Daily Realized Volatility Prediction of S&P 500 Index (SPX)

Valentin Aolaritei [1] **Alberto De Laurentis** [1] **Amirmahdi Hosseinabadi** [1]

Research conducted under the supervision of
Professor : Elise Marie Gourier
Teaching Assistant : Giuseppe Matera

1. Introduction and Research Questions

Volatility forecasting has increasingly attracted the attention of financial modelers, due to its extreme complexity, high non-linearity, and a high degree of temporal variability due to asymmetrical response to unexpected news [1].
The possibility to access high-frequency data (e.g. 5-10 minutes) has shown more evidence in how traditional econometrics models such as Generalized AutoRegressive Conditional Heteroskedasticity (**GARCH**) [2] could be useful to solve this task [3].
In recent years, researchers have also shown an increased interest in using Machine Learning (**ML**) and Natural Language Processing (**NLP**) models to extract relevant information from textual data such as news and use them as a proxy for this aim. Although these new techniques have achieved very good results, they attracted very little attention from the finance community and have been experimented only in the last period. The introduction of these new methods can accommodate a large number of predictors and their multi-way interactions and is useful in by-passing some problems encountered with previous econometrics models [4].
We decided to conduct our study of forecasting realized volatility on Standard & Poor's (S&P 500) Index (**SPX**) since it is the most important index in the NYSE, and a broad variety of studies have been conducted on it through the years.
The aim of our research addresses the following questions:

- **RQ 1:** In both univariate and multivariate modeling, can Machine Learning models show superior out-of-sample forecasting power compared to Econometrics models?

- **RQ 2:** Will macrofinancial indicators and sentiment analysis improve the models' power to forecast daily realized volatility?

[1]Group 3.

2. Literature Review

In this section, we overview and examine the historical evolution of financial econometrics and machine learning models for predicting realized volatility.
To this end, we conducted a wide research on GARCH models family [5] to understand how strong its predictability power was for S&P 500 options [6] and how it has been used for volatility forecasting.
We then looked for an appropriate multivariate financial econometrics model to accomplish our task, which ended up being the VectorAutroregression (**VAR**) model [7], which generalizes the single-variable (univariate) AR model.
To incorporate NLP models, and more specifically a classification task regarding **sentiment analysis**, we also reviewed some literature supporting a possible improvement in the prediction, especially when combining these techniques with Neural Networks family of methods.
Furthermore, ML provides a great variety of supervised learning methods, all showing different strong and weak points. In this study, we chose these methods for both univariate and multivariate tasks.
We began by analyzing the possible performance of linear regression, which is the simplest existent ML model, and later implemented some penalized models (e.g. **Ridge** and **LASSO**), which demonstrated superior performance in several studies [8],[9],[10].
Besides the linear regression field, also gradient descent decision tree methods were investigated. Guided by the findings in [11],[12], we selected eXtreme Gradient Boosting (**XGBoost**) due to its reported efficacy for tasks similar to ours.
Finally, we introduced artificial neural networks (**ANN**), because of their ability to work in synergy with sentiment analysis [13] and the vast majority of studies supported their superiority compared to any other model [14],[15].
Recurrent neural networks (**RNN**) were implemented in two different settings, specifically a basic one and a more complex Long Short-Term Memory (**LSTM**) because of its exceptional performance in volatility forecasting of **VIX**, which is a perfect proxy for S&P 500 Index [16],[17],[18].
LSTM was the only model used for both univariate and multivariate tasks, to understand if it unleashes a higher predictive power when trained with more factors and a larger dataset.

The last part of this review involves the choice of macrofinancial indicators, which will be deeply explained later in the report. To accomplish this task, the most relevant factors used in [19],[20] were selected. These factors were chosen for their ability to capture the U.S. Market's economic condition, as stated in the aforementioned studies.

3. Data Preprocessing and Visualization

3.1. Data Extraction

As the initial step for our project, we had to construct an appropriate dataset by taking data from different sources, since predicting volatility requires reliable and abundant data. However, the acquisition of this data presented quite a challenge. While several prominent platforms, such as *Yahoo Finance*, provided daily price data extending back several years, intra-day price data proved harder to come by.

3.1.1. Realized Volatility

The first source was the **Oxford-Man Institute's** *realized library*, which contains daily non-parametric measures of financial assets or indexes volatility between 03/01/2000 – 31/1/2020. This dataset comprehends daily realized variance for S&P 500 computed based on 5-minute realized return, rv_5.
The second source used to obtain our dataset was *Twelve Data*, a free data service provider from which we collected data between 01/12/2019 and 31/12/2023. This data reported prices of S&P 500 on a 5-minute frequency which were used as will be described later to compute daily realized volatility.
To merge the two datasets, their time zones were set to Eastern Daylight Time (**EDT**)[1], since *realised library* was set in Greenwich Mean Time (**GMT**)[2]. The consistency and reliability of the two datasets were checked by overlapping daily closing prices for the time window covered by both of them, i.e. 01/12/2019 - 31/1/2020.
Computing the daily realized volatility from the *realised library* was pretty straightforward and was done in the following way:

$$RV_t^{(d)} = \sqrt{rv_5} \tag{1}$$

Obtaining the same results for *Twelve Data* dataset was not as easy as before, since we had to calculate $RV_t^{(d)}$ by using the sum of intra-day squared log returns[3] based on closing prices as suggested in [21],[22].
Hence, with a Δ-frequency return, the standard definition (for an equally spaced returns series) of the realized volatility over a time interval of one day is

$$\Delta = \frac{1d}{M}$$

$$r_{t-j\cdot\Delta} = log\left(\frac{p(t-j\cdot\Delta)}{p(t-(j+1)\cdot\Delta)}\right) \tag{2}$$

$$RV_t^{(d)} = \sqrt{\sum_{j=0}^{M-1} r_{t-j\cdot\Delta}^2} \tag{3}$$

Where $r_{tj\cdot\Delta}$ defines continuously compounded Δ-frequency returns, that is, intra-day returns sampled at time interval Δ (here, the subscript t indexes the day while j indexes the time within the day t). Further, our research was focused on a smaller sample of the data collected, spanning from the beginning of January 2016 to the end of December 2023.
After this brief introduction regarding our target variable to forecast, it is important to state which other variables were chosen to create a multivariate time series model to accomplish the same task.

3.1.2. Macrofinancial Indicators

Our main goal in this part is to list the macrofinancial indicators engaged for the aim of this research and give an exhaustive description of their characteristics.
As stated before, our choice was guided by previous research and was widened by including also other proxies that seemed to be empirically significant from our point of view. Daily data for the traded factor were obtained from *Yahoo Finance* while research constructed indexes[4] were available at their respective websites. **Table 1** displays all those macrofinancial factors.

Table 1. **Macrofinancial Factors**

Name	Description
ADS Index	Aruoba, Diebold, and Scotti (2009) business conditions index.
EPU Index	Economic Policy Uncertainty index for U.S. Market (2016).[5]
VIX Index	Cboe Volatility Index based on S&P 500 index options.
S&P 500 Trading Volume	Daily trading volume of S&P 500 Index.
U.S. Dollar Index[6]	U.S.D.'s value relative to the majority of its significant trading partners.
13 Week Treasury Bill[7]	The 13 week yield on short term U.S. securities.
Term Spread	Long term yield on government bonds minus Treasury Bill.

[1]New York Time Zone
[2]London Time Zone
[3]Hereinafter referred as returns

[4]ADS and EPU Indexes

3.2. Data Visualization

Initially, we displayed daily returns of S&P 500 based on the last close price recorded for each day, because it gives a first insight into the volatility of the index and suggests the presence of volatility clustering (see **Figure 1**).

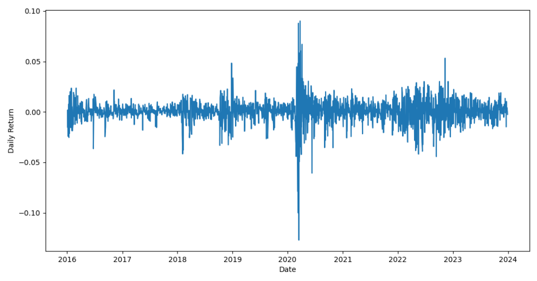


Figure 1. **Daily Return of S&P 500**

After this first plot, it was important to visualize the main feature of our study, the daily realized volatility of S&P 500, and compare how its trend matches the one of returns, as can be seen in **Figure 18**.

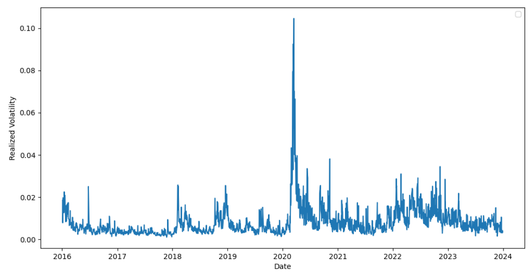


Figure 2. **Daily Realized Volatility of S&P 500**

Displaying a time series of macrofinancial indicators was even more interesting since some of them showed some really meaningful insight into U.S. Market condition that could not be spotted only by looking at S&P returns.
For example, **ADS** showed how fast the business cycle rebounded from the COVID-19 crisis in 2020, instead **13 Week Treasury Bill**, **Term Spread** and **U.S. Dollar Index** displayed perfectly the rise of inflation after Ukrainian War and the simultaneous hike of the interest rates from **Federal Reserves** to fight it in 2022. A comprehensive display of all the indicators is shown in **Figure 3**.

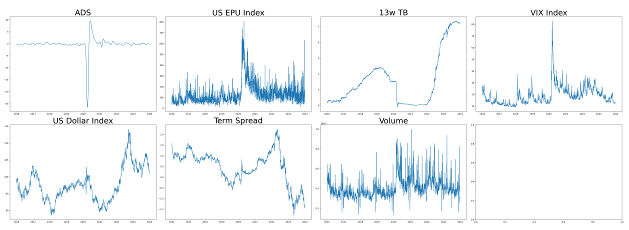


Figure 3. **Macrofinancial Factors**

3.3. Descriptive Statistics

In this section, we are going to give a more quantitative counterpart to the data visualized above by looking at descriptive statistics. These statistics were computed for returns, realized volatility, and as well for macrofinancial factors, to spot extreme outliers and to address precisely the magnitude of our data (see **Table 2**). After those steps, it was important to visualize correlations between the target variable and all the other variables, but also within the non-target variables themselves as shown in the heatmap in **Figure 4**.

