Predicting Gas Prices with Machine Learning

Hoang Trung Dung*, Le Minh Kiet*, Le Tam Quang*,
Nguyen Vu Thuy*, Nguyen Tat Hung*

Hanoi University of Science and Technology
*These authors contributed equally. The order of authorship is arbitrary.

May 22, 2025

Abstract

Gas prices significantly influence daily life and the broader economy, making their analysis both relevant and impactful. Numerous studies have explored this domain, yet ongoing volatility and evolving data availability continue to motivate further investigation. In this work, we apply a range of machine learning models and mathematical theory to better understand and predict gas price behavior, drawing on concepts learned throughout our coursework. We gathered a custon dataset from 2003 to 2025 to evaluate several models and find that Linear SVM and Huber Regression delivered the most consistent results for 3-day forecasts, whereas Gaussian Process Regression shows signs of overfitting with the given dataset. A demonstration of our system is available via a Gradio interface. This report includes a decent amount of mathematical theory, since the authors want to make sure they really understand the tools at hand. This report is not only for the professor to evaluate, it is also for the undergraduate authors to solidify their own understanding.

1 Introduction

Although not everyone is involved in the stock market, nearly everyone is affected by the transportation system. Whether people drive, take public transit, bike, or walk, fuel plays a central role in daily life. Even those who avoid driving still rely on goods and services that are delivered using gas-powered vehicles.

There are over 58 million registered motorcycles and mopeds in Vietnam, with approximately 615 bikes per 1000 inhabitants. Gas prices, therefore, have a deep and direct impact on everyday life in Vietnam - we are heavily affected by fluctuations in fuel costs and all the instabilities that comes with them [3, 6].

For drivers in particular, small changes in fuel prices can add up over time. Being able to predict these changes could help people make better decisions about when to fill up and potentially save money. It could also benefit businesses involved in transportation, logistics, and retail [3].

Given the substantial economic benefits derived from precise forecasting, numerous methodologies have been explored. [7] used autoregressive moving average (ARMA) and artificial neural networks (ANNs), [8] used autoregressive integrated moving average (ARIMA) model with Texas data from EIA. In this project, we explore the use of machine learning techniques taught in the course to forecast gas prices. This report outlines the dataset we collected, the methods we used, and the results we obtained. Our goal is to assess how well certain predictive models can anticipate changes in fuel prices.

For the purpose of this course - and to make full use of what we've learned without stepping too far into advanced or future material - we've selected methods that are both appropriate and interpretable. Despite their relative simplicity, these approaches have yielded very respectable results.

The full implementation is available in our GitHub repository[1].

[1] https://github.com/NgHngK/IT-3190E-Machine-Learning-project

2 Methodology

This section outlines the mathematical and algorithmic foundations underlying the machine learning methods used in this project. It is intended to serve as a reference for understanding the performance and behavior of the models presented later in the report. In addition to documenting the technical basis of our approach, this section also serves as an opportunity for the undergraduate authors to solidify their understanding of the tools they are applying.

While the content here may be familiar to sufficiently proficient data analysts or machine learning practitioners, we encourage fellow undergraduate readers to review or skim through the material. Doing so may provide useful insights or reinforce key concepts relevant to the interpretation of our results. This section also lays the groundwork for the explanations and justifications that appear in the Results section.

2.1 Linear Regression

The first considered method is linear regression. It is a supervised machine learning algorithm that learns from the dataset and maps them with the most optimal linear relationship. It predicts a hyperplane (a line in 2D) to best fit the data points by minimizing a specific function. There are many practical applications for this model due to its simplicity in both understanding and implementation. Throughout years of development, linear regression is expanded to similar but more advanced models.

Linear regression is trained on a dataset, and the goal is minimizing the objective function (mostly means square error). The training dataset is often required to be linear, and the features should be independent from others.

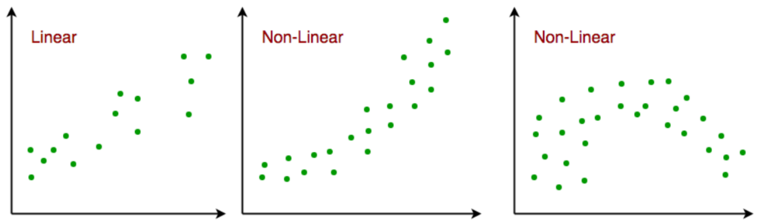


Figure 1: Distribution of data.

The prediction of the linear regression model is a hyperplane, which is a line in 2D and a plane in 3D. The formula for the hyperplane is given by:

$$f(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w_0 + w^T x$$

where w_0 is the bias, w_i is the weight of the i-th attribute in the dataset ($i \neq 0$), x_i is the value in the i-th column, $w = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix}^T$, and $x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$. Sometimes, we denote:

$$w = \begin{bmatrix} w_0 & w_1 & \dots & w_n \end{bmatrix}^T, \quad x = \begin{bmatrix} 1 & x_1 & \dots & x_n \end{bmatrix}^T,$$

so that $f(x, w) = w^T x$, which is for convenience.

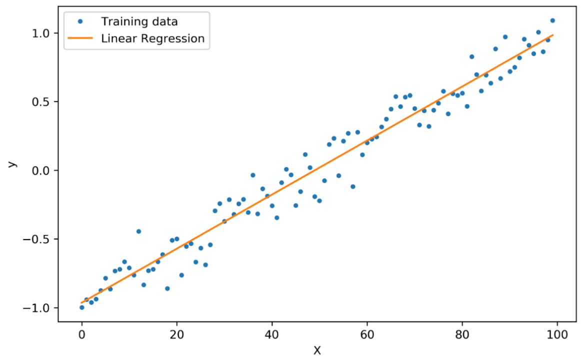


Figure 2: Best fit line in linear regression model.

We compute the vector w that minimizes the following function:

$$L(f, D) = \frac{1}{M}\text{RSS}(f) = \frac{1}{M}\sum_{i=1}^{M}(y_i - f(x_i, w))^2 \tag{1}$$

where $x_i = \begin{bmatrix}1 & x_{i1} & \dots & x_{in}\end{bmatrix}^T$ is the i-th training record and y_i is the i-th label.

Taking the derivative with respect to w on both sides of (1), we can compute the vector w^* that minimizes the mean square error. In fact, the optimal w^* is given by:

$$w^* = (A^TA)^{-1}A^Ty$$

where A is a matrix of size $M \times (n+1)$, with the i-th row of A being x_i, and $y = \begin{bmatrix}y_1 & y_2 & \dots & y_M\end{bmatrix}^T$ is the vector of combined labels. Note that this approach is valid if and only if A^TA is invertible.

2.2 Huber Regression

The major problem with many linear regression models is the essence of outliers in the dataset. These data points can disrupt the fitting line, leading to inaccurate predictions. This problem is solved by using Huber regression models, its objective function is the combination between mean absolute error and mean square error. In practice, outliers are recorded to have little effects on Huber regression, unlike the original form.

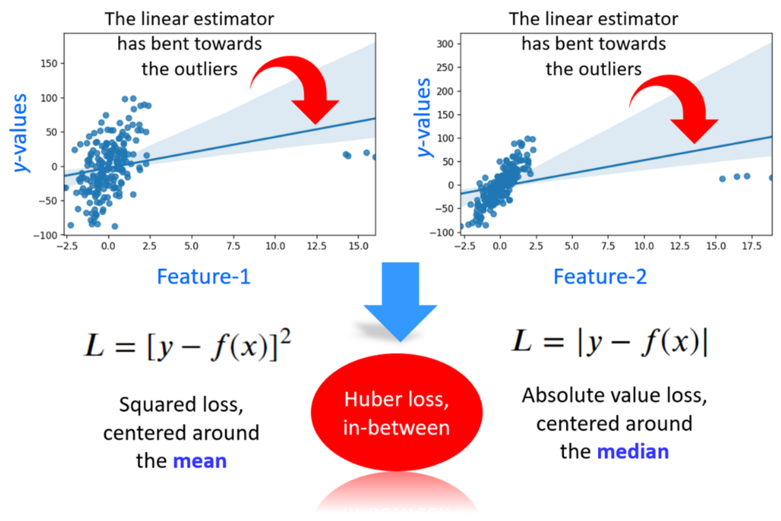


Figure 3: The outliers make the fitting lines inaccurate, so we need to combine mean absolute error and mean square error to tackle this issue.

The objective function we need to minimize in Huber regression is:

$$f_\delta(x, w) = \frac{1}{M}\sum_{i=1}^{M} L_\delta(y_i, x_i)$$

where the Huber loss L_δ is defined as:

$$L_\delta(y_i, x_i) = \begin{cases}(y_i - f(x_i, w))^2, & \text{if } |y_i - f(x_i, w)| \le \delta \\ \delta\left(|y_i - f(x_i, w)| - \frac{1}{2}\delta\right), & \text{otherwise}\end{cases}$$

Note that the absolute part of the above term was designed to guarantee continuous and differentiable properties as well as robustness and optimization efficiency.

The mean square error comes from the unbiased estimator around the mean whereas the mean absolute error comes from an unbiased estimator around the median value. Median is much more robust to outliers than mean, but it does not completely ignore them. So, this function balances the difference between these two types of error and use them to achieve better fitting lines.

