OptiStock

Smarter Inventory Decisions for Warehouse Operations

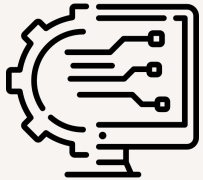
Mirna Hurst and
Sam Beilenson

*Mission Statement: To **harness real-time data** and predictive insights to **eliminate guesswork**, **reduce inefficiencies,** and empower warehouse managers to **make faster, smarter decisions.***

Today’s Agenda

1. The Problem
2. Future Vision
3. Tech Breakdown
4. Impact & Next Steps
5. Questions





The Problem

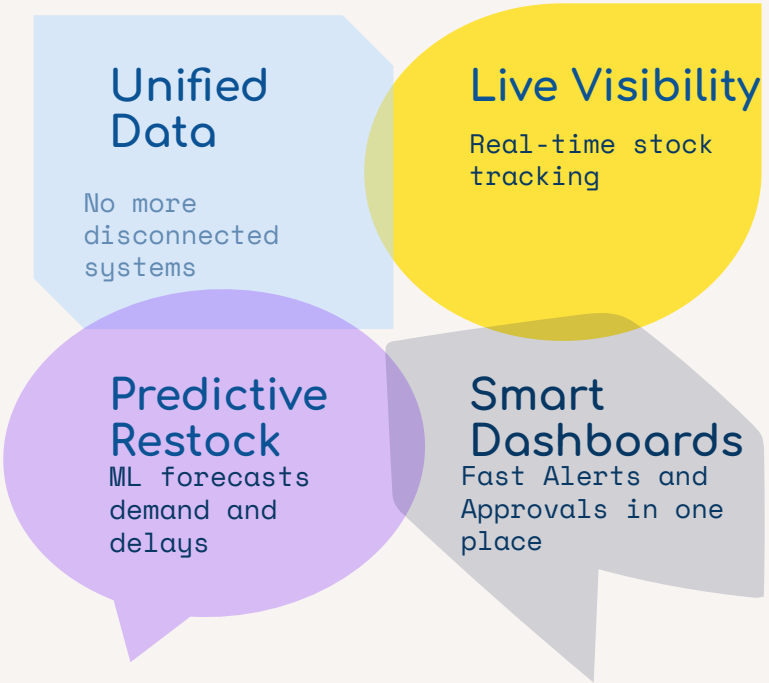
❌ **Manual Tracking & Disconnected Systems**

❌ **High Risk of Stockouts & Overstock**

❌ **Delays in Restocking Decisions**

❌ **Limited Insights from Data**

Future Vision



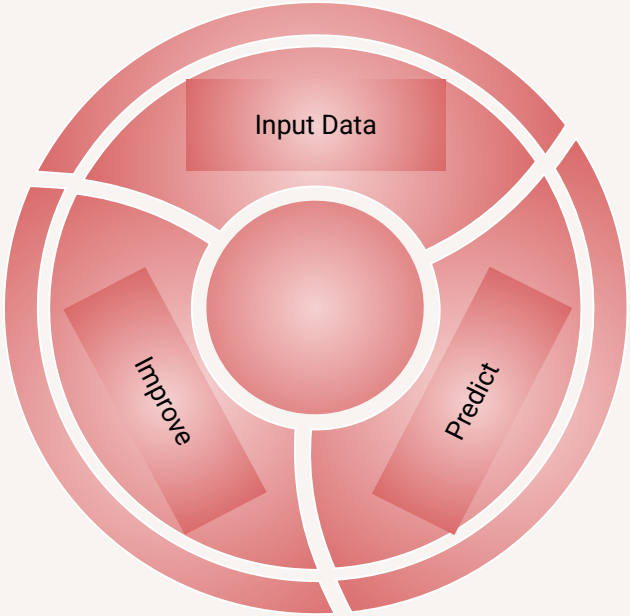

Our Solution

Solution Overview - ETL pipeline

Extract	Transform	Load
Pulls data from ERP, CRM, barcodes, and supplier systems	*Cleans and formats the data to be useful*	*Stores the organized data in one place for analysis*
(e.g., what was sold, when it arrived, what's running low)	*(removes duplicates, fixes errors, fills gaps)*	*(a centralized warehouse database that supports forecasting)*

