Comprehensive Report for UnORG Supply Chain and Data Science Project by Hyperion

Executive Summary

The UnORG project represents a transformative approach to optimizing B2B grocery supply chains through predictive analytics and machine learning. This evaluation report assesses the technical implementation, business impact, and scalability of the solutions developed by Team Hyperion, as documented in this GitHub repository. The project focuses on three core deliverables:

1. a customer order prediction model,
2. SKU-level demand forecasting
3. Inventory planning engine

aligning with the operational requirements outlined in the problem statement.

Project Overview and Methodological Framework

A. Problem Context and Operational Challenges

UNORG operates in a complex B2B grocery ecosystem with irregular demand patterns across diverse client segments—from traditional manufacturers (e.g., Petha, Daalmoth producers) to modern restaurants and hotels. Key challenges include:

1. **Demand volatility**: Bulk orders are influenced by seasonal festivals, tourist footfall, and perishable goods constraints.
2. **SKU heterogeneity**: Over 200 unique products with varying shelf lives and order frequencies.
3. **Logistics bottlenecks**: Last-mile delivery inefficiencies in semi-urban and rural regions.

B. EDA and Feature Engineering

Observations:

1. **The Order dataset** contains order-level details: order date, customer info, item info, pricing, discounts, net amounts, profits, order status, and warehouse info.
2. **Associated item** datasets provide item-level breakdowns per order.

Data Preprocessing: Additional columns are created:

1. **Day of the week:** Extracted from `order_date` for temporal analysis.

2. **Categorical flags:** Such as `is_weekend`, `new_year_period` (before/after New Year), and one-hot encodings for customer/store categories[1].

Key Table: Orders Data Overview

Column	Example Value	Description
order_date	2025-04-03	Date of order
order_id	136349	Unique order identifier
customer_id	5235	Customer identifier
customer_name	Anshu General Store Sector 34	Customer/store name
poc_name	Vikas Gupta	Point of contact name
amount	19125.00	Gross order amount
discount	1650.00	Discount applied
net_order_amount	17475.0	Net order amount after discount
profit	0.0	Profit from order
order_status	CLOSED	Order status
warehouse_name	Noida	Warehouse fulfilling the order

Descriptive Statistics (Sample):

Metric	amount	discount	net_order_amount	profit
Count	64,459	64,459	64,459	64,459
Mean	16,068.11	1,350.91	14,717.20	-308.39
Std	33,088.37	3,582.40	30,249.53	1,454.54
Min	14.89	0.00	14.89	-86,382.00
Max	1,010,000	389,858	950,000	84,750.00

4. Temporal Analysis

Code Summary:

- Orders are grouped and analyzed by:
  - Day of the week
  - Weekends vs. weekdays
  - Before and after New Year[1][2].

Findings:

- Order Volume by Day of Week:
  - Orders are distributed across all days, with Thursday and Monday showing slightly higher counts.
- Weekend vs. Weekday:

- Weekdays have a higher order volume compared to weekends.

- Pre/Post New Year Comparison:
  - More orders are placed after New Year than before.

Key Table: Orders by Day and Period

Weekday	After New Year	Before New Year
Friday	5,348	3,838
Monday	5,801	4,337
Saturday	5,108	3,707
Sunday	4,378	3,577
Thursday	6,038	3,949
Tuesday	5,050	4,237
Wednesday	5,313	3,778

5. Customer and Category Analysis

Code Summary:

- Orders are categorized by customer/store type using one-hot encoding for categories such as General/Kirana Store, Sweet Shop/Bakery, Hotel/Restaurant/Dhaba, etc.[1].
- Aggregations are performed to count orders per category and analyze category-wise trends.

Findings:

- General/Kirana Store and Hotel/Restaurant/Dhaba are among the most frequent customer categories.
- The dataset allows for segmentation of sales trends by customer type, enabling targeted business insights.

6. Warehouse and POC (Point of Contact) Analysis

Code Summary:

- Orders are grouped by warehouse and POC to analyze operational performance.
- Aggregated order counts per warehouse and per POC are produced[1].