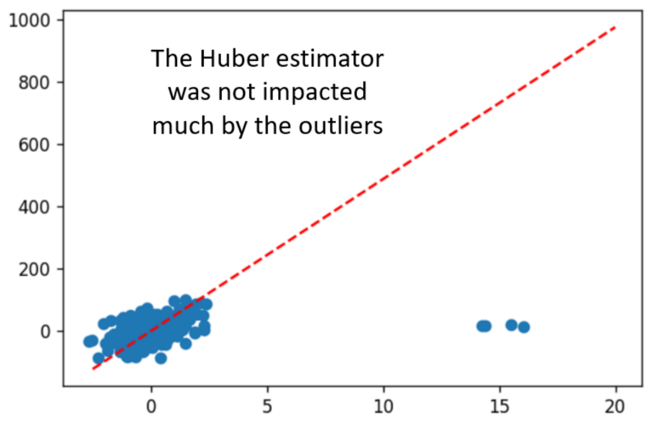


Figure 4: Huber regressor is not affected much by the outliers

2.3 Gaussian Process Regression

This section will provide a cursory overview of the key aspects of Gaussian Process Regression (GPR) . The primary focus of this section is to introduce the various kernels that will be used for our gas price prediction task. In GPR, we'll be working with the prior and the posterior of function values (or outputs). Given some known data $X = [x_1, x_2, x_3, \dots]$ and the corresponding function values $f = [f(x_1), f(x_2), \dots]$, GPR assumes that the prior follows a multivariate normal distribution with mean $\mu = [m(x_1), m(x_2), \dots]$, which is the mean function; and variance K, which is the kernel function:

$$P(f|X) = \mathcal{N}(f|\mu, K)$$

Now, let's introduce some newly observed data X^* and our goal is to predict $f(X^*)$, also denoted as f^*. The joint distribution of f and f^* is a multivariate normal distribution [29] given by:

$$\begin{bmatrix} f \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X^*) \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right)$$

where:

$$K = k(X, X), \quad K_* = k(X, X^*), \quad K_{**} = k(X^*, X^*)$$

The mean function is assumed to be $(m(X), m(X^*)) = 0$. This is good and all, but we need the conditional distribution for regression to work. Using marginal and conditional theorems and a lot of work, the result is obtained as:

$$f^*|f, X, X^* \sim \mathcal{N}\left(K_*^T K^{-1} f, K_{**} - K_*^T K^{-1} K_*\right)$$

This might seem like a lot, but fortunately, we'll be working on the computer, so this is mostly just foundational knowledge for when we let the computer handle all the heavy lifting. Essentially, given some data and a user-specified kernel, GPR will perform regression to find the optimal hyperparameters, and then find f^* for every point in the input space to predict or interpolate our targets.

Up until this point, I have been glossing over the concept of the kernel K, yet it appears everywhere in GPR. In simple terms, the kernel defines the similarity between two points. Each choice of kernel emphasizes certain types of similarities between data points, and this ultimately dictates the shape and behavior of the resulting fitting function. The following section will go in detail about the kernels that we will be using in our project:

2.3.1 Squared exponential (SE)

The squared exponential kernel, also known as the radial basis function (RBF) kernel, has become the de-facto default kernel for GPR and SVM. There are a few possible reasons for this preference. The SE kernel is a universal approximator [16], and can be integrated against most functions. And every function in its prior has infinitely many derivatives. This is just to say that any function drawn from GPR using the SE kernel is very smooth. The SE kernel takes the following form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

A key feature of the squared exponential (SE) kernel is that it cares about the Euclidean distance between two points. Points that are close together in the input space are considered highly similar, with a similarity close to 1. As the distance between points increases, their similarity decays rapidly towards 0. The lengthscale parameter ℓ controls the rate at which this decay occurs. And the output variance σ^2 governs the average distance of the function from its mean. This parameter is present in every kernel, serving as a scale factor that adjusts the overall magnitude of the function.

2.3.2 Rational quadratic

Now that we've talked about the SE kernel, a natural extension of it would be the rational quadratic kernel. This is equivalent to adding together many SE kernels with different lengthscales.

$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

As can be seen, this kernel also has the lengthscale parameter ℓ and the output variance σ^2. However, it includes an additional parameter α that determines the relative weighting of large-scale and small-scale variations. When $\alpha \to \infty$, the rational quadratic kernel becomes identical to the SE kernel.

2.3.3 Matern

The Matérn kernel is considered a generalization of the Squared Exponential (SE) kernel [23], with the parameter ν controlling the smoothness of the function modeled by the kernel. As $\nu \to \infty$, the Matérn kernel behaves similarly to the SE kernel. The Matérn is given by the following expression:

$$k_{\mathrm{Matern}}(x, x') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|x-x'|}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}|x-x'|}{\ell}\right)$$

where Γ is the Gamma function, K_ν is the modified Bessel function of the second kind, and ℓ and the aforementioned ν are positive parameters of the kernel. In our project we will be using a Matérn kernel with parameter 2.5 for a twice differentiable resulting function.

2.4 Decision Trees

A decision tree [20] is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called the *root* that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an *internal* or *test node*, and nodes without outgoing edges are called *leaves* (also known as *terminal* or *decision nodes*).

In a decision tree, each internal node splits the instance space into two or more subspaces according to a discrete function of the input attribute values. In the simplest and most common case, each test considers a single attribute, so the instance space is partitioned according to that attribute's value. For numeric attributes, the condition usually refers to a threshold or range.

We assume that a problem has a training set S, input feature set A, and target feature y, and is solved by growing a decision tree using some univariate splitting criterion. The tree may be pruned into a new tree T, and the process ends when a stopping criterion is met.

The growing phase continues until a stopping rule is triggered. Common stopping criteria include:

1. All instances in the training set belong to a single class label y.
2. The maximum tree depth has been reached.
3. The number of cases in a terminal node is less than the minimum number allowed for a parent node.
4. If the node were split, one or more resulting child nodes would contain fewer cases than the minimum allowed.
5. The best splitting criterion is not greater than a specified threshold.

2.4.1 Impurity-based Criteria

Given a random variable x with k discrete values, distributed according to $P = (p_1, p_2, \ldots, p_k)$, an impurity measure is a function $\varphi : [0, 1]k \to R$ that satisfies the following conditions:

- $\varphi(P) \geq 0$
- $\varphi(P)$ is minimized if $\exists i$ such that component $p_i = 1$
- $\varphi(P)$ is maximized if $\forall i,\ 1 \leq i \leq k,\ p_i = \frac{1}{k}$
- $\varphi(P)$ is symmetric with respect to the components of P
- $\varphi(P)$ is smooth in its range

Note that if the probability vector has a component equal to 1 (i.e., the variable x takes only one value), then the variable is said to be pure. On the other hand, if all components are equal, the level of impurity reaches its maximum. Given a training set S, the probability vector of the target attribute y is defined as:

$$P_y(S) = \left(\frac{|\sigma_{y=C_1}(S)|}{|S|},\ \ldots,\ \frac{|\sigma_{y=C_{|\mathrm{dom}(y)|}}(S)|}{|S|} \right)$$

The goodness-of-split due to a discrete attribute a_i is defined as the reduction in impurity of the target attribute after partitioning S according to the values $v_{i,j} \in \mathrm{dom}(a_i)$:

$$\Delta\varphi(a_i, S) = \varphi(P_y(S)) - \sum_{j=1}^{|\mathrm{dom}(a_i)|} \left(\frac{|\sigma_{a_i=v_{i,j}}(S)|}{|S|} \cdot \varphi\left(P_y\left(\sigma_{a_i=v_{i,j}}(S)\right)\right) \right)$$

2.4.2 Iterative Dichotomiser 3 (ID3)

ID3 is an algorithm of decision tree that focuses on classification problems and categorical attributes. ID3 uses impurity-based criteria such as Information gain and Gain ratio:

Information gain is an impurity-based criterion that uses the entropy measure (originating from information theory) as the impurity measure:

$$\Delta\varphi(a_i, S) = \mathrm{InformationGain}(a_i, S)$$
$$\varphi(P_y(S)) = \mathrm{Entropy}(y, S) = -\sum_{c_j \in \mathrm{dom}(y)} \frac{|\sigma_{y=c_j}(S)|}{|S|} \cdot \log_2 \left(\frac{|\sigma_{y=c_j}(S)|}{|S|} \right)$$

The gain ratio "normalizes" the information gain as follows [20]:

$$\mathrm{GainRatio}(a_i, S) = \frac{\mathrm{InformationGain}(a_i, S)}{\mathrm{Entropy}(a_i, S)}$$

Note that this ratio is not defined when the denominator is zero. Also, the ratio may tend to favor attributes for which the denominator is very small. Consequently, a two-stage procedure is suggested. First, we compute the information gain for all attributes. And then we consider only the attributes with information gain greater than or equal to the average gain, and select the one with the highest gain ratio.