Tech Breakdown

Solution Overview - Machine Learning Engine



Analyzes historical sales, supplier delays, and seasonal trends

Identifies patterns that correlate with future stockouts

Outputs recommended restock quantities with urgency levels (*Example output in appendices)*

Model is periodically updated (monthly) with new data to improve accuracy

- Our team is dedicated to the integrity of the ML engine and will run diagnostic tests as included with our current sales package.

Solution Overview- SQL Automation

✅ Check Stock Levels

Runs daily inventory scans using rules set by managers

🚩 Flag Inventory Risks

Detects low stock, slow-moving items, and supplier delays

📣 Send Alerts to Dashboard

Triggers urgency scores and restock suggestions in real time

Tech Breakdown

Solution Overview Manager Dashboard

📋 **Live Inventory View**
Real-time stock levels across all warehouses

✅ **Restock Suggestions**
Automatically generated from backend forecasting

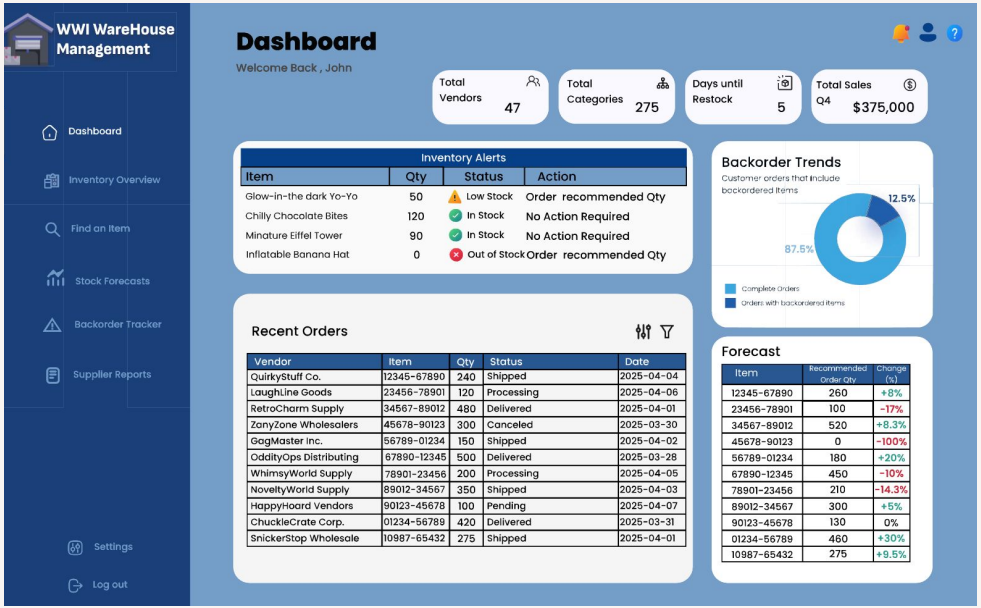
🔄 **Manual Overrides**
Approve or adjust quantities as needed—no code required