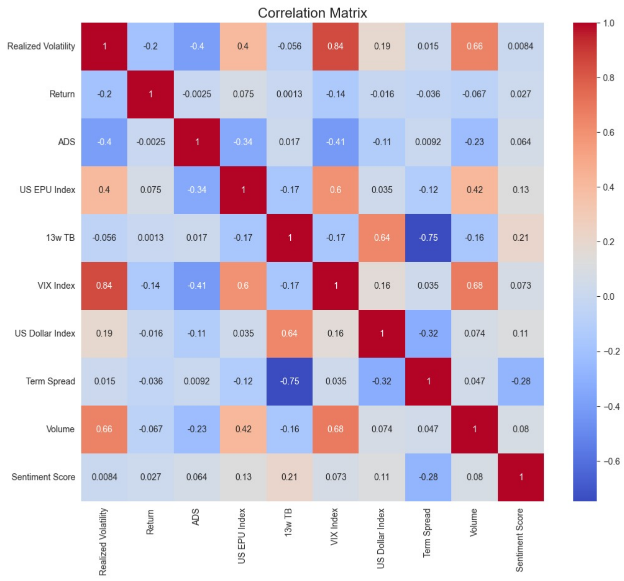


Figure 4. **Heatmap Correlation Matrix**

Statistics	Return	Volatility	ADS	EPU
Count	1881	1881	1881	1881
Mean	0.0004	0.0084	-0.2299	133.3155
Std	0.012	0.0073	3.2941	97.3572
Min	-0.1271	0.0011	-26.4456	10.92
25%	-0.0039	0.0042	-0.3593	73.6
Median	0.0007	0.0065	-0.0916	106.06
75%	0.0061	0.0101	0.2208	155.95
Max	0.0902	0.1046	9.4437	807.66

Statistics	Vix	Volume	USD	13w TB	Spread
Count	1881	1881	1881	1881	1881
Mean	19.1432	4.1×10^9	97.6357	1.589	0.4672
Std	7.9286	1.0×10^9	4.9882	1.671	0.7728
Min	9.14	1.3×10^9	88.59	-0.105	-1.779
25%	13.53	3.5×10^9	93.83	0.173	0.095
Median	17.42	3.9×10^9	96.96	1.045	0.71
75%	22.62	4.5×10^9	101.03	2.328	0.96
Max	82.69	10.0×10^9	114.11	5.348	2.245

Table 2. **Summary Statistics**

We included in this heatmap also sentiment scores obtained with our NLP model, which were adapted to our dataset and will be deeply described in **Section 4**.

3.4. Feature Engineering

Feature engineering is a crucial step in the data preprocessing phase of machine learning and data science. It involves creating new features or modifying existing ones to improve the performance of predictive models. We will explain the different steps we took for this part. **Feature Transformation** and **Data Normalization** are the only two steps in feature engineering that were applied to both multivariate time series models, econometrics, and ML ones.

3.4.1. Feature Transformation

In time series modeling, it is important to have stationary data. We ran the Augmented Dickey-Fuller (A**DF**) test on all the features. **U.S. Dollar Index**, **13 Week T-Bill**, and **Term Spread** failed to reject the null hypothesis of unit root, meaning that they are non-stationary. To solve this issue, these variables were substituted with their log returns which then resulted in stationary data checked again with ADF test.
Next, we decided to replace the raw returns and sentiment scores with their absolute values $|r_t|$, $|SentimentScore_t|$. The reason we did this is that the intensity of sentiments, and the size of price changes are more relevant than the direction in predicting realized volatility. This transformation aligns better with the nature of volatility, which measures the magnitude of price changes irrespective of their direction. After these steps, we visualized correlations between the new variables, as shown in the heatmap in **Figure 5**.

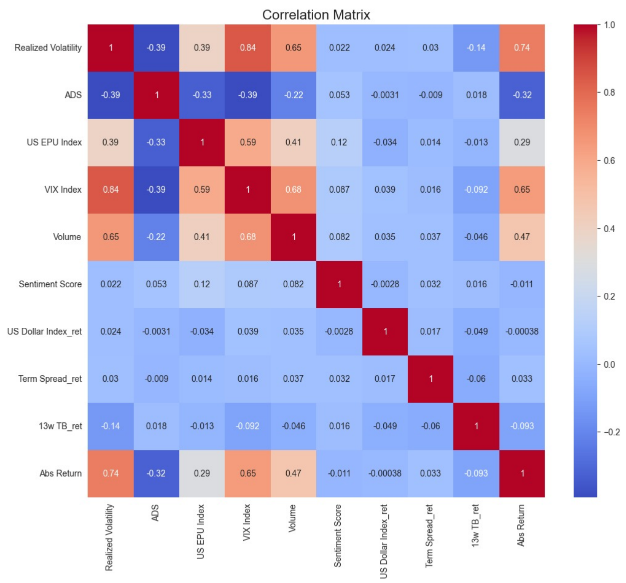


Figure 5. **Heatmap Correlation Matrix**

3.4.2. Feature Creation

When creating features for our time series forecasting model, we made sure not to use any data from time t when predicting at time t. Instead, we used the **first and second lags** of both the explanatory and target variables. This way, the model bases its predictions on past values, keeping things causally correct.
We took as an assumption that our data had temporal dependencies. This seems like a legitimate assumption, as stock price volatility generally exhibits time-series properties, due to volatility being affected by many factors, such as macroeconomic conditions or market sentiment, which are time-dependent. We also added the **square of the first lag** of all variables to our features. This helps the model capture some non-linear relationships and interactions in the data, making our predictions more accurate.

3.4.3. Feature Selection

As our final step, we applied **backward elimination** for feature selection after creating and transforming our features. This method involves starting with all potential features and iteratively removing the least significant ones, based on statistical criteria, until only the most impactful features remain. It is important to state that for regression backward elimination was applied on lagged values and the square of the first lag of variables to address non-linearity. Instead for ML models only lagged values were used.

3.4.4. Data Normalization

MinMaxScaler, a popular technique used in machine learning, was implemented to scale features to a fixed range. It makes it easier to compare different features that may originally have different scales. This normalization prevents features with larger scales from dominating the model training process, enhances numerical stability, and improves convergence speed.

3.5. Datasets

When developing predictive models, different datasets are often used to suit the specific needs of each model. For instance, techniques like backward elimination can be employed to select key features, which are then used in models such as linear regression or machine learning algorithms to enhance performance and interpretability. Conversely, models like Vector Autoregression (VAR) might utilize raw data to fully capture the complex interdependencies among variables. This tailored approach ensures that each model leverages the most appropriate data, optimizing accuracy and effectiveness. Here we will resume each dataset used for each model (see **Table 3**).

Model	Dataset
Univariate	
Garch	Daily Realized Returns
Lstm	Daily Lagged Realized Volatility
Multivariate	
VAR	Raw data transformed
LR	Features selected by Backward Elimination on Lagged values and Powers
Ridge	All features transformed with their lagged values and powers
Lasso	All features transformed with their lagged values and powers
RF	Features selected by Backward Elimination on Lagged values
Xgboost	Features selected by Backward Elimination on Lagged values
Rnn	Features selected by Backward Elimination on Lagged values
LSTM	Features selected by Backward Elimination on Lagged values

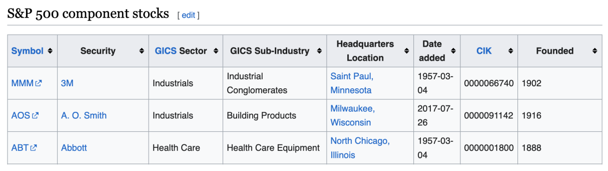
Table 3. **Dataset Choice**

4. Sentimental Analysis

Before diving into the univariate time series modeling and introducing multivariate models, we would like to shift our attention to the NLP model we introduced, and to the implementation of sentiment analysis, since it is an extremely important milestone for this project.
Sentiment analysis is the field of studies which analyzes people's opinions, attitudes, emotions, and evaluations. By applying it to financial markets, it can be used as a tool to make better decisions and to evaluate financial forecasting [23].
In this project, this technique was employed to determine whether financial headlines regarding the S&P 500 underlying stocks can improve volatility forecasting. To reach this goal, Valence Aware Dictionary for sEntiment Reasoning (**VADER**) model was used.

4.1. Data Extraction

To run a sentimental analysis, firstly a large quantity of data regarding the S&P 500 was required. However, every year the S&P 500 stocks may change due to performances or other events, such as bankruptcy. Hence, firstly we opted to scrap all stocks that belong to the index up from *Wikipedia*. By doing so, we were sure about reliability since Wikipedia is updated regularly. Plus, we could also extract stock information such as ticker, name, GICS sector, and IPO date, as can be observed in **Figure 6**.
After obtaining the tickers, news headlines were scraped from *Business Market Insider*, a multinational business and financial news website. The choice fell on this website because of its reliability and large, evenly distributed quantity of data during the period of the study, 2016-2023. As can be observed in **Figure 7**, the website provides news from diverse providers, among which *Benziga* and *Seeking Alpha*.


S&P 500 component stocks [edit]

Symbol	Security	GICS Sector	GICS Sub-Industry	Headquarters Location	Date added	CIK	Founded
MMM	3M	Industrials	Industrial Conglomerates	Saint Paul, Minnesota	1957-03-04	0000066740	1902
AOS	A. O. Smith	Industrials	Building Products	Milwaukee, Wisconsin	2017-07-26	0000091142	1916
ABT	Abbott	Health Care	Health Care Equipment	North Chicago, Illinois	1957-03-04	0000001800	1888

Figure 6. **Wikipedia : S&P 500 Component Stocks example**

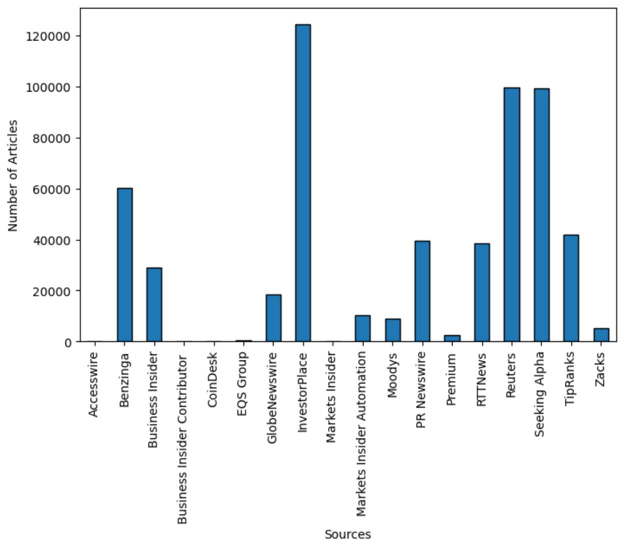


Figure 7. **Number of News for each Source**

Additionally, **Figure 8** shows that the number of articles per year in the last 8 years was very large, which allowed us to use these data for sentimental analysis.
Hence, using *Market Business Insider* for news scraping was a reasonable and appropriate choice for this project. Furthermore, among all years and months within each year, the number of news seems approximately uniformly distributed, as **Figure 9** and **Figure 10** indicate.

