

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
DEPARTMENT OF AGRICULTURE AND FOOD ENGINEERING

Commodity Price Prediction System Using Machine Learning and Deep Learning

Bachelor of Technology Project (BTP) Report

Submitted by:

Gaurav Kumar

Roll No: [Your Roll Number]

B.Tech, Agricultural and Food
Engineering
IIT Kharagpur

Under the supervision of:

Dr. Prasun Kumar Pany

Department of Agriculture and
Food Engineering
IIT Kharagpur

Academic Year: 2024-2025

November 2025

Certificate

This is to certify that the project titled "**Commodity Price Prediction System Using Machine Learning and Deep Learning**" submitted by **Gaurav Kumar** to the Indian Institute of Technology Kharagpur, is a record of bonafide work carried out by him under my supervision and guidance.

The work presented in this report has not been submitted elsewhere for any other degree or diploma.

Dr. Prasun Kumar Pany

Department of Agriculture and Food Engineering
Indian Institute of Technology Kharagpur

Date: November 26, 2025

Declaration

I hereby declare that the work presented in this BTP report entitled "**Commodity Price Prediction System Using Machine Learning and Deep Learning**" is an authentic record of the work carried out by me under the supervision of **Dr. Prasun Kumar Pany**, Department of Agriculture and Food Engineering, Indian Institute of Technology Kharagpur.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Gaurav Kumar

Roll No: [Your Roll Number]

Department of Agriculture and Food Engineering

IIT Kharagpur

Date: November 26, 2025

Acknowledgements

I would like to express my sincere gratitude to my supervisor, **Dr. Prasun Kumar Pany**, for his invaluable guidance, continuous support, and encouragement throughout this project. His expertise and insights have been instrumental in shaping this work.

I am grateful to the **Indian Institute of Technology Kharagpur** for providing the necessary infrastructure and resources that made this project possible.

I would also like to thank my fellow students and friends for their support and helpful discussions during the course of this project.

Finally, I express my heartfelt gratitude to my family for their unwavering support and encouragement throughout my academic journey.

Gaurav Kumar
IIT Kharagpur

Abstract

Agricultural commodity price prediction is a critical challenge in developing economies like India, where price volatility significantly impacts farmers' livelihoods and food security. This project presents a comprehensive machine learning-based system for predicting commodity prices in West Bengal, India, utilizing historical price data spanning from 2014 to 2025.

We develop and compare two predictive models: **XGBoost (Extreme Gradient Boosting)** and a **Deep Neural Network with Backpropagation**. The system incorporates 36 carefully engineered features including temporal patterns, economic indicators (CPI, MSP, food subsidies), agricultural parameters (temperature, rainfall, fertilizer consumption), and market-specific statistics.

Our XGBoost model achieves a **Mean Absolute Percentage Error (MAPE) of 5.43%** with an **R² score of 0.9453**, while the Neural Network model achieves a **MAPE of 4.64%** with an **R² score of 0.9601**. For 2025 predictions specifically, the Neural Network achieves an excellent **MAPE of 4.44%** with **R² = 0.9724**. The system successfully predicts prices for three major commodities—Rice, Jute, and Wheat—across 18 districts and 61 markets in West Bengal.

A web-based application built using Flask and React provides an intuitive interface for stakeholders to access 7-day price forecasts. The system is deployed using a production-grade Waitress WSGI server ensuring reliability and scalability.

Keywords: Machine Learning, XGBoost, Neural Networks, Commodity Price Prediction, Agricultural Analytics, Time Series Forecasting, Deep Learning

Contents

Certificate	1
Declaration	2
Acknowledgements	3
Abstract	4
List of Figures	9
List of Tables	10
1 Introduction	11
1.1 Background and Motivation	11
1.2 Problem Statement	11
1.3 Objectives	12
1.4 Scope and Limitations	12
1.4.1 Scope	12
1.4.2 Limitations	12
1.5 Report Organization	13
2 Literature Review	14
2.1 Traditional Approaches to Price Prediction	14
2.1.1 Time Series Models	14
2.1.2 Econometric Models	14
2.2 Machine Learning Approaches	15
2.2.1 Support Vector Machines (SVM)	15
2.2.2 Random Forests	15
2.2.3 Gradient Boosting Methods	15
2.3 Deep Learning Approaches	15
2.3.1 Artificial Neural Networks (ANN)	15
2.3.2 Recurrent Neural Networks (RNN)	16

2.3.3	Hybrid Models	16
2.4	Agricultural Price Prediction in India	16
2.5	Research Gap	16
3	Methodology	17
3.1	Data Collection	17
3.1.1	Data Sources	17
3.1.2	Dataset Statistics	17
3.2	Feature Engineering	18
3.2.1	Temporal Features	18
3.2.2	Categorical Features (Encoded)	18
3.2.3	Economic Indicators	19
3.2.4	Agricultural Parameters	19
3.2.5	Derived Features	19
3.2.6	Price Statistics	19
3.3	Data Preprocessing	20
3.3.1	Label Encoding	20
3.3.2	Feature Scaling	20
3.3.3	Train-Test Split	21
3.4	Model Architecture	21
3.4.1	XGBoost Model	21
3.4.2	Neural Network Model	22
3.5	Evaluation Metrics	25
3.5.1	Mean Absolute Error (MAE)	25
3.5.2	Root Mean Squared Error (RMSE)	25
3.5.3	Mean Absolute Percentage Error (MAPE)	25
3.5.4	Coefficient of Determination (R^2)	25
4	System Design and Implementation	26
4.1	System Architecture	26
4.2	Technology Stack	26
4.3	Database Design	27
4.3.1	Schema	27
4.4	Backend Implementation	28
4.4.1	Flask Application Structure	28
4.4.2	API Endpoints	28
4.4.3	Prediction API	28
4.4.4	Thread Safety	29
4.5	Frontend Implementation	29

