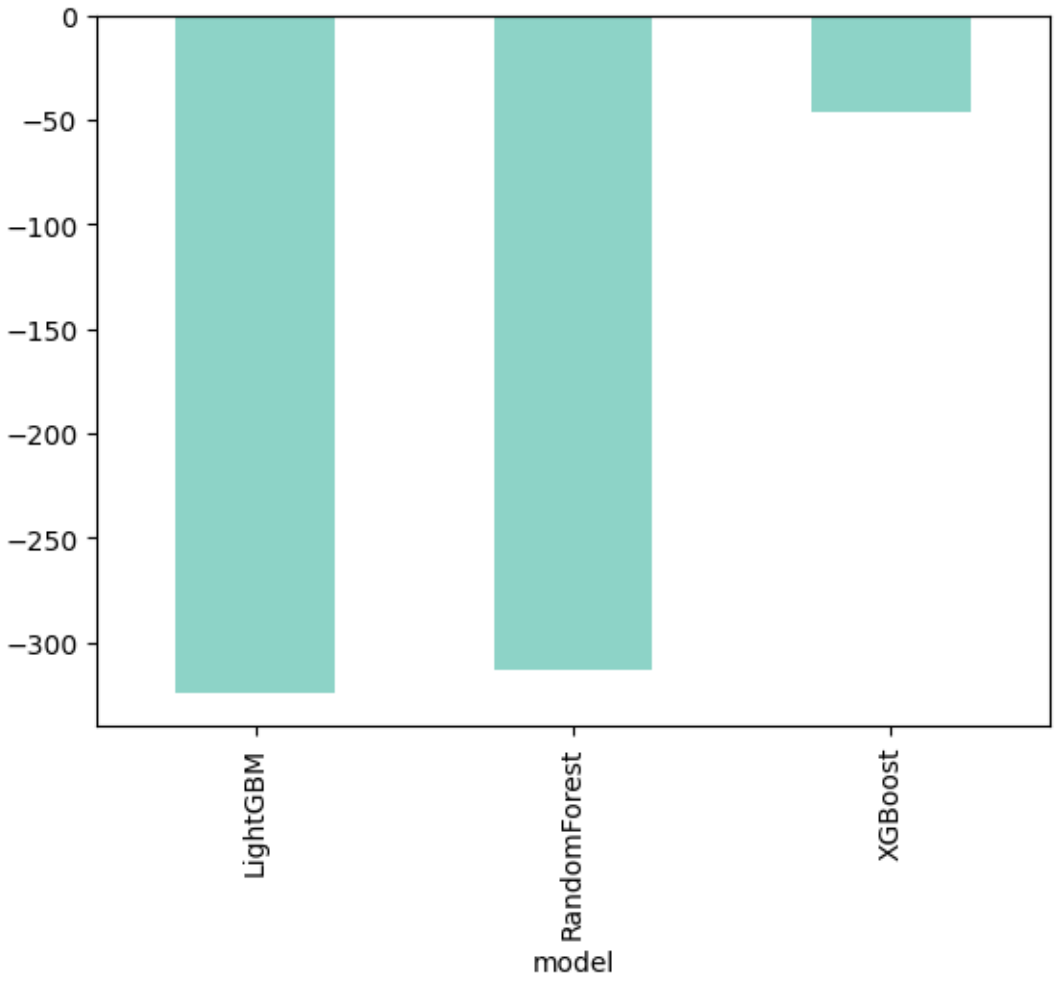
	product	model	best_score	best_params
0	BEVERAGES	RandomForest	-379.945847	{'RF__max_depth': 10, 'RF__n_estimators': 10}
1	BEVERAGES	XGBoost	-60.765024	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
2	BEVERAGES	LightGBM	-340.247965	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}
3	CLEANING	RandomForest	-262.544810	{'RF__max_depth': 10, 'RF__n_estimators': 10}
4	CLEANING	XGBoost	-38.880899	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
5	CLEANING	LightGBM	-267.913110	{'LGB__max_depth': 5, 'LGB__n_estimators': 100}
6	DAIRY	RandomForest	-104.502614	{'RF__max_depth': 10, 'RF__n_estimators': 10}
7	DAIRY	XGBoost	-14.985900	{'XGB__max_depth': 10, 'XGB__n_estimators': 100}
8	DAIRY	LightGBM	-105.765573	
9	GROCERY I	RandomForest	-820.853197	
10	GROCERY I	XGBoost	-109.588523	
11	GROCERY I	LightGBM	-795.383109	
12	PRODUCE	RandomForest	-0.438669	
13	PRODUCE	XGBoost	-6.321448	
14	PRODUCE	LightGBM	-110.948429	

```
8    {'LGB__max_depth': 5, 'LGB__n_estimators': 100}
9      {'RF__max_depth': 10, 'RF__n_estimators': 10}
10  {'XGB__max_depth': 10, 'XGB__n_estimators': 100}
11   {'LGB__max_depth': 5, 'LGB__n_estimators': 100}
12     {'RF__max_depth': 10, 'RF__n_estimators': 10}
13  {'XGB__max_depth': 10, 'XGB__n_estimators': 100}
14   {'LGB__max_depth': 5, 'LGB__n_estimators': 100}
```

[46]:
```
result_data['rank'] = result_data.groupby('product')['best_score'].
 ↪rank(ascending=False, method='first')
best_data = result_data[result_data['rank']==1]
best_data['best_params'] = best_data['best_params'].astype(str)
print ('Best Params:\n',best_data.groupby(['model','best_params'])['rank'].
 ↪count().sort_values(ascending=False).head())

result_data.groupby('model')['best_score'].mean().plot.bar()
```

```
Best Params:
 model         best_params
XGBoost       {'XGB__max_depth': 10, 'XGB__n_estimators': 100}    4
RandomForest  {'RF__max_depth': 10, 'RF__n_estimators': 10}       1
Name: rank, dtype: int64
```