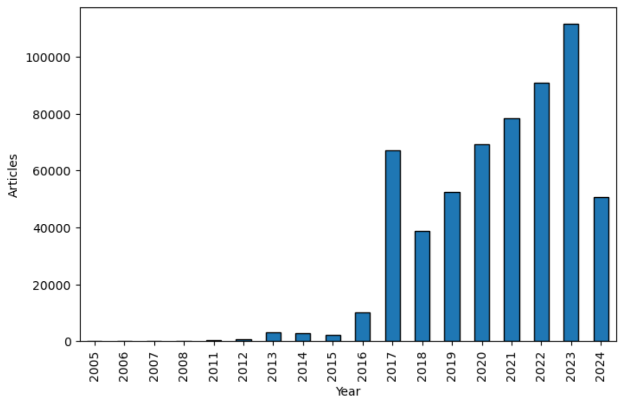


Figure 8. **Number of News per Year**

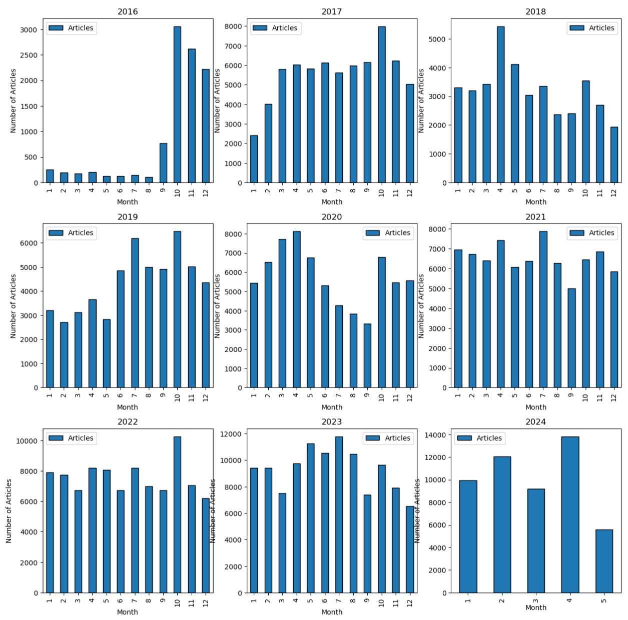


Figure 9. **Number of News through every Year**

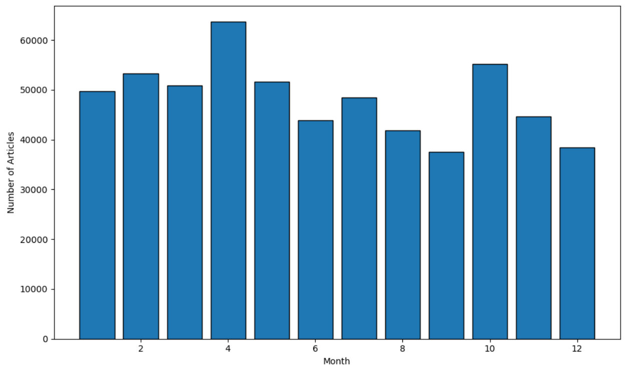


Figure 10. **Number of News through every Month**

4.2. Data Analysis

After choosing the source data for sentiment analysis, we started to scrap news regarding all S&P 500 stocks, totaling $581'707$ news during the period $2005-2024$. However, only a few stocks had a large volume of news, as shown in **Figure 11**.

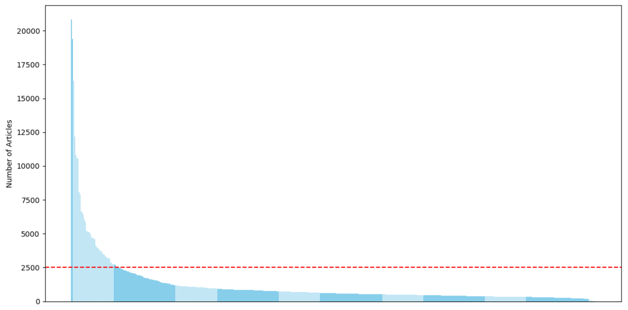


Figure 11. **Distribution of News for each Stock**

Specifically, **Figure 12** depicts that only 43 stocks have more than $2'500$ news.

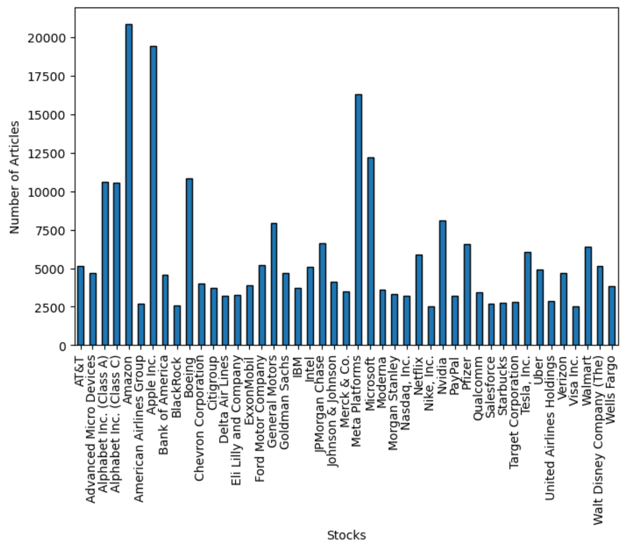
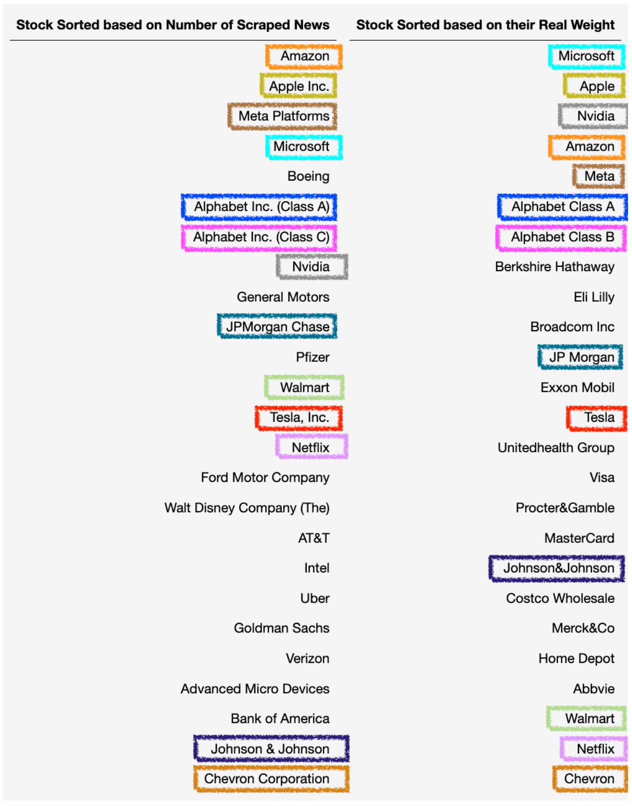


Figure 12. **Stock with more than 2'500 News**



Stock Sorted based on Number of Scraped News	Stock Sorted based on their Real Weight
Amazon	Microsoft
Apple Inc.	Apple
Meta Platforms	Nvidia
Microsoft	Amazon
Boeing	Meta
Alphabet Inc. (Class A)	Alphabet Class A
Alphabet Inc. (Class C)	Alphabet Class B
Nvidia	Berkshire Hathaway
General Motors	Eli Lilly
JPMorgan Chase	Broadcom Inc
Pfizer	JP Morgan
Walmart	Exxon Mobil
Tesla, Inc.	Tesla
Netflix	Unitedhealth Group
Ford Motor Company	Visa
Walt Disney Company (The)	Procter&Gamble
AT&T	MasterCard
Intel	Johnson&Johnson
Uber	Costco Wholesale
Goldman Sachs	Merck&Co
Verizon	Home Depot
Advanced Micro Devices	Abbvie
Bank of America	Walmart
Johnson & Johnson	Netflix
Chevron Corporation	Chevron

Figure 13. **Comparison of Stock with highest Number of News and Stock by Market Cap**

Those findings drove us to focus our attention more on understanding deeply the S&P 500 composition since it is a market-weighted index. A closer look at these figures indicates that the stocks with the most news coverage seem to

correspond to the Top 50[8] of the S&P 500 based on market capitalization. To address for clarity and readability, but still give an overall idea, we decided to display only the Top 25 stocks in **Figure 13**.
In fact, there are only a few outliers in the healthcare, automobile, and aviation manufacturing sectors, such as Pfizer (50° position), Uber, Ford, General Motors, and Boeing. Most probably the first two recorded a high number of news due to the COVID-19 crisis, while Ford and GE were highly involved in the green transactions process. Finally regarding Boeing, the situation is more complex and can be explained by some fatal crashes in the last years.
Given that the Top 50 stocks have more news coverage and greater market weights, we decided to conduct the sentiment analysis only on those news (221'168) and come out with an idea to compute weighted scores.

4.2.1. Weight Computation

To achieve this task, we opted to use the market capitalization data from *CRSP*, sourced from the *Wharton Research Data Service*, whose access is given by EPFL. More precisely, the daily total market capitalization of the Top 50 and the S&P 500 Index were analyzed during the period of our choice, i.e. 2016-2023.
Subsequently, the ratio between the aggregate market capitalization of the Top 50 and the market capitalization of the S&P 500 was computed daily. Finally, we took the mean of these values, which corresponds to $\approx 50\%$. This indicates that the market capitalization of the first 50 firms represents on average half of the total market capitalization of the S&P 500, justifying the weights for their news.
Those weights were daily computed as:

$$w_{stock_{i,t}} = \frac{Market\ Cap_{stock_{i,t}}}{\sum_{j=1}^{50} Market\ Cap_{stock_{j,t}}} \tag{4}$$

$\forall i \in$ Top 50.

In order to simplify computational issues, the mean of the weights of the stocks for the entire period were obtained as

$$\bar{w}_{stock_i} = \frac{w_{stock_{i,t}}}{t} \tag{5}$$

The weights of the first half of the firms in the Top 25 can be observed in **Figure 14** as a representative example.

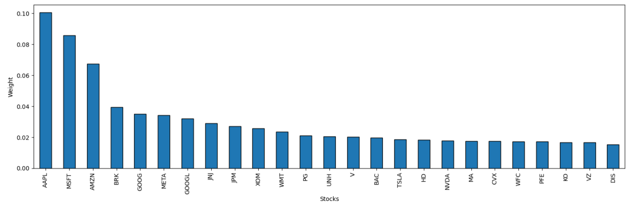


Figure 14. **Weights Distribution for the Top 25**

[8]Hereinafter referred as Top 50.

