MODEL FOR PREDICTING THE EFFECT OF NEWS ON STOCK VALUE

**Author: Nguyen Hung Thanh**

Ho Chi Minh City, December 25, 2023

**MODEL FOR PREDICTING THE EFFECT OF NEWS ON STOCK VALUE**

**Abstract**

In this project, my goal is to build a model to predict whether news about a company will affect the value of that company's stock. With the dataset collected from scraping news from Google to make a feature. labels will be marked based on the fluctuation of the stock value, where increase and decrease will be marked as changed. News scraped from Google needs to go through processing steps with functions in Python's natural language processing library to obtain appropriate summary information for the predictive model.

For predicting whether the stock value is changed or unchanged, the problem can be viewed as a classification problem of supervised learning with 2 labels: changed or unchanged. But in the future, the problem can be expanded to predict the relative increase or decrease in stock value. At this point it will become a regression problem of supervised learning.

Within the scope of this project, I will use Logistic Regression model, SVM model, Random Forest and k-Nearest Neighbors model for testing. Because natural language is a relatively complex domain, if you want to predict more accurately or extend the problem further, you may need to use models with higher complexity.

Index

[1. Introduction 3](#_Toc162136952)

[1.1. Purpose of document implementation 3](#_Toc162136953)

[1.2. The goal of the subject 3](#_Toc162136954)

[1.3. Methodology used 3](#_Toc162136955)

[2. Data Collection and Preprocessing 4](#_Toc162136956)

[2.1. Data Collection 4](#_Toc162136957)

[2.2. Preprocessing Data 14](#_Toc162136958)

[3. Methodology 16](#_Toc162136959)

[3.1. Logistic Regression 16](#_Toc162136960)

[3.2. Support Vector Machines (SVM) 17](#_Toc162136961)

[3.3. Random Forest Classification 19](#_Toc162136962)

[3.4. K-Nearest Neighbors 20](#_Toc162136963)

[3.5. Feature engineering 21](#_Toc162136964)

[4. Model Development 28](#_Toc162136965)

[5. Results 29](#_Toc162136966)

[6. Conclusion 30](#_Toc162136967)

# Introduction

## Purpose of document implementation

My aim is to predict stock market fluctuations based on the news during that time period. Because stocks are affected by many objective factors and news is only a part of them, it is impossible to predict too much based on news alone. So within the scope of this project, I only predict whether news can change stock values or not.

Although it is currently only possible to expect how news in a specific period of time will affect stock value, it will also be a valuable reference source for stock buyers to preliminary assess market volatility prices based on recent news sources.

Later, with models with more complex algorithms and more complete information, the problem can be expanded towards more specific predictions.

## The goal of the subject

Build a model to predict whether news about a company will affect the value of that company's stock. With dataset being a summary of news scraped from google to make a feature. Labels will be marked based on the fluctuation of the stock value, where increase and decrease will be marked as changed.

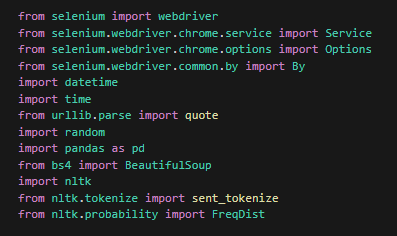
## Methodology used

Within the scope of this project, I will use Logistic Regression, SVM, Random Forest, and k-Nearest Neighbors models for testing. Since natural language is a relatively complex domain, for more accurate predictions or to extend the problem further, it may be necessary to use models with higher complexity.

# Data Collection and Preprocessing

## Data Collection

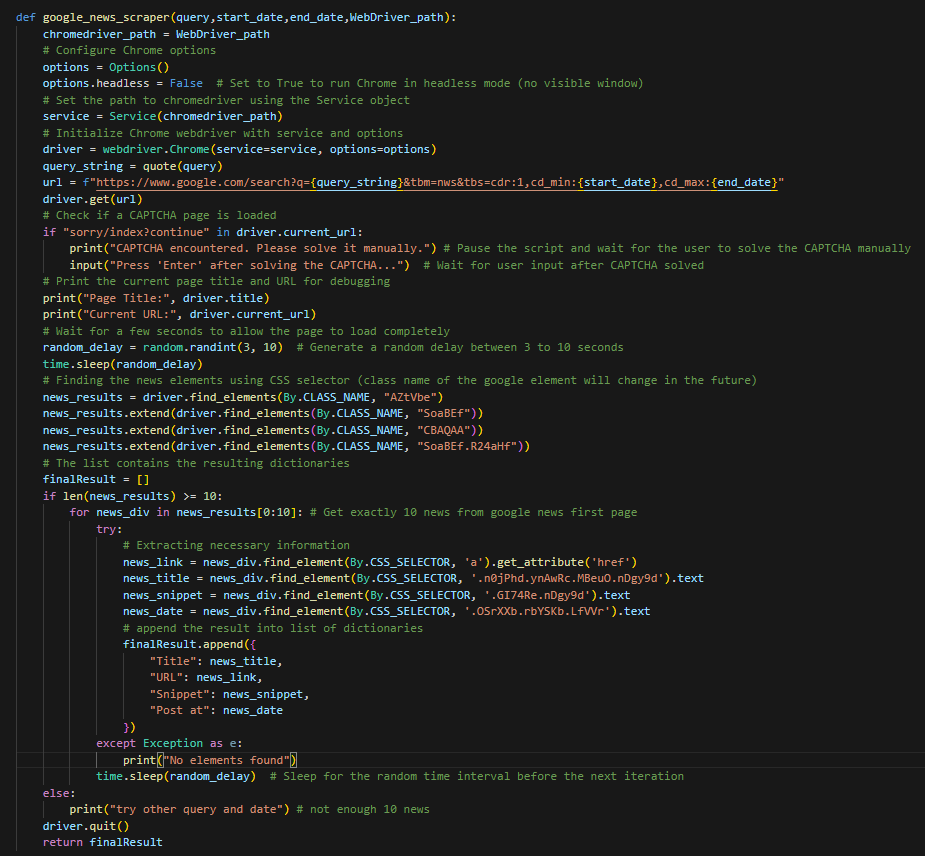
To collect data to create a dataset, I use the selenium library and its packages to build a function that can scrape data from Google's news page over a specific period of time. Thereby finding information about outstanding articles related to the search goal such as title, article link, snippet, publication time,... as well as summarize the content of each articles by combining with other libraries



2‑1: Library and package used for making dataset

The Selenium library is a popular tool used for automating web browsers. It enables interaction with web elements, simulating user actions such as clicks and form submissions, scraping data that might be generated via JavaScript, and handling complex scenarios involving dynamic content, such as JavaScript-driven web applications. Its ability to simulate user activity can help bypass Google's censorship programs and prevent automatic data scraping.