2.4.3 Classification and Regression Trees (CART)

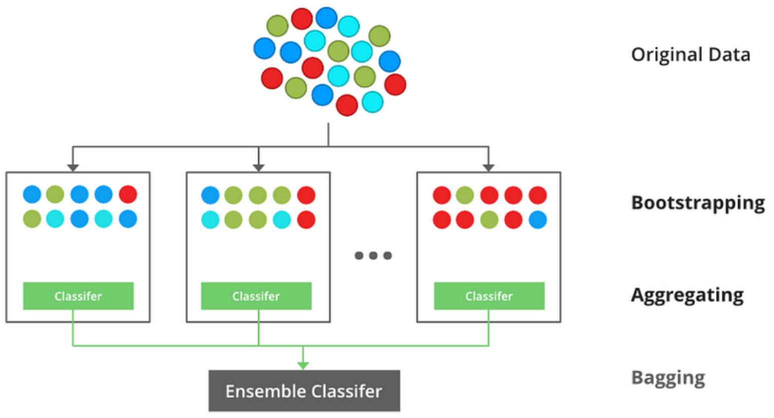
This inducer performs in binary trees and supports both classification and regression problems. For classification problems, it uses the Gini index as the splitting criterion. The Gini index is an impurity-based criterion that measures the divergence between the probability distributions of the target attribute's values.

$$\Delta\varphi(a_i, S) = \mathrm{GiniGain}(a_i, S)$$
$$\varphi(P_y(S)) = \mathrm{Gini}(y, S) = 1 - \sum_{c_j \in \mathrm{dom}(y)} \left(\frac{|\sigma_{y=c_j}(S)|}{|S|} \right)^2$$

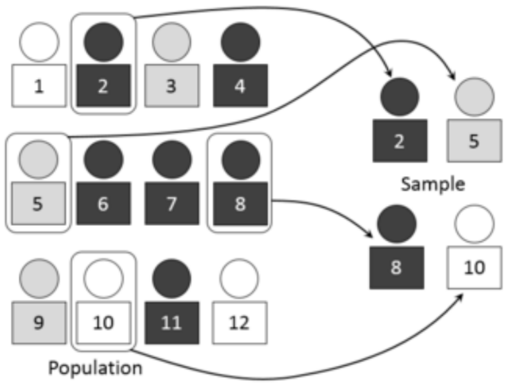
For regression problems, we use residual reduction. Residual reduction is a measure of how much the average squared difference between the predicted values and the actual values for the target variable is reduced by splitting the subset. The lower the residual reduction, the better the model fits the data.

2.5 Bagged Trees

Bagged Trees [2] is an ensemble learning method using bootstrapping and aggregating, for purposes of minimizing the effects of bias and improving reliability of predictions.



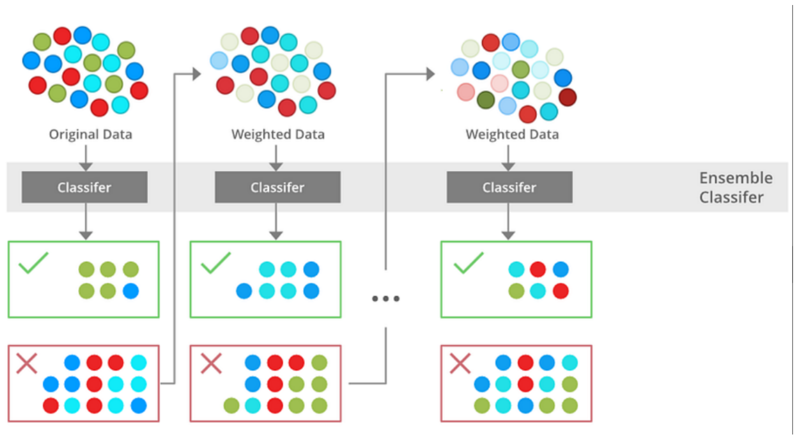
First, we have bootstrapping [5]: Given a dataset D of size n, bagging generates m new training sets D_i, each of size n'. Each new dataset is created by sampling from the original dataset (sampling). Each sample in the original dataset is chosen with a probability following a population distribution (e.g., Gaussian distribution).



Next, in the aggregating step, we use stump trees to train new datasets. Stump trees are trees that have starting node and leaves (depth equal 1), this is considered as weak learners. And result given by these learners will be overwhelmed voting if classification, otherwise will be averaged.

2.6 Boosting Trees

"Boosting" [22] is a general method for improving the performance of any learning algorithm. In theory, boosting can be used to significantly reduce the error of any "weak" learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing.



To apply the boosting approach, we start with a method or algorithm for finding the rough rules of thumb. The boosting algorithm calls this “weak” or “base” learning algorithm repeatedly, each time feeding it a different subset of the training examples (or to be more precise, a different distribution or weighting over the training examples). Each time it is called, the base learning algorithm generates a new weak prediction rule, and after many rounds, the boosting algorithm must combine these weak rules into a single prediction rule that, hopefully, will be much more accurate than any one of the weak rules.

We will focus on a specific boosting algorithm, called Adaptive Boost (AdaBoost) [9]:

Algorithm 1 AdaBoost

```
Require: Training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, where $y_i \in \{-1, +1\}$
Require: Number of iterations $T$
 1: Initialize weights $w_i \leftarrow \frac{1}{m}$ for $i = 1, \dots, m$
 2: for $t = 1$ to $T$ do
 3:     Train weak learner $h_t$ using weights $w_i$
 4:     Compute error $\epsilon_t = \sum_{i=1}^{m} w_i \cdot \mathbb{I}(h_t(x_i) \neq y_i)$
 5:     if $\epsilon_t > 0.5$ then
 6:         break
 7:     end if
 8:     Compute $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
 9:     for $i = 1$ to $m$ do
10:         $w_i \leftarrow w_i \cdot \exp(-\alpha_t y_i h_t(x_i))$
11:     end for
12:     Normalize weights: $w_i \leftarrow \frac{w_i}{\sum_{j=1}^{m} w_j}$
13: end for
14: return Final classifier: $H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$
```

For the purposes of the project, we will be using boosting trees with a maximum depth of 3, and bagged trees with a maximum depth of 10.

2.7 Support Vector Machines

Support Vector Machine (SVM) works by finding the optimal boundary - called a hyperplane - that best separates data points belonging to different classes. This boundary is chosen so that it maximizes the margin, or distance, between the closest points of the classes and the hyperplane. These closest points are known as support vectors, and they play a crucial role in defining the position and orientation of the hyperplane.

SVM can use the kernel trick to transform data into higher dimensions, making it possible to find a separating hyperplane even when the original data is not linearly separable. However, in this section, we will focus on the linear SVM, which assumes that the data can be separated by a straight line (or a flat hyperplane in higher dimensions).

In linear SVM, the goal is to find the best linear decision boundary that separates two classes of data points. There are two main variants of linear SVM, depending on how strictly the data must be separated:

1. The hard margin SVM is used when the data is perfectly linearly separable. It is highly sensitive to noise and outliers. Just a single misclassified point can make it impossible to find a separating hyperplane, making this approach impractical for most real-world data.

2. To overcome the limitations of the hard margin, the soft margin SVM allows some flexibility by permitting a small number of misclassifications or margin violations. It introduces a parameter (commonly denoted as C) that controls the trade-off between maximizing the margin and minimizing classification error. It is more suitable for real-world datasets that are not perfectly separable.

2.7.1 Hard Margin SVM

a. Primal problem

The goal of support vector machines (SVM) is to find a hyperplane $w^T x + b = 0$ that separates the classes while maximizing the minimum distance (margin) from the data points. For a training set $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$, the distance from a point x_i to the hyperplane is given by $\frac{|w^T x_i + b|}{\|w\|}$ [28].

By scaling w and b such that the closest points satisfy $|w^T x_i + b| = 1$, the margin becomes $\frac{2}{\|w\|}$. Therefore, maximizing the margin is equivalent to minimizing $\|w\|$, subject to correct classification. Concretely, the primal optimization problem is:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 \tag{1}$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, n \tag{2}$$

Here, $\|w\|$ denotes the Euclidean norm. Each constraint $y_i(w^T x_i + b) \geq 1$ ensures that x_i lies on or outside the margin boundaries. Geometrically, the margin boundaries are defined by $w^T x + b = \pm 1$, and the support vectors (the closest data points) satisfy $y_i(w^T x_i + b) = 1$ [28].

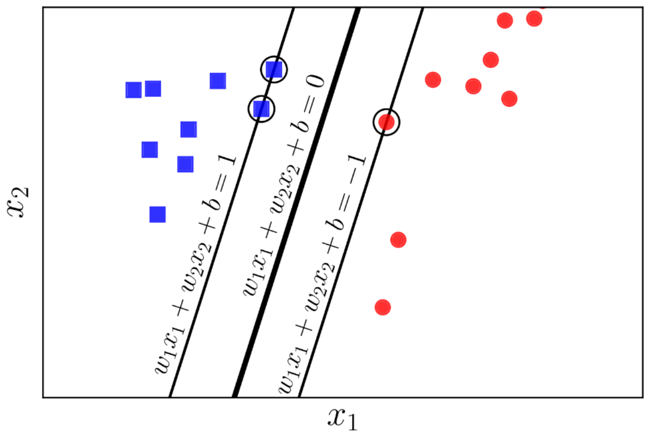

$w_1x_1 + w_2x_2 + b = 1$
$w_1x_1 + w_2x_2 + b = 0$
$w_1x_1 + w_2x_2 + b = -1$
x_2
x_1

Figure 5: This figure illustrates a hard-margin support vector machine (SVM) in $\mathbb{R}^2$. The bold line represents the decision boundary defined by $w^T x + b = 0$, while the dashed lines denote the margin boundaries given by $w^T x + b = \pm 1$. Blue points correspond to the positive class ($y = +1$), and red points represent the negative class ($y = -1$). Only the support vectors—indicated as solid points on the margin boundaries—lie exactly at a perpendicular distance of $\frac{1}{\|w\|}$ from the decision boundary. Since the total margin width is $\frac{2}{\|w\|}$, minimizing $\|w\|^2$ serves to maximize this separation [14].