4.3. VADER

VADER is a lexicon-based sentiment analysis method specifically designed for analyzing sentiment in social media and blog texts [24]. For this reason, it can deal with the unique characteristics of social media language, such as slang, emoticons, and abbreviations. Additionally, it also takes into account factors like capitalization, punctuation, and degree modifiers as in **Figure 15**. For instance, a phrase containing "very" will have a lower sentiment score compared to the same phrase where "very" is replaced by "extremely".

Input	neg	neu	pos	compound
"This computer is a good deal."	0	0.58	0.42	0.44
"This computer is a very good deal."	0	0.61	0.39	0.49
"This computer is a very good deal!!"	0	0.57	0.43	0.58
This computer is a very good deal!! :-)"	0	0.44	0.56	0.74
This computer is a VERY good deal!! :-)"	0	0.393	0.61	0.82

Figure 15. **VADER Score Example**

This model provides four scores as output: **positive**, **neutral**, **negative**, and **compound**. The positive, neutral, and negative scores are all bounded between 0 and 1. The compound score is calculated by summing the valence scores of each word in the lexicon and then normalizing the result to be between -1 and +1. The compound score indicates negative sentiment if its value is $\leq$ -0.05, positive if its value is $\geq$ 0.05, and neutral otherwise.

4.3.1. Sentiment Score

Firstly, sentiment scores were obtained by using VADER for the multiple news for each stock. Those were normalized to have a single daily value for each stock in the 2016-2023 period.
To incorporate the weights computed in 4.2.1, the daily sentiment scores for each stock obtained previously were multiplied by their respective weights and then summed to get an aggregate daily sentiment score for the index. This can be expressed as

$$Sentiment\ Score_t = \sum_{i=1}^{50} \bar{w}_{stock_i} \times \overline{Sentiment\ Score}_{stock_{i,t}} \tag{6}$$

This adjustment applies to every stock in the Top 50, while for all other stocks, the given weight is zero, implying that their sentiment scores are set to zero. The most negative score was recorded in 2020 during the COVID-19 crisis. Additionally, one of the lowest compound scores was also noted during this period, and to better understand the magnitude of the aggregate compounded sentiment scores, summary statistics are collected in **Table 4** and the time series trend is plotted in **Figure 16**.

STATISTICS	POSITIVE SCORE	NEUTRAL SCORE	NEGATIVE SCORE	COMPOUND SCORE
COUNT	2874	2874	2874	2874
MEAN	0.0565	0.4413	0.0258	0.0500
STD	0.0374	0.2166	0.0175	0.0601
MIN	0.0000	0.0000	0.0000	-0.1536
25%	0.0256	0.2621	0.0126	0.0051
MEDIAN	0.0544	0.4925	0.0246	0.0436
75%	0.0823	0.6261	0.0366	0.0888
MAX	0.1796	0.8612	0.0956	0.3258

Table 4. **Summary Statistics Sentiment Score**

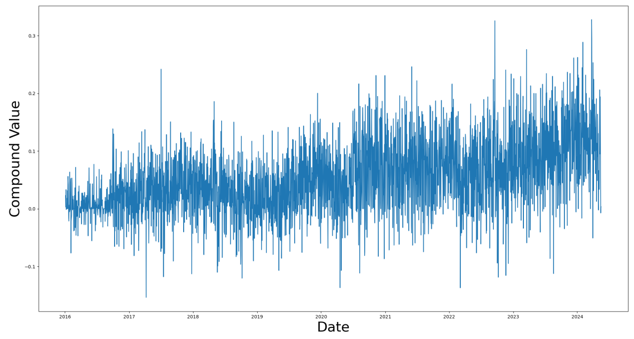


Figure 16. **Daily Compound Sentiment Score**

The impact of COVID-19 and the Russia-Ukraine war is visible in 2020 and 2022 respectively, as can be observed in **Figure 17**, where the monthly average of sentiment score is displayed as a times series.

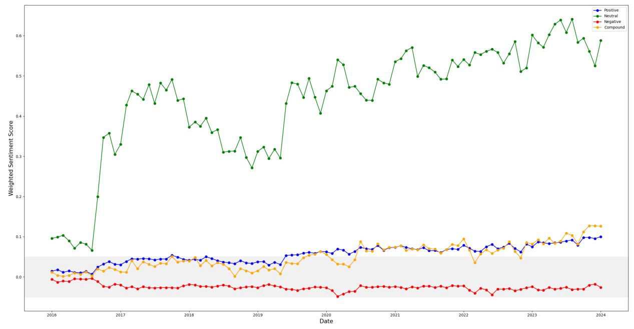


Figure 17. **Average Monthly Sentiment Score**

4.4. Sentiment Score Adaptation

A last remarkable consideration about our sentiment scores is that they were manipulated in order to be merged with the rest of the dataset since news was available for almost all days in each year instead, all the other variables are computed only on 252 working days each year. This happens because exchanges are closed during weekends and several other days during the year.

5. Methodology

Given the data that we obtained, next, we wanted to test the predictive power of different models. For all the models, we used the same methodology for training and testing models which we will explain now.

5.1. Data Split

As can be seen in Figure 18, we split our dataset into training and test sets. We decided to consider the first 90% of the dataset as the training set, i.e. from 04/01/2016 to 31/03/2023, and the remaining 10% as the test set for our forecasting, i.e. from 01/04/2023 to 29/12/2023. It is important to note that the test set was only accessed after the training phase to ensure the models' performance is evaluated on data it has not seen. The training set was handled differently for GARCH and ML models, and insights will be explained in **Section 5.3**.

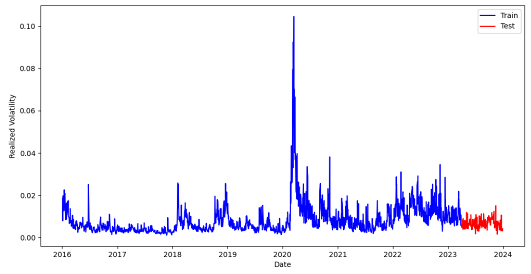


Figure 18. **Training and Test Split for Daily Realized Volatility**

5.2. Evaluation Metrics

For all models, We decided to compute the following 4 metrics for both the training and testing:

- $\boldsymbol{R^2}$ statistic measures the proportion of total variance of **Y** that is explained by linear relationship between **X** and **Y**

$$R^2 = 1 - \frac{RSS}{TSS} \tag{7}$$

where RSS is the residual sum of squares, and TSS is the total sum of squares. The higher the better.

- Root mean square error (**RMSE**) which is defined as

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(RV_t^{(d)} - \sigma_t^{(d)})^2} \tag{8}$$

where $RV_t^{(d)}$ is the actual value of realized volatility at time t, $\sigma_t^{(d)}$ is the predicted value of realized volatility at time t. The lower the better.

- Mean absolute error (**MAE**) which punishes large errors less than **RMSE** and is defined as

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|RV_t^{(d)} - \sigma_t^{(d)}| \tag{9}$$

The lower the better.

- Mean absolute percentage error (**MAPE**)

$$MAPE = \frac{100}{T}\sum_{t=1}^{T}\left|\frac{RV_t^{(d)} - \sigma_t^{(d)}}{RV_t^{(d)}}\right| \tag{10}$$

The lower the better.

It is further important to state that the main discussions and findings will be based on R^2, since it is the most explainable criteria.

5.3. Hyperparameter Tuning & Cross Validation

Hyperparameter fine-tuning involves adjusting the model's pre-set parameters, such as learning rate, number of layers, or regularization strength, to improve its performance. Hyperparameters are not learned during training but are set before the learning process begins. This process often uses systematic search methods like grid search to explore different hyperparameter combinations.
Time series cross-validation is a crucial technique for hyperparameter tuning in models involving temporal data. It is a more robust approach compared to the naive approach of having only one validation set and unlike standard cross-validation, which randomly splits data into training and validation sets, time series cross-validation respects the sequential order of data.
The training set was split by following cross-validation on a rolling basis [25] as shown in Figure 19. We then chose the set of hyperparameters which gave the minimum amount of cross-validation error. We used 5-split cross-validation in our implementation and used the Mean Absolute Percentage Error (**MAPE**) as the cross-validation error.

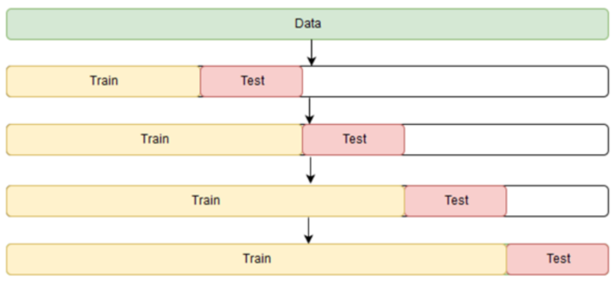


Figure 19. **Cross-Validation on Rolling Basis**

6. Univariate Models

We started our volatility forecasting with the univariate forecasting models, which are the most simple ones. First, we used the GARCH model which is the main econometric model for volatility analysis. Secondly, as a machine learning approach, we used the univariate LSTM model. Given the time series of realized volatility, the LSTM model would learn to predict the realized volatility at time t, $RV_t^{(d)}$, using the first lag $RV_{t-1}^{(d)}$. By doing so, first of all, we would be able to compare the out-of-sample predictive power of univariate econometrics and machine learning models[9]. Secondly, they would serve as benchmark models, which would be compared with multivariate models. This comparison would enable us to evaluate whether incorporating additional explanatory variables to econometric and machine learning models would improve their predictive power. Next, we will explain our methodology and results for the GARCH, instead a deep understanding of LSTM will be given in **Section 8**.

6.1. GARCH (*p,q*)

The GARCH (p, q) process, Generalized Autoregressive Conditional Heteroskedasticity, as proposed by Bollerslev (1986), is given by:

$$r_t = \mu_t + \epsilon_t, \quad \epsilon_t = u_t\sigma_t, \quad u_t \sim f(x),$$

$$\sigma_t^2 = \omega + \sum_{i=1}^{p}\alpha_i\epsilon_{t-i}^2 + \sum_{j=1}^{q}\beta_j\sigma_{t-j}^2. \tag{11}$$

$$\omega > 0, \alpha_i \geq 0, \beta_j \geq 0, \quad i = 1, \ldots, p, \quad j = 1, \ldots, q.$$

Where $f(x)$ represents the distribution of the shock considered in the model. When the model was first proposed, the author used a **Standard Normal** distribution, but it is possible to use other shock distributions, to allow for fatter tails or additional skewness, such as the **Standard Student-t** distribution and **Standard Skewed Student-t** distribution. In the case where $q = 0$, the model simplifies to an ARCH (p) model. The main difference between ARCH and GARCH models is that the latter allows lagged conditional variance to influence the conditional variance σ_t^2.
This model can be seen as an ARMA model for squared returns. An important aspect of this model that needs to be underlined is that we have to impose positivity constraints on the parameters of the process to guarantee the positivity of the estimated conditional variance.
The GARCH model is the simplest model considered in this analysis. First, we will expose what statistical properties our time series requires to fit the GARCH model and whether or

[9]GARCH model uses the single time series of returns, and LSTM uses the single time series of first lagged values of realized volatility.

not our data satisfies these conditions. Then, we will explain our model selection process and last we will go through our methodology for out-of-sample prediction.

