Application of Machine Learning and Statistical Models in Predicting VIX Stock Prices

Nguyen Phan Phuong Ngan[22520934], Le Pham My Ngoc[22520954], Vo Minh Ngoc[22520962], Pham Quoc Khanh[22520649], and Tran Binh Minh[22520887]

University of Information Technology, Thu Duc, HCMC, Vietnam
`https://www.uit.edu.vn/`

Abstract. Forecasting stock prices remains a complex task due to the nonlinear, seasonal, and volatile nature of financial data. This study focuses on forecasting the closing price of VIX (VIX Securities Joint Stock Company), a representative of the Vietnamese financial sector. We apply and compare several time series and machine learning models, including ARIMA, Multiple Linear Regression, K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Facebook Prophet, Long Short-Term Memory (LSTM), Bayesian Structural Time Series (BSTS), and Vector Autoregression (VAR). The models are trained and evaluated under three different dataset split strategies: 75%-10%-15%, 70%-10%-20%, and 65%-10%-25% for train, validation, and test sets, respectively. To ensure a comprehensive comparison, we employ multiple performance metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and R-squared (R^2). The goal is to identify the most accurate and robust model for stock price prediction in the context of the Vietnamese market. Among the evaluated models, Multiple Linear Regression (MLR), Support Vector Regression (SVR), and K-Nearest Neighbors (KNN) demonstrated the most consistent and accurate forecasting performance across various data splits.

Keywords: VIX stock price prediction · ARIMA · Long short-term memory (LSTM) · Multiple Linear Regression (MLR) · K-Nearest Neighbors (KNN) · Support Vector Regression (SVR) · Facebook Prophet · Bayesian Structural Time Series (BSTS) · Vector Autoregression (VAR) · PCA .

1 Introduction

The stock market plays a crucial role in modern economies. It not only facilitates capital mobilization for businesses and governments but also provides profit-generating opportunities for individual and institutional investors. In today's volatile financial environment — shaped by macroeconomic shifts such as interest rate changes, fiscal policy, and global events — accurate stock price forecasting has become increasingly important in supporting sound investment decisions.

Stock price prediction is inherently complex due to non-linear behavior, seasonal trends, and high volatility in financial data. Recent studies have demonstrated that machine learning and time series analysis techniques can yield promising results for such tasks. For example, Zhang et al. (2020) used LSTM networks for S&P 500 forecasting [1], while Hyndman&Athanasopoulos (2018) applied ARIMA and Prophet to financial time series [2]. Inspired by these academic approaches and our own practical experimentation, we apply methodologies similar to those of the Vietnamese stock market.

In this study, we focus on the VIX Securities Joint Stock Company (VIX) stock - a notable firm in Vietnam's financial sector - and our goal is to forecast its closing price for the next 30 consecutive days. Our approach includes the implementation and comparison of multiple forecasting models : ARIMA, LSTM, Multiple Linear Regression, KNN, SVR, Facebook Prophet, Bayesian Structural Time Series (BSTS), and VAR. Through empirical evaluation, we aim to identify the most suitable model for stock price forecasting in the context of Vietnam's emerging financial market.

2 Related Work

Some research work related to the application of machine learning models to forecast stock prices.

James Hosker et al. (2018) [3] compared traditional models (MLR, PCA, ARIMA) with machine learning methods (RNN, LSTM) for predicting 3-day and 5-day VIX futures. Results showed that LSTM and RNN outperformed traditional models. For example, with 10-fold CV, MLR gave $R^2 = 0.325$ and MSE = 26.76 (3-day), while LSTM achieved higher accuracy and lower error across all horizons.

Yinuo Zhai (2024) [4] used Random Forest, SVR, and XGBoost to predict VIX volatility. Among them, XGBoost achieved the best performance with MSE = 0.0022, MAE = 0.0354, and $R^2 = 1.0000$, outperforming Random Forest (MSE = 0.0380) and SVR (MSE = 0.0066), showing its superiority on small, low-dimensional time series data.

Bai & Cai (2023) [5] Using an automated machine learning framework with 278 economic and financial variables to forecast the VIX index which choose 6 methods of Naive Bayes, Logistic Regression, Decision Tree, Random Forest, Adaptive Boosting, Multi-Layer Perceptron, and Ensemble model. Adaptive Boosting achieved an average accuracy rate of 57% during the validation period.

Nguyen Thi Thanh Loan et al. (2025) [6] compared the performance of the ARIMA and LSTM models in forecasting stock prices of major Vietnamese companies. For example, in the prediction of the VCB stock, ARIMA achieved RMSE = 13.22 and MAPE = 11. 2%, while LSTM outperformed significantly with RMSE = 3.35 and MAPE = 3.3%. Similar improvements were observed in other stocks such as VHM (RMSE reduced from 6.09 to 2.21) and VIC (RMSE reduced from 11.78 to 3.39), showing that LSTM consistently achieved lower RMSE and MAPE values than ARIMA.

Shengtao Gao (2023) [7] proposed a trend-based stock price prediction method using the KNN algorithm. He compared multiple models including Linear Regression, SVM, KNN, and Improved KNN. The results showed that Improved KNN outperformed the others with RMSE = 0.394 and $R^2 = 0.988$, while Linear Regression had RMSE = 1.126 and $R^2 = 0.903$, and standard KNN had RMSE = 1.237 and $R^2 = 0.883$. This demonstrates the superior accuracy of the Improved KNN model in stock price prediction.

Nguyen Trung Tuan et al. (2022) [8] used the Stacked LSTM model to predict stock prices in Vietnam and tested its performance on AAPL, AAA, and VCB stocks with 2, 3, and 4 LSTM layers. For AAPL, the best result was with 2 layers: RMSE = 0.725, MAE = 0.503, MAPE = 0.009. For AAA, 2 layers also yielded the best performance with RMSE = 0.285, MAE = 0.219, MAPE = 0.014. Similarly, for VCB, 2 layers outperformed deeper models with RMSE = 1.793, MAE = 0.976, MAPE = 0.010, showing the effectiveness of shallower Stacked LSTM for time-series prediction.

3 Method

3.1 Data Overview

The dataset used in this study consists of historical stock price data of VIX Securities (VIX), collected from Investing.com [9]. The data spans from August 1, 2021, to May 16, 2025, comprising a total of 1,082 records corresponding to daily trading sessions on the Ho Chi Minh Stock Exchange (HoSE).

Each data entry includes the attributes: Date, Open, High, Low, Close, Volume, and Change. In this study, only the Close price is used for forecasting. The data was preprocessed by converting the date field to datetime format, handling missing values, and transforming the Change column from raw values into percentages.

The dataset is divided into three subsets: training, validation, and testing, with three different splitting strategies: 75/10/15, 70/10/20, and 65/10/25. The aim is to select the most optimal model configuration. This partitioning ensures that the model is trained on sufficient historical data, properly validated, and fairly evaluated.

