**2.1 Time Series Forecasting**

Neglecting to understand and account for future price trends can be detrimental for any farmer, trader, or policymaker who is trying to plan and shape decisions and policies for the future. Time series forecasting offers a methodology which can help anticipate these price trends and offer crucial insights to the future, by analyzing past time series data (TSD) and predicting future values based on it.

Performance of these models is highly dependent on the understanding and correct analysis of the TSD and its key components, as well as its connected concept of stationarity. This section will serve as an introduction to these concepts.

**2.1.1 Time Series Components**

TSD is typically composed of the three primary components: trend, seasonality and the residual.

1. **Trend:** The trend component, also referred to as the level depicts the slow-moving, general direction of a time series over time, either increasing or decreasing steadily (Jose, Page 4). Trends are critical for long term price forecasting as they offer insights into the general direction of the market, an example being inflation, steadily moving prices upwards in the long term.
2. **Seasonality:** The seasonal component shows the regular and predictable patterns of a time series that reoccur at fixed intervals of time, such as daily, monthly, or yearly (Jose, Page 5). These patterns indicate the positive and negative deviations from the trend and can be caused by natural or artificial factors. For example, prices of certain fruits can vary over the year based on their seasonal availability.
3. **Residual:** After accounting for the trend and seasonality, all remaining variations are represented within the residual. It captures the random noise or unexplained fluctuations, that cannot be attributed to the main components of the series and cannot be foreseen (Jose, Page 5). Examples can be unexpected natural disaster inhibiting crop growth and shortening supply.

TSD can be broken down into its components through decomposition. This process is essential in better understanding and analyzing the underlying properties of a time series.

**2.1.2 Stationarity and Its Importance**

Stationarity is a fundamental concept in time series analysis and presumed by many models, such as the autoregressive ones. A time series is presumed stationary, when its properties, such as the mean, variance and autocorrelation, do not change over time and are therefore independent of it.

This presumption is necessary to ensure that selected model parameters remain consistent with the given time series. Changing properties would otherwise affect the models fit to the data and render possible predictions inaccurate. In most real-world cases, stationary time series are not present from the get-go. This is often due to trend and seasonality being naturally present, requiring the time series to be transformed to become stationary before modelling.

Differencing is the simplest method of achieving stationarity. It involves stabilizing the mean, by subtracting the previous value of a timestamp from the present value , resulting in the differenced value .

Additionally, in order to stabilize the variance a log function can be applied to the series.

Applying differencing once is referred to as first-order differencing. If necessary, multiple rounds of differencing (second-order, third-order differencing etc.) can be applied until the series is ultimately stationary.

For TSD with a seasonal component, seasonal differencing can be applied. Here, instead of subtracting neighboring values from another, the differencing is applied on values data points apart to account for the seasonality.

It is important to remember to inverse difference the model results afterwards, to return the datapoints to their original unit measurements and make them interpretable.

After each round of differencing the augmented Dickey-Fuller (ADF) test can be applied to test whether the transformation successfully made the series stationary. This method determines stationarity by testing for the null hypothesis that a unit root is present. If the root of a time series lies between -1 and 1, the series is stationary, as there is no unit root present. Conversely, if the root equals 1, this indicates the presence of a unit root, making the series non-stationary, making another round of differencing necessary. When applying the method in code, a p-value of less than 0.05 rejects the null hypothesis and indicates stationarity.

**2.3 Lasso Regression**

The Least Absolute Shrinkage and Selection Operator, more commonly known as Lasso regression, is a regularization technique used in statistical modelling and machine learning to enhance the prediction accuracy and interpretability of models. It achieves this by imposing a constraint on the sum of the absolute values of the models, effectively driving some of them towards near zero. This characteristic renders it particularly fit for feature selection, as well as creating parsimonious models.

Traditionally, linear regression determines the best-fitted line by minimizing the sum of the squared residuals. Lasso regression alters this approach by adding a penalty term λ that is proportional to the sum of the absolute values of the coefficients. The loss function can be defined as follows:

Here λ is a non-negative hyperparameter that controls the strength of the penalty. When λ is set to 0, Lasso regression operates the same as ordinary least squares regression, where no penalty is applied at all. Increasing λ and therefore increasing the penalty, leads to a greater shrinkage of coefficient estimates towards zero.