6.1.1. Statistical Requirements

The initial requirement for time series analysis (including GARCH model) is the stationarity of the time series. For this reason, we tested the **stationarity of returns** using an Augmented Dickey-Fuller (ADF (1979)) test. The ADF test suggested rejecting the null hypothesis of a unit root, thereby indicating that the time series was stationary.
Another requirement of the GARCH model is that the **residuals should be white noise**. Using an ARIMA-GARCH model instead of a simple GARCH model is crucial when there is autocorrelation in returns, because ARIMA effectively captures the mean dynamics, including trends and autocorrelation, which a simple GARCH model cannot.
By modeling the mean process with ARIMA, the residuals become closer to white noise, making them more suitable for GARCH to model the volatility clustering. So first, we checked the autocorrelation in returns by plotting the Autocorrelation function (**ACF**) and partial autocorrelation function (**PACF**) for different lags of r_t.

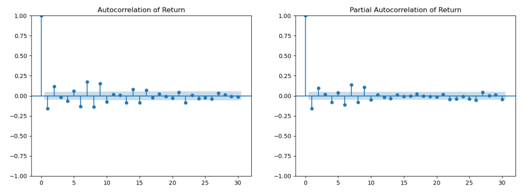


Figure 20. **ACF & PACF plot for returns**

As shown in **Figure 20**, we encountered results significantly different from zero for several lags. To spot those results not only visually but also quantitatively, a **Ljung-Box Test** for serial autocorrelation for error terms was applied and we rejected the null hypothesis of no autocorrelation. Hence, the returns were not white noise and, as previously explained, modeling the mean of returns with ARIMA models was required.
For different (*p, d, q*)[10] tuples, we fitted an ARIMA (*p, d, q*) models to the training set and chose the best model based on information criteria: **AIC**[11](Akaike, 1973), **BIC**[12](Schwarz, 1978). The best model was the **ARIMA (0, 0, 2)** model, equivalent to a MA(2) model.
As we just mentioned, GARCH is based on the assumption that residuals are white noise. So, we re-tested this assumption by using the Ljung-Box test and the ACF, PACF plots(recorded in **Figure 21**), but this time on the residuals of the MA(2) model.

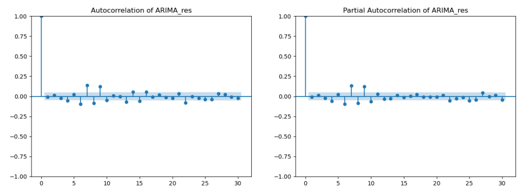


Figure 21. **ACF & PACF for ARIMA residuals**

Although the autocorrelation was removed from the first few lags, we still encountered autocorrelation in bigger lags which may suggest that **GARCH model would not perfectly work for the task.**
The last test was about **autocorrelation in the squared residuals**. GARCH models are designed to capture conditional heteroskedasticity, where the variance of the residuals varies over time, often exhibiting periods of high and low volatility clustering together. Hence, another assumption of the GARCH model is that there should be autocorrelation in the squared residuals. We observed that this condition was satisfied by applying the Ljung-Box test and the ACF/PACF plots (see **Figure 22**).

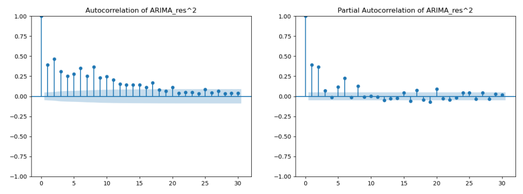


Figure 22. **ACF & PACF for ARIMA residuals squared**

6.1.2. Model Selection

After exploiting all the precedents statistical requirements of the GARCH model, we can select the appropriate GARCH model. We employed a systematic approach using the residuals from an ARMA (0,2). We performed a grid search over a range of GARCH model parameters (*p, q*) and different distributions. For each combination of parameters, we used a cross-validation on a rolling basis approach to fit the model to the training data and forecast the volatility over the validation period. The performance of each model was evaluated using the metrics explained in part 5.2. By comparing these metrics, we identified that the **GARCH (1,2)** model with **Standard Skewed Student-t distributed** residuals provided the best performance across different criteria.
After fitting the chosen model to the residuals, to validate our model we executed an **ARCH effect test** (Engle (1982)) to check if the squared residuals were correlated. This was the case for the selected model.

[10]p: Autoregressive (AR) order, d: Differencing order, q: Moving Average (MA) order
[11]AIC (M$_k$) $= -2\log L(M_k) + 2k$
[12]BIC (M$_k$) $= -2\log L(M_k) + \log(n)\cdot k$

6.1.3. Evaluation

After selecting the best model, we evaluated the out-of-sample performance on the test set. Similar to the approach used in model selection, we did the forecasting on a rolling basis. The model's predictions were compared to the actual realized volatility in the test set. The results are shown in **Section 9**.

6.2. LSTM

The second univariate model that we used was the LSTM model. The structure of LSTM models will be explained in 8.3.2. We used the same methodology as for other machine learning models, which will be demonstrated in **Section** 8. The out-of-sample performance of this model is shown in **Section 9** as well.

7. Multivariate Econometric Models

In this section, we will focus on the Vector Autoregression (VAR) model, which is a multivariate econometric modeling method for volatility forecasting. By doing this, first of all, we were able to compare it with the GARCH model. Second, we could compare it with machine learning methods which we will explain in **Section 8**. We will start by explaining the VAR model and then go through the statistical requirements of this model.

7.1. VAR (*p*)

Vector Autoregression VAR (p) is a stochastic process model designed in the following way:

$$\mathbf{y}_t = \boldsymbol{\phi}_0 + \boldsymbol{\phi}_1 \mathbf{y}_{t-1} + \boldsymbol{\phi}_2 \mathbf{y}_{t-2} + \ldots + \boldsymbol{\phi}_p \mathbf{y}_{t-p} + \boldsymbol{\epsilon}_t \quad (12)$$

where:

$\mathbf{y}_t$ is a $k \times 1$ stochastic vector,

$\boldsymbol{\phi}_0$ is a $k \times 1$ vector or intercepts,

$\boldsymbol{\phi}_j$ are $k \times k$ matrices of parameters for $j = 1, \ldots, p$,

$\boldsymbol{\epsilon}_t$ is a $k \times 1$ vector white noise with $\mathbb{E}_{t-1}[\boldsymbol{\epsilon}_t] = 0$.

VAR model is a powerful tool in time series analysis used to capture dynamic interrelationships among multiple variables. In a VAR model, each variable is regressed on its lagged values as well as the lagged values of all other variables in the system. This means that each variable in the system is modeled as a linear function of its past values and the past values of all other variables in the system. One of the key advantages of VAR models is their ability to capture complex interactions and feedback mechanisms among variables without imposing strict structural assumptions, making them versatile tools for understanding and forecasting multivariate time series data.

7.1.1. Statistical Requirements

The first requirement for the VAR model, similar to the GARCH model, is the **stationarity** of the data. As we mentioned in 3.5, we use the transformed stationary data so that this requirement is fulfilled.
The next step in the VAR model is to apply **Granger Causality** test. Even if some variables had high p-values, they were not discarded, since we were not interested in the exogeneity of variables, but more in their economic background.

7.1.2. Model Selection

After fulfilling the statistical requirements, we fitted VAR models with different lag lengths to the training data and chose the one that fitted the best based on several information criteria (AIC, BIC, etc.). The best model was a VAR(9), which means that our target variable, daily realized volatility, was regressed on its first 9 lagged values and also on all the first 9 lagged values of macrofinancial indicators and modified sentiment score, as explained in 3.5.

7.1.3. Evaluation

After selecting the best model, we evaluated the out-of-sample performance on the test set. Similar to the approach used in GARCH, we did the forecasting on a rolling basis. The model's predictions were compared to the actual realized volatility in the test set. The results are shown in **Section 9**.

8. Multivariate Machine Learning Models

In this section, we are going to discuss machine learning models, which represent a dynamic counterpart to the traditional econometric models explained before. ML models offer a distinct advantage by leveraging the flexibility and complexity of algorithms to capture intricate patterns in data.
By incorporating features such as non-linearity, high dimensionality, and interactions among variables, ML models can often outperform their econometric counterparts, especially in scenarios where data exhibits non-linear and time-varying behavior.

8.1. Regression Models

Here, we will analyze the regression models starting from the simplest model, linear regression, to more advanced techniques such as Ridge and LASSO.

8.1.1. Linear Regression

Linear regression is a fundamental statistical method used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. In its simplest form, linear regression assumes a linear relationship between the input variables and the target variable, represented by a straight line as

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \epsilon \tag{13}$$

where $X = (X_1, ..., X_p)$ contains the explanatory variables and ϵ is the "error" term.
The goal of linear regression is to find the best-fitting line that minimizes the difference between the observed values and the values predicted by the model as

$$\min_\beta \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 = \min_\beta ||\mathbf{y} - \mathbf{X}\beta||^2 \tag{14}$$

Despite its simplicity, linear regression remains a powerful and interpretable tool for both predictive modeling and inferential analysis. It is important to state which variables were chosen for this model through backward elimination, which are recorded in the following **Table 5**.

VAR.	|RETURN|	VOLATILITY	ADS	EPU	VIX
T-1	X	X	-	X	X
T-2	-	X	-	-	-
T-1 & ^{2}	-	-	-	-	-

VAR.	|SENT.|	VOLUME	USD	13W TB	SPREAD
T-1	X	-	-	X	-
T-2	-	X	-	X	X
T-1 & ^{2}	-	-	-	-	-

Table 5. **Cells with X represent variables selected with Backward Elimination for LR**

8.1.2. Ridge Regression

For Ridge regression, since it is a regularized method, backward elimination is not required in the preprocessing step. Hence, we used all scaled features as inputs without backward elimination.
Ridge regression is a special case of Tikhonov regularization used when having too many predictors, or with a high degree of multi-collinearity between each other and helps avoid overfitting.
It adds a penalty term to the ordinary least squares (**OLS**) objective function, known as the ridge penalty or **L2 regularization** term, which penalizes large coefficients by shrinking them towards zero. This penalty term is controlled by a hyperparameter λ (**lambda**).
As λ increases, the impact of the penalty term becomes stronger, leading to smaller coefficients and a simpler model.
The model can be resumed by the following equation

$$\hat{\beta}_R = \arg\min_\beta \left(\frac{1}{2}||\mathbf{y} - \mathbf{X}\beta||^2 + \frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2 \right) \tag{15}$$

However, it is essential to tune the regularization parameter carefully with cross-validation to find the right balance between bias and variance, as too much regularization can lead to underfitting. Regularization parameter λ was set equal to 0.1.