The urllib.parse module in Python provides several functions for parsing URLs and manipulating query parameters. The quote() function from this module is used for URL encoding or percent-encoding strings, which is particularly useful when constructing URLs with special characters.



2‑2: Function used to scrape news articles from google news

This function refines the scraping process by addressing possible CAPTCHA interruptions and attempts to fetch news articles from the Google News search page.

The function performs the following steps:

* Configures Chrome options and initializes the Chrome WebDriver.
* Constructs the Google News URL with the provided query and date range.
* Navigates to the constructed URL using the WebDriver.
* Checks for CAPTCHA pages, prompting user intervention if a CAPTCHA is detected.
* Scrapes information (title, URL, snippet, date) from news elements on the Google News page.
* Gathers the resulting information into a list of dictionaries (finalResult) and returns it.

The URL variable contains a formatted string constructed for the Google News search with the query, start date, and end date parameters. When you fill in the placeholders {query\_string}, {start\_date}, and {end\_date} with the appropriate values, the formed URL will execute a Google News search with the specified query and date range.

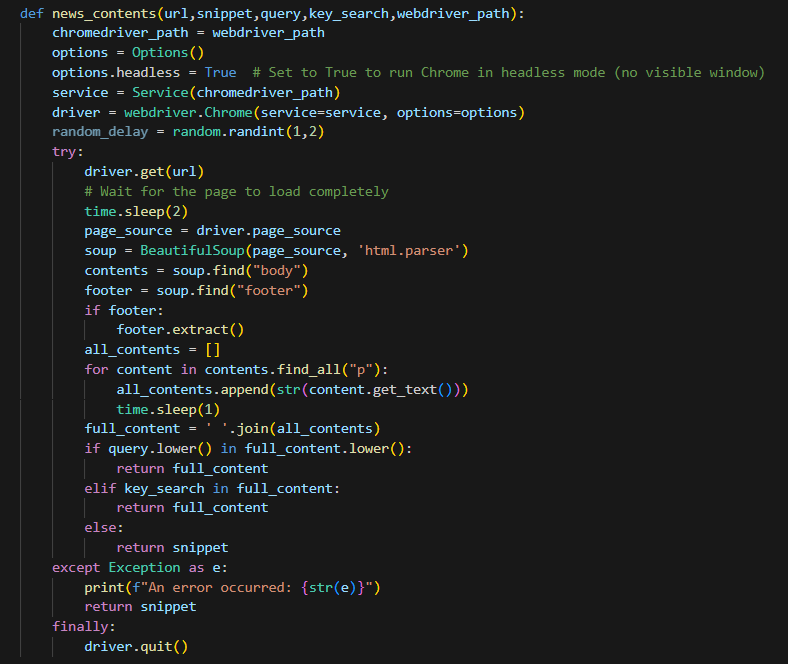
The dates parameters format of the Google search URL is mm/dd/yyyy, while Vietnamese typically use dates with the dd/mm/yyyy format. For convenience, I have created a simple date\_format function to convert the input date from dd/mm/yyyy to mm/dd/yyyy.

The main use of the time.sleep() function is to pause the execution of the script for a random duration generated by the random.randint() function. This random delay can be useful in web scraping scenarios to simulate more natural behavior, avoiding constant and predictable intervals between actions. It helps evade detection or restrictions by websites that monitor for automated or bot-like behavior.

Google's HTML structure may change over time, leading to potential obsolescence of the CSS selectors used to find elements. Regular maintenance may be necessary to update the selectors accordingly.

Note that Google's HTML structure may change over time, so the CSS selectors used to find elements (such as "AZtVbe", "SoaBEf", "CBAQAA", "SoaBEf.R24aHf" for the articles class name or 'n0jPhd ynAwRc MBeuO nDgy9d' for the title class name, 'GI74Re nDgy9d' for the snippet class name, 'OSrXXb rbYSKb LfVVr' for the publication date class name) may become outdated. Regular maintenance may be required to update selectors accordingly.

After obtaining the link of each article, I utilize the news\_content function to access and retrieve the entire content of each article.



2‑3: Function used to scrape entire content of each article

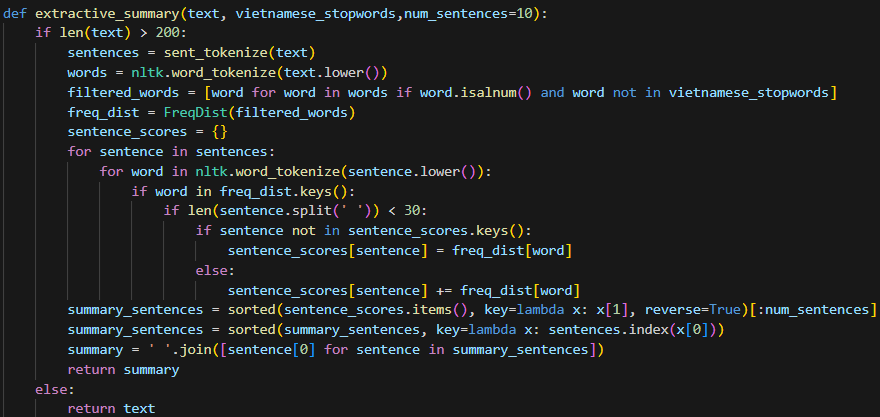
Similar to the function above, the Selenium library packages are employed to simulate user activity and scrape the entire content of the article. Additionally, the function utilizes BeautifulSoup packages to extract specific parts of the article from the HTML structure of the website.

This function aims to fetch and extract the main content of a news article from the provided URL using web scraping techniques. It then checks if specific keywords (query or key\_search) are present in the extracted content and returns either the full content or a snippet based on the search results.