Tech Breakdown

What Managers Will See

Dashboard

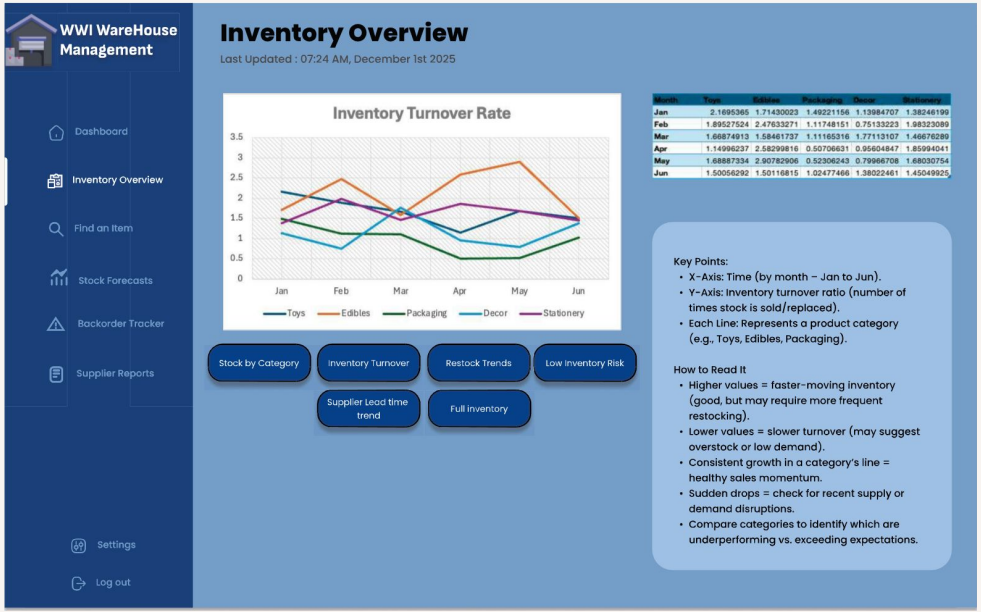
- Inventory Alerts
- Backorder Trends
- Recent orders



Tech Breakdown

Inventory Overview

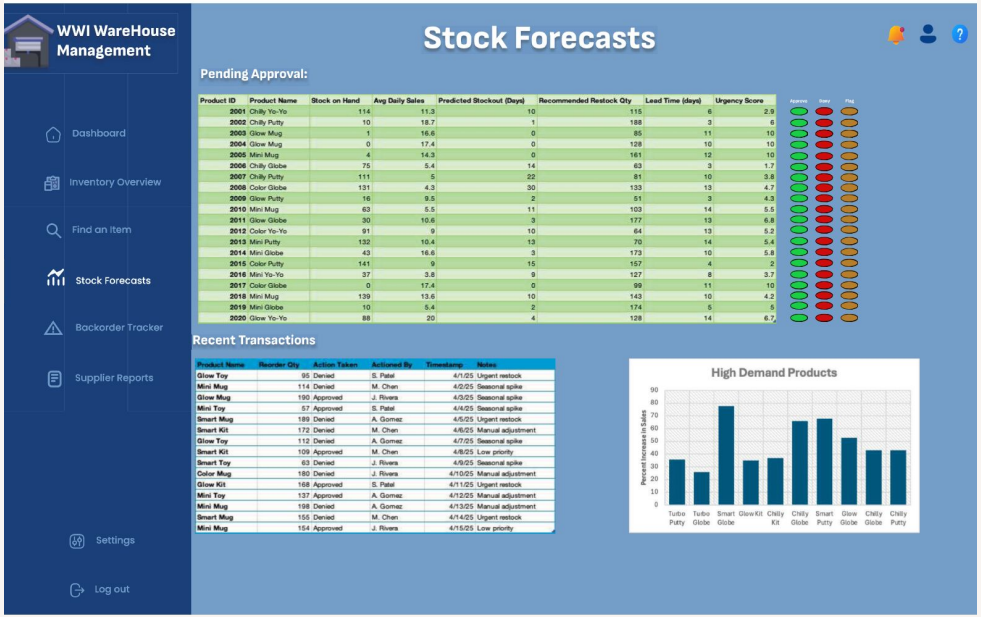
- Stock by Category
- Inventory Turnover
- Restock Trends
- Low inventory risk
- Supplier lead time trends
- Full inventory



Tech Breakdown

Stock Forecast

- Recommended restocking quantity
  - Approve, Deny, review recommendation
- Recent transactions



Tech Breakdown

Business Impact

Quantified Impacts for WWI

01	**30–40% Fewer Stockouts**	• More products available when customers want them
02	**20–25% Less Overstock Waste**	• Avoid tying up money in unnecessary inventory
03	**10–15 Labor Hours Saved Weekly**	• Staff spend less time counting, more time optimizing
04	**Full Visibility of Inventory**	• Real-time data means informed business decisions company-wide

