# SuperKart Sales Forecasting Model Training and Deployment

## Project 7 | XGBoost Regression with a Flask API, Streamlit UI and Docker

Kimberly Wiethoff, MBA, PMP, PMI-ACP | November 2026

*A machine learning solution that forecasts product-store sales revenue and serves the forecasts through a containerized web API and app for SuperKart's planning teams.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- Data Background and Contents
- EDA Results
- Data Preprocessing
- Model Building and Hyperparameter Tuning
- Model Performance Summary
- Deployment: Architecture, Backend, and Frontend
- Deployment: Single and Batch Inference
- Actionable Insights & Recommendations
- Additional Recommendations & Next Steps

*Appendix: notebook charts for every EDA view, model diagnostics and the deployment files.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Executive Summary

*SuperKart Sales Forecasting: a tuned model, served live from two containers*

- SuperKart must forecast next quarter's revenue for each product in each store to plan inventory and regional strategy. Its four stores differ by up to 2.8× in revenue per product.
- Using 8,763 historical product-store records, Random Forest and XGBoost pipelines were trained and tuned. The tuned XGBoost model explains 93% of the variance in sales on unseen data.
- Forecasts are typically within $112 (about 4%) of actual sales. Tuning cut the train-test RMSE gap from 172 to 70, so the model generalises well.
- The model is served by a Flask API and a Streamlit app, each in its own Docker container in GitHub Codespaces, for single forecasts and batch (CSV) forecasts.

***Bottom line:*** *Store format, city tier and price drive revenue. The deployed forecaster lets planners score whole assortments in seconds and plan stock store by store.*

**0.93**

test $R^2$: share of sales variance explained

**≈ 4%**

average forecast error (MAPE); MAE $112

**2 containers**

Flask API and Streamlit app, live in Codespaces

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Business Problem Overview

*Why SuperKart needs an accurate, scalable quarterly sales forecast*

### Inventory Depends on the Forecast

Procurement buys stock for the coming quarter. Over-forecasting ties up cash and wastes perishables; under-forecasting causes stock-outs.

### Stores Perform Very Differently

Four outlets in Tier 1-3 cities average $1,763 to $4,947 of revenue per product, so a single network-wide average misleads planners.

### Planning Is Manual and Slow

Without a model, regional teams estimate revenue product by product from history. This is slow, inconsistent and hard to benchmark.

### Insights Must Reach Every Store

A one-off analysis is not enough. The forecast must plug into SuperKart's decision systems and be usable across the store network.

**KPIs at risk:** forecast accuracy · inventory holding cost · perishable waste · stock-out rate · regional revenue planning

***Net effect:*** *SuperKart needs a model that predicts product-store revenue accurately, and a deployment that puts that forecast in planners' hands.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Solution Approach

*From historical sales data to a containerized forecasting service*




**Understand Data**

8,763 rows, 12 attributes; quality and store profiles checked

**Explore (EDA)**

Distributions, drivers of sales and store revenue mix

**Prepare**

Fix labels, engineer 3 features, encode, 80/20 split

**Model**

Random Forest and XGBoost pipelines

**Tune**

GridSearchCV, 5-fold CV, best by RMSE

**Deploy**

Flask API and Streamlit app in Docker on Codespaces

### Regression with Ensemble Trees

The target is Product_Store_Sales_Total. RMSE is the primary metric because it is in dollars and penalises large misses; $R^2$, MAE and MAPE are reported alongside it.

### Decoupled Deployment

The backend serves the serialized pipeline over REST. The frontend calls it over a shared Docker network, so other systems can use the same API.

*Steps 1-5 run in the notebook. Step 6 packages the chosen model so any SuperKart team can request forecasts one at a time or in batches.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Data Background and Contents

*Historical product-store sales: 8,763 records, 12 attributes, no missing values*

| Attribute | Type | Description | Range / notes |
|---|---|---|---|
| **Product_Id** | Text | Unique product code (2-letter prefix + number) | 8,763 unique; prefix kept (FD, DR, NC) |
| **Product_Weight** | Numeric | Weight of the product | 4 to 22, mean 12.65 |
| **Product_Sugar_Content** | Category | Low Sugar, Regular, No Sugar | 'reg' label fixed (108 rows) |
| **Product_Allocated_Area** | Numeric | Share of the store's display area | 0.004 to 0.298, right-skewed |
| **Product_Type** | Category | 16 product categories | Grouped: perishable vs non-perishable |
| **Product_MRP** | Numeric | Maximum retail price | 31 to 266, mean 147 |
| **Store_Id** | Category | Store identifier | 4 stores; OUT004 = 53% of rows |
| **Store_Establishment_Year** | Numeric | Year the store opened | 1987 to 2009; converted to store age |
| **Store_Size** | Category | High, Medium, Small | Medium = 69% of rows |
| **Store_Location_City_Type** | Category | Tier 1, Tier 2, Tier 3 | Tier 2 = 71% of rows |
| **Store_Type** | Category | Departmental Store, Supermarket Type1/2, Food Mart | One type per store |
| **Product_Store_Sales_Total** | Target | Revenue from the product in that store | $33 to $8,000, mean $3,464 |

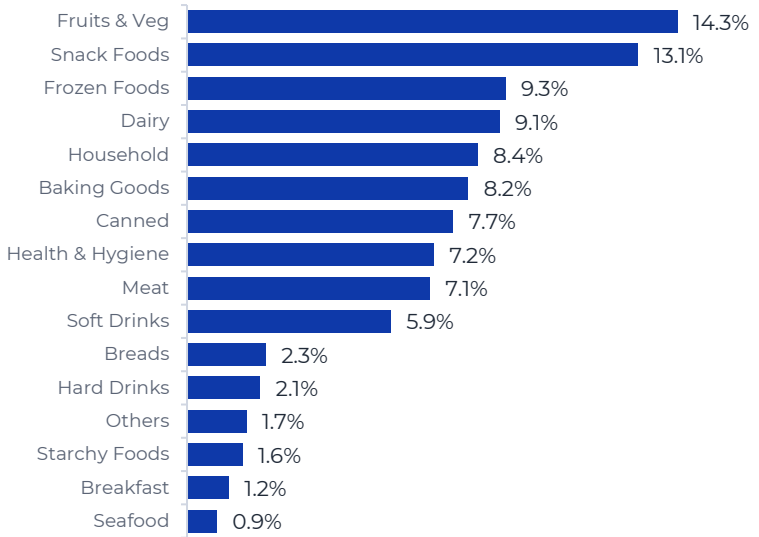
***Data quality is high:*** *0 missing values, 0 duplicate rows and one inconsistent label. Each Store_Id maps to one fixed profile (size, tier, type, year).*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# EDA Results: Distributions

*Product attributes are well-behaved; store variables are concentrated in one outlet*

## Share of products by type (%)




### Sales (target)

Mean $3,464, range $33 to $8,000. Near-normal (skew 0.09); 1.4% outliers.

### Product MRP

Mean 147, range 31 to 266. Near-normal, with prices spread widely.

### Product Weight

Mean 12.65, range 4 to 22. Tightly centred; 0.6% outliers.

### Allocated Area

Median 0.056 and right-skewed (skew 1.1): most products get little shelf space.