The time.sleep() functions here are primarily used to pause the execution of the script, allowing the WebDriver to load website information and preventing the website from being overloaded when requesting too many actions in a short time.

All code is encapsulated within a try block to handle any exceptions that might occur during the process, such as the website no longer existing, DNS addresses being banned, IP addresses being banned, etc. In case of an exception, the function returns the provided snippet as the article content along with an error message.

Afterward, I utilize the full content of each article to summarize it using the extractive\_summary function.



2‑4: Function used to summarize content of each article

The extractive\_summary function generates an extractive summary of a text based on the frequency of words occurring in sentences. It utilizes NLTK (Natural Language Toolkit) for text processing.

If the length of the input text is greater than 200 characters, the function proceeds with summarization. Otherwise, if it is an article’s snippet, it returns the input text itself.

It utilizes the nltk package's functions to tokenize the input text into sentences and words, converting them to lowercase. Then, it filters out non-alphanumeric words and removes Vietnamese stopwords from the list of words. Vietnamese stopwords are often filtered out because they may not carry significant meaning in the context of analysis or summarization.

After obtaining a list of meaningful words, it utilizes the FreqDist function from the nltk package to generate a frequency distribution of the filtered words. Following this, it calculates sentence scores based on the sum of word frequencies within each sentence and assigns scores to sentences based on the sum of frequencies of their constituent words. Priority is given to longer sentences, typically those with fewer than 30 words.

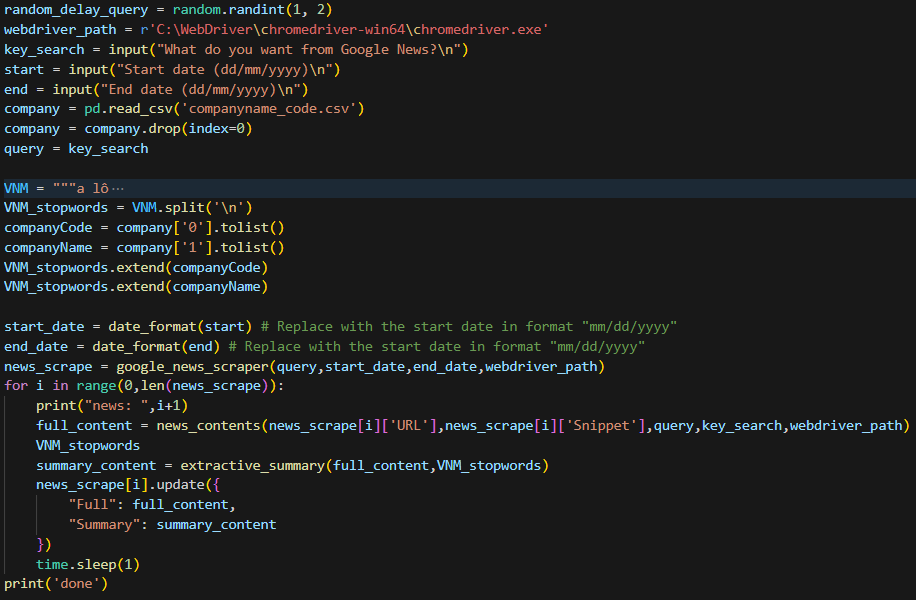
The function retrieves the top num\_sentences sentences with the highest scores as the summary sentences. It sorts the summary sentences based on their original order in the text and joins the selected summary sentences into a single string (summary) to form the final extractive summary. Then, it returns the generated summary if the text length is sufficient for summarization; Otherwise, it returns the original text.

The extractive\_summary function follows an extractive approach where sentences containing the most significant words based on frequency are chosen for the summary. However, it's important to note that extractive summarization methods may not fully capture the context and coherence of the original text.

Note that this implementation might not provide ideal summaries for longer or complex texts, and its effectiveness might vary based on the content being summarized. Additionally, further testing and refinements may be needed for robustness and accuracy in summary generation.

**But in terms of training a prediction model based on the frequency distribution of each word, I believe it will be sufficient.**

This is the execution code that utilizes all the aforementioned functions to generate each feature of the dataset from Google News.



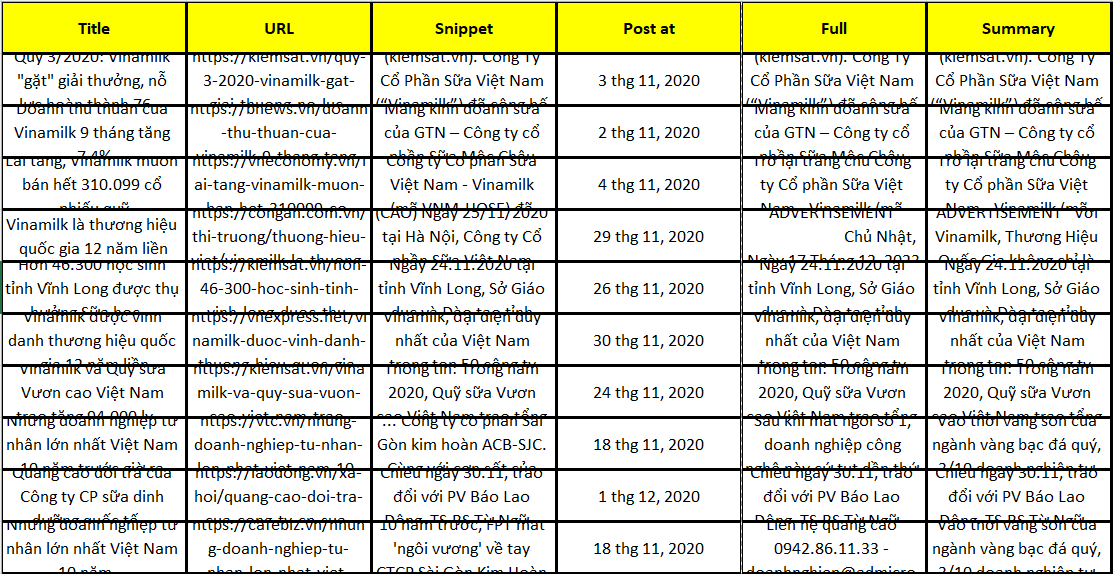
2‑5: Execution code

VNM\_stopwords is derived from a substring of all Vietnamese stopwords I found combined with the names and codes of listed stock companies from the 'companyname\_code.csv' file.

