ChatGPT

Developing a Spoilage-Risk Prediction Model for Indian Produce

1. Introduction

Losses of fruits, vegetables and grains in India are substantial. Improper post-harvest handling can cause **10–40% losses** in horticultural crops [1]. Globally, about **25–30%** of harvested fruits and vegetables are lost before consumption [2]. Accurate spoilage prediction can reduce these losses (AI systems may cut post-harvest losses by 20–40% for perishables [3]). We focus on fresh primary agricultural produce (grains, vegetables, fruits) – not processed or cooked foods – since spoilage dynamics differ. The goal is a Python/TensorFlow (or similar) model that ingests multi-source government data and outputs, **for each commodity batch**: - *Estimated remaining shelf life* (days) – a regression output,
- *Spoilage risk level* (low/medium/high) – a classification,
- *Predicted spoilage percentage* (e.g. 70%) – another regression or probabilistic output.

We will filter out any item whose predicted shelf life is below a set threshold (to avoid listing nearly-expired goods). In this report we detail each step of the development pipeline: data sources, cleaning, feature engineering, proxy-labeling, model selection/training, evaluation metrics, and deployment.

2. Data Sources

We will integrate multiple open datasets (especially government/official data):

- **Variety-wise Daily Market Prices (AGMARKNET):** The Government's portal provides daily mandi prices for each commodity variety (wholesale max, min, modal prices) [4]. We can use the *Agmarknet* "Current Daily Price" dataset from data.gov.in, updated May 2025, which lists prices by market and variety [4].
- **Daily FCI (Food Corporation of India) Stock Positions:** Daily inventory levels of staple grains (wheat, rice, etc.) by region. For example, an FCI dataset contains records like *date, region, commodity (e.g. wheat), district, and stock levels* [5]. Such data (via data.gov.in APIs) shows how much of each commodity is in storage at each district, which affects supply/demand and storage usage.
- **Post-Harvest Loss Data:** Government/industry reports (e.g. MoFPI/NABARD studies) give estimated percentage losses by commodity category. For instance, India's NABARD has crop-wise loss estimates (often 10–30% of production) during storage and transport. We will use this to **proxy** spoilage (see Section 6).
- **Daily Soil Moisture:** District-level volumetric soil moisture (from NRSC/India's VIC model) provides moisture content of soil. High soil moisture can indicate humid conditions that promote rot. (India's OGD portal and AI platforms host "Daily Soil Moisture" data from 2018 onward).
- **Daily Rainfall (NRSC/VIC model):** District/sub-basin rainfall (mm) from the NRSC VIC hydrology model. Rainfall and humidity are key environmental factors: excessive rain can increase humidity in storage, encouraging spoilage.

These datasets will be downloaded (via APIs or bulk CSVs from data.gov.in or related portals) and merged by date, location, and commodity type. Table 1 lists key features from each source:

Data Source	Key Fields	Usage
Agmarknet Daily Prices [4]	Date, Market, Commodity/ Variety, Max/Min/Modal Price	Indicates demand/supply balance and price trends. Price drops or volatility may signal oversupply/spoilage.
FCI Daily Stocks [5]	Date, Region/District, Commodity, Stock Level	Shows inventory; high stocks with falling prices could imply excess and potential waste.
Post-Harvest Loss Data	Commodity, Loss Category %, Seasonal factors	Historical % losses by commodity category; can be used to calibrate spoilage risk labels.
Soil Moisture (daily)	Date, District, Volumetric Soil Moisture %	High soil moisture can elevate humidity/risk of spoilage in stored produce.
Rainfall (daily)	Date, District, Rainfall (mm)	High rainfall can raise ambient humidity/ temperature, affecting spoilage rate.

Table 1: Data sources and their roles. These will be filtered to relevant crops (grains, fruits, vegetables) and aligned by time and location.

3. Data Collection and Cleaning

Step 1 – Collect data: Use Python (e.g. Pandas) to fetch or load each dataset. For example, one can use the DataGovIndia Python client or requests to retrieve the Agmarknet price data and FCI stock data via OGD APIs [4] [5]. Similarly, download soil moisture and rainfall from government sources (e.g. NRSC/ ISRO portals). Use commodity codes/names to filter only primary agricultural goods (no processed/ cooked foods).

Step 2 – Unify formats: Ensure all datasets use a common calendar format. Convert all dates to `YYYY-MM-DD`. If data are at different granularities (e.g. some daily, some weekly), resample or interpolate as needed.

Step 3 – Merge datasets: Join on keys like (Date, Commodity, State/District). Market prices may be at the mandi level – aggregate by state or crop variety. Align FCI stocks by region. The result should be a consolidated table of feature values for each (date, commodity, location) record.

