# Predict product stock levels hourly to optimize procurement

Cognizant's technology-led clients, Gala Groceries

# About

## Challenge:

Gala Groceries, known for local, fresh produce, needs help finding the sweet spot for stocking perishable items. Overstock = waste. Understock = lost customers.

## Goal:

Improve inventory management to minimize waste and satisfy customers.

## Hope:

Cognizant can provide the tech-savvy grocery chain with a solution to optimize its supply chain and maintain its fresh produce edge.

# Table of contents


Month
Year
March
April
May
Total Balance
$ 20,000
50%
Income
$ 2,200
80%
7
8
9

# 01

# EDA

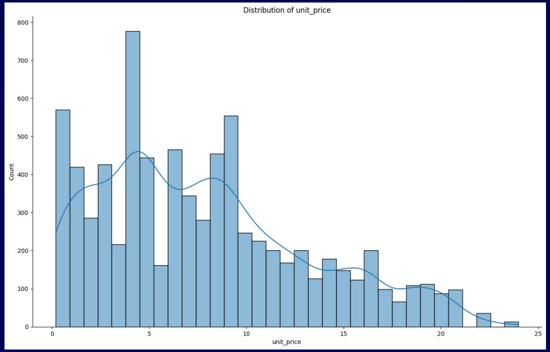
# The sample data

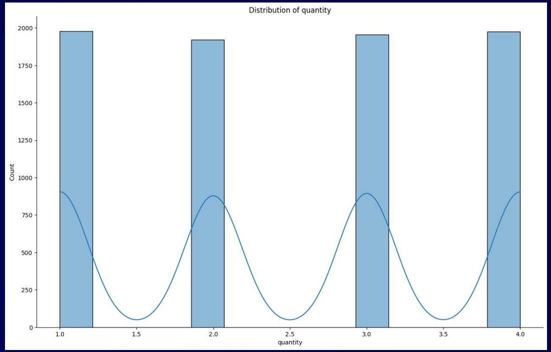
- transaction_id = this is a unique ID that is assigned to each transaction
- timestamp = this is the datetime at which the transaction was made
- product_id = this is an ID that is assigned to the product that was sold. Each product has a unique ID
- category = this is the category that the product is contained within
- customer_type = this is the type of customer that made the transaction
- unit_price = the price that 1 unit of this item sells for
- quantity = the number of units sold for this product within this transaction
- total = the total amount payable by the customer
- payment_type = the payment method used by the customer

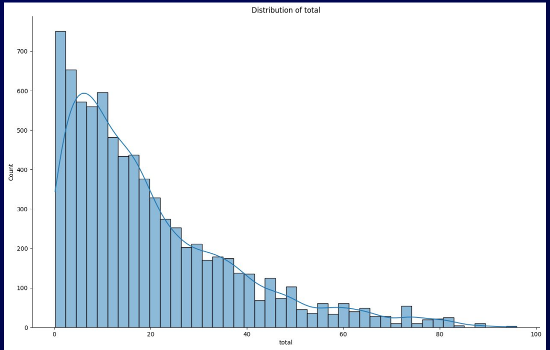
```
RangeIndex: 7829 entries, 0 to 7828
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   transaction_id  7829 non-null   object 
 1   timestamp       7829 non-null   object 
 2   product_id      7829 non-null   object 
 3   category        7829 non-null   object 
 4   customer_type   7829 non-null   object 
 5   unit_price      7829 non-null   float64
 6   quantity        7829 non-null   int64  
 7   total           7829 non-null   float64
 8   payment_type    7829 non-null   object 
dtypes: float64(2), int64(1), object(6)
memory usage: 550.6+ KB
```

# Analyzing the data

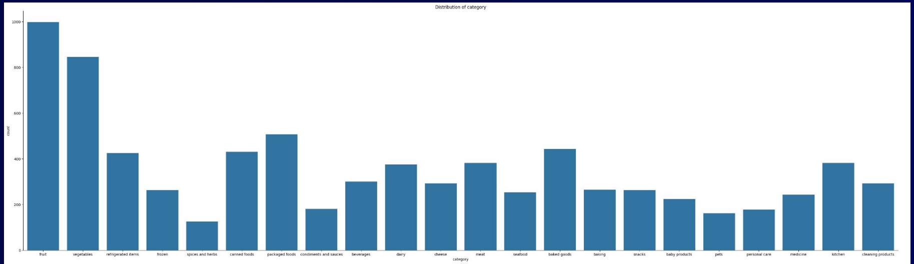
- **Distribution of prices and quantities:**
  - unit_price: Positively skewed with more sales for cheaper products.
  - quantity: Almost uniform distribution with 1-4 units bought equally often.
  - total: Even more positively skewed than unit_price, indicating more transactions with lower totals.

