DATA3888: Predicting Stock Volatility

2025-06-01

Table of contents

Chapter 1

Executive Summary

1.1 Research Question

Short-term volatility forecasting is critical for high-frequency trading, risk management, and market-making. This study investigates whether advanced deep learning architectures can enhance predictive accuracy on ultra-high-frequency Level-2 Limit Order Book (LOB) data. Specifically, we evaluate whether a *Transformer-based* model can outperform traditional *Long Short-Term Memory (LSTM) networks* when both models are trained on identically preprocessed datasets with equivalent engineered features.

1.1.1 Hypothesis

1.1.1.1 Null Hypothesis (H)

Given equivalent data quality, feature engineering, and preprocessing conditions, a Transformer-based model **does not outperform** LSTM networks in short-term volatility forecasting on Level-2 LOB data.

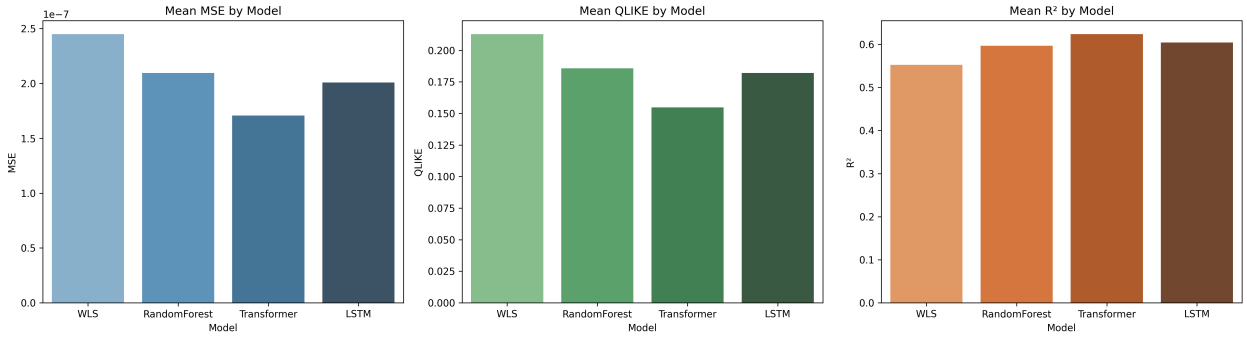
1.1.1.2 Alternative Hypothesis (H)

Given equivalent data quality, feature engineering, and preprocessing conditions, a Transformer-based model **outperforms** LSTM networks in short-term volatility forecasting on Level-2 LOB data.

1.2 Key Findings

Our analysis shows that the Transformer model achieves superior performance in short-term volatility forecasting, consistently outperforming LSTM networks. In general, it attains an **out-of-sample R^2 of 0.62** and **QLIKE loss of 0.14**, capturing 62% of realized volatility variation—substantially

higher than the *WLS* and *Random Forest* baselines, and measurably better than the *LSTM*. In favorable market regimes, performance improves to **out-of-sample R^2 of 0.78** and **QLIKE loss of 0.05**. The Transformer's attention mechanism effectively identifies temporally localized predictive structures in high-frequency order flow, yielding robust and highly accurate forecasts.



1.3 Relevance

The demonstrated accuracy of the Transformer model has direct implications in high-frequency trading, risk management, and option pricing. High-quality real-time volatility estimates support tighter bid-ask spreads, more efficient hedging, and improved capital deployment under microsecond constraints. By capturing core properties of volatility (clustering, persistence, and asymmetry) the Transformer allows better real-time decision-making in trading and portfolio optimisation contexts.

Chapter 2

Volatility Prediction

Volatility is the degree to which the return of an asset deviates from its expected value over a given time horizon. Formally if r_t denotes the logarithmic return at time t and $\mu = \mathbb{E}[r_t]$ is its mean, then the (unconditional) volatility σ can be defined as the standard deviation of r_t:

$$\sigma \;=\; \sqrt{\mathbb{E}[(r_t - \mu)^2]}$$

In empirical, high-frequency settings, one often works with realized volatility over a discrete intervals. Formally, if $\{r_{t,i}\}_{i=1}^{N}$ are the equally-spaced intraday returns within that interval, then the realized volatility is given by:

$$\sigma_{\text{realized}} = \sqrt{\sum_{i=1}^{N} r_{t,i}^2}$$

2.1 Loading Data

The dataset consists of ultra-high-frequency Level-2 LOB snapshots recorded at 1-second resolution across 600-second intervals. Each row corresponds to a fixed-time observation of a single stock and includes:

- **Price features:** Best bid and ask prices at levels 1 and 2.
- **Volume features:** Corresponding sizes at these levels.
- **Time identifiers:** `time_id` and `seconds_in_bucket`.
- **Stock identifier:** `stock_id` uniquely indexing each equity.

We load the data from all *112 individual CSV files*, each indexed by `time_id` and `seconds_in_bucket`. The data is scanned lazily using **Polars** with strict schema enforce-

ment to ensure memory efficiency. Files are then consolidated and written as a compressed **Parquet** object for downstream processing.

2.2 Filtering

Now, we aim to *predict*, *rank*, and *cluster* stocks based on the predictability of their short-term (10-minute) volatility. First, we extract snapshot and aggregate features from the raw LOB data. For each stock, we compute a 13-dimensional feature vector by averaging across all intervals. Finally, we apply K-Means clustering to group stocks with similar microstructure volatility profiles, train a lagged linear model, and select the top 30 stocks.

2.2.1 Snapshot features

Using the existing features in our dataset we create the following:

- **Microprice** (m_t)

$$m_t = \frac{P_{1,t}^{(b)} \cdot V_{1,t}^{(a)} + P_{1,t}^{(a)} \cdot V_{1,t}^{(b)}}{V_{1,t}^{(b)} + V_{1,t}^{(a)} + \varepsilon}$$

Here ε is a small constant used to avoid division by 0. It is set to 10^{-12}. The microprice tilts toward the side with greater resting depth, capturing which side (bid vs. ask) is likelier to execute next. If $V_{1,t}^{(b)} + V_{1,t}^{(a)} = 0$, we fallback to the mid-price.

