Columbia University

STATGU5261

Volatility Forecasting: A Comparative Study of GARCH, XGBoost, and LSTM Models

am6490 Arsh Misra,
xs2567 Xiang Si,
mw3821 Muqing Wen,
tz2626 Tiancheng Zhou,
xh2690 Xinyao Han

Final Report
May 13th, 2025

1. Introduction

Volatility is a key measure that has served as an essential tool for strategies in arbitrage, derivative pricing, portfolio management, and hedging. As a result, financial institutions and funds have experimented with ways to accurately forecast volatility in order to effectively take advantage of market cycles or foresee periods of financial turmoil. Some of the tools used include GARCH models (Bollerslev, 1986), which have served as the conventional option for forecasting. However, in recent years, deep learning models have entered the limelight. Deep learning algorithms have increasingly been applied to enhance the accuracy and reliability of stock market volatility forecasting (Zhang, 2023), prompting much discussion about their performance in comparison to traditional models. This study seeks to evaluate the comparative performance of one traditional model, GARCH, and two deep learning models, XGBoost (Chen & Guestrin, 2006) and LSTM (Hochreiter & Schmidhuber, 1997), in predicting S&P 500 volatility. The time-horizon chosen spans Jan 2018 to Dec 2024, including several black-swan events such as the COVID-19 pandemic. This was done to provide the models with a dynamic forecasting environment with a range of different regimes.

2. Research Objectives and Data Preprocessing

This study aims to examine the following research objectives:

- Construct three robust combinations of predictive models to forecast volatility for the S&P 500 Index
- Evaluate model performance using three key metrics: R-squared ($\boldsymbol{R^2}$), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE)
- Compare the respective model performances to determine the most viable model

To model volatility, we collected historical daily price and volume data for the S&P 500 from January 4, 2018, to December 30, 2024. The data was obtained from the Tushare database by using the API. The initial data was on daily price and volume. As volatility is linked to return fluctuations, we used returns from the data pre-calculated as follows:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}} \tag{1}$$

Additionally, derived variables such as percentage change and swing were calculated to capture short-term price movements. The volatility values were obtained using daily returns and a rolling 22-day volatility measure, formulated as:

$$vol_t = \sqrt{\frac{1}{N-1} \sum_{i=t-N+1}^{t} (r_t - \bar{r})^2} \tag{2}$$

3. Models

3.1 GARCH

$$\sigma_t^2 = \omega + \alpha \varepsilon_t^2 + \beta \sigma_{t-1}^2 \tag{3}$$

The distribution of SPX daily returns resembles a Generalized Error Distribution (GED) (see

Appendix 1), deviating from normality with leptokurtosis and mild negative skewness. It features a sharp central peak, indicating frequent small fluctuations, and fat tails, suggesting a higher probability of extreme returns, particularly beyond ±10%.

Traditional linear models, like ARIMA, often struggle to capture these characteristics due to their assumption of homoscedasticity. In contrast, GARCH models effectively handle time-varying volatility by modeling conditional variance based on past squared returns and variances. This makes GARCH well-suited for forecasting volatility in financial time series, accommodating both non-normality and volatility clustering inherent in SPX daily returns.

We selected GARCH(1,1) as the final model after carefully balancing model complexity and predictive accuracy. To determine the optimal lag orders, we adopted a two-step approach: first, we analyzed the Partial Autocorrelation Function (PACF) (Appendix 2.1) of squared returns to identify volatility clustering and short-memory effects, and second, we minimized the Akaike Information Criterion (AIC) (Appendix 2.2) to evaluate model fit and complexity.

The Partial Autocorrelation Function (PACF) of the squared returns showed significant spikes at low lags (particularly at lags 1 and 2), indicating short-memory effects and favoring lower-order GARCH terms. Additionally, a grid search over GARCH(p, q) models ($1 \leq p, q \leq 5$) using the Akaike Information Criterion (AIC) confirmed that GARCH(1,1) had the lowest AIC value, reflecting the best trade-off between fit and simplicity. While higher-order models like GARCH(4,4) offered slightly better dynamic capture, residual diagnostics indicated only marginal improvements, reinforcing the choice of GARCH(1,1) as the more efficient model. The model summary results are shown in Appendix 2.3 & 2.4.

To adapt to changing market conditions, we implemented a rolling forecasting approach. In this method, the GARCH model is repeatedly re-estimated at each step using the most recent data within an expanding window. This dynamic updating process allows the model to better capture rapid regime shifts, thereby improving predictive accuracy in non-stationary environments.

3.2 XGBoost

XGBoost is a tree-based ensemble learning algorithm optimized for speed and performance. It models the nonlinear relationship between engineered time features and future volatility. Our XGBoost model adopts several key hyperparameters (Appendix 3.1) to balance model complexity and generalization performance. The most important are:

- learning_rate = 0.3 (default value): Controls the contribution of each tree to the final prediction. A higher rate speeds up learning but increases the risk of overfitting.
- max_depth = 6 (default value): Limits tree complexity. Deeper trees model more interactions but may overfit; 6 offers a practical trade-off.

We constructed features that capture both autoregressive structure and seasonal calendar patterns. These lag features allow the model to learn temporal dependence from recent fluctuations:

- Lagged Returns and Volatility: These features capture short-term memory effects in volatility by incorporating recent fluctuations. Volatility tends to cluster, and including prior-day volatilities helps the model recognize this temporal dependence.

- Calendar Features: Financial volatility often follows seasonal patterns, such as higher uncertainty on Mondays or month-end. These features allow the model to account for systematic, calendar-driven influences on volatility.

As a result of the Feature Importance Analysis shown (Appendix 3.2), “pre_1”, “dayofyear”, and “dayofweek” are the top 3 important features. This indicates that the model heavily relies on recent market conditions, as captured by lagged volatility (pre_1), reflecting the well-known phenomenon of volatility clustering. Meanwhile, the strong influence of “dayofyear” and “dayofweek” suggests that XGBoost has successfully identified calendar-based seasonal patterns, such as heightened volatility at the start of the week or around specific times of the year, demonstrating its capacity to incorporate both short-term memory and cyclical market behaviors.

3.3 LSTM

LSTM stands for long-short term memory, and the model belongs to the recurrent neural network family. It differs from a traditional RNN in that it is more suitable for learning long-term patterns, something that a RNN often fails to do effectively due to the vanishing gradient problem. It does this by deciding what memory or pattern is useful to keep and what is not needed and can be discarded. LSTMs mainly use three types of gates to do this, a forget gate, an input gate, and an output gate. This makes it a powerful tool for predicting patterns in time-series volatility problems.