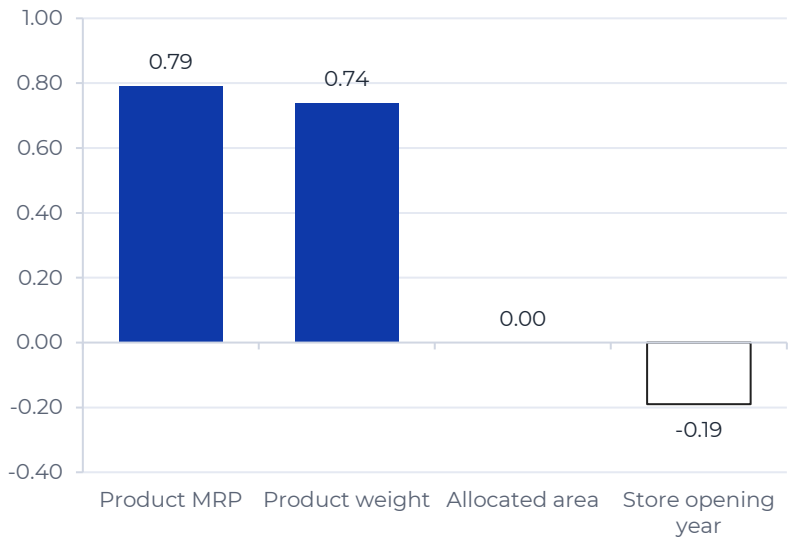
*Fruits & Vegetables and Snack Foods are 27% of products; Seafood and Breakfast are each under 1.5%. 53% of records come from one store (OUT004).*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

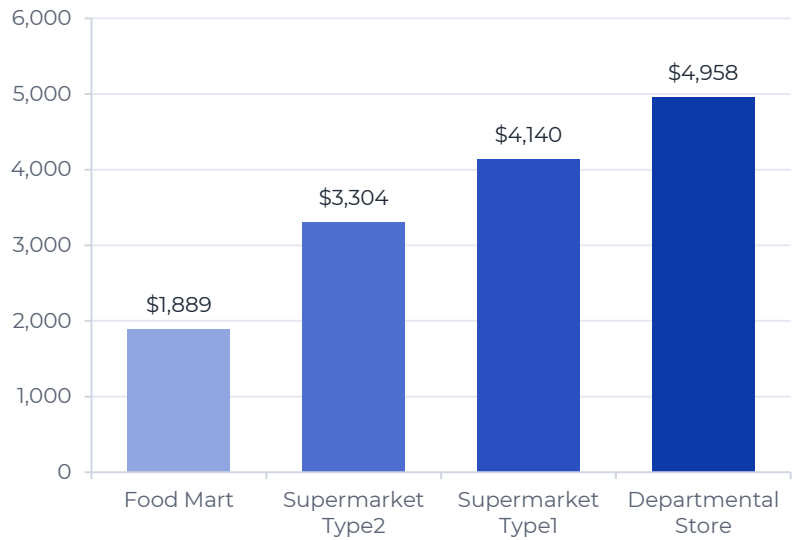
# EDA Results: What Drives Sales

*Price and store format drive revenue; display area and sugar content barely matter*

### Correlation with sales (r)




### Median sales per product by store type ($)




*City tier mirrors store type (median: Tier 1 $4,958, Tier 2 $3,427, Tier 3 $1,889). Product-type medians differ by about 10% and sugar-content medians by under 3%.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# EDA Results: Revenue by Store

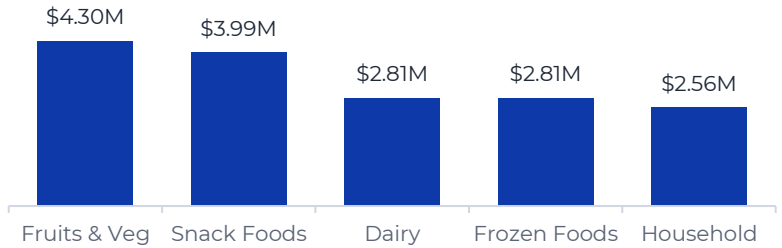
*OUT004 earns half the revenue through volume; OUT003 earns the most per product*

| Store | Type | Size | Tier | Opened | Products | Revenue | Share | Avg / product | Median MRP |
|---|---|---|---|---|---|---|---|---|---|
| **OUT004** | Supermarket Type2 | Medium | Tier 2 | 2009 | 4,676 | $15.43M | 50.8% | $3,299 | 142.8 |
| **OUT003** | Departmental Store | Medium | Tier 1 | 1999 | 1,349 | $6.67M | 22.0% | $4,947 | 179.7 |
| **OUT001** | Supermarket Type1 | High | Tier 2 | 1987 | 1,586 | $6.22M | 20.5% | $3,924 | 168.3 |
| **OUT002** | Food Mart | Small | Tier 3 | 1998 | 1,152 | $2.03M | 6.7% | $1,763 | 104.7 |

### Share of total revenue by store




### Top 5 product types by revenue ($M)




*Total historical revenue is $30.4M. Fruits & Vegetables and Snack Foods lead in every store (27% of revenue combined).*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Data Preprocessing

*Cleaning, feature engineering and preparation produce exactly the inputs the deployed API expects*

### Clean

'reg' relabelled as 'Regular' (108 rows). No missing values or duplicate rows to treat.

### Outlier Check

The IQR rule flags 0.6-1.4% of values per column. These are kept: they are realistic, and tree models are robust to them.

### Prepare for Modeling

Drop 4 ID/redundant columns, split 80/20 (7,010 train, 1,753 test) and one-hot encode 6 categoricals inside the pipeline.

| Engineered feature | Derived from | Logic | Why |
|---|---|---|---|
| **Product_Id_char** | Product_Id | First 2 letters: FD food, DR drinks, NC non-consumables | IDs are unique; the prefix generalises |
| **Store_Age_Years** | Store_Establishment_Year | 2025 minus opening year (16 to 38 years) | Interpretable store maturity |
| **Product_Type_Category** | Product_Type | Perishables (Dairy, Meat, Fruits & Veg, Breakfast, Breads, Seafood) vs Non Perishables | Reduces 16 sparse types to 2 (35% perishable) |

***Model input:*** *10 features (4 numerical, 6 categorical). The API and Batch_Data_SuperKart.csv use the same columns, so training and production inputs match.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Model Building

*Two ensemble regressors, each wrapped in a pipeline with the preprocessor*

### Random Forest

100 fully grown trees on bootstrap samples; averaging reduces variance.

### XGBoost

Gradient-boosted trees built in sequence, each correcting the last one's errors.

### Evaluation

RMSE (primary), MAE, $R^2$, adjusted $R^2$ and MAPE on training and test data.

| Default models | RF train | RF test | XGB train | XGB test |
|---|---|---|---|---|
| **RMSE** | 107.58 | 290.89 | 132.71 | 304.29 |
| **MAE** | 40.16 | 110.00 | 60.62 | 131.86 |
| **$R^2$** | 0.990 | 0.927 | 0.984 | 0.920 |
| **Adj. $R^2$** | 0.990 | 0.926 | 0.984 | 0.919 |
| **MAPE** | 1.5% | 3.8% | 2.2% | 4.6% |

### Both models overfit

Test $R^2$ of at least 0.92 shows that both models generalise, but test RMSE is 2-3× training RMSE: default trees memorise the training data. Tuning should target this gap.