[46]:
```
<Axes: xlabel='model'>
```



5.3 Prediction:

[47]:
```
deep_result = pd.DataFrame()
shallow_result = pd.DataFrame()
df = data_backup.copy()

# Create dummy columns before preprocessing
df[['relative_monthly_sales_cluster','relative_daily_sales_cluster','relative_monthly_offer_cl
  ↪= 0

# df = df[df['product_type'].isin(msp_list)]

unique_products = df['product_type'].unique()

for product_type in unique_products:
```

```
    product_data = df.loc[(df['product_type'] == product_type)]
    product_data = product_data.reset_index(drop=True)

    # train test split
    product_train = product_data[product_data['date'].dt.
↪to_period('M')<='2017-07']
    product_train['type'] = 'Training Set'
    product_train =␣
↪product_train[product_train['date']>=product_train['start_date']]
    product_train['date'] = product_train['date'].apply(lambda x: x.toordinal())
    product_train['start_date'] = product_train['start_date'].apply(lambda x: x.
↪toordinal())

    product_test = product_data[product_data['date'].dt.
↪to_period('M')>'2017-07']
    product_test['type'] = 'Test Set'
    product_test['date'] = product_test['date'].apply(lambda x: x.toordinal())

    # Preprocessing
    preprocessing_start = time.time()

    preprocessor_train = ModelPreprocessing(product_train, level =␣
↪'product_type')
    preprocessor_test = ModelPreprocessing(product_test, level = 'product_type')

    # adding lag features
    preprocessor_train.set_lag_feature(preprocessor_train.
↪dataset,['product_type','store_nbr','week_num','year'], 'sales',␣
↪'lag_yoy_sales', method='yearly')
    preprocessor_test.set_lag_feature(preprocessor_train.
↪dataset,['product_type','store_nbr','week_num','year'], 'sales',␣
↪'lag_yoy_sales', method='yearly')

    # selecting range
    preprocessor_train.set_range('2015-07', 'date')

    # drop outliers
    preprocessor_train.drop_outliers('sales', 'special_offer', 'month')

    # cluster columns creation
    monthly_sales_cluster = preprocessor_train.
↪get_freq_group_cluster('relative_monthly_sales_cluster', 'month', 'sales')
    preprocessor_train.set_freq_group_cluster(monthly_sales_cluster,␣
↪'relative_monthly_sales_cluster', 'month')
```

```
  daily_sales_cluster = preprocessor_train.
↪get_freq_group_cluster('relative_daily_sales_cluster', 'day', 'sales',10)
  preprocessor_train.set_freq_group_cluster(daily_sales_cluster,␣
↪'relative_daily_sales_cluster', 'day')

  monthly_offer_cluster = preprocessor_train.
↪get_freq_group_cluster('relative_monthly_offer_cluster', 'month',␣
↪'special_offer')
  preprocessor_train.set_freq_group_cluster(monthly_offer_cluster,␣
↪'relative_monthly_offer_cluster', 'month')

  store_cluster = preprocessor_train.
↪get_store_cluster('relative_store_cluster','store_nbr','sales')
  preprocessor_train.
↪set_store_cluster(store_cluster,'relative_store_cluster','store_nbr')

  # drop special offer cluster if not effective
  is_offer_effective = preprocessor_train.
↪is_special_offer_effective('relative_monthly_offer_cluster',['special_offer',␣
↪'sales'])



  # apply clusters based on train data

  preprocessor_test.set_freq_group_cluster(monthly_sales_cluster,␣
↪'relative_monthly_sales_cluster', 'month')
  preprocessor_test.set_freq_group_cluster(daily_sales_cluster,␣
↪'relative_daily_sales_cluster', 'day')
  preprocessor_test.set_freq_group_cluster(monthly_offer_cluster,␣
↪'relative_monthly_offer_cluster', 'month')
  preprocessor_test.
↪set_store_cluster(store_cluster,'relative_store_cluster','store_nbr')


  # remove inactive products for test set
  zero_prediction_dataset = preprocessor_test.is_product_active('store_nbr')
  zero_prediction_dataset['predicted_sales'] = 0


  if is_offer_effective==False:
          preprocessor_test.dataset.drop('relative_monthly_offer_cluster',␣
↪axis=1, inplace=True)
          zero_prediction_dataset.drop('relative_monthly_offer_cluster',␣
↪axis=1, inplace=True)
```

```
    non_selected_features =␣
↪['id','product_type','special_offer','week_num','type','sales','start_date','date','day','s

    # applying model
    model_training_start = time.time()


    model_xgb = XGBRegressor(random_state = 42, learning_rate = 0.1, max_depth␣
↪= 10, n_estimators=100)



    model_xgb.fit(preprocessor_train.dataset.
↪drop(non_selected_features,axis=1), preprocessor_train.dataset['sales'])



    model_prediction_start = time.time()
    y_pred_xgb = model_xgb.predict(preprocessor_test.dataset.
↪drop(non_selected_features,axis=1))

    # Print time performance
    print("Data Preprocessing for", product_type, "took",␣
↪round((model_training_start-preprocessing_start),1), "Seconds!")
    print("Model Training for", product_type, "took",␣
↪round((model_prediction_start-model_training_start),1), "Seconds!")


    rmse_xgb = np.sqrt(mean_squared_error(preprocessor_test.dataset['sales'],␣
↪y_pred_xgb))
    mae_xgb = mean_absolute_error(preprocessor_test.dataset['sales'],␣
↪y_pred_xgb)
    r2_xgb = r2_score(preprocessor_test.dataset['sales'], y_pred_xgb)


    # Storing individual results
    result_data = preprocessor_test.dataset.copy()
    result_data['predicted_sales'] = y_pred_xgb
    result_data['error'] = result_data['predicted_sales'] - result_data['sales']
    zero_prediction_dataset['error'] =␣
↪zero_prediction_dataset['predicted_sales'] - zero_prediction_dataset['sales']
```

```
    deep_result = pd.concat([deep_result, result_data,␣
 ↪zero_prediction_dataset[result_data.columns]])

    # Storing Overall results
    shallow_result_data = pd.DataFrame([{'product_type': product_type,␣
 ↪'avg_sales': result_data['sales'].mean(), 'avg_predicted_sales':␣
 ↪result_data['predicted_sales'].mean(), 'mae': mae_xgb, 'rmse': rmse_xgb,␣
 ↪'r2': r2_xgb}])
    shallow_result = pd.concat([shallow_result, shallow_result_data])

print ('Features Taken:', preprocessor_train.dataset.
 ↪drop(non_selected_features,axis=1).columns)

for column in ['avg_sales','avg_predicted_sales','mae','rmse','r2']:
    shallow_result[column] = np.round(shallow_result[column].astype(float),2)
deep_result['date'] = deep_result['date'].apply(lambda x: date.fromordinal(x))
```

```
Data Preprocessing for AUTOMOTIVE took 1.3 Seconds!
Model Training for AUTOMOTIVE took 1.4 Seconds!
Data Preprocessing for BABY CARE took 0.7 Seconds!
Model Training for BABY CARE took 1.5 Seconds!
Data Preprocessing for BEAUTY took 1.3 Seconds!
Model Training for BEAUTY took 1.5 Seconds!
Data Preprocessing for BEVERAGES took 0.7 Seconds!
Model Training for BEVERAGES took 1.5 Seconds!
Data Preprocessing for BOOKS took 0.4 Seconds!
Model Training for BOOKS took 0.9 Seconds!
Data Preprocessing for BREAD/BAKERY took 0.9 Seconds!
Model Training for BREAD/BAKERY took 1.9 Seconds!
Data Preprocessing for CELEBRATION took 0.7 Seconds!
Model Training for CELEBRATION took 1.8 Seconds!
Data Preprocessing for CLEANING took 0.9 Seconds!
Model Training for CLEANING took 3.8 Seconds!
Data Preprocessing for DAIRY took 0.7 Seconds!
Model Training for DAIRY took 1.3 Seconds!
Data Preprocessing for DELI took 0.7 Seconds!
Model Training for DELI took 1.3 Seconds!
Data Preprocessing for EGGS took 0.9 Seconds!
Model Training for EGGS took 1.2 Seconds!
Data Preprocessing for FROZEN FOODS took 0.8 Seconds!
Model Training for FROZEN FOODS took 1.3 Seconds!
Data Preprocessing for GROCERY I took 1.0 Seconds!
Model Training for GROCERY I took 1.7 Seconds!
Data Preprocessing for GROCERY II took 1.1 Seconds!
Model Training for GROCERY II took 1.6 Seconds!
Data Preprocessing for HARDWARE took 0.8 Seconds!
Model Training for HARDWARE took 1.4 Seconds!
```

```
Data Preprocessing for HOME AND KITCHEN I took 0.7 Seconds!
Model Training for HOME AND KITCHEN I took 1.3 Seconds!
Data Preprocessing for HOME AND KITCHEN II took 0.7 Seconds!
Model Training for HOME AND KITCHEN II took 1.4 Seconds!
Data Preprocessing for HOME APPLIANCES took 0.7 Seconds!
Model Training for HOME APPLIANCES took 1.5 Seconds!
Data Preprocessing for HOME CARE took 0.6 Seconds!
Model Training for HOME CARE took 1.4 Seconds!
Data Preprocessing for LADIESWEAR took 0.7 Seconds!
Model Training for LADIESWEAR took 1.3 Seconds!
Data Preprocessing for LAWN AND GARDEN took 0.7 Seconds!
Model Training for LAWN AND GARDEN took 1.2 Seconds!
Data Preprocessing for LINGERIE took 0.8 Seconds!
Model Training for LINGERIE took 1.4 Seconds!
Data Preprocessing for LIQUOR,WINE,BEER took 0.8 Seconds!
Model Training for LIQUOR,WINE,BEER took 1.4 Seconds!
Data Preprocessing for MAGAZINES took 0.8 Seconds!
Model Training for MAGAZINES took 1.7 Seconds!
Data Preprocessing for MEATS took 0.8 Seconds!
Model Training for MEATS took 1.3 Seconds!
Data Preprocessing for PERSONAL CARE took 0.8 Seconds!
Model Training for PERSONAL CARE took 1.4 Seconds!
Data Preprocessing for PET SUPPLIES took 0.9 Seconds!
Model Training for PET SUPPLIES took 1.6 Seconds!
Data Preprocessing for PLAYERS AND ELECTRONICS took 0.7 Seconds!
Model Training for PLAYERS AND ELECTRONICS took 1.3 Seconds!
Data Preprocessing for POULTRY took 0.7 Seconds!
Model Training for POULTRY took 1.7 Seconds!
Data Preprocessing for PREPARED FOODS took 0.8 Seconds!
Model Training for PREPARED FOODS took 1.3 Seconds!
Data Preprocessing for PRODUCE took 0.7 Seconds!
Model Training for PRODUCE took 1.4 Seconds!
Data Preprocessing for SCHOOL AND OFFICE SUPPLIES took 0.8 Seconds!
Model Training for SCHOOL AND OFFICE SUPPLIES took 0.9 Seconds!
Data Preprocessing for SEAFOOD took 0.7 Seconds!
Model Training for SEAFOOD took 1.2 Seconds!
Features Taken: Index(['year', 'month', 'dow', 'lag_yoy_sales',
       'relative_monthly_sales_cluster', 'relative_daily_sales_cluster',
       'relative_monthly_offer_cluster', 'relative_store_cluster'],
      dtype='object')
```