- **Spreads** ($\Delta_t^{(1)}$, $\Delta_t^{(2)}$)

$$\Delta_t^{(1)} = P_{1,t}^{(a)} - P_{1,t}^{(b)}$$

$$\Delta_t^{(2)} = P_{2,t}^{(a)} - P_{2,t}^{(b)}$$

Spreads measure instantaneous liquidity, and empirically, wider spreads often precede higher short-term volatility.

- **Imbalance** ($I_{1,t}$)

$$I_{1,t} = \frac{V_{1,t}^{(b)} - V_{1,t}^{(a)}}{V_{1,t}^{(b)} + V_{1,t}^{(a)} + \varepsilon}$$

This imbalance at top-of-book often signals short-term directional pressure. A positive value indicates *bid-side dominance* (upward pressure) while a negative value indicates *sell-side dominance*

(downward pressure).

- **Book Pressure** (B_t)

$$B_t = \frac{V_{1,t}^{(b)} + V_{2,t}^{(b)}}{V_{1,t}^{(b)} + V_{2,t}^{(b)} + V_{1,t}^{(a)} + V_{2,t}^{(a)} + \varepsilon}$$

- **Mean Snapshot Features** ($\overline{x}_\tau$)

For each snapshot feature we also calculate:

$$\overline{x}_\tau = \frac{1}{|\mathcal{T}_\tau|} \sum_{t \in \mathcal{T}_\tau} x_t$$

Here $\mathcal{T}_\tau$ is the set of valid seconds in interval τ. Averaging instantaneous features over the entire 10-minute block yields measures of the prevailing book state. These means form core components of each stock's 13-dimensional *fingerprint*.

2.2.2 Aggregate Features

We build our aggregate features on the second-by-second *log-returns* that captures the *instantaneous price changes*.

- **Microprice log-return** (r_t)

$$r_t = \ln m_t - \ln m_{t-1} = \ln(m_t/m_{t-1})$$

- **Realized Volatility** (RV_τ)

$$\mathrm{RV}_\tau = \sqrt{\sum_{t \in \mathcal{T}_\tau} \left[\ln(m_t) - \ln(m_{t-1})\right]^2}$$

We drop any $r_t = \pm\infty$ (which can occur if m_{t-1} or m_t is zero) and omit *NaNs*. If fewer than two valid log-returns exist, we set $\mathrm{RV}_\tau = 0$ for that interval.

- **Realized Skewness** (κ_τ)

$$\kappa_\tau = \frac{1}{|\mathcal{T}_\tau|} \sum_{t \in \mathcal{T}_\tau} \left(\frac{\ln(m_t/m_{t-1}) - \frac{1}{|\mathcal{T}_\tau|} \sum_{t \in \mathcal{T}_\tau} \ln(m_t/m_{t-1})}{\sqrt{\frac{1}{|\mathcal{T}_\tau|-1} \sum_{t \in \mathcal{T}_\tau} (\ln(m_t/m_{t-1}) - \bar{r})^2}} \right)^3$$

Skewness indicates whether returns in window τ have a heavy tail on one side, which can help differentiate intervals driven by one-sided imbalance or asymmetric order flow. If $|\mathcal{T}_\tau| < 2$ we set

$\kappa_\tau = 0$.

- **Autocorrelation of Log-Returns** (ρ_τ)

$$\rho_\tau = \frac{\sum_{t \in \mathcal{T}_\tau \setminus \{\min\}} (r_t - \bar{r})\,(r_{t-1} - \bar{r})}{\sum_{t \in \mathcal{T}_\tau} (r_t - \bar{r})^2}$$

If we have fewer than 3 valid returns, we set $\rho_\tau = 0$.

- **Tick Frequency** (F_τ)

$$F_\tau = |\mathcal{T}_\tau|$$

When combined with the averaged snapshot features, we obtain a 13-dimensional summary for each 10-minute window. Averaging these over all intervals yields the per-stock fingerprint used in clustering.

2.2.3 IQR-Based Filtering

Some stocks exhibit extreme average volatility—often due to illiquidity or data quirks—which can dominate clustering and forecast evaluation. To address this, we apply *IQR-based filtering* on each stock's average realised volatility.

For each stock i, compute its mean realized volatility:

$$\mu_i = \frac{1}{T_i} \sum_{\tau=1}^{T_i} \mathrm{RV}_\tau^{(i)}.$$

Let Q_1 and Q_3 be the 25th and 75th percentiles of $\{\mu_i\}$, and define

$$\mathrm{IQR} = Q_3 - Q_1, \quad \text{lower} = Q_1 - M \times \mathrm{IQR}, \quad \text{upper} = Q_3 + M \times \mathrm{IQR},$$

with $M = 1.5$. We exclude stocks with $\mu_i <$ lower or $\mu_i >$ upper. Stocks above the bound tend to have sporadic spikes, while those below the bound often have near-zero volatility. Removing these extremes ensures the remaining sample reflects typical trading behavior.

2.2.4 K-Means Clustering

For each filtered stock i, compute its 13-dimensional fingerprint as:

$$\mu^{(i)} = \frac{1}{T_i} \sum_{\tau=1}^{T_i} \mathbf{z}_\tau^{(i)},$$

where $\mathbf{z}_\tau^{(i)}$ is the snapshot-aggregate vector for stock i in interval τ.

Because these 13 components span heterogeneous units, we standardize each dimension to prevents features with large raw magnitudes from dominating *Euclidean distances* and aligns each dimension's influence on clustering.:

$$\tilde{\mu}_j^{(i)} = \frac{\mu_j^{(i)} - \bar{\mu}_j}{s_j}, \quad \bar{\mu}_j = \frac{1}{N_{\text{filtered}}} \sum_{i=1}^{N_{\text{filtered}}} \mu_j^{(i)}, \quad s_j = \sqrt{\frac{1}{N_{\text{filtered}} - 1} \sum_{i=1}^{N_{\text{filtered}}} (\mu_j^{(i)} - \bar{\mu}_j)^2}.$$

We then run K-Means on $\{\tilde{\mu}^{(i)}\}_{i=1}^{N_{\text{filtered}}}$ with $k = 5$, chosen via **elbow-method** and **silhouette-score analyses**. This groups stocks into homogeneous liquidity–volatility profiles, facilitating both qualitative interpretation and the option to fit separate predictive models per cluster.

