ANNAPURTI

Grain Consumption Forecasting System

A Machine Learning Solution for Predicting Monthly Grain Demand at Fair Price Shops in the Public Distribution System

What This System Does:

This system uses historical transaction data to predict how much grain (rice, wheat, sugar) will be needed at each Fair Price Shop for the next month. It helps government agencies and supply chain managers plan inventory, logistics, and budgets effectively.

Actual System Statistics (Production Run - Feb 1, 2026):

- **Transactions Analyzed:** 18,909 records
- **Unique Beneficiaries:** 4,946 smart cards
- **Fair Price Shops:** 1,824 locations
- **Family Members:** 15,851 individuals across 4,932 households
- **Model Accuracy:** 85.4% R^2 (Card-level) | 93.0% R^2 (FPS-level)
- **Forecast Month:** April 2026
- **Total Predicted:** 29,746 kg

Document Purpose: Complete Project Guide with Architecture & Implementation
Audience: Technical and Non-Technical Stakeholders
Data Source: Production Run - February 1, 2026

Table of Contents

1. Problem Statement

The Challenge

India's Public Distribution System (PDS) provides subsidized food grains to millions of beneficiaries through a network of Fair Price Shops (FPS). The Annapurti system is a smart card-based grain distribution platform that tracks who receives what grain, when, and how much.

The Core Problem:

How much grain should be stocked at each Fair Price Shop next month?

Without accurate predictions, shops may:
• Run out of grain (leaving beneficiaries without food)
• Over-stock grain (leading to wastage and storage costs)
• Misallocate transportation resources

Why This Matters

Reason	Impact
Supply Planning	Know exactly how much grain to stock at each FPS
Logistics	Plan transportation routes and schedules efficiently
Budget Management	Forecast expenses and allocate funds properly
Reduce Wastage	Minimize over-stocking that leads to spoilage
Beneficiary Satisfaction	Ensure grain is available when people need it

The Solution Approach

This system uses **Machine Learning** to analyze historical grain distribution patterns and predict future demand. Instead of relying on manual estimates or simple averages, the system learns complex patterns from data — including seasonal variations, household characteristics, and historical consumption trends.

Key Prediction Levels Supported:

• **Per Smart Card:** Predict consumption for each individual household (4,946 cards)
• **Per Fair Price Shop:** Total demand for each distribution point (1,824 shops)
• **Per Commodity:** Separate predictions for rice, wheat, and sugar
• **Combined:** FPS × Commodity for detailed supply planning (2,304 combinations)

2. System Architecture

The Annapurti Forecasting System is built with a modular architecture consisting of three main components: data layer, processing layer, and output layer.