This enables Lasso regression to shrink some coefficients to exactly zero whenever λ is large enough. Through this, feature selection among predictors can be facilitated, by excluding less important predictors, leading to simpler and more interpretable models. This capability particularly demonstrates its usefulness in high dimensional datasets, where selecting variable automatically is crucial in producing sparse models, stripped of redundant attributes.

It is important to note that choosing an appropriate λ is important, as it controls the amount of shrinkage. A common method for determining λ is cross-validation, which helps avoid overfitting by balancing the model’s simplicity and its predictive performance on unseen data.

**2.3.1 The Lasso for Time Series**

These functionalities of Lasso regression prove especially useful in time series forecasting, as high dimensionality is commonly found when dealing with time series data. Effective handling of overfitting and determination of the most relevant predictors is achieved through focusing on the important lagged variables (autoregressive terms), while excluding weak ones, thus leading to more streamlined models.

A univariate autoregressive (AR) model can be defined as:

Where:

* indicate the value at time ,
* are lagged observations,
* are coefficient estimates.

Lasso regression modifies the AR model by introducing the penalty on the absolute values of the coefficients, ensuring some coefficients are driven to 0. The resulting optimization problem of Lasso regression for a time series can be formulated as:

Here:

* is the number of observations,
* is the number of lagged covariates,
* is a hyperparameter controlling the regularization strength.

The penalty term enforces the desired sparsity, driving the coefficients of irrelevant variables to zero.

This makes Lasso regression a valuable tool for simplifying time series models by focusing on the most relevant lagged variables.

**2.4 XGBoost for Regression**

Extreme Gradient Boosting, or short XGBoost is an optimized implementation of the gradient boosting algorithm designed for achieving superior performance and accuracy by leveraging advanced optimization techniques and pushing computational limits (Wade 2020, Chapter 5, para. “Design features”). It is especially popular as it allows for efficient scaling on high-dimensional, large-scale datasets, as is often the case in time series data.

**2.4.1 Boosting in XGBoost**

At its core, XGBoost is a supervised learning algorithm based on the boosting principle, where an ensemble of weak learners, typically decision trees, are combined sequentially to produce a strong final model. In this iterative approach, each subsequent tree tries to minimize the residual error (loss) of the previous trees. This stands in contrast to bagging techniques, as used by Random Forest, where each tree is trained independently and grown randomly. This targeted learning mechanism is what allows XGBoost to efficiently model complex, non-linear relationships.

Mathematically, the goal of boosting is to minimize the following objective function:

Where is the loss function, representing the difference between the true value and the predicted value and is the regularization term to penalize the model complexity. Further, represent the model parameters and the the number of trees in the model.

The regularization term is given as:

where penalizes the number of leaves in the tree and puts a L2 penalty on the leaf weights ​. This regularization mechanism prevents XGBoost from fitting overly complex models that are susceptible to overfitting, while still capturing essential patterns in the data.

**2.4.2 Gradient Boosting with Second-Order Approximation**

To further improve both accuracy and computational efficiency, XGBoost optimizes the objective using a second-order Taylor approximation of the loss function. This leverages both the first-order gradient () and second-order Hessian () of the loss with respect to the model’s prediction at iteration :

where:

* is the first-order gradient, indicating the direction of the loss,
* is the second-order Hessian, which captures the curvature of the loss function.

The inclusion of the Hessian into XGBoost provides information about the curvature of the loss, which enables faster and more stable optimization than other methods where only the first-order information is used.

**2.4.3 Leaf Weight Optimization**

Given a tree structure, XGBoost optimizes the weights of every leaf to minimize the regularized loss. The optimal weight for a leaf is computed as:

where is the set of instances assigned to leaf .

The resulting quality score for a tree is derived as:

This score is used to determine how well the split fits and is used as a criterion for selecting the best split when constructing the tree.

**2.4.4 Sparsity-aware Learning and Regularization Techniques**

XGBoost uses advanced techniques to improve efficiency and generalization. For datasets with missing or sparse features a sparsity aware algorithm learns the best default split directions for handling missing values. Furthermore, shrinkage (also known as learning rate scaling) and column subsampling are used to prevent overfitting and improve computation. Shrinkage multiplies the contribution of each new tree by a constant , which provides a way to obtain a good generalization of the model.

