
Maximizing Profits: Navigating the Closing Bell

Yongyuan Qu

yq2361@columbia.edu

Yufei Lang

yl5404@columbia.edu

Xuefei Xu

xx2437@columbia.edu

Jiexin Pan

jp4479@columbia.edu

Yuhao Wang

yw3924@columbia.edu

Rui Dong

rd3082@columbia.edu

And

Xinyang Chen

xc2713@columbia.edu

Introduction

The international stock market operates in a dynamic and high-stakes environment, characterized by substantial economic ramifications and extensive complexities on a daily basis. The period of utmost activity occurs in the ten minutes immediately preceding the closure of trading sessions, specifically on NASDAQ and other reputable stock exchanges. These transient occurrences are distinguished by their exceptionally high level of volatility, frequently resulting in significant price fluctuations that subsequently influence the economic dialogue for that particular day. The Nasdaq Closing Cross auction, which establishes the official closing prices for exchange-cataloged securities, is a vital element of NASDAQ's trading routine. The prices mentioned above serve as fundamental indicators for a diverse range of market participants, such as analysts, investors, and market observers. They provide support in conducting comprehensive evaluations of both individual security performances and the overall market health. Based on predication purposes, several models are established:

The first model is time series model. Our team incorporate the **GARCH model** to make the forecast **conditional variance** of the stock. GARCH provides insights into the risk dynamics of stocks, enabling traders to adjust their portfolios in anticipation of periods of high volatility. Unlike models that assume constant volatility, GARCH captures the reality of financial markets where volatility tends to cluster, meaning periods of high volatility are followed by high volatility and low by low. This characteristic is effectively applied to the stock price predication. Then, our team try to select several **features** to create the multi-linear regression models. The results shows that these features are meaningful and can be used in explaining the changes of the stock price.

The second model is **random forest** model. Random Forests stand as one of the most powerful and versatile machine learning algorithms today. Random forest is constructed by operating a group of decision trees during training of the model, and outputting the mean prediction of the individual trees. This technique, also known as **ensemble learning**, is behind the success of our latest Random Forest regression model, designed to predict the '**ask price**' for hundreds of Nasdaq listed stocks using data from the order book as well as continuous market orders of stocks. In the following paragraphs, we'll explore the intricacies of our model, how it was built, and its performance on our test data.

The accuracy of the model will be assessed using the **Mean Absolute Error (MAE)** calculated from the predicted return to the observed target. The Mean Absolute Error (MAE) is a reliable metric that provides information regarding the average magnitude of discrepancies between predicted and observed results, irrespective of their direction. **VaR** model will also be added to this part in order to measure the tail risk of the portfolio.

After these models, our team will make a conclusion and interpret our results.

Keywords: GARCH, Conditional Variance, Multi-linear Regression, MAE

II Our Approach

In order to predict the future performance of the equities, it is necessary to cleanse the data prior to backtesting the historical volatility using time series. Finally, we constructed forecasting models and utilized MAE and VaR to quantify the risks.

- 1) Clean and wrangle the datasets
- 2) Apply seasonality test and detrend
- 3) Apply GARCH to find the volatility
- 4) Apply Multi-linear Regression model to predict the price
- 5) Use Random forest to predict the price
- 6) Create MAE and VaR to measure the different risks
- 7) Compare and make conclusions.

III Time Series Analysis

The Autocorrelation Function (ACF) plots for ``ask_price_diff``, ``matched_size_diff``, and ``imbalance_size_diff`` reveal significant autocorrelations at various lags. Notably, the ACF of ``ask_price_diff`` suggests that previous price changes influence future changes, a characteristic crucial in time series forecasting. This pattern of memory and predictability is a key factor in developing financial models and strategies.

The analysis also delves into the aspect of seasonality, a recurrent theme in financial time series. Through month and quarter plots of a constructed seasonal time series, the study reveals clear and consistent seasonal patterns. This seasonality is a critical factor in financial forecasting, suggesting that certain trends or patterns recur at specific times of the year, an understanding that can be leveraged for strategic planning and decision-making in financial contexts.