6 Performance Evaluation

[56]:
```
print (preprocessor_train.dataset.drop(non_selected_features,axis=1).columns)
```

```
Index(['year', 'month', 'dow', 'lag_yoy_sales',
       'relative_monthly_sales_cluster', 'relative_daily_sales_cluster',
```

```
       'relative_monthly_offer_cluster', 'relative_store_cluster'],
      dtype='object')
```

6.1 Overall Result

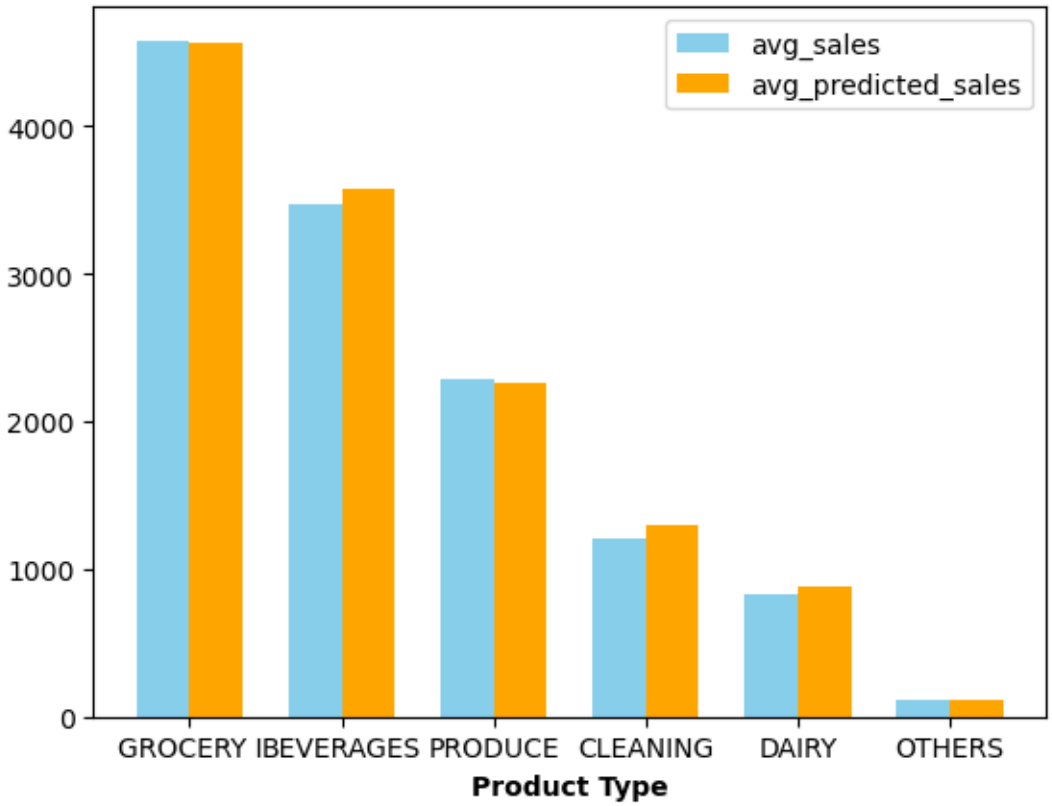
[48]:
```
other_shallow_result = (shallow_result[~shallow_result['product_type'].
 ↪isin(msp_list)]).copy()

other_results = pd.DataFrame([{'product_type': 'OTHERS'
                               , 'avg_sales': np.
 ↪round(other_shallow_result['avg_sales'].mean(),2)
                               , 'avg_predicted_sales': np.
 ↪round(other_shallow_result['avg_predicted_sales'].mean(),2)
                               , 'mae': np.round(other_shallow_result['mae'].
 ↪mean(),2)
                               , 'rmse': np.round(other_shallow_result['rmse'].
 ↪mean(),2)}])

tuned_shallow_result = pd.concat([shallow_result[shallow_result['product_type'].
 ↪isin(msp_list)].drop('r2',axis=1), other_results])
tuned_shallow_result
```

