Forecasting and operations report

Multi-Channel Reorder Forecasting System

A forecasting and decision tool for a direct-to-consumer health and wellness brand selling through Shopify and Square. Sales history from both channels is reconciled per SKU and run through a five-state reorder engine that produces a daily purchase queue, delivered as a Streamlit application on PostgreSQL.

Prepared by	Leah Hall, Leah Kat Insights
Date	June 2026
Contact	leah@leahkatinsights.com, leahkatinsights.com

Table of Contents

Introduction . . . 3

1 Executive summary . . . 3

Key findings . . . 3

2 Background and objective . . . 3

3 Data and method . . . 4

Pipeline . . . 4

Reorder states . . . 4

Testing . . . 5

4 Results . . . 5

5 Limitations . . . 6

6 Recommendations . . . 7

A Technical stack and repository . . . 7

B Glossary . . . 8

About the author . . . 8

Introduction

This report describes an inventory reorder system for a multi-channel direct-to-consumer brand: the operating problem, the forecasting method, the decision rules, and what the operator sees.

Figures are from a seeded demonstration dataset, staged as Aura Wellness: 200 SKUs across four categories and four vendors with 365 days of sales. The application renders identically on every run in demo mode and connects to live Shopify and Square stores with credentials.

Code, schema, and 25 tests are public on GitHub.

1. Executive summary

Of 200 SKUs, 56 are flagged for immediate reorder this cycle, with a recommended purchase order value of $1.18M grouped by vendor with minimum order quantities applied. Every recommendation traces to its inputs: velocity, days of supply, lead time, and outstanding purchase orders.

> **What the system replaces**
> Reorder timing is usually managed in a spreadsheet, which fails when a SKU turns seasonal or a second sales channel is added. This tool replaces the spreadsheet with a model whose every output can be opened to its inputs.

Key findings

56	**SKUs in the reorder-now state** out of 200, the morning action queue for the buyer.
$1.18M	**Recommended purchase order value**, grouped by vendor with each vendor's minimum order quantity already applied.
9.0x	**Velocity spread** between top- and bottom-quartile SKUs on a recency-weighted basis, which is why one reorder rule for all SKUs fails.
365	**Days of rolling history** reconciled from two sales channels into one per-SKU view.

2. Background and objective

A brand selling through its own web store and a point-of-sale system has two order histories that disagree on SKU naming, timing, and refunds. Buyers typically reconcile them by hand in a spreadsheet and set reorder points from gut feel. That works for a dozen products and breaks at two hundred, especially when demand shifts with the season.

The objective was a system that pulls both channels automatically, estimates demand per SKU with a method that adapts to recent sales, assigns each SKU a clear state, and produces a purchase queue the buyer can act on each morning, with the arithmetic behind every number visible.

3. Data and method

Pipeline

Table 1. Pipeline stages

Stage	What happens	Output
Sync	Python pulls daily orders from the Shopify Admin API and the Square API, normalizes both to one schema of SKU, date, units, revenue, and source, and keeps 365 days of history.	Unified sales history
Schema	Products, vendors, settings, purchase log, and forecast log live in a normalized PostgreSQL schema with foreign keys and indexes. Every SKU must carry a vendor, lead time, MOQ, and unit cost.	PostgreSQL tables
Velocity	A 90-day rolling average of daily sales per SKU with weights ramping linearly from 0.5 on the oldest day to 1.5 on the most recent, plus an optional per-SKU growth factor. Cold-start and dead SKUs are explicit cases.	Units per day per SKU
Engine	Days of supply equals on-hand divided by velocity, compared against target cover plus vendor lead time to assign one of five states. Outstanding purchase orders override to on order. Recommended quantity covers the target window and rounds up to the vendor MOQ.	State and quantity per SKU
App	Six Streamlit pages: Overview, Reorder Alerts, All Products, Forecast Detail, Purchase Log, Vendors. Filters carry across pages. Each refresh writes a forecast log row for audit.	Operator application

Reorder states

Table 2. States assigned by the engine

State	Meaning
reorder_now	Days of supply will not cover lead time plus target cover; order this cycle.
reorder_soon	Cover is adequate now but will not be by the next review; plan the order.
healthy	Cover is sufficient; no action.
slow	Low velocity; review before committing to a vendor minimum.
dead	No recent sales; do not reorder.
on_order	Override: an outstanding purchase order already covers the gap.
manual_override	Override: the buyer has pinned a state by hand.
insufficient_history	Override: too few days of sales to estimate velocity.