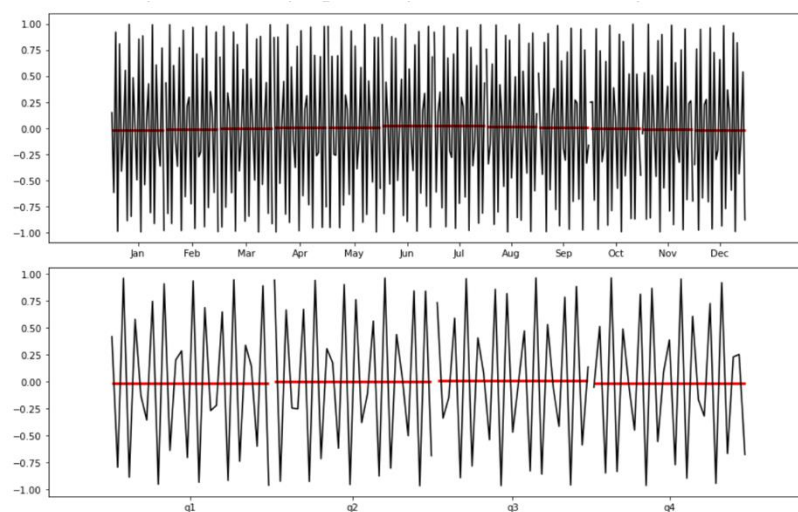


Figure 1 : Seasonality Test

A cornerstone of time series analysis is assessing the stationarity of the data. The Dickey-Fuller test conducted on the `ask_price` series yields a highly significant result (Test Statistic: $-7.28e+11$, p-value: 0.0), rejecting the null hypothesis of non-stationarity. This finding implies that the series has a consistent mean and variance over time, making it suitable for various forecasting models and ensuring that predictions made are reliable and stable.

The GJR-GARCH model, tailored for capturing asymmetries in financial data, reveals significant coefficients. Notably, the model includes a gamma coefficient of 0.8071, indicating a strong asymmetric effect. This finding suggests that negative shocks have a greater impact on volatility than positive ones, an essential insight for risk assessment and management in financial markets.

In conclusion, Significant Autocorrelation indicates that past values are predictive of future values, which is vital for developing forecasting models. Confirmed Stationarity ensures the reliability and stability of the statistical properties of the series over time. Volatility Insights from GJR-GARCH Model unveils the asymmetric impact of shocks on volatility, crucial for risk management.

IV Models Results

The theory behind Multiple Linear Regression (MLR) assumes that the relationship between the dependent and independent variables is linear, errors are normally distributed and independent, and there is no multicollinearity among the independent variables. Prior to building the model, diagnostics tests for these assumptions were conducted to ensure the robustness of the model. The MLR model was constructed using a pipeline that involved preprocessing steps such as imputation of missing values using the median and standard scaling of features. The fitting process signifies that the model is trained and ready for validation.

The scatter plots provide a visual understanding of the relationship between the `'ask_price'` and other variables like `'near_price'`, `'far_price'`, `'bid_price'`, `'imbalance_size'`, and `'matched_size'`. These plots indicate in **Appendix 1**.

We build the multiple linear regression model based on the independent variables like `'near_price'`, `'far_price'`, `'bid_price'`, `'imbalance_size'`, and `'matched_size'` and dependent variable `'ask_price'`. The R-squared value is 0.948. This is a very high value and usually indicates a good fit for the model.

Adjusted R-squared is also 0.948, which provides a correction for the explanatory power of the model considering the number of independent variables. This value is very close to R-squared, indicating that each variable introduced contributes positively to the model. The coefficients of all variables (except for the constant term) are significant, with p-values well below 0.05, meaning that they are all important for predicting `'ask_price'`. The coefficient of `'bid_price'` is 0.9727, indicating that it has a strong positive effect on `'ask_price'`, and `'ask_price'` increases by 0.9727 units on average for every unit increase of `'bid_price'`. Other variables, such as `'imbalance_size'`, `'matched_size'`, `'far_price'` and `'near_price'`, although their coefficients are smaller,

they are still statistically significant and contribute to the explanatory power of the model. The number of conditions of the model is very large ($8.61e+10$), which indicates a strong multicollinearity problem. Although the variables are significant, multicollinearity may affect the stability and interpretation of the coefficients.