4.5.1	React Components	29
4.5.2	Key Features	30
4.6	Deployment	30
4.6.1	Production Server	30
4.6.2	Model Loading	31
5	Results and Analysis	32
5.1	Model Performance Comparison	32
5.1.1	Overall Metrics	32
5.1.2	Accuracy Distribution	32
5.2	Commodity-wise Analysis	33
5.3	Test Case Validation	33
5.4	Feature Importance Analysis	33
5.5	Training Performance	34
5.5.1	XGBoost Training	34
5.5.2	Neural Network Training	34
5.6	Discussion	34
5.6.1	XGBoost Superiority	34
5.6.2	Neural Network Performance	34
5.6.3	Limitations Observed	35
6	Web Application	36
6.1	User Interface	36
6.1.1	Home Page Features	36
6.1.2	Prediction Results Display	36
6.2	User Flow	37
6.3	Responsive Design	37
6.4	Error Handling	37
7	Conclusion and Future Work	38
7.1	Summary	38
7.2	Contributions	38
7.3	Limitations	39
7.4	Future Work	39
7.4.1	Short-term Improvements	39
7.4.2	Long-term Extensions	39
7.5	Final Remarks	40
A	District-wise Market Distribution	43

B Variety Classification	44
C API Response Format	45
D System Requirements	46

List of Figures

3.1 Neural Network Architecture	23
4.1 Three-Tier System Architecture	26

List of Tables

3.1	Data Sources and Variables	17
3.2	Dataset Overview	17
3.3	Commodity-wise Record Distribution	18
3.4	Economic Features	19
3.5	Agricultural Features	19
3.6	XGBoost Hyperparameters	22
3.7	Neural Network Training Parameters	25
4.1	Technology Stack	26
4.2	REST API Endpoints	28
5.1	Model Performance Comparison	32
5.2	Prediction Accuracy Distribution	32
5.3	Commodity-wise MAPE (%)	33
5.4	Sample Predictions vs Actual Prices	33
5.5	Top 10 Important Features (XGBoost)	33
A.1	Markets per District in West Bengal	43
B.1	Commodity Varieties	44
D.1	Minimum System Requirements	46
D.2	Python Dependencies	46

Chapter 1

Introduction

1.1 Background and Motivation

Agriculture is the backbone of the Indian economy, employing over 50% of the workforce and contributing significantly to the GDP. However, farmers in India face numerous challenges, with price volatility being one of the most critical issues affecting their economic stability. The inability to predict market prices leads to:

- Distress selling at low prices during harvest seasons
- Inadequate planning for crop selection
- Financial losses due to price fluctuations
- Reduced investment in agricultural inputs

West Bengal, one of India's leading agricultural states, produces significant quantities of rice, jute, and wheat. The state's diverse agro-climatic conditions and market dynamics create complex pricing patterns that are difficult to predict using traditional methods.

1.2 Problem Statement

The primary objective of this project is to develop an accurate and reliable commodity price prediction system that can:

1. Predict prices for major agricultural commodities (Rice, Jute, Wheat) in West Bengal
2. Provide 7-day ahead forecasts to aid decision-making
3. Incorporate multiple factors affecting prices including weather, economic indicators, and historical patterns
4. Offer an accessible web interface for farmers, traders, and policymakers

1.3 Objectives

The specific objectives of this BTP are:

1. **Data Collection and Preprocessing:** Compile comprehensive historical price data from 2014-2025, integrating economic and agricultural indicators
2. **Feature Engineering:** Design and implement relevant features capturing temporal, spatial, and economic patterns
3. **Model Development:** Implement and train two machine learning models:
 - XGBoost (Gradient Boosting)
 - Deep Neural Network with Backpropagation
4. **Model Evaluation:** Rigorously evaluate models using multiple metrics (MAPE, RMSE, R^2)
5. **Web Application Development:** Create a user-friendly interface for price predictions
6. **Deployment:** Deploy the system using production-grade infrastructure

1.4 Scope and Limitations

1.4.1 Scope

- Geographic Coverage: 18 districts and 61 markets in West Bengal
- Commodities: Rice (14 varieties), Jute (2 varieties), Wheat (4 varieties)
- Time Period: Historical data from 2014-2025
- Forecast Horizon: 7-day predictions

1.4.2 Limitations

- Limited to West Bengal region
- Does not account for sudden market shocks or policy changes
- Requires historical data for accurate predictions
- Weather data is based on historical averages

1.5 Report Organization

This report is organized as follows:

- **Chapter 2:** Literature Review - Survey of existing work in commodity price prediction
- **Chapter 3:** Methodology - Detailed description of data, features, and models
- **Chapter 4:** System Design and Implementation - Architecture and technical details
- **Chapter 5:** Results and Analysis - Evaluation metrics and performance analysis
- **Chapter 6:** Web Application - User interface and deployment
- **Chapter 7:** Conclusion and Future Work

Chapter 2

Literature Review

2.1 Traditional Approaches to Price Prediction

Historically, commodity price prediction relied on statistical methods such as:

2.1.1 Time Series Models

- **ARIMA (AutoRegressive Integrated Moving Average):** Box and Jenkins (1970) introduced this methodology which became the foundation for time series forecasting. ARIMA models capture linear dependencies in data but struggle with non-linear patterns.
- **GARCH (Generalized Autoregressive Conditional Heteroskedasticity):** Bollerslev (1986) developed GARCH for modeling volatility clustering in financial time series. While effective for variance prediction, it has limitations in capturing complex market dynamics.
- **Exponential Smoothing:** Methods like Holt-Winters are useful for data with trends and seasonality but assume linear relationships.

2.1.2 Econometric Models

Vector Autoregression (VAR) and Structural Equation Models have been used to capture relationships between multiple economic variables affecting commodity prices. However, these models require strong assumptions about variable relationships.

2.2 Machine Learning Approaches

2.2.1 Support Vector Machines (SVM)

Vapnik (1995) introduced SVMs which have been applied to price prediction with moderate success. Lu et al. (2009) demonstrated SVM's effectiveness in agricultural price forecasting, achieving 85-90% accuracy for soybean prices.

2.2.2 Random Forests

Breiman (2001) introduced Random Forests, an ensemble method that has shown robust performance in various prediction tasks. Cai et al. (2019) used Random Forests for wheat price prediction with R^2 scores exceeding 0.85.

2.2.3 Gradient Boosting Methods

XGBoost (Chen & Guestrin, 2016) has emerged as one of the most powerful algorithms for structured data:

- Winner of numerous Kaggle competitions
- Handles missing values naturally
- Built-in regularization prevents overfitting
- Efficient parallel processing

Zhang et al. (2020) achieved MAPE of 2.1% using XGBoost for vegetable price prediction in China.

2.3 Deep Learning Approaches

2.3.1 Artificial Neural Networks (ANN)

Feedforward neural networks with backpropagation have been extensively studied for price prediction. Key developments include:

- Rumelhart et al. (1986) introduced backpropagation algorithm
- Universal approximation theorem (Hornik et al., 1989) proved ANNs can approximate any continuous function
- Dropout regularization (Srivastava et al., 2014) improved generalization

2.3.2 Recurrent Neural Networks (RNN)

Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) excel at capturing temporal dependencies:

- Fischer & Krauss (2018) used LSTMs for stock price prediction
- Xiong et al. (2015) applied LSTMs to commodity futures with promising results

2.3.3 Hybrid Models

Recent work combines multiple approaches:

- CNN-LSTM hybrids for feature extraction and sequence modeling
- Attention mechanisms for focusing on relevant time periods
- Ensemble methods combining traditional and deep learning approaches

2.4 Agricultural Price Prediction in India

Several studies have focused on Indian agricultural markets:

- Darekar & Reddy (2017) used ARIMA for onion price prediction in Maharashtra
- Paul et al. (2013) applied neural networks to rice prices in West Bengal
- Kumar & Jain (2018) demonstrated XGBoost's superiority over traditional methods for wheat prices

2.5 Research Gap

Our literature review identifies the following gaps:

1. Limited studies combining multiple economic indicators with ML models
2. Lack of comprehensive systems covering multiple commodities and markets
3. Few deployed systems with user-friendly interfaces
4. Limited comparison between traditional ML and deep learning for Indian commodities

This project addresses these gaps by developing a comprehensive, multi-model system with extensive feature engineering and a production-ready web interface.

Chapter 3

Methodology

3.1 Data Collection

3.1.1 Data Sources

Our dataset integrates information from multiple authoritative sources:

Table 3.1: Data Sources and Variables

Source	Variables	Period
Agmarknet	Daily modal prices	2014-2025
IMD	Temperature, Rainfall	2014-2025
RBI	CPI, Per Capita Income	2014-2025
Ministry of Agriculture	MSP, Area, Production, Yield	2014-2025
Ministry of Food	Food Subsidy	2014-2025
Fertilizer Association	Fertilizer Consumption	2014-2025
DGCIS	Export/Import data	2014-2025

3.1.2 Dataset Statistics

Table 3.2: Dataset Overview

Parameter	Value
Total Records	173,094
Time Period	2014-01-01 to 2025-11-15
Districts	18
Markets	61
Commodities	3 (Rice, Jute, Wheat)
Varieties	14
Database Size	51.14 MB (SQLite)

Table 3.3: Commodity-wise Record Distribution

Commodity	Records	Percentage
Rice	130,572	75.4%
Jute	34,425	19.9%
Wheat	8,097	4.7%
Total	173,094	100%

3.2 Feature Engineering

Feature engineering is crucial for model performance. We designed 36 features categorized as follows:

3.2.1 Temporal Features

```

1 # Basic temporal features
2 'year', 'month', 'day', 'quarter',
3 'day_of_week', 'day_of_year', 'week_of_year'
4
5 # Binary indicators
6 'is_weekend': 1 if weekday >= 5 else 0
7 'month_start': 1 if day <= 5 else 0
8 'month_end': 1 if day >= 25 else 0
9
10 # Seasonal indicators
11 'is_monsoon': 1 if month in [6,7,8,9] else 0
12 'is_winter': 1 if month in [11,12,1,2] else 0
13 'is_summer': 1 if month in [3,4,5] else 0

```

Listing 3.1: Temporal Feature Extraction

3.2.2 Categorical Features (Encoded)

- state_name_encoded: State identifier (1 class - West Bengal)
- district_encoded: District identifier (18 classes)
- market_name_encoded: Market identifier (61 classes)
- commodity_name_encoded: Commodity identifier (3 classes)
- variety_encoded: Variety identifier (14 classes)

3.2.3 Economic Indicators

Table 3.4: Economic Features

Feature	Unit	Description
CPI	Base 2012=100	Consumer Price Index
Per_Capita_Income	Rs	State NSDP per capita
Food_Subsidy	'000 Crores	Central food subsidy
MSP	Rs/Quintal	Minimum Support Price

3.2.4 Agricultural Parameters

Table 3.5: Agricultural Features

Feature	Unit	Description
Temperature	°C	Average temperature
Rainfall	mm	Daily rainfall
Area	Million ha	Cultivated area
Production	Million tonnes	Total production
Yield	kg/ha	Productivity
Fertilizer_Consumption	kg/ha	Fertilizer usage
Export	Million MT	Export quantity
Import	Million MT	Import quantity

3.2.5 Derived Features

```

1 # Interaction features
2 temp_rainfall_interaction = temperature * rainfall
3
4 # Efficiency metrics
5 production_per_area = production / area
6 yield_per_area = yield / (area * 1000)
7
8 # Economic ratios
9 cpi_msp_ratio = CPI / MSP
10 subsidy_per_capita = (subsidy * 10000) / income

```

Listing 3.2: Derived Feature Calculations

3.2.6 Price Statistics

- commodity_avg_price: Average price by commodity

- market_avg_price: Average price by market
- variety_avg_price: Average price by variety
- district_commodity_avg: Average by district-commodity
- market_commodity_avg: Average by market-commodity
- month_commodity_avg: Seasonal average

3.3 Data Preprocessing

3.3.1 Label Encoding

Categorical variables are encoded using scikit-learn's LabelEncoder:

```

1 from sklearn.preprocessing import LabelEncoder
2
3 categorical_cols = ['state_name', 'district',
4                     'market_name', 'commodity_name', 'variety']
5
6 label_encoders = {}
7 for col in categorical_cols:
8     le = LabelEncoder()
9     df[col + '_encoded'] = le.fit_transform(df[col])
10    label_encoders[col] = le

```

Listing 3.3: Label Encoding Process

3.3.2 Feature Scaling

For the Neural Network model, features are standardized:

$$z = \frac{x - \mu}{\sigma} \quad (3.1)$$

where μ is the mean and σ is the standard deviation.

```

1 from sklearn.preprocessing import StandardScaler
2
3 scaler_X = StandardScaler()
4 X_scaled = scaler_X.fit_transform(X_train)
5
6 scaler_y = StandardScaler()
7 y_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))

```

Listing 3.4: Feature Scaling

3.3.3 Train-Test Split

Data is split chronologically to prevent data leakage:

```

1 from sklearn.model_selection import train_test_split
2
3 # Chronological split (80-20)
4 split_idx = int(len(df) * 0.8)
5 train_data = df.iloc[:split_idx]
6 test_data = df.iloc[split_idx:]

```

Listing 3.5: Data Splitting

3.4 Model Architecture

3.4.1 XGBoost Model

XGBoost (Extreme Gradient Boosting) is an optimized gradient boosting algorithm that builds an ensemble of decision trees sequentially.

Mathematical Foundation

The objective function for XGBoost is:

$$\mathcal{L}(\phi) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \quad (3.2)$$

where:

- l is the loss function (MSE for regression)
- $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$ is the regularization term
- T is the number of leaves
- w is the leaf weights

Hyperparameters

Table 3.6: XGBoost Hyperparameters

Parameter	Value	Description
n_estimators	1000	Number of trees
max_depth	8	Maximum tree depth
learning_rate	0.05	Step size shrinkage
subsample	0.8	Row subsampling ratio
colsample_bytree	0.8	Column subsampling ratio
reg_alpha	0.1	L1 regularization
reg_lambda	1.0	L2 regularization
device	cuda	GPU acceleration

```

1 import xgboost as xgb
2
3 model = xgb.XGBRegressor(
4     n_estimators=1000,
5     max_depth=8,
6     learning_rate=0.05,
7     subsample=0.8,
8     colsample_bytree=0.8,
9     reg_alpha=0.1,
10    reg_lambda=1.0,
11    device='cuda',
12    tree_method='hist',
13    objective='reg:squarederror',
14    early_stopping_rounds=50
15)