Together, these features enable efficient handling of large datasets while providing a robust model for time series data and prediction tasks.

**2.5 Long Short-Term Memory Networks (LSTM)**

Long Short-Term Memory networks, short LSTMs, are a type of recurrent neural network (RNN) that are designed to address the challenges associated with learning long-term dependencies in sequential data while avoiding issues such as vanishing and exploding gradients. This capability is particularly helpful in time series analysis, where dependencies may extend over long temporal intervals.

**2.5.1 Fundamental Architecture**

LSTMs incorporate a mechanism known as Constant Error Carousel (CEC) which is responsible for preserving a constant error flow during training. This is critical for enabling the network to accommodate to very long sequences, exceeding a thousand timestep lags, so that stable gradients can be preserved over long periods of time.

Each LSTM unit contains a set of gates to manage the information flow: the input gate, the forget gate, the output gate as well as the central component, the cell state. These elements are responsible for the adaptability of the LSTM to different data sequences.

* **Cell State:** The cell state is like a conveyor belt, carrying relevant information throughout the sequencing process without out many transformations. It gets updated through operations that are regulated by the input and forget gates.
* **Input gate:** The input gate determines how much new information enters the cell state. It's defined through the following equation:

where σ denotes the sigmoid function, is the weight matrix for the input gate, ​ is the previous output, is the current input, and is the bias.

* **Forget Gate:** The forget gate controls what information is removed from the cell state. It is formulated as:

The parameter represents the weight matrix for the forget gate.

* **Cell State Update**: Cell state is updated by:

here is the weight matrix associated with the cell state, and is the hyperbolic tangent function that adds non-linearity to the model.

* **Output Gate:** The output gate controls the output from the cell state to the next layer:

and the output of the LSTM unit can be defined by:

where ​ is the weight matrix associated with the output gate.

**2.5.2 Application to Time Series**

The ability of LSTMs to remember and forget is critical in time series forecasting. This selective memory allows the network to keep only essential historical information that is useful for the model, such as trends or seasonality. This feature enables LSTMs to forecast future data points based on complex patterns and dependencies in past data, which is critical for accurate predictions in areas like financial markets, energy load forecasting, and more.

**2.5.3 Enhanced Variants and Practical Use**

Several variants of the basic LSTM architecture, including peephole connections and gated recurrent units (GRUs), have been proposed to improve computational efficiency and enhance model performance. Such improvements enable the LSTMs architecture to be tailored to a particular task, which often require faster calculations without a significant trade-off in performance.

Such flexibility and efficiency position Long Short-Term Memory networks as a key component for modelling a range of complex time series modeling applications across multiple domains.

**3 Related Work**

**3.1 Advancements in Commodity Price Forecasting**

There is an increasing body of literature highlighting several machine learning techniques, which are able to model the complex, non-linear dynamics of commodity price movements. For instance, Mishra et al. (2022) demonstrate how ML models can process high-dimensional data and make reliable predictions in the case of cotton prices. Their study highlights the benefits of feature selection and integration of external variables such as weather patterns and market indices into the model. These findings align with general forecasting trends, with growing incorporation of external variables like global demand fluctuations and trade policies into predictive models.

Similarly, Suresh et al. (2023) utilized neural network-based approaches for agricultural price forecasting, specifying in long short-term memory networks. The study concluded that LSTMs outperform classical regression models by capturing temporal dependencies and avoiding overfitting for noisy datasets. These findings are supported by other studies which show that LSTMs are well-suited for the typical seasonality and sudden price fluctuations in agricultural markets. However, they warn against computational the computational intensity and data requirements of these models, which might not be feasible for all applications.

In a broader context, studies have researched the robustness of simpler models like LASSO regression and more advanced ensemble methods like XGBoost. Research by Sharma et al. (2020) on agricultural commodity price forecasting shows that these models perform well in scenarios with limited data availability, offering balance between interpretability and performance. LASSO regression is recognized for its ability to address multicollinearity, which is a common challenge in economic data, by automatically selecting the most relevant features and avoiding overfitting the model. On the other hand, XGBoost is known for its ability to handle large datasets and its versatility, which allow it to be used for large-scale prediction tasks.