*The default Random Forest leads on test data (RMSE 290.9 vs 304.3); XGBoost has more room to improve through tuning.*

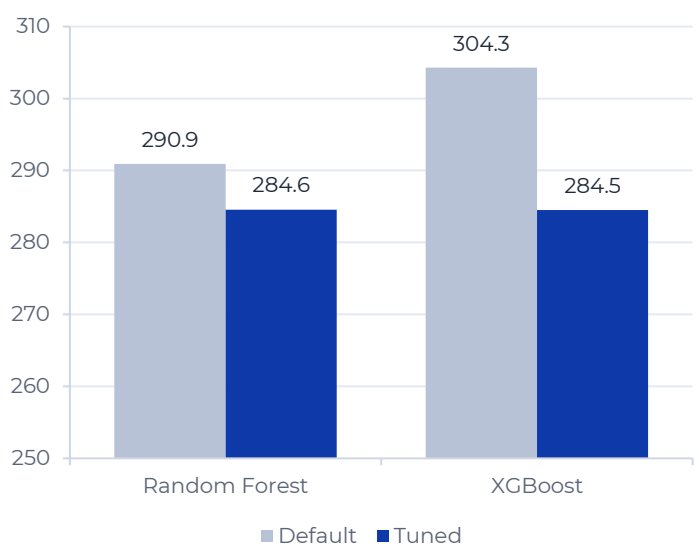
Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Model Improvement: Hyperparameter Tuning

*GridSearchCV (5-fold, negative RMSE) improved both models and narrowed the overfitting gap*

| Model | Parameters searched | Best parameters |
|---|---|---|
| **Random Forest (36 combinations)** | n_estimators [100, 200] · max_depth [10, 20, None] · min_samples_leaf [1, 3] · max_features [sqrt, 0.5, 1.0] | 200 trees · depth 10 · leaf 1 · all features |
| **XGBoost (72 combinations)** | n_estimators [100, 200, 300] · learning_rate [0.05, 0.1] · max_depth [3, 5, 7] · subsample [0.8, 1.0] · colsample_bytree [0.8, 1.0] | 100 trees · learning rate 0.05 · depth 7 · subsample 1.0 · colsample 1.0 |

**Cross-validated RMSE:** 281.4 (Random Forest) and 281.6 (XGBoost), close to the test RMSE of about 284.5. The tuned scores are stable across folds, not the result of a lucky split.




*Tuning improved XGBoost the most (test RMSE −19.8) and cut its train-test gap from 172 to 70, by using shallower, slower-learning trees.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Model Performance Summary

*Final model: tuned XGBoost, with the best test RMSE and the smallest overfitting gap*

| Model | Train RMSE | Test RMSE | Gap | Test $R^2$ | Test MAE | Test MAPE |
|---|---|---|---|---|---|---|
| **Random Forest** | 107.58 | 290.89 | 183.3 | 0.927 | 110.0 | 3.8% |
| **Random Forest (tuned)** | 191.85 | 284.57 | 92.7 | 0.930 | 112.9 | 4.0% |
| **XGBoost** | 132.71 | 304.29 | 171.6 | 0.920 | 131.9 | 4.6% |
| **XGBoost (tuned) ★** | **214.67** | **284.52** | **69.9** | **0.930** | **112.2** | **4.0%** |

## Final model and parameters

- XGBRegressor(n_estimators=100, learning_rate=0.05, max_depth=7, subsample=1.0, colsample_bytree=1.0, random_state=1)
- Pipeline: OneHotEncoder(handle_unknown='ignore') on 6 categoricals; 4 numerics passed through
- Serialized as superkart_model.joblib (674 KB); predictions from the reloaded file match the trained model

**$R^2$**

**0.930**

test (train 0.959)

**RMSE**

**$284.5**

test (train $214.7)

**MAE**

**$112.2**

test (train $83.0)

**MAPE**

**3.98%**

test (train 3.06%)

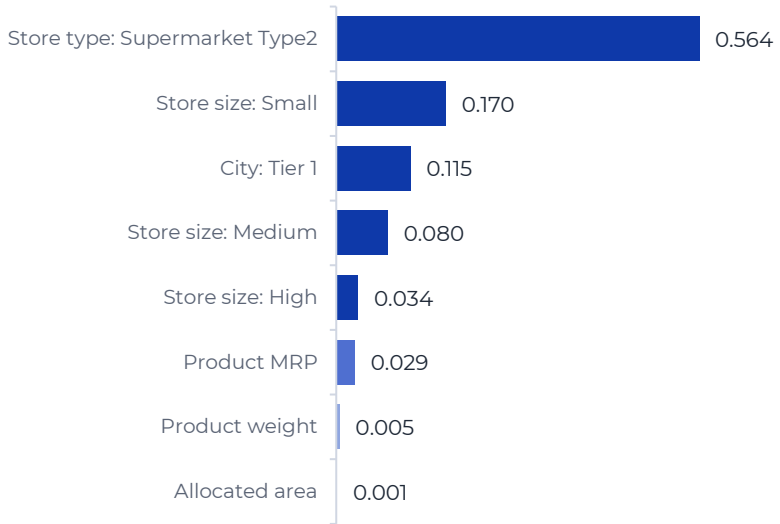
***Why XGBoost:*** *the tuned Random Forest is as accurate, but tuned XGBoost generalises more consistently and its model file is about 22× smaller (0.7 MB vs 14.9 MB), keeping the container light.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

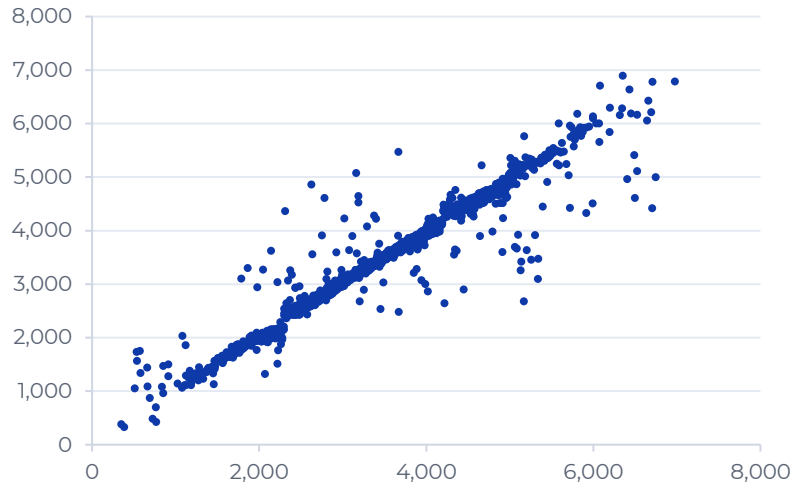
# Final Model: What It Learned

*Store context dominates the model; price is the strongest product-level signal*

## Top feature importances (gain)




## Actual vs predicted sales, test set ($)




*Points cluster tightly along the diagonal; the largest misses are rare extremes.*

*Importance is gain-based: store one-hot features absorb the large between-store differences, while MRP and weight explain variation within a store.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Deployment Architecture