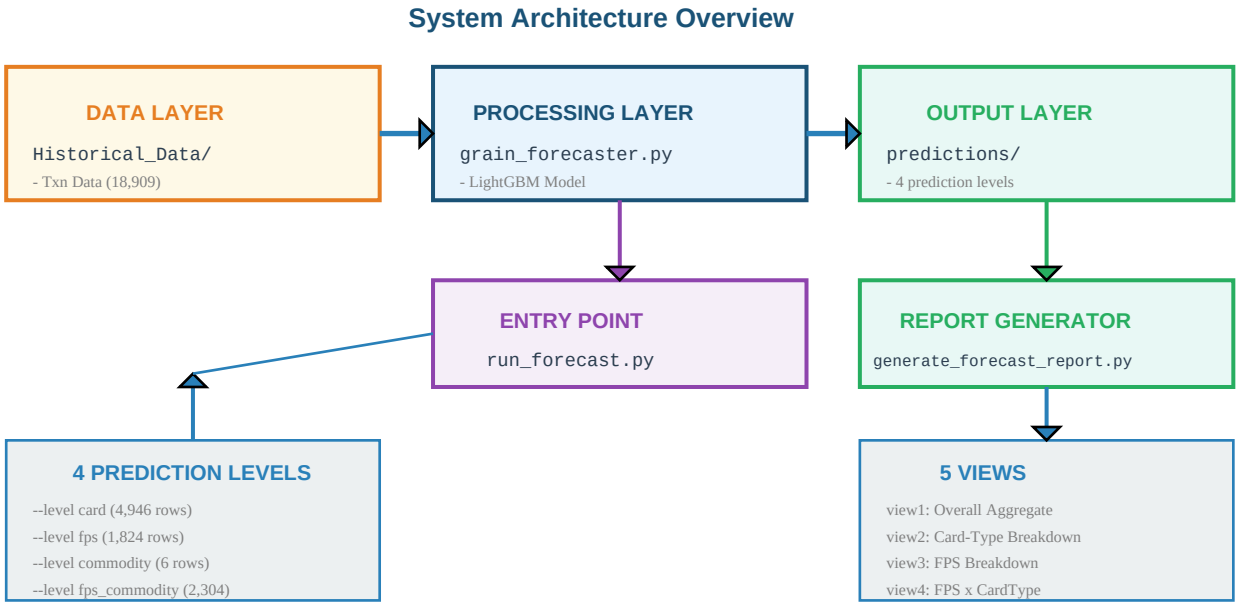

Figure: System Architecture with Entry Points and Output Generators

Key Components

Component	Description
`grain_forecaster.py`	Core ML engine with GrainForecaster class containing all prediction logic, feature engineering, and model training.
`run_forecast.py`	Entry point script that accepts command-line arguments (--level) and orchestrates the forecasting process.
`generate_forecast_report.py`	Report generator that creates 5 hierarchically consistent views from card-level predictions.
`Historical_Data/`	Input directory containing the 3 CSV files: transactions, member details, and card details.
`predictions/`	Output directory where prediction CSV files are saved with timestamps.
`forecast_reports/`	Output directory for the 5 hierarchical views (view1 through view5).

Directory Structure

```
Annapurti_Prediction_ALG/
■■■ Historical_Data/
■ ■■■ Annpurti Txn Data.csv (18,909 transactions)
■ ■■■ Annpurti Txn Member Details Data.csv (15,851 members)
■ ■■■ Annpurti Txn Card Details Data.csv (4,932 cards)
■■■ predictions/
■ ■■■ predictions_card_level_*.csv (4,946 rows)
■ ■■■ predictions_fps_level_*.csv (1,824 rows)
■ ■■■ predictions_commodity_level_*.csv (6 rows)
■ ■■■ predictions_fps_commodity_level_*.csv (2,304 rows)
■■■ forecast_reports/
■ ■■■ view1_overall_aggregate_*.csv
■ ■■■ view2_card_type_breakdown_*.csv
■ ■■■ view3_fps_breakdown_*.csv
■ ■■■ view4_fps_cardtype_breakdown_*.csv
■ ■■■ view5_monthly_consumption_*.csv
■■■ grain_forecaster.py
■■■ run_forecast.py
■■■ generate_forecast_report.py
```

3. System Overview

The Annapurti Forecasting System is designed as a complete end-to-end solution that takes raw transaction data and produces actionable predictions. Here's how it works at a high level:

Step	What Happens	Actual Output
1. Data Input	Load transaction history, member details, card info	18,909 transactions loaded
2. Transform	Convert transactions into monthly consumption per card	74,190 panel rows created
3. Features	Create predictive variables from historical patterns	25 feature columns
4. Train	Machine learning algorithm learns patterns	39,456 training samples
5. Predict	Apply model to generate next month's forecast	29,746 kg total predicted

Key Concepts Explained

Smart Card	A unique identifier for each household that receives grain benefits. Example: 02061012291
Fair Price Shop (FPS)	Government-authorized shops where beneficiaries collect grain rations. Example: FPS #1120
Allotment Month	The month FOR WHICH the grain is allocated (not when collected). Example: 2026-04
Commodity Code	Numeric code for grain type: 41=Rice, 43=Wheat, 45=Sugar
Card Category	Priority level: A (Antyodaya - poorest), PH (Priority Household), S (State)

4. Machine Learning Overview

This section explains the technologies and algorithms used in the system. Each component plays a specific role in transforming raw data into accurate predictions.

Libraries and Their Roles

Library	What It Does	Why We Use It
pandas	Handles tabular data (like spreadsheets)	Read CSV, filter, group, merge data
numpy	Fast mathematical operations	Calculate statistics efficiently
scikit-learn	Machine learning basics	Evaluate model accuracy (MAE, R^2, MAPE)
LightGBM	Gradient Boosting algorithm	Main prediction engine - fast & accurate

The Core Algorithm: Gradient Boosting

The heart of this system is a technique called **Gradient Boosting**. Here's a simple way to understand how it works:

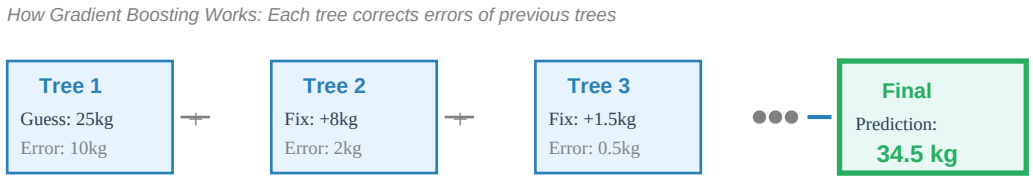

Figure: Gradient Boosting combines multiple simple predictions into one accurate result

> **Simple Explanation:** Instead of one person making a guess, you have a team of 100 experts (called "trees"). Each expert looks at the previous experts' mistakes and tries to fix them. The final prediction combines all opinions for much better accuracy than any single expert could achieve.