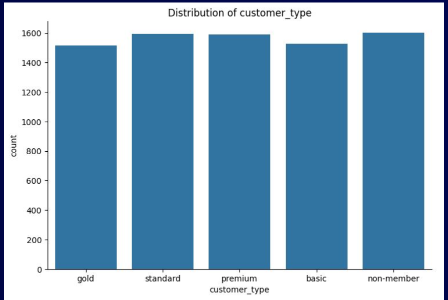


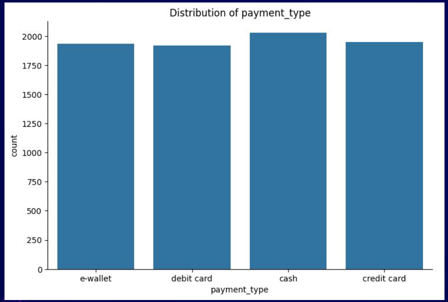



# Analyzing the data

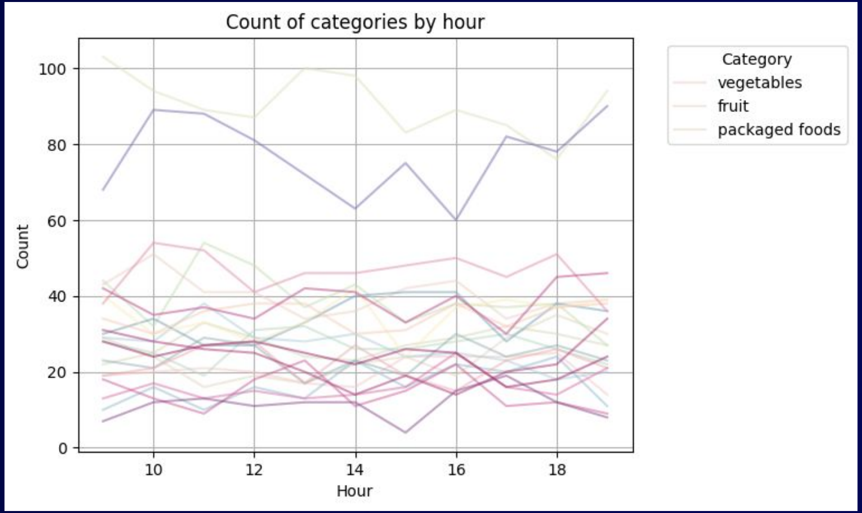



- **Categorical variables:**
  - product_id: 300 unique products, with one product sold 11 times and another just 3 times.
  - category: 22 unique categories, with "fruit" and "vegetables" most popular and "spices and herbs" least popular.
  - customer_type: 5 evenly distributed types, with non-member slightly more frequent.
  - payment_type: 4 evenly distributed types, with cash most frequent and debit card least frequent.
  - timestamp: Lots of unique values suggesting a datetime format, with busiest hours around 11th, 16th, and 18th of the day.

# Analyzing the data




- **Categorical variables:**

  We can see clearly that vegetables and fruit are the most buyable items in all hour.