*Two Docker containers in GitHub Codespaces, connected by a shared Docker network*

### Google Colab

Train, tune, serialize model; write app files

*git push*

### GitHub repository

superkart-sales-forecasting: backend, frontend, run script

**GitHub Codespace (Docker)**

### Streamlit frontend

Container superkart-frontend · port 8501 · online form and batch CSV upload

REST

### Flask backend

Container superkart-backend · Gunicorn · port 7860 · superkart_model.joblib

**Docker network superkart-net:** the frontend reaches http://superkart-backend:7860 by container name

**Planners** open the public 8501 URL in a browser

**Systems and notebooks** call the public 7860 URL directly (/v1/predict, /v1/predictbatch)

*Decoupling lets other SuperKart systems call the API directly while planners use the app. run_containers.sh rebuilds and restarts both containers with one command.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Deployment: Backend

*A Flask REST API served by Gunicorn in a slim Python container on port 7860*

| Route | Method | Input | Output |
|---|---|---|---|
| **/** | GET | None | Health-check message |
| **/v1/predict** | POST | JSON with the 10 model features | {"Predicted Sales (in dollars)": value} |
| **/v1/predictbatch** | POST | CSV upload (form field 'file') | {"row index": forecast, ...} |

### Input Validation

Missing fields or non-numeric values return HTTP 400 with a clear message instead of an error page.

### Model Loaded Once

The pipeline loads at start-up and handles encoding, so raw values are accepted.

### Pinned Dependencies

numpy, pandas, scikit-learn, xgboost and joblib are pinned to the training versions; Gunicorn runs 4 workers.

### Tested Before Push

Local test: $3,709.79 predicted vs $3,714.07 actual. A missing feature returned 400; the batch call returned 200.

***Lesson learned:*** *the container must use the training library versions. An older XGBoost dropped the model's base score and shifted every forecast by −$3,454 until requirements.txt matched.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Deployment: Frontend

*A Streamlit web app gives planners single forecasts and batch CSV scoring*

### Online Prediction Tab

A form for one product-store combination: 10 inputs (dropdowns and number fields) and a Predict button that shows the forecast.

### Batch Prediction Tab

Upload a CSV, preview it, forecast every row in one API call, see total forecast revenue and download the results.

### Connects Through the API

BACKEND_URL (default http://superkart-backend:7860) points to the Flask container; connection and API errors are shown in the app.

### Own Container

A slim Python image runs Streamlit on port 8501. CORS/XSRF checks are relaxed so uploads work through the Codespaces proxy.

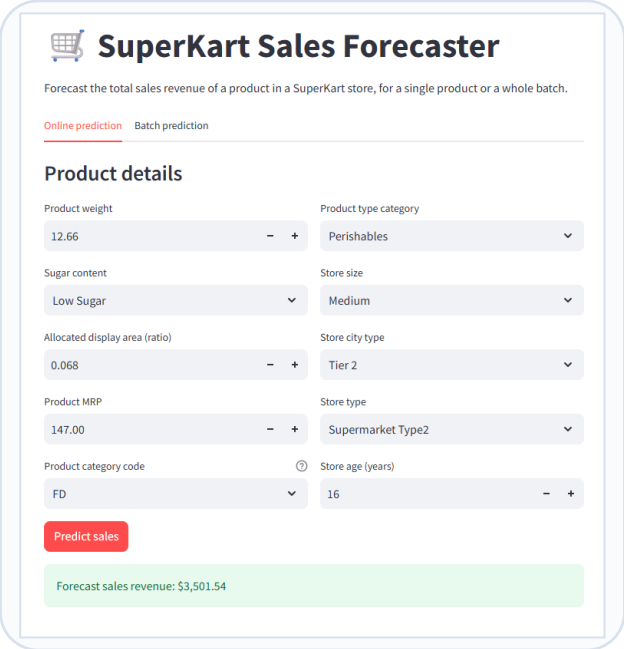
| Input | Widget | Options / range |
|---|---|---|
| **Product weight** | Number | 0 to 50 |
| **Sugar content** | Dropdown | Low, Regular, No Sugar |
| **Allocated area** | Number | 0 to 1 (ratio) |
| **Product MRP** | Number | 0 to 500 |
| **Category code** | Dropdown | FD, DR, NC |
| **Type category** | Dropdown | Perishables, Non Perishables |
| **Store size** | Dropdown | Small, Medium, High |
| **City type** | Dropdown | Tier 1, 2, 3 |
| **Store type** | Dropdown | 4 store formats |
| **Store age** | Number | 0 to 100 years |

*The app needs no Python knowledge: planners pick values or upload a file, and the backend does all preprocessing and prediction.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Deployment: Single Inference

*One product-store record in, one revenue forecast out, in under a second*




```
POST /v1/predict
{
 "Product_Weight": 12.66,
 "Product_Sugar_Content":
    "Low Sugar",
 "Product_Allocated_Area": 0.068,
 "Product_MRP": 147.00,
 "Store_Size": "Medium",
 "Store_Location_City_Type": "Tier 2",
 "Store_Type": "Supermarket Type2",
 "Product_Id_char": "FD",
 "Store_Age_Years": 16,
 "Product_Type_Category":
    "Perishables"
}
```

**$3,501.54**

Streamlit forecast · HTTP 200

| Input | Notebook sample | Streamlit run |
|---|---|---|
| **Product MRP** | 117.08 | 147.00 |
| **Allocated area** | 0.027 | 0.068 |
| **Type category** | Non Perishables | Perishables |
| **Other 7 inputs** | Same | Same |
| **Forecast** | **$2,876.61** | **$3,501.54** |

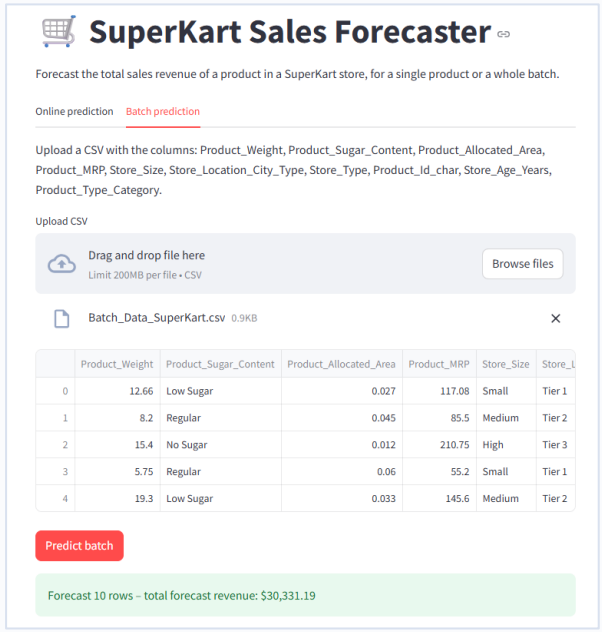
**What changed the forecast?** Raising only MRP from 117.08 to 147 adds $634. Area and category each move it by less than $10.

