Artificial Intelligence (CET313)

Intelligent Prototype Development

FreshForecast: Leveraging LSTM Model to Analyze and Predict Kalimati Fruit and Vegetable Market Prices

Shishir Shrestha

B.Sc. (Hons) Computer Systems Engineering

Student ID: 220076720

Email: shishirshrestha@ismt.edu.np

Module Accessor: Dr. Kate MacFarlane

Submitted date: 2023-10-13

Abstract

The Kalimati vegetable and fruit market holds a pivotal role in feeding the bustling Kathmandu Valley, being the major source of fresh produce for its inhabitants. With its influence comes the crucial responsibility of managing the ever-changing prices of its goods, a task significant yet strenuous due to its unpredictable nature. This project materializes from a sincere effort to navigate through these fluctuations, crafting a system that doesn't just analyze past prices but also predicts future ones, aiming to bring a semblance of stability and foresight into the vibrant chaos of daily market exchanges. Through straightforward technology using an LSTM model, this initiative doesn't merely introduce machine learning into the traditional market scenario but does so with a keen eye towards maintaining simplicity in interaction for the vendors and consumers, ensuring that the insights generated are not just accurate, but also easily comprehensible and utilizable by the community it seeks to serve. Thus, it stands not just as a tech solution, but as a mindful bridge between the age-old practices of the market and the new-age technological advancements, designed with, and for, the people of Kalimati Market.

Table of Contents

Table of Figures

1. Introduction

Kalimati, located in the heart of Kathmandu, has a unique place in Nepal's vegetable and fruit supply chain. Known widely as the Kalimati Fruits and Vegetable Market, it's a place that vibrates with lively trades, bargaining voices of sellers and buyers, and the vibrant colors of fresh produce. This market isn't just a trading place – it's a crucial component for the livelihoods of many and the dietary needs of the populous Kathmandu Valley. It's estimated that a whopping 60-70% of the whole valley's demand for fresh vegetables and fruits is met by this bustling market (Department of Agriculture, 2022).

Digging a bit deeper, the situation in Kalimati is not just greens and fruits. Behind the scenes, there's a complicated story of fluctuating prices that play with the everyday lives of both sellers and buyers. Some days, the prices shoot up, making vegetables and fruits somewhat a luxury for common people. On other days, a sudden drop in prices results in losses for the hardworking farmers and traders. So, it's like a see-saw that keeps balancing between profit and loss, affordability, and expense. This price irregularity isn't only a local phenomenon. According to studies, agricultural price unpredictability is a global challenge that affects all, from individual farmers to the national economy (Anjan *et. al*, 2016).

In a bid to find a technological solution, this project tries to peek into the future of vegetable prices using past data. Here, a system, crafted with machine learning, notably the LSTM model, analyzes past price data to predict future prices. Picture this: before a farmer sends his tomatoes to the market, he already has a pretty good idea about the price it would fetch. Not only farmers but also traders, and common people could plan and act more accurately, balancing the scale between demand and supply more effectively. This not just assures more stability in the market but also ensures that every stakeholder, from farmer to consumer, gets a fair deal. So, through this endeavor, a bridge between technology and traditional market practices is aimed to be built, ensuring a smoother, more predictable trading environment in the throbbing heart of Kathmandu's vegetable and fruit trade.

2. Aims & Objectives

I started this project because I wanted to find a way to guess the future prices of veggies and other goods at the Kalimati market. Prices go up and down a lot, and I thought it would be cool if a computer could look at past prices and tell us what might happen next. So, my big goal was to make an AI model that could learn from old prices and give us smart guesses about future ones. I hoped this would help people think about what to buy and sell in the market, making it a bit easier to plan ahead. I have noticed that the prices of things change a lot and thought about how great it would be if we could know what the prices might be in the future. It would be super helpful for everyone - folks buying things and also the people selling them. So, I thought why not try to make something that can guess what these future prices might be by looking at what prices were like before? That way, people can think and decide about what to buy or sell beforehand. It's like giving them a little sneak peek into what might happen next in the market.

The main objectives of my project are as follows:

1. Gather and Fix the Data:

- Find a good bunch of data about past agricultural product prices.
- Clean it up and organize it so the computer can understand and learn from it.

2. Build the Model:

- Use the cleaned-up data to build a model.
- Make sure the model can learn from the data about how prices change.

3. Testing the Model:

- Check how good the model is by guessing prices.
- Use some data that the model has never seen before and see how close its guesses are to the real prices.

4. Create a Web App:

- Make a friendly web app where people can use the model to see future price guesses.
- Ensure that the app is easy for people to use and understand.

5. Check and Think Again:

- Look closely at how the model and app are working.
- Find out where things might be going wrong or could be done better.

4. Overview: e-Portfolio

5. Prototype Identification and Planning

For predicting agricultural prices, a few models like Linear Regression, Decision Trees, and ARIMA naturally presented themselves as potential candidates given their historical usage in financial and market forecasting. However, I leaned towards utilizing Long Short-Term Memory (LSTM) networks for my prototype. Why LSTMs? Primarily, their capability for "remembering" and leveraging past data patterns in predicting future points was fascinating, especially considering the sequential nature of price data. This capacity to recall and utilize past information efficiently aids LSTMs in accurately predicting upcoming prices based on observed historical patterns. Additionally, their proven reliability in similar tasks, such as stock price predictions where they skillfully navigate through time-series data, reassured me of their aptitude in efficiently predicting agricultural prices in our context. So, the LSTM model became the selected tool in my arsenal for developing a potent price prediction prototype.

5.1 Literature Review

a) The Rising Need for Predictive Models in Trading

In the global world of trading, prices change like the wind. For farmers, traders, and customers, knowing the future price can be a game-changer. Artificial Intelligence (AI) comes into play here. AI can look at old data and guess future prices. One such AI tool is the Long Short-Term Memory (LSTM) network. LSTM is like a smart brain that remembers patterns and can predict future outcomes based on them. It has been especially useful for looking at price changes (Hochreiter & Schmidhuber, 1997).

b) Looking at Current Models and Their Issues

Many researchers have used LSTM to predict prices. In a study, LSTM was used to guess cocoa prices. This model was good at making general predictions. However, it struggled when sudden, unexpected changes happened in the market (Sheth & Shah, 2023). Another study in India tried to guess the daily price of onions using LSTM. This model worked Ill on most days but had issues when there were sudden changes in government rules or unexpected events (Kumar & Thenmozhi, 2019).

From these examples, one thing becomes clear: while LSTM models are smart, they can sometimes get confused by sudden, big changes in the market. They are like a weatherman who can tell you it's going to rain based on old data but can't always predict a sudden storm.

c) Why LSTM is a Good Fit for Kalimati Market's Needs

For the bustling Kalimati Market in Nepal, a special kind of model is needed. Our idea is to make an LSTM model that doesn't just look at the past but is also ready for unexpected events. To do this, I have added data about things like Iather, holidays, and government rules. With this data, our model can better guess the future prices in the Kalimati Market.

Another important point is that LSTM is a proven tool. Many studies have shown that LSTM can make reliable guesses for different things. For a place like the Kalimati Market, which has so many things affecting prices, an LSTM model is a good fit.

d) My Model: Learning from the Past and Planning for the Future

Taking lessons from other LSTM models, this prototype has a special feature. It is designed to handle the surprises of the market better. By adding more types of data and using advanced AI techniques, our model can make even better predictions for the Kalimati Market.

In the end, our model's goal is simple: to help the people of the Kalimati Market by giving them a tool to guess future prices. With better predictions, traders can plan better, farmers can get fair prices, and customers can save money.

The world of trading is full of ups and downs. Prices can change quickly and without warning. But with tools like LSTM, I can be better prepared for the future. By looking at old data and planning for unexpected events, LSTM models can make reliable guesses about future prices. For the Kalimati Market in Nepal, our prototype offers a way to face the future with more confidence.

5.2 Reflection on the Prototype Identification

Embarking on this journey of prototype identification and planning has been a meticulous blend of challenges and revelations. Delving into the vast world of AI and LSTM models, it became evident that while there is a plethora of research and practical applications available, each model carried its unique adaptations and faced different hurdles depending upon the specific use-case scenarios. The academic exploration of existing models unearthed the subtleties and technical nuances that played pivotal roles in shaping the prototype for the Kalimati Market.

The literature profoundly emphasized the pivotal role of accurate data and the challenges related to external variable management in predictive models. Practically, during the preliminary phases of prototype planning and data analysis, it became apparent that the model must be attuned not only to the historical pricing data but also to the myriad of variables that could potentially impact market pricing, such as Iather conditions, festive seasons, and political factors, which was a significant realization.

Ensuring that the model remains agile, adaptive, and precise while handling real-world data and scenarios was a key learning. As I proceed, these learnings and reflections not only serve as guideposts but also as a reminder that theory and practice can beautifully converge when informed by thoughtful research and reflection.