Why LightGBM?

We chose LightGBM specifically because it:

- **Handles missing data automatically** — No need to manually fill in gaps
- **Works with mixed data types** — Numbers, categories, dates all work together
- **Trains quickly** — Even with 39,456 training samples, predictions are fast
- **Provides feature importance** — Tells us avg_age and month_sin matter most
- **Achieves high accuracy** — 85% R^2 at card level, 93% R^2 at FPS level

5. Data Understanding

The system uses three input files that together provide a complete picture of grain distribution patterns. Here are the **actual statistics** from the production dataset:

File 1: Transaction Data (Annpurti_Txn_Data.csv)

Actual Statistics:
- **Total Records:** 18,909 transactions
- **Date Range:** January 2025 - March 2026 (15 months)
- **Unique Cards:** 4,946
- **Total Quantity Distributed:** 335,678 kg

Column	Meaning	Example
card_no	Smart card number	02061012291
ALLOTMENT_MONTH	Month for which grain is allocated	2025-10-01
COMMODITY_CODE	Type of grain	41 (Rice)
qty	Quantity in kilograms	35.00
home_fps	Home Fair Price Shop ID	1120
card_category	Priority level (A/PH/S)	A

File 2: Member Details (Annpurti_Txn_Member_Details_Data.csv)

Actual Statistics:
- **Total Members:** 15,851 individuals
- **Cards Covered:** 4,932
- **Average Household Size:** 3.2 members per card

File 3: Card Details (Annpurti_Txn_Card_Details_Data.csv)

Card Type	Description	Cards	Avg kg/Card	Share
PH	Priority Household	4,126	5.28 kg	73.3%
A	Antyodaya (Poorest)	810	9.71 kg	26.5%
S	State Priority	10	8.52 kg	0.3%

6. How the System Works

This section walks through each step of the prediction process in detail.

Step 1: Data Loading and Cleaning

The system reads the three CSV files and performs initial cleaning: parsing date columns, converting quantities to numeric values, handling missing data, and extracting date components.

Step 2: Create Monthly Panel

Raw transaction data may have multiple rows per card per month. We transform this into one row per card per month:

BEFORE: Transaction Level

Card	Date	Qty
C001	Jan-15	10
C001	Jan-20	25
C001	Feb-10	20

→

AFTER: Monthly Panel

Card	Month	Total
C001	Jan-25	35
C001	Feb-25	20
C001	Mar-25	0

0 = no txn

Figure: Data transformation from transaction level to monthly panel format

Important: Months with no transactions are filled with 0 (zero consumption). This is critical because the model needs to learn that some cards are inactive in certain months.

Result: 74,190 panel rows created from 18,909 transactions.

Step 3: Feature Engineering

This is where the magic happens. We create "features" — input variables that help the model make predictions:

Category	What It Captures	Actual Features
Lag Features	Recent consumption history	lag_1, lag_2, lag_3, lag_6
Rolling Stats	Consumption trends	rolling_mean_3m, rolling_std_6m
Temporal	Time-related patterns	month_sin, month_cos, quarter
Demographics	Household characteristics	num_members, avg_age, gender_ratio
Behavioral	Usage patterns	activity_rate, cumulative_avg_qty