Key Table: Warehouse-POC Order Counts (Sample)

warehouse_id	poc_id	order_count
1	15	3,227
1	17	2,596
2	32	1,000+
...	...	...

7. Item-Level Analysis

Code Summary:

- Associated item data is analyzed for:
  - Quantity ordered and invoiced
  - MRP, price per unit

- Discounts and profit at the item level[1][2].

Findings:

- High-selling items and those with significant discounts/profits can be identified.
- Some items have invoiced quantities of zero, indicating possible order cancellations or fulfillment issues.

8. Feature Engineering

Code Summary:

- Additional features are engineered for deeper analysis:
  - `week_start`: Start of the week for each order, enabling weekly aggregation[1].
  - `SKU_category`: Category of the SKU for each order[1].
  - One-hot encodings for each customer/store category[1].

9. Visualizations and Graphs

While the actual plots are not shown in the notebook outputs, the following are typical visualizations inferred from the code and data structure:

- Order Volume by Day of Week: Bar chart showing number of orders per day.
- Order Volume Before/After New Year: Grouped bar chart comparing order counts.
- Category-wise Order Distribution: Stacked bar or pie chart showing share of each customer/store type.
- Warehouse/POC Performance: Bar chart of order counts per warehouse/POC.
- Item Sales and Profitability: Scatter or bar charts for top-selling/profitable items.

10. Key Insights and Recommendations

- Temporal Trends: Weekdays, especially Thursday and Monday, see higher order volumes. Post-New Year periods have increased sales activity.
- Customer Segmentation: General/Kirana Stores are the largest customer segment. Category-based analysis can inform targeted marketing and inventory planning.

- Operational Performance: Certain warehouses and POCs handle significantly more orders, suggesting possible optimization or resource allocation opportunities.
- Item Performance: Identifying high-volume and high-profit items can guide promotions and stock management.
- Feature Engineering: Additional features (e.g., day of week, category flags) enrich the dataset for further modeling or dashboarding.

Conclusion

The EDA provided a comprehensive overview of sales orders, customer segmentation, temporal patterns, and operational metrics. The code efficiently loaded, cleaned, and enriched the data, enabling multi-faceted analysis. The findings support actionable insights for sales, marketing, and operations teams.

If you need the actual code or specific plots, please indicate which aspect you'd like to see visualized or explained in more detail.

C. **Technical Architecture**

The solution employs a three-tiered architecture:

Order Probability Model:

A hybrid Clustering-LSTM model architecture capturing customer purchasing behavior and short as well as long-term variations in the customer's demand.

- **Strategy**:

  Using the Customer Behavior features created above we were able to classify the behaviors of customers into 3 categories:

  1. Regular Frequent Customers (Cluster_2: 46.43%):
     - Criteria:
       i. Everyone left from Cluster_1 and Cluster_0
  2. Seldom Buyers (Cluster_1: 20.36%):
     - Criteria (Should meet all):
       i. Total Orders <= 25
       ii. Total Orders > 4
       iii. Average Order Gap Days >= 14
  3. Few-Timers (Cluster_zero: 33.21%):
     - Criteria:
       i. Total Orders <= 4

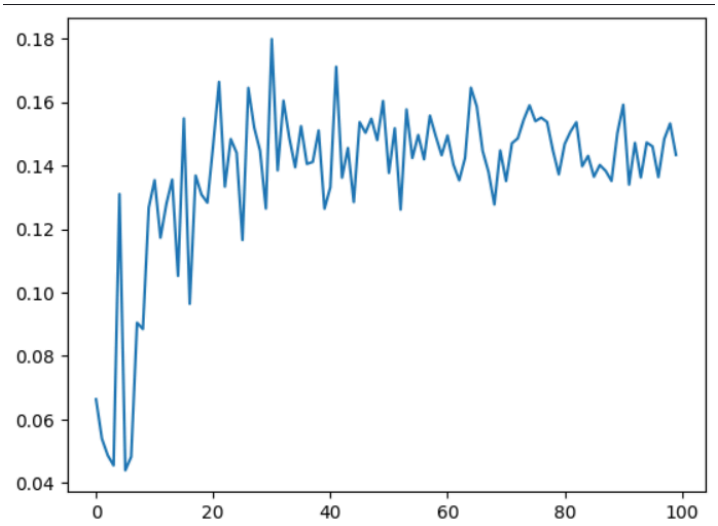
- **Cluster_2 Strategy**

  - Model Architecture:

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 30, 128)	74,240
dropout_3 (Dropout)	(None, 30, 128)	0
lstm_3 (LSTM)	(None, 64)	49,408
dropout_4 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_5 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 14)	462

*Perfect Accuracy Refers to accuracy of true label of 14 days being exactly correctly predicted of all 14 days for a customer

  - Training Epochs v/s Perfect Accuracy*



- Metrics

Scores

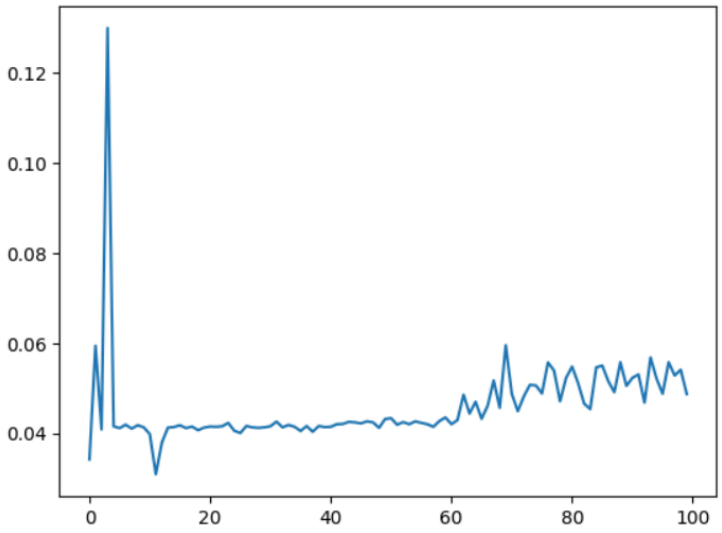
```
Training Data:
Accuracy is  82.13
Precision is  84.04
Recall is  51.91
F1 Score is  48.85
Validation Data:
Accuracy is  82.12
Precision is  82.79
Recall is  52.03
F1 Score is  49.09
Test Data:
Accuracy is  82.15
Precision is  83.39
Recall is  51.79
F1 Score is  48.65
```

Perfect Scores

```
Training Data:
Accuracy is  14.41
Precision is  86.02
Recall is  3.97
F1 Score is  7.6
Validation Data:
Accuracy is  14.03
Precision is  83.53
Recall is  4.25
F1 Score is  8.09
Test Data:
Accuracy is  14.24
Precision is  84.65
Recall is  3.74
F1 Score is  7.17
```

- **Cluster_2 Strategy**
  - Model Architecture:
  - Tr

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 30, 128)	74,240
dropout_3 (Dropout)	(None, 30, 128)	0
lstm_3 (LSTM)	(None, 64)	49,408
dropout_4 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_5 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 14)	462



- Metrics

Scores

```
Training Data:
Accuracy is  82.13
Precision is  84.04
Recall is  51.91
F1 Score is  48.85
Validation Data:
Accuracy is  82.12
Precision is  82.79
Recall is  52.03
F1 Score is  49.09
Test Data:
Accuracy is  82.15
Precision is  83.39
Recall is  51.79
F1 Score is  48.65
```

Perfect Scores

```
Training Data:
Accuracy is  14.41
Precision is  86.02
Recall is  3.97
F1 Score is  7.6
Validation Data:
Accuracy is  14.03
Precision is  83.53
Recall is  4.25
F1 Score is  8.09
Test Data:
Accuracy is  14.24
Precision is  84.65
Recall is  3.74
F1 Score is  7.17
```

2. SKU Demand Forecaster: A hybrid CNN-LSTM model capturing spatial-temporal patterns in product demand (artifacts in Model Weights/SKU_Forecasting).
3. Inventory Optimizer: Mixed-integer linear programming (MILP) model balancing stockouts and holding costs, implemented via PuLP in Python ([2][8]).