Considering that, we need to conduct the multicollinearity check. ``imbalace_size``: The **VIF** value is 1.356646, indicating that there is little collinearity between this variable and the others. ``matched_size``: The VIF value is 1.356450, which also indicates a low collinearity problem. ``far_price``: The VIF value is 1.828506, which is slightly higher than ``imbalace_size`` and ``matched_size``, but still well below 10, indicating that collinearity is not a problem. ``near_price``: The VIF value of 1.978243 is the highest of all the variables, but it is still below 10, indicating that even if there is some collinearity, it is not enough to affect the model. ``bid_price``: The VIF value is only 0.108117, which is a very low VIF value and almost negligible collinearity level.

The cross-validation output shows how the model performed in the 50-fold cross-validation, with scores of 0.9472, 0.9654, 0.9666, 0.9344, and 0.9050, respectively. These scores are high, indicating that the model has better prediction and generalization ability on different data subsets. Differences between scores (especially if the last score is relatively low) may indicate that some parts of the data are less predictable than others, or that the model may not be stable enough in a given situation. In general, the overall performance of the model is satisfactory.

The mean square error (MSE) of the model is very low, which is $4.82e-07$, and the mean absolute error (MAE) is also low, which is 0.000366, indicating that the prediction accuracy of the model is very high and the error is small.

For **random forest** model, Our dataset was massive, with over 5 million rows. Since a random forest is a computation consuming model, we need to sample down to a manageable size. By resampling only 10% of the data randomly, we ensured our model was trained efficiently without compromising the quality of our predictions, as well as maintaining the variability and reducing the bias of the data.

The **features we have extracted**—``scaler_bid_price``, ``scaler_matched_size``, ``scaler_imbalace_size``, ``impute_scale_near_price``, ``impute_scale_far_price``—were pre-processed to ensure optimal model performance. Our preprocessing pipeline included imputation of missing values and scaling to normalize feature magnitude. Then we integrated the Random Forest Regressor into this pipeline, which was designed to run efficiently on our servers by specifying `"n_jobs = -1"`, thereby utilizing all available CPU cores.

To fine-tune our model, we employed a RandomizedSearchCV approach. This method probes a range of hyperparameters, selecting the best combination based on cross-validated performance. Our best model used 36 estimators, with a maximum depth of 10, minimum samples split of 7, and minimum samples leaf of 4.

As for the model evaluation part, we tested our model on test dataset, and the performance metrics we chose are Mean squared Error (MSE) and Mean Absolute Error (MAE), both of which are crucial to understand our regression model's predictive power by quantifying the model's prediction accuracy. As we can see, the **MSE for**

our model is 2.002 and MAE is 0.00027, which is acceptable, indicating that our random forest model is performing well on prediction.

An intriguing aspect of Random Forests is feature importance, providing insight into which variables most strongly influence the prediction. Our model found that 'scaler_bid_price' was the most influential, which is understandable since 'ask price' and 'bid price' are highly correlated, indicating the most competitive sell/buy level in the non-auction book.

The following are residual plot and Q-Q plot in, which provide a visual representation of our model's performance.