b. Dual problem

Introducing Lagrange multipliers $\alpha_i \geq 0$ for each constraint leads to the dual formulation of the SVM problem. By taking derivatives of the Lagrangian and setting them to zero, i.e., $\frac{\partial L}{\partial w} = 0$ and $\frac{\partial L}{\partial b} = 0$, one obtains the conditions:

$$w = \sum_{i=1}^n \alpha_i y_i x_i \quad \text{and} \quad \sum_{i=1}^n \alpha_i y_i = 0.$$

Substituting into the Lagrangian leads to the dual optimization problem [30]:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$\text{subject to} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0. \tag{2}$$

The dual problem depends only on the inner products $x_i \cdot x_j$, which allows the use of kernels to implicitly operate in high-dimensional feature spaces. Solving the dual yields the same w as before [21]:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

Importantly, most α_i are zero—only the non-zero α_i correspond to the support vectors [26], i.e., points on the margin that actively define the classifier.

c. KKT Conditions

The Karush–Kuhn–Tucker (KKT) conditions ensure the equivalence between the primal and dual formulations of the support vector machine (SVM) optimization problem. For an optimal solution (w^*, b^*, α^*), the KKT conditions include:

- Stationarity: Taking derivatives of the Lagrangian and setting them to zero gives [21]:

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i \tag{3}$$

$$\sum_{i=1}^{n} \alpha_i^* y_i = 0 \tag{4}$$

- Primal feasibility:

$$y_i(w^\top x_i + b) \geq 1, \quad \forall i \tag{3}$$

- Dual feasibility:

$$\alpha_i^* \geq 0, \quad \forall i \tag{4}$$

- Complementary slackness:

$$\alpha_i^* \left[y_i(w^\top x_i + b) - 1 \right] = 0, \quad \forall i \tag{5}$$

From complementary slackness (5), any point with $\alpha_i^* > 0$ lies exactly on the margin, i.e., satisfies $y_i(w^\top x_i + b) = 1$. These are the support vectors. Any point strictly outside the margin has $\alpha_i = 0$, meaning it does not affect the decision boundary [26].

d. Objective formula

After obtaining α from equation (2), one can deduce w from (3) and b from (5) and (4). It is clear that we only need to care about $\alpha_i \neq 0$.

Let $S = \{i : \alpha_i \neq 0\}$ and N_s be the number of elements in the set S. For each $i \in S$, we have:

$$1 = y_i(w^\top x_i + b) \iff b + w^\top x_i = y_i \tag{8}$$

Although for any pair (x_i, y_i), one could immediately deduce b if one knew w, a different method for calculating b is often used and deduced to be more numerically stable:

$$b = \frac{1}{N_s} \sum_{i \in S} \left(y_i - w^\top x_i \right) = \frac{1}{N_s} \sum_{i \in S} \left(y_i - \sum_{j \in S} \alpha_j y_j x_j^\top x_i \right) \tag{9}$$

Previously, w was deduced to be:

$$w = \sum_{j \in S} \alpha_j y_j x_j \tag{10}$$

according to (3).

Observation: To determine which class a new point x belongs to, one needs to determine the sign of the expression [14]:

$$w^\top x + b = \sum_{j \in S} \alpha_j y_j x_j^\top x + \frac{1}{N_s} \sum_{i \in S} \left(y_i - \sum_{j \in S} \alpha_j y_j x_j^\top x_i \right) \tag{11}$$

2.7.2 Soft Margin SVM

a. Primal problem

When data are not perfectly separable, slack variables $\xi_i \geq 0$ allow margin violations. The soft-margin primal adds a penalty for misclassification:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \end{aligned} \tag{6}$$

Here, $C > 0$ is a regularization parameter controlling the tradeoff: Larger C enforces fewer slack violations. Geometrically, some points are allowed within or beyond the margin at cost $C \cdot \xi_i$, but we still aim to keep them close to correct classification.

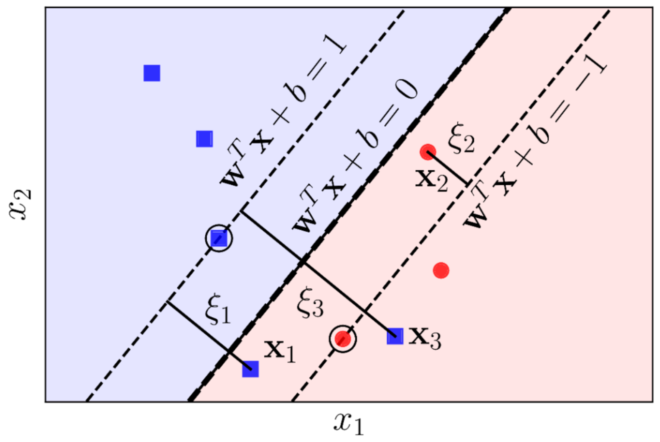

$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$
ξ_1
ξ_2
ξ_3
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
x_1
x_2

Figure 6: Illustration of the slack variables ξ_i. For points that lie in the safe region mathematically, $\xi_i = 0$. For points that lie in the unsafe region compared to the corresponding separating boundary, such as $0 < \xi_i < 1$, for example x_2. For points that lie on the wrong side of the class compared to the separating boundary, such as $\xi_i > 1$, for example x_1 and x_3 [15].

b. Dual problem

The Lagrangian dual for the soft-margin SVM turns out to have the same quadratic objective as before, but with box constraints on α_i. One obtains:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C \quad (i = 1, \ldots, n) \end{aligned} \tag{7}$$

As detailed in standard SVM derivations, the dual is identical to the hard-margin case except that each α_i is upper-bounded by C. The solution again gives $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ and remains sparse. The advantages of the dual are the same as before: only inner products appear (allowing kernels), and the sparsity ($\alpha_i > 0$ for only a few points) still holds.

c. KKT Conditions

At optimality $(w^*, b^*, \xi^*, \alpha^*, \mu^*)$, the KKT conditions [21] include:

- **Stationarity:**

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i^* y_i = 0 \tag{8}$$

- **Primal feasibility:**

$$y_i(w^T x_i + b) \geq 1 - \xi_i^* \quad \text{and} \quad \xi_i^* \geq 0 \tag{9}$$

- **Dual feasibility:**

$$0 \leq \alpha_i^* \leq C \quad \text{and} \quad \mu_i^* \geq 0 \tag{10}$$

  (where μ_i^* is the Lagrange multiplier for $\xi_i^* \geq 0$)

- **Complementary slackness:**

$$\alpha_i^*[y_i(w^T x_i + b) - 1 + \xi_i^*] = 0 \quad \text{and} \quad \mu_i^* \xi_i^* = 0 \tag{11}$$

These conditions imply the following familiar support-vector rules [13]:

- If $0 < \alpha_i^* < C$, then $\xi_i^* = 0$ and $y_i(w^T x_i + b) = 1$ (The point lies exactly on the margin).
- If $\alpha_i^* = C$, then $y_i(w^T x_i + b) \leq 1$ (The point is either on or inside the margin, possibly misclassified).
- If $\alpha_i^* = 0$, then $y_i(w^T x_i + b) \geq 1$ (The point lies strictly outside the margin).

d. Objective formula

Let $M = \{i : 0 < \alpha_i < C\}$ and $S = \{j : 0 < \alpha_j \leq C\}$. That is, M is the set of indices of points that lie exactly on the margins—or support the computation of b, and S is the set of indices of the support vectors—or support the computation of w. Similar to the case of Hard-Margin SVM, the parameters w and b can be determined as follows:

$$w = \sum_{j \in S} \alpha_j y_j x_j \tag{12}$$

$$b = \frac{1}{N_M} \sum_{i \in M} (y_i - w^T x_i) = \frac{1}{N_M} \sum_{i \in M} \left(y_i - \sum_{j \in S} \alpha_j y_j x_j^T x_i \right) \tag{13}$$

Note that the ultimate goal remains to determine the label for each new point, which does not have to be calculating w and b to quantify the optimal way to determine the values of the expression after substituting a new point x [15]:

$$w^T x + b = \sum_{j \in S} \alpha_j y_j x_j^T x + \frac{1}{N_M} \sum_{i \in M} \left(y_i - \sum_{j \in S} \alpha_j y_j x_j^T x_i \right) \tag{14}$$

2.8 Polynomial Regression

Polynomial regression is an extended version of linear regression that adds power terms of the original predictor variables, enabling the fitted model to trace smooth curves rather than straight lines [4] of simple linear regression as shown the difference in Figure 1. Because it remains linear in the unknown coefficients, polynomial regression keeps the interpretability, closed-form solutions, and well-understood inference machinery of classical linear models while gaining the flexibility to capture systematic, nonlinear patterns that a purely linear specification would miss [11]. Practitioners reach for it when scatter plots suggest curvature, when engineering systems follow known polynomial laws (e.g., projectile motion, beam deflection), or when a low-degree polynomial can approximate an unknown smooth function more parsimoniously than a high-dimensional non-parametric technique [4]. With careful choice of degree – guided by theory, cross-validation, or information criteria – polynomial regression often delivers a sweet spot between underfitting straight lines and the variance inflation of more complex models, while preserving the familiar diagnostics and confidence intervals of the linear-model toolkit [12].