***Same API, different inputs:*** *the higher MRP explains almost all of the $625 difference between the two forecasts, consistent with price being the strongest product-level driver in the EDA.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Deployment: Batch Inference

*A 10-row CSV is forecast in a single API call; the app shows the total and offers a download*




| # | Store type | Size | Tier | MRP | Forecast ($) |
|---|---|---|---|---|---|
| **0** | Supermarket Type1 | Small | Tier 1 | 117.08 | 1,103.28 |
| **1** | Supermarket Type2 | Medium | Tier 2 | 85.50 | 3,782.51 |
| **2** | Departmental Store | High | Tier 3 | 210.75 | 4,402.94 |
| **3** | Food Mart | Small | Tier 1 | 55.20 | 328.81 |
| **4** | Supermarket Type3 * | Medium | Tier 2 | 145.60 | 4,203.87 |
| **5** | Supermarket Type1 | High | Tier 1 | 92.40 | 5,412.44 |
| **6** | Supermarket Type2 | Small | Tier 3 | 178.90 | 1,516.69 |
| **7** | Food Mart | Medium | Tier 2 | 63.25 | 2,613.75 |
| **8** | Departmental Store | High | Tier 1 | 250.00 | 5,641.91 |
| **9** | Supermarket Type3 * | Small | Tier 3 | 130.45 | 1,324.99 |
| | **Total** | | | | **30,331.19** |

**** Supermarket Type3 is not in the training data:*** *the model ignores the unknown type and forecasts from store size and tier, treating rows 4 and 9 like the closest existing store. Options are on the next slide.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Batch Inference: Handling Unseen Store Types

*What the model does with Supermarket Type3 today, and how to get more reliable forecasts*

## What happens today

| Row | Size / tier | As Type3 | As each known type |
|---|---|---|---|
| 4 | Medium / Tier 2 | $4,204 | $4,141 to $4,204 |
| 9 | Small / Tier 3 | $1,325 | $1,325 (all four) |

- The encoder sets every store-type column to 0 for an unknown type, so no error is raised.
- The forecast then comes from store size and city tier, which point to an existing store. Row 9 (Small, Tier 3) is forecast like the Food Mart, OUT002.
- The result is a reasonable estimate, but it quietly assumes the new format sells like that store, and nobody is told.

### 1. Flag it (recommended now)

Still forecast but return a warning and add a Flag column so planners know the row is an estimate.

### 2. Map to a known format

SuperKart decides which existing format Type3 resembles; convert before predicting and label the result as mapped.

### 3. Reject strictly

Return HTTP 400 for unknown categories so no forecast is used without real data. Safest, but less convenient.

### 4. Retrain with Type3 data

After a quarter of Type3 sales, add it to training so the model learns the format's own pattern. The long-term fix.

***Broader finding:*** *with only four stores, store type, size and tier all describe the same four stores, so the model barely needs store type once size and tier are known. New formats need their own sales data.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Actionable Insights & Recommendations

*Benefits: about 4% forecast error · instant single and batch forecasts · one model for every store*

### Plan Quarterly Inventory With the Forecast

Score each store's planned assortment through the batch API every quarter and size procurement to the forecast: less perishable waste, fewer stock-outs.

### Grow Premium Ranges in High-Value Formats

OUT003 (Tier 1 Departmental Store) and OUT001 (High-size Supermarket Type1) earn $4,947 and $3,924 per product. Prioritise higher-MRP items and launches there.

### Lift OUT004's Revenue per Product

It earns 51% of revenue through volume at $3,299 per product. Moving part of its mix to higher-MRP lines targets OUT001-level productivity.

### Rethink the Food Mart (OUT002)

Lowest prices (median MRP 105) and only 7% of revenue. Focus on fast-moving essentials, and test price or promotion changes in the model before rollout.

### Reallocate Display Space and Keep the Model Current

Allocated area shows no link to revenue, so test moving space to high-MRP items. Retrain quarterly, add new formats (e.g. Supermarket Type3) and seasonality, and monitor live forecast error.

***Bottom line:*** *store format and price set revenue. With the forecaster live, SuperKart can plan stock by store, test pricing what-ifs and benchmark outlets every quarter.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Additional Recommendations & Next Steps

*Further opportunities from the data, plus what the model and deployment need next*

## Business

**Grow small, high-earning categories**
Seafood and Starchy Foods are each under 2% of products but have the highest median sales (~$3,680 and $3,655 vs $3,450 overall).

**Benchmark against OUT003**
Track revenue per product for every store; OUT003 earns $4,947 vs a $3,464 average.

**Test Tier 1 expansion**
Tier 1 sells the most per product but has one store. A hypothesis to test: tier, format and price overlap.

**Plan perishables separately**
35% of products spoil. Use store-level forecasts to set order sizes and markdown timing.

## Model and Data

**Collect time-stamped sales**
The data has no dates. Weekly or monthly sales, promotions and stock-outs would enable a true quarterly forecast.

**Give a range, not one number**
Show about ±$285 (test RMSE) with each forecast so planners can set safety stock.

**Hold off on new stores**
With four stores the model learns store identity. New formats such as Supermarket Type3 need their own data first.

## Deployment

**Move to permanent hosting**
Codespaces is for development and sleeps when idle. Host the API on an always-on cloud service.

**Secure and monitor the API**
Add an access key, request logging and a check that flags unknown categories.

**Automate retraining**
Retrain quarterly and redeploy with pinned training versions, so the version mismatch can't recur.

***Priority:*** *start collecting time-stamped sales now. It is the change that most improves the forecast, and every other model recommendation builds on it.*

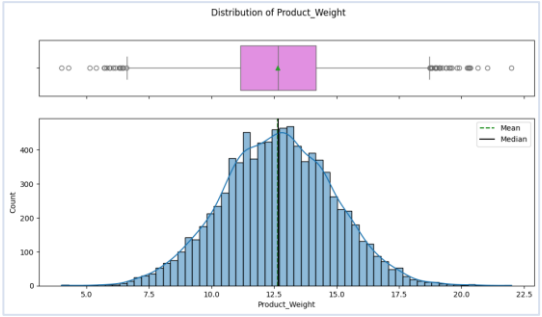
Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# APPENDIX

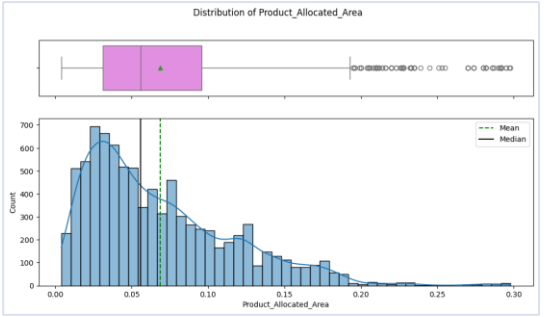
Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Univariate Analysis, Numerical

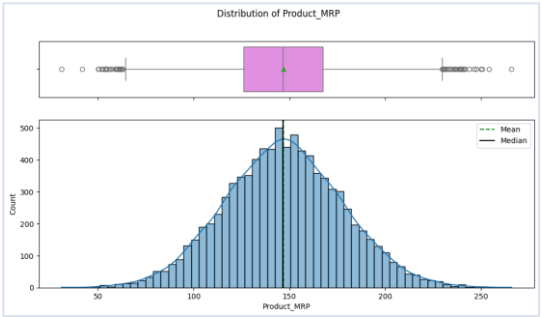
*Notebook output: histogram and boxplot for each numerical variable*



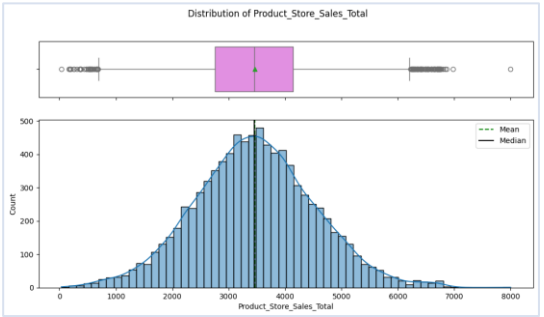

**Observation:** Near-normal around a mean of 12.65 (range 4 to 22). Mean and median almost coincide; only 0.6% outliers on either tail.