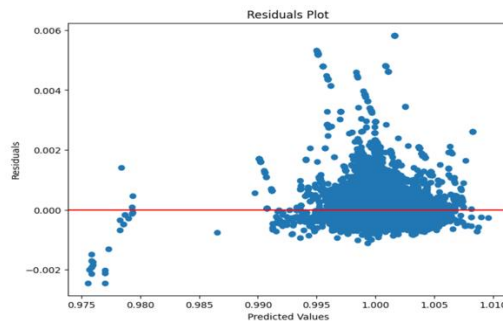


Figure 3: Residual Plot

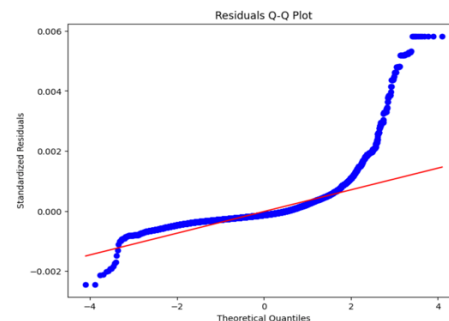


Figure 4: Q-Q Plot

As we can see, the residual plotted against the predicted values shows a random pattern, suggesting a good fit. The Q-Q plot, on the other hand, revealed how closely the residuals followed a normal distribution, indicating the model's assumptions were appropriate for the data.

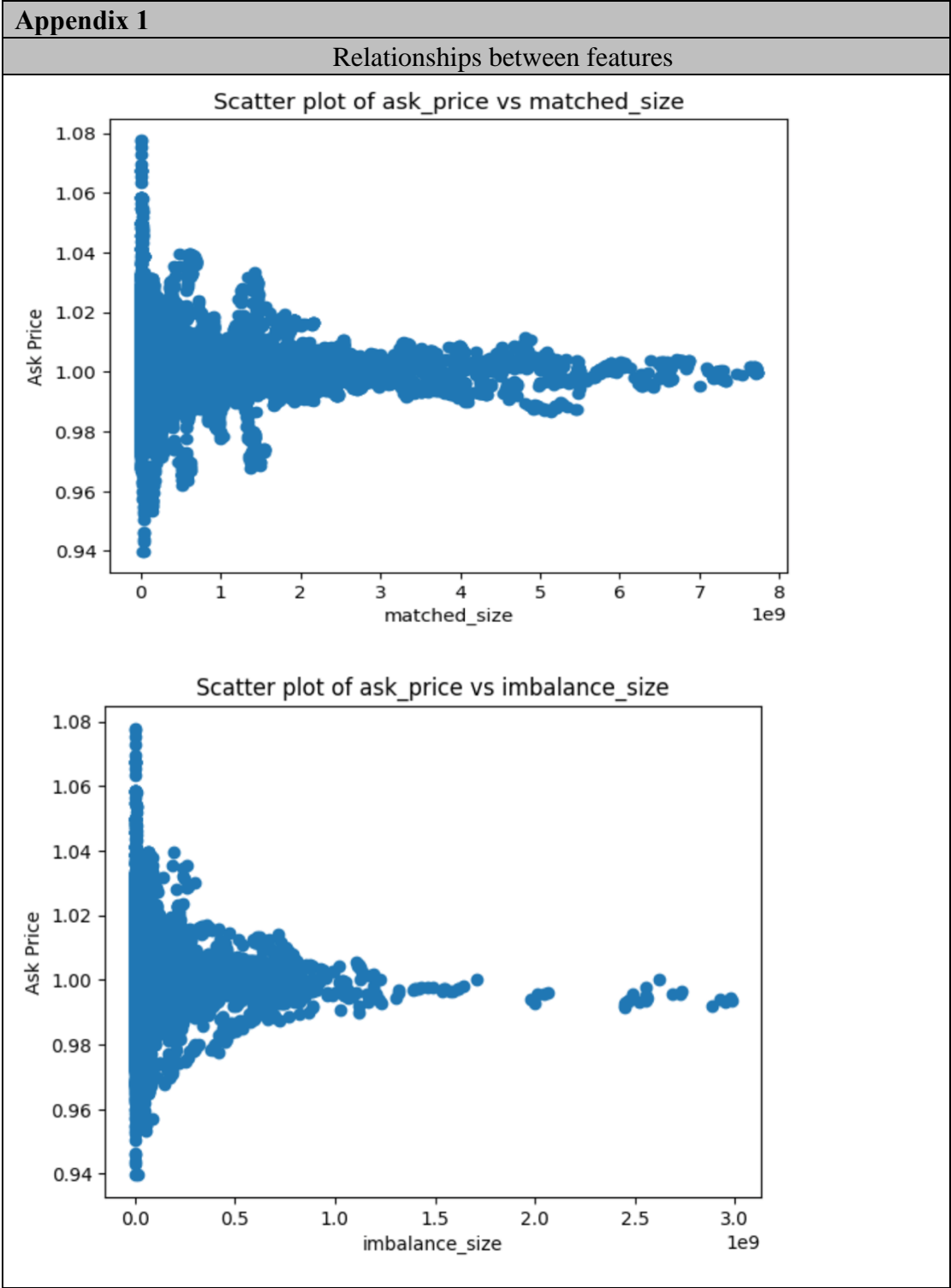
In conclusion, our random forest regression model demonstrates excellent performance and interpretability. By carefully sampling our data, selecting relevant features, and tuning our model, we've achieved a robust tool for predicting the ask price with decent accuracy.