[48]:

	product_type	avg_sales	avg_predicted_sales	mae	rmse
0	BEVERAGES	3469.02	3576.01	751.68	1084.93
0	CLEANING	1204.76	1301.95	314.43	527.65
0	DAIRY	832.77	884.40	118.71	182.63
0	GROCERY I	4580.49	4569.45	677.03	1027.26
0	PRODUCE	2290.86	2254.22	313.27	476.31
0	OTHERS	107.01	107.38	24.57	44.98

[49]:
```
bar_width = 0.35

r1 = np.arange(len(tuned_shallow_result))
r2 = r1+0.35

tuned_shallow_result= tuned_shallow_result.sort_values(by='avg_sales',␣
 ↪ascending=False)

plt.bar(r1, tuned_shallow_result['avg_sales'], color='skyblue',␣
 ↪width=bar_width, label='avg_sales')
plt.bar(r2, tuned_shallow_result['avg_predicted_sales'], color='orange',␣
 ↪width=bar_width, label='avg_predicted_sales')

plt.xlabel('Product Type', fontweight='bold')
plt.xticks((r1+r2)/2, tuned_shallow_result['product_type'])

plt.legend()
```

```
plt.show()
```



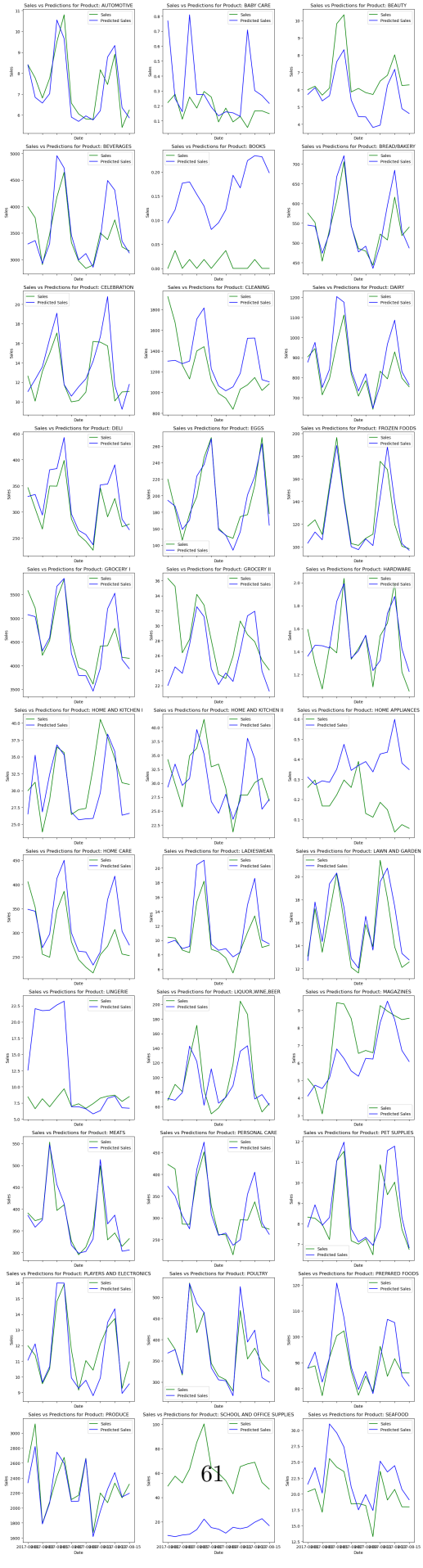
6.2 Daily Prediction Evaluation

6.2.1 Sales vs Prediction Trend

[50]:
```
def plot_result(result, level, graph, data_dict, iterator, graphs_per_row=3):
    num_rows = math.ceil(len(iterator)/graphs_per_row)
    num_cols = min(len(iterator),graphs_per_row)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows),␣
 ↪sharex=True)
    sns.set_palette("Set1")
    for i, row in enumerate(iterator):
        row_idx = i//graphs_per_row
        col_idx = i%graphs_per_row

        tmp_result = result[result[level] == row]

        for j in data_dict['data']:
            if graph=='line':
```

```
                axes[row_idx,col_idx].plot(tmp_result[j['x']],␣
↪tmp_result[j['y']], label=j['label'], color = j['color'])
            axes[row_idx,col_idx].set_xlabel(data_dict['x_label'])
            axes[row_idx,col_idx].set_ylabel(data_dict['y_label'])
            axes[row_idx,col_idx].set_title(data_dict['title']+row)
            axes[row_idx,col_idx].legend()

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

[51]:
```
product_result = deep_result.
↪groupby(['product_type','date'])[['sales','predicted_sales','error']].mean().
↪reset_index()