2.2.5 Stock Selection

Having grouped stocks, we assess the predictability of each 10-minute realized volatility using a **lagged linear model** on five meta-features. For stock i, let $z_\tau^{(i)} \in \mathbb{R}^5$ be the selected features at interval τ. We fit:

$$\text{RV}_\tau^{(i)} = \gamma_0 + \sum_{j=1}^{5} \gamma_j \, z_{\tau-1,j}^{(i)} + \varepsilon_\tau$$

via **expanding-window OLS** over $\tau = \tau_{\text{start}}, \ldots, T_i$, where $\tau_{\text{start}} = \max\{2, \lfloor T_i/2 \rfloor\}$ ensures at least half the data is available after a 1-lag burn-in. At each τ, we estimate on all data from τ_{start} to $\tau - 1$ and forecast $\widehat{\text{RV}}_\tau^{(i)}$. From these out-of-sample forecasts, we compute **Out-of-sample** R^2 and **QLIKE**. We combine these into a score:

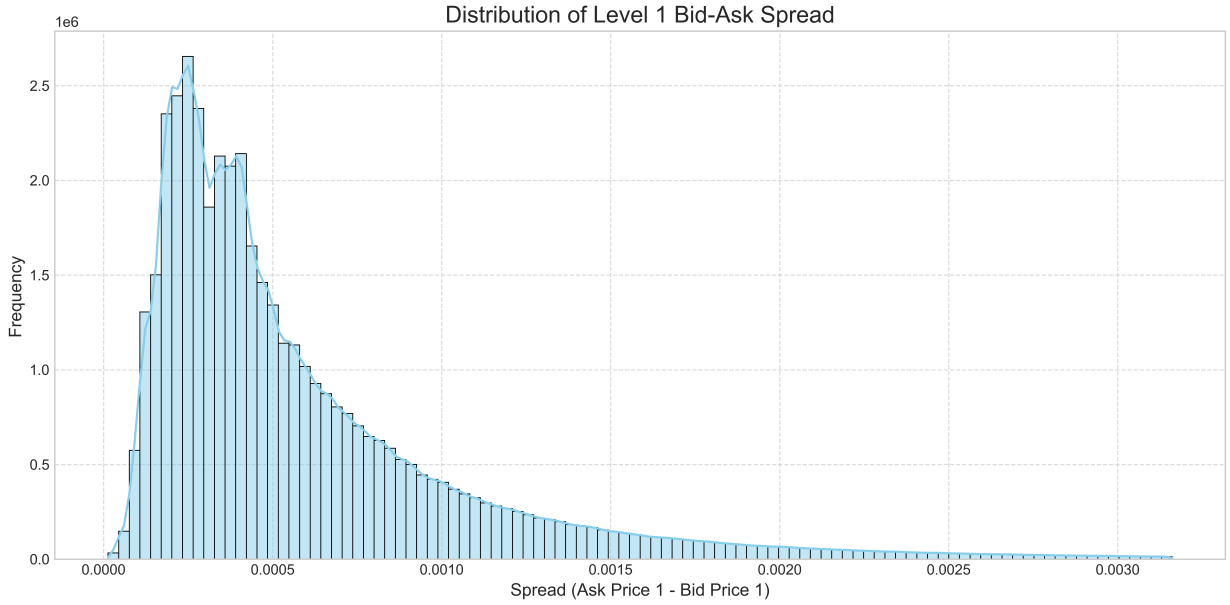
$$S_i = 0.5\, R_i^2 \; - \; 0.5\, \text{QLIKE}_i.$$

Stocks are ranked by S_i, and the top 30 are selected to keep downstream modeling and cross-validation tractable.