LSTMs have several components that can be tweaked depending on the requirements of the research as well as the type of data being analyzed. For this project, we focused on components such as layer and neuron architecture, dropout rate, and different feature combinations. The best performing model utilizes the following:

- Features: 4 previous days of true lagged volatility
- 1 LSTM layer and 1 dense output layer with 64 neurons in the LSTM layer
- A dropout rate of 0.2

In this model, we used rolling true 22-day volatility with a 10-day lookback. What’s important to emphasize here is we used a walk-forward approach to prevent data leaks, splitting the training and test sets before using the scaler. If we used the scalar after splitting, it would introduce training data into the test set, which would produce incorrect results.

Additionally, the 10-day lookback also had to be carefully constructed to prevent the LSTM from absorbing patterns from previous lookback periods, which would also cause a data leak. Essentially, the LSTM uses the 10-day lookback period to predict the 11th day, but the memory of the predicted value of the 11th day does not carry over into the prediction for the 12th day, and so on. This is a key concept because it does not introduce a circular dependency into the training and testing of the LSTM, which would produce invalid and inaccurate results.

4. Results & Evaluation

After training and testing the models and calculating the relevant evaluation metrics, we have the following table:

Table 1. Model Training/Fitting Scores

Train / %	**GARCH**	**XGBoost**	**LSTM**
$\boldsymbol{R^2}$	0.7654	0.9925	0.9820
RMSE	0.3848	0.0689	0.0270
MAPE	0.2212	0.4442	0.0569

Table 2. Prediction Scores

Prediction / %	**GARCH**	**XGBoost**	**LSTM**
$\boldsymbol{R^2}$	0.3898	0.9484	0.9619
RMSE	0.1644	0.0633	0.0127
MAPE	0.1706	0.0463	0.0660

The GARCH model performed worse than expected, with a fitting $\boldsymbol{R^2}$ of 76.54% and a prediction $\boldsymbol{R^2}$ 38.98%. This may be because the model struggles to differentiate between one-off shocks versus more sustained volatility periods, hence the lower training score. The test score being low is simply a consequence of this inadequacy. The out-of-sample performance further highlights this, as the GARCH model showed limited alignment with actual volatility, especially during regime shifts. Its forecasts are based on static, one-step-ahead dynamics that don't adjust well to rapid changes, resulting in a mismatch (see section 4.1 in the Appendix).

The best performing models turn out to be of the deep learning variety, with the LSTM having training and test scores of 98.20% and 96.19% respectively, and the XGBoost model trailing closely behind with scores of 99.25% and 94.84%. These are quite strong, and considering that the data likely has a high degree of autocorrelation, it is no surprise that the LSTM was likely able to catch onto these patterns. The MAPE helps corroborate this as the predicted versus actual values differ only by 6.6% on average. The plot in section 4.3 of the Appendix also aligns with the scores calculated.

The XGBoost model also demonstrates strong training performance, closely tracking the real volatility throughout the training period (Appendix 4.2). It accurately replicates both baseline and spiking volatility regimes, including extreme market shocks, reflecting the model's effectiveness in capturing short-term dependencies and seasonality patterns. Although slight discrepancies emerge around inflection points, which may have prevented it from performing better than the LSTM, the predicted path remains stable and responsive. This indicates solid short-term generalization, especially in low-volatility environments where structural patterns persist.

5. Conclusion

Overall, the report reveals that modern sequence and ensemble methods can substantially enhance predictive accuracy under varied market conditions. By integrating temporal memory through LSTM and capturing nonlinear interactions via XGBoost, we achieved more resilient forecasts than a traditional GARCH, particularly during abrupt regime shifts. This performance gap underscores the importance of flexible model architectures that adapt to both persistent autocorrelation and extreme events. The results found in this study also support previous studies that utilize comparisons between traditional models, such as Campisi et. al's comparisons of tree models with classical linear regression models, finding that the machine learning models significantly outperform classical architectures when it comes to volatility prediction (Campisi et. al 2023).

However, there are several things to note. Our results hinge on a specific sample period (Jan 2018–Dec 2024) and may not generalize to other crisis regimes or longer horizons, while the high accuracy of XGBoost and LSTM models raises concerns about overfitting despite walk-forward validation. Moreover, deep learning approaches demand extensive hyperparameter tuning and greater computational resources and limiting inputs to lagged volatility and calendar features may overlook critical exogenous drivers such as market sentiment or liquidity. Future work should address these limitations by testing hybrid GARCH–neural architectures, expanding to intraday and alternative asset classes, and integrating exogenous drivers to enhance both interpretability and robustness.

Our results prompt several avenues for future study: Can hybrid GARCH–neural frameworks (e.g., those used by Dessie et. al) blend economic interpretability with deep learning's adaptability? How would these models scale to intraday data or alternative asset classes such as commodities and cryptocurrencies? Additionally, incorporating exogenous signals like market sentiment or liquidity measures may further refine volatility estimates. Pursuing these questions will help bridge the gap between theoretical advances and robust, real-world volatility forecasting.

References

Baillie, R. T., Bollerslev, T., & Mikkelsen, H. O. (1996). *Fractionally integrated generalized autoregressive conditional heteroskedasticity*. Journal of Econometrics*, 74*(1), 3–30. https://doi.org/10.1016/0304-4076(95)01749-6

Campisi, G., Muzzioli, S., and De Baets, B., *A comparison of machine learning methods for predicting the direction of the US stock market on the basis of volatility indices,* International Journal of Forecasting, 40,

Chen, T., & Guestrin, C. (2016). *XGBoost: A scalable tree boosting system*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). Association for Computing Machinery. https://doi.org/10.1145/2939672.2939785

Dessie, E., Birhane, T., Mohammed, A., & Walelign, A. (2025). *A hybrid GARCH, convolutional neural network and long short-term memory methods for volatility prediction in the stock market*. Combinatorial Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, *30*, 6000–6010.

Zhang, J. (2023). *Some application of machine learning in quantitative finance: Model calibration, volatility formation mechanism and news screening* (Doctoral dissertation). Institut Polytechnique de Paris.