# Limitations and next steps

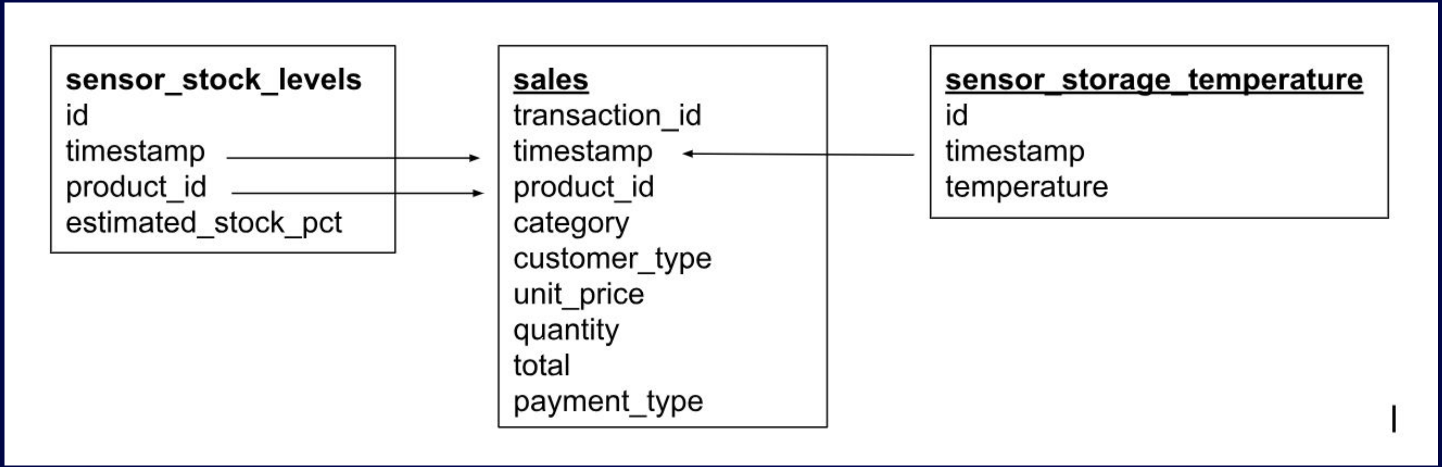
- Data limitations:
    - Sample size: Only 1 week of data from 1 store, making insights unreliable.
    - Problem statement lacking specificity: The current "better stocking" goal is too broad.
    - Feature limitations: More features relevant to the specific problem might be needed.
- Recommendations:
    - Gather more data: Collect data from multiple stores and a longer period to gain statistically significant insights.
    - Refine the problem statement: Define a specific goal like optimizing stock for peak hours or predicting demand for high-value items.

Based on our recommendations, Client wants to focus on the following problem statement:

*“Can we accurately predict the stock levels of products based on sales data and sensor data on an hourly basis in order to more intelligently procure products from our suppliers?”*

# Data model diagram

Client adds valuable data: **Sensor readings from warehouse & store stock levels**



# Hypothesis

- **Product price**: Product price may be related to product inventory levels. Cheaper products may have higher demand and therefore need to be stocked more.
- **Product quantity:** Product quantity may be related to product inventory levels. Products sold in larger quantities may need to be stocked more.
- **Time of day:** Time of day may be related to product inventory levels. Products that are sold more at certain times of the day may need to be stocked more.
- **Sensor data:** Sensor data about the warehouse and store may be related to product inventory levels. For example, warehouse temperature can affect product shelf life and therefore needs to be considered when forecasting demand.


Month
Year
March
May
Total Balance
$ 20,000
50%
Income
$ 2,200
80%
7
8
9

# 02

# Modeling

# Strategic Plan

## Data Pipeline

- **Merge**: Combine data from "sales", "sensor_storage_temperature", and "sensor_stock_levels" based on product_id and location.
- **Clean & Transform:** Handle missing values, outliers, and feature scaling.
- **Engineer Features**: Extract relevant features (e.g., demand seasonality, temperature impact).

## Modeling Workflow

- **Experiment**: Train and compare various predictive models (e.g., demand forecasting, spoilage prediction).
- **Cross-validate**: Assess model performance on unseen data.
- **Evaluate & Iterate**: Optimize hyperparameters and select the best performing model.
- **Productionize:** Package the chosen model as an API for deployment.

## Deployment & Monitoring

- **QA & DevOps:** Validate model performance and integrate with existing systems.
- **Monitor & Feedback:** Continuously monitor model performance and iterate based on new data.

# Merged Data

Combined all 3 datasets using timestamps.

```
Int64Index: 10845 entries, 0 to 10844
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   timestamp            10845 non-null  datetime64[ns]
 1   product_id           10845 non-null  object
 2   estimated_stock_pct  10845 non-null  float64
 3   quantity             10845 non-null  float64
 4   temperature          10845 non-null  float64
 5   category             10845 non-null  object
 6   unit_price           10845 non-null  float64
dtypes: datetime64[ns](1), float64(4), object(2)
memory usage: 677.8+ KB
```

# Feature Engineering

- Transformed timestamps to day, month, hour.
- Created numerical features from categorical data.