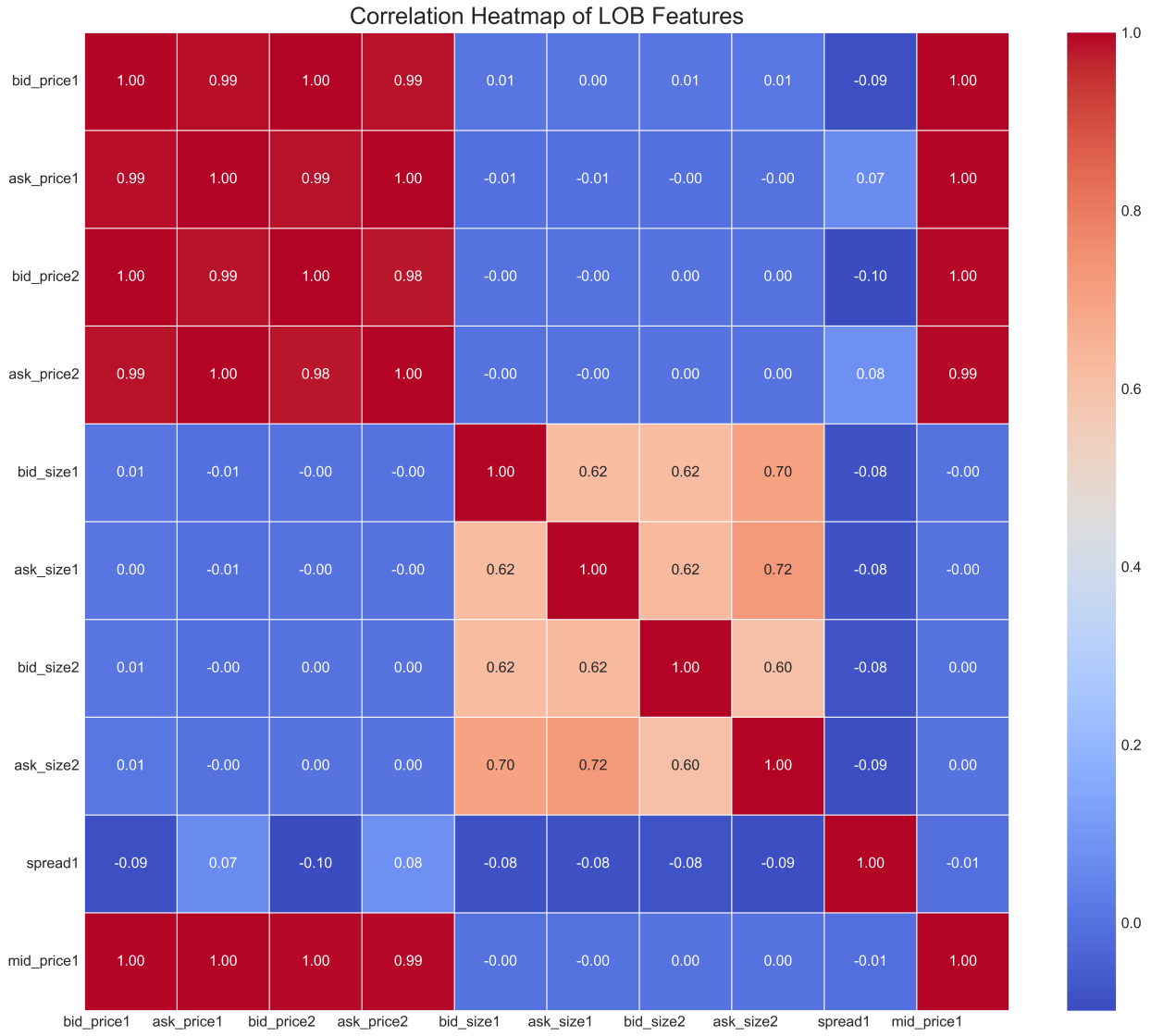
2.3 Exploratory data analysis

In EDA, we extracted key features and examined their distributions to guide scaling decisions. These distributions were **right-skewed** with small positive values, indicating a transaction costs in

trading.



Comparing across our selected stocks revealed differences in *median* and *dispersion*, highlighting *liquidity variation*. We then generated a correlation **heatmap** to assess linear relationships among features and performed a *grid search* to determine the optimal window size for rolling realized volatility.



2.4 Feature Engineering

For the top 30 stocks, along with the snapshot and aggregate features mentioned in the *filtering* section we engineered various second-by-second and rolling-window features. These are designed to capture price dynamics, order-book complexity, and temporal patterns, yielding a richer input set for modeling.

1. **Shannon's entropy**

$$s_t = [V_{1,t}^{(b)}, V_{2,t}^{(b)}, V_{1,t}^{(a)}, V_{2,t}^{(a)}], \quad p_{t,i} = \frac{s_{t,i}}{\sum_{j=1}^{4} s_{t,j}}$$

$$H_t = -\sum_{i=1}^{4} p_{t,i} \ln(p_{t,i}), \quad H_t^{Normal} = \frac{H_t}{\ln(4)}$$

This captures the uncertainty in order book volume distribution,scaled to $[0, 1]$ for compatibility. High entropy indicates balanced bid/ask sizes, implying more unpredictable price movements.

2. **Future Volatility Target**

$$RV_t^{Future} = RV_{t+30}$$

We shift the realized volatility forward by *30 seconds* to create the one-step-ahead target.

3. **Bipower Variation**

$$Bvar_t = \frac{1}{30} \sum_{j=1}^{30} |r_{t-j}| \cdot |r_{t-j-1}|$$

4. **Weighted Average Price (WAP)**

$$WAP_t = \frac{P_{1,t}^{(b)} V_{1,t}^{(a)} + P_{1,t}^{(a)} V_{1,t}^{(b)}}{V_{1,t}^{(b)} + V_{1,t}^{(a)}}$$

$$WAP_t^{(log)} = \ln(WAP_t / WAP_{t-1})$$

5. **Lagged Features**

$$x_{t-1},\ x_{t-2},$$

For each of **imbalance**, **book_pressure**, and **log_return**, we create first and second lags within each `time_id` group.

6. **Rolling-Window Summary**

$$Log - Returns_t^{Rolling} = \sqrt{\frac{1}{29} \sum_{j=1}^{30} \left(r_{t-j} - \overline{r}_{t-1:t-30}\right)^2}$$

7. **Intrablock Temporal Encoding**

$$\theta_t = 2\pi \frac{\text{seconds\_in\_bucket}_t}{600}, \quad \text{sec\_sin}_t = \sin(\theta_t), \quad \text{sec\_cos}_t = \cos(\theta_t).$$

These encode the position within each 10-minute block and capture cyclical effects without imposing a linear time trend.

8. **Log-Transformation of Book Sizes**

$$\text{size\_log}_t = \ln(1 + \text{size}_t),$$

Finally, we apply a **log1p** transform to each raw size column ($V_1^{(b)}$, $V_1^{(a)}$, $V_2^{(b)}$, $V_2^{(a)}$) replacing the original size columns with their log-scaled counterparts. This reduces heavy-tail effects in volume distributions, improving numerical stability. Rows with **NaNs** or **infs** are dropped after transformation and we also cast all `float64` values to `float32` to minimize memory usage.

2.5 Feature Selection

2.5.1 Variance Thresholding

First we drop any feature with **zero variance** across all stocks and all seconds within each 10-minute bucket. As a constant predictor cannot explain future volatility and only adds noise. For each column, compute its empirical variance over 30 stocks and all time points; if $\text{Var}(x) = 0$, we remove that feature.

2.5.2 Spearman Correlation

Next we compute **Spearman's rank correlation** to detect *near-collinearity* among remaining features. For variables x and y over n observations, Spearman's ρ_s is

$$\rho_s(x, y) \;=\; 1 \;-\; \frac{6}{n(n^2-1)} \sum_{k=1}^{n} \big(\text{rank}(x_k) - \text{rank}(y_k)\big)^2.$$

We build the full $p \times p$ **Spearman matrix** and, whenever $|\rho_s| > 0.98$, identify one redundant feature. To choose which to drop, sum each feature's absolute correlations with all others; the feature with the smaller total is removed.