Estimates based on benchmarks from companies of similar size

Impact & Next Steps

Phase Rollout Plan- Fast, Measured, Scalable

01 Pilot (Month 1-2)

- Implement in 1 high-volume warehouse
- Sync ERP + CRM data into database
- Test RFID/barcode tracking and alerts
- Collect manager feedback

02 Validation (Month 3-4)

- Add 2–3 additional warehouse locations
- Activate machine learning forecasts
- Train Operations staff

03 Full Launch (Month 5-6)

- Company-wide deployment
- Ongoing model retraining
- Weekly reporting
  - Monitor failure, review usage metrics

Impact & Next Steps

Next Steps

Let’s Take the Next Step

- ✅ **Select Pilot Warehouse**

  Identify a high-impact location to launch OptiStock

- 🛠️ **Further Customize the System**

  Integrate ERP/CRM data, configure alert thresholds, and finalize dashboard setup

- 👥 **Select Testing Cohort**

  Quick onboarding to ensure confident use of the platform

- 📊 **Launch & Measure Impact**

  Track key metrics: stockouts, labor hours saved, and inventory efficiency

- 🚀 **Iterate and Scale**

  Expand to additional warehouses and activate full automation features

Why OptiStock works

Smarter Inventory, Real Impact

What We Offer

- **AI-powered forecasting**- Predicts demand & restocks with data-driven accuracy
- **Live dashboards & alerts**- real time visibility, action-ready insights
- **Monitoring**- Flags issues before they disrupt operations
- **Scalable Data Infrastructure**- Built on ETL SQL for seamless integration

Why It Matters

- ★ 30-40% Fewer Stockouts
- ★ 20-25% Less Overstock Waste
- ★ 520-780 Labor Hours Saved annually
- ★ 100% Visibility Across Warehouses

Built to Grow With You:

Modular, easy to roll out, and built for continuous improvement. Measure. Iterate. Scale. We grow alongside you operations

Impact & Next Steps


Questions?

APPENDICES

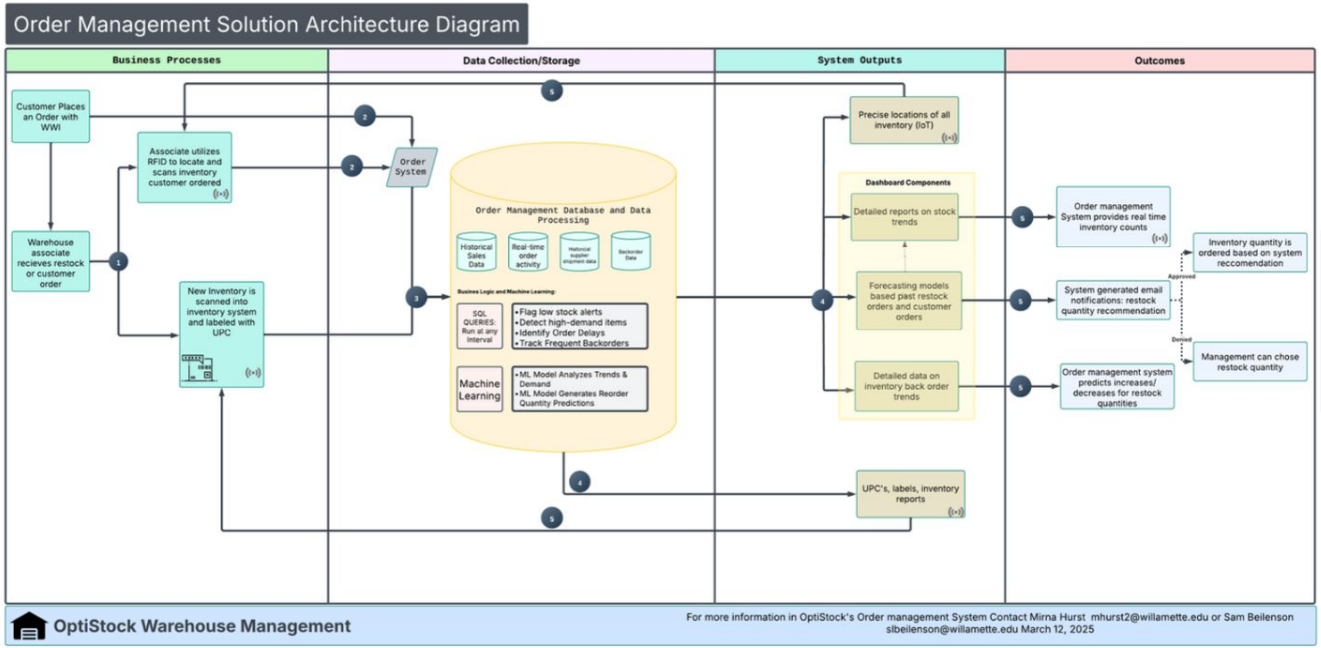
I. Solution Architecture Diagram (20)
II. ML inputs, outputs, code (21-23)
III. SQL Snippets (24)
IV. Python Auto Email Snippets (25)