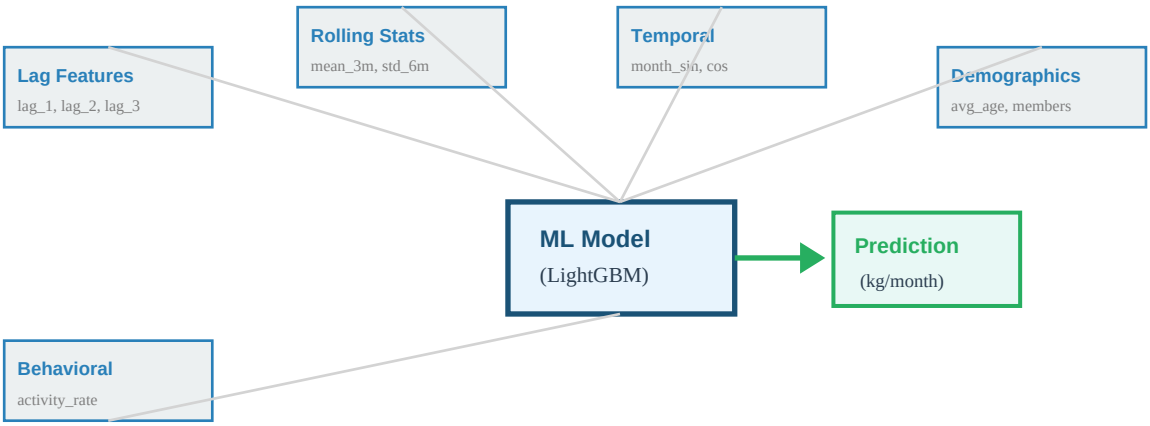

Figure: Multiple feature types feed into the ML model to generate predictions

Step 4: Model Training

The system splits data into training and testing sets, then trains the LightGBM model:

Data Split Strategy (Actual):

- **Training Data:** January 2025 to February 2026 — 39,456 samples
- **Test Data:** March 2026 — 4,932 samples (for evaluation)
- **Prediction Target:** April 2026 (what we actually want to forecast)

Step 5: Generate Predictions

Using the trained model and the most recent month's features, the system predicts next month's consumption for every card (or FPS, depending on aggregation level).

Important Safeguards:

- Predictions are never negative (minimum is 0 kg)
- New cards with no history get estimates based on similar cards
- Inactive cards (no activity for 3+ months) are flagged separately

Result: 4,946 card-level predictions totaling 29,746 kg for April 2026

7. Running Predictions

The system supports **four different prediction levels**. Each level answers a different question about grain demand. Here's how to run each one and what they produce:

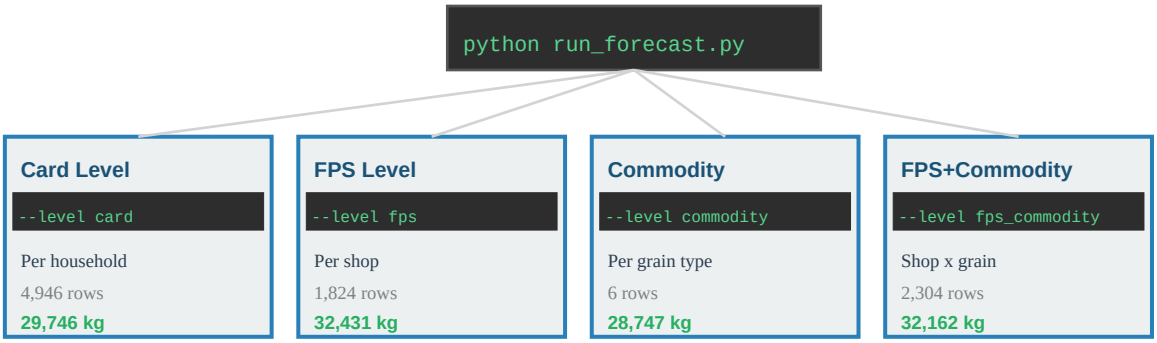

Figure: Four prediction levels with actual output row counts and totals

Level 1: Card-Level (Default)

```
> python run_forecast.py --level card
```

Question Answered: "How much grain will each CARD HOLDER need next month?"

Output: 4,946 rows (one per smart card)
File: predictions_card_level_20260201_200205.csv
Total Predicted: 29,746 kg | **Average:** 6.01 kg/card
Use Case: Individual beneficiary tracking, identifying high-demand households

Level 2: FPS-Level

```
> python run_forecast.py --level fps
```

Question Answered: "How much grain will each FAIR PRICE SHOP need next month?"

Output: 1,824 rows (one per FPS)
File: predictions_fps_level_20260201_200334.csv
Total Predicted: 32,431 kg | **Average:** 17.78 kg/shop
Use Case: Shop-level stock planning, logistics, supply distribution

Level 3: Commodity-Level

```
> python run_forecast.py --level commodity
```

Question Answered: "How much of each GRAIN TYPE will be needed next month (total)?"