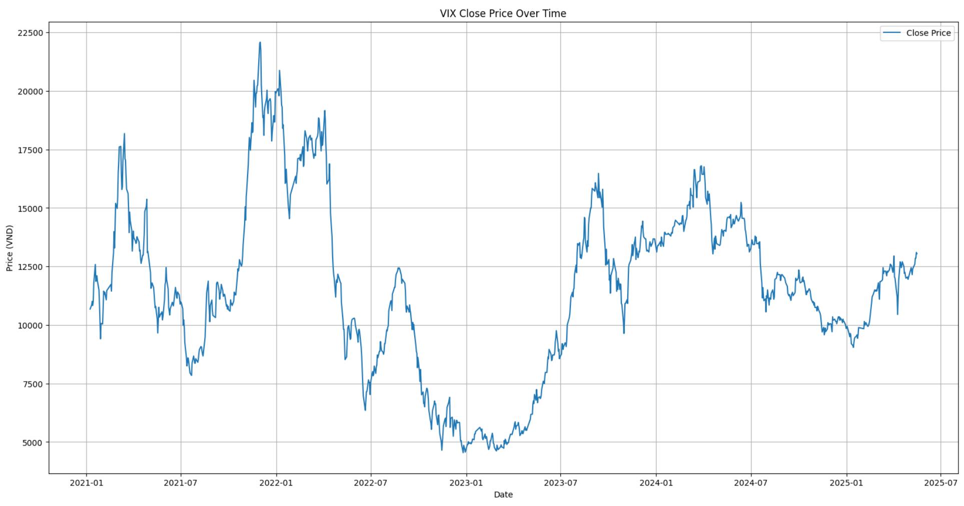

Fig. 1. Overview of VIX Data

Table 1. Data Summary

Variable	**Count**	**Mean**	**Std**	**Min**	**25%**	**50%**	**75%**	**Max**
Price (Close)	1082	11558.34	3660.59	4542.5	9410.63	11470.65	13739.2	22084.1
Open	1082	11581.61	3676.72	4325.3	9423.9	11500.0	13749.85	22084.1
High	1082	11816.57	3738.26	4739.4	9612.5	11758.75	13982.33	22687.5
Low	1082	11318.60	3579.36	4325.3	9201.48	11251.65	13517.6	21842.8
Volume	1082	20619029.39	16592374.57	959800.0	8942500.0	15180000.0	27600000.0	113120000.0
Change	1082	0.00084	0.03462	-0.1859	-0.017375	0.0	0.0184	0.0699

3.2 Employed Prediction Models

This study considers and compares the performance of several common machine learning and statistical algorithms for the task of resale price prediction, including Multiple Linear Regression, K-Nearest Neighbors (KNN), Support Vector Regression (SVR), ARIMA, and Gradient Boosting .

1. **Multiple Linear Regression with PCA**
   Multiple Linear Regression (MLR) is one of the most basic and widely used predictive algorithms.[11] It models the relationship between a dependent variable y (e.g., resale price or closing price) and multiple independent variables $X = \{X_1, X_2, ..., X_p\}$. The goal of MLR is to find the best-fitting linear function by minimizing the sum of squared errors between predicted and actual values.

   The general form of the MLR equation is:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon \tag{1}$$

   Where:
   - y is the predicted value (e.g., the stock's closing price).
   - X_j are the independent variables (e.g., Open, High, Low, Volume, Change, Day_of_Week, Month).
   - β_0 is the intercept term.
   - β_j are the regression coefficients estimated from training data using the Ordinary Least Squares (OLS) method.
   - ϵ is the random error term.

Incorporating Principal Component Analysis (PCA): To enhance the performance of the regression model and mitigate multicollinearity, Principal Component Analysis (PCA) is used as a dimensionality reduction technique. PCA transforms a high-dimensional dataset into a lower-dimensional space while preserving most of the variance in the data. [11] The standard PCA steps include:

– Standardizing the data.
– Computing the covariance matrix.
– Calculating eigenvectors and eigenvalues.
– Selecting principal components based on the cumulative explained variance.

In this project, PCA is specifically applied to the three price-related features: *Open*, *High*, and *Low*. These are compressed into a single principal component called *Price_ PC1*, which captures the essential variance across the price metrics, reduces noise, and avoids multicollinearity.

Combining MLR with PCA: The final model combines the advantages of both techniques [12]:

– It reduces the dimensionality of the input space, making the model more interpretable.
– It eliminates redundancy among correlated features (e.g., Open, High, Low).
– It improves model stability and generalization by reducing the risk of overfitting.

After PCA transformation, the resulting principal components are used alongside other standardized features as inputs to the MLR model to predict the *Close* price of the VIX stock[13].

The model is built by combining *ColumnTransformer*, *Pipeline*, and *LinearRegression* from the *Scikit-learn* library. The processing consists of two main steps: feature preprocessing and model training.

Data Preprocessing

– **Price_PCA**: Apply *StandardScaler* and *PCA* with *n_ components = 1* to the columns *Open*, *High*,*Low.*
– **Other features**: Apply *StandardScaler* to the variables: *Volume*, *Change*, *Day_ of_ week*, *Month.*

Machine Learning Model Use the linear regression model: *LinearRegression()* (without regularization).

Table 2. Model Configuration Parameters

Component	Value
PCA n_components	1
Model	`LinearRegression()`
Scaling	`StandardScaler()`
Input features	`Open, High, Low, Volume, Change, Day_of_week, Month`
Target variable	`Close`

2. **K-Nearest Neighbors (KNN)**

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for both classification and regression. For regression, KNN predicts the target value of a new data point based on the average (or median) of the target values of its K nearest neighbors in the feature space. The "closeness" of neighbors is typically determined using a distance metric, such as Euclidean distance[14]: The Euclidean distance is defined as:

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2}$$

In the formula above, **p** and **q** represent two data points in the feature space, and n denotes the number of features[15].

The choice of K and the distance metric are crucial hyperparameters. KNN is simple to understand but can be computationally expensive with large datasets and sensitive to the scale of features.

In case, we use **K = 5**, the KNN model will find 5 data points in training set that are closest to the new data point (based on Euclidean distance). Then, the model will compute the average of the target values corresponding to these 5 nearest neighbors to produce the predicted output value for the new point.

3. **Support Vector Regression (SVR)**
Support Vector Regression (SVR) is a variant of Support Vector Machine (SVM) extended to address regression tasks rather than classification. SVR aims to find a function $f(x)$ such that the deviation between the predicted and actual values is within a predefined threshold ε, while simultaneously ensuring the model remains as flat as possible. This approach balances prediction accuracy and model complexity, contributing to better generalization performance [16,17]. The goal of Support Vector Regression (SVR) is to find a regression function $f(x)$ such that:

$$|f(x_i) - y_i| \leq \varepsilon \tag{3}$$

for most of the training data, while keeping the function as **flat** as possible. To handle outliers beyond the ε margin, SVR introduces *slack variables* ξ_i and ξ_i^* to allow some flexibility [31].

Basic SVR Formulation: Assume a linear regression function of the form[31]:

$$f(x) = \langle w, x \rangle + b \tag{4}$$

Then the SVR regression function becomes[31]:

$$f(x) = w^T x + b \tag{5}$$

Where:

– x: **Input feature vector**, e.g., `[Open, High, Low, Volume]` for a day
– w: **Weight vector** associated with each input feature (learned during training)
– $w^T x$: **Dot product** between weights and input features — this is the raw predicted output
– b: **Bias term** — allows the model to better fit the actual data by shifting the prediction line

Optimization Problem: SVR solves the following optimization problem [31]:

$$\min_{w,\xi_i,\xi_i^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{6}$$

Subject to the constraints [31]:

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{7}$$

Key Components

– ε (**epsilon**): Tolerance margin — the width of the ε-tube within which no penalty is given
– C (**regularization parameter**): Controls the trade-off between model complexity and training error
– ξ_i, ξ_i^* (**slack variables**): Represent the degree of violation of the ε-margin
– **Kernel function**: Extends SVR to nonlinear problems by mapping data to higher-dimensional feature spaces

Hyperparameters Used The training process of the SVR model involves several important hyperparameters tuned via GridSearchCV:

- **C = 100**: The regularization parameter. Higher values of C aim to reduce bias but can risk overfitting.
- **epsilon = 0.01**: The width of the epsilon-insensitive tube. This defines the tolerance within which no penalty is given to errors.
- **gamma = 0.1**: The kernel coefficient for the RBF kernel. Controls the influence of single training examples; larger values lead to tighter influence.