Testing

Twenty-five pytest tests cover the velocity model, the reorder state logic, and invariants of the demonstration dataset.

4. Results

The Overview page (Figure 1) gives the context around every recommendation: revenue, units, average daily revenue, SKUs in scope, and the reorder count, with the sales trend and category mix.

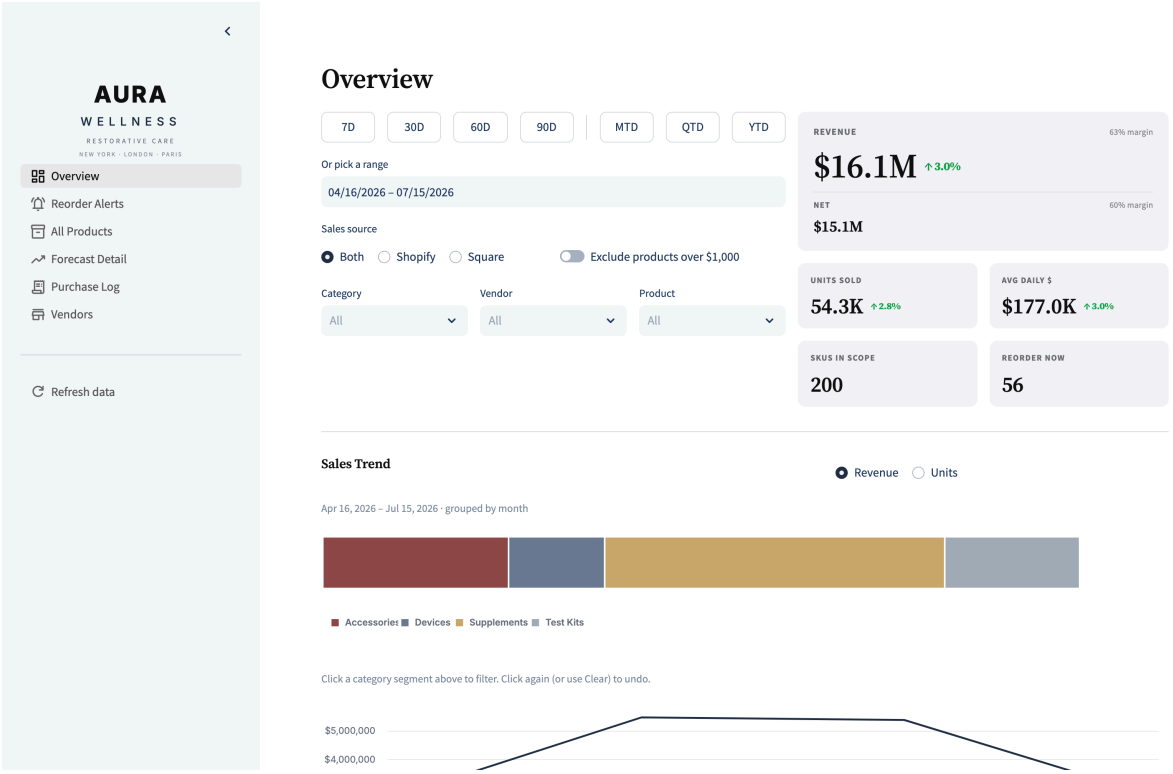


Figure 1. Overview: KPIs, sales trend, and category mix

Source: Shopify Admin API and Square API order history, synthetic for this demonstration.

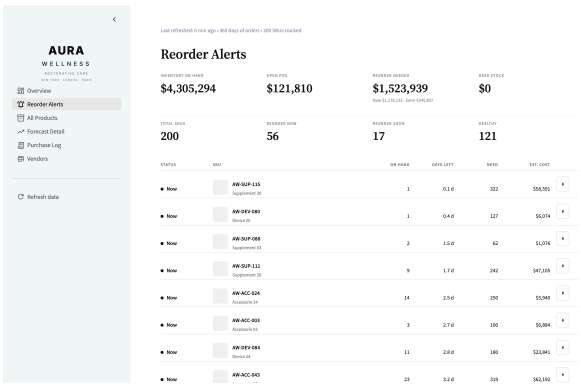


Figure 2. Reorder alerts, filtered to action rows

Source: Demonstration dataset.