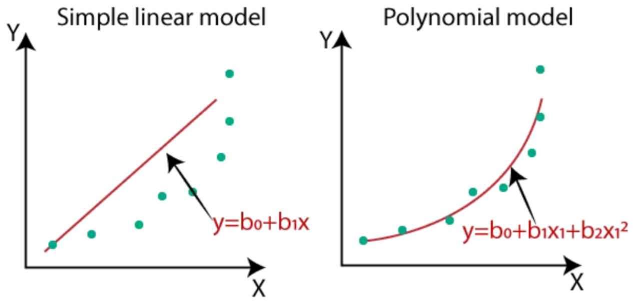


Figure 7: Linear Regression vs Polynomial Regression

2.8.1 The multiple linear regression model

The basic multiple regression model of a dependent variable Y on a set of k independent variables $X_1, X_2, \dots, X_n$ can be expressed as Equation 15 [18].

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_k x_{1k} + \varepsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_k x_{2k} + \varepsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_k x_{nk} + \varepsilon_n \end{aligned} \tag{15}$$

where y_i is the value of the dependent variable y for the i^{th} case, x_{ij} is the value of the j^{th} independent variable x_j for the i^{th} case. β_0 is the Y-intercept of the regression surface, each β_j, $j = 1, 2, \dots, k$, is the slope of the regression surface with respect to variable x_j, and ε_i is the random error component for the i^{th} case. In Equation 15, we have n observations and k predictors $(n > k + 1)$.

In matrix notation, we can rewrite model 15 as Equation 16.

$$Y = X\beta + \varepsilon \tag{16}$$

where response vector Y and error vector ε are column vectors of length n. The vector of parameters β is a column vector of length $k+1$. The matrix X has n rows and $k+1$ columns (with its first column having all elements equal to 1, the second column being filled by the observed values of X_1, etc.). We want to estimate the unknown values of β and ε.

2.8.2 Simple polynomial regression

Polynomial Regression is a model used when the response variable is non-linear, i.e., the scatter plot gives a non-linear or curvilinear structure [10]. The general equation form for polynomial regression is of the form in Equation 17.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2^2 + \dots + \beta_k x_k^k + \varepsilon \tag{17}$$

To solve the problem of polynomial regression, it can be converted into Equation 15 of multivariate linear regression with k regressor variables of the form in Equation 18.

$$y = \beta_0 + \beta_1 z_1 + \beta_2 z_2 + \dots + \beta_k z_k + \varepsilon \tag{18}$$

Where $z_i = x^i$. So, at the end, Equation 17 can be transformed into Equation 16 and use the same method as linear regression to find the weights.

2.8.3 Multivariate polynomial regression

The general form of multivariate polynomial regression is shown in Equation 19 [25].

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k}\sum_{j=1}^{k} \beta_{ij} x_i x_j + \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} \beta_{ijl} x_i x_j x_l + \ldots + \varepsilon \tag{19}$$

A second order multiple polynomial regression can be expressed as Equation 20.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{12} x_1 x_2 + \beta_{22} x_2^2 + \varepsilon \tag{20}$$

Where:

- β_0 : Intercept
- β_1, β_2 : Linear effect parameters
- β_{11}, β_{22} : Quadratic effect parameters
- β_{12} : Interaction effect parameter

Equation 20 is converted to matrix form as Equation 16 to find the weights.

2.8.4 Ordinarily least squares

In order to find the optimal weights when training regression models, the residual sum of squares [24], as shown in Equation 21, is used.

$$\mathrm{RSS}(f) = \sum_i [y_i - f(x)]^2 = (Y - X\beta)^T (Y - X\beta) \tag{21}$$

Where X is the independent input matrix, Y is the dependent output matrix, and β is the matrix of weights.

The derivative is applied, as in Equation 22, to calculate the local minimum. The local minimum is found when the derivative value is zero.

$$\frac{\partial\,\mathrm{RSS}(f)}{\partial\beta} = 0 \quad \Rightarrow \quad -2X^T(Y - X\beta) = 0 \quad \Rightarrow \quad \beta = (X^T X)^{-1} X^T Y \tag{22}$$

By solving this optimization problem, we obtain the optimal vector β.

3 Dataset

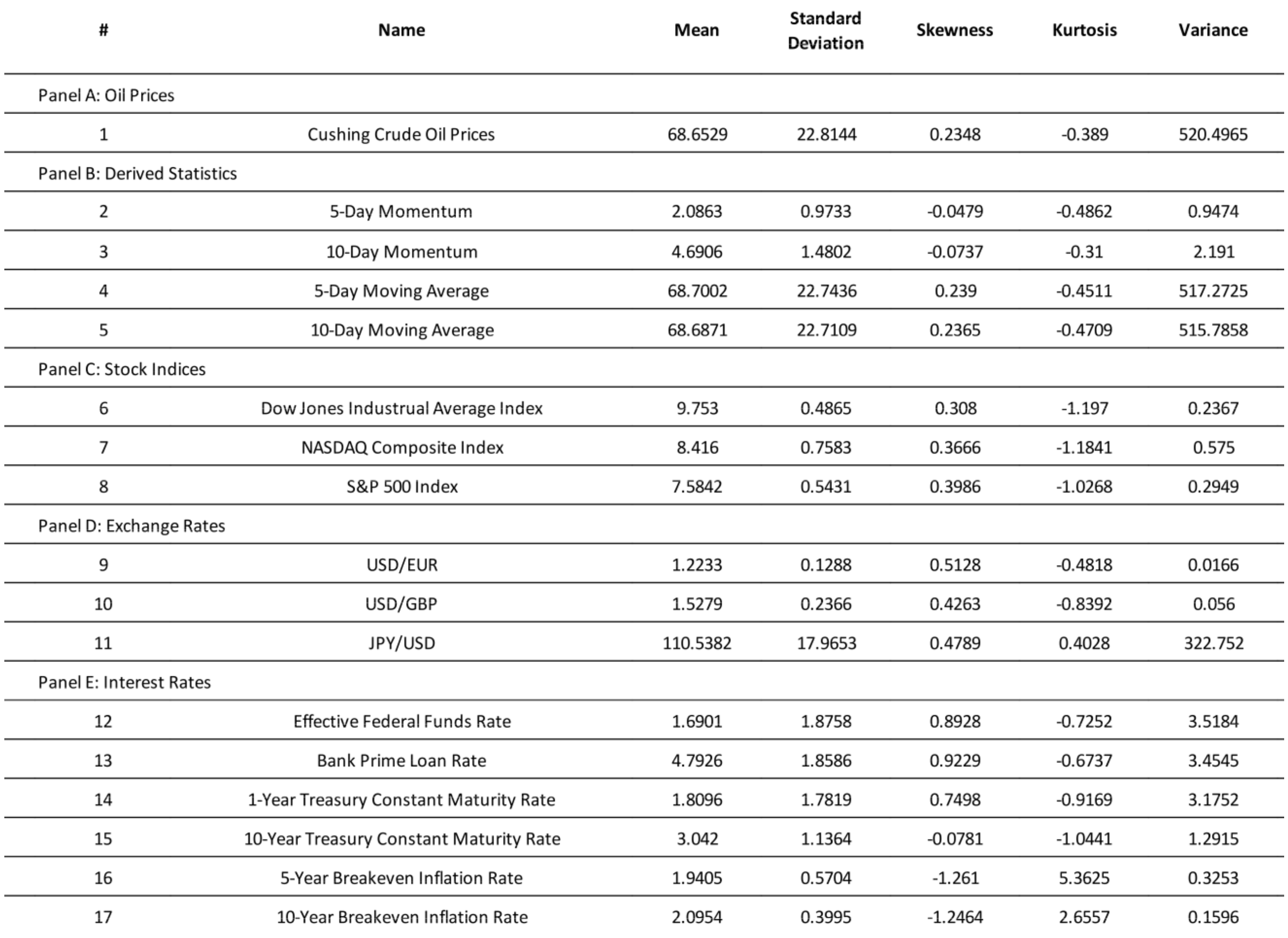
3.1 Raw Attributes

We obtained a set of 16 explanatory variables that were selected based on the relevant literature (for instance, [1, 17, 27]). Three of the attributes are stock markets such as Dow Jones, NASDAQ Composite, S&P 500. Others are usual rates related to United State like currency exchange rate, inflation rate, etc. The selected variables were then fed into the forecasting models through a training–testing learning process, resulting in an efficient and less error-prone models for natural gas price forecasting.

To support this modeling process, we searched for publicly available datasets from organizations in the world responsible for reporting data records of economic matters, rates like exchange rates, inflation rates, currency. In particular, we focused on two key institutions: the U.S. Energy Information Administration (EIA) and the Federal Reserve Economic Data (FRED).

- The U.S. Energy Information Administration (EIA) is a principal agency of the U.S. Federal Statistical System responsible for collecting, analyzing, and disseminating independent and impartial energy information. EIA provides comprehensive datasets on various aspects of the energy sector, including crude oil production, inventories, imports and exports, consumption trends, and energy prices.

- The Federal Reserve Economic Data (FRED) platform is a comprehensive online database maintained by the Federal Reserve Bank of St. Louis, offering access to hundreds of thousands of U.S. and international time series datasets. FRED covers a broad range of economic indicators, including interest rates, inflation, employment statistics, GDP figures, and commodity price indexes.