8.1.3. Lasso Regression

As for Ridge, also this time we don't need backward elimination because Lasso itself has the feature selection approach. LASSO regression, short for **Least Absolute Shrinkage and Selection Operator**, is another regularization technique used in linear regression. Similar to Ridge regression, LASSO regression adds a penalty term to the ordinary least squares (OLS) objective function.
However, instead of using the L2 norm penalty as in Ridge regression, LASSO regression employs the **L1 norm penalty**.

$$\min_\beta \left(\frac{1}{2}||\mathbf{y} - \mathbf{X}\beta||^2 + \lambda||\beta||_1 \right), \quad ||\beta||_1 = \sum_{j=1}^{n}|\beta_j| \tag{16}$$

Like Ridge regression, LASSO regression also requires tuning of the regularization parameter λ to balance between model complexity and prediction accuracy. Regularization parameter λ was set equal to 0.0001. This results in a different shrinkage behavior: while Ridge regression shrinks the coefficients towards zero, LASSO regression has the additional property of performing variable selection by pushing some coefficients exactly to zero as shown in **Table 6**.

Table 6. **Cells with X represent variables selected from LASSO**

VAR.	|RETURN|	VOLATILITY	ADS	EPU	VIX
T-1	X	X	X	X	X
T-2	X	X	-	X	X
T-1 & ^{2}	-	-	X	-	-

VAR.	|SENT.|	VOLUME	USD	13W TB	SPREAD
T-1	X	-	-	-	-
T-2	-	-	X	-	X
T-1 & ^{2}	-	X	-	-	-

This makes LASSO regression particularly useful for feature selection, as it can automatically select the most relevant features and discard irrelevant ones.

8.2. Decision Tree-based Models

Decision tree models are versatile and interpretable machine learning algorithms used for both classification and regression tasks. They operate by recursively partitioning the input space into regions based on the values of input features, with each partition represented by a tree node.
At each node, the algorithm selects the feature and threshold that best splits the data into subsets that are as homogeneous as possible concerning the target variable. This process continues recursively until a stopping criterion is met, such as reaching a maximum tree depth, having a minimum number of samples in each leaf node, or no further improvement in purity can be achieved.
The resulting tree structure can be visualized and easily interpreted, making decision trees popular for tasks requiring transparent and explainable models. Moreover, decision trees can handle both numerical and categorical data, handle interactions between features, and are robust to outliers and missing values. Next, we will go through 2 different decision-tree-based approaches that we used.

8.2.1. Random Forest

Random Forest is a versatile and powerful ensemble learning technique used for both classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputs the mode or average prediction of the individual trees for classification or regression, respectively.
What makes Random Forest unique is its use of randomness in both feature selection and tree construction. At each split in each decision tree, a random subset of features is considered, which helps to decorrelate the trees and reduce overfitting. Additionally, each tree is trained on a bootstrapped sample of the original dataset, further adding randomness to the process.
This ensemble approach leads to robust and accurate predictions, even in the presence of noisy data or outliers. Random Forests are known for their flexibility, scalability, and ability to handle high-dimensional datasets with ease. Tuning hyperparameters plays a very important role in Random Forest method. The hyperparameters to tune are the following

- **T :** number of trees. Increasing the number of trees generally leads to a more robust model but also increases computational cost.
- **m :** number of variables checked at each split. It helps to introduce randomness into the model and prevent individual trees from becoming too specialized to the training data.
- **Size :** Depth of trees. Deeper trees can capture more complex relationships in the data but may lead to overfitting.
- **Minimum Split :** minimum number of samples required to split an internal node. A higher value prevents the tree from splitting nodes that have too few samples, helping to control model complexity and prevent overfitting.
- **Minimum Leaf :** minimum number of samples required to be at a leaf node (external). It prevents the tree from creating nodes with few samples, which can help to smooth the model and prevent overfitting.

The final model presented the following hypeparameters: **T =** 200, **m =** log_2 and **Size =** 10, **Minimum Split =** 5 and **Minimum Leaf =** 2. It was made of 200 trees, with the log_2 of number of variables checked at each split and trees' depth equal to 10, 5 samples minimum to split an internal node and 2 samples to split a leaf node which is a trivial assumption.

8.2.2. XGBoost

XGBoost, short for **Extreme Gradient Boosting**, is an efficient machine learning library renowned by its high-performance implementation of gradient boosting algorithms. It builds upon the principles of ensemble learning, specifically gradient boosting, by sequentially training weak learners (typically decision trees) in a stage-wise manner.
XGBoost's key innovation lies in its optimization of the gradient boosting framework through several enhancements, including a novel regularization technique called "regularized learning objective," which penalizes model complexity to prevent overfitting, and a sophisticated algorithm for tree building that enables parallelization and distributed computing.
These optimizations lead to superior predictive performance and computational efficiency, making XGBoost a popular choice across a wide range of applications, including classification, regression, ranking, and recommendation systems. Additionally, XGBoost provides a rich set of hyperparameters for fine-tuning model performance and flexibility in handling diverse datasets. Its robustness, scalability, and interpretability have solidified its position as one of the most powerful and widely used machine learning libraries in both academia and industry. As for Random Forest, also for this model tuning hyperparameters was a crucial step, and they are described as

- **T :** number of trees. Increasing the number of estimators can improve model performance but may also increase computational cost.
- **Size :** Depth of trees. Deeper trees can capture more complex relationships in the data but may lead to overfitting.

- **C :** fraction of columns to be random samples for each tree. It introduces randomness into the model and helps prevent overfitting by training each tree on a different subset of features.
- **Minimum Child:** minimum sum of weights of observations required in a child. Higher values prevent the model from partitioning leaf nodes that have fewer instances, helping to control model complexity and prevent overfitting.

Our final model for XGBoost had the hyperparameters set respectively to **T =** 100, **Size =** 3, **C =** 0.7 and **Minimum child =** 1, resulting in 100 trees, with a 3 depth size, 0.7 columns random samples for each tree and sum of weights of observation in a child to overcome the 1 threshold.

8.3. Recurrent Neural Networks Models

Our focus will now move to recurrent neural networks from their most simple formulation to more complex architectures. For both simple RNN and LSTM, input variables were the features selected by backward elimination on lagged features as described in part 3.5. The importance of those features will discussed in 10.3.

8.3.1. Simple RNN

A simple Recurrent Neural Network (**RNN**) is a type of neural network architecture designed to model sequential data by capturing dependencies between elements in the sequence.
Unlike traditional feed-forward neural networks, which process inputs independently of each other, RNNs have connections that form directed cycles, allowing them to maintain a state or memory of previous inputs as they process subsequent inputs. In a simple RNN, the network consists of a single recurrent layer where each neuron receives input not only from the current time step, but also from its output at the previous time step.
This recurrent connection enables the network to learn temporal patterns and dependencies in the data, making it well-suited for tasks such as time series prediction, language modeling, and sequential decision-making. Despite their effectiveness, simple RNNs often suffer from the vanishing gradient problem, where gradients diminish exponentially as they propagate back through time, limiting their ability to capture long-term dependencies.
It is remarkable also for simple RNNs to introduce their main hyperparameters to fine-tune which are

- **Unit :** the number of units (neurons) in the recurrent layer of the RNN. A higher number of units can increase the model's capacity to learn complex patterns in the data but also increases computational cost and the risk of overfitting.
- **Activation Function :** activation functions introduce non-linearity into the model, allowing it to learn complex relationships in the data.
- **Dropout :** it is a regularization technique used to prevent overfitting by randomly setting a fraction of input units to zero during each training iteration. It helps the model to generalize better to unseen data by reducing reliance on specific features.
- **Optimizer :** determines the algorithm used to update the weights of the network during training
- **Epochs :** epoch refers to one complete pass through the entire training dataset. This hyperparameter specifies the number of times the training algorithm will work through the entire dataset. Training for more epochs allows the model to learn more complex patterns but may also increase the risk of overfitting.
- **Batch Size :** batch size defines the number of samples processed before updating the model's parameters. Training with mini-batches instead of the entire dataset at once can help improve convergence and utilize parallel processing resources efficiently.
- η **:** learning rate, which controls the step size at which the model's parameters are updated during training. A higher learning rate can speed up convergence but may risk overshooting the optimal solution, while a lower learning rate may lead to slow convergence but more precise updates.

The hyperparameters chosen via cross-validation for simple RNN are displayed in **Table 7**.

Parameter	Value
Unit	20
Activation Function	ReLU
Dropout	0.1
Optimizer	ADAM
Epochs	100
Batch Size	32
η	0.001

Table 7. **Configuration for simple RNN**

8.3.2. LSTM

To address the vanishing gradient problem, limited capability to capture long-term dependencies, and improve the performance of simple RNNs on sequential data tasks, a more advanced architecture has been developed and its name is Long Short-Term Memory (LSTM).

Unlike traditional RNNs, which struggle to retain information over long sequences due to the fading of gradients during back-propagation, LSTM networks utilize a sophisticated memory cell that allows them to learn and remember information over extended periods.
This memory cell contains various components, including input, forget, and output gates, each controlled by sigmoid activation functions that regulate the flow of information. The input gate determines how much new information is stored in the cell state, the forget gate controls what information should be discarded from the cell state, and the output gate regulates the information that is passed to the next time step.
By dynamically updating and controlling the flow of information through these gates, LSTM networks can effectively learn complex temporal patterns and dependencies in the data, making them well-suited for a wide range of sequential tasks such as natural language processing, speech recognition, and time series forecasting.
Additionally, also for this final model, fine-tuning the hyperparameters is crucial. These hyperparameters are the same as those described for simple RNNs in 8.3.1, with one addition:

- **Number of Layers :** LSTM networks can have multiple layers stacked on top of each other. Each layer consists of one or more LSTM cells. Increasing the number of layers allows the model to capture more complex temporal dependencies but also increases computational complexity and the risk of overfitting.

The set of fine-tuned hyperparameters for the univariate model and the multivariate model are shown in **Table 8** and **Table 9** respectively.

