Title: Random Forest Regression Model for Optiver dataset

The dataset used in this project originates from the Optiver trading challenge, focusing on trading activities at the close of financial markets. It comprises various trading-related features, including:

- **stock_id**: Identifier for individual stocks.
- **date_id**: The specific date of trading.
- **seconds_in_bucket**: Time elapsed on a particular trading day.
- **imbalance_size**: The size of order book imbalances.
- **imbalance_buy_sell_flag**: Indicator of the direction of order book imbalance.
- **reference_price**: Reference price for the trades.
- **matched_size**: Size of trades that were matched.
- **far_price**, **near_price**: Different price metrics relevant to trading.
- **bid_price**, **ask_price**: Prices at which buyers are willing to buy and sellers are willing to sell.
- **bid_size**, **ask_size**: Size of orders at the bid and ask prices.
- **wap**: Weighted average price.
- **target**: The target variable to predict, representing a feature of market volatility.
- **time_id**, **row_id**: Additional identifiers.

This dataset presents a snapshot of market conditions and is instrumental in understanding market dynamics at the close of trading.

```
# Step 3: Print the first 5 rows
print(df.head())
```

```
   stock_id  date_id  seconds_in_bucket  imbalance_size  \
0         0        0                  0      3180602.69
1         1        0                  0       166603.91
2         2        0                  0       302879.87
3         3        0                  0     11917682.27
4         4        0                  0       447549.96

   imbalance_buy_sell_flag  reference_price  matched_size  far_price  \
0                        1         0.999812   13380276.64        NaN
1                       -1         0.999896    1642214.25        NaN
2                       -1         0.999561    1819368.03        NaN
3                       -1         1.000171   18389745.62        NaN
4                       -1         0.999532   17860614.95        NaN

   near_price  bid_price  bid_size  ask_price   ask_size  wap    target  \
0         NaN   0.999812  60651.50   1.000026    8493.03  1.0 -3.029704
1         NaN   0.999896   3233.04   1.000660   20605.09  1.0 -5.519986
2         NaN   0.999403  37956.00   1.000298   18995.00  1.0 -8.389950
3         NaN   0.999999   2324.90   1.000214  479032.40  1.0 -4.010200
4         NaN   0.999394  16485.54   1.000016     434.10  1.0 -7.349849

   time_id row_id
0        0  0_0_0
1        0  0_0_1
2        0  0_0_2
3        0  0_0_3
4        0  0_0_4
```

Approach and Initial Implementation

Regression Models Used

The project involved applying various regression models to predict the **target** variable using the given dataset. The models tested included:

- **Linear Regression**: A baseline model for performance comparison.
- **SGD Regressor**: A linear model optimized using stochastic gradient descent.
- **Random Forest Regressor**: An ensemble model known for its robustness and ability to handle non-linear data.
- **LGBM Regressor**: A gradient boosting framework known for its efficiency with large datasets.

Basic Data Handling

The initial steps included basic data preprocessing, such as handling null values. Simple strategies like median and mean imputation were employed to fill missing values in columns such as **far_price**, **near_price**, **bid_price**, **ask_price**, and **wap**. These methods provided a straightforward way to handle missing data, ensuring the dataset was complete for model training. Some columns with null target values were imputed entirely because of absence of ground truth for these cases.

In [51]:
```
# handle missing values

# Drop rows with missing 'target' values
train.dropna(subset=['target'], inplace=True)

from sklearn.impute import SimpleImputer

# Median imputer
median_imputer = SimpleImputer(strategy='median')

# Impute the missing values
df['imbalance_size'] = median_imputer.fit_transform(df[['imbalance_size']])
df['reference_price'] = median_imputer.fit_transform(df[['reference_price']])
df['matched_size'] = median_imputer.fit_transform(df[['matched_size']])

# Mean imputation
train['bid_price'].fillna(df['bid_price'].mean(), inplace=True)
train['ask_price'].fillna(df['ask_price'].mean(), inplace=True)
train['wap'].fillna(df['wap'].mean(), inplace=True)


train['far_price'].fillna(-1, inplace=True)
train['near_price'].fillna(-1, inplace=True)

# Drop rows with nan still values
train.dropna(subset=['imbalance_size'], inplace=True)
```

Performance Metrics and Initial Conclusions

The models were evaluated using metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the coefficient of determination (R^2). From the evaluation:

- The **Linear Regression** model showed a wide dispersion in predictions, indicating poor performance, especially for extreme values.
- The **SGD Regressor** showed atypical results, possibly due to issues in convergence or data scaling, as indicated by its performance metrics.
- The **Random Forest Regressor** displayed a better alignment of predictions, with more consistent residuals, suggesting superior performance.
- The **LGBM Regressor** demonstrated reasonable prediction accuracy but with room for improvement.