#	Name	Mean	Standard Deviation	Skewness	Kurtosis	Variance
Panel A: Oil Prices						
1	Cushing Crude Oil Prices	68.6529	22.8144	0.2348	-0.389	520.4965
Panel B: Derived Statistics						
2	5-Day Momentum	2.0863	0.9733	-0.0479	-0.4862	0.9474
3	10-Day Momentum	4.6906	1.4802	-0.0737	-0.31	2.191
4	5-Day Moving Average	68.7002	22.7436	0.239	-0.4511	517.2725
5	10-Day Moving Average	68.6871	22.7109	0.2365	-0.4709	515.7858
Panel C: Stock Indices						
6	Dow Jones Industrual Average Index	9.753	0.4865	0.308	-1.197	0.2367
7	NASDAQ Composite Index	8.416	0.7583	0.3666	-1.1841	0.575
8	S&P 500 Index	7.5842	0.5431	0.3986	-1.0268	0.2949
Panel D: Exchange Rates						
9	USD/EUR	1.2233	0.1288	0.5128	-0.4818	0.0166
10	USD/GBP	1.5279	0.2366	0.4263	-0.8392	0.056
11	JPY/USD	110.5382	17.9653	0.4789	0.4028	322.752
Panel E: Interest Rates						
12	Effective Federal Funds Rate	1.6901	1.8758	0.8928	-0.7252	3.5184
13	Bank Prime Loan Rate	4.7926	1.8586	0.9229	-0.6737	3.4545
14	1-Year Treasury Constant Maturity Rate	1.8096	1.7819	0.7498	-0.9169	3.1752
15	10-Year Treasury Constant Maturity Rate	3.042	1.1364	-0.0781	-1.0441	1.2915
16	5-Year Breakeven Inflation Rate	1.9405	0.5704	-1.261	5.3625	0.3253
17	10-Year Breakeven Inflation Rate	2.0954	0.3995	-1.2464	2.6557	0.1596

Figure 8: List of explanatory variables along with their mean, standard deviation, skewness, kurtosis, and variance as of May 2025

The information from EIA and FRED will be updated periodically and we take that through a registered API key on their website. This allows our dataset to be a rolling dataset.

Although we initially attempted to use Vietnamese datasets to achieve better local relevance, data availability proved to be a major limitation. Petrolimex, Vietnam's largest petroleum distributor, does not provide public access to its detailed data. While PVOil does publish some data, it is limited in scope, only covering the period from 2018 onward, with approximately 200 usable data points [19]. Other potentially valuable economic and energy indicators in Vietnam are either inaccessible or not recorded systematically. As a result, we opted to use U.S. datasets due to their richness, consistency, and comprehensiveness, making them more suitable for building robust forecasting models.

3.2 Data Preprocessing

Firstly, we addressed the issue of missing values in the dataset. Although the data is updated periodically, some entries are missing due to holidays. Records containing missing values were handled either through deletion (in the case of sparse rows) or imputation based on the semantics of each respective column.

Moreover, we observed that the values of stock market indicators are relatively large, which can complicate computations. To address this, we applied a logarithmic transformation to all numerical variables using the formula

$\log(1 + x)$. This transformation helps to reduce skewness and heteroscedasticity, while preserving zero values and minimizing the influence of extreme outliers.

Regarding the polynomial regression model, direct support is limited in some libraries. However, we utilized the `scikit-learn` library to generate polynomial features, enabling the construction of polynomial regression models by transforming the original features accordingly.

4 Implementation

At the core, a `model()` class in `models.py` encapsulates every step from data acquisition to prediction. When the class is instantiated, it immediately downloads Cushing crude-oil prices from Yahoo Finance together with a rich set of macro-economic indicators (Dow Jones, S&P 500, NASDAQ, FRED interest-rate and inflation series) and derives technical features such as 5-/10-day momenta and moving averages. Missing values are dropped, and equity indices are log-transformed so that all explanatory variables are on comparable scales. The resulting cleaned `pandas DataFrame` is stored in memory and reused for every user request, which keeps the interface highly responsive.

A modelling toolbox is then built up as a Python dictionary with estimators: linear regression, four kernel variants of SVM, two Gaussian-process kernels, two tree-based ensemble methods (bagging and gradient boosting) and a custom polynomial-regression pipeline. This design makes it trivial to add or remove algorithms while giving the front-end a single uniform handle to trigger training and inference.

When a forecast is requested, the program first calls `model_assessment`, which tries each eligible model, trains it on 70% of the most recent data, and checks its average percentage error on the remaining 30%. The model with the smallest error is kept, and its score is displayed in the interface as an accurate measure. That model is then retrained with the freshest data and the `predict_for_future_days` function projects oil prices up to 50 trading days ahead, returning both the accuracy value and a tidy `DataFrame` of predicted prices ready for plotting.

The entire workflow is wrapped in an intuitive Gradio interface defined in `app.py`. A landing “greeting” column leads into three functional tabs:

- Price Display plots historical prices for any year/month combination supplied by dropdowns.
- Model Prediction lets the user pick an algorithm (and, where relevant, kernel or ensemble method) and visualises predicted versus true prices on the hold-out set.
- Predict Part allows entry of a forecasting horizon, shows the automatically selected model’s accuracy, and renders the forward curve as a scatter plot.

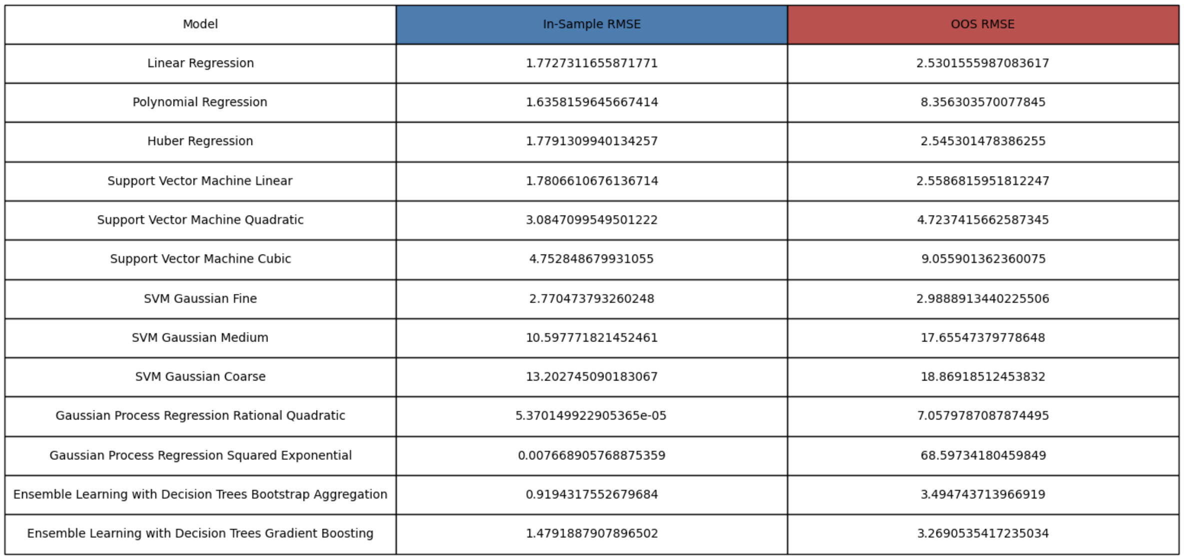
All components share a pastel theme and custom CSS, and the whole block is launched on `0.0.0.0:7860`, making it accessible both locally and inside a container.

Eight well-maintained libraries are required – Gradio for the UI, `scikit-learn` and supporting scientific-Python packages for modelling, and `yfinance`, `fredapi`, `pandas-datareader` for data collection. With the Docker container, the service can be deployed consistently on any server or cloud platform without additional configuration.

5 Results

5.1 Timeframe t + 1

The results for the one-day-ahead forecast (t + 1) are presented in the table. For this time horizon, the Polynomial Regression model achieved the best in-sample performance with an RMSE of 1.6358, while Linear Regression demonstrated the most balanced out-of-sample prediction with an RMSE of 2.5302. The SVM Linear model also performed consistently, yielding in-sample (1.7807) and out-of-sample (2.5587) results. Notably, Gaussian Process Regression with a Squared Exponential kernel showed signs of overfitting, delivering good in-sample accuracy (0.0077 RMSE) but failing out-of-sample (68.5973 RMSE).