Step 4 – Clean and impute: Handle missing values. For any missing price or stock entries, consider forward-fill/backfill or interpolation. Outliers (e.g. extreme one-day jumps) should be investigated – if erroneous, remove or smooth them. Standardize categorical fields (commodity names, district names) to consistent codes.

Step 5 – Filter irrelevant records: Remove any entries below a minimum shelf-life threshold upfront. For example, if we know items with shelf life < X days should not be listed, those can be flagged now. (Actual filtering will be done post-prediction, but invalid input records can be dropped early if needed.)

Throughout, document data quality issues. Use logging to note any large gaps or anomalies. This cleaned dataset will serve as input for feature engineering.

4. Feature Engineering

From the cleaned data, we derive predictive features. Key engineered features include:

- **Price Features:** For each commodity-location-day, compute rolling statistics of market price: e.g. 7-day mean price, price volatility (standard deviation), and recent price change (percent drop from last week). Volatile or falling prices may indicate oversupply or anticipated spoilage.
- **Inventory Features:** Use FCI stock data to gauge storage loads. For instance, create `Stock_Change = (today's stock – yesterday's stock) / yesterday's stock`. Sudden increases in stock (harvest arrival) or very high total stock might predict supply gluts and spoilage.
- **Weather/Humidity:** Include the day's rainfall and soil moisture, plus lagged sums (e.g. total rain in last 3 or 7 days). High sustained rainfall/humidity correlates with faster spoilage [1] [6].
- **Storage Proxy:** If available, features like storage capacity usage (e.g. percent full warehouses from FCI data) or ambient temperature can be included. (If temperature is missing, soil moisture + rainfall give a humidity proxy.)
- **Seasonality:** Add temporal features such as month or week-of-year to capture seasonal harvesting and consumption patterns.
- **Commodity-specific constants:** Each commodity/variety has inherent shelf-life norms. If known, include this as a baseline shelf-life (from literature or extension data).

All features should be normalized or standardized (especially numeric ones) before feeding into models. Also consider dimensionality reduction if there are very many lagged variables. For tree-based models (RF/XGBoost), raw values can be used; for neural networks, normalize inputs to e.g. zero mean unit variance.

5. Proxy Label Estimation (No Direct Spoilage Data)

There is **no direct historical label** for "actual spoilage %" in these datasets. We must therefore create proxy labels for training:

- **Post-Harvest Loss Surveys:** Government studies (e.g. NABARD) report typical loss percentages for each crop. Use these as target spoilage percentages for training, possibly adjusting by season/location. For example, if maize typically incurs 15% loss post-harvest, use ~15% as the spoilage label for maize entries, modifying it upward if inventory is high and weather is adverse. As a rule of thumb, agricultural losses can reach 10–40% [1] [2], so labels in that range are plausible.
- **Price-Derived Proxy:** Sudden commodity **price drops** often indicate oversupply (sometimes due to unsold/spoiled produce flooding the market). We can define events (e.g. a 10% price drop week-over-week) as "high spoilage risk." Quantitatively, we could set the label *Spoilage%* = f(max(0, -%price_change)) where price change is negative. This creates a continuous proxy for loss.
- **Weather-Derived Proxy:** If recent rainfall or humidity exceeds a threshold (e.g. 100 mm in 3 days), tag the risk as higher. For instance, add 5–10% to the spoilage label for extreme weather conditions.
- **Clustering:** Use unsupervised learning on the feature space (e.g. K-means) to cluster historical days into *low/med/high spoilage regimes* based on patterns of price plunge, stock surge, and moisture. Then assign class labels from the clusters.

By combining these heuristics, we can construct target variables: a continuous *Remaining Shelf Life* (e.g. if typical shelf life is 20 days and conditions are poor, label 10 days) and *Spoilage%*. We then derive the

Risk Level as a categorical label (e.g. if shelf life <X or spoilage>% Y, mark as “High Risk”; else medium/low). This approach borrows from the fact that India’s food loss is typically 10–30% in transit/storage [2], so our proxy labels should roughly align with such magnitudes.

6. Model Selection

We need models that can handle heterogeneous tabular data (prices, stocks, weather) and output both numeric and categorical predictions. Common choices include:

- **Regression models for shelf life and spoilage%:** One can use any regression algorithm to predict *remaining shelf life (days)* and *spoilage percentage*. Good candidates are **Random Forest Regressors** or **Gradient Boosting (XGBoost, LightGBM)** for their robustness on structured data [7]. These ensemble models automatically capture non-linear interactions (e.g. price×humidity) and are less sensitive to scaling. Alternatively, a **Neural Network** (e.g. a multi-layer perceptron or a recurrent LSTM if using sequential inputs) can learn complex patterns, though it requires more data and tuning.
- **Classification for risk level:** A multi-class classifier (e.g. Random Forest Classifier, SVM, or a neural net) can predict {Low/Medium/High} risk. If using an ensemble (RF/XGBoost) as above, one could either have a separate classifier model or derive risk by thresholding the regression outputs. The cited horticulture review notes that *RF and KNN have been effective for tasks like disease detection, while SVMs and neural nets also perform well in complex tasks* [7]. We recommend starting with ensemble trees for ease of training and interpretability.
- **Multi-output models:** It is also possible to train a single TensorFlow model with multiple outputs (e.g. two regression heads for shelf life and spoilage%, plus a softmax head for risk). This may exploit shared features. For example, a feed-forward neural network could output [days_left, spoilage_pct, risk_logits]. Using Keras, one can set up a model with three output layers and train with a composite loss (MSE for regressions + categorical cross-entropy for risk).

Justification: Tree ensembles (RF/XGBoost) often excel on tabular data and provide feature importances to interpret which factors (price, weather, storage) drive spoilage predictions [7]. Neural nets (TensorFlow) are more flexible if data is vast and non-linearities are extreme, but require careful tuning to avoid overfitting. Given the multi-faceted outputs, an *ensemble + simple classifier* pipeline is a strong baseline, with a neural network as a complementary approach if resources permit.

7. Model Training and Evaluation

Step 1 – Data split: Partition the labeled dataset into training, validation, and test sets. A typical split is 70% train / 15% validation / 15% test, ensuring all time periods and commodity types are represented. If treating this as time-series, one might do time-based split (train on older data, test on newer) to simulate live deployment. As shown in Figure 1 below, we split the original data into train/test, train the model on the training set, then evaluate on the test set [8].

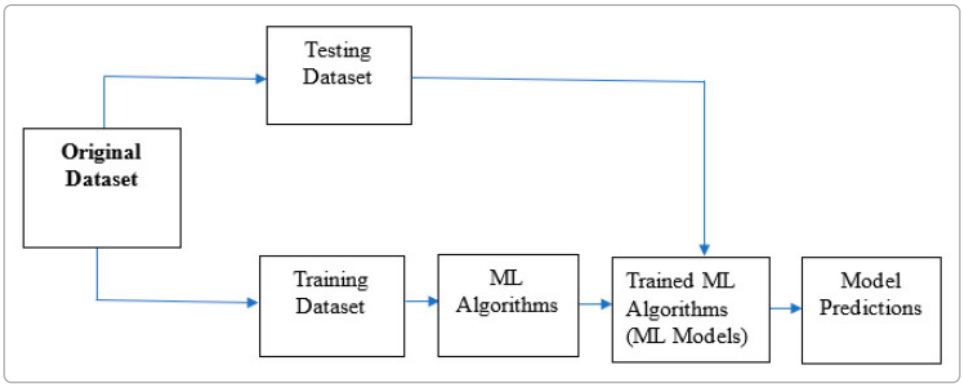

Figure: Example machine learning workflow. The original dataset is split into training and testing sets; models (regression/classification) are trained on the training set and then used to make predictions on test data [8] .

Step 2 – Training: Train the chosen models on the training set. For regression (shelf life, spoilage%), minimize mean squared error (MSE) or mean absolute error (MAE). For classification (risk level), use categorical cross-entropy (for neural nets) or Gini/entropy split criteria (for tree classifiers). Use the validation set to tune hyperparameters (e.g. tree depth, number of trees, neural net layers, learning rate). Perform k-fold cross-validation (or time-series CV) to get robust performance estimates.

Step 3 – Metrics: Evaluate performance separately for each output: - *Shelf-life (regression):* report MAE and RMSE on days; a small MAE (e.g. ±2 days) indicates accuracy.
- *Spoilage% (regression):* similarly, use MAE.
- *Risk level (classification):* compute **accuracy**, **precision**, **recall**, and **F1-score** [9] . Since false negatives (labeling a high-risk as low-risk) have high cost, also report class-wise recall. The horticulture review notes that "precision and recall are important when the cost of false positives/negatives is high" [9] . For example, a high recall for the "High Risk" class ensures few dangerous cases are missed.
- *Overall:* Use confusion matrices to see where misclassifications occur, and check scatter plots of predicted vs. actual shelf life.

If using a combined multi-output network, one can track a weighted sum of all losses, but still evaluate each output separately. Track metrics on validation data after each epoch to avoid overfitting (early stopping).

Step 4 – Thresholding for risk levels: If risk level is not directly predicted, define thresholds on shelf life/spoilage to categorize risk. For instance, items with predicted shelf life < 5 days might be "High risk", 5–15 days as "Medium", >15 "Low". Tune these thresholds based on validation data to balance the class distribution.