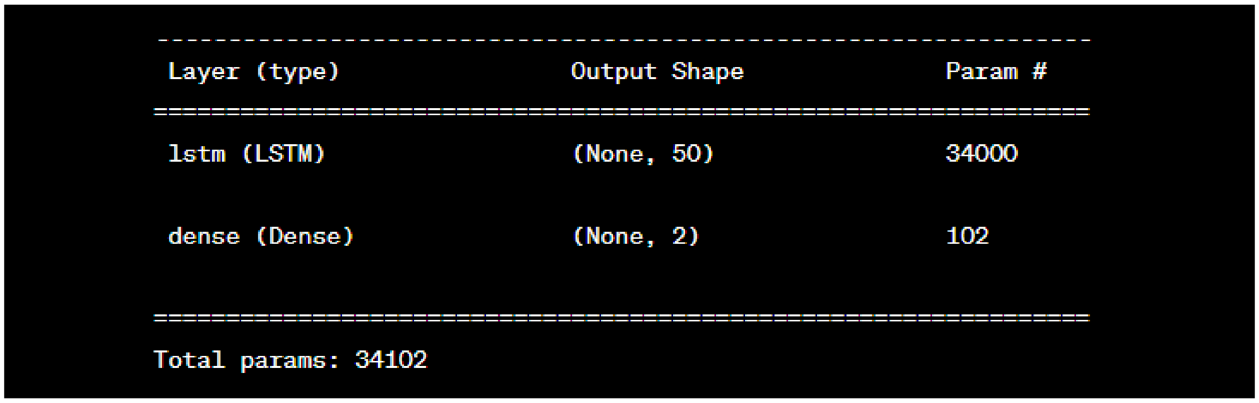
6. Prototype Development

Firstly, I worked on getting our data ready. I used Pandas to load our data. Then, I changed some text columns into a format that our model can understand using one-hot encoding, and also extracted year, month, and day from the date. After cleaning the data and removing any missing values, I used Scikit-learn to scale our data so our model can easily learn from it. Next, I built our LSTM model using Keras, giving it 50 neurons and training it for 50 epochs (rounds) to learn the patterns in our data.

6.1 Model Development

In simple terms, I chose the LSTM model because it’s really good at understanding patterns over time, which is perfect for predicting prices that change every day. Imagine trying to remember a pattern of moving objects - the LSTM model does this but with numbers in the dataset.

Here's a little look into the model's structure:



```
 _________________________________________________________________
  Layer (type)                Output Shape              Param #
=================================================================
  lstm (LSTM)                 (None, 50)                34000

  dense (Dense)               (None, 2)                 102

=================================================================
Total params: 34102
```

Figure 1 - Model Structure

The LSTM layer with 50 neurons keeps track of the patterns it sees in the data, and the dense layer at the end gives us our two prediction values (minimum and maximum prices).

During training (over 50 epochs), the model tried to understand the patterns in the price changes, adjusting itself to predict prices as closely as possible to the actual future prices. After the model was trained, I tested it by asking it to predict prices and compared these to the actual prices.

I measured its accuracy using MAE and MSE, and the R^2 score helped understand how well the model's predictions match the actual prices. And the results were overwhelming.

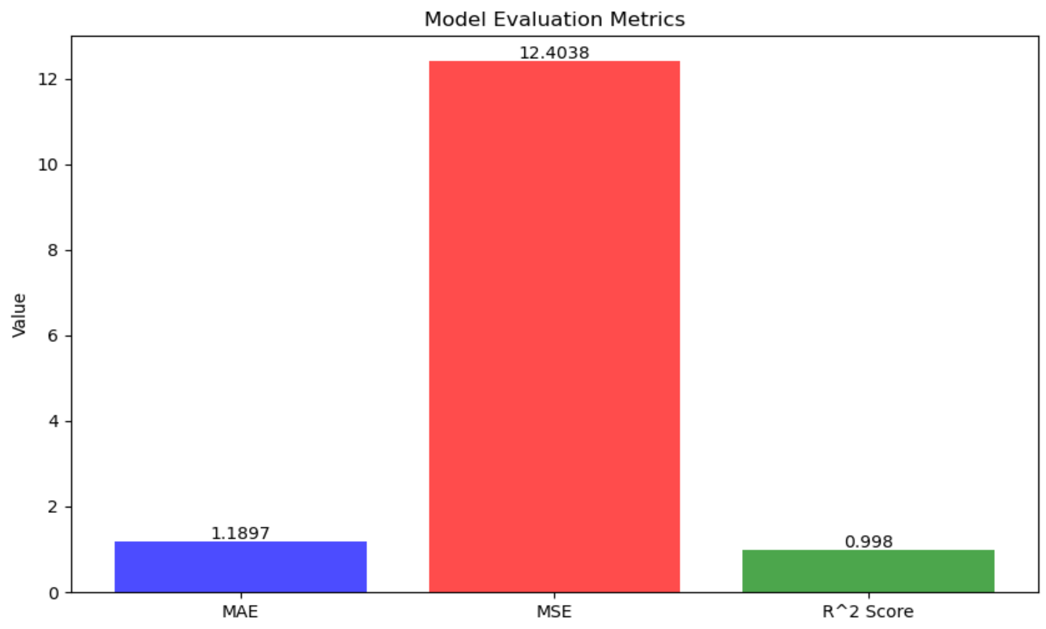


Figure 2 - Model Evaluation Metrics Graph

The figure above provides a glimpse into the numerical proficiency of the model in predicting prices. MAE (Mean Absolute Error) sits at a petite 1.19, indicating that on average, the model's predictions are approximately 1.19 units away from the actual prices. Whereas MSE (Mean Squared Error), penalizing larger errors, reflects a value of 12.40. Significantly, the R² Score soars at an impressive 0.998, hinting that the model can explain approximately 99.8% of the variation in the observed data, showcasing a robust predictive ability. These metrics cohesively signal not only the reliability but also the accuracy of the model in foreseeing future price trajectories, thus substantiating its applicability in practical scenarios. Looking at the R2 Score, our model has a very high predictive accuracy.

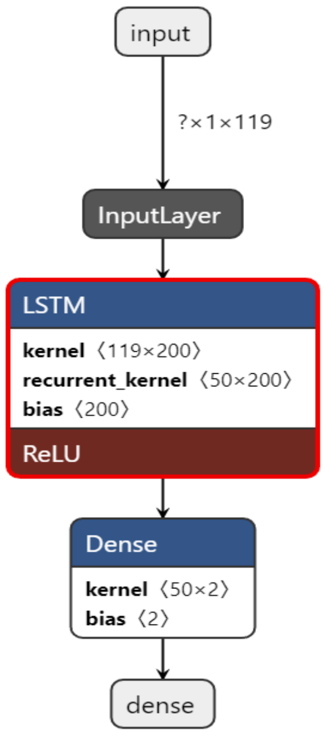


Figure 3 - Model Architecture

To visualize the training process, I plotted the model's training and validation loss, which helped to see how Ill it's learning from the data over time. The visualization is shown in Figure 4 below.

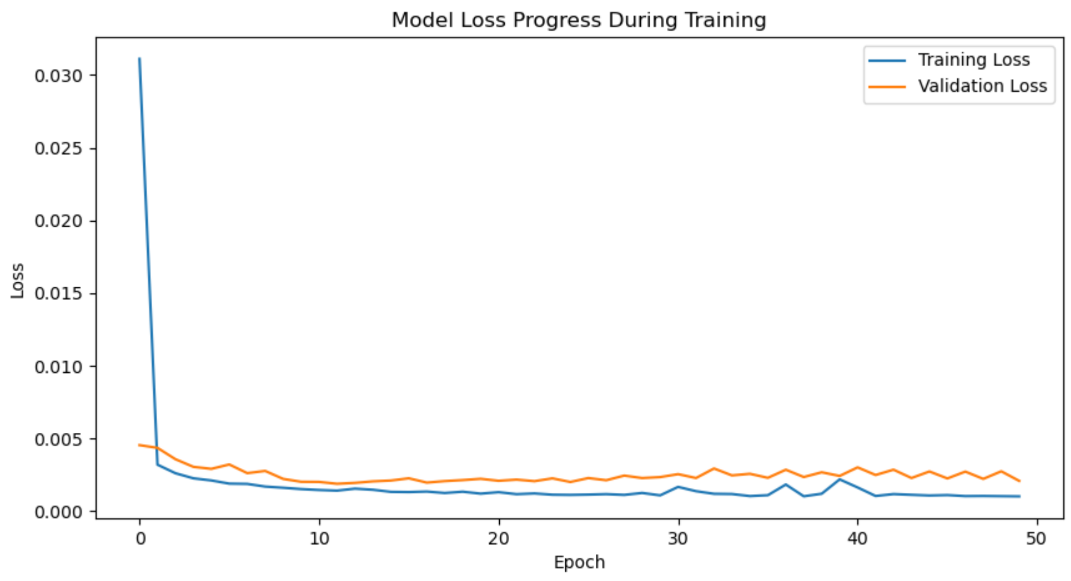


Figure 4 - Training vs. Validation Loss

In the graph in Figure 5, there are two lines: one shows the actual prices of the commodity from 2013 to 2021, and the other shows the prices our model predicted for the same time. The actual prices line goes up and down a bit, following the real market changes in those years. The predicted prices line, on the other hand, is a little bit above the actual prices line most of the time. This means our model thought prices would be a bit higher than they were in reality. But it's not too far off - the predicted line still follows the same general path as the actual prices, going up and down at the same time, just a little higher. This shows our model is doing a good job of seeing the general price trends but might be overestimating the prices a bit. So, it's quite close and gives us useful predictions, but it's not perfect and there's a bit of room to make it even better in the future.