2.6 Models

2.6.1 Weighted Least Squares

To forecast next-interval realized volatility, we fit a WLS model using selected features. WLS accounts for heteroscedasticity by assigning inverse-variance weights. Let $X \in \mathbb{R}^{N \times p}$ be the predictor matrix (including an intercept), $\mathbf{y} \in \mathbb{R}^N$ the one-step-ahead realized-volatility target, and $\mathbf{w} \in \mathbb{R}^N$ the weights, where for each observation t:

$$w_t = \frac{1}{\mathrm{Var}(y_{t-2000:t}) + \varepsilon},$$

with a rolling window of size 2000 and small ε to avoid division by zero. This down-weights periods of high volatility noise, stabilizing coefficient estimates.

The WLS minimises:

$$\min_{\beta} \sum_{t=1}^{N} w_t (y_t - \beta_0 - \mathbf{x}_t^\top \beta)^2,$$

where β_0 is the intercept and $\mathbf{x}_t$ the p-dimensional features at t.

We split the data chronologically (80% train, 20% test), add a constant column, and fit WLS. This yields coefficient estimates that are less influenced by high-variance intervals, producing more stable forecasts under volatility heterogeneity.

2.6.2 Random Forest

A Random Forest is an ensemble of decision trees that reduces variance and captures nonlinear feature interactions, making it well-suited for predicting realized volatility which exhibits spikes and heteroscedasticity.

We first apply a **log1p** transform to the target to compress large spikes:

$$y_t^{(\log)} = \ln(1 + \mathrm{rv\_future}_t)$$

We then split the data by `time_id` to preserve temporal ordering: the first 80% of sessions are reserved for *train* and *validation* and the last 20% for *test*. Within the first 80%, the first 90% of sessions form the training set and the remaining forms validation set. For each split we extract the feature matrix X and log-target.

We train the model with the following parameters:

```
rf = RandomForestRegressor(
    n_estimators=500,
    max_depth=None,
    max_features='sqrt',
    min_samples_leaf=3,
    bootstrap=True,
    n_jobs=-1,
    random_state=42,
```

```
    verbose=1
)
rf.fit(X_train, y_train)
```

After fitting, we predict on validation to compute the out-of-sample RMSE for manual hyperparameter adjsment:

$$\text{RMSE}_{\text{val}} = \sqrt{\frac{1}{N_{\text{val}}} \sum_{i=1}^{N_{\text{val}}} \left(y_i^{(\log)} - \widehat{y}_i^{(\log)}\right)^2},$$

Finally, we evaluate on the held-out test set by predicting $\widehat{y}_{\text{test}}^{(\log)}$ to compute final metrics (RMSE, R^2, and QLIKE) in *log-space* for stability.

2.6.3 LSTM

An LSTM captures temporal dependencies over multiple seconds, making it well-suited for modeling sequential volatility patterns as well.

First we fit a **MinMaxScaler** on the training set. For the target, we again use **log1p** scaling:

$$y^{(\log)} = \ln(1 + RV^{Future}), \quad y^{(\text{scaled})} = \frac{y^{(\log)} - \min(y^{(\log)})}{\max(y^{(\log)}) - \min(y^{(\log)})}.$$

Then we group the data by `time_id` and slide a 30-second window over the scaled feature matrix $X \in \mathbb{R}^{T \times p}$ to form input sequences of shape $(T - 30,\, 30,\, p)$, with targets at $t + 30$. We split the sessions chronologically: 80% for train and validation (90% train, 10% validation), 20% for test.

Model Architecture:

Layer (type)

Output Shape

Param #

lstm_8 (LSTM)

(None, 30, 64)

25,088

dropout_8 (Dropout)

(None, 30, 64)

0

lstm_9 (LSTM)

(None, 32)

12,416

dropout_9 (Dropout)

(None, 32)

0

dense_8 (Dense)

(None, 16)

528

dense_9 (Dense)

(None, 1)

17

We train using the **Adam** optimizer with a learning rate of 10^{-4} and *minimize MSE* on the scaled log-volatility. Training uses a batch size of 128 and runs for up to 50 epochs, with **early stopping** triggered after 5 epochs of no validation improvement. We have $38,049$ trainable parameters.

After training, we predict the scaled log-volatility $\widehat{y}^{(\text{scaled})}$ on the test set and invert it using the stored training minima and maxima:

$$\widehat{y}^{(\log)} = \widehat{y}^{(\text{scaled})} \times (\max(y^{(\log)}) - \min(y^{(\log)})) + \min(y^{(\log)}), \quad \widehat{RV^{Future}} = \exp(\widehat{y}^{(\log)}) - 1.$$

We then evaluate performance on the original volatility scale using RMSE, R^2, and QLIKE.

2.6.4 Transformer

A Transformer captures long-range dependencies via self-attention, making it very effective for volatility sequences with complex patterns.

Similar to LSTM we fit **MinMax scalers** on training data. For the target, we again use **log1p** scaling:

$$y^{(\log)} = \ln(1 + RV^{Future}), \quad y^{(\text{scaled})} = \frac{y^{(\log)} - \min(y^{(\log)})}{\max(y^{(\log)}) - \min(y^{(\log)})}.$$

The sequence construction and the train-test-validation split is identical to the LSTM model. This model is trained on $101,569$ parameter, taking more 300 GPU-hours to train.