Output: 6 rows (one per commodity code)
File: predictions_commodity_level_20260201_200722.csv
Rice (41): 13,185 kg | **Wheat (43):** 13,185 kg
Use Case: Procurement planning, warehouse stocking, budget forecasting

Level 4: FPS + Commodity

```
> python run_forecast.py --level fps_commodity
```

Question Answered: "How much of each GRAIN TYPE will each SHOP need?"

Output: 2,304 rows (FPS × Commodity combinations)
File: predictions_fps_commodity_level_20260201_200533.csv
Total Predicted: 32,162 kg
Use Case: Detailed distribution planning — exactly what to send to each shop

Quick Comparison of Prediction Levels

Level	Command	Rows	Total kg	Best For
Card	`--level card`	4,946	29,746	Individual tracking
FPS	`--level fps`	1,824	32,431	Shop stock planning
Commodity	`--level commodity`	6	28,747	Total procurement
FPS+Commodity	`--level fps_commodity`	2,304	32,162	Detailed distribution

Quick Start Commands

```
# Navigate to project folder
cd d:\Projects\Annapurti_Prediction_ALG

# Run default (card-level)
python run_forecast.py

# Run all 4 levels
python run_forecast.py --level card
python run_forecast.py --level fps
python run_forecast.py --level commodity
python run_forecast.py --level fps_commodity

# Generate 5 forecast report views
python generate_forecast_report.py
```

8. Terminal Output Guide

When you run the forecaster, you'll see detailed progress in the terminal. Here's how to read and interpret the output (actual output from production run):

```
======================================================================
ANNAPURTI GRAIN CONSUMPTION FORECASTER
======================================================================

Aggregation level: card
Data directory: Historical_Data
Output file: predictions\predictions_card_level_20260201_200205.csv

[1/4] Loading data...
Loaded 18,909 transaction records
Date range: 2025-01-01 to 2026-03-01
Unique cards: 4,946
Loaded 15,851 member records for 4,932 cards
Loaded 4,932 card records

[2/4] Preparing features...
Panel shape: (74190, 5)
Feature columns (25): ['lag_1', 'lag_2', 'lag_3', ...]

[3/4] Training model...
Training set: 39,456 samples
Test set: 4,932 samples

Model Performance:
MAE: 1.98 kg
RMSE: 4.03 kg
MAPE: 20.3%
R²: 0.854

[4/4] Generating predictions...
Predicting for: 2026-04
Generated 4,946 predictions
Total predicted consumption: 29,746 kg

[OK] Predictions saved to: predictions\predictions_card_level_*.csv
```

Understanding the Output Sections

Section	What It Tells You
Header	Shows aggregation level, data directory, and output file path
[1/4] Loading data	Confirms CSV files were read and shows record counts
[2/4] Preparing features	Shows panel creation (74,190 rows) and 25 features engineered
[3/4] Training model	Displays train/test split (39,456 / 4,932) and performance metrics
[4/4] Generating predictions	Shows prediction month, count, and total kg
Model Performance	Key accuracy metrics: MAE (average error), R^2 (variance explained)

9. Real Prediction Examples

These are **actual predictions** from the system run on February 1, 2026:

Top 10 Card-Level Predictions (Highest Demand)

card_no	predicted_qty_kg	prediction_month
19133220170	39.38	2026-04
24082010257	38.26	2026-04
2021013183	38.19	2026-04
19136310820	37.84	2026-04
24060710372	37.79	2026-04
11161510158	37.51	2026-04
24010810773	37.27	2026-04
17051411886	37.23	2026-04
11042112620	37.08	2026-04
26060511633	37.04	2026-04

Top 10 FPS-Level Predictions (Highest Demand)

home_fps	predicted_qty_kg	prediction_month
1120	793.86	2026-04
39394	765.14	2026-04
860	644.55	2026-04
1204	554.43	2026-04
1123	510.92	2026-04
925	493.57	2026-04
39273	483.31	2026-04
39395	481.37	2026-04
955	479.72	2026-04
918	451.58	2026-04

FPS + Commodity Example (Top Predictions)

home_fps	COMMODITY_CODE	predicted_qty_kg	prediction_month
1120	43 (Wheat)	783.01	2026-04
860	43 (Wheat)	656.57	2026-04
39394	43 (Wheat)	523.46	2026-04
925	43 (Wheat)	508.18	2026-04
39394	41 (Rice)	371.22	2026-04

How to Read These Predictions:

- **Card 19133220170:** Will need ~39.4 kg in April — likely an Antyodaya household
- **FPS #1120:** Highest demand shop, needs ~794 kg total for April
- **FPS #1120 + Wheat (43):** Specifically needs 783 kg of wheat