**3.2 Comparative Insights Across Studies**

A recurring theme across literature is the trade-off between model complexity and real-world applicability. While more advanced models such as LSTMs can provide better accuracy in learning complex non-linear trends and seasonal components, simpler models can be suitable for in cases where compute power is limited, or only short-term forecasting is required. For example, Sharma et al. (2020) show that given the right conditions, linear regression models with regularization such as the LASSO can perform as well or better than more complex algorithms. This reinforces the need for specifically tailoring the model selection to the given data and environment conditions.

Notably, a study by Agarwal et al. (2021) highlights how domain-specific knowledge contributes to model selection and performance tuning. Their study of commodity markets shows how integrating market fundamentals with data-driven approaches can improve interpretability of predictions. For instance, by leveraging multiple data sources, such as aggregating price trends with insights into supply chain disruptions or weather patterns, one can not only enhance the accuracy of forecasts but also generate actionable, well-informed insights to support decision making.

**3.3 Challenges and Emerging Directions**

Despite notable advancements, several challenges regarding commodity price forecasting remain. One major problem is the inherently volatile and unpredictable environment of commodity markets, which are often subject to external shocks, like geopolitical events or climate anomalies. Traditional statistical models do not handle sudden changes very well, while ML models require an enormous amount of retraining, which can be recourse intensive. Additionally, ML-based approaches are frequently prone to overfitting, especially if the models are trained on smaller datasets or fail to generalize well past historical data.

Another challenge depicts itself within the quality and availability of data. Advanced models require high-frequency and granular data, which are often unavailable, especially in underdeveloped parts of the world. As pointed out by Zhang et al. (2013), innovative strategies for data preprocessing and augmentation are vital for addressing these limitations. Techniques like missing value imputation, normalization and feature engineering are considered the key to enabling improved model performance.

**3.4 Summary and Conclusion of Literature**

The literature on commodity price forecasting shows a wide variety of methodologies, each with distinct strengths and weaknesses. Machine Learning models like LSTMs and XGBoost offer state-of-the-art performance for capturing complex temporal relations. However, less complex models like LASSO regression still provide good performance and high interpretability, while requiring less computational power.

Therefore, in the case of price predictions for Nepalese commodities, data availability will play a crucial role in evaluating the performance of the models, as well as effectively leveraging their trade-offs.

**4 Methodology**

**4.1 Data Collection**

**4.1.1 Original Dataset Description**

The studies dataset was obtained from Kaggle's Agriculture Vegetables and Fruits Time Series Prices dataset. It consists of daily price records for different fruit and vegetables collected across Nepalese markets from 1. January 2013 to 31. December 2021. It is comprised of seven columns including SN (serial number), Commodity, Unit (e.g. kg), Date, Minimum, Maximum and Average.

For this implementation the analysis and predictions were limited to the commodity “Potato Red”, an important vegetable in Nepal’s agricultural sector. The data was filtered based on the commodity column to only include records for “Potato Red”. After filtering, two key columns were retained:

* Date: The date column provides the temporal dimension of the data, with daily records that enable detailed analysis for price variations over time. In the case of the red potatoes the time frame ranged from the 16.06.2013 to the 13.05-2021.
* Average: This column represents the average daily price of red potatoes on a given date and is the target variable for the prediction. The choice to focus on the average price, instead of minimum or maximum, was made to provide a generalized measure of market conditions, avoiding unnecessary complexity from modeling price ranges.

Other columns like SN, Unit, Minimum and Maximum were dropped as they provide no additional information relevant to the time series.

In order to capture information about the seasonal and periodic trends available to the model, features such as Year, Quarter, Month, Week and Day were derived from the Date feature. For instance:

* **Year** can reflect long-term trends, like gradual price rises or falls brought on by macroeconomic shifts.
* **Quarter** and **Month** may denote seasonal cycles associated with harvest periods or demand variation.
* **Week** and **Day** can help detect short-term trends and periodic patterns like weekly market activities.

This data set formed the basis for the predictive modeling task with its daily granularity providing insights into short-term fluctuations as well as seasonal trends. By focusing on the Date and Average columns, the dataset provides the necessary structure to analyze the relationship between the daily price of "Potato Red" and external factors such as economic indicators, weather conditions, and holidays.