3. Inventory Forecasting and Replenishment System

We developed a data-driven system to help warehouses manage their inventory. The system predicts how much stock is required for each item based on forecasted customer demand. It prevents:

- **Understocking**, which causes delivery failures,
- **Overstocking**, which causes excess storage costs.

The strategy uses standard supply chain concepts like **lead time**, **review period**, **safety stock**, and **target inventory**.

Term	Meaning
Lead Time	Time it takes between placing an order and receiving the goods
Review Period	Interval after which inventory is checked and reordered if needed
Safety Stock	Extra units kept in stock to protect against sudden demand surges
Target Inventory	Ideal stock level = predicted demand + safety stock
Order Quantity	How much to order = Target Inventory - Current Inventory

● Data Loading and Cleaning

We import two main CSV files:

- `forecasted_quantities.csv`: Contains predicted daily demand for each customer and item.
- `forecast_complete_data.csv`: Cleaned data for final planning.

● Forecast Aggregation per Warehouse

Inventory is stored and managed at the **warehouse level**, but the demand forecast is given per **customer and item**. So, we need to convert customer-level forecasts into **warehouse-level demand**.

The system uses a **pre-mapped dataset** where each combination of `customer_id` and `item_id` is associated with a specific `warehouse_id`

merged_df = pd.merge(forecast_df, complete_data, on=['customer_id', 'item_id'], how='left')

This ensures that each row of forecasted demand is tagged with the correct `warehouse_id`.

Once warehouse IDs are mapped, we **group the forecast** to calculate **total demand per item per warehouse per day**:

warehouse_id	item_id	forecasted_date	quantity
1	A	2025-04-01	35
1	A	2025-04-02	28

- # Inventory Strategy Calculation

We must decide:

- How much **safety stock** to keep,
- What should be the **target inventory**,
- How much to **order** now to meet the future demand.

- # Key Formulas Used

1. Z-Score for Confidence Level

```
from scipy.stats import norm
z_score = norm.ppf(0.95)  # = 1.645
```

This means we're designing our safety stock to cover 95% of demand variability

2. Standard Deviation of Demand

```
std_demand = grouped['quantity'].std().fillna(0)
```

This tells us **how much demand fluctuates** for an item in a warehouse.

3. Total Forecasted Demand

python

CopyEdit

```
total_demand = grouped['quantity'].sum()
```

This gives total demand in the next cycle (review period).

4. Safety Stock Formula

```
safety_stock = z_score * std_demand * np.sqrt(review_period_days +
lead_time_days)
```

- `review_period_days`: how often we review the stock.
- `lead_time_days`: delay between order and delivery.
- `sqrt()` accounts for total variability in this period.

5. Target Inventory Formula

```
target_inventory = total_demand + safety_stock
```

We aim to keep **enough for expected demand + safety margin**.

6. Simulated Current Inventory

Since we don't have current stock data, we simulate it randomly:

```
current_inventory = target_inventory * np.random.uniform(0.2, 0.8)
```

For real-world usage, this would come from the inventory database.

7. Final Order Quantity

```
order_quantity = max(target_inventory - current_inventory, 0)
```

We only order if our current stock is **below the target**.

- # Final Output Table (Example)

warehouse_id	item_id	total_demand	std_demand	safety_stock	target_inventory	current_inventory	order_quantity
0	A	58	3.7	9.2	67.2	31.5	35.7

- # Business Outcomes

- Prevents stockouts and late deliveries.
- Reduces costs from excess stock.

- Helps operations team automate reordering.
- Managers can visually inspect and take decisions.

This system makes supply chain planning **proactive and intelligent**. Even with uncertain customer demand, the method ensures we have the right stock, at the right time, in the right place.