Zhao, Q., Fan, Z., He, R., Yang, L., & Huang, Y. (2018). *Prediction of plant-derived xenomiRs from plant miRNA sequences using random forest and one-dimensional convolutional neural network models*. BMC Genomics*, 19*, Article 839. https://doi.org/10.1186/s12864-018-5227-3

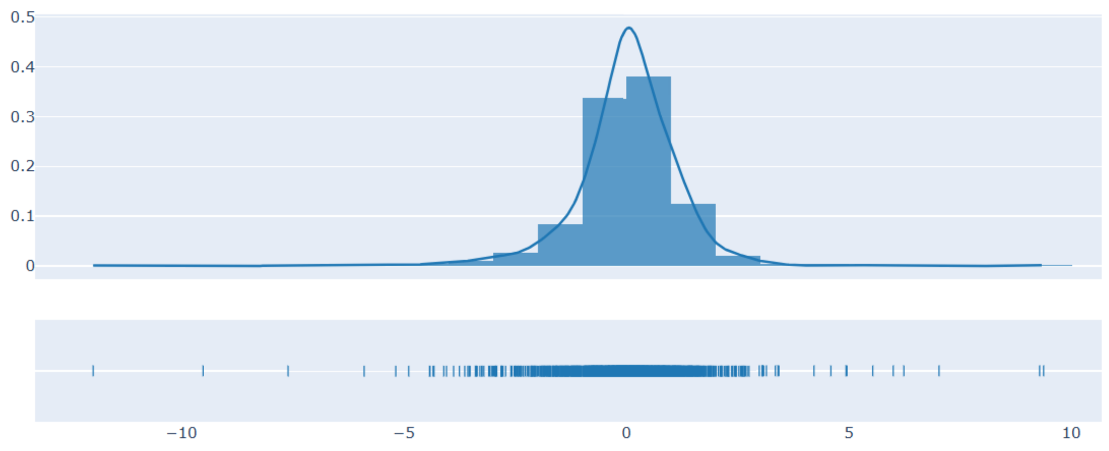
Appendix

0. Data Source Information:

The data used in this analysis was obtained from the following source:

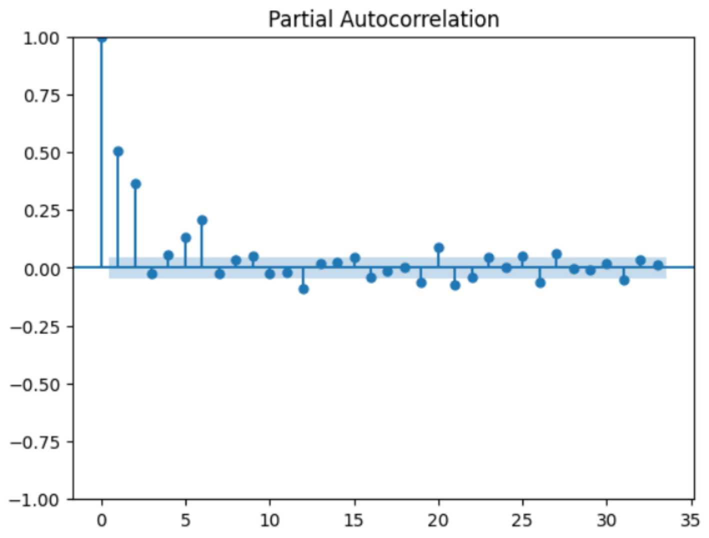
- Website: https://tushare.pro/document/2?doc_id=211
- Provider: Tushare
- Description: Financial data and documentation provided for research and analysis purposes.

1. Distribution of Daily SPX Returns

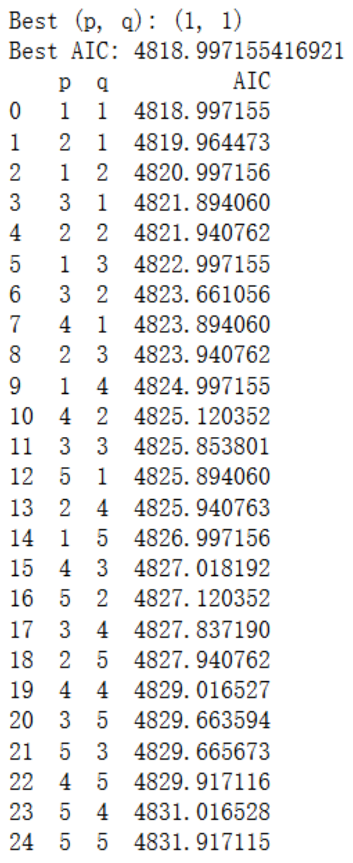


2. GARCH

2.1 Partial Autocorrelation Analysis (PACF)

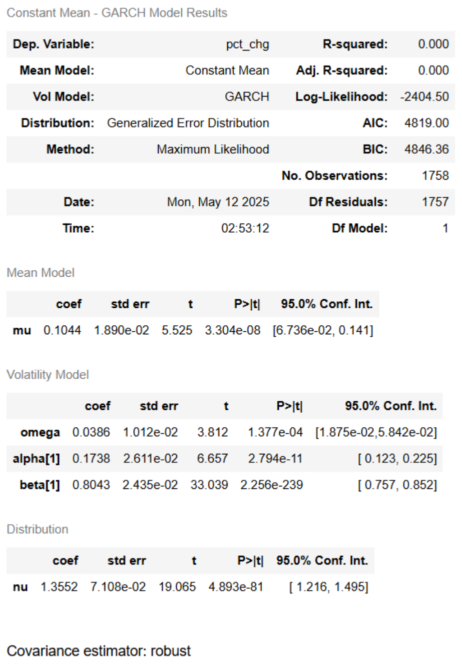


2.2. p&q Selection Analysis with AIC Minimizing



```
Best (p, q): (1, 1)
Best AIC: 4818.997155416921
    p  q          AIC
0   1  1  4818.997155
1   2  1  4819.964473
2   1  2  4820.997156
3   3  1  4821.894060
4   2  2  4821.940762
5   1  3  4822.997155
6   3  2  4823.661056
7   4  1  4823.894060
8   2  3  4823.940762
9   1  4  4824.997155
10  4  2  4825.120352
11  3  3  4825.853801
12  5  1  4825.894060
13  2  4  4825.940763
14  1  5  4826.997156
15  4  3  4827.018192
16  5  2  4827.120352
17  3  4  4827.837190
18  2  5  4827.940762
19  4  4  4829.016527
20  3  5  4829.663594
21  5  3  4829.665673
22  4  5  4829.917116
23  5  4  4831.016528
24  5  5  4831.917115
```

2.3 Outcome of GARCH(1,1)



Constant Mean - GARCH Model Results

Dep. Variable:	pct_chg	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2404.50
Distribution:	Generalized Error Distribution	**AIC:**	4819.00
Method:	Maximum Likelihood	**BIC:**	4846.36
		No. Observations:	1758
Date:	Mon, May 12 2025	**Df Residuals:**	1757
Time:	02:53:12	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.1044	1.890e-02	5.525	3.304e-08	[6.736e-02, 0.141]