plot_result(product_result, level='product_type', graph='line',
            data_dict = {'data': [{'x': 'date', 'y': 'sales', 'label': 'Sales',␣
↪'color': 'green'},
                          {'x': 'date', 'y': 'predicted_sales', 'label':␣
↪'Predicted Sales' , 'color': 'blue'}],
                          'x_label': 'Date',
                          'y_label': 'Sales',
                          'title': 'Sales vs Predictions for Product: '},
            iterator = product_result['product_type'].unique())
```


Sales vs Predictions for Product: AUTOMOTIVE
Sales vs Predictions for Product: BABY CARE
Sales vs Predictions for Product: BEAUTY
Sales vs Predictions for Product: BEVERAGES
Sales vs Predictions for Product: BOOKS
Sales vs Predictions for Product: BREAD/BAKERY
Sales vs Predictions for Product: CELEBRATION
Sales vs Predictions for Product: CLEANING
Sales vs Predictions for Product: DAIRY
Sales vs Predictions for Product: DELI
Sales vs Predictions for Product: EGGS
Sales vs Predictions for Product: FROZEN FOODS
Sales vs Predictions for Product: GROCERY I
Sales vs Predictions for Product: GROCERY II
Sales vs Predictions for Product: HARDWARE
Sales vs Predictions for Product: HOME AND KITCHEN I
Sales vs Predictions for Product: HOME AND KITCHEN II
Sales vs Predictions for Product: HOME APPLIANCES
Sales vs Predictions for Product: HOME CARE
Sales vs Predictions for Product: LADIESWEAR
Sales vs Predictions for Product: LAWN AND GARDEN
Sales vs Predictions for Product: LINGERIE
Sales vs Predictions for Product: LIQUOR,WINE,BEER
Sales vs Predictions for Product: MAGAZINES
Sales vs Predictions for Product: MEATS
Sales vs Predictions for Product: PERSONAL CARE
Sales vs Predictions for Product: PET SUPPLIES
Sales vs Predictions for Product: PLAYERS AND ELECTRONICS
Sales vs Predictions for Product: POULTRY
Sales vs Predictions for Product: PREPARED FOODS
Sales vs Predictions for Product: PRODUCE
Sales vs Predictions for Product: SCHOOL AND OFFICE SUPPLIES
Sales vs Predictions for Product: SEAFOOD
Sales
Predicted Sales
Date
Sales

6.2.2 Percentage Deviation

[52]:
```
deep_result['deviation'] = (deep_result['error']/np.
 ↪where(deep_result['sales']==0,1,deep_result['sales']*0.01))
deep_result['deviation'].fillna(0,inplace=True)