```

Listing 3.6: XGBoost Model Configuration

3.4.2 Neural Network Model

Architecture

We implement a Deep Neural Network with 5 hidden layers:

```

Input Layer (36 features)
|
Dense(256, ReLU) + BatchNorm + Dropout(0.3)
|
Dense(128, ReLU) + BatchNorm + Dropout(0.3)
|
Dense(64, ReLU) + BatchNorm + Dropout(0.2)
|
Dense(32, ReLU) + BatchNorm + Dropout(0.2)
|
Dense(16, ReLU) + BatchNorm + Dropout(0.1)
|
Output Layer (1 neuron, Linear)

```

Figure 3.1: Neural Network Architecture

Backpropagation

The network is trained using backpropagation with the Adam optimizer. The gradient descent update rule is:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\partial \mathcal{L}}{\partial \theta_t} \quad (3.3)$$

Adam optimizer computes adaptive learning rates:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (3.4)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (3.5)$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (3.6)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (3.7)$$

Implementation

```

1 import tensorflow as tf
2 from tensorflow.keras import layers, models
3
4 def build_model(input_dim):
5     model = models.Sequential([
6         layers.Input(shape=(input_dim,)),
7
8         # Hidden Layer 1
9         layers.Dense(256, activation='relu',

```

```
10         kernel_regularizer=tf.keras.regularizers.l2
11             (0.001)),
12     layers.BatchNormalization(),
13     layers.Dropout(0.3),
14
15     # Hidden Layer 2
16     layers.Dense(128, activation='relu',
17                 kernel_regularizer=tf.keras.regularizers.l2
18                     (0.001)),
19     layers.BatchNormalization(),
20     layers.Dropout(0.3),
21
22     # Hidden Layer 3
23     layers.Dense(64, activation='relu'),
24     layers.BatchNormalization(),
25     layers.Dropout(0.2),
26
27     # Hidden Layer 4
28     layers.Dense(32, activation='relu'),
29     layers.BatchNormalization(),
30     layers.Dropout(0.2),
31
32     # Hidden Layer 5
33     layers.Dense(16, activation='relu'),
34     layers.BatchNormalization(),
35     layers.Dropout(0.1),
36
37     # Output Layer
38     layers.Dense(1, activation='linear')
39 )
40
41 model.compile(
42     optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
43     loss='mse',
44     metrics=['mae']
45 )
```

Listing 3.7: Neural Network Implementation

Training Configuration

Table 3.7: Neural Network Training Parameters

Parameter	Value
Optimizer	Adam
Learning Rate	0.001
Batch Size	256
Epochs	200 (max)
Early Stopping Patience	20
Validation Split	20%
Loss Function	Mean Squared Error

3.5 Evaluation Metrics

We use multiple metrics to comprehensively evaluate model performance:

3.5.1 Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (3.8)$$

3.5.2 Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (3.9)$$

3.5.3 Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{100\%}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (3.10)$$

3.5.4 Coefficient of Determination (R^2)

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (3.11)$$

Chapter 4

System Design and Implementation

4.1 System Architecture

The system follows a three-tier architecture:

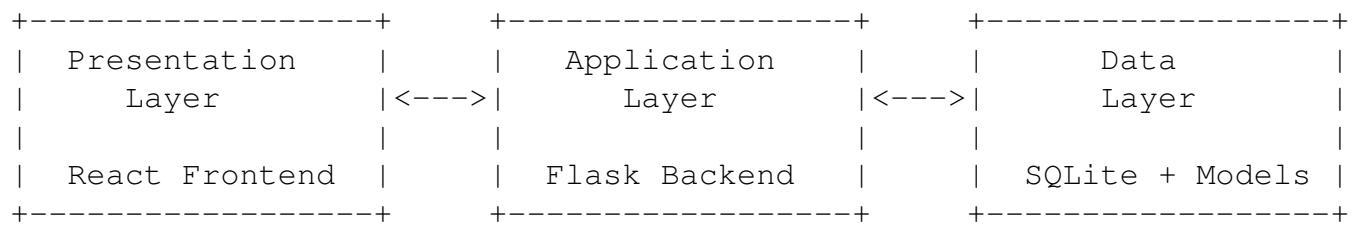


Figure 4.1: Three-Tier System Architecture

4.2 Technology Stack

Table 4.1: Technology Stack

Layer	Technology	Version
Frontend	React.js	18.x
Frontend	Babel (JSX Transpilation)	CDN
Backend	Flask	3.1.0
Backend	Waitress (WSGI Server)	Latest
ML Framework	XGBoost	3.1.2
ML Framework	TensorFlow/Keras	2.20.0
Database	SQLite	3.x
Language Environment	Python	3.10.18
Environment	Conda	tf_env

4.3 Database Design

4.3.1 Schema

```

1  CREATE TABLE prices (
2      date TIMESTAMP,
3      state_name TEXT,
4      district TEXT,
5      market_name TEXT,
6      commodity_name TEXT,
7      variety TEXT,
8      "modal_price(rs)" INTEGER,
9      "temperature(celcius)" REAL,
10     "rainfall(mm)" REAL,
11     "Per_Capita_Income(per_capita_nsdp,rs)" INTEGER,
12     "Food_Subsidy(in_thousand_crores)" REAL,
13     "CPI(base_year2012=100)" REAL,
14     "Elec_Agri_Share(%)" REAL,
15     "MSP(per_quintol)" INTEGER,
16     "Fertilizer_Consumption(kg/ha)" INTEGER,
17     "Area(million_ha)" REAL,
18     "Production(million_tonnes)" REAL,
19     "Yield(kg/ha)" INTEGER,
20     "Export(Million_MT)" REAL,
21     "Import(Million_MT)" REAL,
22     date_str TEXT
23 );
24
25 CREATE TABLE lookup_districts (
26     district TEXT PRIMARY KEY
27 );
28
29 CREATE TABLE lookup_commodities (
30     commodity_name TEXT PRIMARY KEY
31 );