Volatility Model

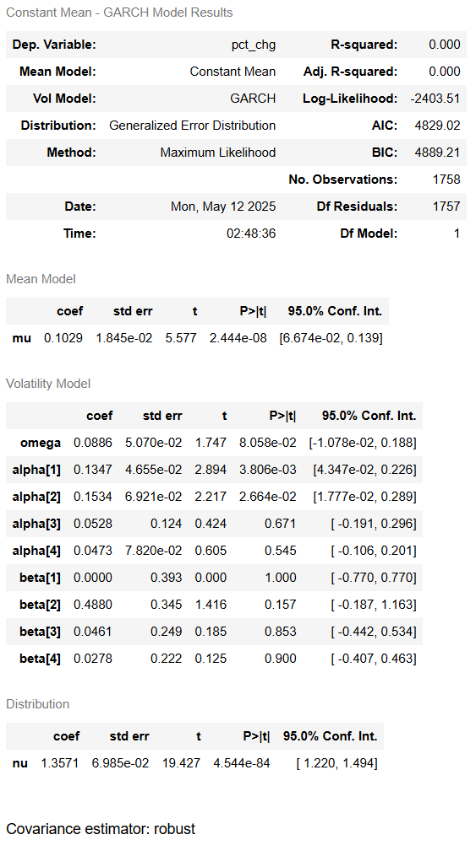
	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0386	1.012e-02	3.812	1.377e-04	[1.875e-02,5.842e-02]
alpha[1]	0.1738	2.611e-02	6.657	2.794e-11	[0.123, 0.225]
beta[1]	0.8043	2.435e-02	33.039	2.256e-239	[0.757, 0.852]

Distribution

	coef	std err	t	P>|t|	95.0% Conf. Int.
nu	1.3552	7.108e-02	19.065	4.893e-81	[1.216, 1.495]

Covariance estimator: robust

2.4 Outcome of GARCH(4,4)



Constant Mean - GARCH Model Results

Dep. Variable:	pct_chg	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2403.51
Distribution:	Generalized Error Distribution	**AIC:**	4829.02
Method:	Maximum Likelihood	**BIC:**	4889.21
		No. Observations:	1758
Date:	Mon, May 12 2025	**Df Residuals:**	1757
Time:	02:48:36	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.1029	1.845e-02	5.577	2.444e-08	[6.674e-02, 0.139]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0886	5.070e-02	1.747	8.058e-02	[-1.078e-02, 0.188]
alpha[1]	0.1347	4.655e-02	2.894	3.806e-03	[4.347e-02, 0.226]
alpha[2]	0.1534	6.921e-02	2.217	2.664e-02	[1.777e-02, 0.289]
alpha[3]	0.0528	0.124	0.424	0.671	[-0.191, 0.296]
alpha[4]	0.0473	7.820e-02	0.605	0.545	[-0.106, 0.201]
beta[1]	0.0000	0.393	0.000	1.000	[-0.770, 0.770]
beta[2]	0.4880	0.345	1.416	0.157	[-0.187, 1.163]
beta[3]	0.0461	0.249	0.185	0.853	[-0.442, 0.534]
beta[4]	0.0278	0.222	0.125	0.900	[-0.407, 0.463]

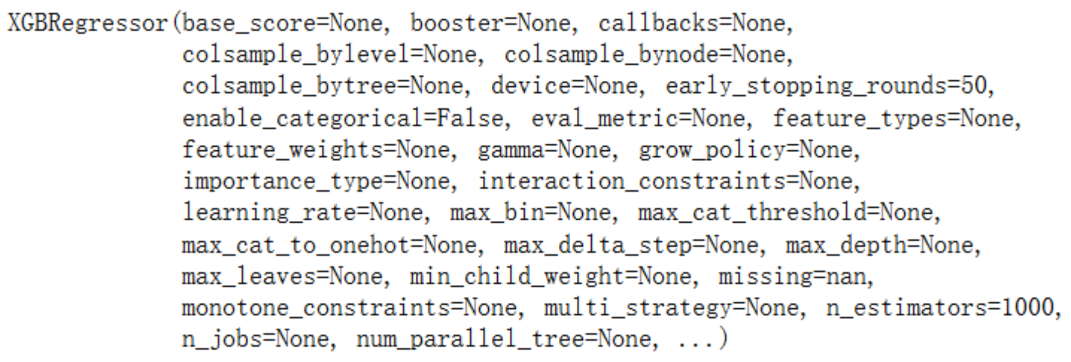
Distribution

	coef	std err	t	P>|t|	95.0% Conf. Int.
nu	1.3571	6.985e-02	19.427	4.544e-84	[1.220, 1.494]