The webdriver\_path variable contains the file path to the Chrome WebDriver executable (chromedriver.exe) on my laptop. This file path is specified using a raw string literal (r'...') in Python, which helps avoid interpreting backslashes as escape characters.

These lines of code scrape news articles from Google News, store them in the news\_scrape variable, and then retrieve the link of each article from news\_scrape to scrape the entire content of each article. Subsequently, they summarize the content and update the news\_scrape variable with the full content and summary collected.

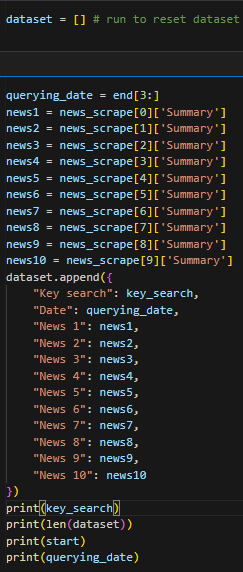
The output will be shown below:



2‑6: Output of News Scraper in file

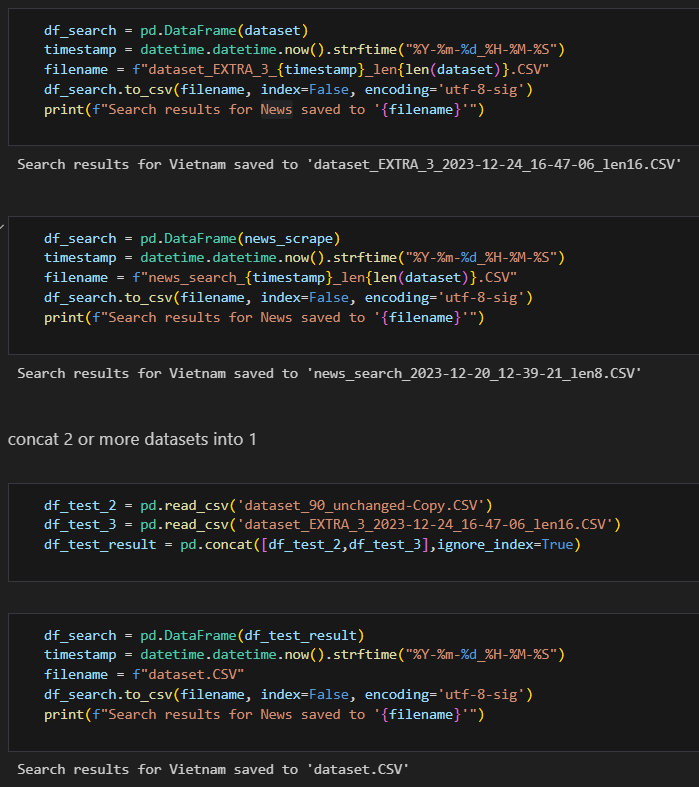
These are all 10 news articles in 1 month that will be used to create a feature of the dataset.

Each of these summary columns will be converted into dataset features using the following lines of code.



2‑7: Code to convert summary into dataset’s feature

Each time you want to create the dataset, simply run "dataset = []" once to reset the dataset list. Then, run all the other lines every time you scrape a news article to append it to your dataset list.



2‑8: Code to save dictionaries as file

The third block will concat my dataset into 1 when I have create 2 or more different dataset



2‑9: Output of the dataset creator

## Preprocessing Data

After finishing creating all my dataset, I added the stock prices that I had investigated before by searching for the company code from the "<https://s.cafef.vn/lich-su-giao-dich-vnindex-1.chn>“ website. I then calculated "Price fluctuations" by subtracting the price of the company from the month I took with the previous price of it.

Then, I created a "Fluctuations" column with the condition: if the "Price fluctuations" is greater than 1, it is labeled as "increases"; if the "Price fluctuations" is lower than -1, it is labeled as "decreases"; otherwise, it remains unchanged.

As there is no null data, I have completed my dataset as follows.



2‑10: Dataset

# Methodology

As mentioned above, within the scope of this project, I will use Logistic Regression, SVM, Random Forest, and k-Nearest Neighbors models for testing because it is a classification problem.

## Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for binary classification tasks. Despite its name containing "regression," it's actually a classification algorithm commonly used when the target variable is categorical.

Logistic Regression is specifically used for binary classification problems where the target variable has two possible outcomes (e.g., yes/no, true/false, 0/1).

The algorithm uses the sigmoid function to map predictions between 0 and 1, representing probabilities. This function ensures that the output is in the range [0, 1], making it suitable for classification.

Logistic Regression finds a decision boundary in the input feature space to separate data points into different classes based on their features.

How It Works:

* Hypothesis Function: The logistic regression model calculates the weighted sum of input features along with a bias term.
* Sigmoid Activation: It then applies the sigmoid (logistic) function to the output of the hypothesis function. The sigmoid function maps any real-valued number to the range [0, 1].
* Loss Function: Logistic Regression uses a loss function called log loss (or cross-entropy loss) to measure the error between predicted and actual values.
* Optimization: The objective is to minimize this loss function by adjusting the model parameters (weights and bias) using optimization techniques like Gradient Descent.

Logistic Regression is a fundamental algorithm in machine learning and serves as a starting point for many classification problems. However, it's limited to linear decision boundaries and might not perform well on complex datasets. In such cases, more sophisticated algorithms like Support Vector Machines (SVMs) or ensemble methods like Random Forests or Gradient Boosting might be more suitable.

## Support Vector Machines (SVM)

Support Vector Machines (SVMs) are powerful supervised machine learning models used for classification and regression tasks. They are effective for both linear and non-linear data separation in classification and can also be used for regression.

SVMs are primarily used for classification tasks but can perform regression as well.

The main idea behind SVMs in classification is to find the optimal hyperplane that maximizes the margin between different classes. The hyperplane is the decision boundary that best separates different classes in the feature space.

The main idea behind SVMs in classification is to find the optimal hyperplane that maximizes the margin between different classes. The hyperplane is the decision boundary that best separates different classes in the feature space.

Support vectors are the data points closest to the hyperplane and play a crucial role in defining the decision boundary. These points influence the position and orientation of the hyperplane.

SVMs can efficiently handle non-linear boundaries by using a technique called the kernel trick. Kernels transform the input data into a higher-dimensional space, making it possible to find a linear separation in that space.