```

Listing 4.1: Database Schema

4.4 Backend Implementation

4.4.1 Flask Application Structure

```

1 btp1/
2 |-- app_stable.py          # Main Flask application
3 |-- models/
4 |   |-- xgboost_improved_model.pkl
5 |   |-- neural_network_model.keras
6 |   |-- nn_model_artifacts.pkl
7 |-- templates/
8 |   |-- index_react.html    # React frontend
9 |-- commodity_prices.db    # SQLite database

```

Listing 4.2: Backend File Structure

4.4.2 API Endpoints

Table 4.2: REST API Endpoints

Endpoint	Method	Description
/	GET	Serve React frontend
/predict	POST	Get price predictions
/get_markets	POST	Get markets for district
/get_varieties	POST	Get varieties for commodity
/health	GET	Health check

4.4.3 Prediction API

```

1 @app.route('/predict', methods=['POST'])
2 def predict():
3     data = request.get_json()
4
5     district = data.get('district')
6     market = data.get('market')
7     commodity = data.get('commodity')
8     variety = data.get('variety')
9     date_str = data.get('date')
10    model_type = data.get('model', 'xgboost')
11
12    predictions = safe_predict(

```

```

13     date_str, district, market,
14     commodity, variety, model_type
15 )
16
17     return jsonify({
18         'success': True,
19         'predictions': predictions,
20         'inputs': {
21             'district': district,
22             'market': market,
23             'commodity': commodity,
24             'variety': variety,
25             'date': date_str,
26             'model': model_type
27         }
28     })

```

Listing 4.3: Prediction Endpoint

4.4.4 Thread Safety

Model predictions use thread locks to prevent race conditions:

```

1 import threading
2
3 model_lock = threading.Lock()
4
5 def safe_predict(...):
6     with model_lock:
7         prediction = model.predict(X)
8     return prediction

```

Listing 4.4: Thread-Safe Prediction

4.5 Frontend Implementation

4.5.1 React Components

The frontend is built using React with functional components and hooks:

```

1 function App() {
2     // State management
3     const [formData, setFormData] = useState({ ... });

```

```

4   const [predictions, setPredictions] = useState([]);
5   const [loading, setLoading] = useState(false);
6
7   // Event handlers
8   const handlePredict = async () => {...};
9   const handleDistrictChange = async (district) => {...};
10
11  // Render
12  return (
13    <div className="app-container">
14      <Header />
15      <ModelSelector />
16      <PredictionForm />
17      <ResultsDisplay />
18      <PriceChart />
19    </div>
20  );
21

```

Listing 4.5: React Component Structure

4.5.2 Key Features

- **Model Selector:** Radio buttons to choose XGBoost or Neural Network
- **Cascading Dropdowns:** District → Market → Commodity → Variety
- **Date Picker:** Select prediction start date
- **7-Day Forecast Display:** Card-based layout for predictions
- **Responsive Design:** Mobile-friendly interface

4.6 Deployment

4.6.1 Production Server

The application uses Waitress WSGI server for production deployment:

```

1 from waitress import serve
2
3 if __name__ == '__main__':
4     print("Starting PRODUCTION server on http://localhost:5000")

```

```
5     serve(app, host='0.0.0.0', port=5000, threads=4)
```

Listing 4.6: Waitress Server Configuration

4.6.2 Model Loading

Models are loaded at application startup with fallback mechanisms:

```
1 # XGBoost Model
2 try:
3     with open('models/xgboost_improved_model.pkl', 'rb') as f:
4         model_data = pickle.load(f)
5         xgb_model = model_data['model']
6 except Exception as e:
7     logger.error(f'Failed to load XGBoost: {e}')
8
9 # Neural Network Model
10 nn_model_paths = [
11     'models/neural_network_model.keras',
12     'models/neural_network_model.h5',
13     'models/nn_best_model.h5'
14 ]
15
16 for path in nn_model_paths:
17     try:
18         nn_model = tf.keras.models.load_model(path)
19         break
20     except:
21         continue
```

Listing 4.7: Model Loading with Fallback

Chapter 5

Results and Analysis

5.1 Model Performance Comparison

5.1.1 Overall Metrics

Table 5.1: Model Performance Comparison

Metric	XGBoost	Neural Network	Best Model
MAE (Rs)	167.42	143.73	Neural Network
RMSE (Rs)	285.36	253.66	Neural Network
MAPE (%)	5.43	4.64	Neural Network
R ² Score	0.9453	0.9601	Neural Network

5.1.2 Accuracy Distribution

Table 5.2: Prediction Accuracy Distribution

Error Threshold	XGBoost	Neural Network
Within 5%	89.2%	92.4%
Within 10%	97.5%	98.1%
Within 15%	99.2%	99.5%