PARAMETER	VALUE
UNIT	20
ACTIVATION FUNCTION	RELU
DROPOUT	0.1
OPTIMIZER	ADAM
EPOCHS	100
BATCH SIZE	32
η	0.01
NUMBER OF LAYERS	1

Table 8. **Configuration of LSTM for Univariate Modelling**

9. Results

In this section we are going to show the results obtained from all the cited models in our main task, forecasting daily realized volatility of S&P 500. These results will be the ones obtained with our out-of-sample analysis, made in the test set, i.e. 01/04/2023-29/12/2023. We are going to analyze first the prediction based on univariate time series models and after that, we will take a look at more complex forecasting based on multivariate time series. At the end of this section, we will have all the information needed to have a good discussion about the efficiency of the models, and we will be able to make an exhaustive comparison between all of them.

PARAMETER	VALUE
UNIT	20
ACTIVATION FUNCTION	RELU
DROPOUT	0.2
OPTIMIZER	ADAM
EPOCHS	200
BATCH SIZE	32
η	0.01
NUMBER OF LAYERS	1

Table 9. **Configuration of LSTM for Multivariate Modelling**

9.1. Univariate Modelling

As stated previously, the two models used for univariate forecasting were GARCH (1,2) with Standard Skewed Student-t distributed residuals with return as input, and LSTM with first lagged realized volatility as input.
We can have a look at their time series forecasting in **Figures 23 and 24** , where it is possible to spot qualitatively their trends.

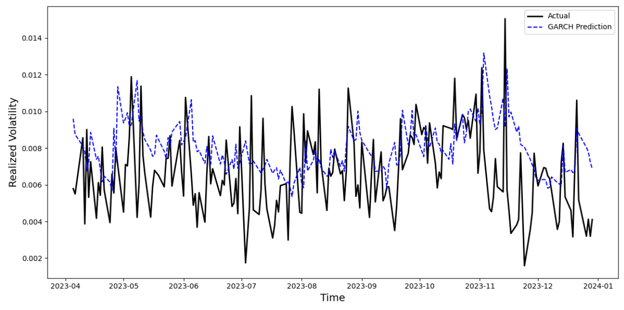


Figure 23. **GARCH Prediction Out-Of-Sample**

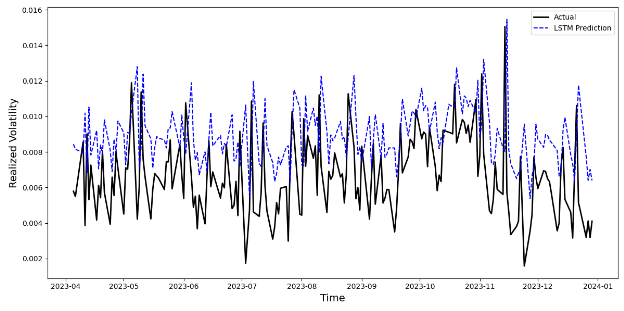


Figure 24. **LSTM Prediction Out-Of-Sample**

To have a better quantitative understating, all evaluation metrics out-of-sample are displayed in **Table 10**, which shows that LSTM with one input, performs worse than

GARCH model in the task.

MODEL	R^2	RMSE	MAE	MAPE
GARCH	-0.5415	0.002720	0.002194	0.4325
LSTM	-1.1814	0.003246	0.002671	0.5262

Table 10. **Univariate Models Performance OOS**

9.2. Multivariate Modelling

To assess the performance out-of-sample of multivariate time series models, it is useful to first show results for VAR model, and then give results for ML models grouped by their branch.

9.2.1. VAR

VAR (9), which was the econometrics model deputed to this task, will be a benchmark model to compare results of all the ML models. Its out-of-sample performance is resumed in **Table 11** and its time series visualization is displayed in **Figure 25**.

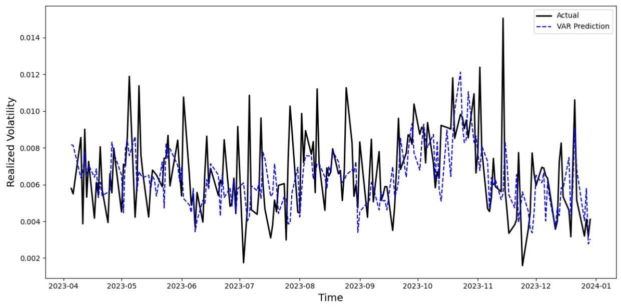


Figure 25. **VAR Prediction Out-Of-Sample**

MODEL	R^2	RMSE	MAE	MAPE
VAR	0.001582	0.002196	0.001649	0.2696

Table 11. **VAR Performance OOS**

9.2.2. REGRESSION MODELS

It is interesting for regression models to compare all of them together and to show visually how different their predictions look out-of-sample, as shown in **Figure 26**.

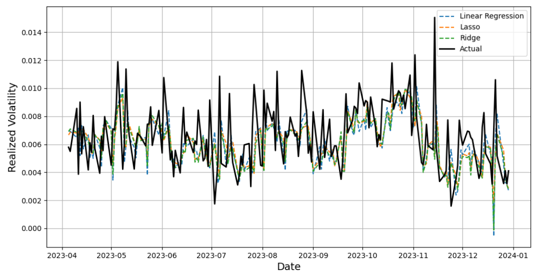


Figure 26. **Regression Models Prediction Out-Of-Sample**

Also, all the evaluations of those models are covered in **Table 12** to understand easily the improvement that penalization methods can achieve compared to linear regression quantitatively.

MODEL	R^2	RMSE	MAE	MAPE
LR	-0.1251	0.002332	0.001744	0.2825
RIDGE	-0.0944	0.002299	0.001642	0.2588
LASSO	-0.0431	0.002245	0.001615	0.2570

Table 12. **Regression Models Performance OOS**

9.2.3. DECISION TREE-BASED MODELS

For decision tree-based models, Random Forest and XGBoost, we decided as well to display time series prediction out-of-sample in a single plot to see qualitatively how they co-move by being similar methods as shown in **Figure 27**.

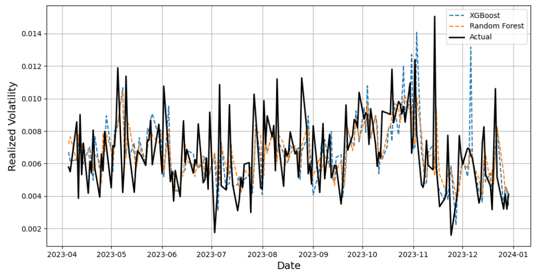


Figure 27. **Decision Tree-based Models Prediction Out-Of-Sample**

Goodness of fit achieved in prediction from those two models is represented in **Table 13**.

MODEL	R^2	RMSE	MAE	MAPE
RANDOM FOREST	0.0231	0.002172	0.001618	0.2732
XGBOOST	-0.0896	0.002294	0.001691	0.2768

Table 13. **Decision Tree-based Models Performance OOS**

9.2.4. RECURRENT NEURAL NETWORKS MODELS

The last models for which results will be reported are recurrent neural networks, a simple one, and LSTM which recorded the best results of all the pool of multivariate models. Their time series forecasting out-of-sample are presented in **Figure 28** and interesting quantitative counterpart is recorded in **Table 14**.

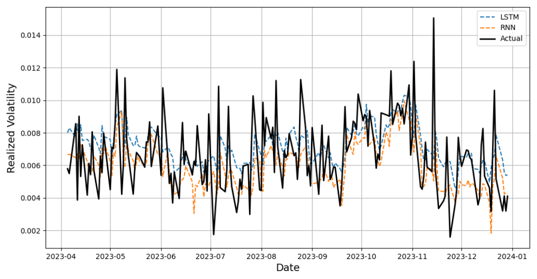


Figure 28. **RNN Models Prediction Out-Of-Sample**

MODEL	R^2	RMSE	MAE	MAPE
SIMPLE RNN	0.0271	0.002168	0.001565	0.2437
LSTM	0.1265	0.002054	0.001581	0.2937

Table 14. **RNN Models Models Perfomance OOS**

9.2.5. ALL TOGETHER

To conclude this section and provide an overall view of all the models both in univariate and multivariate settings and understand in a clear way performances in prediction out-of-samples, we believe it is useful to resume all the previous findings in **Table 15**, which will be used as a starting point for our discussion in the following section.

MODEL	R^2	RMSE	MAE	MAPE
UNIVARIATE				
GARCH	-0.5415	0.002720	0.002194	0.4325
LSTM	-1.1814	0.003246	0.002671	0.5262
MULTIVARIATE				
VAR	0.0016	0.002196	0.001649	0.2696
LR	-0.1251	0.002332	0.001744	0.2825
RIDGE	-0.0944	0.002299	0.001642	0.2588
LASSO	-0.0431	0.002245	0.001615	0.2570
RF	0.0231	0.002172	0.001618	0.2732
XGBOOST	-0.0896	0.002294	0.001691	0.2768
RNN	0.0271	0.002168	**0.001565**	**0.2437**
LSTM	**0.1265**	**0.002054**	0.001581	0.2937

Table 15. **Models Performance OOS**

10. Discussion

10.1. Evaluation metric

As expressed in 5.2, our main evaluation metric will be R^2. Even though R^2 is bounded in (0,1) when fitting a model on the training set, in out-of-sample forecasting, it can take negative values. In our models, it happened for the majority of them that the R^2 on the test set was negative, representing their predictions were worse than a constant function that predicts the out-of-sample average of the data.

10.2. Models Comparison

Starting with univariate models, we observe that both models perform very poorly, showing that neither the returns nor the lagged realized volatilities have enough information to have predictive power. However, in terms of all metrics, GARCH has performed way better than the univariate LSTM. One possible explanation could be that since LSTM models are much more complex (have way more parameters) than GARCH, they require more data to learn the underlying pattern. More data is either more explanatory variables (number of columns) or more observations (number of rows). Since the number of observations in our dataset is less than 2000, and we are using only the lagged values of realized volatility (only one column) for the LSTM model, there is a high risk of overfitting resulting in poor out-of-sample predictions.
Next, we move on to the multivariate models. First of all, we observe that all these models have better out-of-sample prediction than both univariate models. The main reason could be due to their ability to capture the interdependencies among multiple variables. Financial markets are influenced by a variety of factors, and considering multiple variables allows the model to incorporate a broader range of information, leading to more accurate and robust predictions.
Among multivariate models, LSTM performed way better than others, with RNN as the next best. LSTM models are designed for sequential data and can capture long-term dependencies. While RNNs can also handle sequential data, they are more prone to losing important information over time, making them less effective than LSTMs.
Looking at the decision-tree-based models (the Random Forest, and the XGBoost), we observe that on average they perform worse than the RNN-based models (LSTM and simple RNN). This is due to their inability to naturally capture and model temporal dependencies inherent in time-series data. These models treat each observation independently and do not have a mechanism to inherently capture the order or sequence of the data points. As a result, they fail to leverage the temporal relationships and dynamics that are key to understanding and predicting volatility in time-series data.
All the regression models have negative R^2. This is due to many reasons. First, although we have included the first two lags of the features and their second power, they still struggle to capture the complex, non-linear, and sequential dependencies present in the data. Since these models have a limited number of explanatory variables and lack com-

plexity and non-linearity, they have the risk of overfitting. This is why among these models, LASSO performs best due to its ability to perform feature selection by shrinking some coefficients to zero and preventing overfitting. Ridge regression, while also regularizing the model, does not perform feature selection but still mitigates overfitting better than plain linear regression. That is why it has a higher R^2 than linear regression. Nonetheless, all these models are outperformed by more sophisticated methods like RNNs and LSTMs, which are specifically designed to handle time-dependent data.