Covariance estimator: robust

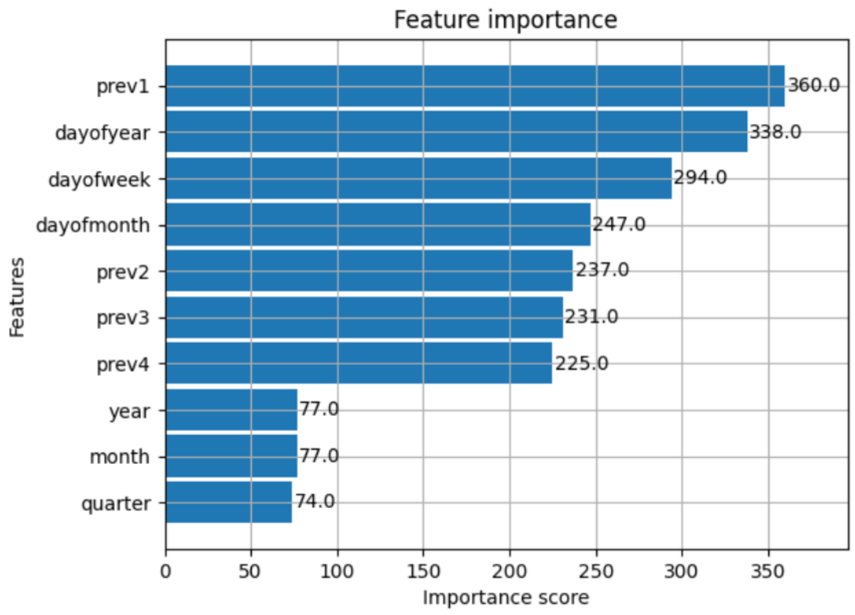
3. XGBoost

3.1 Hyperparameter Settings



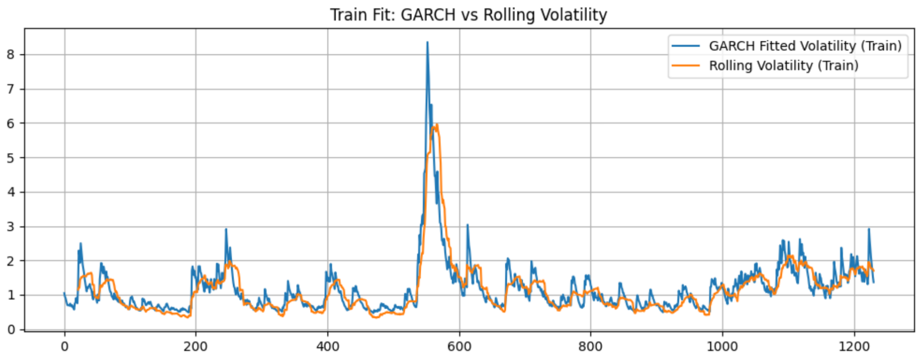
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=1000,
             n_jobs=None, num_parallel_tree=None, ...)
```

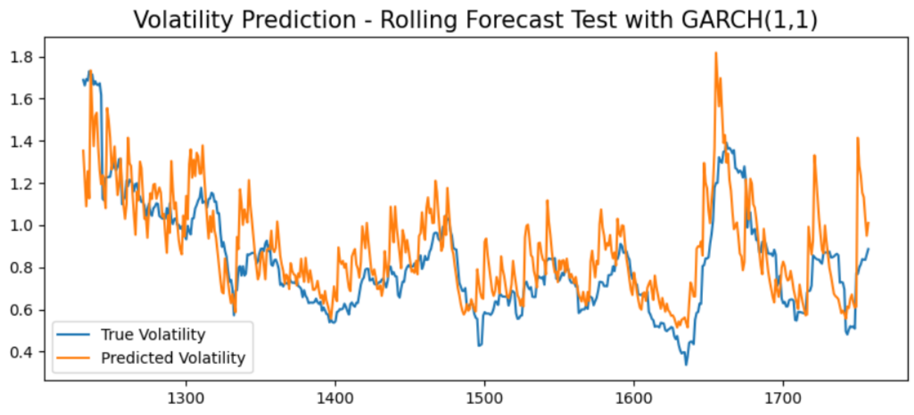
3.2 Scoring of XGBoost Features



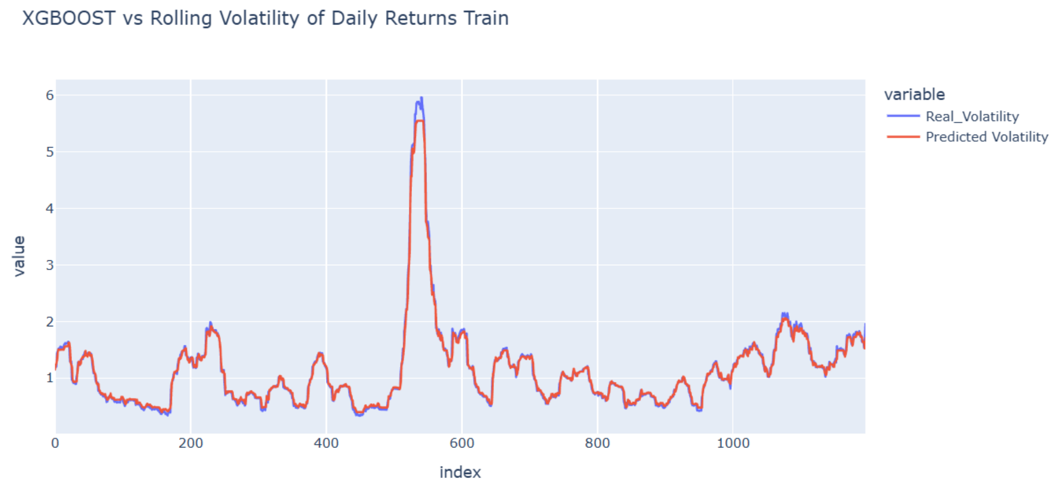
4. Models, Train & Prediction Results

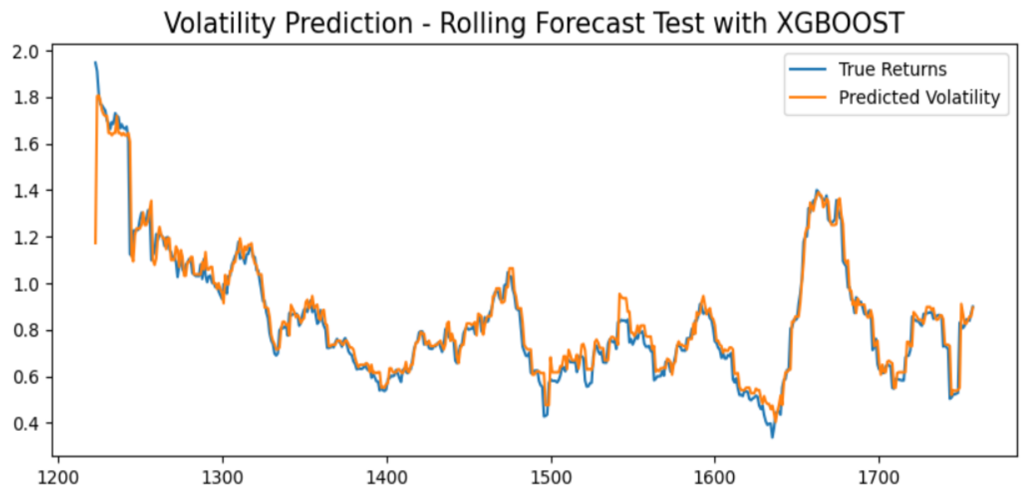
4.1 GARCH

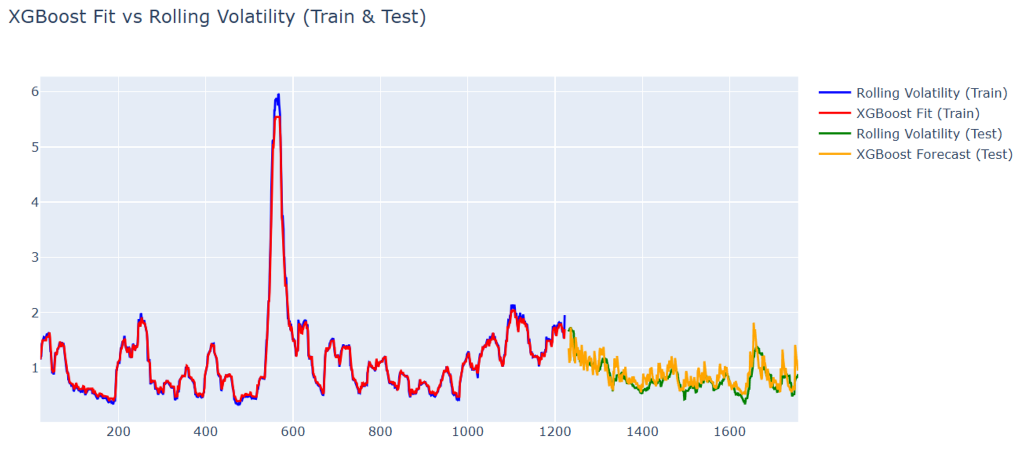




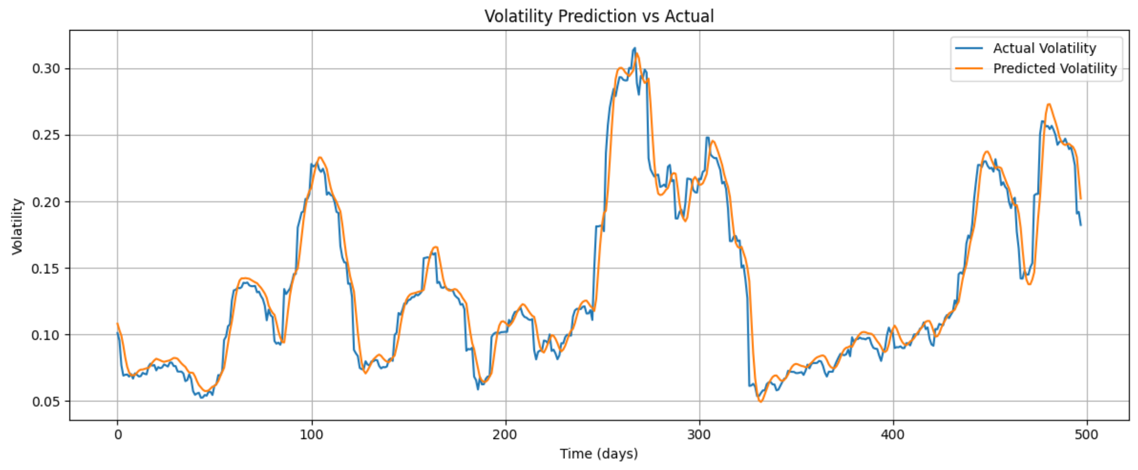
4.2 XGBoost


XGBOOST vs Rolling Volatility of Daily Returns Train
variable
Real_Volatility
Predicted Volatility
value
index


Volatility Prediction - Rolling Forecast Test with XGBOOST
True Returns
Predicted Volatility


XGBoost Fit vs Rolling Volatility (Train & Test)
Rolling Volatility (Train)
XGBoost Fit (Train)
Rolling Volatility (Test)
XGBoost Forecast (Test)

4.3 LSTM



5. Model Training/Fitting Scores

Train / %	GARCH	XGBoost	LSTM
R^2	0.7654	0.9925	0.9820
RMSE	0.3848	0.0689	0.0270
MAPE	0.2212	0.4442	0.0569

6. Prediction Scores

Prediction / %	GARCH	XGBoost	LSTM
R^2	0.3898	0.9484	0.9619
RMSE	0.1644	0.0633	0.0127
MAPE	0.1706	0.0463	0.0660

7. Code

GARCH:

```
import math

import numpy as np

import pandas as pd

import plotly.express as px

import plotly.figure_factory as ff

import matplotlib.pyplot as plt

import seaborn as sns

from tabulate import tabulate

from arch import arch_model

from statsmodels.graphics.tsaplots import plot_pacf

from      sklearn.metrics      import      mean_absolute_percentage_error,
mean_squared_error

from ipywidgets import HBox, VBox



df_spx     =     pro.index_global(ts_code='SPX',     start_date='20180104',
end_date='20241230')

df_spx = df_spx.sort_values(by='trade_date').reset_index(drop=True)

returns_spx = pd.DataFrame(df_spx['pct_chg'])

returns_spx.columns = ['pct_chg']

#Volatility Estimation

daily_vol = returns_spx['pct_chg'].std()

monthly_vol = np.sqrt(21) * daily_vol

annual_vol = np.sqrt(252) * daily_vol
```

```
print(tabulate(

    [['SPX', daily_vol, monthly_vol, annual_vol]],

    headers=['Index', 'Daily Volatility %', 'Monthly Volatility %', 'Annual
Volatility %'],

    tablefmt='github', stralign='center', numalign='center', floatfmt=".2f"

))

#Analyst &Visualizations

returns_spx.plot(figsize=(16, 5), title='SPX Daily Returns')



dist_plot       =       ff.create_distplot([returns_spx['pct_chg'].values],
group_labels=[' '])

dist_plot.update_layout(showlegend=False, title_text='Distribution of Daily
SPX Returns', width=1000, height=500)

dist_plot.show()



plot_pacf(returns_spx['pct_chg'] ** 2, method="yw")

plt.show()



# GARCH(p,q) Model Order Selection

max_p, max_q = 5, 5

best_aic = float('inf')

best_order = None

results = []



for p in range(1, max_p + 1):
    for q in range(1, max_q + 1):

        try:

           model = arch_model(returns_spx, vol='GARCH', p=p, q=q, dist='ged')

            result = model.fit(disp='off')

            results.append((p, q, result.aic))

            if result.aic < best_aic:

                best_aic = result.aic

                best_order = (p, q)

        except Exception as e:

            print(f"Failed for (p={p}, q={q}): {e}")


# Fit Best Model on Full Data & model summary

best_p, best_q = best_order

final_model  =  arch_model(returns_spx,  vol='GARCH',  p=best_p,  q=best_q,
dist='ged')

final_result = final_model.fit(disp='off')

print(final_result.summary())

print("Best (p, q):", best_order)

print("Best AIC:", best_aic)

print(results)