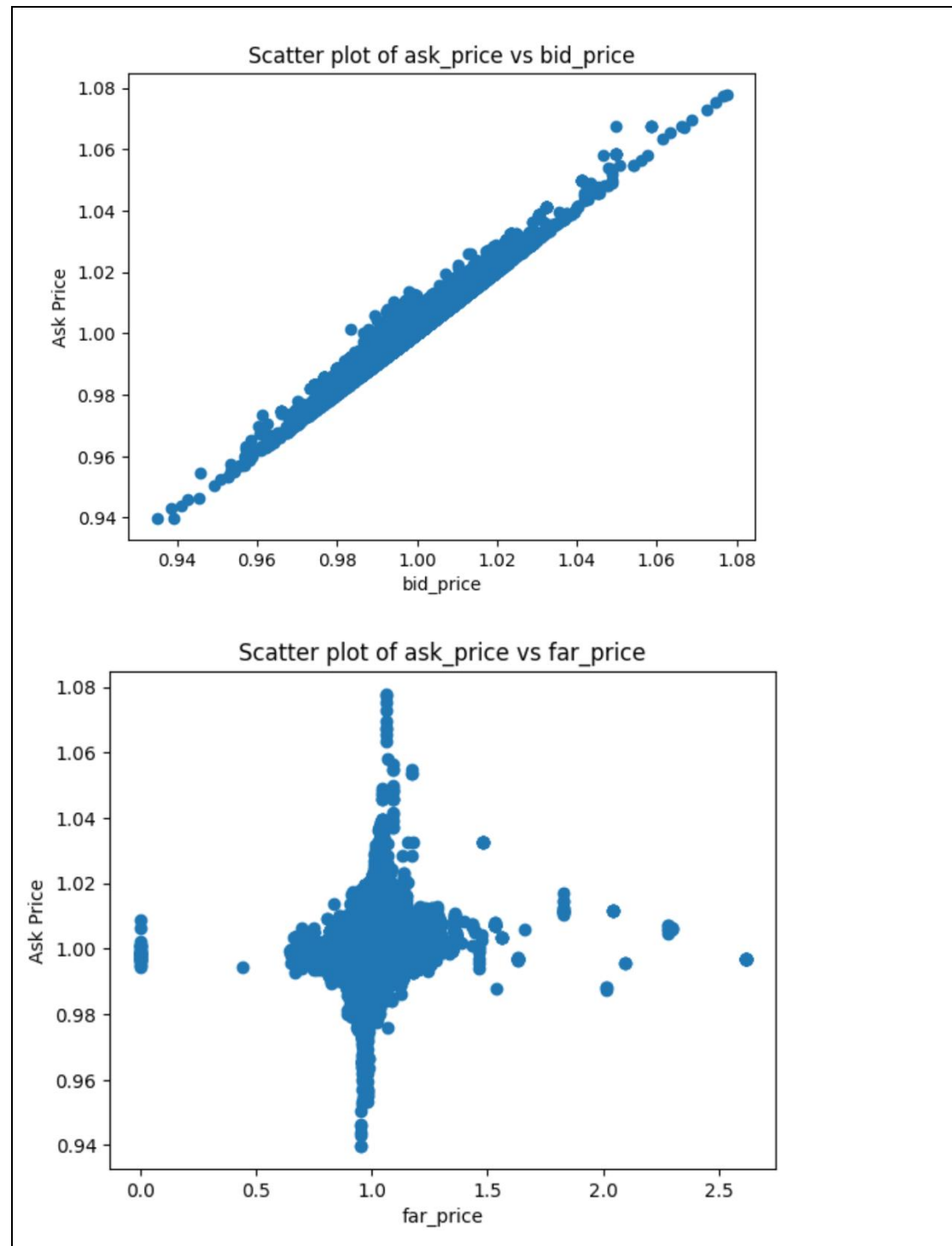
Then, our team firstly calculated the log returns, and then we transformed the mean and standard deviation of these returns back to the mean and standard deviation of the original price ratios. It's commonly assumed that returns follow a normal distribution. This value actually represents that, at the given 95% confidence level, the maximum expected loss in terms of the original price ratio within one day is 1.005785, rather than the typical loss in returns. Therefore, this value seems exceptionally high, suggesting a very high level of risk.