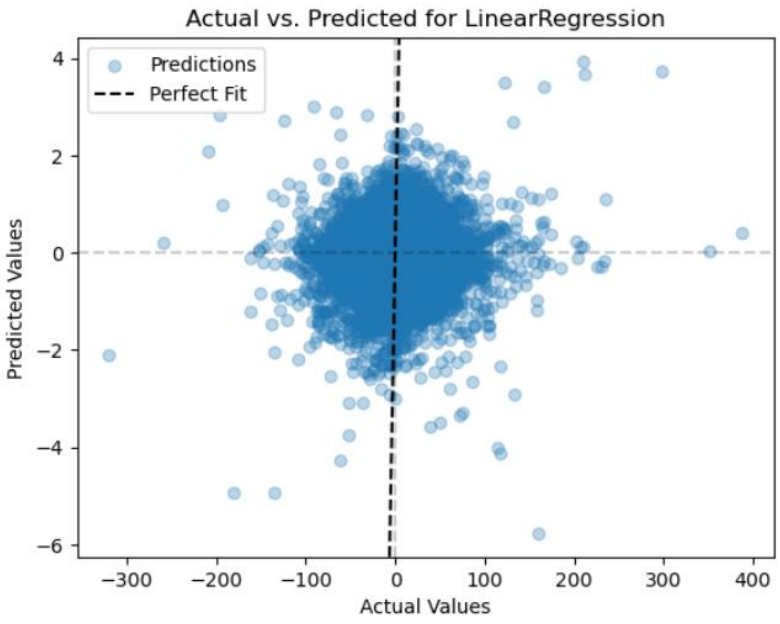
I used scatter plot graph and residual graph to visualize the predictions with the disparity from truth.

Linear Regression Model:

The scatter plot shows a wide dispersion of points with a concentration around the line y = 0, which indicates poor prediction for extreme values.

The residual plot reinforces this finding with a large spread of residuals that don't cluster around zero, indicating inconsistent prediction errors across the range of predictions.

The metrics show a high MAE and RMSE, and an R^2 value close to zero, suggesting the model is not adequately capturing the underlying data structure.



LinearRegression - MAE: 6.403237227800439, RMSE: 9.434295336233196, R^2: 0.0007353488459372892

In a scatter plot comparing actual versus predicted values for a regression model:

Each blue dot represents a single prediction by the model. The position of the dot on the X-axis indicates the actual value from the test dataset, while the position on the Y-axis indicates the predicted value generated by the model for that same data point.

The reference line is typically a straight line where the predicted values are equal to the actual values. This is known as the line of perfect prediction. In the graph, it's usually drawn as a dashed line diagonally across the plot.

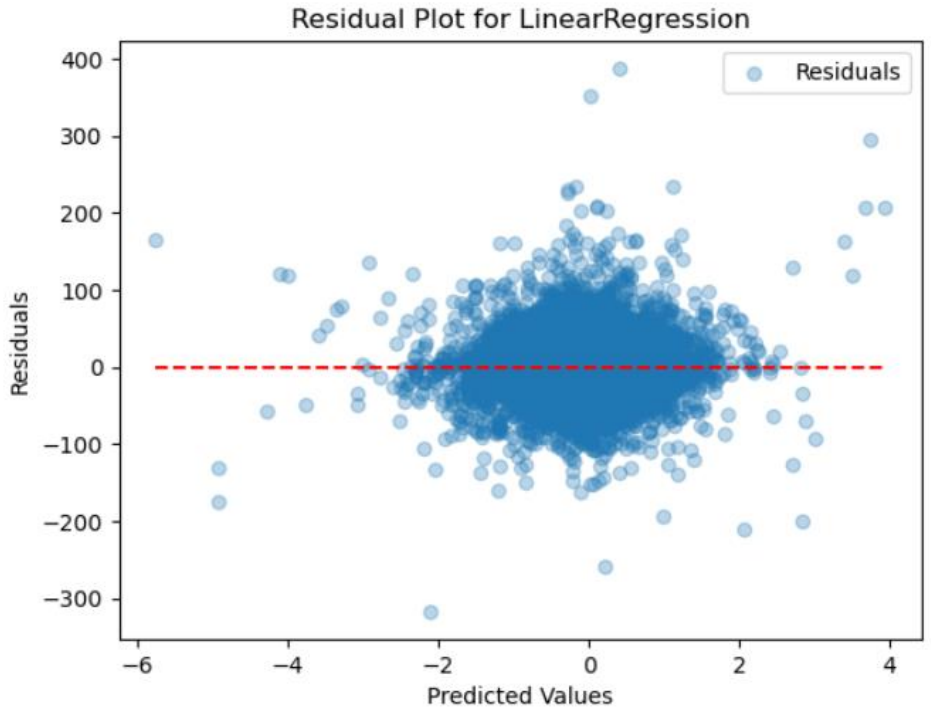
Interpreting the Plot

Dots on the Line: If a blue dot lies exactly on the reference line, it means the model's prediction for that data point was perfect — the predicted value equals the actual value.