**4.1.2 External Feature Collection**

The feature collection process integrated a variety of external data to construct a solid foundation of information about external factors influencing vegetable prices in Nepal. It focused on economic and agricultural indicators, weather data and information about national holidays, with the aim to capture key variables critical to price dynamics, while maintaining consistent temporal resolution.

Economic data was obtained via the World Bank API, covering indicators such as Consumer Price Index (CPI), exchange rates, GDP growth and agricultural productivity metrics. They were selected based on their relevance to economic and agricultural conditions that affect pricing. Due to the data only being available on a yearly basis, interpolation was used to reduce the data to monthly intervals. While this approach introduces risk of distorting the data it was deemed necessary to align with the temporal resolution of the other data.

Additionally, FAOSTAT data on agricultural metrics regarding red potatoes specifically, was sourced to enrich the dataset with information on the commodity itself. This data contained annual statistics on import and export volumes, production and yield quantities and harvested area. Once again, as with the economic data these metrics also had to be resampled to a monthly resolution using interpolation.

Unsing the Open-Meteo API and due to its impact on agricultural yield and possible availability of red potatoes weather data was collected for the city of Kathmandu, which reflects the agricultural climate of central Nepal. The data, which is available on a daily basis was aggregated to monthly values, ensuring consistency.

Finally, public holiday data was joint into month wise counts to account for possible market disruptions or demand shifts during festival seasons.

The monthly resolution was chosen as a compromise between the coarser intervals of economic and agricultural data and the finer daily granularity of the target price variable, as well as the weather data. Even though interpolation is likely to introduce some risk of distortion or loss of temporal detail, the monthly resolution ensures consistency across all data and provides a practical foundation for integrating diverse sources of information, while also retaining a reasonable level of temporal detail.

The resulting dataset combines macroeconomic, agricultural, environmental, and social factors. It offers a foundation for predictive modeling while acknowledging trade-offs in aligning data from varied sources.

**4.1.3 Data Leakage Prevention**

In time-series forecasting and many other ML tasks, avoiding data leakage is crucial to ensure the validity and reliability of the model’s performance. Data leakage occurs when information that shouldn’t be available at the time of prediction gets leaked into the training data, resulting in overly optimistic predictions that could not be achieved under realistic conditions. This becomes particularly significant for external features as they get integrated into the modeling pipeline, since they also gave to be constraint to the respectable timeframe.

In this study, the introduction of external features, such as economic and weather data, introduced unique challenges in that regard. Since the goal is to forecast into the future, subsequent test features, couldn’t be sourced from the available dataset, as that would introduce information about the data, which otherwise wouldn’t be available yet in a real scenario. As a result, the prediction pipeline would have to include predicting the external features themselves, before using them for the final model training. This approach is essential in ensuring that data leakage can’t occur in the modeling pipeline.

**4.1.4 Feature Prediction Framework**

The ARIMA (AutoRegressive Integrated Moving Average) model was used for the feature predictions. ARIMA is a common, statistical model that combines autoregressive (AR) terms, differencing to achieve stationarity (I), and moving average (MA) terms. Each feature was treated as an independent time series, and stationarity tests were performed prior to modeling. Where required, differencing was applied to make the series stationary.

However, this method encountered substantial constraints due to the distortion introduced by interpolation and aggregation beforehand. ARIMA-predicted values for many of these features deviated substantially from the true values, leading to high error margins and unreliable predictions.

**4.1.5 Feature Exclusion**

Due to the poor predictive accuracy of the economic and agricultural features, generally deviating by upwards of 600%, dropping them for the final modeling pipeline was necessary. While these variables are theoretically important for explaining price dynamics, their poor prediction accuracy would have introduced a lot of noise into the model. Additionally, their original temporal resolution rendered them impractical for real world forecasting at a finer frequency level.

With the exclusion of the economic and agricultural data, the modeling shifted from a monthly to a daily approach. The decision was driven by the daily availability of all the remaining features, e.g. price, weather and holidays. By working with the original frequency of the data the pipeline avoided unnecessary resampling and aggregation, ensuring consistency across all features.