Model Architecture:

Layer (type)

Output Shape

Param #

Connected to

input_layer (InputLayer)

(None, 30, 23)

0

–

dense (Dense)

(None, 30, 64)

1,536

input_layer[0][0]

multi_head_attention (MultiHeadAttention)

(None, 30, 64)

16,640

dense[0][0] (q, k, v)

add (Add)

(None, 30, 64)

0

dense[0][0], multi_head_attention

layer_normalization (LayerNormalization)

(None, 30, 64)

128

add[0][0]

dense_1 (Dense)

(None, 30, 256)

16,640

layer_normalization

dense_2 (Dense)

(None, 30, 64)

16,448

dense_1[0][0]

dropout_1 (Dropout)

(None, 30, 64)

0

dense_2[0][0]

add_1 (Add)

(None, 30, 64)

0

layer_normalization, dropout_1[0][0]

layer_normalization_2 (LayerNormalization)

(None, 30, 64)

128

add_1[0][0]

multi_head_attention_2 (MultiHeadAttention)

(None, 30, 64)

16,640

layer_normalization_2

add_2 (Add)

(None, 30, 64)

0

layer_normalization_2, multi_head_attention_2

layer_normalization_3 (LayerNormalization)

(None, 30, 64)

128

add_2[0][0]

dense_3 (Dense)

(None, 30, 256)

16,640

layer_normalization_3

dense_4 (Dense)

(None, 30, 64)

16,448

dense_3[0][0]

dropout_3 (Dropout)

(None, 30, 64)

0

dense_4[0][0]

add_3 (Add)

(None, 30, 64)

0

layer_normalization_3, dropout_3[0][0]

layer_normalization_4 (LayerNormalization)

(None, 30, 64)

128

add_3[0][0]

global_average_pooling (GlobalAveragePooling1D)

(None, 64)

0

layer_normalization_4

dropout_4 (Dropout)

(None, 64)

0

global_average_pooling

dense_5 (Dense)

(None, 1)

65

dropout_4[0][0]

We train using the **Adam** optimizer with a learning rate of 10^{-3}, minimizing MSE on the scaled target. Training proceeds with a batch size of 32 for up to 50 epochs, using early stopping based on validation MSE with a patience of 15 epochs; the best weights are restored upon stopping.

After training, we predict $\widehat{y}^{(\text{scaled})}$ on the test set and invert it using the stored train-set extrema:

$$\widehat{y}^{(\log)} = \widehat{y}^{(\text{scaled})} \times (\max(y^{(\log)}) - \min(y^{(\log)})) + \min(y^{(\log)}), \quad \widehat{RV^{Future}} = \exp(\widehat{y}^{(\log)}) - 1.$$

Model performance is evaluated on the original volatility scale using RMSE, R^2, and QLIKE.

2.7 Evaluation Metrics

2.7.1 R-Squared

$$\overline{y} \;=\; \frac{1}{N}\sum_{i=1}^{N} y_i, \quad R^2 \;=\; 1 \;-\; \frac{\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}.$$

R-Squared measures the fraction of variance in realized volatility explained by the forecasts. Higher R^2 (closer to 1) indicates stronger alignment between predicted and actual volatility. For N test observations $\{y_i\}$ and predictions $\{\widehat{y}_i\}$:

2.7.2 QLIKE

$$\text{QLIKE} = \frac{1}{N}\sum_{i=1}^{N}(r_i - \ln r_i - 1), \quad r_i = \frac{y_i}{\widehat{y}_i},$$

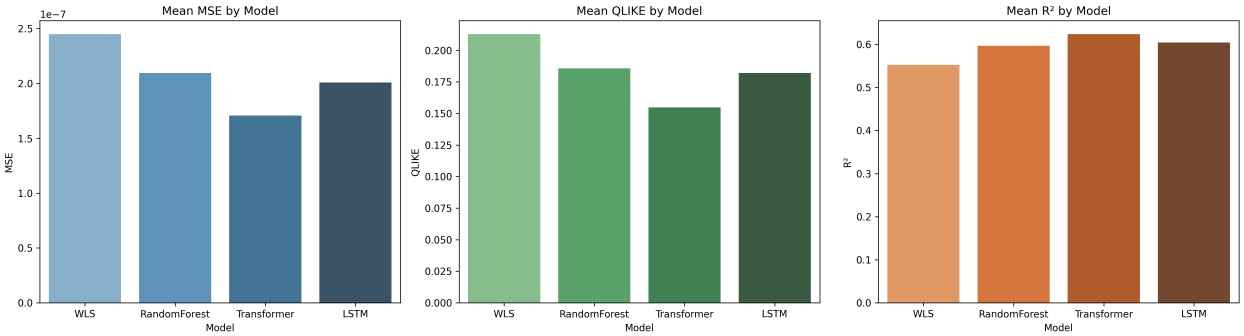
QLIKE is a loss tailored to volatility forecasting that penalizes underestimation more heavily. Lower QLIKE indicates better calibration to actual volatility, especially during spikes. Note that both y_i and $\widehat{y}_i$ clipped to a small $\varepsilon > 0$.

2.7.3 MSE

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2, \quad \text{RMSE} = \sqrt{\text{MSE}}.$$

Mean Squared Error quantifies average squared deviation between forecasts and actual values. Computed on the original scale, MSE penalizes large errors quadratically; on the log scale, it emphasizes relative deviations.

2.7.4 Results



As shown in the comparison chart, the **Transformer** outperformed all other models, achieving the lowest MSE ($\approx 1.7 \times 10^{-}7$), lowest QLIKE ($\approx 0.15$), and highest R^2 (≈ 0.62). The **LSTM** ranked second (MSE $\approx 2.0 \times 10^{-}7$, QLIKE ≈ 0.18, $R^2 \approx 0.59$), followed by **Random Forest** (MSE $\approx 2.2 \times 10^{-}7$, QLIKE ≈ 0.19, $R^2 \approx 0.57$). **WLS** performed worst (MSE $\approx 2.5 \times 10^{-}7$, QLIKE ≈ 0.20, $R^2 \approx 0.55$). This confirms the Transformer's superior accuracy and calibration for 10-minute volatility forecasting.

2.8 Limitations & Improvements

- **Scaling & Preprocessing**
  MinMax and IQR filtering can be distorted by extreme outliers or evolving distributions, leading to unstable features and retained mid-range anomalies. Use robust scaling (e.g., median/IQR, Winsorization) and dynamic trimming (rolling statistics) to adapt to real-time volatility shifts.