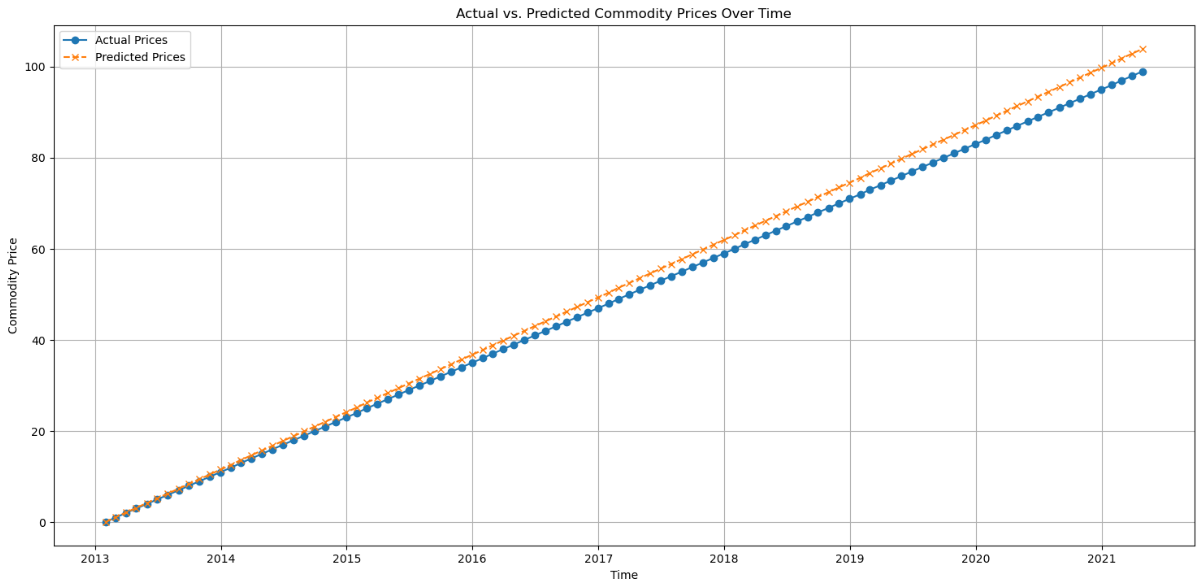
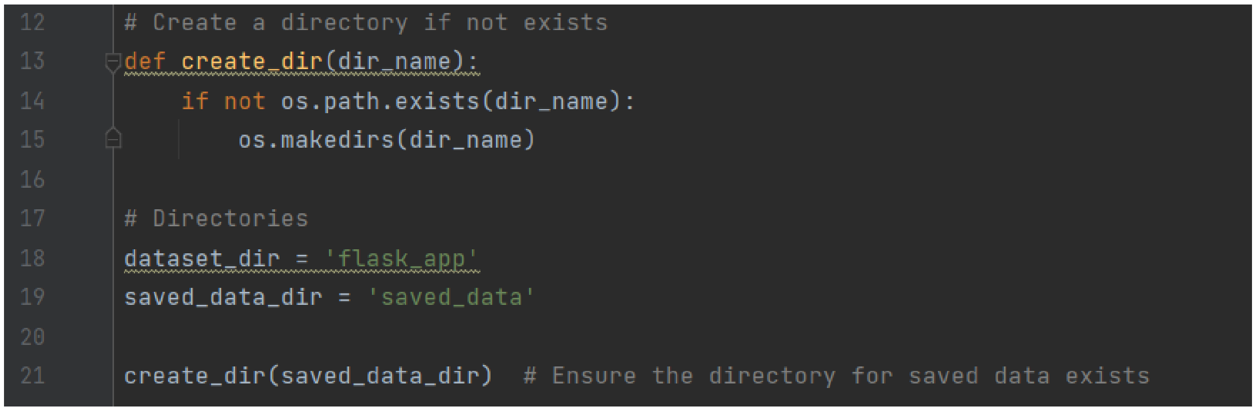


Figure 5 - Actual vs. Predicted Price Range

6.1.1 Code Explanations and Screenshots

a) Creating Directories and Loading Data:

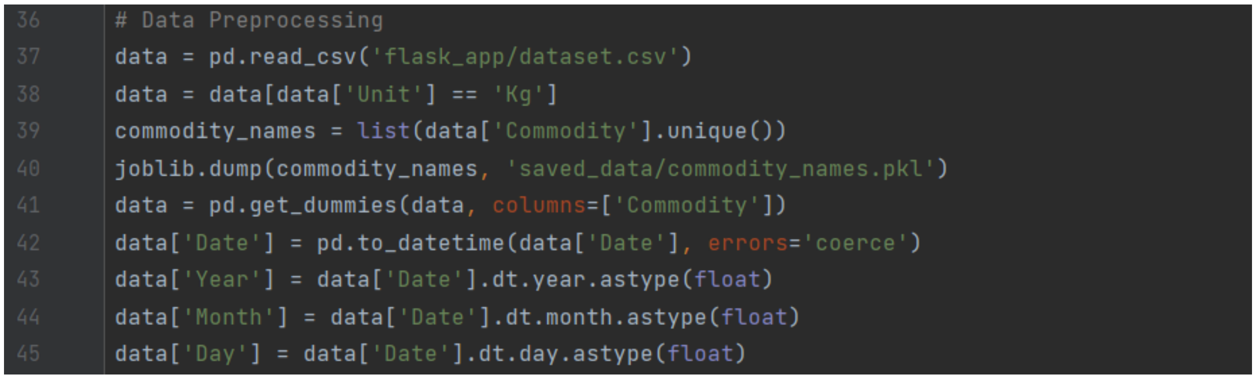


```python
# Create a directory if not exists
def create_dir(dir_name):
    if not os.path.exists(dir_name):
        os.makedirs(dir_name)

# Directories
dataset_dir = 'flask_app'
saved_data_dir = 'saved_data'

create_dir(saved_data_dir)  # Ensure the directory for saved data exists
```

Figure 6 - Code Screenshot: Loading data, Creating Directory

Here, a function create_dir is defined to make new directories if they don't already exist. Two directories, flask_app and saved_data, are used to manage dataset and saved model/data respectively.

b) Data Preprocessing:

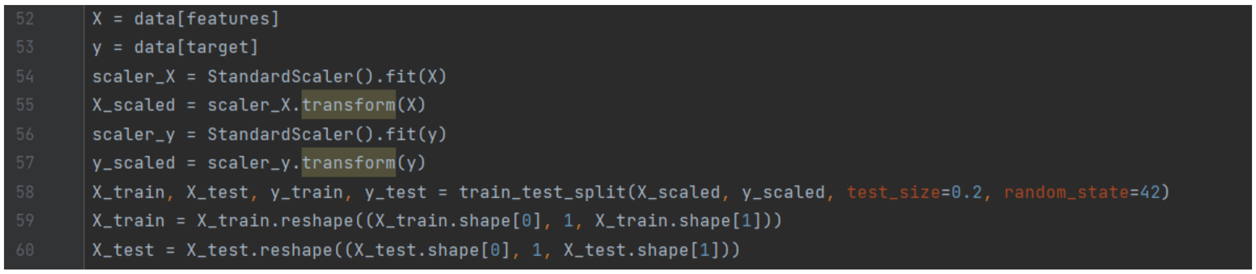


```python
# Data Preprocessing
data = pd.read_csv('flask_app/dataset.csv')
data = data[data['Unit'] == 'Kg']
commodity_names = list(data['Commodity'].unique())
joblib.dump(commodity_names, 'saved_data/commodity_names.pkl')
data = pd.get_dummies(data, columns=['Commodity'])
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')
data['Year'] = data['Date'].dt.year.astype(float)
data['Month'] = data['Date'].dt.month.astype(float)
data['Day'] = data['Date'].dt.day.astype(float)
```