Furthermore, due to general weather prediction being very accurate to real-world events, it can be assumed that the data present in the test set, would be available at the time of performing predictions. Therefore, although the ARIMA-predicted values were fairly accurate with a deviation of around 15%, they were deemed unnecessary and thus the original weather data was kept for the modeling process. This approach ensured simplicity and reliability without compromising the model’s real-world applicability.

**4.2 Initial Modeling**

Building on the refined dataset and the decision to exclude economic features, the modeling phase focused on implementing Lasso Regression, XGBoost, and an LSTM neural network. These models were chosen to explore both traditional machine learning techniques and advanced deep learning methods for predicting the daily price of red potatoes. This section outlines the additional preprocessing steps and the implementation of each model, the results of hyperparameter tuning and the insights gathered from the feature importance analyses.

**4.2.1 Preprocessing**

For the preprocessing a MinMaxScaler was used to scale both the target variable as well as the all the features to a value range between 0 and 1. Both LASSO regression and LSTMs are highly dependent on scaled data. The L1 regularization in LASSO penalizes the coefficients relative to their absolute values, not scaling them would therefore result in features with larger value ranges disproportionately influencing the regularization process, potentially skewing feature selection. Regarding LSTMS, scaling ensures stable gradient decent optimization during training. XGBoost natively handles scaling due to its decision tree-based architecture, so scaling it manually wouldn’t be required specifically, but was still performed for consistency.

For LSTMS, the data was also reshaped into 7-day sequences, enabling the model to learn from weekly dependencies over the past weeks. This sequencing step is critical for LSTMs, as it allows the model to process time-series data in chunks, leveraging its memory mechanism to capture short- and long-term dependencies. Categorical data, such as holidays, was encoded as numerical values to ensure seamless integration with the machine learning pipeline.

**4.2.2 LASSO Regression**

LASSO regression was chosen as a baseline model because of its simplicity and its ability to perform feature selection through regularization.

1. **Initial Implementation:** The model was trained on the MinMax scaled features and achieved a mean absolute error (MAE) of 17.2 on the test set.
2. **Feature Importance:** Among the features, Quarter (17.19) and Year (10.20) emerged as the most influential, suggesting that long-term temporal trends were key drivers of price dynamics. Short-term temporal features (Day, Month, Week) and weather variables were excluded completely, with coefficients at 0. This indicates that Lasso struggled to capture the impact of non-linear or shorter-term patterns.
3. **Hyperparameter Tuning:** Hyperparameter tuning was conducted using a grid search with cross-validation, optimizing parameters such as *alpha*, *max\_iter*, and *tol*. The best configuration included *alpha* = 0.24, *max\_iter* = 1000, and *tol* = 0.0001. However, the MAE increased slightly to 17.77 after tuning, suggesting that the initial model had overfitted to the training data, and tuning improved generalization at the expense of marginally worse test performance.

**4.2.3 XGBoost**

To capture non-linear relationships between features and the target variable, XGBoost was used for modeling. Its capacity to model complex interactions and capture non-linear dependencies made it a strong candidate for capturing the dependencies between weather, time and price features.

1. **Initial Implementation:** The initial XGBoost model, trained with default parameters, achieved an MAE of 17.17, comparable to Lasso Regression.
2. **Feature Importance:** Computed feature importance scores proposed both temporal and weather-related features as key contributors. The most important features were Week (score of 2499), Year (2447) and temperature\_2m\_mean (1284), followed by other weather variables such as temperature\_2m\_max (1263) and precipitation\_sum (775). Where LASSO failed to capture the effects of weather features XGBoost was able to effectively capture these relationships, highlighting its ability to model non-linear relationships.
3. **Hyperparameter Tuning:** Performed by RandomizedSearchCV to optimize parameters such as *learning\_rate*, *max\_depth*, *min\_child\_weight*, *gamma*, *subsample*, *colsample\_bytree*, and n\_estimators. The best configuration included a *learning\_rate* of 0.0049, *max\_depth* of 10, and *n\_estimators* of 400. Post-tuning, the MAE slightly increased to 17.97, reflecting that the initial model overfitted to the training data, and tuning improved generalization at the expense of test performance.

**4.2.4 LSTM**

Recognizing the sequential nature of the data, an LSTM neural network was implemented to leverage its ability to capture temporal dependencies.