Finally, we see that the VAR model performs better than the regression models but again is not as strong as RNN or LSTM. VAR models are specifically designed for multivariate time series analysis and can effectively model the relationships and dynamics among several variables over time. This makes VAR superior to regression models. However, it is a linear model. Hence, similar to the reasons that we explained for regression models, it is not able to capture the non-linear complex patterns which make it perform worse than LSTM. Further, evaluating lagged dependent variables involved in VAR reveals that a lot of them show high p-values.

10.3. Feaure Importance Analysis

After running an LSTM model for volatility prediction, the **SHAP** values revealed the significance of various features in the model which are displayed in **Figure 29**. The VIX_{t-1} emerged as the most important predictor, which is expected because the VIX, known as the "fear gauge," measures market expectations of near-term volatility for S&P 500.

Both lags of realized volatility, $RV_{t-1}^{(d)}$, $RV_{t-2}^{(d)}$ are also highly significant showing that past realized volatility gives us information about future realized volatility.

Economic Policy Uncertainty lagged (i.e. EPU_{t-1}) is another critical feature, indicating that uncertainty regarding economic policies has a significant impact on market behavior and volatility, as investors react to potential changes in the economic environment.

The sentiment magnitude, $|SentimentScore_{t-1}|$ also has a significant effect on the model outputs. What is interesting is that for all data points, a higher sentiment magnitude results in higher volatility which makes sense because when sentiment is highly positive or negative, it can lead to increased trading activity and sharper price swings, contributing to higher volatility.

$Volume_{t-2}$ and $Term\ Spread_{t-2}$ were less important but still relevant, indicating that trading volume and interest rate spreads impact market stability. However, the returns from 13-Week Treasury Bills were not significant. To give a further sense of completeness, SHAP values were computed also for RNN and it is interesting to notice that almost all the features show the same magnitude of importance (see **Figure 30**).

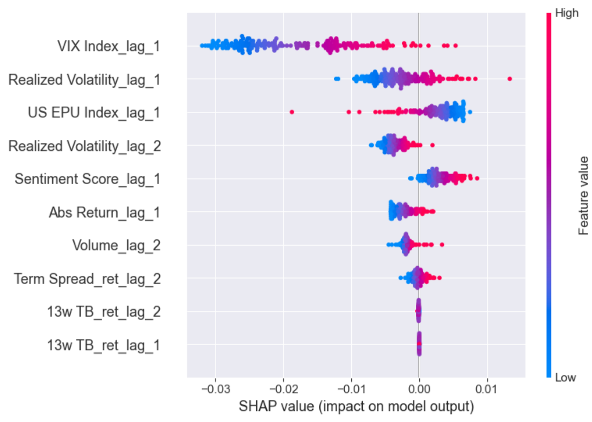


Figure 29. **SHAP values for LSTM**

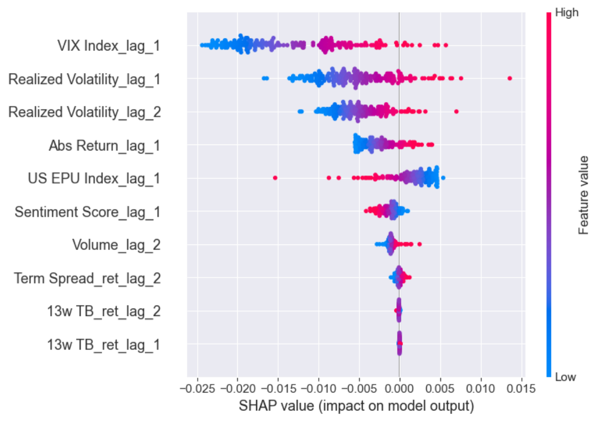


Figure 30. **SHAP values for Simple RNN Model**

11. Conclusion

In this project, we aimed to address two main research questions regarding the forecasting power of different models for predicting daily realized volatility and the impact of macrofinancial indicators and sentiment analysis on these predictions.

Regarding the first research question, our findings indicate that in the multivariate models, machine learning models, particularly the LSTM and RNN, show superior out-of-sample forecasting power compared to traditional econometric models like VAR. However, in the case of univariate models, although both approaches performed poorly, the GARCH model was better than the univariate LSTM due to its simplicity and suitability for the dataset size.

Regarding the second research question, we observe that all multivariate models outperformed the univariate approaches indicating that indeed adding the macrofinancial variables and the sentiment score improved the predictive power of the models. The feature importance analysis using SHAP values is another piece of evidence demonstrating this. In particular, we observe that The US Economic Policy Uncertainty (EPU) index, as a macrofinancial factor, and the sentiment scores had high importance in predicting volatility.

In conclusion, our study confirms that machine learning models, especially LSTM and RNN, outperform traditional econometric models in volatility prediction when using multivariate data. Additionally, incorporating macrofinancial indicators and sentiment analysis enhances the predictive power of these models, providing a more comprehensive understanding of market dynamics and improving forecasting accuracy.

References

[1] F. Black, “Noise,” *Journal of Finance*, vol. 41, pp. 529–543, 1976.

[2] T. Bollerslev, “Generalized autoregressive conditional heteroskedasticity,” *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986.

[3] T. G. Andersen and T. Bollerslev, “Heterogeneous information arrivals and return volatility dynamics: Uncovering the long-run in high frequency returns,” *Journal of Finance*, vol. 52, no. 3, pp. 975–1005, 1997.

[4] A. David and P. Veronesi, “What ties return volatilities to price valuations and fundamentals?,” *Journal of Political Economy*, vol. 121, no. 4, pp. 682 – 746, 2013.

[5] I. M. M. Ghani and H. A. Rahim, “Modeling and Forecasting of Volatility using ARMA-GARCH: Case Study on Malaysia Natural Rubber Prices,” *IOP Conference Series: Materials Science and Engineering*, vol. 548, p. 012023, jun 2019.

[6] A. Wilhelmsson, “GARCH forecasting performance under different distribution assumptions,” *Journal of Forecasting*, vol. 25, no. 8, pp. 561–578, 2006.

[7] T. Bollerslev, F. Diebold, T. Andersen, and P. Labys, “Modeling and forecasting realized volatility,” *Econometrica*, vol. 71, pp. 579–625, 03 2003.

[8] P. Carr, L. Wu, and Z. Zhang, “Using Machine Learning to Predict Realized Variance,” *ArXiv*, Sept. 2019.

[9] Y. Zhang, M. Wahab, and Y. Wang, “Forecasting crude oil market volatility using variable selection and common factor,” *International Journal of Forecasting*, vol. 39, no. 1, pp. 486–502, 2023.

[10] O. Çepni, R. Gupta, D. Pienaar, and C. Pierdzioch, “Forecasting the realized variance of oil-price returns using machine learning: Is there a role for U.S. state-level uncertainty?,” *Energy Economics*, vol. 114, p. 106229, 2022.

[11] D. Li, L. Zhang, and L. Li, “Forecasting stock volatility with economic policy uncertainty: A smooth transition GARCH-MIDAS model,” *International Review of Financial Analysis*, vol. 88, no. C, 2023.

[12] A. Prasad, P. Bakhshi, and A. Seetharaman, “The impact of the u.s. macroeconomic variables on the cboe vix index,” *Journal of Risk and Financial Management*, vol. 15, no. 3, 2022.

[13] Y. Liu, Z. Qin, P. Li, and T. Wan, “Stock volatility prediction using recurrent neural networks with sentiment analysis,” in *Advances in Artificial Intelligence: From Theory to Practice* (S. Benferhat, K. Tabia, and M. Ali, eds.), (Cham), pp. 192–201, Springer International Publishing, 2017.

[14] C. Zhang, Y. Zhang, M. Cucuringu, and Z. Qian, “Volatility forecasting with machine learning and intraday commonality,” *Journal of Financial Econometrics*, vol. 22, pp. 492–530, May 1 2024.

[15] X. Lu, F. Ma, J. Xu, and Z. Zhang, “Oil futures volatility predictability: New evidence based on machine learning models,” February 10 2022.

[16] Y. Liu, “Novel volatility forecasting using deep learning–Long Short Term Memory Recurrent Neural Networks,” *Expert Systems with Applications*, vol. 132, pp. 99–109, 2019.

[17] M. Sardelich and S. Manandhar, “Multimodal deep learning for short-term stock volatility prediction,” *ArXiv*, vol. abs/1812.10479, 2018.

[18] J. Osterrieder, D. Kucharczyk, S. Rudolf, and D. Wittwer, “Neural networks and arbitrage in the vix,” *Digital Finance*, vol. 2, pp. 97–115, August 2020.

[19] D. Filipovic and A. Khalilzadeh, “Machine learning for predicting stock return volatility,” *Swiss Finance Institute Research Paper*, December 23 2021.

[20] K. Christensen, M. Siggaard, and B. Veliyev, “A machine learning approach to volatility forecasting,” January 13 2021.

[21] O. Barndorff-Nielsen and N. Shephard, “Estimating quadratic variation using realized variance,” *Journal of Applied Econometrics*, vol. 17, no. 5, pp. 457–477, 2002.

[22] O. E. Barndorff-Nielsen and N. Shephard, “Econometric analysis of realized volatility and its use in estimating stochastic volatility models,” *Journal of the Royal Statistical Society Series B*, vol. 64, no. 2, pp. 253–280, 2002.

[23] K. Du, F. Xing, R. Mao, and E. Cambria, “Financial sentiment analysis: Techniques and applications,” *ACM Computing Surveys*, 2024.

[24] C. Hutto and E. Gilbert, “Vader: A parsimonious rule-based model for sentiment analysis of social media text,” in *Proceedings of the international AAAI conference on web and social media*, 2014.

[25] E. M. Gourier, “Supervised learning, lecture notes,” *Machine Learning in Finance*, 2024.