4. **Facebook Prophet: A Time Series Forecasting Model** Facebook Prophet is a time series forecasting model developed by Facebook (now Meta), designed with the goals of [19] ,[20]:

- Easy to use for both experts and non-experts.
- Effective in forecasting data with strong seasonal effects and nonlinear trends.
- Robust to missing data, outliers, and trend changes.

Prophet models a time series using the following additive model :

$$y_t = g(t) + s(t) + h(t) + \varepsilon_t \tag{8}$$

where:
- $g(t)$: trend component
- $s(t)$: seasonality component
- $h(t)$: holiday effects
- ε_t: noise term

Trend Component: Prophet uses a piecewise linear trend model:

$$g(t) = \left(k + \mathbf{a}(t)^\top \boldsymbol{\delta}\right)(t - t_0) + \left(m + \mathbf{a}(t)^\top \boldsymbol{\gamma}\right) \tag{9}$$

where:
- k: initial growth rate
- $\boldsymbol{\delta}$: vector of rate adjustments at changepoints
- $\mathbf{a}(t)$: indicator function determining whether t is after a changepoint
- m: offset parameter (intercept)
- t_0: reference time (usually normalized to zero)

Seasonality Component: Seasonality is modeled using a Fourier series:

$$s(t) = \sum_{n=1}^{N} \left[a_n \cos\left(\frac{2\pi n t}{P}\right) + b_n \sin\left(\frac{2\pi n t}{P}\right) \right] \tag{10}$$

where:
- P: periodicity (e.g., 365 days for yearly, 7 for weekly)
- N: number of Fourier terms (higher values allow greater flexibility)
- a_n, b_n: Fourier coefficients learned during training

Holiday Effects: Holiday effects are modeled as:

$$h(t) = \sum_{j=1}^{J} Z_j(t)\kappa_j \tag{11}$$

where:
- $Z_j(t)$: indicator function (equals 1 if t is the j-th holiday, otherwise 0)
- κ_j: magnitude of the effect for holiday j

Noise Term: The residual component is assumed to follow white noise:

$$\varepsilon_t \sim \mathcal{N}(0, \sigma^2) \tag{12}$$

where:

- ε_t: error at time t
- σ^2: variance of the noise

5. **Long Short-Term Memory (LSTM)**
S. Hochreiter et al. came up with the idea by introducing a novel, efficient, gradient based method called long short-term memory (LSTM) to address the issue of vanishing gradients in traditional Recurrent Neural Networks. The LSTM network architecture consists of three parts; these three parts of an LSTM unit are known as gates. They control the flow of information in and out of the memory cell or LSTM cell. The first gate is called Forget gate, the second gate is known as the Input gate, and the last one is the Output gate. An LSTM unit that consists of these three gates and a memory cell or LSTM cell can be considered as a layer of neurons in a traditional feedforward neural network, with each neuron having a hidden layer and a current state [21],[22].

Forget Gate The first stage in architecture is Forget Gate. In this stage, the LSTM neural network will determine which elements of the cell state (long-term memory) are relevant based on the previous hidden state and the new input data. Calculate the forget gate according to the following formula:

$$f_t = \sigma(W_f X_t + U_f h_{t-1} + b_f) \tag{13}$$

The output values of the forget gate (f_t) are then multiplied element-wise with the previous cell state (C_{t-1}) as part of the cell state update (see Equation 16).

Input Gate The following stage involves the input gate and the new memory network. The objective of this step is to identify what new information should be incorporated into the network's long-term memory (cell state), based on the previous hidden state and the current input data. Calculate the input gate according to the following formula:

$$i_t = \sigma(W_i X_t + U_i h_{t-1} + b_i) \tag{14}$$

The new memory network is a neural network that uses the tanh activation function and has been trained to create a "new memory update vector" by combining the previous hidden state and the current input data. Calculate the new memory network according to the following formula:

$$\tilde{C_t} = \tanh(W_c X_t + U_c h_{t-1} + b_c) \tag{15}$$

Cell State Update The cell state is updated by combining the influence of the forget gate on the previous cell state and the input gate on the new candidate values.

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C_t} \tag{16}$$

Output Gate In the final stage of an LSTM, the new hidden state is determined using the newly updated cell state, previous hidden state, and new input data. The output gate performs this decision-making process. Calculate the output gate according to the following formula:

$$O_t = \sigma(W_o X_t + U_o h_{t-1} + b_o) \tag{17}$$

The updated cell state is then passed through a tanh activation to limit its values to [-1,1] before being multiplied pointwise by the output of the output gate network to generate the final new hidden state.

$$h_t = O_t \circ \tanh(C_t) \tag{18}$$

Hyperparameters Used The training process of the LSTM model involves several important hyperparameters:

- **lookback = 30**: This parameter defines the number of past time steps used to construct each input sequence for the LSTM. It determines how much historical context the model considers when making predictions.
- **batch_size = 32**: The number of training samples used in one forward/backward pass during training. A smaller batch size can lead to more stable but slower training, while a larger batch size might speed up the process but can reduce generalization.
- **epochs = 50**: The maximum number of complete passes through the training dataset. The model is trained for up to 50 epochs unless early stopping is triggered.
- **EarlyStopping (patience = 5)**: This callback monitors the validation loss during training and stops the training process if the loss does not improve for 5 consecutive epochs, helping to prevent overfitting.

6. **Bayesian Structural Time Series (BSTS)**

State Space Model Nature: BSTS assumes that the observed data (y_t) is generated by a set of latent (unobserved) states (α_t) that evolve over time. These states represent structural components of the time series [23], [24]

Bayesian Inference: BSTS employs Bayesian methods to:

- Estimate latent states and model parameters.
- Incorporate prior knowledge.
- Quantify uncertainty in estimates and forecasts.

Core Algorithms:

- **Kalman Filter:** Used to recursively estimate latent states.
- **Spike-and-Slab Prior:** Commonly used for variable selection in the regression component, identifying significant predictors.
- **Markov Chain Monte Carlo (MCMC):** Used to sample from the posterior distribution of parameters and states when analytical computation is intractable.
- **Bayesian Model Averaging (BMA):** Combines predictions from multiple models (or parameter sets) to improve robustness.

The BSTS model is typically expressed in state space form with two main equations [25], [26]:

Observation Equation: Links the observed data y_t to the latent state vector α_t:

$$y_t = Z_t^T \alpha_t + \epsilon_t \tag{19}$$

Where:

- y_t: Observed value at time t.
- α_t: Latent (unobserved) state vector at time t.
- Z_t^T: Vector (or matrix) linking the states to the observation.
- ϵ_t: Observation noise, typically assumed $\epsilon_t \sim N(0, H_t)$ (e.g., $H_t = \sigma_\epsilon^2$).

State Transition Equation: Describes the evolution of the state vector α_t over time:

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t \tag{20}$$

Where:

- T_t: Transition matrix, describing the dynamics of the states.
- R_t: Control matrix (or noise loading matrix).
- η_t: System noise (state disturbance), typically assumed $\eta_t \sim N(0, Q_t)$.

Hyperparameters Used The training process of the BSTS model involves the following key hyperparameters:

- **num_steps = 300**: This defines the number of training steps used in the Variational Inference (VI) procedure. A higher number of steps allows the model to better approximate the posterior distribution, potentially improving convergence.
- **Trend component = LocalLinearTrend**: This component models the underlying trend in the time series as a locally linear process with stochastic noise. It is suitable for capturing gradual but potentially non-stationary changes over time.
- **Optimizer = Adam (learning_rate = 0.1)**: The Adam optimizer is used to minimize the variational loss during training. A learning rate of 0.1 is relatively high, allowing the model to update weights more aggressively. While this can accelerate convergence, it must be balanced to avoid instability.
- **Posterior samples = 50**: After training, 50 samples are drawn from the surrogate posterior distribution. These samples are used to generate forecasts and quantify uncertainty in the predictions. A larger number of samples generally improves the stability and accuracy of probabilistic forecasts.