Figure 7 - Code Screenshot: Data Preprocessing

The data is loaded and filtered to only use rows where the unit is 'Kg'. Unique commodity names are saved using joblib. get_dummies is used to convert categorical variable (Commodity) into dummy/indicator variables.

c) Data splitting and Scaling:

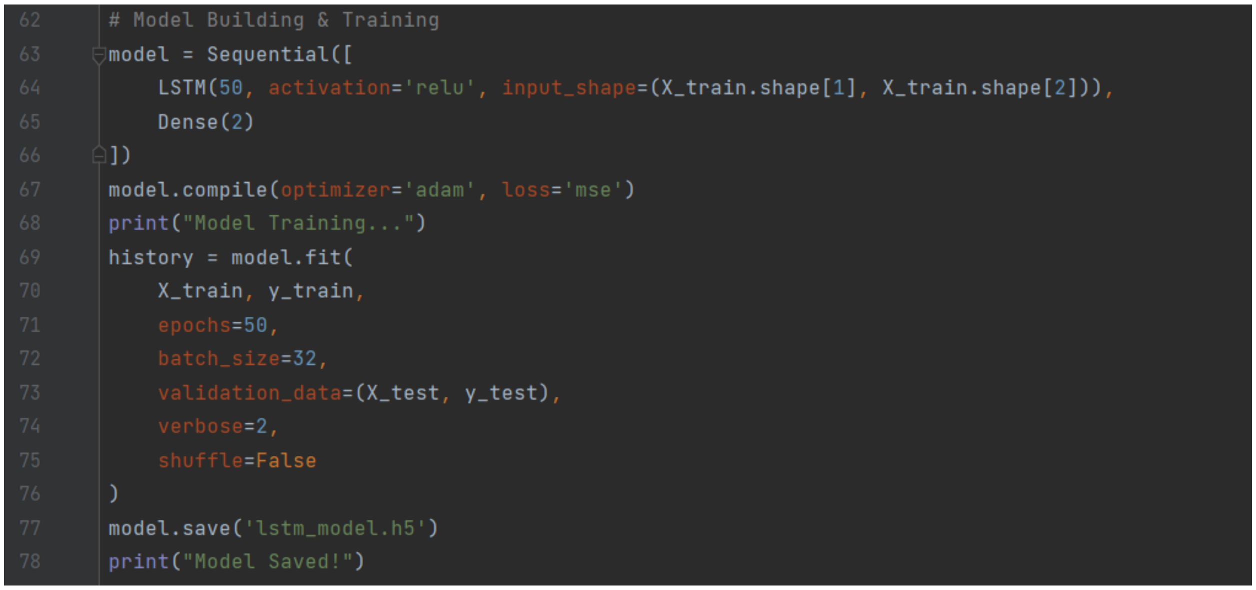


```
X = data[features]
y = data[target]
scaler_X = StandardScaler().fit(X)
X_scaled = scaler_X.transform(X)
scaler_y = StandardScaler().fit(y)
y_scaled = scaler_y.transform(y)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

Figure 8 - Code Screenshot: Data Splitting, Scaling

Data features (X) and targets (y) are scaled using StandardScaler and split into training and test datasets using train_test_split with 80% of the data used for training and 20% used for testing.

d) Model Building, Training, and Saving:

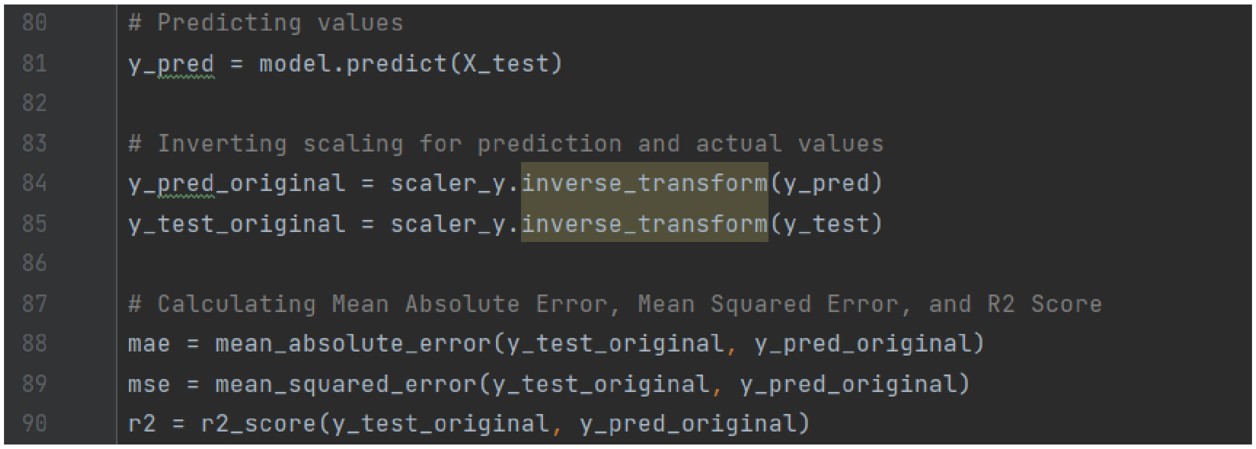


```
# Model Building & Training
model = Sequential([
    LSTM(50, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(2)
])
model.compile(optimizer='adam', loss='mse')
print("Model Training...")
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test),
    verbose=2,
    shuffle=False
)
model.save('lstm_model.h5')
print("Model Saved!")
```

Figure 9 - Code Screenshot: Model Build, Train, Save

A Sequential model with LSTM and Dense layers is created and compiled with the ‘adam’ optimizer and mean squared error (‘MSE’) as the loss function. The model is trained for 50 epochs and then saved.

e) Predicting and Evaluating the Model:

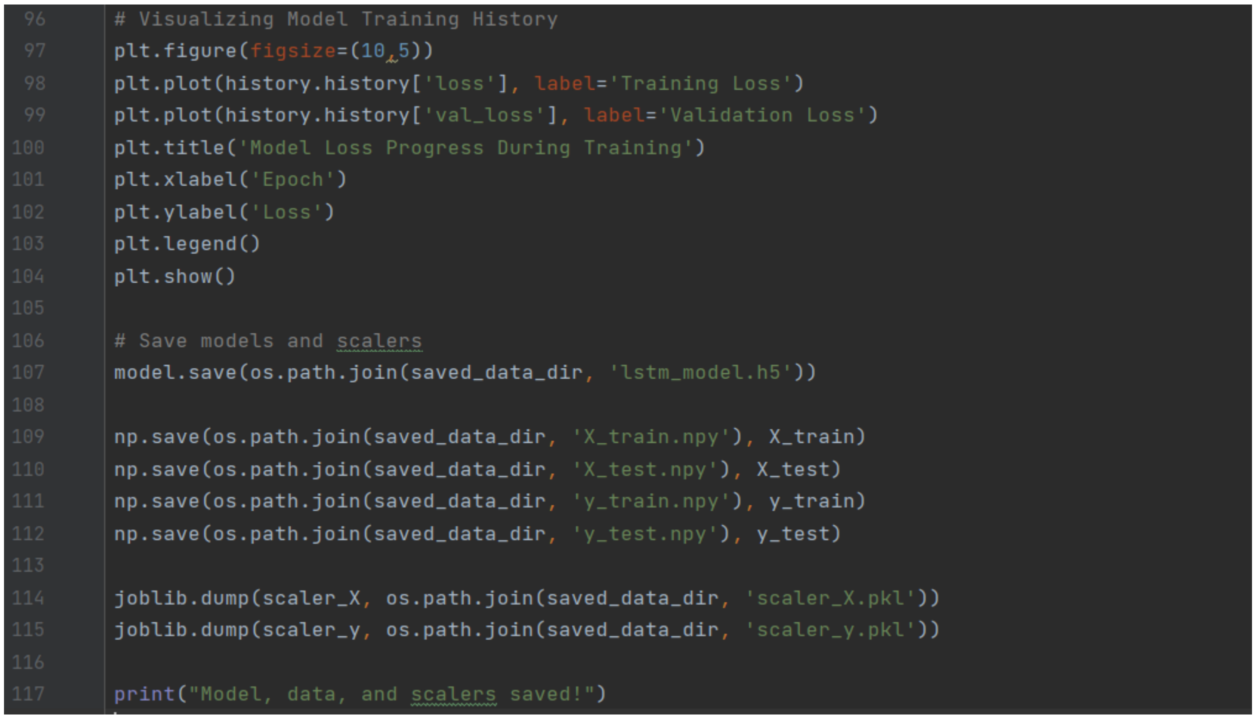


```python
# Predicting values
y_pred = model.predict(X_test)