**Observation:** Right-skewed (skew 1.1): median 0.056 with a long tail to 0.298. Most products get very little display space.




**Observation:** Near-normal, mean 147 (range 31 to 266). 0.65% outliers at both ends: budget and premium products.



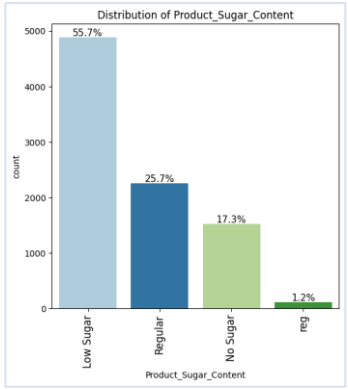

**Observation:** Target is near-normal: mean $3,464, std $1,066. 1.4% outliers: a few very low (under $1,000) and very high (over $6,000) sellers.

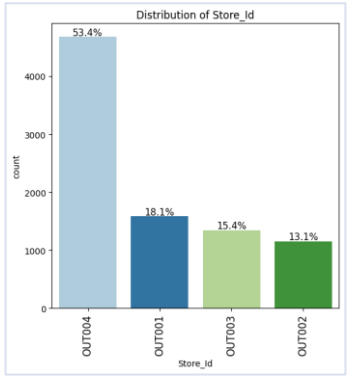
*Product attributes need no transformation for tree-based models; outliers are realistic and kept.*

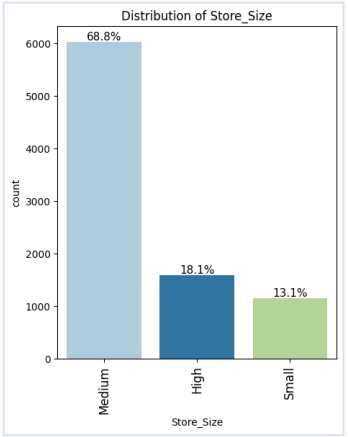
Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

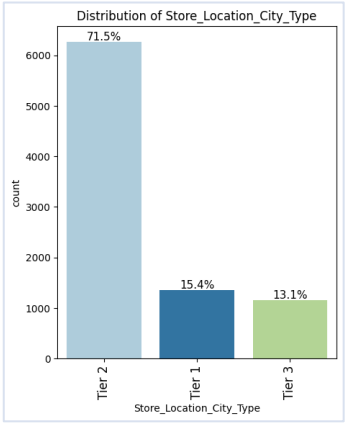
# Appendix: Univariate Analysis, Categorical

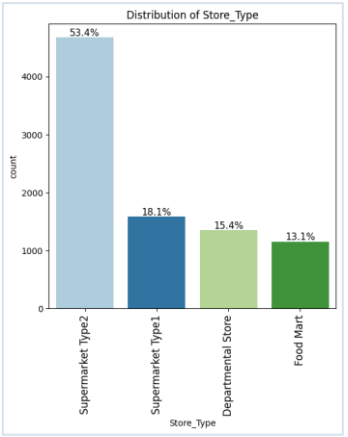
*Notebook output: share of records in each category*




**Observation:** Low Sugar dominates (55.7%). 'reg' (1.2%) duplicates 'Regular' and was merged during preprocessing.

**Observation:** OUT004 holds 53.4% of records; the other three stores have 13-18% each.

**Observation:** Medium is 68.8% (OUT003 + OUT004). High (18.1%) and Small (13.1%) are one store each.

**Observation:** Tier 2 is 71.5% (OUT001 + OUT004). Tier 1 (15.4%) and Tier 3 (13.1%) are one store each.

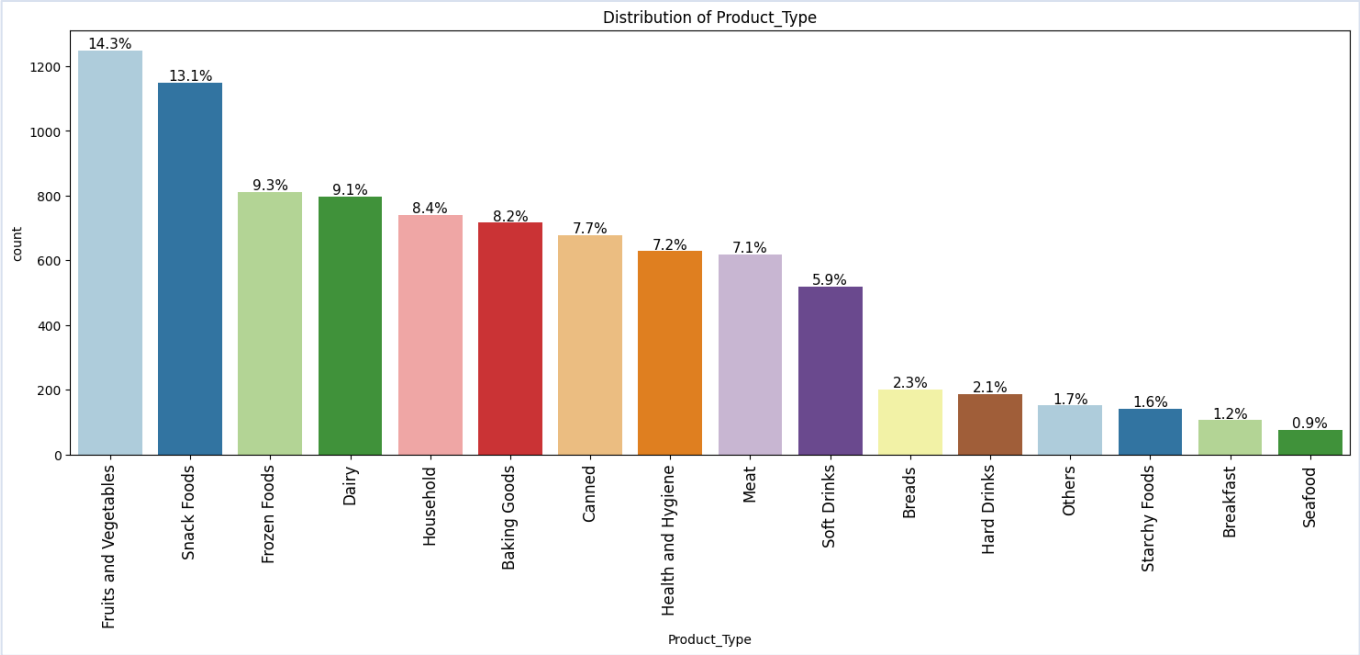
**Observation:** Each store type is exactly one store, so Supermarket Type2 (53.4%) is OUT004 alone.