Solution Architecture Diagram

Workflow

1. Warehouse associate scans new inventory into the system using a UPC. RFID updates stock location and inventory count in real time.
2. When a customer places an order, RFID tracks inventory movement, and the warehouse associate scans items for order fulfillment.
3. OptiStock Database integrates inventory data, customer orders, and stock movement into a centralized system.
4. The database generates real-time outputs, including RFID inventory tracking, stock trends, demand forecasting, and backorder risk alerts.
5. The order management system provides live inventory counts, predicts restocking needs based on historical data, and sends automated restock recommendations for manager approval.

Acronyms
UPC: Unique Product Code
RFID: Radio Frequency IDentification



WHAT THE MODEL SEES AND WHAT IT PREDICTS

Input:

Feature	Value (Example)
Product ID	104 – Chilly Chocolate Bites
Avg 30-day Sales	42 units/day
Stock on Hand	95 units
Lead Time (Supplier X)	7 days
Backorder Frequency	18%
Seasonality Factor	High (Q4 Holiday Spike)

WHAT THE MODEL SEES AND WHAT IT PREDICTS

Output : Our test RMSE of ±12 units means the model's restock predictions are, on average, within 12 units of the actual optimal reorder quantity

Product	Stock on Hand	30-Day Avg Sales	Predicted Stockout	Recommended Restock	Urgency Score*
Chilly Chocolate Bites	95	42 units/day	In 3 days	200 units	9.2 / 10
Inflatable Banana Hat	15	10 units/day	In 2 days	80 units	8.7 / 10
Mini Snow Globe	60	5 units/day	In 12 days	50 units	6.3 / 10
Magnetic Bookmark Set	180	12 units/day	In 15 days	100 units	4.5 / 10
LED Slime Putty	35	25 units/day	In 1 day	120 units	9.6 / 10

* Urgency Score is a weighted combination of predicted stockout timing, sales velocity, current stock level, and supplier lead time -- where products closer to running out, with faster sales and longer restock delays, receive a higher score (on a scale from 1 to 10).
* The model was trained on historical sales, inventory movement, supplier lead times, and backorder records, and used a regression-based machine learning model (such as XGBoost or Random Forest) to create these predictions.

CODE SNIPPET AND SUMMARY OF BENEFITS

```r
# Load libraries

library(randomForest)

library(tidyverse)



# Example dataset

set.seed(123)

data <- tibble(

  product_id = 1:100,

  avg_30_day_sales = runif(100, 10, 50),

  stock_on_hand = runif(100, 0, 200),

  backorder_frequency = runif(100, 0, 1),

  supplier_lead_time = runif(100, 3, 14),

  seasonality_factor = sample(c(0.8, 1.0, 1.2,
1.5), 100, replace = TRUE),

  restock_quantity = avg_30_day_sales *
seasonality_factor + backorder_frequency * 50
+ supplier_lead_time

)

# Split data
train_indices <- sample(1:nrow(data),
0.8 * nrow(data))
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

# Train Random Forest Model
rf_model <- randomForest(
  restock_quantity ~ avg_30_day_sales +
stock_on_hand + backorder_frequency +
    supplier_lead_time +
seasonality_factor,
  data = train_data,
  ntree = 100
)

# Predict
predictions <- predict(rf_model, newdata
= test_data)

# View results
results <- test_data %>%
  select(product_id, restock_quantity)
%>%
  mutate(predicted_restock =
round(predictions, 1))

print(head(results))
```