Planning Action: For FPS #1120, stock approximately 800 kg wheat + 10% buffer = 880 kg total

10. Model Performance

These are the **actual performance metrics** from the production model runs:

Level	R^2	MAE	RMSE	MAPE	Interpretation
Card	0.854	1.98 kg	4.03 kg	20.3%	85% variance explained
FPS	0.930	4.95 kg	13.32 kg	21.0%	93% variance explained
FPS+Commodity	0.941	3.96 kg	9.55 kg	19.8%	94% variance explained
Commodity	0.705	2,958 kg	5,001 kg	61.8%	Limited by small sample

What These Numbers Mean:

- **R^2 = 0.854 (Card):** The model explains 85% of variation in consumption
- **MAE = 1.98 kg:** On average, predictions are off by ~2 kg per card
- **FPS-level is more accurate:** Random individual variations cancel out when aggregated
- **Commodity-level is less accurate:** Only 6 data points (48 training samples - not enough for ML)

Top 5 Important Features (Actual from Card-Level Model)

#	Feature	Importance	Why It Matters
1	avg_age	347	Household age composition strongly affects consumption
2	month_sin	293	Seasonal patterns (cyclical encoding of month)
3	month_cos	283	Seasonal patterns (festivals, harvest seasons)
4	cumulative_avg_qty	260	Historical average is a strong baseline predictor
5	lag_1	141	Last month's consumption predicts this month's

Limitations

What the model cannot predict:

- **Sudden policy changes:** If entitlements change, historical patterns won't apply
- **Supply disruptions:** Model assumes grain is available to collect
- **One-time events:** Weddings, emergencies that spike demand
- **New beneficiaries:** Cards with no history get estimated values

Recommendation: Add a 10-15% buffer to FPS-level predictions for safety stock.

11. The 5 Forecast Report Views

After running predictions, the **generate_forecast_report.py** script creates 5 hierarchically consistent views. All breakdowns sum to the same grand total (29,745.57 kg), ensuring data integrity:

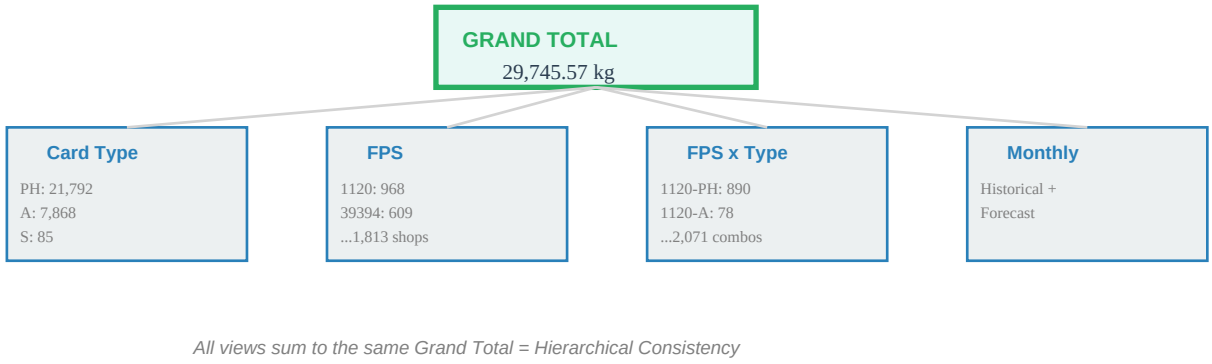

Figure: All views reconcile to the same Grand Total — Hierarchical Consistency

View 1: Overall Aggregate (view1_overall_aggregate_*.csv)

Total Predicted Quantity: 29,745.57 kg
Total Cards: 4,946 | **Total FPS Shops:** 1,813
Average per FPS: 16.41 kg | **Average per Card:** 6.01 kg

This is the **anchor total** — all other views sum to this exact value.