date_result = np.round(pd.pivot_table(deep_result, index = 'product_type',␣
 ↪columns = 'date', values='deviation'),2)
date_result
```

[52]:

date	2017-08-01	2017-08-02	2017-08-03	2017-08-04	\
product_type					
AUTOMOTIVE	30.81	33.90	34.36	40.64	
BABY CARE	-6.80	-7.92	-3.86	2.39	
BEAUTY	25.45	37.75	13.58	21.72	
BEVERAGES	-15.20	-9.44	3.55	-0.61	
BOOKS	0.09	3.38	0.18	3.91	
BREAD/BAKERY	-6.12	-2.69	5.39	0.31	
CELEBRATION	33.84	50.34	55.34	39.07	
CLEANING	-26.35	-17.39	29.64	24.83	
DAIRY	0.25	5.99	6.22	8.35	
DELI	-2.67	10.28	18.36	15.64	
EGGS	6.59	13.33	22.95	5.11	
FROZEN FOODS	-7.59	-0.39	3.99	6.87	
GROCERY I	-6.90	-2.18	3.59	2.55	
GROCERY II	-20.57	-7.20	21.63	12.56	
HARDWARE	-8.97	0.08	8.93	-5.10	
HOME AND KITCHEN I	16.24	36.66	37.00	39.13	
HOME AND KITCHEN II	-0.45	26.73	33.87	53.41	
HOME APPLIANCES	-10.04	-10.74	-4.47	-6.22	
HOME CARE	-9.84	1.58	7.69	22.62	
LADIESWEAR	16.54	34.11	23.32	49.62	
LAWN AND GARDEN	15.23	16.71	32.29	51.51	
LINGERIE	123.59	101.20	281.66	232.71	
LIQUOR,WINE,BEER	113.12	87.38	84.96	28.51	
MAGAZINES	-10.50	8.07	32.77	19.51	
MEATS	-3.11	-1.04	12.17	9.13	
PERSONAL CARE	-7.66	-5.73	11.46	2.62	
PET SUPPLIES	20.90	28.70	21.53	27.53	
PLAYERS AND ELECTRONICS	15.33	49.82	43.98	40.13	
POULTRY	-10.25	0.14	3.25	8.27	
PREPARED FOODS	7.73	19.04	38.44	11.73	
PRODUCE	-5.09	-3.67	-0.40	-1.78	
SCHOOL AND OFFICE SUPPLIES	74.93	37.23	264.11	176.59	

SEAFOOD	56.01	28.16	66.27	51.63	

date	2017-08-05	2017-08-06	2017-08-07	2017-08-08	\
product_type					
AUTOMOTIVE	35.31	4.98	12.38	18.14	
BABY CARE	-5.15	-9.69	-11.40	-7.43	
BEAUTY	20.32	14.71	12.63	9.21	
BEVERAGES	23.37	12.86	6.86	6.05	
BOOKS	0.15	1.93	0.08	2.01	
BREAD/BAKERY	8.51	8.87	0.94	-1.71	
CELEBRATION	48.16	50.71	52.52	42.11	
CLEANING	34.82	35.28	11.08	11.52	
DAIRY	26.22	12.61	5.22	5.82	
DELI	12.82	14.59	8.82	9.48	
EGGS	22.51	5.60	14.05	6.29	
FROZEN FOODS	0.51	5.25	12.11	1.85	
GROCERY I	4.10	2.20	-6.31	-3.94	
GROCERY II	26.74	9.19	0.40	25.62	
HARDWARE	29.09	-0.54	-3.21	-2.08	
HOME AND KITCHEN I	20.22	41.26	45.56	26.56	
HOME AND KITCHEN II	38.57	19.63	24.06	22.26	
HOME APPLIANCES	-9.52	-7.00	-10.78	-7.88	
HOME CARE	23.36	22.73	9.69	12.80	
LADIESWEAR	56.06	40.97	32.11	9.60	
LAWN AND GARDEN	20.03	32.76	40.42	51.16	
LINGERIE	289.68	173.58	56.13	40.60	
LIQUOR,WINE,BEER	-23.49	2.63	142.73	81.84	
MAGAZINES	3.12	-5.18	-6.03	-8.57	
MEATS	17.73	5.65	-2.20	5.99	
PERSONAL CARE	10.07	8.20	-3.22	0.81	
PET SUPPLIES	29.81	22.68	29.29	12.86	
PLAYERS AND ELECTRONICS	28.31	33.70	12.01	28.92	
POULTRY	27.48	4.40	-2.00	-4.70	
PREPARED FOODS	28.75	19.00	9.40	10.03	
PRODUCE	17.29	1.26	-1.63	1.72	
SCHOOL AND OFFICE SUPPLIES	114.86	353.59	35.41	-6.19	
SEAFOOD	73.09	71.14	49.08	57.42	

date	2017-08-09	2017-08-10	2017-08-11	2017-08-12	\
product_type					
AUTOMOTIVE	32.58	35.29	7.64	67.03	
BABY CARE	-4.67	-4.13	-9.04	29.16	
BEAUTY	-1.70	-3.17	-5.21	12.17	
BEVERAGES	13.50	4.21	4.93	35.34	
BOOKS	-0.48	0.19	0.17	0.22	
BREAD/BAKERY	0.92	0.35	0.50	19.01	
CELEBRATION	56.58	12.07	36.58	82.25	

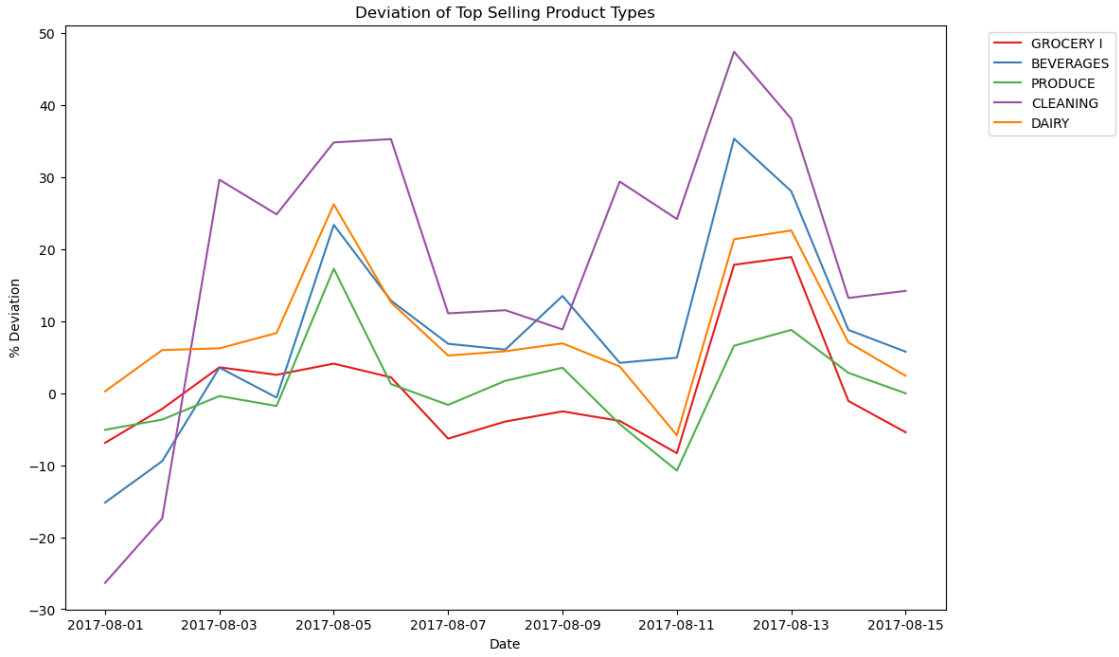
CLEANING	8.85	29.38	24.18	47.41
DAIRY	6.92	3.71	-5.85	21.37
DELI	8.82	7.40	8.41	26.41
EGGS	6.38	2.70	-5.73	22.83
FROZEN FOODS	5.97	6.32	-10.34	26.29
GROCERY I	-2.53	-3.85	-8.33	17.83
GROCERY II	34.03	-2.24	9.33	26.21
HARDWARE	12.14	0.67	-0.18	-0.72
HOME AND KITCHEN I	41.46	19.82	1.65	38.87
HOME AND KITCHEN II	32.85	39.98	14.67	55.31
HOME APPLIANCES	-1.94	0.95	-1.90	-1.91
HOME CARE	20.86	11.19	9.09	38.33
LADIESWEAR	21.36	75.09	43.07	42.62
LAWN AND GARDEN	19.90	8.27	37.81	28.99
LINGERIE	71.76	23.02	17.88	71.84
LIQUOR,WINE,BEER	22.07	14.58	-7.18	-5.43
MAGAZINES	18.47	-6.61	16.97	33.18
MEATS	5.18	1.54	13.16	15.71
PERSONAL CARE	2.62	14.79	-9.36	20.30
PET SUPPLIES	18.83	30.07	-8.21	48.84
PLAYERS AND ELECTRONICS	19.96	6.09	15.52	34.35
POULTRY	4.71	-4.13	-0.35	11.10
PREPARED FOODS	13.87	6.45	6.12	35.66
PRODUCE	3.53	-4.30	-10.76	6.59
SCHOOL AND OFFICE SUPPLIES	115.63	234.82	6.72	235.76
SEAFOOD	35.37	61.97	37.89	56.83

date	2017-08-13	2017-08-14	2017-08-15
product_type			
AUTOMOTIVE	59.42	52.82	28.11
BABY CARE	-6.77	-4.49	-2.60
BEAUTY	38.50	14.31	-5.57
BEVERAGES	28.04	8.78	5.76
BOOKS	1.59	0.23	0.20
BREAD/BAKERY	15.56	2.13	-5.67
CELEBRATION	62.51	52.91	53.75
CLEANING	38.10	13.22	14.21
DAIRY	22.60	7.04	2.41
DELI	23.51	7.31	-1.21
EGGS	19.86	18.90	5.33
FROZEN FOODS	28.32	6.57	2.45
GROCERY I	18.90	-1.09	-5.43
GROCERY II	82.35	49.18	23.29
HARDWARE	13.00	-1.43	2.96
HOME AND KITCHEN I	31.76	26.37	18.13
HOME AND KITCHEN II	52.23	9.27	17.17
HOME APPLIANCES	4.05	1.58	0.84

HOME CARE	38.53	19.65	11.16
LADIESWEAR	65.02	30.17	42.06
LAWN AND GARDEN	35.32	44.31	14.08
LINGERIE	93.57	67.25	26.19
LIQUOR,WINE,BEER	25.34	162.40	56.41
MAGAZINES	34.99	-16.43	-17.39
MEATS	15.27	-0.85	-6.67
PERSONAL CARE	24.56	8.69	-2.84
PET SUPPLIES	36.49	28.36	7.20
PLAYERS AND ELECTRONICS	31.89	24.74	8.10
POULTRY	12.41	-5.02	-2.41
PREPARED FOODS	28.02	10.96	-1.03
PRODUCE	8.79	2.82	-0.02
SCHOOL AND OFFICE SUPPLIES	90.94	173.74	75.66
SEAFOOD	53.36	36.03	42.90

6.2.3 Percentage Deviation for Most Selling Products (MSP)

[53]:
```
data_result_transposed = date_result.transpose()