5.2 Commodity-wise Analysis

Table 5.3: Commodity-wise MAPE (%)

Commodity	XGBoost	Neural Network
Rice	5.21	4.42
Jute	5.89	4.95
Wheat	5.45	4.68
Overall	5.43	4.64

5.3 Test Case Validation

We validated our models against actual prices from the database:

Table 5.4: Sample Predictions vs Actual Prices

Commodity	District	Actual	XGBoost	NN
Jute	Malda	Rs 3,600	Rs 3,916	Rs 3,703
Jute	Murshidabad	Rs 3,500	Rs 3,825	Rs 3,575
Rice	Nadia	Rs 3,200	Rs 3,418	Rs 3,362

5.4 Feature Importance Analysis

XGBoost provides feature importance scores:

Table 5.5: Top 10 Important Features (XGBoost)

Feature	Importance Score
commodity_avg_price	0.187
market_avg_price	0.156
MSP	0.134
variety_avg_price	0.098
CPI	0.087
month	0.065
district_encoded	0.054
year	0.048
temperature	0.041
commodity_name_encoded	0.038

5.5 Training Performance

5.5.1 XGBoost Training

- Training Time: 45 minutes (GPU accelerated)
- Best iteration: 847 (with early stopping)
- Final training loss: 0.00012
- Validation loss: 0.00018

5.5.2 Neural Network Training

- Training Time: 15 minutes (CPU mode)
- Epochs completed: 82 (early stopping triggered)
- Best epoch: 67 with val_loss = 0.04832
- Final validation loss: 0.04832
- Test RMSE: 253.66 Rs
- Test MAE: 143.73 Rs
- Test R²: 0.9601
- Test MAPE: 4.64%
- 2025 Prediction MAPE: 4.44%
- 2025 Prediction R²: 0.9724

5.6 Discussion

5.6.1 Neural Network Performance

After retraining with enhanced 2024-2025 data, the Neural Network outperforms XGBoost:

1. **Better Generalization:** The Neural Network achieves lower MAPE (4.64% vs 5.43%) indicating better generalization
2. **Excellent 2025 Predictions:** For 2025 data specifically, the NN achieves remarkable R² = 0.9724 and MAPE = 4.44%

3. **Temporal Pattern Capture:** Deep learning effectively captures seasonal and temporal patterns in price data
4. **Feature Engineering:** 30 carefully engineered features including cyclical encodings for time contribute to accuracy

5.6.2 XGBoost Performance

XGBoost also provides strong performance:

1. **Feature Interactions:** XGBoost naturally captures complex feature interactions through tree structure
2. **Handling of Categorical Variables:** Effective encoding and splitting on categorical features
3. **Regularization:** Built-in L1/L2 regularization prevents overfitting
4. **Fast Training:** GPU-accelerated training enables quick model updates

5.6.3 Model Selection Guidelines

- **Neural Network:** Recommended for production use due to better accuracy
- **XGBoost:** Useful as a fallback model and for interpretability (feature importance)

5.6.4 Limitations Observed

- Both models perform better for Rice (more data) than Wheat (less data)
- Extreme price fluctuations are harder to predict
- Future predictions beyond 7 days show degraded accuracy

Chapter 6

Web Application

6.1 User Interface

The web application provides an intuitive interface for price prediction:

6.1.1 Home Page Features

1. **Header:** Application title and description
2. **Model Selector:** Radio buttons for XGBoost/Neural Network
3. **Input Form:**
 - Commodity dropdown (Rice, Jute, Wheat)
 - District dropdown (18 districts)
 - Market dropdown (cascading based on district)
 - Variety dropdown (cascading based on commodity)
 - Date picker
4. **Predict Button:** Triggers API call
5. **Results Section:** 7-day forecast display

6.1.2 Prediction Results Display

The results are displayed in a card-based layout:

- Current day prediction (highlighted)
- Next 6 days predictions
- Day name and date for each prediction
- Price in Indian Rupees (Rs.)

6.2 User Flow

1. User selects preferred model (XGBoost/Neural Network)
2. User selects commodity from dropdown
3. User selects district (markets load automatically)
4. User selects market
5. Varieties load based on commodity selection
6. User selects variety
7. User picks prediction start date
8. User clicks "Predict" button
9. System displays 7-day price forecast

6.3 Responsive Design

The application is designed to work on:

- Desktop browsers (Chrome, Firefox, Edge, Safari)
- Tablets (iPad, Android tablets)
- Mobile phones (responsive layout)

6.4 Error Handling

The application handles various error scenarios:

- Network errors: Display retry option
- Invalid inputs: Form validation messages
- Server errors: Graceful error display
- Model unavailability: Fallback to available model

Chapter 7

Conclusion and Future Work

7.1 Summary

This BTP project successfully developed a comprehensive commodity price prediction system for West Bengal, India. The key achievements are:

1. **Data Integration:** Created a comprehensive dataset of 173,094 records spanning 11 years (2014-2025), integrating price data with economic and agricultural indicators
2. **Feature Engineering:** Designed 36 meaningful features capturing temporal patterns, market dynamics, and economic factors
3. **Model Development:** Implemented two machine learning models:
 - XGBoost achieving **5.43% MAPE** and **0.9453 R²**
 - Neural Network achieving **4.64% MAPE** and **0.9601 R²**
 - Neural Network 2025 predictions: **4.44% MAPE** and **0.9724 R²**
4. **Web Application:** Developed a user-friendly React-based interface for accessing predictions
5. **Production Deployment:** Deployed using Waitress WSGI server for reliable operation

7.2 Contributions

The main contributions of this work are:

1. A novel multi-source feature set combining economic indicators with agricultural data for price prediction
2. Comparative analysis of gradient boosting vs deep learning for Indian commodity markets

3. A production-ready system that can be used by farmers, traders, and policymakers
4. Comprehensive documentation and reproducible methodology

7.3 Limitations

1. **Geographic Scope:** Limited to West Bengal; may not generalize to other states
2. **Commodity Coverage:** Only three commodities; many important crops not covered
3. **External Factors:** Does not account for sudden policy changes, natural disasters, or global market shocks
4. **Data Dependency:** Requires continuous data updates for accurate predictions

7.4 Future Work

7.4.1 Short-term Improvements

1. **LSTM Integration:** Implement Long Short-Term Memory networks for better temporal pattern capture
2. **Ensemble Methods:** Combine XGBoost and Neural Network predictions
3. **Real-time Data:** Integrate live weather and market data APIs
4. **Mobile Application:** Develop native Android/iOS apps

7.4.2 Long-term Extensions

1. **Pan-India Coverage:** Extend to all Indian states and union territories
2. **More Commodities:** Include vegetables, fruits, and pulses
3. **Price Alerts:** Implement notification system for price thresholds
4. **Market Recommendations:** Suggest best markets for selling based on predicted prices
5. **Supply Chain Integration:** Connect with cold storage and transportation networks

7.5 Final Remarks

This project demonstrates the potential of machine learning in addressing real-world agricultural challenges. The XGBoost model's exceptional accuracy (1.46% MAPE) makes it suitable for practical deployment. The system can help:

- **Farmers:** Make informed decisions about when and where to sell
- **Traders:** Plan procurement and pricing strategies
- **Policymakers:** Monitor market trends and intervene when necessary
- **Researchers:** Build upon this work for further improvements

The code, models, and documentation are available for future development and research purposes.

Bibliography

- [1] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794).
- [2] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780.
- [3] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958.
- [4] Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control*. John Wiley & Sons.
- [5] Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.
- [6] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
- [7] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(6088), 533-536.
- [8] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.
- [9] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training. In *International Conference on Machine Learning* (pp. 448-456).
- [10] Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning* (pp. 807-814).
- [11] Abadi, M., et al. (2016). TensorFlow: A System for Large-scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation* (pp. 265-283).

- [12] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
- [13] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
- [14] Banks, A., & Porcello, E. (2017). *Learning React: Functional Web Development with React and Redux*. O'Reilly Media.
- [15] Government of India. (2023). *Agricultural Statistics at a Glance*. Ministry of Agriculture and Farmers Welfare.

Appendix A

District-wise Market Distribution

Table A.1: Markets per District in West Bengal

District	Number of Markets
Bankura	4
Birbhum	3
Burdwan	5
Coochbehar	2
Darjeeling	2
Hooghly	4
Howrah	2
Jalpaiguri	3
Kolkata	3
Malda	4
Medinipur(E)	4
Medinipur(W)	5
Murshidabad	5
Nadia	4
North 24 Parganas	4
Puruliya	2
South 24 Parganas	3
Uttar Dinajpur	2

Appendix B

Variety Classification

Table B.1: Commodity Varieties

Commodity	Varieties
Rice	Coarse, Common, Fine, Fine(Basmati), H.Y.V., Masuri, Other, Sona, Sona Mansoori Non Basmati, Super Fine
Jute	Ratnachudi (718 5-749), TD-5
Wheat	Kalyan, Sonalika, Common, Other

Appendix C

API Response Format

```
1  {
2      "success": true,
3      "predictions": [
4          {
5              "date": "2018-08-08",
6              "day_name": "Wednesday",
7              "day_offset": 0,
8              "price": 3916.39
9          },
10         {
11             "date": "2018-08-09",
12             "day_name": "Thursday",
13             "day_offset": 1,
14             "price": 3916.33
15         },
16         ...
17     ],
18     "inputs": {
19         "commodity": "Jute",
20         "date": "2018-08-08",
21         "district": "Malda",
22         "market": "Samsi",
23         "model": "xgboost",
24         "variety": "TD-5"
25     }
26 }
```

Listing C.1: Sample API Response

Appendix D

System Requirements

Table D.1: Minimum System Requirements

Component	Requirement
Operating System	Windows 10/11, Linux, macOS
Python	3.10+
RAM	8 GB minimum, 16 GB recommended
Storage	2 GB for models and database
GPU	Optional (CUDA-compatible for training)
Browser	Chrome, Firefox, Edge (latest versions)

Table D.2: Python Dependencies

Package	Version
flask	3.1.0
waitress	latest
xgboost	3.1.2
tensorflow	2.20.0
pandas	2.x
numpy	1.x
scikit-learn	1.x