SVMs aim to maximize the margin between classes, which leads to better generalization and potentially higher accuracy. Different kernel functions (e.g., linear, polynomial, radial basis function (RBF)) can be used to handle linear and non-linear data distributions. SVMs are less prone to overfitting, especially in high-dimensional spaces, due to the margin concept.

How It Works:

* Data Mapping: For non-linear data, SVMs map the input data into a higher-dimensional space using a kernel function.
* Optimal Hyperplane: In this higher-dimensional space, SVM finds the hyperplane that maximizes the margin between classes while minimizing classification errors.
* Kernel Trick: The use of a kernel function allows SVMs to implicitly compute the dot product between input samples in the higher-dimensional space without explicitly transforming the data.
* Support Vectors: The final decision boundary is determined by the support vectors, which are the data points closest to the hyperplane.

SVMs are versatile and can handle both linear and non-linear problems, making them popular in various domains. However, they might not perform well on larger datasets due to their computational complexity and memory requirements. Additionally, selecting the right kernel and tuning hyperparameters are essential for achieving optimal performance.

## Random Forest Classification

Random Forest Classification is used for classification tasks, where the goal is to predict the class or category of an input based on its features.

Random Forest is an ensemble of decision trees, where each tree is trained on a random subset of the training data and makes independent predictions.

The final prediction is determined by a majority vote (for classification) or averaging (for regression) across all the individual trees.

Key aspects of the Random Forest Classifier:

* Ensemble of Decision Trees: Random Forest builds a collection of decision trees during the training phase. Each decision tree is trained on a random subset of the training data, and each tree has the potential to make different predictions.
* Random Feature Selection: At each split in a decision tree, a random subset of features is considered. This introduces diversity among the trees and helps prevent overfitting.
* Bootstrap Aggregating (Bagging): Random Forest uses a technique called bootstrapping, where multiple random samples (with replacement) are drawn from the training dataset. Each decision tree is trained on one of these bootstrap samples.
* Voting for Classification: In classification tasks, each tree in the forest predicts a class for a given input. The final prediction is determined by a majority vote among all the trees.
* Hyperparameters: Random Forest has hyperparameters that can be tuned for optimal performance, such as the number of trees (n\_estimators), the maximum depth of each tree (max\_depth), and the number of features considered at each split (max\_features).
* Out-of-Bag (OOB) Score: Random Forest can estimate its own generalization performance using out-of-bag samples. These are data points not included in the bootstrap sample used to train a particular tree, and they can be used for validation.
* Feature Importance: Random Forest can provide information about the importance of each feature in making predictions. This can be useful for feature selection and understanding the impact of different features on the model's performance.

## K-Nearest Neighbors

The k-Nearest Neighbors is a type of instance-based or lazy learning algorithm, where the model is not explicitly trained during the training phase.

It memorizes the training instances and makes predictions for new data points based on their proximity to known examples in the feature space.

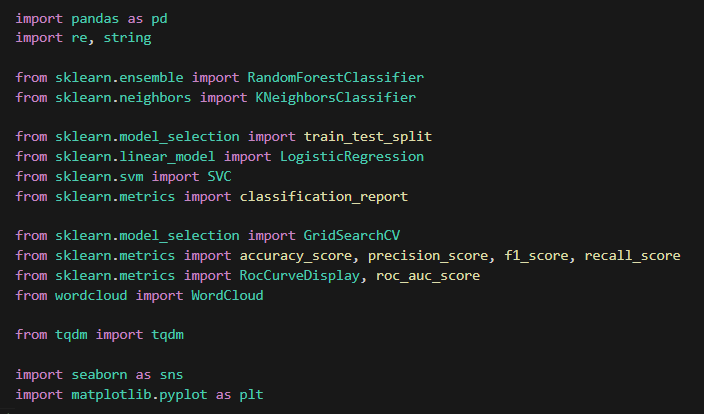
How it works:

* Training Phase: During the training phase, the algorithm simply stores the training dataset.
* Prediction Phase: For a new data point, the algorithm identifies the k-nearest neighbors in the feature space. The prediction is then made based on the majority class (for classification) or the average (for regression) of the labels of the k-nearest neighbors.
* Distance Metric: The choice of distance metric (e.g., Euclidean distance, Manhattan distance) is crucial and depends on the nature of the data.
* Hyperparameter k: The hyperparameter "k" represents the number of neighbors to consider when making a prediction. The value of k is typically chosen based on cross-validation or other model evaluation techniques.
* Decision Boundary: k-NN does not explicitly learn a decision boundary. Instead, it classifies or predicts based on the local neighborhood of a data point.
* Scalability: k-NN can be computationally expensive, especially as the size of the dataset increases. Techniques like KD-trees or Ball trees are used to optimize the search for nearest neighbors.

You can adjust the value of n\_neighbors based on your specific requirements and the characteristics of your dataset. Additionally, it's important to scale the features when using k-NN, as the algorithm is sensitive to the scale of the input variables.

## Feature engineering

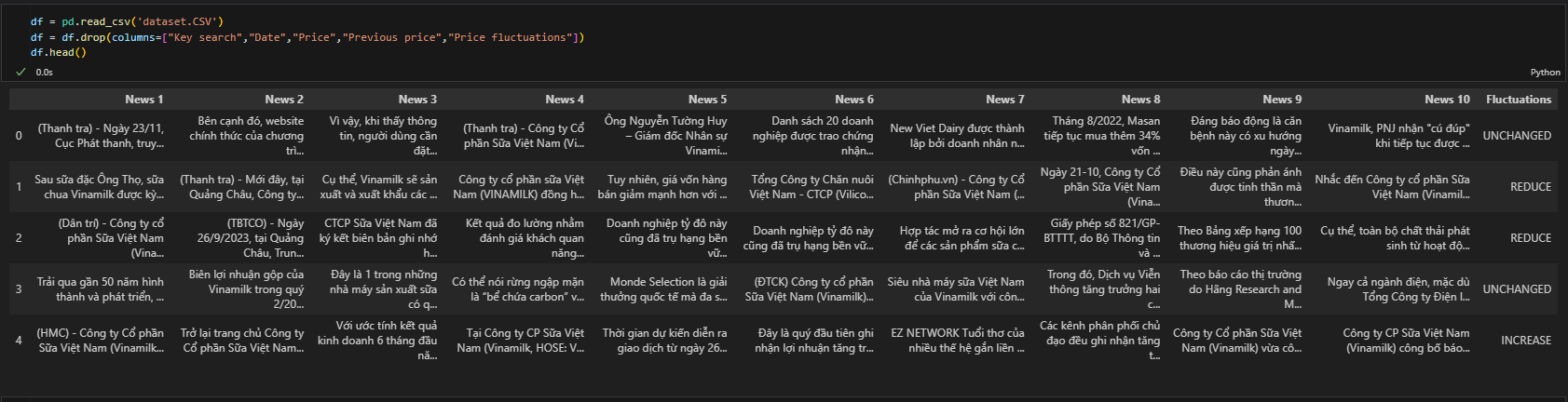
First of all, import all the libraries and packages that can be used in the model