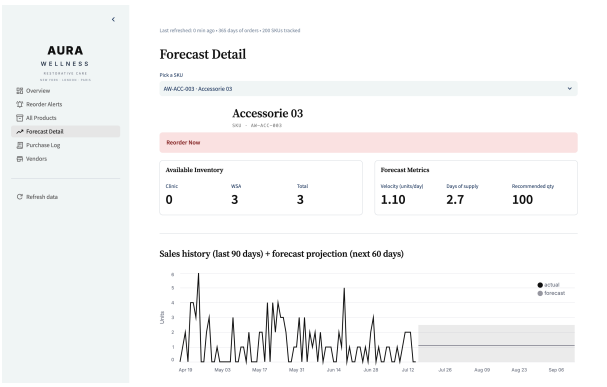


Figure 3. Forecast detail with the calculation shown

Source: Demonstration dataset.

The alerts view (Figure 2) is the morning queue: each SKU with its state, days of supply, recommended quantity, and purchase order value, grouped by vendor. The forecast detail view (Figure 3) opens one SKU to its sales history, projection, and the velocity and quantity arithmetic, so a buyer can check any recommendation before placing an order.

Table 3. Headline results, this cycle

Measure	Value
SKUs flagged reorder_now	56 of 200
Recommended purchase order value, MOQ applied	$1.18M
Velocity spread, top versus bottom quartile	9.0x
Days of rolling history	365

5. Limitations

- **Demonstration data.** The figures come from a seeded synthetic brand. The method is unchanged on live stores, but the numbers are illustrative.
- **Univariate velocity.** Demand is estimated from each SKU's own recent sales plus an optional growth factor. Promotions, stockouts, and seasonality are not modeled explicitly.
- **Static vendor terms.** Lead time and MOQ are fixed per SKU. Vendor performance against quoted lead time is logged but not yet fed back into the model.
- **No service-level target.** Target cover is a setting rather than derived from a fill-rate goal and demand variability.

6. Recommendations

1. **Run the queue daily.** Review the reorder-now rows each morning by vendor and release purchase orders from the app so the purchase log stays complete.
2. **Audit overrides weekly.** Manual overrides and insufficient-history SKUs should be reviewed so they do not become permanent exceptions.
3. **Add seasonality and promotion adjustments** to the velocity model once a full year of reconciled history is available.
4. **Derive target cover from a service level.** Replace the fixed setting with safety stock computed from demand variability and the desired fill rate.

Appendix A. Technical stack and repository

Table 4. Stack

Layer	Tool
App	Streamlit, Altair
Data	PostgreSQL (Supabase)
Sources	Shopify Admin API, Square API
Modeling	pandas, numpy
Tests	pytest, 25 tests

Repository layout

```
app.py              Streamlit entry point and page routing
lib/forecast.py     weighted rolling velocity model
lib/reorder.py      status logic and recommended quantity
lib/finance.py      KPI functions over recommendations and purchase log
lib/shopify.py      Shopify Admin API client
lib/square.py       Square Orders API client, same output shape as Shopify
lib/supabase_io.py  Postgres reads and writes
lib/demo_data.py    seeded fictional dataset for demo mode
schema.sql          Postgres schema
tests/              pytest
```

Appendix B. Glossary

Days of supply	On-hand inventory divided by daily velocity: how long current stock lasts.
Lead time	Days from placing a purchase order to receiving it.
MOQ	Minimum order quantity a vendor will accept.
SKU	Stock keeping unit: one sellable product variant.
Target cover	The number of days of demand the brand wants on hand after a delivery lands.
Velocity	Estimated units sold per day, here a recency-weighted 90-day rolling average.

About the author

Leah Hall is a data and analytics professional based in Seal Beach, California. She builds data pipelines, forecasting models, and financial analyses for ecommerce, finance, and hospitality, and publishes the work at leahkatinsights.com.

Project page	leahkatinsights.com/projects.html#project-04
Source code	github.com/leahkatinsights/reorder-forecasting-system
Email	leah@leahkatinsights.com
Website	leahkatinsights.com
LinkedIn	linkedin.com/in/leah-hall
GitHub	github.com/leahkatinsights

Portfolio project. Aura Wellness is a fictional brand and figures shown are from synthetic order data. Code is on GitHub.