# Inverting scaling for prediction and actual values
y_pred_original = scaler_y.inverse_transform(y_pred)
y_test_original = scaler_y.inverse_transform(y_test)

# Calculating Mean Absolute Error, Mean Squared Error, and R2 Score
mae = mean_absolute_error(y_test_original, y_pred_original)
mse = mean_squared_error(y_test_original, y_pred_original)
r2 = r2_score(y_test_original, y_pred_original)
```

Figure 10 - Code Screenshot: Predict, Evaluate Model

The model predicts the test data (X_test) and evaluates these predictions using three metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and R2 Score.

f) Visualizing Training History and Saving Data:



```python
# Visualizing Model Training History
plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss Progress During Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Save models and scalers
model.save(os.path.join(saved_data_dir, 'lstm_model.h5'))

np.save(os.path.join(saved_data_dir, 'X_train.npy'), X_train)
np.save(os.path.join(saved_data_dir, 'X_test.npy'), X_test)
np.save(os.path.join(saved_data_dir, 'y_train.npy'), y_train)
np.save(os.path.join(saved_data_dir, 'y_test.npy'), y_test)

joblib.dump(scaler_X, os.path.join(saved_data_dir, 'scaler_X.pkl'))
joblib.dump(scaler_y, os.path.join(saved_data_dir, 'scaler_y.pkl'))