3‑1: Import Library

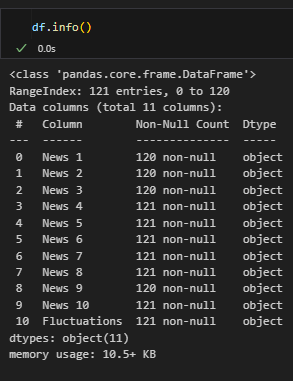
Use the pandas library to read the dataset as a dataframe I've created above and drop all unnecessary columns from that dataframe.

The dataframe will become as below:



3‑2: Base dataframe

Check df.info() to ensure there is no null data in the dataframe

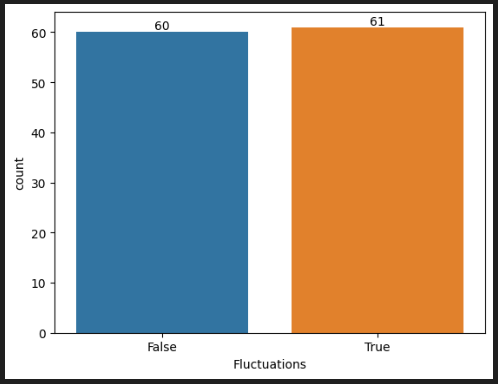


3‑3: Check dataframe information

Set up VNM\_stopwords by using a substring of all Vietnamese stopwords I found combined with all names and codes of listed stock companies from the 'companyname\_code.csv' file.

Rename the 'Fluctuations' column to label the output with two types of data: 'CHANGED' and 'UNCHANGED'. Here, I will rename the 'CHANGED' labels as 'True' and 'UNCHANGED' as 'False' to facilitate calculation.

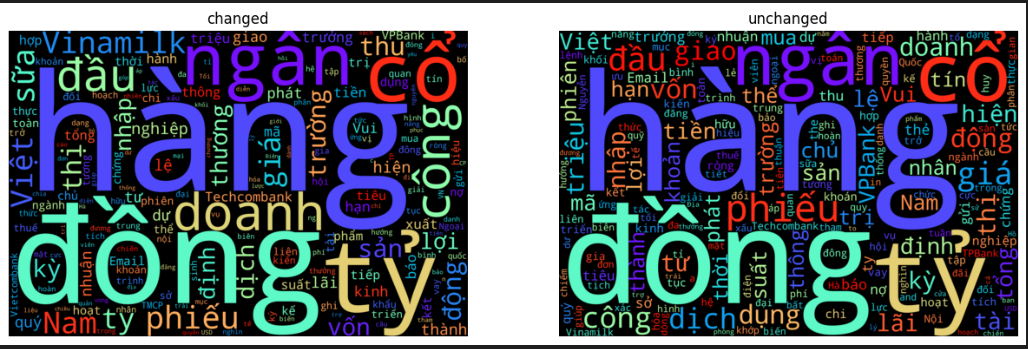
Visualize the "Fluctuations" column to see if the label output is balanced.



3‑4: Visualize “Fluctuations” column

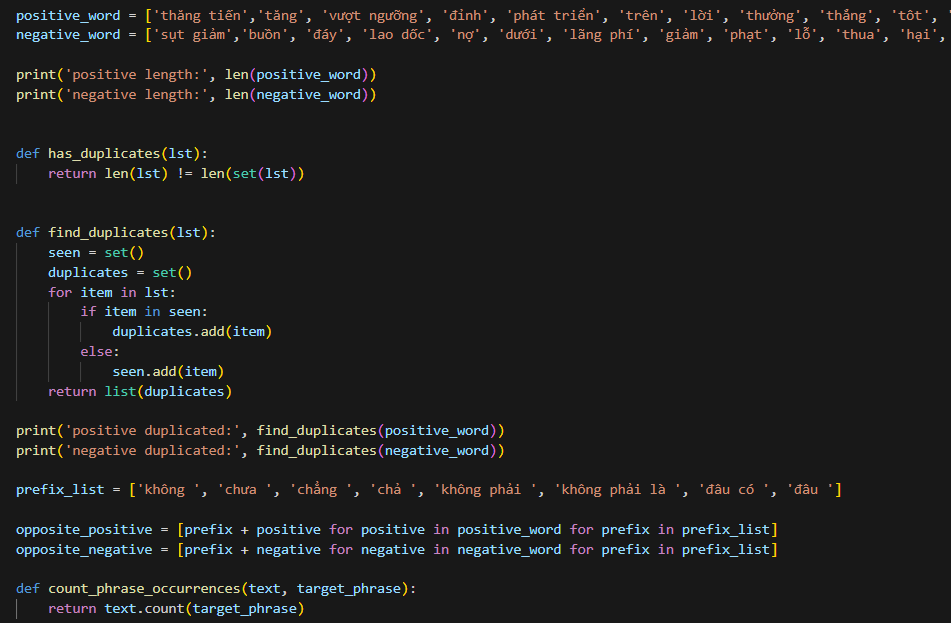
With the 'True' label representing the 'CHANGED' price, meaning 'INCREASE' or 'DECREASE', and the 'FALSE' label representing the 'UNCHANGED' price, it seems quite balanced.

This is the wordcloud feature of "CHANGED" and "UNCHANGED" labels.