# Split into train/test

total_len = len(returns_spx)

test_size = int(0.3 * total_len)

train_size = total_len - test_size
```

```
train_returns = returns_spx['pct_chg'].iloc[:train_size]

test_returns = returns_spx['pct_chg'].iloc[train_size:]


print(total_len)

print(test_size)

print(train_size)


cond_vol_train = fitted_model.conditional_volatility

rolling_vol_train = train_returns.rolling(22).std().abs()


plt.figure(figsize=(10, 4))

plt.plot(cond_vol_train, label='GARCH Fitted Volatility (Train)')

plt.plot(rolling_vol_train, label='Rolling Volatility (Train)')

plt.title('Train Fit: GARCH vs Rolling Volatility')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()


#Rolling forecast on test set (expanding window)

rolling_preds = []

for i in range(test_size):

    expanding_window = returns_spx['pct_chg'].iloc[:train_size + i]
    model = arch_model(expanding_window, vol='GARCH', p=1, q=1, dist='ged')

    fitted = model.fit(disp='off')

    forecast = fitted.forecast(horizon=1)

    pred_vol = np.sqrt(forecast.variance.values[-1, 0])

    rolling_preds.append(pred_vol)


rolling_preds    =    pd.Series(rolling_preds,    index=returns_spx.index[-
test_size:])


#Compare test forecast

rolling_vol_test = test_returns.rolling(22).std().abs()


plt.figure(figsize=(10, 4))

plt.plot(rolling_vol_test, label='True Volatility (Test)')

plt.plot(rolling_preds, label='GARCH Predicted Volatility (Test)')

plt.title('GARCH(1,1) Rolling Forecast: Test Set')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```

XGBoost:

```
returns_spx_ml = df_spx[['trade_date', 'pct_chg']]

print(returns_spx_ml.head())
print(returns_spx_ml.shape)

returns_spx_ml                                                                    =
returns_spx_ml.drop(returns_spx_ml[returns_spx_ml["pct_chg"] == 0].index)

returns_spx_ml = returns_spx_ml.dropna()



#rolling volatility of 22 days as benchmark

returns_spx_ml["pct_chg"]
=  abs(returns_spx_ml["pct_chg"].rolling(window=22, min_periods=22).std())

returns_spx_ml = returns_spx_ml.dropna()

returns_spx_ml["pct_chg"].isna().sum()



returns_spx_ml["trade_date"] = pd.to_datetime(returns_spx_ml["trade_date"])

serie_for_xgboost = returns_spx_ml

serie_for_xgboost



test_size = 527

#Split train and test

train_ml = serie_for_xgboost[:-(test_size)].dropna()

test_ml = serie_for_xgboost[-(test_size):].dropna()

print(train_ml.shape)

print(test_ml.shape)



#UDF for extracting features from date

def create_features(df, label=None):
    """
    Creates time series features from datetime index
    """
    df['dayofweek'] = df['trade_date'].dt.dayofweek
    df['quarter'] = df['trade_date'].dt.quarter
    df['month'] = df['trade_date'].dt.month
    df['year'] = df['trade_date'].dt.year
    df['dayofyear'] = df['trade_date'].dt.dayofyear
    df['dayofmonth'] = df['trade_date'].dt.day



    X = df[['dayofweek','quarter','month','year',
           'dayofyear','dayofmonth']]
    if label:
        y = df[label]
        return X, y
    return X


#Creating the features for the train and test sets
X_train, y_train = create_features(train_ml, label="pct_chg")
X_test, y_test = create_features(test_ml, label="pct_chg")


#Creating an aditional feature that uses the 4 previous days of rolling
volatility, incorporating the autoregressive component to our ML model
```

```
X_train['prev1']=train_ml['pct_chg'].shift(1)

X_test['prev1']=test_ml['pct_chg'].shift(1)

X_train['prev2'] =train_ml['pct_chg'].shift(2)

X_test['prev2']=test_ml['pct_chg'].shift(2)

X_train['prev3'] =train_ml['pct_chg'].shift(3)

X_test['prev3']=test_ml['pct_chg'].shift(3)

X_train['prev4'] =train_ml['pct_chg'].shift(4)

X_test['prev4']=test_ml['pct_chg'].shift(4)



import xgboost as xgb

import plotly.graph_objects as go



# === Train XGBoost Regressor ===

xgb_model = xgb.XGBRegressor(n_estimators=1000, early_stopping_rounds=50)

xgb_model.fit(

    X_train, y_train,

    eval_set=[(X_train, y_train), (X_test, y_test)],

    verbose=False

)

print(repr(xgb_model))



from xgboost import plot_importance, plot_tree

feature_importance = plot_importance(xgb_model, height=0.9)
```

```
#Predicting with our model for both the train and test data

train_ml["Predictions"] = xgb_model.predict(X_train)

test_ml['Prediction'] = xgb_model.predict(X_test)



#Creating the dataframe with both real and predicted vol

XGBoost_and_rolling                                                    =
pd.concat([pd.DataFrame(list(train_ml["Predictions"]),list(train_ml["pct_c
hg"]))], axis=1).dropna().reset_index()

XGBoost_and_rolling.rename(columns={"index":"Real_Volatility",0:"Predicted
Volatility"}, inplace=True)

XGBoost_and_rolling.head(10)



# Plot Training Fit

df_train_results = pd.DataFrame({

    'Real_Volatility': train_ml['pct_chg'].values,

    'Predicted_Volatility': train_ml['Predictions'].values

}).dropna()



px.line(

    df_train_results,

    title='XGBoost vs Rolling Volatility of Daily Returns (Train)',

    labels={'value': 'Volatility'},

    width=1000,

    height=500

).show()
```

```
#Plot Test Prediction

plt.figure(figsize=(10, 4))

plt.plot(test_ml["pct_chg"], label='True Returns')

plt.plot(test_ml["Prediction"], label='Predicted Volatility')

plt.title('Volatility Prediction - Rolling Forecast Test with XGBoost',
fontsize=15)

plt.legend(fontsize=10)

plt.show()


#Combined Train & Test Plot

fig = go.Figure()


# Train Section

fig.add_trace(go.Scatter(

    x=train_ml.index, y=train_ml["pct_chg"],

   mode='lines', name='Rolling Volatility (Train)', line=dict(color='blue')

))

fig.add_trace(go.Scatter(

    x=train_ml.index, y=train_ml["Predictions"],

    mode='lines', name='XGBoost Fit (Train)', line=dict(color='red')

))


# Test Section
```

```
fig.add_trace(go.Scatter(

    x=rolling_predictions.index, y=rolling_vol[-test_size:],

    mode='lines', name='Rolling Volatility (Test)', line=dict(color='green')

))

fig.add_trace(go.Scatter(

    x=rolling_predictions.index, y=rolling_predictions,

    mode='lines', name='XGBoost Forecast (Test)', line=dict(color='orange')

))


fig.update_layout(

    title='XGBoost Fit vs Rolling Volatility (Train & Test)',

    width=1000,

    height=500

)

fig.show()


from     sklearn.metrics     import     r2_score,     mean_squared_error,
mean_absolute_percentage_error



common_index                                                                 =
cond_vol_train.dropna().index.intersection(rolling_vol_train.dropna().inde
x)

y_true_train = rolling_vol_train.loc[common_index].values

y_pred_train = cond_vol_train.loc[common_index].values
```

```
resid_train = y_true_train - y_pred_train


print("GARCH Train Fitting Residual Summary")

print("R²   :", r2_score(y_true_train, y_pred_train))

print("RMSE :", np.sqrt(mean_squared_error(y_true_train, y_pred_train)))

print("MAPE :", mean_absolute_percentage_error(y_true_train, y_pred_train))


y_true_xgb_train = train_ml["pct_chg"].values

y_pred_xgb_train = train_ml["Predictions"].values


resid_xgb_train = y_true_xgb_train - y_pred_xgb_train


print("XGBoost Train Fitting Residual Summary")

print("R²   :", r2_score(y_true_xgb_train, y_pred_xgb_train))

print("RMSE        :",        np.sqrt(mean_squared_error(y_true_xgb_train,
y_pred_xgb_train)))

print("MAPE      :",      mean_absolute_percentage_error(y_true_xgb_train,
y_pred_xgb_train))


y_true = rolling_vol_test[-len(rolling_preds):]

y_pred = rolling_preds


df_valid = pd.DataFrame({

    'true': y_true,
    'pred': y_pred

}).dropna()


y_true_clean = df_valid['true'].values

y_pred_clean = df_valid['pred'].values


resid = y_true_clean - y_pred_clean


print("GARCH Test Forecast Residual Summary")

print("R²   :", r2_score(y_true_clean, y_pred_clean))

print("RMSE :", np.sqrt(mean_squared_error(y_true_clean, y_pred_clean)))

print("MAPE :", mean_absolute_percentage_error(y_true_clean, y_pred_clean))


y_true = rolling_vol_test[-len(rolling_preds):]

y_pred = rolling_preds


df_valid = pd.DataFrame({

    'true': y_true,

    'pred': y_pred

}).dropna()


y_true_clean = df_valid['true'].values

y_pred_clean = df_valid['pred'].values
```

```
resid = y_true_clean - y_pred_clean


print("GARCH Test Forecast Residual Summary")

print("R²   :", r2_score(y_true_clean, y_pred_clean))

print("RMSE :", np.sqrt(mean_squared_error(y_true_clean, y_pred_clean)))

print("MAPE :", mean_absolute_percentage_error(y_true_clean, y_pred_clean))
```