*Store variables are imbalanced because there are only four stores, and one store holds over half the records.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Product Type Distribution

*Notebook output: share of records by product type*

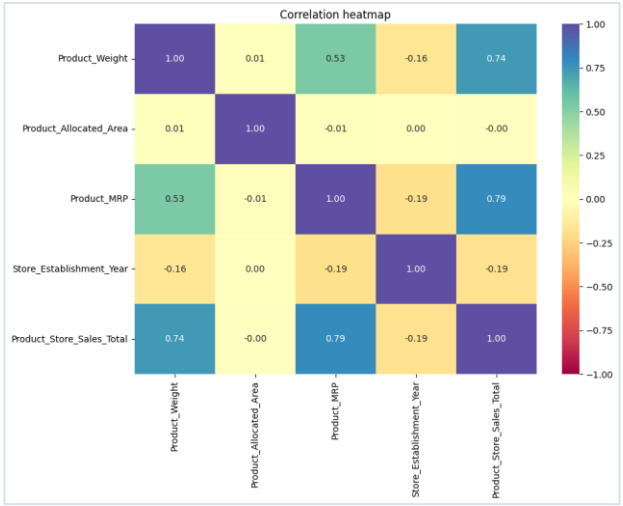


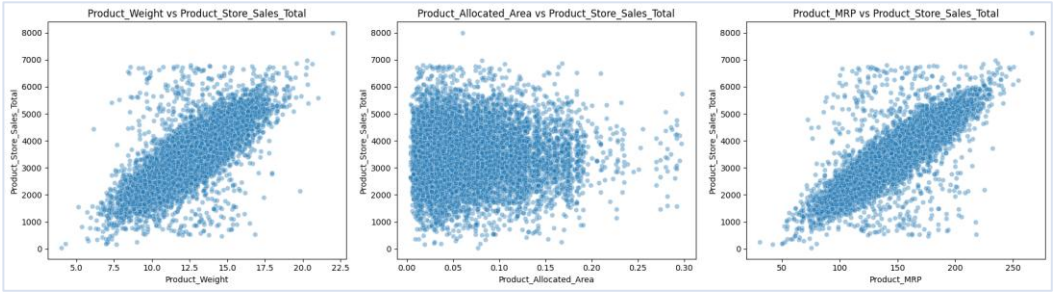

*Fruits & Vegetables (14.3%) and Snack Foods (13.1%) lead; Seafood (0.9%) is the rarest type.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Bivariate Analysis

*Notebook output: correlation heatmap and numerical features vs sales*




**Observation:** Sales correlate strongly with MRP (0.79) and weight (0.74), which are themselves related (0.53). Allocated area is uncorrelated with everything; store year is weakly negative (−0.19).

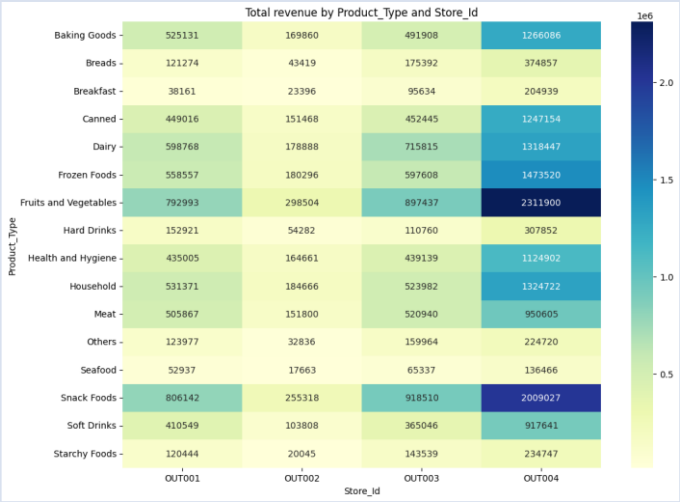
**Observation:** Weight and MRP show clear upward linear trends with sales. Allocated area is a shapeless cloud: more shelf space alone does not lift revenue.

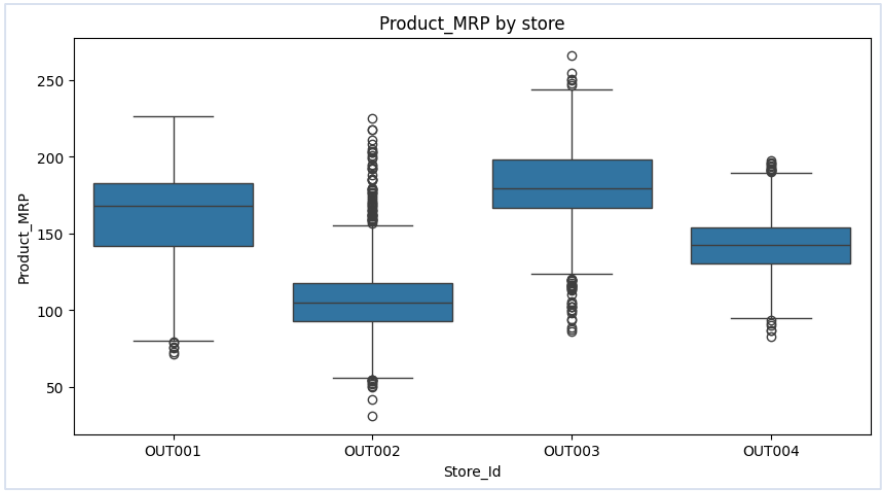
*Price and weight are the strongest numerical signals; display area carries no linear signal.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Sales by Store and Product Type

*Notebook output: total revenue by product type and store, and product prices by store*




**Observation:** OUT004 has the highest revenue in every product type. Fruits & Vegetables and Snack Foods are the top two categories in every store; Seafood and Breakfast are among the smallest.

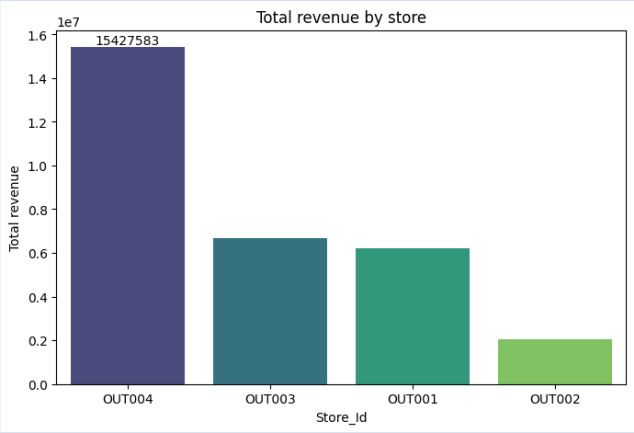
**Observation:** OUT003 stocks the highest-priced products (median MRP 180) and OUT002 the lowest (105), with OUT001 (168) and OUT004 (143) in between. Price levels mirror revenue per product.