7. **Autoregressive Integrated Moving Average (ARIMA)**
ARIMA is an extension of the multiple linear regression model, widely used in time series analysis and forecasting, particularly effective for non-seasonal data [27].
The ARIMA model integrates three main components:
- **AR (AutoRegressive)**: Autoregression
- **I (Integrated)**: Differencing (to make the series stationary)
- **MA (Moving Average)**: Moving average

The ARIMA model is defined by three parameters[28]:
- p: the order of the autoregressive (AR) part, representing the number of lagged observations included in the model.
- d: the degree of differencing needed to make the time series stationary.
- q: the order of the moving average (MA) part, representing the number of past forecast errors used in the prediction.

The model is generally expressed as: **ARIMA(p, d, q)**.

The AR(p) Model The AR(p) model forecasts the current value based on a linear combination of its past values:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \tag{21}$$

Where:
- y_t: value of the series at time t
- c: constant term
- $\phi_1, \phi_2, \ldots, \phi_p$: autoregressive coefficients
- ε_t: white noise error term at time t

The I(d) Component Differencing is applied to the time series to make it stationary by removing trends and seasonality:

$$I(d) = \varDelta^d(y_t) \tag{22}$$

Where:
- $\varDelta^d$: the differencing operator of order d
- y_t: original time series

The MA(q) Model The MA(q) model describes the current value as a linear combination of past white noise error terms:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \tag{23}$$

Where:
- y_t: value of the series at time t
- μ: mean of the series (if applicable)
- ε_t: error term at time t
- $\theta_1, \theta_2, \ldots, \theta_q$: MA coefficients
- q: number of lagged forecast errors used in the model

Code Example

```
from pmdarima import auto_arima

model = auto_arima(ts_series, seasonal=False, trace=True, suppress_warnings=True)
print(model.summary())
```

Explanation

- `ts_series`: the input time series.
- `seasonal=False`: disables seasonal component (i.e., searches for ARIMA, not SARIMA).
- `trace=True`: prints the process of trying different parameter combinations.
- `auto_arima(...)`: automatically searches through many combinations of (p, d, q) parameters.

How `auto_arima` Chooses Parameters

(a) Iterates over all combinations of (p, d, q) within a specified range.
(b) Fits an ARIMA model for each combination.
(c) Selects the model with the lowest AIC (Akaike Information Criterion).

AIC Formula

$$\text{AIC} = 2k - 2\ln(\hat{L}) \tag{24}$$

Where:

- k: Number of parameters in the model
- $\hat{L}$: The maximum value of the likelihood function

8. **Vector Autoregression (VAR)** VAR is a statistical model commonly used for modeling and forecasting multivariate time series data. Unlike univariate autoregression (AR) models, which deal with single time series, VAR models handle multiple time series simultaneously, capturing the linear interdependencies among them[29].
VAR assumes that each variable in the system is a linear function of past values of itself and of all other variables in the system. It is widely applied in econometrics and financial time series analysis due to its flexibility and interpretability [30]. The main characteristics of VAR include:
   - Capturing the lagged impact of variables within the system
   - Modeling the interactions among multiple variables
   - Including feedback effects across the variables

   The general form of a VAR(p) model, where p is the lag order, for a $k \times 1$ vector time series $\mathbf{y}_t$ is given by:

$$\mathbf{y}_t = c + A_1\mathbf{y}_{t-1} + A_2\mathbf{y}_{t-2} + \cdots + A_p\mathbf{y}_{t-p} + \varepsilon_t \tag{25}$$

   Where:
   - $\mathbf{y}_t$: vector of k variables at time t (e.g., Open, High, Low, Volume, Close)
   - c: vector of intercept terms (constant)
   - A_i: coefficient matrices $(k \times k)$ for lag i, where $i = 1, 2, ..., p$
   - ε_t: white noise vector with zero mean and constant covariance matrix

Hyperparameters Used The training process of the VAR model involves the following important hyperparameters:

- **maxlags = 15**: This defines the maximum number of lag orders considered when selecting the best lag using the AIC (Akaike Information Criterion). The optimal lag is then determined automatically by the model based on this maximum.
- **Lag order (selected by AIC)**: The best lag order is selected using the AIC metric from the candidate lags up to `maxlags`. This dynamic selection helps balance model fit and complexity.

3.3 Performance Evaluation Metrics

To assess the performance of the motorcycle resale price prediction models listed above, the following metrics will be used: MAE, MSE, RMSE, MAPE, and R-squared.

- *Mean Absolute Error (MAE)*:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{26}$$

  MAE is an unbiased estimator of the expected absolute error. It measures the average L_1 distance between the actual and predicted values.[31]
- *Mean Squared Error (MSE):*

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{27}$$

  MSE is an estimator of the expected squared error. It measures the average L_2 distance between the actual and predicted values. MSE corresponds to using the squared error loss function, also known as L_2 loss, which is of the form $(y_i - \hat{y}_i)^2$. [31]
- *Root Mean Squared Error (RMSE):*

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{28}$$

  RMSE is the square root of MSE. The main purpose of taking the square root is to bring the metric back to the same units as the target variable (y_i), which makes interpretation more intuitive. Statistically, RMSE is the standard deviation of the residuals. Comparing RMSE across models can provide insight into the average prediction accuracy on the same scale as the original data. [31]
- *Mean Absolute Percentage Error (MAPE):*

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\% \tag{29}$$

  The mean_absolute_percentage_error (MAPE), also known as mean absolute percentage deviation (MAPD), is an evaluation metric for regression problems. The idea of this metric is to be sensitive to relative errors. It is for example not changed by a global scaling of the target variable.[31]
- *R-squared (R^2) - Coefficient of Determination:*

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{30}$$

  R^2 It represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.[31] As such variance is dataset dependent, may not be meaningfully comparable across different datasets. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected (average) value of y, disregarding the input features, would get an score of 0.

In these formulas, n is the total number of samples in the evaluation dataset (test set), y_i is the actual resale price of the i-th motorcycle, $\hat{y}_i$ is the resale price predicted by the model for the i-th motorcycle, and $\bar{y}$ is the mean of the actual resale prices y_i.