LSTM:

```
import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import keras as krs

import tensorflow as tf

from keras.models import Sequential

from keras.layers import Dense

from sklearn.model_selection import train_test_split

# Import data

import os

from google.colab import drive

drive.mount('/content/drive', force_remount = True)

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from     sklearn.metrics     import     r2_score,     mean_squared_error,
mean_absolute_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense


# -------- STEP 1: Compute 22-day volatility from pct_change --------

# Assume df['pct_change'] exists and is in percent (e.g., 0.5 means 0.5%

df['volatility']  =  returns.rolling(window=22).std()  *  np.sqrt(252)    #
annualized

df = df.dropna().reset_index(drop=True)

# -------- STEP 2: Scale volatility (Corrected) --------

# Create the scaler object

scaler = MinMaxScaler()

# -------- STEP 3: Clean 70/30 train-test split BEFORE lagging --------

split_idx = int(len(df) * 0.7)  # Split index based on original data length

# Fit the scaler on the training data only and transform both

train_scaled = scaler.fit_transform(df[['volatility']][:split_idx])

test_scaled = scaler.transform(df[['volatility']][split_idx:])

# -------- STEP 4: Create lagged sequences --------

def create_lagged_data(data, lookback):

    X, y = [], []

    for i in range(lookback, len(data)):

        X.append(data[i - lookback:i])

        y.append(data[i])

    return np.array(X), np.array(y)
```

```
lookback = 10

X_train, y_train = create_lagged_data(train_scaled, lookback)

X_test, y_test = create_lagged_data(test_scaled, lookback)


# -------- STEP 5: Build and train LSTM model --------

model = Sequential()

model.add(LSTM(64, input_shape=(lookback, 1)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.1)

# -------- Get predictions for training data --------

y_train_pred = model.predict(X_train)

y_train_pred_inv = scaler.inverse_transform(y_train_pred)

y_train_inv = scaler.inverse_transform(y_train)

# -------- STEP 6: Predict and evaluate --------

y_pred = model.predict(X_test)

y_pred_inv = scaler.inverse_transform(y_pred)

y_test_inv = scaler.inverse_transform(y_test)

# -------- Calculate training metrics --------

train_r2 = r2_score(y_train_inv, y_train_pred_inv)

train_mse = mean_squared_error(y_train_inv, y_train_pred_inv)

# Calculate training MAPE (with handling for zero values in y_train_inv)

train_mape = np.mean(np.abs((y_train_inv - y_train_pred_inv) /
np.where(y_train_inv != 0, y_train_inv, np.nan))) * 100
```

```
train_mape = np.nan_to_num(train_mape)  # Replace NaN with 0

# Calculate RMSE

rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))



# Calculate MAPE (with handling for zero values in y_test_inv)

mape = np.mean(np.abs((y_test_inv - y_pred_inv) / np.where(y_test_inv != 0,
y_test_inv, np.nan))) * 100

mape = np.nan_to_num(mape) # Replace NaN with 0

# -------- Print training and test metrics --------

print("Training Metrics:")

print("R² Score:", train_r2)

print("Mean Squared Error (MSE):", train_mse)

print("Mean Absolute Percentage Error (MAPE):", train_mape)

# Print metrics

print("Root Mean Squared Error (RMSE):", rmse)

print("Mean Absolute Percentage Error (MAPE):", mape)

r2 = r2_score(y_test_inv, y_pred_inv)

print("R² Score:", r2)

# -------- Show Features Used --------

print("\nFeatures Used:")

for i in range(lookback):

    print(f"Lag {i+1}: Volatility (t-{lookback-i})")

plt.figure(figsize=(12, 5))

plt.plot(y_test_inv, label='Actual Volatility')
plt.plot(y_pred_inv, label='Predicted Volatility')

plt.title("Volatility Prediction vs Actual")

plt.xlabel("Time (days)")

plt.ylabel("Volatility")

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```