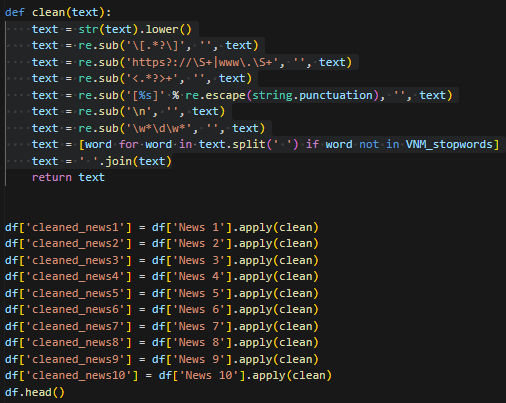
3‑5: Wordclouds

I created two lists of positive and negative words for the purpose of classifying features. Additionally, I also created two more lists of antonyms of the original two lists by adding negative prefixes before them, such as 'không', 'chưa', 'chẳng', 'chả', and so on. Along with that, I developed a few functions to support feature engineering.



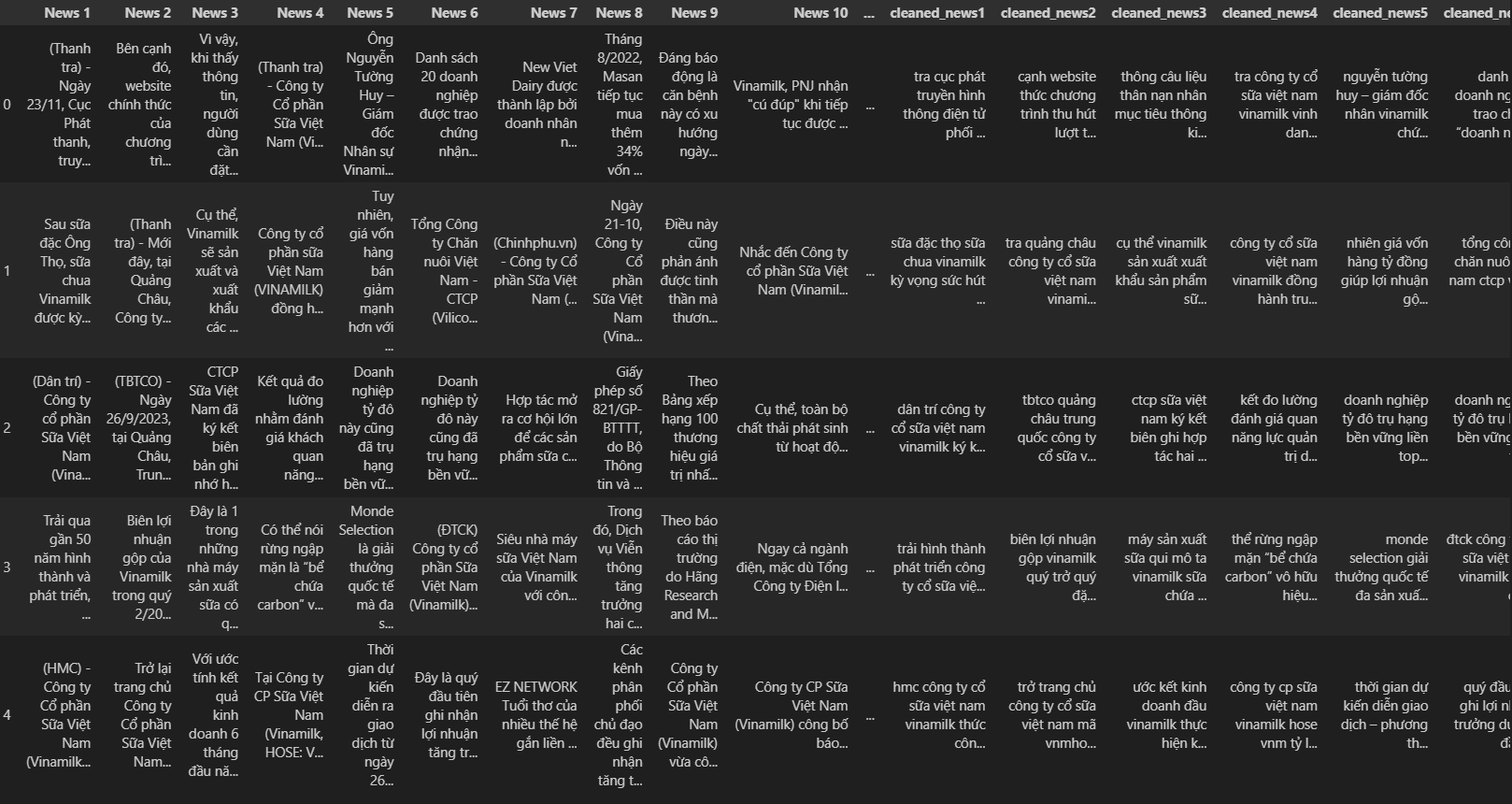
3‑6 Functions and lists to support feature engineering

I need to clean this data to create better features for my models.