4 Result

Table 3. Forecasting Model Comparison per Data Split

Split	Model	MAE	MSE	RMSE	MAPE	R^2
75/10/15	**MLR**	**53.67**	**7,573.29**	**87.02**	**0.47%**	**0.99**
	SVR	82.32	13,539.96	116.36	0.73%	0.99
	KNN	254.67	147,766.71	384.40	2.28%	0.87
	LSTM	271.32	135,023.10	367.45	2.42%	0.88
	VAR	1004.94	1,579,015.22	1256.59	9.76%	-0.38
	BTST	2,252.89	9,346,081.00	3,057.14	19.16%	-7.15
	ARIMA	952.52	1,204,092.03	1,097.31	8.95%	-0.07
	Prophet	11,240.94	128,971,439.73	11,356.56	102.67%	111.46
70/10/20	**MLR**	**61.28**	**8,254.61**	**90.85**	**0.53%**	**0.99**
	SVR	84.75	12,804.20	113.16	0.74%	0.99
	KNN	254.58	135,177.57	367.67	2.24%	0.89
	LSTM	300.74	167,125.51	408.81	2.66%	0.87
	VAR	1397.44	2,777,246.50	1666.51	13.23%	-1.21
	BTST	3,906.74	25,592,042.00	5,058.86	34.18%	-19.39
	ARIMA	2,025.80	5,234,108.27	2,287.82	19.05%	-3.20
	Prophet	12,089.69	147,341,256.10	12,138.42	110.21%	-103.95
65/10/25	**MLR**	**68.59**	**9,752.91**	**98.76**	**0.57%**	**1**
	SVR	99.03	20,434.60	142.95	0.82%	0.99
	KNN	255.90	118,142.97	343.72	2.18%	0.95
	LSTM	712.56	782,914.00	884.82	6.34%	0.67
	VAR	946.54	1,233,433.18	1,110.60	8.4%	0.48
	BTST	4,967.82	29,260,142.00	5,409.26	44.68%	-11.28
	ARIMA	3,426.99	14,114,678.32	3,756.95	31.02%	-4.92
	Prophet	2,7669.62	767,709,461.49	2,7707.57	251.47%	-545.84

Among the evaluated models, **MLR** consistently demonstrates the best forecasting performance across all data splits. It achieves the lowest MAE, MSE, RMSE, and MAPE values, while maintaining an exceptionally high R^2 of 0.99 in every configuration.

Following MLR, **SVR** also shows strong and stable performance, with relatively low error metrics and similarly high R^2 values. **KNN** ranks next, performing reasonably well but with noticeably higher errors and slightly lower R^2 scores compared to MLR and SVR.

On the other end of the spectrum, both **BTST** and **Prophet** consistently yield the poorest results. These models exhibit significantly higher MAE, RMSE, and MAPE values, often accompanied by negative R^2 scores. Notably, under the 65/10/25 split, Prophet reaches a MAPE of 251.47% and an R^2 of –545.84, highlighting substantial forecasting inaccuracies.

MLR (Multiple Linear Regression) performs exceptionally well in terms of accuracy and stability, especially under the 75/10/15 split, achieving the lowest MAE (53.67), lowest RMSE (87.02), and the highest R^2 (0.99). This indicates that linear regression was surprisingly effective on this dataset despite its simplicity.

We present the top-performing models for each data split configuration:

- For the **75/10/15**, **70/10/20** and **65/10/25** splits, we show forecasting results for the three best-performing models: **MLR**, **SVR**, and **KNN**. These models consistently achieved the lowest error metrics in the respective configurations.
- In addition, we provide 30-day ahead forecasts of VIX stock prices for **MLR**, **SVR** and **KNN**. These forecasts are generated based on the data split where each model performed best, ensuring the most reliable extrapolation.