plt.figure(figsize=(12, 8))

for product_type in msp_list:
    plt.plot(data_result_transposed.index,␣
 ↪data_result_transposed[product_type], label=product_type)

plt.xlabel('Date')
plt.ylabel('% Deviation')
plt.title('Deviation of Top Selling Product Types')

plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1))

plt.show()
```

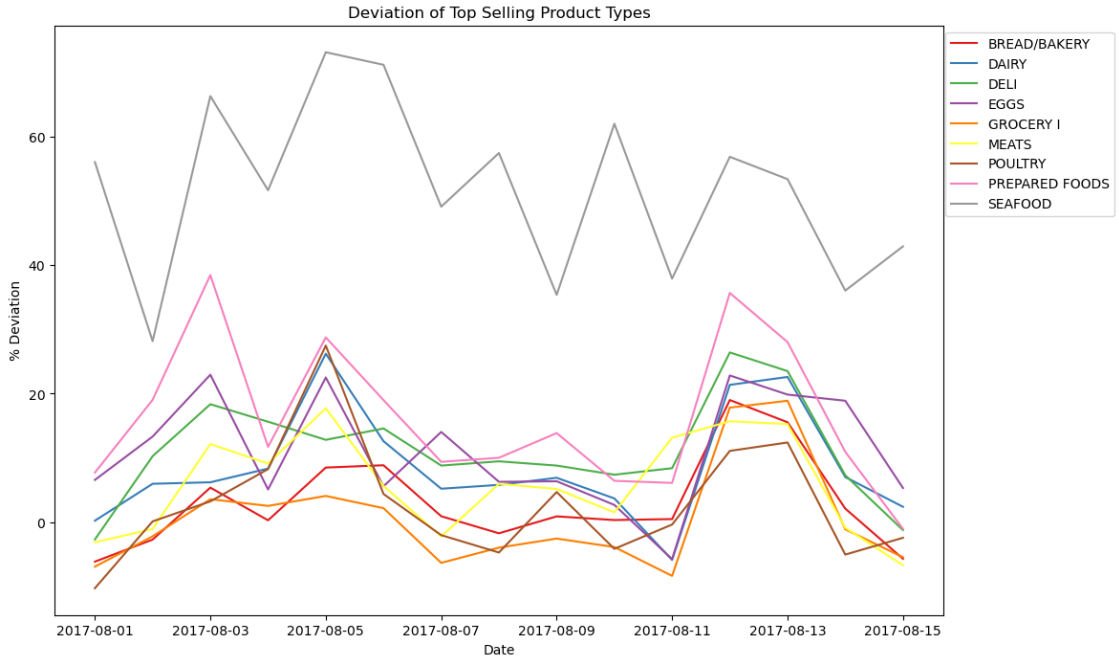


6.2.4 Percentage Deviation for Critical Products (Low Shelf Life)

[ ]:
```
# Low shelf life products have been assumed according to domain knowledge
```

[54]:
```
# In days
shelf_life_mapping = {'BREAD/BAKERY' : 5
                          ,'DAIRY' : 3
                          ,'DELI': 3
                          ,'EGGS':10
                          ,'GROCERY I':3
                          ,'MEATS':3
                          ,'POULTRY':3
                          ,'PREPARED FOODS':2
                          ,'SEAFOOD':5}
plt.figure(figsize=(12, 8))

for product_type in shelf_life_mapping:
    plt.plot(data_result_transposed.index,␣
 ↪data_result_transposed[product_type], label=product_type)

plt.xlabel('Date')
plt.ylabel('% Deviation')
plt.title('Deviation of Top Selling Product Types')

plt.legend(loc='upper right', bbox_to_anchor=(1.2, 1))
```

```
plt.show()