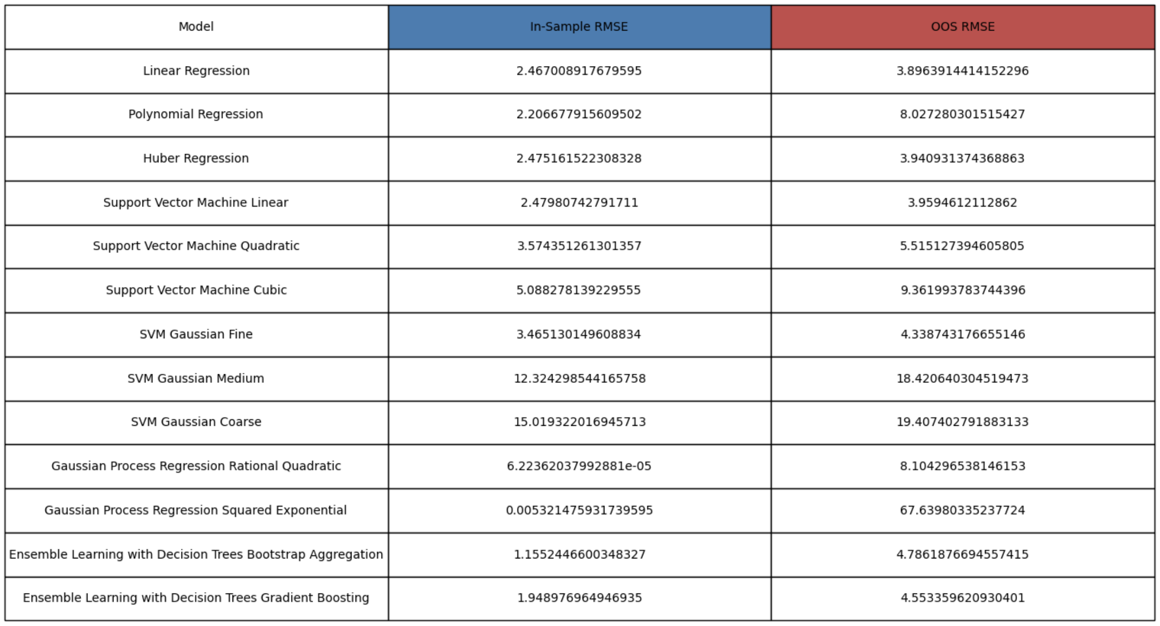


Model	In-Sample RMSE	OOS RMSE
Linear Regression	1.7727311655871771	2.5301555987083617
Polynomial Regression	1.6358159645667414	8.356303570077845
Huber Regression	1.7791309940134257	2.545301478386255
Support Vector Machine Linear	1.7806610676136714	2.5586815951812247
Support Vector Machine Quadratic	3.0847099549501222	4.7237415662587345
Support Vector Machine Cubic	4.752848679931055	9.055901362360075
SVM Gaussian Fine	2.770473793260248	2.9888913440225506
SVM Gaussian Medium	10.597771821452461	17.65547379778648
SVM Gaussian Coarse	13.202745090183067	18.86918512453832
Gaussian Process Regression Rational Quadratic	5.370149922905365e-05	7.0579787087874495
Gaussian Process Regression Squared Exponential	0.007668905768875359	68.59734180459849
Ensemble Learning with Decision Trees Bootstrap Aggregation	0.9194317552679684	3.494743713966919
Ensemble Learning with Decision Trees Gradient Boosting	1.4791887907896502	3.2690535417235034

Figure 9: RMSE for t + 1 forecasting.

5.2 Timeframe t + 3

Polynomial Regression achieved the best in-sample RMSE (2.21), while Linear Regression showed the most stable out-of-sample performance (3.90). Linear SVM and Huber Regression delivered consistent results (∼2.5–4.0 RMSE). Notably, Gaussian Process (Squared Exponential) again demonstrated severe overfitting with near-zero in-sample RMSE but failed out-of-sample (67.64). More complex models (SVM Gaussian Medium/Coarse, Ensemble methods) underperformed compared to simpler linear models.



Model	In-Sample RMSE	OOS RMSE
Linear Regression	2.467008917679595	3.8963914414152296
Polynomial Regression	2.206677915609502	8.027280301515427
Huber Regression	2.475161522308328	3.940931374368863
Support Vector Machine Linear	2.47980742791711	3.9594612112862
Support Vector Machine Quadratic	3.574351261301357	5.515127394605805
Support Vector Machine Cubic	5.088278139229555	9.361993783744396
SVM Gaussian Fine	3.465130149608834	4.338743176655146
SVM Gaussian Medium	12.324298544165758	18.420640304519473
SVM Gaussian Coarse	15.019322016945713	19.407402791883133
Gaussian Process Regression Rational Quadratic	6.22362037992881e-05	8.104296538146153
Gaussian Process Regression Squared Exponential	0.005321475931739595	67.63980335237724
Ensemble Learning with Decision Trees Bootstrap Aggregation	1.1552446600348327	4.7861876694557415
Ensemble Learning with Decision Trees Gradient Boosting	1.948976964946935	4.553359620930401

Figure 10: RMSE for t + 3 forecasting.

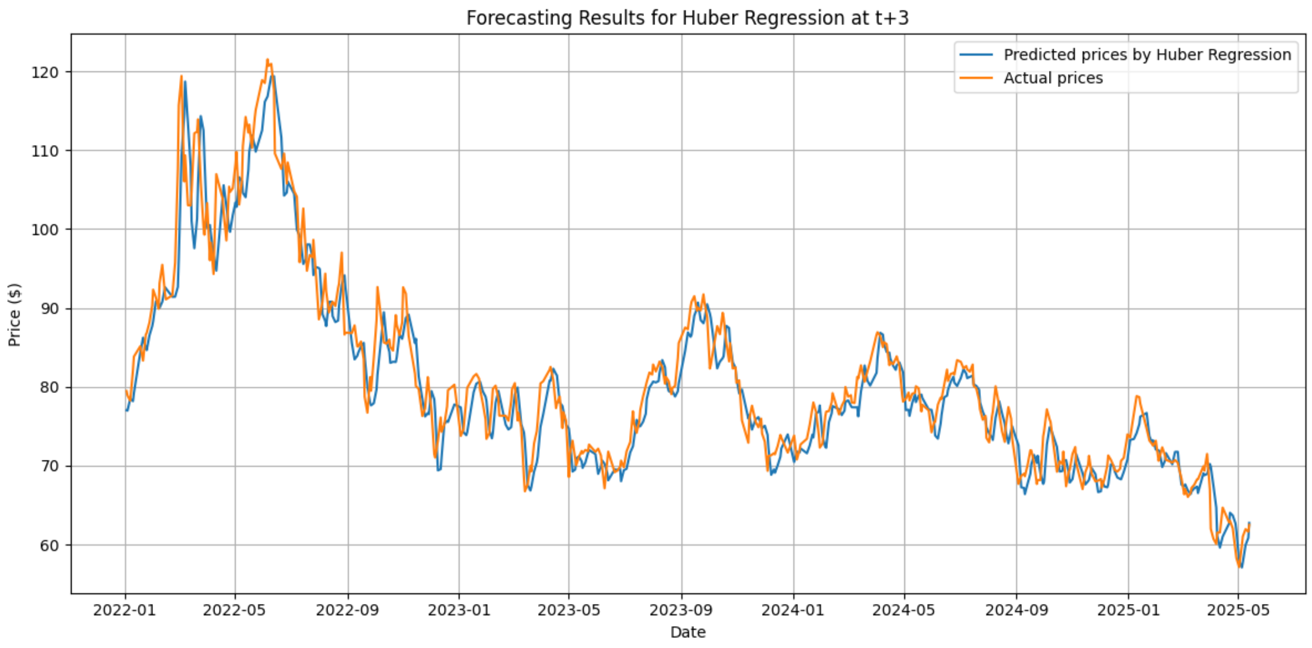
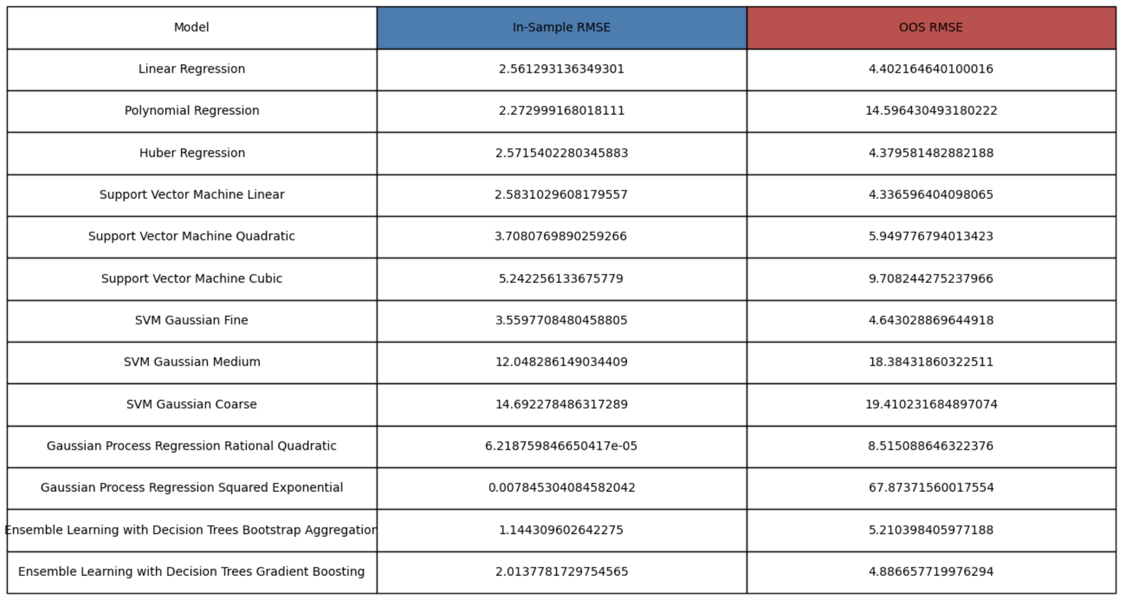


Figure 11: Huber Regression performance

5.3 Timeframe t + 5

Linear SVM delivers best out-of-sample performance (4.34 IMSE), slightly outperforming Linear Regression (4.40) and Huber (4.38). Polynomial Regression shows severe overfitting (2.27 in-sample vs. 14.60 OOS). Gaussian Process fails again (0.0078 vs. 67.87 IMSE). Complex models (SVM Gaussian/Ensembles) underperform simpler linear approaches.