Multiple Linear Regression ratio 75/10/15

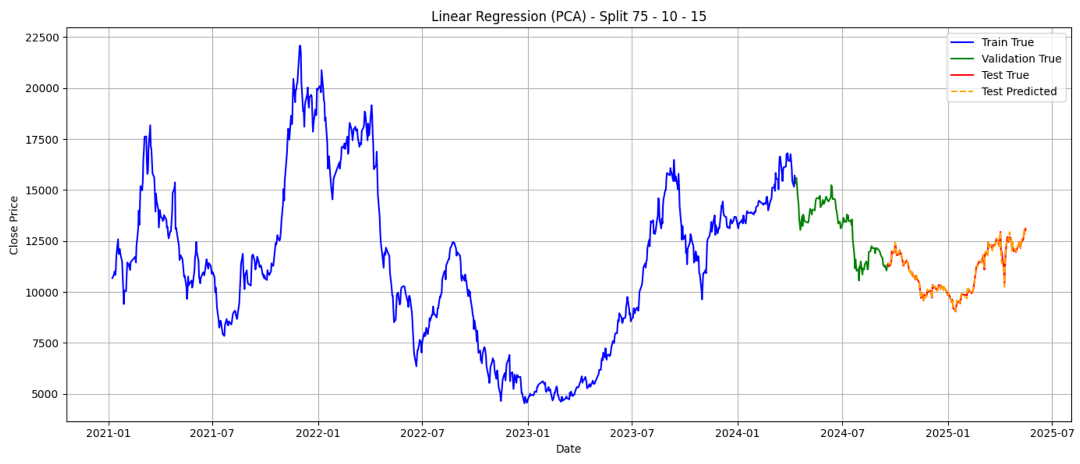

Fig. 2. Result of Multiple Linear Regression model

SVR ratio 75/10/15

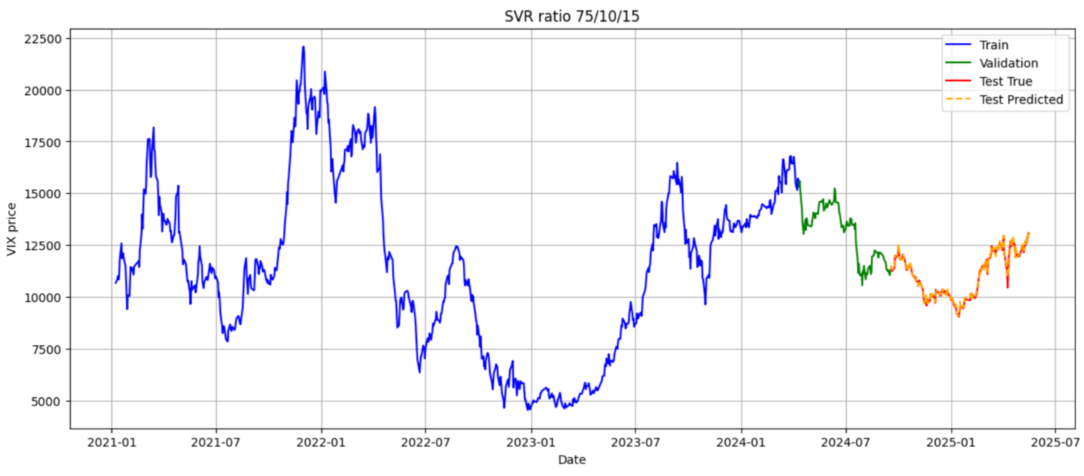

Fig. 3. Result of Support Vector Regression model

KNN ratio 75/10/15

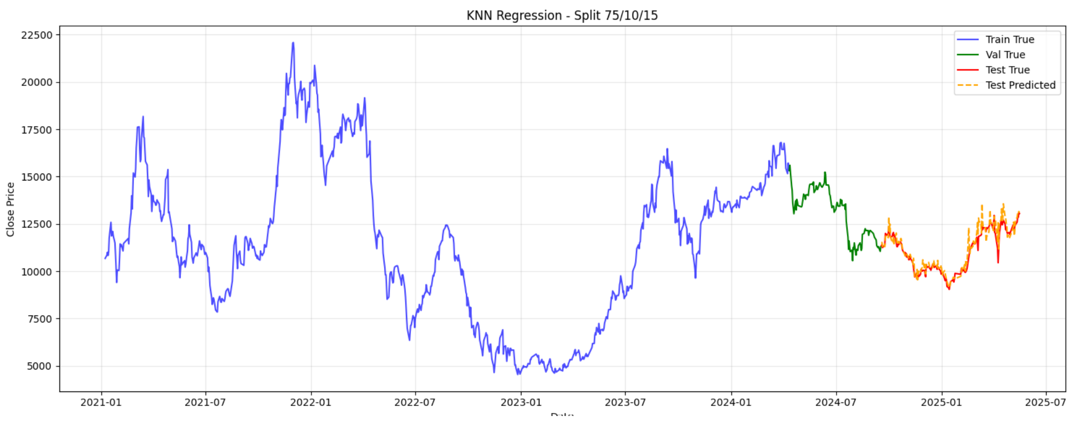

Fig. 4. Result of K-Nearest Neighbors model

Multiple Linear Regression ratio 70/10/20

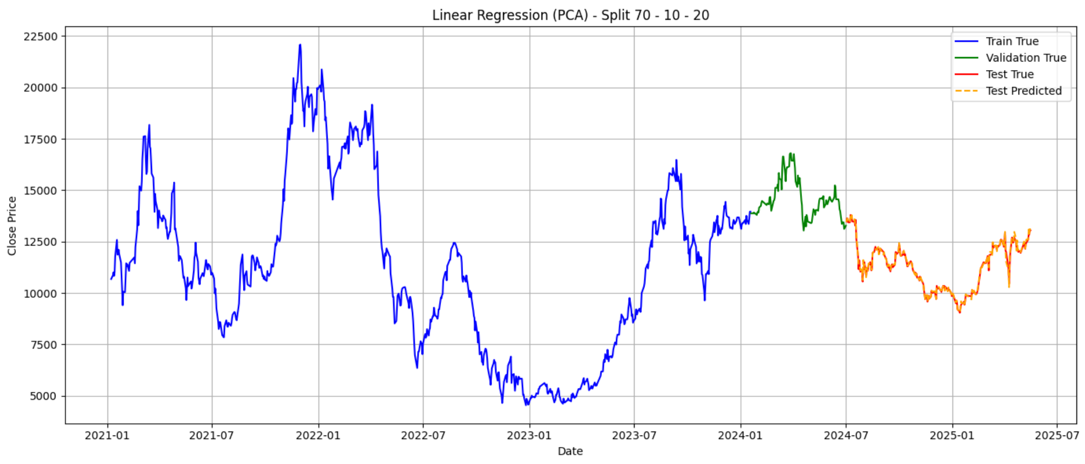

Fig. 5. Result of Multiple Linear Regression model

SVR ratio 70/10/20

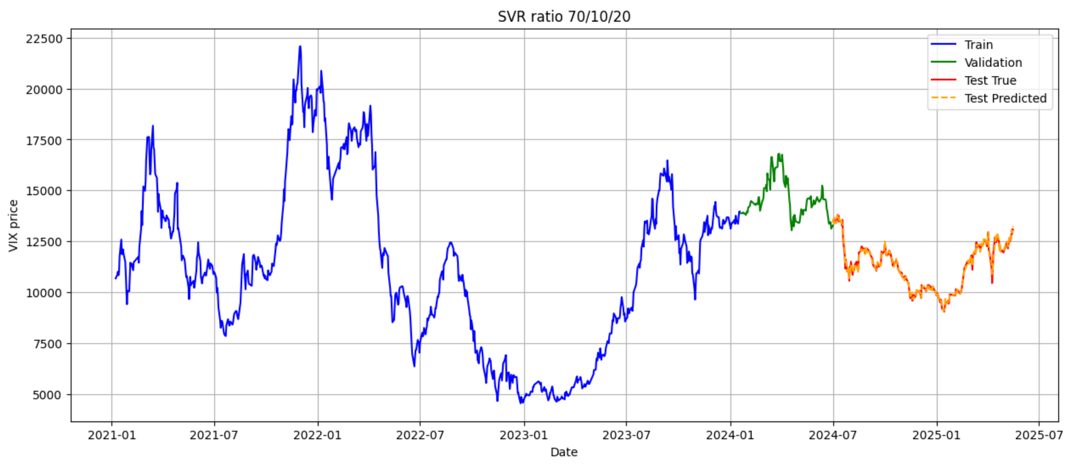

Fig. 6. Result of Support Vector Regression model

KNN ratio 70/10/20

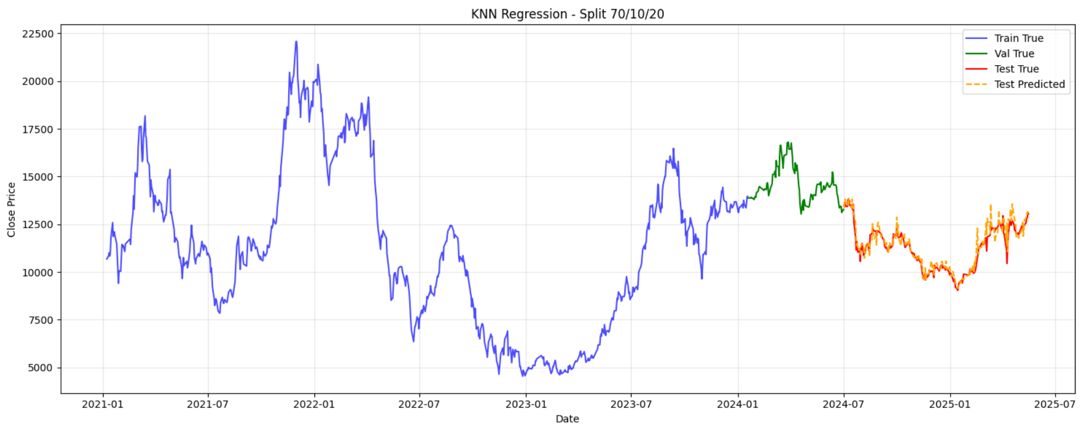

Fig. 7. Result of K-Nearest Neighbors model

Multiple Linear Regression ratio 65/10/25

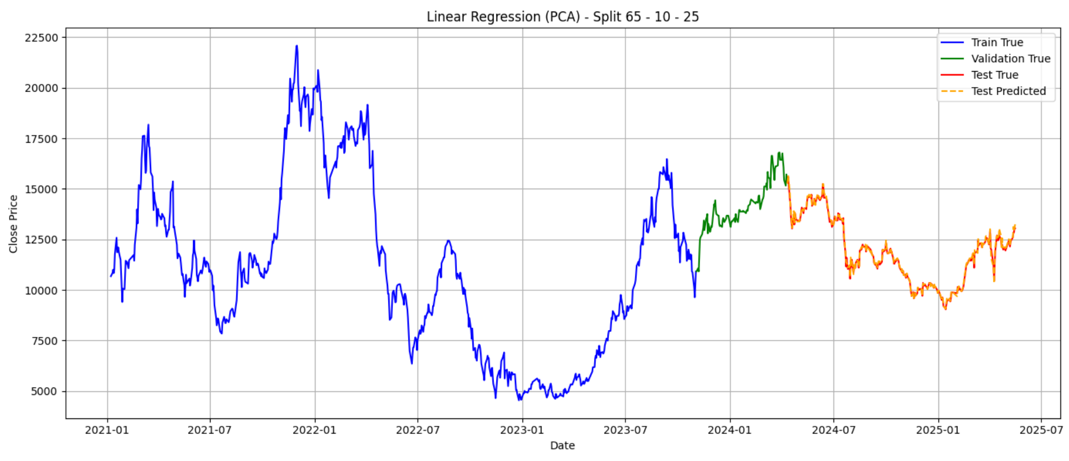

Fig. 8. Result of Multiple Linear Regression model

SVR ratio 65/10/25

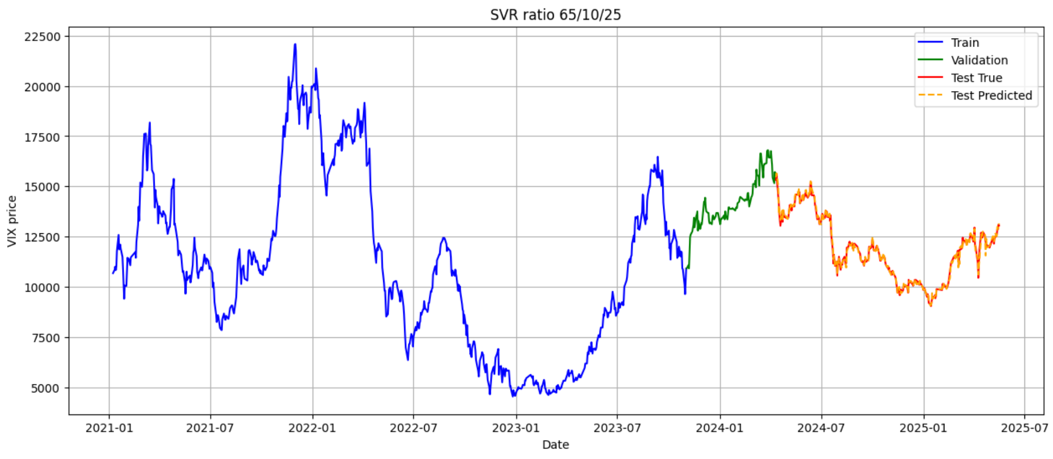

Fig. 9. Result of Support Vector Regression model