print("Model, data, and scalers saved!")
```

Figure 11 - Code Screenshot: Visualize Training, Data Saving

The training and validation loss of the model across epochs are plotted for visualization. The trained model is saved for future use. Scalers, training and testing data are also saved for future.

6.2 Web-App Development

I have designed a simple yet functional web-app, utilizing a Flask backend to communicate with the LSTM model and serve the user's needs seamlessly.

a) Analyze Functionality:

Upon entering the application, users can navigate to the "Analyze" page from the dashboard. Here, users can select or search for a specific commodity, whereby a graphical representation of historical pricing is displayed. The commodity names are fetched from our dataset and displayed through an API call in the Flask app to ensure real-time and accurate information retrieval.

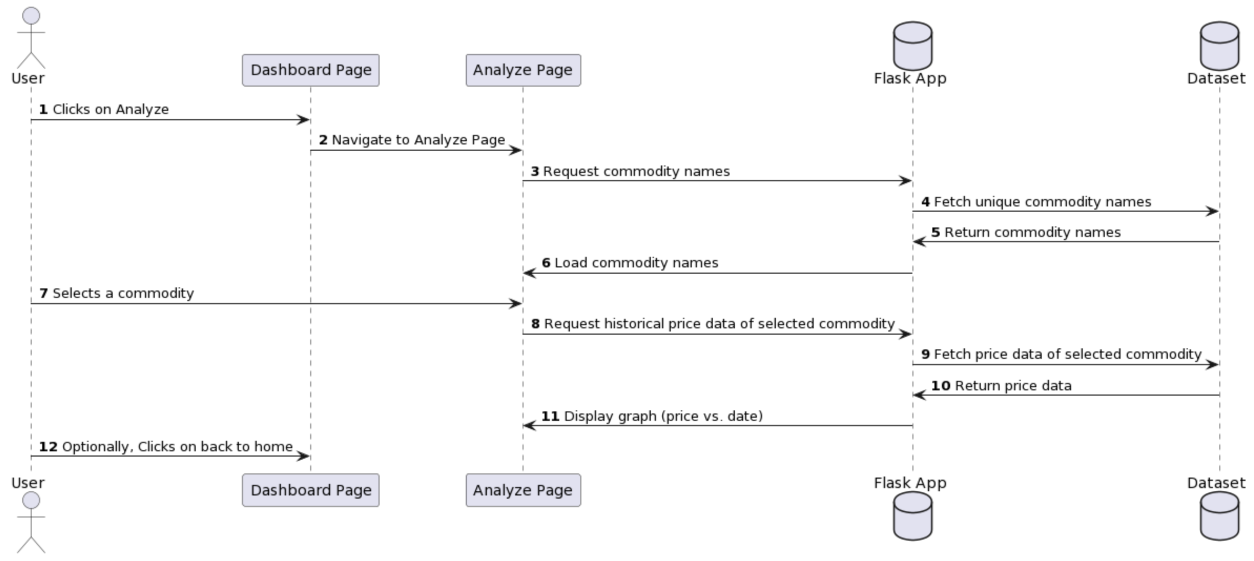

User
Dashboard Page
Analyze Page
Flask App
Dataset
1 Clicks on Analyze
2 Navigate to Analyze Page
3 Request commodity names
4 Fetch unique commodity names
5 Return commodity names
6 Load commodity names
7 Selects a commodity
8 Request historical price data of selected commodity
9 Fetch price data of selected commodity
10 Return price data
11 Display graph (price vs. date)
12 Optionally, Clicks on back to home

Figure 12 - UML Diagram of Analyze Functionality

b) Predict Functionality:

In contrast, the "Predict" functionality allows users to foresee the potential future prices of a selected commodity. From the dashboard, upon selecting "Predict", users are navigated to a new page where they can select a commodity and a future date. After making these selections and initiating the prediction, our LSTM model computes the expected maximum and minimum prices, which are then promptly displayed. These predictions are powered by a trained LSTM model that utilizes historical data to forecast future prices, embodying a crucial tool for strategic planning and decision-making in agribusiness.

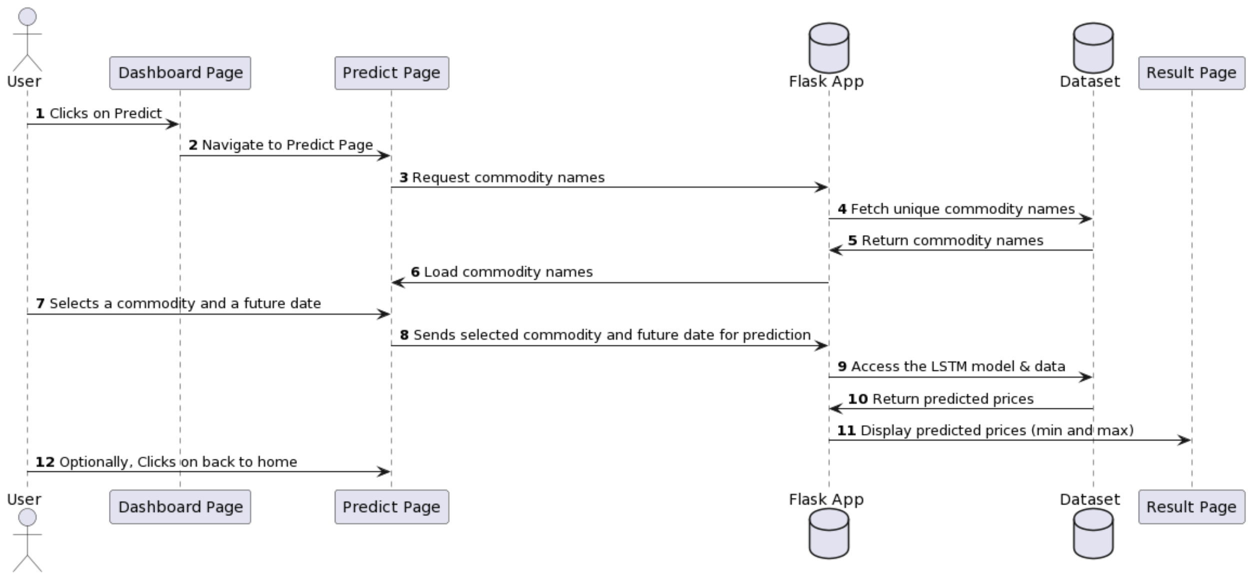
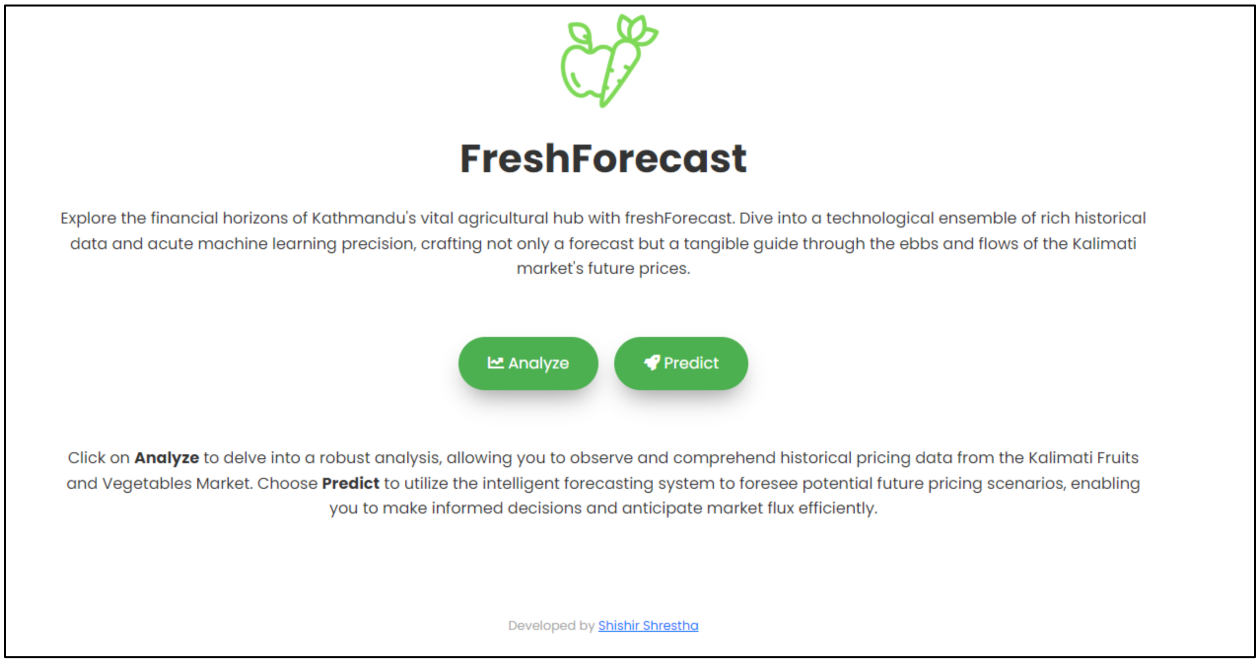

User
Dashboard Page
Predict Page
Flask App
Dataset
Result Page
1 Clicks on Predict
2 Navigate to Predict Page
3 Request commodity names
4 Fetch unique commodity names
5 Return commodity names
6 Load commodity names
7 Selects a commodity and a future date
8 Sends selected commodity and future date for prediction
9 Access the LSTM model & data
10 Return predicted prices
11 Display predicted prices (min and max)
12 Optionally, Clicks on back to home

Figure 13 - UML Diagram of Predict Functionality

6.3 Web-App Screenshots

I've put some pictures of different parts of the app. It's simple to use and does exactly what I need it to do: help users analyze and predict prices easily.

a) Home page:



FreshForecast

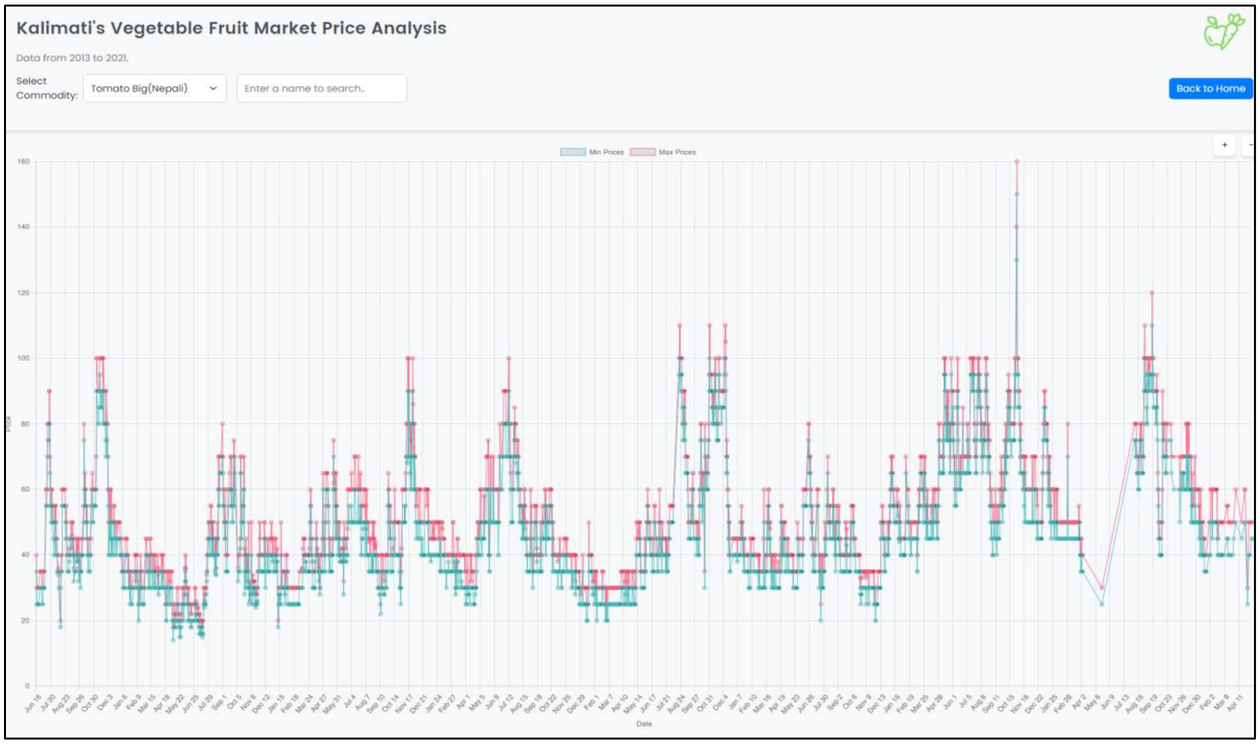
Explore the financial horizons of Kathmandu's vital agricultural hub with freshForecast. Dive into a technological ensemble of rich historical data and acute machine learning precision, crafting not only a forecast but a tangible guide through the ebbs and flows of the Kalimati market's future prices.

Click on **Analyze** to delve into a robust analysis, allowing you to observe and comprehend historical pricing data from the Kalimati Fruits and Vegetables Market. Choose **Predict** to utilize the intelligent forecasting system to foresee potential future pricing scenarios, enabling you to make informed decisions and anticipate market flux efficiently.