# Possible 0 to minimal wastage
```



6.2.5 Out of Stock Evaluation

[55]:
```
oos_dataset = deep_result.copy()
oos_dataset['predicted_sales'] = oos_dataset['predicted_sales']
oos_dataset['oos_flag'] =␣
 ↪((oos_dataset['predicted_sales']-oos_dataset['sales'])<0).astype(int)
oos_report = np.round(oos_dataset.pivot_table(index = ['product_type'], columns␣
 ↪= ['date'], values = ['oos_flag'], aggfunc = np.mean)*100,2)
oos_report
```

[55]:

	oos_flag				\
date	2017-08-01	2017-08-02	2017-08-03	2017-08-04	
product_type					
AUTOMOTIVE	50.00	48.15	40.74	59.26	
BABY CARE	14.81	14.81	7.41	14.81	
BEAUTY	48.15	38.89	44.44	48.15	
BEVERAGES	79.63	64.81	46.30	44.44	
BOOKS	0.00	0.00	0.00	0.00	
BREAD/BAKERY	57.41	61.11	37.04	51.85	
CELEBRATION	57.41	33.33	35.19	42.59	

CLEANING	94.44	83.33	40.74	24.07
DAIRY	59.26	35.19	40.74	33.33
DELI	57.41	33.33	25.93	31.48
EGGS	53.70	48.15	31.48	62.96
FROZEN FOODS	70.37	64.81	51.85	50.00
GROCERY I	64.81	57.41	42.59	40.74
GROCERY II	79.63	68.52	57.41	59.26
HARDWARE	48.15	33.33	25.93	40.74
HOME AND KITCHEN I	59.26	37.04	35.19	33.33
HOME AND KITCHEN II	59.26	51.85	38.89	46.30
HOME APPLIANCES	18.52	16.67	12.96	11.11
HOME CARE	77.78	48.15	33.33	11.11
LADIESWEAR	38.89	33.33	33.33	22.22
LAWN AND GARDEN	37.04	35.19	24.07	24.07
LINGERIE	38.89	35.19	42.59	37.04
LIQUOR,WINE,BEER	31.48	62.96	48.15	29.63
MAGAZINES	61.11	46.30	31.48	40.74
MEATS	62.96	62.96	44.44	50.00
PERSONAL CARE	70.37	57.41	29.63	50.00
PET SUPPLIES	44.44	38.89	42.59	35.19
PLAYERS AND ELECTRONICS	53.70	31.48	40.74	48.15
POULTRY	72.22	59.26	57.41	42.59
PREPARED FOODS	44.44	35.19	29.63	42.59
PRODUCE	68.52	62.96	46.30	48.15
SCHOOL AND OFFICE SUPPLIES	40.74	37.04	38.89	38.89
SEAFOOD	37.04	29.63	29.63	20.37

\

date	2017-08-05	2017-08-06	2017-08-07	2017-08-08
product_type				
AUTOMOTIVE	33.33	61.11	59.26	53.70
BABY CARE	9.26	14.81	18.52	16.67
BEAUTY	62.96	53.70	61.11	64.81
BEVERAGES	24.07	48.15	37.04	37.04
BOOKS	0.00	0.00	0.00	0.00
BREAD/BAKERY	35.19	46.30	50.00	66.67
CELEBRATION	40.74	40.74	35.19	37.04
CLEANING	14.81	16.67	29.63	35.19
DAIRY	11.11	37.04	37.04	38.89
DELI	31.48	31.48	35.19	25.93
EGGS	31.48	53.70	37.04	44.44
FROZEN FOODS	59.26	51.85	50.00	51.85
GROCERY I	40.74	51.85	70.37	50.00
GROCERY II	46.30	57.41	64.81	61.11
HARDWARE	33.33	42.59	42.59	38.89
HOME AND KITCHEN I	42.59	44.44	46.30	37.04
HOME AND KITCHEN II	40.74	50.00	42.59	51.85

HOME APPLIANCES	20.37	16.67	16.67	14.81
HOME CARE	9.26	25.93	29.63	25.93
LADIESWEAR	16.67	22.22	25.93	37.04
LAWN AND GARDEN	35.19	29.63	25.93	33.33
LINGERIE	42.59	37.04	31.48	40.74
LIQUOR,WINE,BEER	85.19	70.37	9.26	25.93
MAGAZINES	57.41	61.11	57.41	53.70
MEATS	24.07	55.56	66.67	44.44
PERSONAL CARE	35.19	40.74	61.11	50.00
PET SUPPLIES	44.44	44.44	44.44	44.44
PLAYERS AND ELECTRONICS	37.04	44.44	55.56	37.04
POULTRY	14.81	50.00	64.81	61.11
PREPARED FOODS	24.07	37.04	44.44	40.74
PRODUCE	16.67	64.81	59.26	59.26
SCHOOL AND OFFICE SUPPLIES	29.63	22.22	27.78	42.59
SEAFOOD	22.22	31.48	29.63	40.74

\

date	2017-08-09	2017-08-10	2017-08-11	2017-08-12
product_type				
AUTOMOTIVE	46.30	40.74	64.81	31.48
BABY CARE	14.81	9.26	22.22	0.00
BEAUTY	59.26	57.41	61.11	51.85
BEVERAGES	37.04	44.44	42.59	14.81
BOOKS	1.85	0.00	0.00	0.00
BREAD/BAKERY	50.00	46.30	59.26	22.22
CELEBRATION	33.33	62.96	42.59	24.07
CLEANING	35.19	12.96	20.37	11.11
DAIRY	29.63	35.19	68.52	20.37
DELI	35.19	46.30	42.59	7.41
EGGS	48.15	57.41	70.37	22.22
FROZEN FOODS	50.00	59.26	75.93	31.48
GROCERY I	66.67	53.70	75.93	29.63
GROCERY II	38.89	55.56	51.85	42.59
HARDWARE	38.89	38.89	42.59	40.74
HOME AND KITCHEN I	46.30	55.56	68.52	35.19
HOME AND KITCHEN II	42.59	38.89	55.56	27.78
HOME APPLIANCES	5.56	5.56	16.67	7.41
HOME CARE	20.37	29.63	38.89	9.26
LADIESWEAR	29.63	14.81	33.33	20.37
LAWN AND GARDEN	27.78	37.04	38.89	27.78
LINGERIE	37.04	51.85	57.41	38.89
LIQUOR,WINE,BEER	48.15	64.81	74.07	64.81
MAGAZINES	46.30	51.85	46.30	40.74
MEATS	50.00	48.15	46.30	37.04
PERSONAL CARE	51.85	38.89	75.93	22.22
PET SUPPLIES	44.44	33.33	61.11	27.78

PLAYERS AND ELECTRONICS	57.41	55.56	51.85	33.33
POULTRY	50.00	62.96	66.67	27.78
PREPARED FOODS	42.59	50.00	55.56	29.63
PRODUCE	38.89	62.96	70.37	35.19
SCHOOL AND OFFICE SUPPLIES	35.19	33.33	55.56	50.00
SEAFOOD	29.63	20.37	33.33	22.22

date	2017-08-13	2017-08-14	2017-08-15
product_type			
AUTOMOTIVE	46.30	37.04	62.96
BABY CARE	12.96	9.26	12.96
BEAUTY	48.15	51.85	61.11
BEVERAGES	27.78	31.48	31.48
BOOKS	0.00	0.00	0.00
BREAD/BAKERY	38.89	40.74	51.85
CELEBRATION	44.44	57.41	37.04
CLEANING	24.07	33.33	40.74
DAIRY	29.63	29.63	46.30
DELI	25.93	37.04	51.85
EGGS	50.00	42.59	61.11
FROZEN FOODS	25.93	46.30	53.70
GROCERY I	25.93	44.44	61.11
GROCERY II	40.74	46.30	51.85
HARDWARE	37.04	35.19	33.33
HOME AND KITCHEN I	37.04	59.26	57.41
HOME AND KITCHEN II	37.04	57.41	53.70
HOME APPLIANCES	1.85	5.56	5.56
HOME CARE	14.81	20.37	31.48
LADIESWEAR	16.67	31.48	37.04
LAWN AND GARDEN	25.93	31.48	33.33
LINGERIE	35.19	44.44	44.44
LIQUOR,WINE,BEER	51.85	14.81	38.89
MAGAZINES	50.00	72.22	66.67
MEATS	35.19	53.70	74.07
PERSONAL CARE	18.52	40.74	57.41
PET SUPPLIES	31.48	37.04	40.74
PLAYERS AND ELECTRONICS	50.00	48.15	51.85
POULTRY	35.19	61.11	61.11
PREPARED FOODS	38.89	46.30	50.00
PRODUCE	38.89	44.44	64.81
SCHOOL AND OFFICE SUPPLIES	37.04	38.89	38.89
SEAFOOD	25.93	31.48	35.19