1. **Accurate Restock Predictions**
   - Anticipates demand and recommends reorder quantities with data-drive precision.
2. **Reduces Stockouts & Overstock**
   - Helps maintain optimal inventory levels, minimizing lost sales and exces holding costs.
3. **Prioritizes Urgent Inventory Needs**
   - Assigns urgency scores so managers know what to act on first.
4. **Learns from Trends & Seasonality**
   - Adjusts predictions based on historical sales patterns, supplier behavior, and seasonal shifts.
5. **Supports Smart Automation**
   - Feeds real-time dashboards and triggers alerts—reducing manual guesswork in restocking.

* The sample code is written in R and uses a random forest model based on an example dataset. In a real case, it would be using data from the ETL pipeline.

CODE SNIPPETS- SQL

Find Low-Stock Items

```
SELECT
    s.StockItemID,
    s.StockItemName,
    w.QuantityOnHand,
    s.RecommendedRetailPrice,
    s.LeadTimeDays
FROM Warehouse.StockItems AS s
JOIN Warehouse.StockItemHoldings AS w
    ON s.StockItemID = w.StockItemID
WHERE w.QuantityOnHand < 50
ORDER BY w.QuantityOnHand ASC;
```

Identify Items Frequently Backordered

```
SELECT
    s.StockItemID,
    s.StockItemName,
    COUNT(*) AS BackorderCount
FROM Sales.OrderLines AS ol
JOIN Warehouse.StockItems AS s
    ON ol.StockItemID = s.StockItemID
WHERE ol.PickedQuantity < ol.Quantity
GROUP BY s.StockItemID, s.StockItemName
HAVING COUNT(*) > 5;
```

Average Daily Sales Over the Last 30 Days

```
SELECT
    s.StockItemID,
    s.StockItemName,
    AVG(ol.Quantity) AS AvgDailySales
FROM Sales.OrderLines AS ol
JOIN Warehouse.StockItems AS s
    ON ol.StockItemID = s.StockItemID
JOIN Sales.Orders AS o
    ON ol.OrderID = o.OrderID
WHERE o.OrderDate >= DATEADD(DAY, -30,
GETDATE())
GROUP BY s.StockItemID, s.StockItemName;
```

CODE SNIPPET- PYTHON (AUTO EMAIL)

```
import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import
MIMEMultipart


# Email Configuration

sender_email = "optistock@example.com"

receiver_email =
"manager@wideworldimporters.com"

subject = "Low Stock Alert: Chilly
Chocolate Bites"

smtp_server = "smtp.gmail.com"

smtp_port = 587

password = "your_app_password"  # Use an
app password or environment variable
```

```
# Create Email Body

body = """
Dear Warehouse Manager,

The following item has fallen below the
critical stock threshold:

Product: Chilly Chocolate Bites
Current Stock: 32 units
Restock Threshold: 50 units
Recommended Reorder: 200 units

Please review and approve restocking as
soon as possible.

— OptiStock Automated Alert System
"""
```

```
# Build the Email

message = MIMEMultipart()
message["From"] = sender_email
message["To"] = receiver_email
message["Subject"] = subject
message.attach(MIMEText(body,
"plain"))

# Send Email

try:
    with smtplib.SMTP(smtp_server,
smtp_port) as server:
        server.starttls()
        server.login(sender_email,
password)
        server.sendmail(sender_email,
receiver_email, message.as_string())
        print(" Alert email sent
successfully.")
except Exception as e:
    print(f" Error sending email:
{e}")
```