- **Transformer**
  A 30-step context window and only two encoder layers (d_model=64) may miss longer-range, multi-scale dynamics. The model is prone to overfitting if validation splits don't capture diverse regimes, and self-attention's cost forced training on a subset of stocks. Introduce hierarchical/dilated attention or lightweight positional encodings to capture broader context without inflating size. Apply pruning/distillation or scale hardware so full data can be used; consider fallback to a shallower temporal CNN if compute is constrained.

- **LSTM**
  Fixed 30-second sequences can overlook dependencies beyond that window; two-layer architecture still risks overfitting, and MinMax scaling is sensitive to session-specific outliers. Experiment with longer (60–120s) or hierarchical windows and add robust normalization (e.g., IQR). Include stronger regularization (batch norm, zoneout) or attention to improve generalization and interpretability.

- **Random Forest**
  Ignores temporal order, relying solely on engineered lags to capture autocorrelation; rare volatility spikes remain underrepresented, biasing leaf estimates. Employ sample-level weighting or oversampling for high-volatility intervals; consider gradient-boosted models (e.g., XGBoost) that can better adapt to tail events, and incorporate sequence-aware features if latency allows.

- **WLS & Linear Models** Linear form cannot capture nonlinear regime shifts; static rolling windows (e.g., size=2000) may generate noisy weights during spikes. Use time-decay or adaptive windowing to downweight outdated data, and augment with spline expansions or switch to regularized rolling regressions (Elastic Net) to model mild nonlinearities.

- **Interpretability & Latency**
  Deep models (Transformer, LSTM) are opaque and incur higher inference latency, which can hinder real-time trading decisions. Integrate explainability methods (e.g., SHAP, attention visualization) and compress models via quantization or distillation. Maintain a mixed-ensemble approach where simpler models (WLS, Random Forest) serve low-latency needs, reserving deep models for periodic batch updates.

Chapter 3

Student Contributions

- **Ayush** was responsible for the entire coding workflow, from reading CSV files to developing and evaluating the final model. He also created the Shiny app, prepared the final presentation, and finalized the written report.
- **Kylie** contributed the industry context for the research, edited presentation materials, and co-wrote the final report.
- **Christy** assisted in editing both the final presentation and the written report.
- **Zichun** contributed to model development and implementation, evaluated model performance, and produced supporting visualizations.
- **Tobit** developed candidate models and integrated Ayush's preprocessing and feature engineering pipelines. He also authored the limitations and interpretability sections of the report.

Chapter 4

References

1. Van Rossum, G., & Drake, F. L. (2009). **Python** (Version 3.x) [Programming language]. https://www.python.org/
2. Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). **NumPy** (Version X.X) [Python package]. https://numpy.org/
3. The pandas development team. (2023). **pandas** (Version X.X) [Python package]. https://pandas.pydata.org/
4. The Polars developers. (2023). **polars** (Version X.X) [Python package]. https://www.pola.rs/
5. Hunter, J. D. (2007). **Matplotlib** (Version X.X) [Python package]. https://matplotlib.org/
6. Waskom, M. L. (2021). **seaborn** (Version X.X) [Python package]. https://seaborn.pydata.org/
7. Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). **scikit-learn** (Version X.X) [Python package]. https://scikit-learn.org/
8. Virtanen, P., Gommers, R., Oliphant, T. E., et al. (2020). **SciPy** (Version X.X) [Python package]. https://scipy.org/
9. Seabold, S., & Perktold, J. (2010). **statsmodels** (Version X.X) [Python package]. https://www.statsmodels.org/
10. Abadi, M., Barham, P., Chen, J., et al. (2016). **TensorFlow** (Version X.X) [Machine learning

framework].
https://www.tensorflow.org/

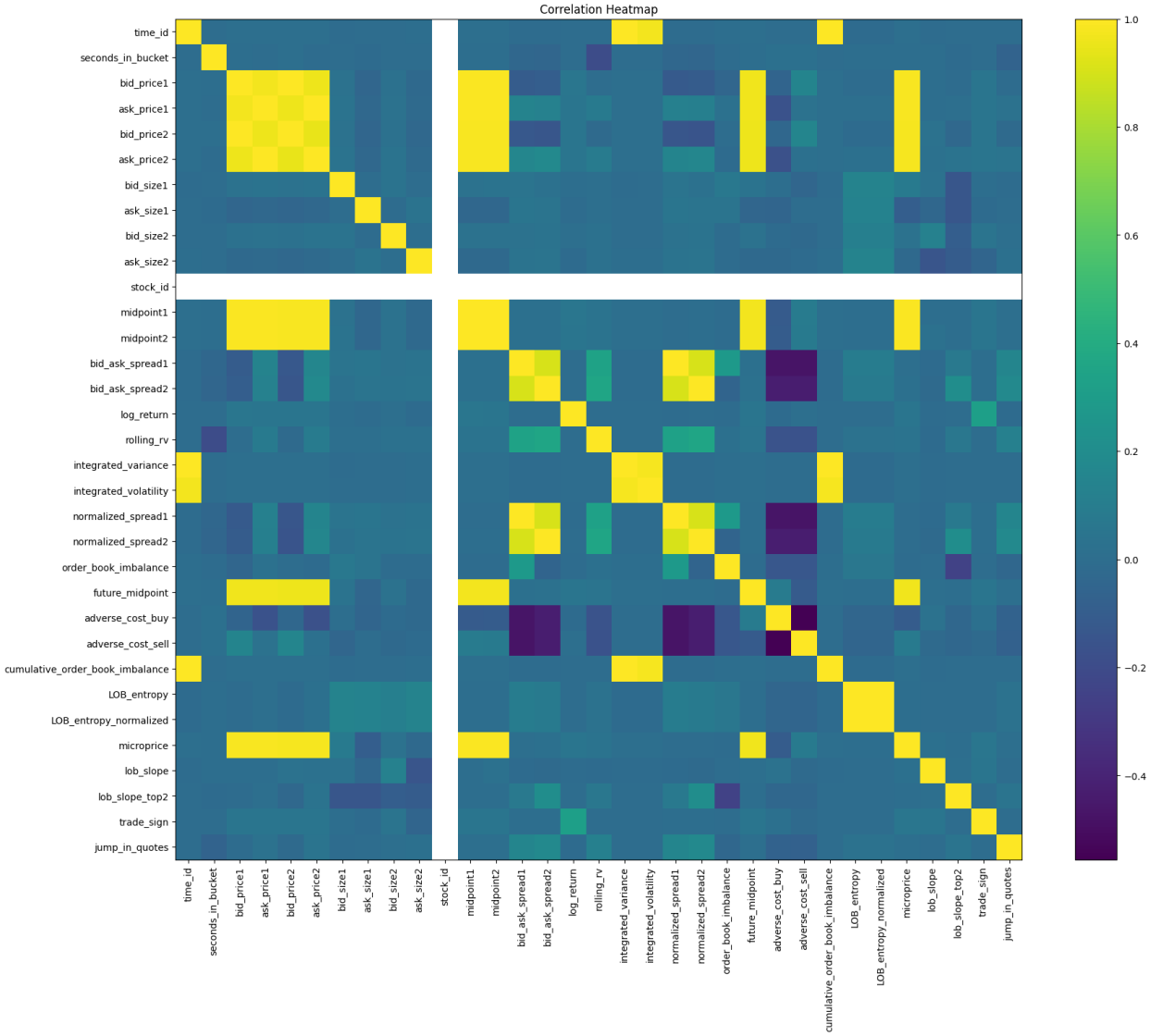
11. Chollet, F. (2015). **Keras** (Version X.X) [Python package].
https://keras.io/