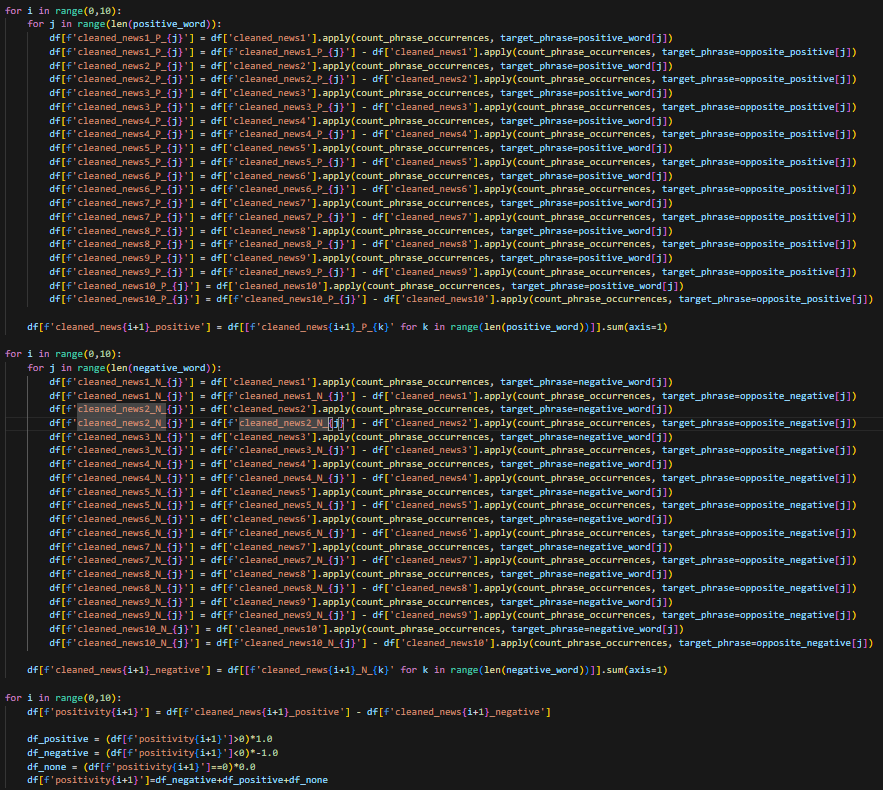
3‑7: Cleaning data

I cleaned it by lowercasing all the text, removing square bracket content, URLs, HTML tags, punctuation, newlines, alphanumeric words, and stopwords. Then, I joined it back into a single string and assigned it to the new columns I have created.



3‑8: Cleaning data results

Using functions to support feature engineering that I've built before, I applied them to each feature of the cleaned data. I created features of positive and negative points for each cleaned feature. Then, I summed each pair of these points of the same cleaned feature to create the positivity feature for each news feature from the base dataframe.



3‑9: Engineering Features

Next, I use the 'positivity' columns to create the features (X) and the 'Fluctuations' column to create the output (y) of the model. Then, I use the train\_test\_split() function to split it into a train set and a validation set.



3‑10: Positivity of each feature

Now that I have my cleaned X and y for training and validation sets, I used them to train my models.



3‑11: Results of the models

All results, although not high, are still acceptable. While the logistic regression model has the lowest accuracy results, the other three models show higher results that are quite similar. However, the SVM and Random Forest models appear to be overfit because the training set accuracy is much higher than the test set accuracy. Additionally, since the main objective is to correctly predict whether the stock price will increase or decrease (i.e., binary classification), rather than predicting the exact price, metrics like precision, recall, and F1-score become more important as they measure the model's ability to correctly classify price movements. The k-nearest neighbor model has the highest F1-score of all models. Therefore, I have chosen the k-nearest neighbor model for this project.

# Model Development

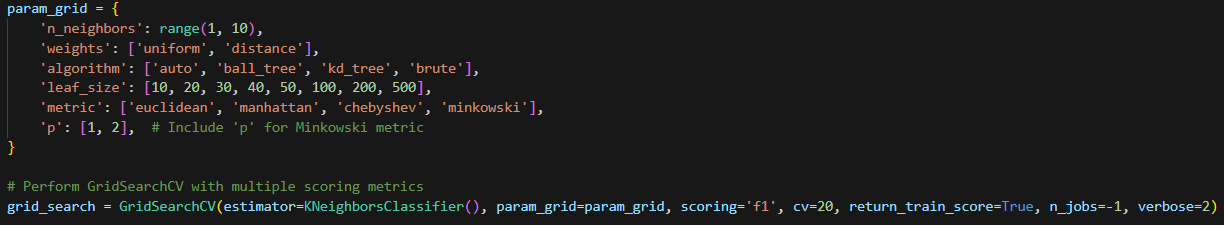
After choosing the k-nearest neighbor model for this project, I attempted to improve predictions by using the GridSearchCV package to tune hyperparameters such as 'n\_neighbors', 'weights', 'algorithm', 'leaf\_size', 'metric' and 'p' to find the best model based on new combinations of these hyperparameters.

GridSearchCV in scikit-learn is a method used for hyperparameter tuning through an exhaustive search over a specified parameter grid. It helps find the best set of hyperparameters for a model.

After fitting the GridSearchCV object to your data, you can access the best parameters and best estimator using 'best\_params\_' and 'best\_estimator\_', respectively. Use the best model for predictions or further analysis. Adjust the parameters and dataset according to your specific use case.

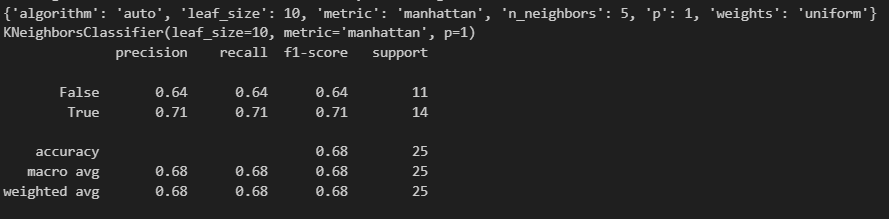
# Results

Set up hyperparameters and folds for cross-validation. The GridSearchCV package will help find the best combination of these hyperparameters to identify the best model.

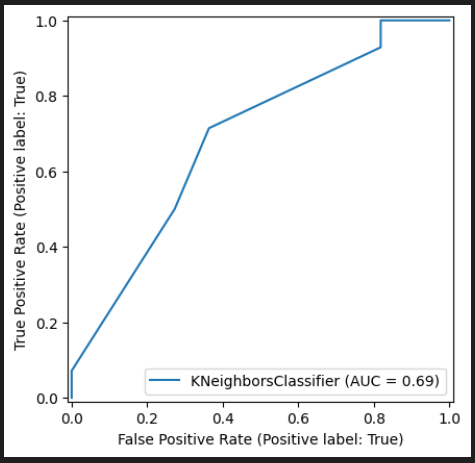


5‑1: Hyperparameters tuning by GridSearchCV

Here are the results of the best model found by GridSearchCV based on the combination of hyperparameters:



5‑2: New results of the tuned k-nearest neighbor model



5‑3: The ROC curve of tuned k-nearest neighbor model

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. It evaluates the performance of a classifier across different threshold values for binary classification problems.

# Conclusion

Overall, the results, although not high, are still acceptable. The relatively lower performance may be attributed to the small size of the dataset, indicating a need for more data to better train the model. However, the most significant factor could be that news is not the sole determinant of stock value changes; there are numerous other influential factors. Therefore, the current model's complexity might not be sufficient to accurately predict these changes.