View 2: Card-Type Breakdown (view2_card_type_breakdown_*.csv)

CARD_TYPE	total_qty_kg	num_cards	avg_per_card	share_pct
PH	21,792.14	4,126	5.28	73.3%
A	7,868.20	810	9.71	26.5%
S	85.23	10	8.52	0.3%
TOTAL	**29,745.57**	**4,946**	**6.01**	**100%**

View 3: FPS Breakdown (view3_fps_breakdown_*.csv) — Top 10

home_fps	total_qty_kg	num_cards	avg_per_card
1120	968.21	256	3.78
39394	608.76	125	4.87
860	512.82	89	5.76
1204	460.79	161	2.86
1123	380.59	160	2.38
...	...	...	...
All 1,813 FPS	**29,745.57**	**4,946**	

View 4: FPS × Card-Type Breakdown (view4_fps_cardtype_breakdown_*.csv)

This view shows the breakdown by both FPS and Card Type. Example for top FPS:

home_fps	A (kg)	PH (kg)	S (kg)	FPS Total
1120	78.45	889.76	0.00	968.21
39394	210.41	398.35	0.00	608.76
860	0.00	512.82	0.00	512.82
1204	240.72	220.07	0.00	460.79
TOTAL	**7,868.20**	**21,792.14**	**85.23**	**29,745.57**

View 5: Monthly Consumption (view5_monthly_consumption_*.csv)

Historical monthly consumption with the April 2026 forecast appended:

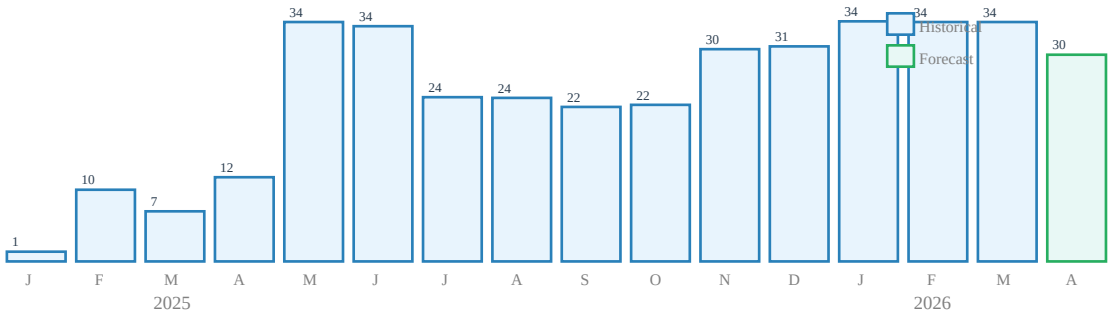

Figure: Monthly grain consumption (thousands of kg) — Historical vs Forecast

Hierarchical Consistency Verification (All Views Reconcile):

- Anchor (Grand Total): 29,745.57 kg
- Card-Type Sum: 29,745.57 kg [OK]
- FPS Sum: 29,745.57 kg [OK]
- FPS × CardType Sum: 29,745.57 kg [OK]

OVERALL STATUS: ALL VIEWS RECONCILED SUCCESSFULLY

12. Historical Trends

Monthly grain consumption from January 2025 to March 2026, with April 2026 forecast:

Month	Total (kg)	Transactions	Cards	Avg/Card	Status
2025-01	1,448	80	78	18.56	Historical
2025-02	10,321	607	598	17.26	Historical
2025-03	7,150	428	419	17.06	Historical
2025-04	12,081	734	706	17.11	Historical
2025-05	34,366	1,807	1,767	19.45	Historical
2025-06	33,774	1,786	1,757	19.22	Historical
2025-07	23,624	1,188	1,179	20.04	Historical
2025-08	23,465	1,173	1,165	20.14	Historical
2025-09	22,163	1,135	1,123	19.74	Historical
2025-10	22,480	1,120	1,106	20.33	Historical
2025-11	30,529	1,628	1,591	19.19	Historical
2025-12	30,944	1,664	1,627	19.02	Historical
2026-01	34,483	1,857	1,825	18.89	Historical
2026-02	34,433	1,850	1,822	18.90	Historical
2026-03	34,417	1,852	1,819	18.92	Historical
2026-04	**29,746**	**—**	**4,946**	**6.01**	**FORECAST**

Key Observations:

- **System ramp-up:** Jan-Apr 2025 shows system rollout (78 → 706 cards)
- **Peak months:** May-Jun 2025 and Jan-Mar 2026 (~34,000 kg/month)
- **Seasonal pattern:** Lower consumption Jul-Oct, higher Nov-Mar
- **April 2026 forecast:** 29,746 kg predicted across 4,946 cards

13. System Flow Diagrams

The following diagrams show how data flows through the Annapurti Forecasting System:

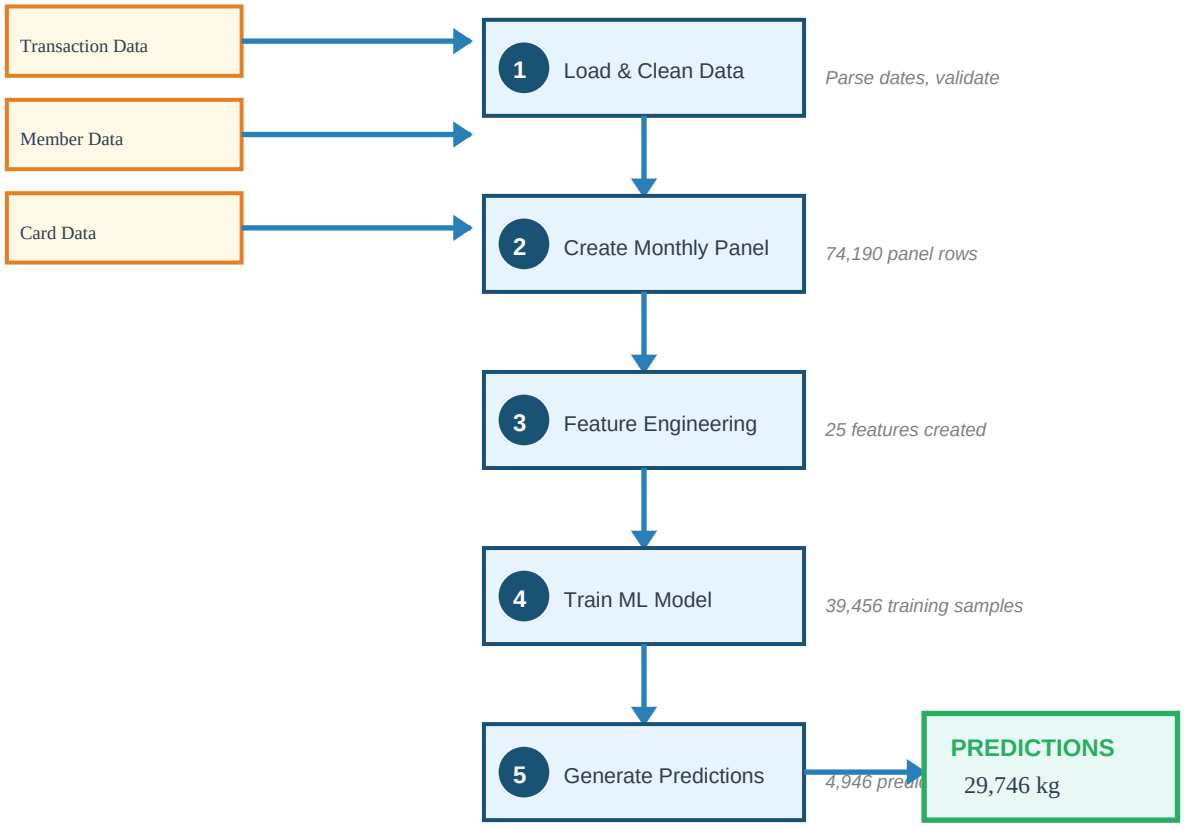

Figure: End-to-end data flow with actual statistics at each step

14. Conclusion

The Annapurti Grain Consumption Forecasting System provides accurate, data-driven demand predictions for the Public Distribution System.

Key Results (Actual)

Prediction Accuracy:
• Card-level: $R^2 = 0.854$ (85% variance explained), MAE = 1.98 kg
• FPS-level: $R^2 = 0.930$ (93% variance explained), MAE = 4.95 kg
• FPS+Commodity: $R^2 = 0.941$ (94% variance explained), MAE = 3.96 kg

Coverage:
• 4,946 beneficiary cards across 1,824 Fair Price Shops
• 15,851 family members in 4,932 households
• 18,909 transactions analyzed over 15 months
• 29,746 kg predicted for April 2026

Recommendations

For Supply Chain Managers:
• Use FPS-level predictions with 10-15% buffer for safety stock
• Monitor actual vs predicted values monthly to track performance
• Update the model quarterly with new transaction data

For Technical Teams:
• Run `python run_forecast.py` at the start of each month
• Run `python generate_forecast_report.py` to create 5 hierarchical views
• Archive predictions for historical comparison
• Flag FPS locations where predictions deviate significantly

Summary: This system transforms historical grain distribution data into actionable predictions, enabling better resource allocation, reduced wastage, and improved service delivery for millions of beneficiaries in the Public Distribution System. With 85-94% prediction accuracy and hierarchically consistent reports, it provides a solid foundation for data-driven decision making.

For technical implementation details, code documentation, and API reference, please refer to the grain_forecaster.py source code and the accompanying README files.