Developed by Shishir Shrestha

Figure 14 - Home Page of FreshForecast

b) Analyze page:


Kalimati's Vegetable Fruit Market Price Analysis

Data from 2013 to 2021.

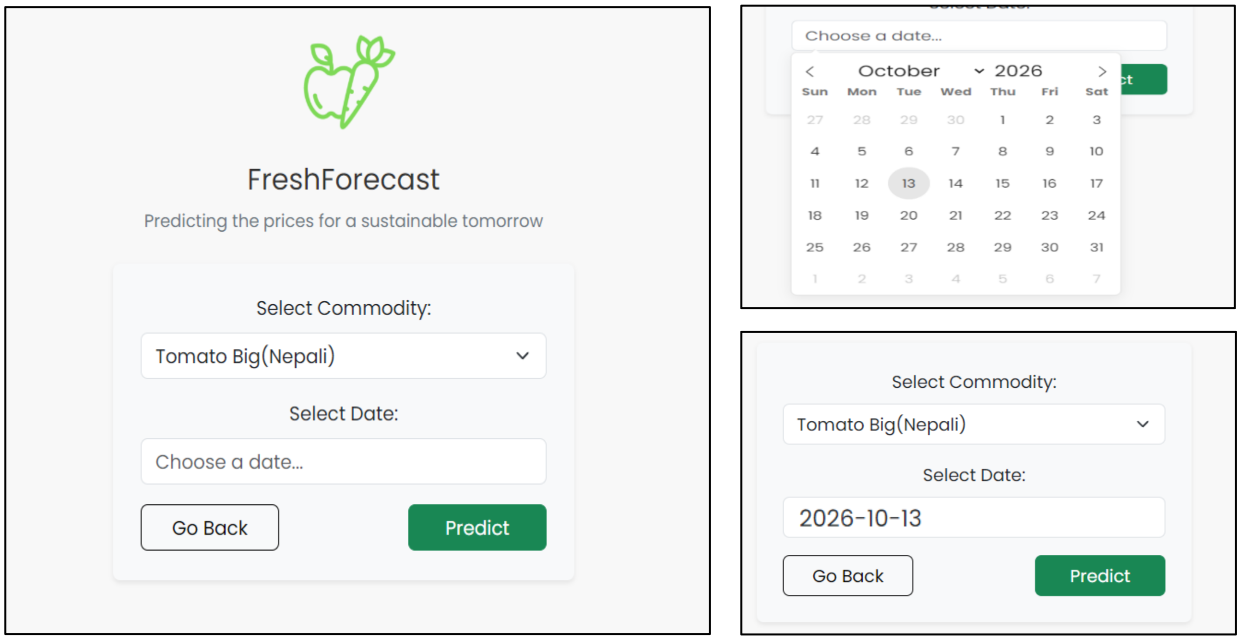
Select Commodity: Tomato Big(Nepali)

Enter a name to search..

Back to Home

Figure 15 - Analysis in FreshForecast

c) Predict page:


FreshForecast

Predicting the prices for a sustainable tomorrow

Select Commodity: Tomato Big(Nepali)

Select Date: Choose a date...

Go Back

Predict

Choose a date...

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

October 2026

Select Commodity: Tomato Big(Nepali)

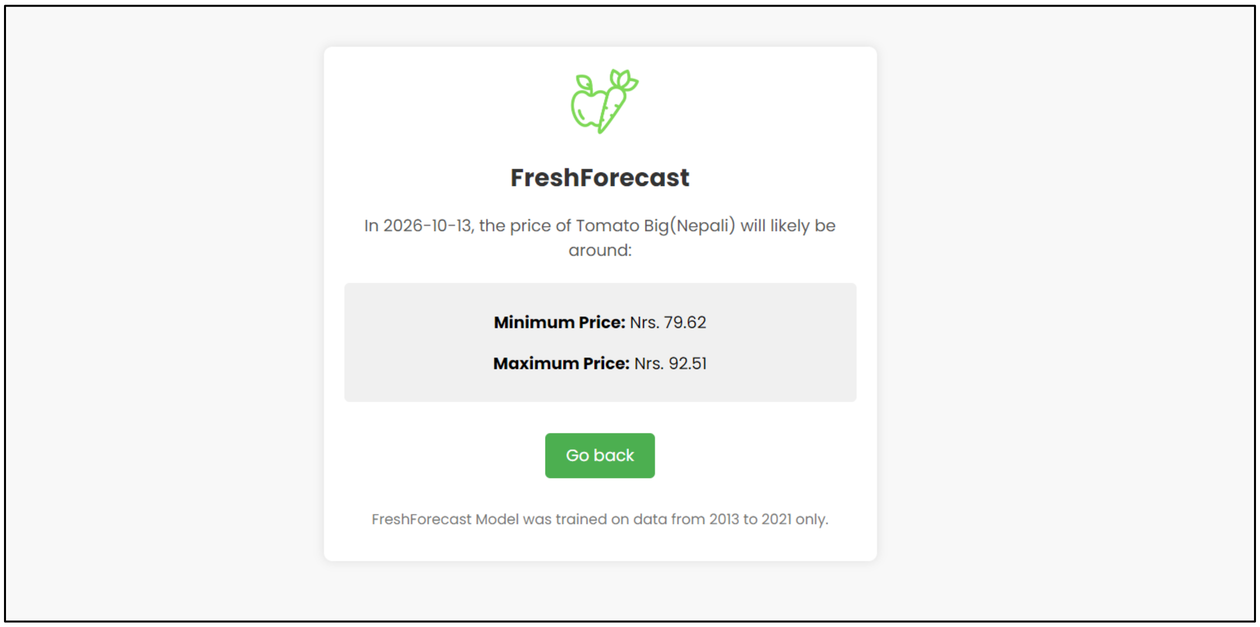
Select Date: 2026-10-13

Go Back

Predict

Figure 16 - Prediction Screen in FreshForecast

d) Prediction result page:


FreshForecast

In 2026-10-13, the price of Tomato Big(Nepali) will likely be around:

Minimum Price: Nrs. 79.62

Maximum Price: Nrs. 92.51

FreshForecast Model was trained on data from 2013 to 2021 only.

Figure 17 - Prediction Result in FreshForecast

After shaping up the web app and embedding our model into it, it's crucial to glance back and ponder upon how things roll when users interact with it. Does it serve accurate predictions? Is it user-friendly? Navigating through these questions and the application's live functionality has given a tangible aspect to all the coding and data crunching that was done behind the scenes. The following Evaluation section dives deeper into measuring, analyzing, and reflecting upon the performance of our model and web app, to discern how close we got to what we aimed for and to uncover spaces where there's room for a bit more finesse. Let's sift through it!

7. Evaluation

At the start, our main task was to make our data ready for the model. The data was a bit messy at first. I only wanted to look at commodities measured in 'Kg', so I filtered the data to keep those parts only. I also made some changes like turning the 'Commodity' names into a format the model can understand using encoding and pulling out the year, month, and day from the date.

7.1 Train/Test Split:

I split our data into two parts: one part for training the model (80% of the data) and the other part to test it (20% of the data). This way, I could confidently check how well our model is doing because it has never seen the test data before.

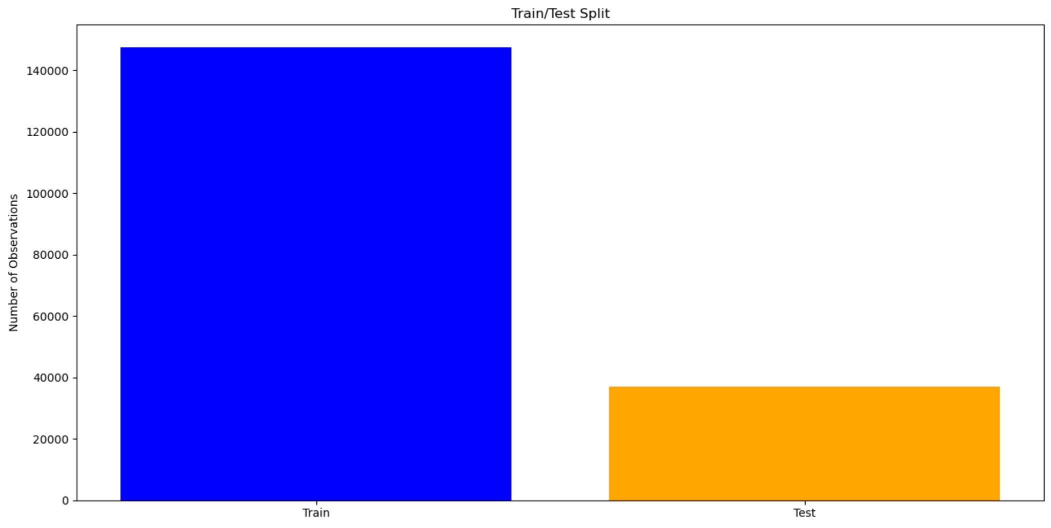


Figure 18 - Train/Test Data Split

The numbers in our data were all over the place, some big, some small. To help our model learn better, I used scalers to squeeze these numbers into a smaller, standard range. Imagine you are trying to compare the weight of an elephant and a mouse - scaling helps us make these big differences smaller, so it's easier to compare and learn from them.

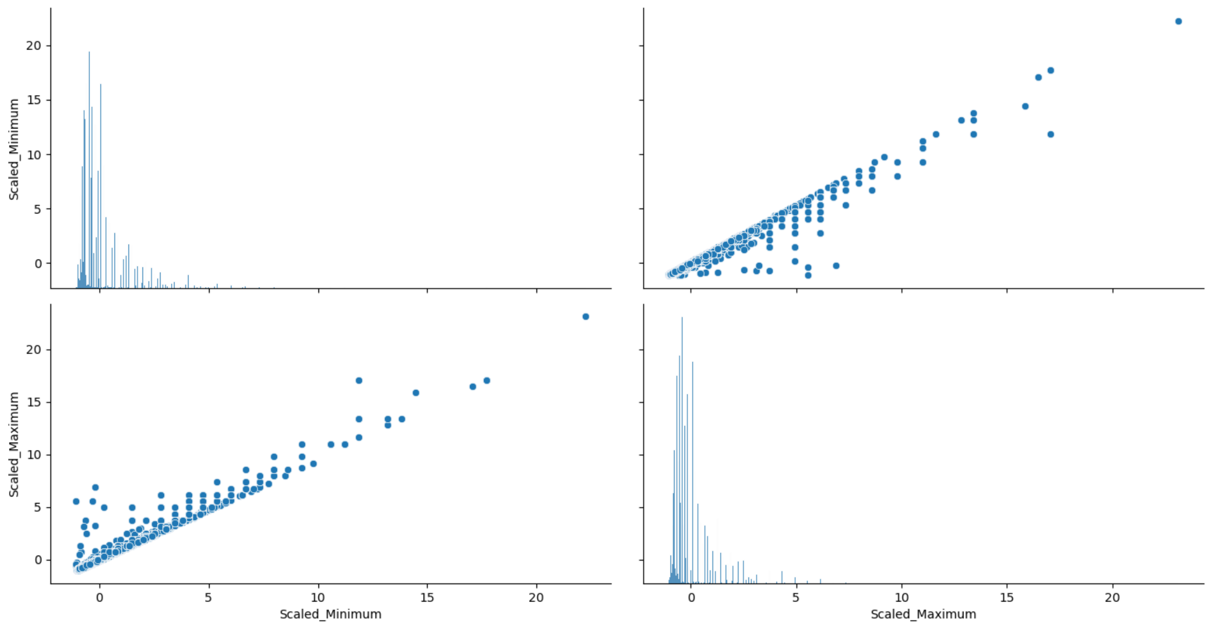
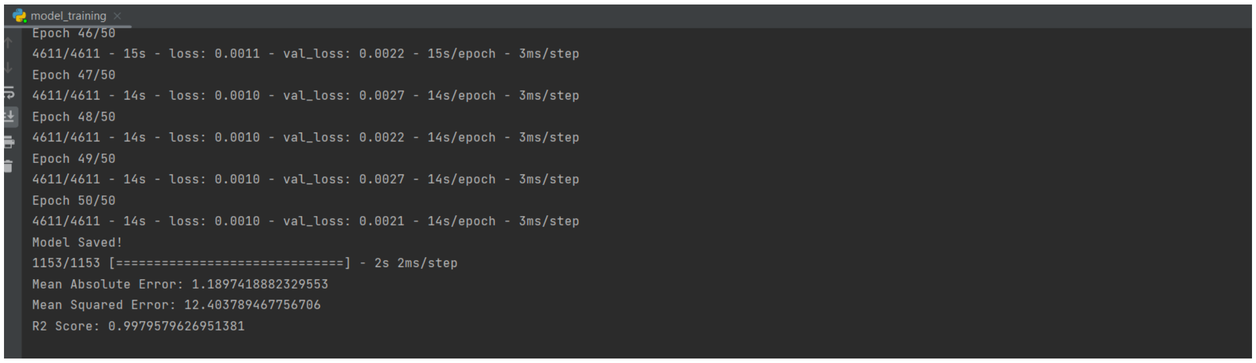


Figure 19 - Pair plot of Trained Scalers

7.2 Test Results

When I tested our model, I saw some interesting things. Our model did really well in some parts, with an R2 Score of 0.9979 (this score is like a grade in school, but here, 1 is perfect). However, the Mean Absolute Error (MAE) and Mean Squared Error (MSE) showed that the model was a bit off in some of its guesses, with MAE being 1.1897 and MSE being 12.4037. So, our model isn't perfect and does make some mistakes, but it's doing really well overall.