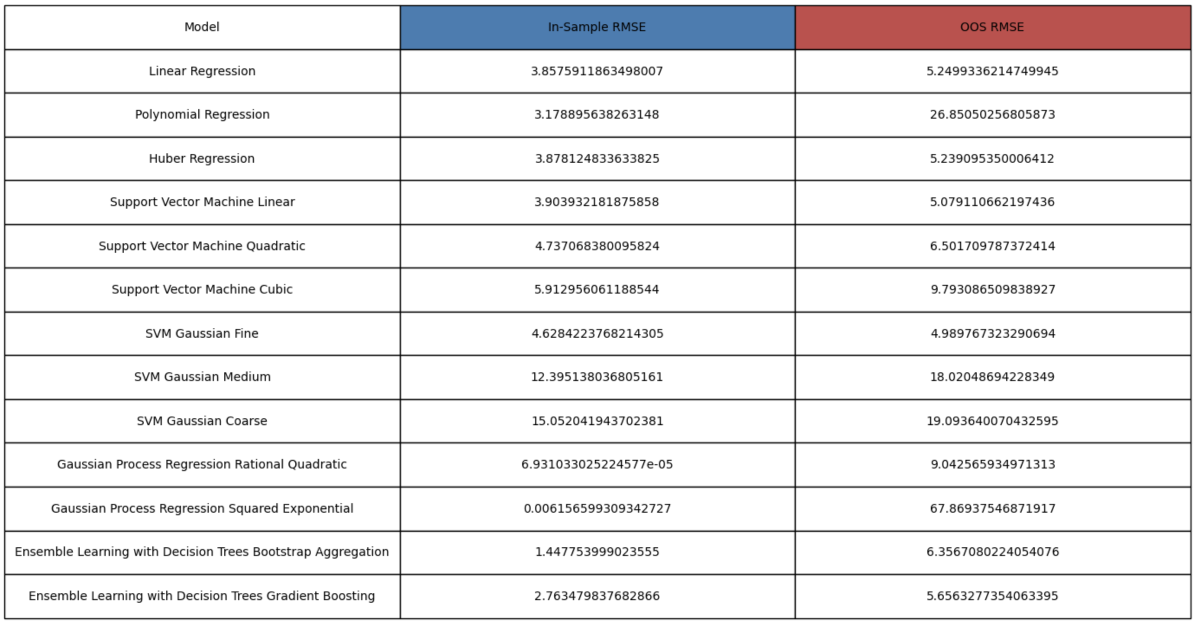


Model	In-Sample RMSE	OOS RMSE
Linear Regression	2.561293136349301	4.402164640100016
Polynomial Regression	2.272999168018111	14.596430493180222
Huber Regression	2.5715402280345883	4.379581482882188
Support Vector Machine Linear	2.5831029608179557	4.336596404098065
Support Vector Machine Quadratic	3.7080769890259266	5.949776794013423
Support Vector Machine Cubic	5.242256133675779	9.708244275237966
SVM Gaussian Fine	3.5597708480458805	4.643028869644918
SVM Gaussian Medium	12.048286149034409	18.38431860322511
SVM Gaussian Coarse	14.692278486317289	19.410231684897074
Gaussian Process Regression Rational Quadratic	6.218759846650417e-05	8.515088646322376
Gaussian Process Regression Squared Exponential	0.007845304084582042	67.87371560017554
Ensemble Learning with Decision Trees Bootstrap Aggregatior	1.144309602642275	5.210398405977188
Ensemble Learning with Decision Trees Gradient Boosting	2.0137781729754565	4.886657719976294

Figure 12: RMSE for t + 5 forecasting.

5.4 Timeframe t + 10

SVM Gaussian Fine achieves best OOS performance (4.99 RMSE), slightly better than Linear SVM (5.08). Polynomial Regression fails catastrophically (3.18 vs. 26.85 RMSE). Gaussian Process (Squared Exponential) shows worst overfitting (0.006 vs. 67.87). Simple models (Linear/Huber) outperform complex ones consistently.



Model	In-Sample RMSE	OOS RMSE
Linear Regression	3.8575911863498007	5.2499336214749945
Polynomial Regression	3.178895638263148	26.85050256805873
Huber Regression	3.878124833633825	5.239095350006412
Support Vector Machine Linear	3.903932181875858	5.079110662197436
Support Vector Machine Quadratic	4.737068380095824	6.501709787372414
Support Vector Machine Cubic	5.912956061188544	9.793086509838927
SVM Gaussian Fine	4.6284223768214305	4.989767323290694
SVM Gaussian Medium	12.395138036805161	18.02048694228349
SVM Gaussian Coarse	15.052041943702381	19.093640070432595
Gaussian Process Regression Rational Quadratic	6.931033025224577e-05	9.042565934971313
Gaussian Process Regression Squared Exponential	0.006156599309342727	67.86937546871917
Ensemble Learning with Decision Trees Bootstrap Aggregation	1.447753999023555	6.3567080224054076
Ensemble Learning with Decision Trees Gradient Boosting	2.763479837682866	5.6563277354063395

Figure 13: RMSE for t + 10 forecasting.

5.5 Limitations

The current approach relies on relatively simple machine learning algorithms, which may limit predictive performance and the ability to capture complex patterns in the data.

5.6 Future work

Future iterations could explore more advanced deep learning models, such as Long Short-Term Memory (LSTM) networks, to better handle temporal dependencies and improve accuracy. This could be a direction for further development in the next semester.

6 Contributions

- Hoang Trung Dung: Gaussian process regression theory and implementation, report editing, and presentation.
- Le Minh Kiet: Decision trees, bagging, and boosting theory and implementation, Gradio interface.
- Le Tam Quang: Linear regression and Huber regression theory and implementation, Gradio interface.
- Nguyen Vu Thuy: Support vector machines theory and implementation, Docker setup.
- Nguyen Tat Hung: Polynomial regression theory and implementation, report editing, Docker setup.

References

[1] Ervin Ceperic. Short-term forecasting of gas prices using machine learning and feature selection algorithms. *Energy*, 140:893–900, 2017.

[2] A.C. Davison and D.V. Hinkley. *Bootstrap Methods and their Application*. Cambridge University Press, 1997.

[3] S. Diab and M. B. Karaki. Do increases in gasoline prices cause higher food prices? *Energy Economics*, 127(Part B):107066, 2023.

[4] Norman R. Draper and Harry Smith. *Applied Regression Analysis*, volume 326. John Wiley & Sons, 1998.

[5] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993.

[6] D.-S. Tran et al. Two-wheelers in vietnam: A baseline analysis of fleet characteristics and fuel consumption in 2019 and 2020, 2022. Working Paper 2022-08, February.

[7] S. Schlüter et al. Interval forecasts for gas prices in the face of structural breaks, 2024. arXiv:2407.16723, Jul 2024.

[8] Z. Zhao et al. Improvement to the prediction of fuel cost distributions using arima model, 2018. Accepted at IEEE PES General Meeting.

[9] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

[10] Hubert Gatignon. *Statistical Analysis of Management Data*. Kluwer Academic Publishers, Boston, MA, 2003.

[11] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and John Franklin. The elements of statistical learning: Data mining, inference and prediction. *The Mathematical Intelligencer*, 27:83–85, 2005.

[12] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 112. Springer, New York, 2013.

[13] Michael I. Jordan. Soft margin svm. https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec6.pdf, 2004. Accessed: 2025-05-16.

[14] machinelearningcoban.com. Bài 19: Support vector machine. https://machinelearningcoban.com/2017/04/09/smv/, 2017. Accessed: 2025-05-16.

[15] machinelearningcoban.com. Bài 20: Soft margin support vector machine. https://machinelearningcoban.com/2017/04/13/softmarginsmv/, 2017. Accessed: 2025-05-16.

[16] Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 7:2651–2667, 2006.

[17] Vinod Mishra and Russell Smyth. Are natural gas spot and futures prices predictable? *Economic Modelling*, 54:178–186, 2016.

[18] Eva Ostertagová. Modelling using polynomial regression. *Procedia Engineering*, 48:500–506, 2012.

[19] PetroVietnam Oil Corporation (PVOIL). Gia xang dau, 2025. Accessed May 2025.

[20] J. Ross Quinlan. C4.5: Programs for machine learning. Morgan Kaufmann Publishers, Inc., 1993.

[21] David Rosenberg. Svm - complementary slackness. https://davidrosenberg.github.io/mlcourse/Archive/2017/Lectures/5a.SVM-ComplementarySlackness.pdf, 2017. Accessed: 2025-05-16.

[22] Robert E. Schapire. The boosting approach to machine learning: An overview. In *Nonlinear Estimation and Classification*, pages 149–171. Springer, 2003.

[23] scikit-learn developers. Matern kernel — scikit-learn 1.4.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.Matern.html. Accessed: 2025-05-02.

[24] Varda Senderovich Madar and Sergio L. Batista. Solving the ordinary least squares in closed form, without inversion or normalization, 2023. arXiv e-prints, arXiv:2301.

[25] Priyanka Sinha. Multivariate polynomial regression in data mining: Methodology, problems and solutions. *International Journal of Scientific and Engineering Research*, 4:962–965, 2013.

[26] MIT 6.034 Course Staff. Svm recitation notes. https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf, 2008. Accessed: 2025-05-16.

[27] Moting Su, Zongyi Zhang, Ye Zhu, Donglan Zha, and Wenying Wen. Data driven natural gas spot price prediction models using machine learning methods. *Energies*, 12(9):1680, 2019.

[28] Cornell University. Lecture 9: Svm. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html, 2018. Accessed: 2025-05-16.

[29] Jie Wang. An intuitive tutorial to gaussian process regression. arXiv:2009.10862 [stat.ML], https://arxiv.org/abs/2009.10862, 2020.

[30] Ruye Wang. Norm soft margin. https://pages.hmc.edu/ruye/e161/lectures/svm/node7.html, 2016. Accessed: 2025-05-16.