8. Integration and Deployment

Once the model is trained and validated, integrate it into the web platform pipeline:

- **Model export:** Save the trained model(s). For TensorFlow/Keras, export a SavedModel or HDF5. For tree models, use joblib or pickle.
- **Backend API:** Build a Python-based API (e.g. Flask, FastAPI) that loads the model on startup. Create endpoints that accept input features (commodity, date, location) and return predictions. As illustrated in deployment guides [10] , the backend takes user input (or runs on new daily data), feeds it to the model, and returns real-time predictions.

- **Scheduling updates:** Set up a daily job to fetch fresh data (market prices, stocks, weather) and run the model to update spoilage predictions for the coming days.
- **Filtering logic:** Implement the shelf-life threshold filter on the backend: if the model predicts *remaining shelf life* < *T* (for example 2 days), exclude that item from listing or flag it. This ensures very short-lived items aren't offered.
- **Frontend display:** In the web UI, display for each item: predicted "Days of Shelf Life", "Spoilage %", and a color-coded risk label (Low/Med/High). For example, an item might show "Shelf Life: 12 days, Spoilage: 30% (Medium risk)".
- **Monitoring:** Log model predictions and compare with any real outcomes over time (e.g. actual spoilage reports). Continuously collect new data to retrain the model periodically as conditions (climate, practices) change.

By following standard ML deployment practices, one ensures the model runs live. In summary, the application workflow is: **Data Ingestion → Feature Prep → Model Prediction → Filter → UI Display**. This mirrors guides on ML integration (train → save → Flask app for input/prediction) [10].

9. Handling the Lack of Direct Spoilage Labels

Since we have no labeled spoilage history, we used proxy strategies (see Section 5). In practice, one may further improve this by:

- **Expert feedback:** Let agronomists or warehouse managers review model predictions initially. Adjust the labeling heuristics based on their insight (e.g. certain crops resist spoilage, others very perish-able).
- **Proxy validation:** Over time, track how often high-risk predictions coincide with actual reported losses or unsold stock. Use these records to refine the model or labels.
- **Simulated data:** If possible, simulate spoilage scenarios under different conditions to create a synthetic training set. For example, model fruit ripening or mold growth under various humidity/temperature and use that physics-based output as labels.

The key point is to start with the best available proxy labels (e.g. *"15% spoilage if all conditions are normal"* from a loss report [2], adjusting upward in adverse conditions) and iteratively refine as real feedback arrives. Many AI applications in agriculture face this challenge; thus unsupervised or semi-supervised methods can complement our supervised model.

10. Conclusion

This guide outlines a comprehensive pipeline to predict spoilage risk for Indian produce using Python-based ML. By combining multiple government datasets (market prices [4], inventory levels [5], weather data, etc.) and engineering relevant features (price volatility, humidity), we create a rich dataset. We recommend tree-ensemble and neural network models for regression/classification tasks [7]. The output (shelf life, spoilage%, risk) can be served live via a web app after filtering by a minimum shelf-life threshold. In the absence of direct spoilage labels, we leverage price and weather proxies and known loss estimates (~10–30% loss [2] [1]) to bootstrap the model. Proper validation (e.g. MAE, F1-score [9]) ensures reliability. Finally, integrating the model into a Flask or similar backend allows real-time prediction for end-users [10].

By deploying such a system, farmers and retailers can proactively manage stocks and reduce waste, ultimately improving food security and economics. Future work includes refining labels with ground-truth spoilage data and incorporating new data streams (e.g. satellite temperature) to further improve accuracy.

Sources: We drew on Indian open datasets for market prices and FCI stocks [4] [5], agricultural ML research on shelf-life prediction [11], and best practices in ML (data splits, metrics) [8] [9]. Agricultural extension literature confirms the importance of storage conditions (e.g. losses up to 40% without proper cooling/humidity) [1], motivating the use of weather features. Deploying ML in a web framework follows known recipes [10]. All steps above synthesize these references into a practical guide.

---

[1] [6] Bulletin #4135, Storage Conditions: Fruits and Vegetables - Cooperative Extension Publications - University of Maine Cooperative Extension
https://extension.umaine.edu/publications/4135e/

[2] [11] TinyML-Sensor for Shelf Life Estimation of Fresh Date Fruits - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC10457898/

[3] AI in Agriculture: Transforming Farming and Yields
https://www.omdena.com/blog/ai-agriculture

[4] Current daily price of various commodities from various markets (Mandi) | Open Government Data (OGD) Platform India
https://www.data.gov.in/catalog/current-daily-price-various-commodities-various-markets-mandi

[5] datagovindia · PyPI
https://pypi.org/project/datagovindia/0.4/

[7] [8] [9] Machine Learning Application in Horticulture and Prospects for Predicting Fresh Produce Losses and Waste: A Review - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC11085577/

[10] Deploy Machine Learning Model using Flask | GeeksforGeeks
https://www.geeksforgeeks.org/deploy-machine-learning-model-using-flask/