1. **Initial Implementation:** The LSTM model consisted of two bidirectional LSTM layers with dropout regularization to prevent overfitting.
2. **Learning Rate Optimization:** To determine the optimal learning rate, a learning rate scheduler was employed. The scheduler adjusted the learning rate dynamically during training using the formula . This approach progressively increased the learning rate over 100 epochs, and the learning rate corresponding to the lowest loss was identified as optimal.
3. **Training**: The Adam optimizer, configured with this dynamically adjusted learning rate, was then used to train the model with a final optimal learning rate of 0.000398.
4. **Results:** The model was compiled using the mean\_squared\_error loss function and trained for 300 epochs. After training, the LSTM achieved a test set MAE of 16.55. While the LSTM model effectively captured sequential patterns, its performance was comparable to Lasso Regression and XGBoost, highlighting the challenge of improving predictions beyond simpler models for this dataset.

**4.2.5 Comparative Analysis and Insights**

The initial modeling phase demonstrated the varying strengths and limitations of the tested approaches. LASSO regression, while interpretable, was noticeably limited by its linearity and therefore couldn’t capture non-linear relationships, nor short-term dynamics. XGBoost was more flexible, capturing non-linear interactions and incorporating the importance of weather features. However, its performance was limited by overfitting in the initial implementation, barring the question of how well the tuned model will perform on varied data. The LSTM, while theoretically well suited for sequential data, still performed very similarly to less complex models such as LASSO and XGBoost.

The feature importance analyses emphasize the significance of temporal features, and in particular Year, Quarter, and Week contributions for capturing price trends. The weather variables, while not being considered by LASSO, demonstrated their importance for non-linear interactions through XGBoost. These results highlight the importance of thoughtful feature engineering and model selection in accordance with the nature of the data.

**4.3 Simulation Study**

To evaluate the robustness of the predictive models developed in the initial modeling phase, a simulation study will be performed using synthetic datasets based on the original data. The study aims to evaluate the various model’s performance under different hypothetical scenarios, representative of real-world complexions, such as structural breaks, outlier, or changes in seasonality. The simulation pipeline was developed to systematically manipulate key elements of the original data while maintaining its general temporal structure.

**4.3.1 Data Preparation**

The basis for the data simulation were the decomposed components of the original dataset: trend, seasonality and residuals. They were derived through decomposition. Each decomposed component was processed to handle missing values via forward and backward filling, ensuring complete datasets for generating synthetic variations.

Nine synthetic datasets were created by systematically altering the decomposed components to simulate various real-world phenomena:

1. **Amplified Trend:** Increased the magnitude of the trend component to simulate stronger long-term growth or decline.
2. **Increased Noise:** Added random Gaussian noise to emulate more volatile data.
3. **Reduced Seasonality:** Decreased the amplitude of seasonal components to mimic weakening periodic effects.
4. **Nonlinear Trend:** Introduced a quadratic trend to simulate non-linear growth patterns.
5. **Added Anomalies:** Inserted random spikes into the data to represent outliers or abrupt changes.
6. **Structural Break:** Imposed an abrupt shift in the trend mid-series to simulate a sudden regime change.
7. **Shortened Seasonality:** Adjusted the seasonal frequency to simulate shorter cycles, such as those caused by external disruptions.
8. **Non-Stationary Variance:** Gradually increased the residual variance over time to reflect changing levels of unpredictability.
9. **Regime Switching:** Modeled two distinct regimes with contrasting trends to test adaptability under varying conditions.

Each synthetic dataset was constructed by combining the modified components with the remaining unaltered elements, ensuring a realistic representation of hypothetical scenarios.

**4.3.2 Model Evaluation Framework**

For each synthetic dataset, the models saved during the initial development phase i.e., the Lasso Regression, XGBoost, and LSTM were used to predict. The evaluation pipeline was as follows:

* **Data Scaling:** Target and Features were scaled with MinMaxScaler to maintain consistency with requirements of the models.
* **Train-Test Split:** The final year (365 days) of data in each synthetic dataset was reserved as the test set, with the preceding data used for training.
* **Prediction and Metrics:** Models were evaluated using the Mean Absolute Error as the metric to determine how well the predicted values aligned with the true values in the synthetic test set.