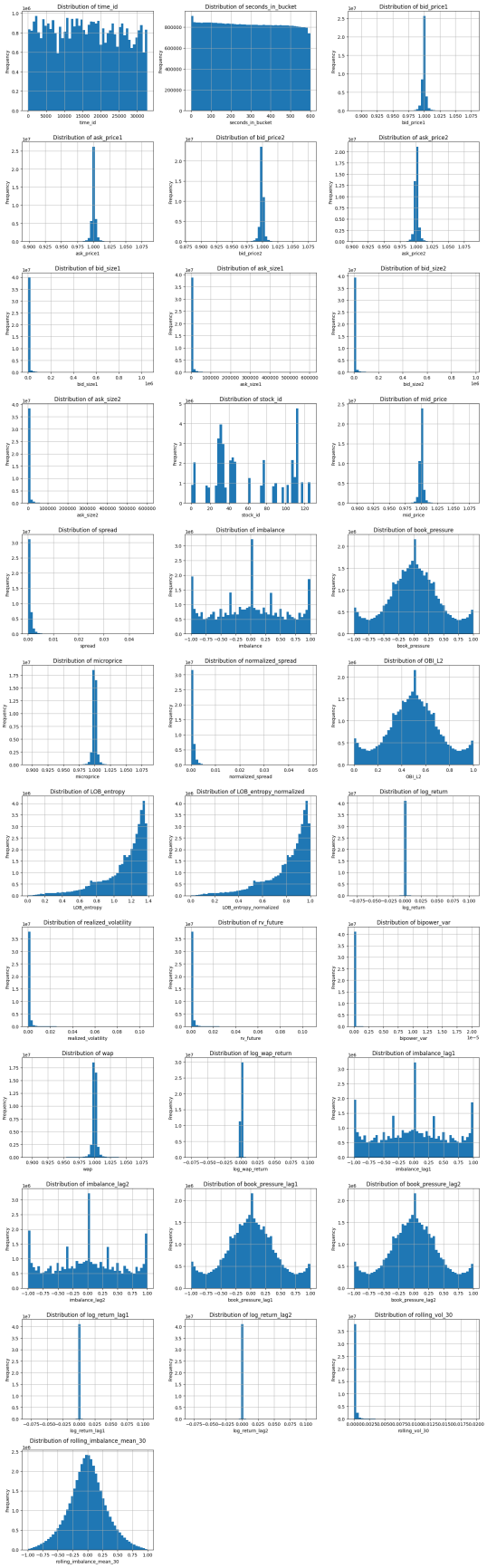
12. The Python Software Foundation. (2023). **glob** (Standard library module).
https://docs.python.org/3/library/glob.html

13. The Python Software Foundation. (2023). **warnings** (Standard library module).
https://docs.python.org/3/library/warnings.html

14. OpenAI. (2023). **ChatGPT** [AI language model].
https://chat.openai.com
This document utilized ChatGPT to correct spelling and grammar errors, format APA-style references for Python packages, and for financial research.

4.1 Plots


Correlation Heatmap
time_id
seconds_in_bucket
bid_price1
ask_price1
bid_price2
ask_price2
bid_size1
ask_size1
bid_size2
ask_size2
stock_id
midpoint1
midpoint2
bid_ask_spread1
bid_ask_spread2
log_return
rolling_rv
integrated_variance
integrated_volatility
normalized_spread1
normalized_spread2
order_book_imbalance
future_midpoint
adverse_cost_buy
adverse_cost_sell
cumulative_order_book_imbalance
LOB_entropy
LOB_entropy_normalized
microprice
lob_slope
lob_slope_top2
trade_sign
jump_in_quotes
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Distribution of time_id
Distribution of seconds_in_bucket
Distribution of bid_price1
Distribution of ask_price1
Distribution of bid_price2
Distribution of ask_price2
Distribution of bid_size1
Distribution of ask_size1
Distribution of bid_size2
Distribution of ask_size2
Distribution of stock_id
Distribution of mid_price
Distribution of spread
Distribution of imbalance
Distribution of book_pressure
Distribution of microprice
Distribution of normalized_spread
Distribution of OBI_L2
Distribution of LOB_entropy
Distribution of LOB_entropy_normalized
Distribution of log_return
Distribution of realized_volatility
Distribution of rv_future
Distribution of bipower_var
Distribution of wap
Distribution of log_wap_return
Distribution of imbalance_lag1
Distribution of imbalance_lag2
Distribution of book_pressure_lag1
Distribution of book_pressure_lag2
Distribution of log_return_lag1
Distribution of log_return_lag2
Distribution of rolling_vol_30
Distribution of rolling_imbalance_mean_30