KNN ratio 65/10/25

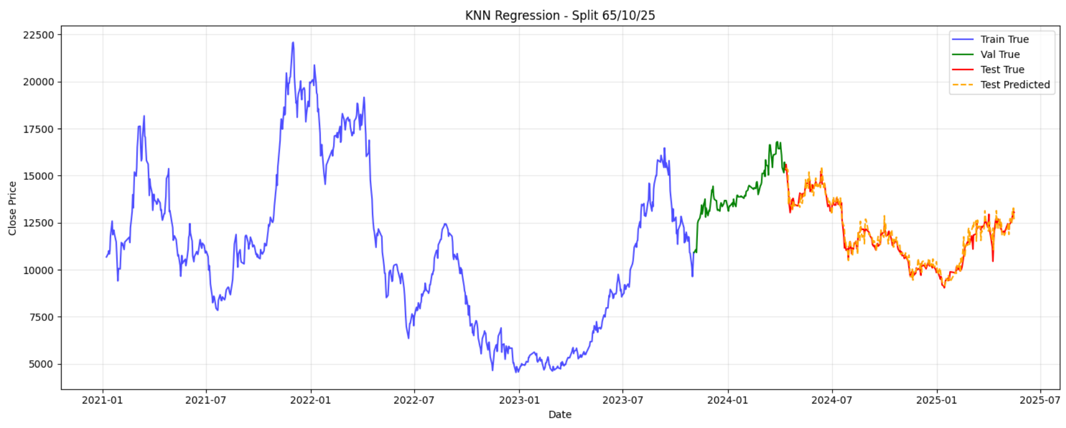

Fig. 10. Result of K-Nearest Neighbors model

MLR ratio 75/10/15 predict the next 30 days

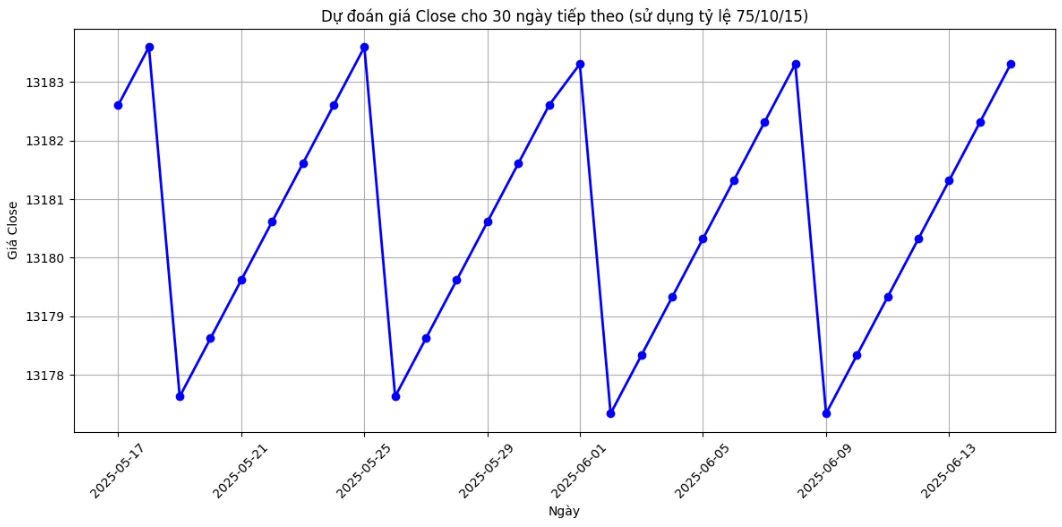

Fig. 11. VIX stock price prediction for the next 30 days with MLR

SVR ratio 75/10/15 predict the next 30 days

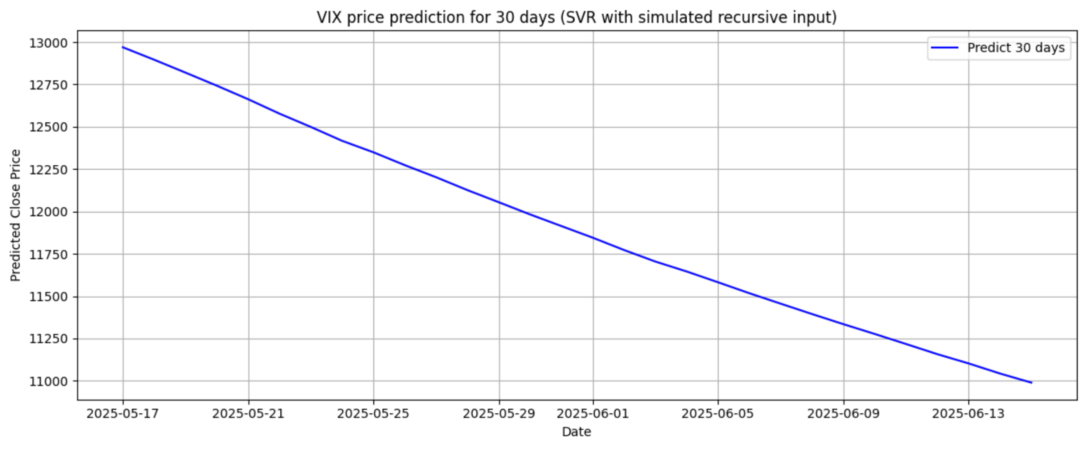

Fig. 12. VIX stock price prediction for the next 30 days with SVR

KNN ratio 65/10/25 predict the next 30 days

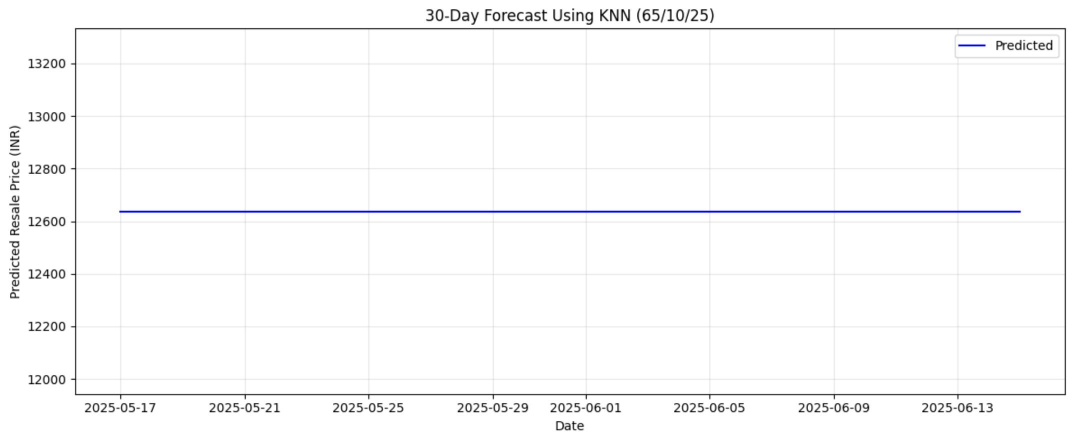

Fig. 13. VIX stock price prediction for the next 30 days with KNN

5 Conclusion

The fact that the three well-performing models (based on past evaluations)—**MLR**, **SVR**, and **KNN**—produce three different future trends does not diminish the value of the evaluation itself. Rather, it highlights the inherent uncertainty of forecasting in volatile financial environments like the stock market, where even strong models can offer differing perspectives on the future.