```
Epoch 46/50
4611/4611 - 15s - loss: 0.0011 - val_loss: 0.0022 - 15s/epoch - 3ms/step
Epoch 47/50
4611/4611 - 14s - loss: 0.0010 - val_loss: 0.0027 - 14s/epoch - 3ms/step
Epoch 48/50
4611/4611 - 14s - loss: 0.0010 - val_loss: 0.0022 - 14s/epoch - 3ms/step
Epoch 49/50
4611/4611 - 14s - loss: 0.0010 - val_loss: 0.0027 - 14s/epoch - 3ms/step
Epoch 50/50
4611/4611 - 14s - loss: 0.0010 - val_loss: 0.0021 - 14s/epoch - 3ms/step
Model Saved!
1153/1153 [==============================] - 2s 2ms/step
Mean Absolute Error: 1.1897418882329553
Mean Squared Error: 12.403789467756706
R2 Score: 0.9979579626951381
```

Figure 20 - Console Output of Trained Model's Metrics

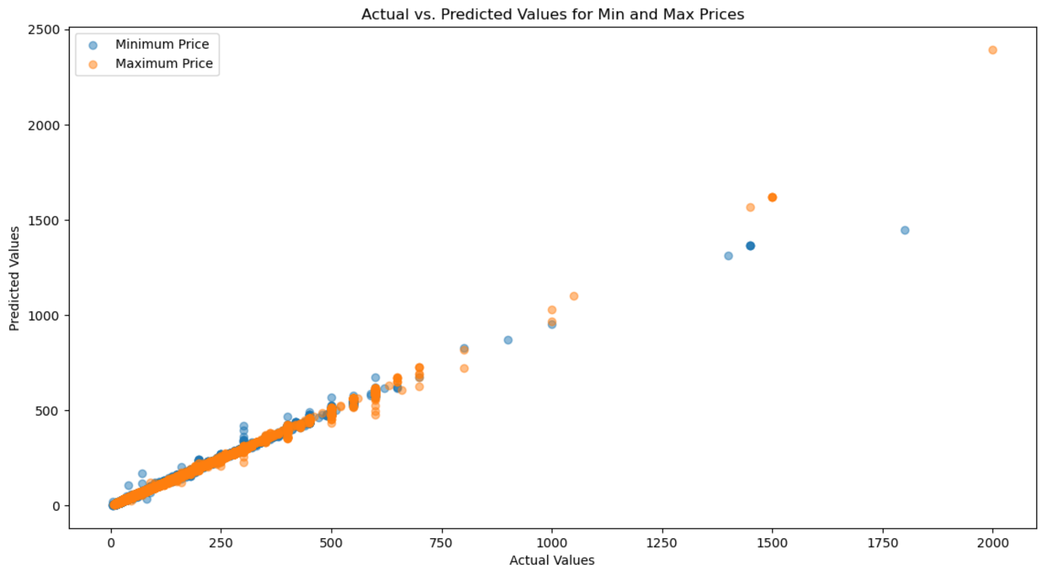


Figure 21 - Scatter Plot of Model's Evaluation Metrics

In the visual comparison of our model's predictions and the actual values, a close alignment is observed along the diagonal in the scatter plot, indicating a potent accuracy in our price predictions. For instance, the first data point had actual values of [65, 70] and the model predicted [63.16, 71.95], demonstrating a commendable proximity. Similarly, the second data point was [140, 150] in reality and our model predicted [140.81, 149.76], again illustrating its apt capacity in accurately forecasting the commodity prices. While these examples and the general trend of the scatter plot suggest a well-performing model, a few deviant points away from the diagonal hint at opportunities for further tuning and investigation, laying a constructive path forward in perfecting our predictive capabilities.

8. Conclusion

Completing this project was like solving a big puzzle, where all the small pieces, like data points, codes, and graphs, helped to see the whole picture clearly. The journey started with a bunch of numbers and information that might have looked confusing at first. But step by step, it all got turned into a structured form that helped build the LSTM model.

Each step, from getting the data ready to creating the model, gave new insights and showed results in different ways. Looking at the organized graphs and understanding the metrics showed how the efforts and decisions made along the way had an impact. It also made clear that there's always room for improvement and learning, even when the results are good.

One key lesson from this project is that no model can be perfect or final. Even with a good R2 Score of 0.9979, the model wasn't flawless. It showed through the small errors and differences in the predicted and actual values. Every mistake or difference is a chance to learn and do better next time. The visuals, like scatter plots, told a silent story about the accuracy and the small mistakes of the model. They highlighted where the model did well and where it needs to be improved, pointing out areas to pay attention to in the future.

Going through the accuracy of the model and identifying mistakes in the test results highlighted important lessons for future projects. The good accuracy achieved showed the importance of getting the data and model tuning right. On the other hand, errors and differences between predicted and actual values showed the importance of keeping things simple and always looking for ways to improve.

So, to wrap it up, this project was a big learning experience, showing that with every step and decision, there's something new to learn and improve upon. Even when things go well, there's always a chance to do better in the future. And that's what makes these projects exciting and valuable.

References

1. Abbasimehr, H., & Paki, R. (2021). *Improving time series forecasting using LSTM and attention models*. Journal of Ambient Intelligence and Humanized Computing, 13, 673–6912. (Accessed: 26th September 2023).

2. Aggarwal, S. (2023). The Ultimate Guide to Building Your Own LSTM Models. (Accessed: 26th September 2023).

3. Anjan V. Thakor *et al.* (2015) *The highs and the lows: A theory of credit risk assessment and pricing through the business cycle*, *Journal of Financial Intermediation*. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1042957315000303?via%3Dihub (Accessed: 19th September 2023).

4. Bag, S. (2022). The Complete LSTM Tutorial with Implementation. (Accessed: 27th September 2023).

5. Brownlee, J. (2018). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery. . (Accessed: 20th September 2023).

6. Department of Agriculture. (2022). Annual Trade Report. [DOA]. Government of Nepal. (Accessed: 01 October 2023).

7. Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural computation, 9(8), 1735-1780. (Accessed: 23rd September 2023).

8. Kumar, S., & Thenmozhi, M. (2019). *Forecasting stock index movement: A comparison of support vector machines and random forest*. Indian Institute of Capital Markets 9th Capital Markets Conference Paper. (Accessed: 19th September 2023).

9. Mehtab, S., Sen, J., & Dutta, A. (2021*). Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models*. (Accessed: 18th September 2023).

10. Sheth, D. and Shah, M. (2023) *Predicting stock market using machine learning: Best and accurate way to know future stock prices - International Journal of System Assurance Engineering and management*, *SpringerLink*. Available at: https://link.springer.com/article/10.1007/s13198-022-01811-1 (Accessed: 26th September 2023).

11. Staudemeyer, R.C., & Morris, E.R. (2019). *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. (Accessed: 24th September 2023).

12. Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). *A review on the long short-term memory model.* Artificial Intelligence Review, 53, 5929–59551. (Accessed: 24th September 2023).