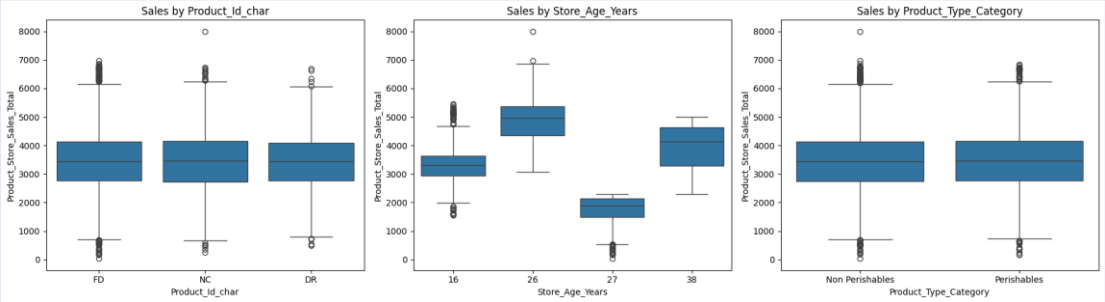
*Store pricing explains much of the gap in revenue per product between stores.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Store Revenue and Engineered Features

*Notebook output: total revenue by store and sales by each engineered feature*




**Observation:** OUT004 earns $15.4M, more than the other three stores combined ($14.9M). The Food Mart (OUT002) earns only $2.0M.

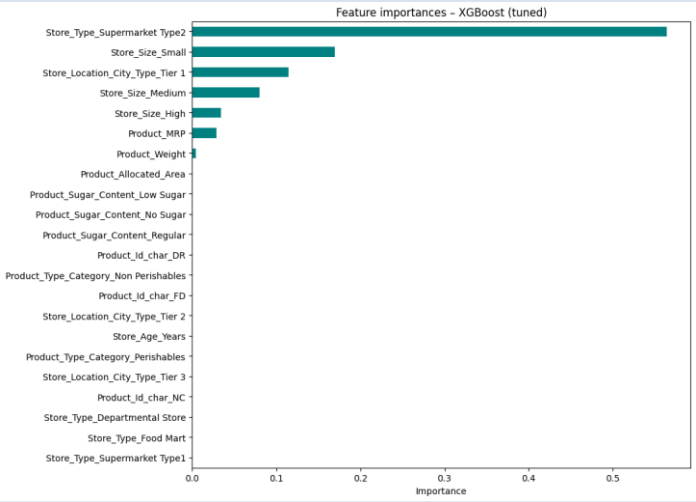
**Observation:** FD, DR and NC have similar sales. Store age separates sales clearly because each age is one store (26 years = OUT003, highest). Perishables and non-perishables look alike.

*Engineered features keep the useful store signal while replacing unique IDs with generalisable groups.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Final Model Diagnostics

*Notebook output: tuned XGBoost feature importances and actual vs predicted sales on the test set*







**Observation:** Store type (Supermarket Type2, 0.56), Small size (0.17) and Tier 1 (0.12) lead. MRP (0.03) is the top product feature; sugar content, prefix and category add almost nothing.

**Observation:** Points hug the perfect-prediction line (test $R^2$ 0.93, MAE $112). Only 82 of 1,753 test products (4.7%) miss by more than $500, split evenly between over- and under-prediction.

*The model relies on store context first and price second, and its errors show no systematic bias.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Deployment Files

*Repository: github.com/kimberlywiethoff/superkart-sales-forecasting*

```
# backend_files/requirements.txt
numpy==2.4.6
pandas==2.2.3
scikit-learn==1.6.1
xgboost==3.4.1
joblib==1.6.0
flask
gunicorn==23.0.0
```

```
# frontend_files/requirements.txt
streamlit>=1.40
requests>=2.32
pandas

# Repository layout
backend_files/  (app.py, model, Dockerfile)
frontend_files/ (app.py, Dockerfile)
run_containers.sh, README.md
```

```
# backend_files/Dockerfile
FROM python:3.13-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir \
    -r requirements.txt
COPY . .
EXPOSE 7860
CMD ["gunicorn", "-w", "4",
     "-b", "0.0.0.0:7860",
     "app:superkart_api"]
```

```
# run_containers.sh (excerpt)
docker network create superkart-net

docker build -t superkart-backend \
  ./backend_files
docker run -d --name superkart-backend \
  --network superkart-net \
  -p 7860:7860 superkart-backend

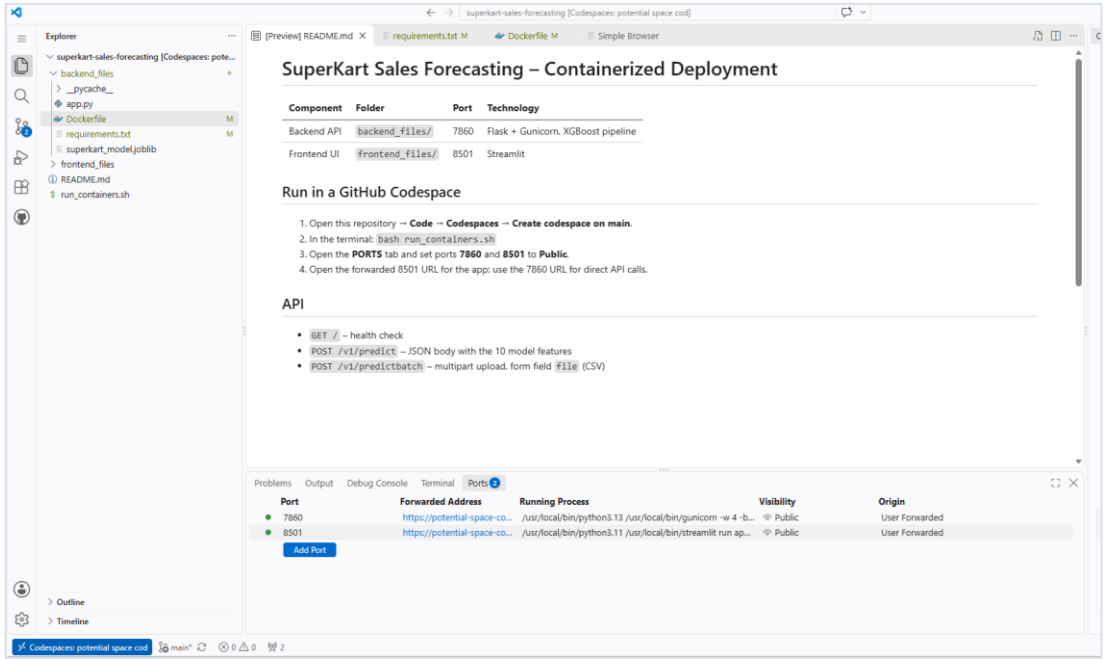
docker build -t superkart-frontend \
  ./frontend_files
docker run -d --name superkart-frontend \
  --network superkart-net -p 8501:8501 \
  -e BACKEND_URL=http://superkart-
backend:7860 \
  superkart-frontend
```

*Backend versions match the Colab training environment exactly (see the Deployment: Backend slide).*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

# Appendix: Running Deployment in Codespaces

*Repository open in the Codespace, with both containers running and ports 7860 and 8501 public*




*Port 7860: Gunicorn serving the Flask API · Port 8501: Streamlit app · both Public, so the notebook and browser can reach them.*

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

Great Learning

# Power Ahead!