In the future, our team plans to continue improving the forecasting models by tuning hyperparameters, expanding experiments to other stocks, and integrating macroeconomic indicators such as interest rates, the CPI, and the VN-Index to enhance real-world applicability. We also plan to explore ensemble methods and deep learning architectures such as Transformers and attention-based models to improve long-term forecasting performance.

References

1. Y. Zhang, C. C. Aggarwal, and G. J. Qi, “Stock price prediction via discovering multi-frequency trading patterns,” in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (KDD)*, 2020, pp. 2379–2387.
2. R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Melbourne, Australia: OTexts, 2018. [Online]. Available: https://otexts.com/fpp2/
3. J. Hosker, S. Djurdjevic, H. Nguyen, and R. Slater, “Improving VIX futures forecasts using machine learning methods,” *SMU Data Science Review*, vol. 1, no. 4, art. 6, 2018. [Online]. Available: https://scholar.smu.edu/datasciencereview/vol1/iss4/6
4. Y. Zhai, “Comparison of the performance of different machine learning methods in predicting VIX volatility,” *Advances in Economics, Management and Political Sciences*, vol. 85, pp. 69–75, May 2024. [Online]. Available: https://www.ewadirect.com/proceedings/aemps/article/view/12676
5. Y. Bai and C. X. Cai, “Predicting VIX with adaptive machine learning,” *Global Association of Risk Professionals (GARP)*, White Paper, Jun. 14, 2021. [Online]. Available: https://www.garp.org/hubfs/Whitepapers/a2r5d000003IhxiAAC_RiskIntell.WP.PredictingVIX.ML.July8-1.pdf
6. N. T. T. Loan, D. N. Hung, and V. T. T. Van, “Application of ARIMA model and deep learning in forecasting stock price in Vietnam,” *Salud, Ciencia y Tecnologia – Serie de Conferencias*, vol. 5, pp. 1320–1329, Jan. 2025. [Online]. Available: https://www.researchgate.net/publication/385998474_Application_of_ARIMA_model_and_deep_learning_in_forecasting_stock_price_in_Vietnam
7. S. Gao, “Trend-based K-nearest neighbor algorithm in stock price prediction,” in *Proc. 3rd Int. Conf. Digital Economy and Computer Application (DECA 2023)*, Atlantis Press, pp. 746–756, Dec. 2023. [Online]. Available: https://doi.org/10.2991/978-94-6463-304-7_78
8. N. T. T. Tuan, T. H. Nguyen, and T. T. H. Duong, “Stock price prediction in Vietnam using stacked LSTM,” in *Intelligence of Things: Technologies and Applications (ICIT 2022), Lecture Notes on Data Engineering and Communications Technologies*, vol. 148, Cham: Springer, pp. 246–255, Aug. 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-15063-0_23
9. Investing.com, “Vincom Securities Historical Data,” *Investing.com.* [Online]. Available: https://www.investing.com/equities/vincomsc-historical-data [Accessed: May 25, 2025].
10. E. Elhariri, M. M. Gaber, and A. E. Hassanien, "Enhancing K-nearest neighbor algorithm: a comprehensive review and analysis," *Journal of Big Data*, vol. 11, no. 1, pp. 1–37, Apr. 2024. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-024-00973-y
11. A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2019.
12. F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org
13. G. P. Zhang, “Time series forecasting using a hybrid ARIMA and neural network model,” *Neurocomputing*, vol. 50, pp. 159–175, 2003
14. "Khoảng cách Euclid," Scholarship Wiki. [Online]. Available: https://wiki.scholarship.edu.vn/Kho%E1%BA%A3ng_c%C3%A1ch_Euclid.
15. Trevor Hastie, Robert Tibshirani, và Jerome Friedman." *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, vol. 7
16. V. N. Vapnik, *The Nature of Statistical Learning Theory.* New York, NY, USA: Springer, 1995. Available: https://doi.org/10.1007/978-1-4757-2440-0
17. A. J. Smola and B. Schölkopf, “A Tutorial on Support Vector Regression,” *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004. Available: https://doi.org/10.1023/B:STCO.0000035301.49549.88
18. Scikit-learn developers, “3.6.1. Support Vector Regression,” *Scikit-learn: Machine Learning in Python*, [Online]. Available: https://scikit-learn.org/stable/modules/svm.html#svr.
19. Facebook, “Prophet,” GitHub repository, 2017. [Online]. Available: https://github.com/facebook/prophet .
20. Facebook, “Prophet Documentation: Quick Start,” 2017. [Online]. Available: https://facebook.github.io/prophet/docs/quick_start.html.
21. S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
22. S. Saxena, “Learn About Long Short-Term Memory (LSTM) Algorithms,” *Analytics Vidhya*, Mar. 16, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.
23. N. Irawan and M. R. Sahlan, “Modeling Time Series Using Bayesian Structural Time Series (BSTS),” *Science and Statistics Journal*, Universitas Muhammadiyah Metro, Indonesia, 2024. [Online]. Available: https://scholar.ummetro.ac.id/index.php/sciencestatistics/article/download/5023/2 .
24. N. Irawan and M. R. Sahlan, “Modeling Time Series Using Bayesian Structural Time Series (BSTS),” *Science and Statistics Journal*, Universitas Muhammadiyah Metro, Indonesia, 2024. [Online]. Available: https://scholar.ummetro.ac.id/index.php/sciencestatistics/article/download/5023/2 .
25. Bean Machine, “Bayesian Structural Time Series Tutorial,” *BeanMachine Documentation.* [Online]. Available: https://beanmachine.org/docs/overview/tutorials/Bayesian_Structural_Time_Series/BayesianStructuralTimeSeries/ .

26. S. L. Scott and K. H. Brodersen, “Predicting the Present with Bayesian Structural Time Series,” *ResearchGate*, Aug. 2014. [Online]. Available: `https://www.researchgate.net/publication/264816307_Predicting_the_Present_with_Bayesian_Structural_Time_Series` .
27. R. Nau, “Introduction to ARIMA models,” Duke University, [Online]. Available: `https://people.duke.edu/~rnau/411arim.htm` .
28. E. Howell, “How To Forecast Time-Series Using Autoregression,” GitHub, [Online Notebook]. Available: `https://github.com/egorhowell/Youtube/blob/main/Time-Series-Crash-Course/17.%20ARIMA.ipynb`.
29. H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*. Berlin, Germany: Springer, 2005. [Online]. Available: `https://link.springer.com/book/10.1007/978-3-540-27752-1`
30. J. D. Hamilton, *Time Series Analysis*. Princeton, NJ, USA: Princeton University Press, 1994. [Online]. Available: `https://press.princeton.edu/books/hardcover/9780691042893/time-series-analysis`
31. Scikit-learn developers, “3.4.6. Regression Metrics,” *Scikit-learn: Machine Learning in Python*, [Online]. Available: `https://scikit-learn.org/stable/modules/model_evaluation.html#multilabel-ranking-metrics`.