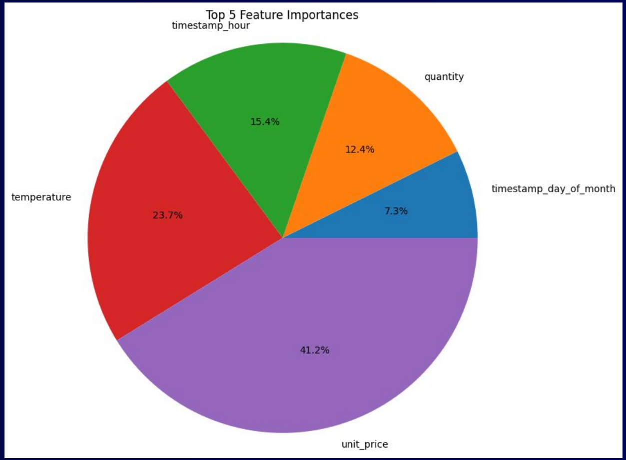
```
Int64Index: 10845 entries, 0 to 10844
Data columns (total 30 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   product_id                      10845 non-null  object
 1   estimated_stock_pct             10845 non-null  float64
 2   quantity                        10845 non-null  float64
 3   temperature                     10845 non-null  float64
 4   unit_price                      10845 non-null  float64
 5   timestamp_day_of_month          10845 non-null  int64
 6   timestamp_day_of_week           10845 non-null  int64
 7   timestamp_hour                  10845 non-null  int64
 8   category_baby products          10845 non-null  uint8
 9   category_baked goods            10845 non-null  uint8
 10  category_baking                 10845 non-null  uint8
 11  category_beverages              10845 non-null  uint8
 12  category_canned foods           10845 non-null  uint8
 13  category_cheese                 10845 non-null  uint8
 14  category_cleaning products      10845 non-null  uint8
 15  category_condiments and sauces  10845 non-null  uint8
 16  category_dairy                  10845 non-null  uint8
 17  category_frozen                 10845 non-null  uint8
 18  category_fruit                  10845 non-null  uint8
 19  category_kitchen                10845 non-null  uint8
 20  category_meat                   10845 non-null  uint8
 21  category_medicine               10845 non-null  uint8
 22  category_packaged foods         10845 non-null  uint8
 23  category_personal care          10845 non-null  uint8
 24  category_pets                   10845 non-null  uint8
 25  category_refrigerated items     10845 non-null  uint8
 26  category_seafood                10845 non-null  uint8
 27  category_snacks                 10845 non-null  uint8
 28  category_spices and herbs       10845 non-null  uint8
 29  category_vegetables             10845 non-null  uint8
dtypes: float64(4), int64(3), object(1), uint8(22)
memory usage: 995.5+ KB
```

# Model Training

- Used Random Forest Regression with cross-validation (K-folds).
- Achieved consistent Mean Absolute Error (MAE) ~0.25 (50% accuracy).

```
Fold 1: MAE = 0.236
Fold 2: MAE = 0.237
Fold 3: MAE = 0.237
Fold 4: MAE = 0.237
Fold 5: MAE = 0.236
Fold 6: MAE = 0.236
Fold 7: MAE = 0.238
Fold 8: MAE = 0.236
Fold 9: MAE = 0.236
Fold 10: MAE = 0.236
Average MAE: 0.24
```

# Model Results




## Key Insights:

Model is robust: Performs consistently across different data samples.

Important features: Unit price, temperature, hour of day.

Prediction accuracy: Moderate (50%), needs improvement.

## Next Steps:

- Refine feature selection and engineering for better accuracy.
- Explore different machine learning models for potential improvement.
- Develop and implement an hourly stock prediction dashboard.

# Evaluate and improve

## Data Inclusion

- Deliveries: Directly impacts stock levels and likely changes based on weather patterns.
- Weather: Influences customer behavior and demand for specific products.

## Algorithm Exploration

- Explore:
    - Neural Networks: Highly complex and potentially more accurate than Random Forest.
- Considerations:
    - Interpretation difficulty compared to Random Forest.
    - Requires careful hyperparameter tuning and resource availability.

# Evaluate and improve

## Model Interpretation

- Limitations:
  - Neural networks offer less interpretability than Random Forest.
- Alternatives for Random Forest:
  - Feature importance analysis to identify key decision-making factors.

## Performance Optimization

Optimize Random Forest:

- Hyperparameter tuning: Adjust parameters like number of trees and features per split to improve MAE.

# Thank you for listening!