V Conclusion

In conclusion, our research not only successfully predicted the price changes before the stock market closed, but also provided value insights for market participants, helped them better understand the market dynamics, and provided scientific basis for formulating investment strategies. Future work will explore more data sources and advanced models to further improve prediction accuracy and model generalization.

VI Appendix





<div data-bbox="268 212 1161 913"><p>Scatter plot of ask_price vs near_price</p><p>A scatter plot showing the relationship between 'near_price' on the x-axis and 'ask_price' on the y-axis. The x-axis ranges from 0.8 to 1.3 with major ticks every 0.1. The y-axis ranges from 0.94 to 1.08 with major ticks every 0.02. The data points are blue dots, showing a positive correlation. There is a dense cluster of points between near_price 0.9 and 1.1, and ask_price 0.96 and 1.04. There are some outliers at higher near_price values (around 1.2 to 1.3) with ask_price around 1.01 to 1.07.</p></div>	
Appendix 2	
Important code for random forest	
<pre># Define the parameter distribution for RandomForestRegressor with a reduced range and simpler models param_dist = { 'regressor__n_estimators': randint(20, 50), # fewer trees 'regressor__max_depth': [5, 10], # fewer options for max depth 'regressor__min_samples_split': randint(6, 10), # fewer split 'regressor__min_samples_leaf': randint(4, 6) # fewer leaves } # Instantiate the RandomizedSearchCV object with the pipeline and parameter distribution random_search = RandomizedSearchCV(RF_pipeline, param_distributions=param_dist, n_iter=5, # fewer iterations cv=3, scoring='neg_mean_squared_error', n_jobs=-1) # Fit the RandomizedSearchCV object to the data random_search.fit(X_train_sample, y_train_sample) # View the best parameters from the hyperparameter tuning print("Best hyperparameters:", random_search.best_params_) # View the best score found during the search print("Best MSE score from Random Search:", -random_search.best_score_) Best hyperparameters: {'regressor__max_depth': 10, 'regressor__min_samples_leaf': 4, 'regressor__min_samples_split': 7, 'regressor__n_estimators': 34} Best MSE score from Random Search: 2.1953517229735208e-07</pre>	
Appendix 3	
VaR	

```
df = data.copy()
df['log_ask_price_returns'] = np.log(df['ask_price'] / df['ask_price'].shift(1))
df = df.replace([np.inf, -np.inf], np.nan).dropna()

# Calculate the mean and standard deviation (volatility) of returns
mean_returns = np.exp(df['log_ask_price_returns']).mean()
std_dev_returns = np.exp(df['log_ask_price_returns']).std()

# Define the confidence level
confidence_level = 0.95

# Calculate VaR using the variance-covariance method
VaR = norm.ppf(confidence_level, mean_returns, std_dev_returns)
print(f"VaR at {confidence_level * 100}% confidence level: {VaR}")
```

VaR at 95.0% confidence level: 1.0057842213004762