Dots above the Line: If a dot is above the reference line, the model overestimated the value. The vertical distance from the dot to the line represents the magnitude of the overestimation. Conversely, if a dot is below the reference line, the model underestimated the value, with the vertical distance indicating the error magnitude.

Ideal Scenario: All dots would lie directly on the reference line, indicating that the model predicts the actual value perfectly every time. However, this is almost never the case in real-world scenarios due to the complexity and noise inherent in the data.

Conclusion: This graph is a visual tool to quickly assess the accuracy of a regression model's predictions. The closer the dots are to the reference line, the more accurate the predictions. Conversely, if many dots are far from the line, it suggests the model's predictions are often inaccurate. The overall pattern can also suggest whether there are systematic errors in the model's predictions, such as consistently over or underestimating across certain ranges of values.



The residual plot is used to assess the performance of a regression model. It plots the difference between the actual and predicted values (residuals) against the predicted values. Here's how to interpret it:

Residuals: The vertical axis shows the residuals, which are the differences between the actual values (y_test) and the predicted values (y_pred). A residual is positive if the prediction is lower than the actual value (underprediction) and negative if it is higher (overprediction).

Predicted Values: The horizontal axis represents the predicted values. It helps to determine if there are any patterns in the residuals across different predicted values.

Horizontal Line at Zero: The dashed horizontal line at zero represents no error. If a point lies on this line, it means the prediction was exactly correct for that instance.

Significance of Patterns in the Residual Plot:

Random Distribution: Ideally, the residuals should be randomly distributed around the horizontal line, indicating that the model's errors are distributed randomly, and the model predicts equally well across all values.

Systematic Patterns: If there are systematic patterns, such as a curve or a distinct shape, this suggests that the model is not capturing some aspect of the data's structure. This could be due to the model being too simple (underfitting), incorrect assumptions about the data, or the need for feature transformation.

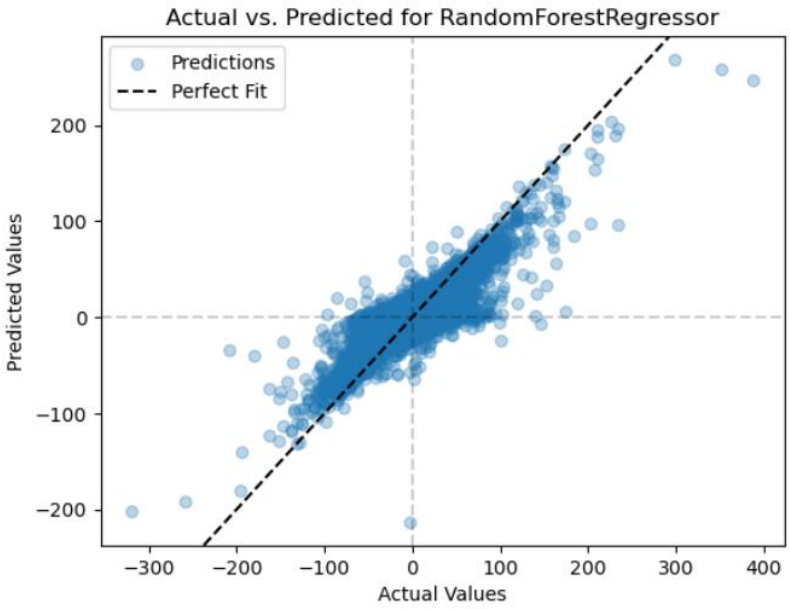
Homoscedasticity: If the spread of the residuals is consistent across the range of predicted values, the model is homoscedastic. This is a good indication that the model is stable across different levels of prediction.

Heteroscedasticity: If the residuals fan out or in (variance changes with the level of prediction), this indicates heteroscedasticity, suggesting that the model's performance varies at different levels of prediction. This can be a sign that the model may benefit from transformations of the dependent variable or the addition of features that can explain the variance at different levels.

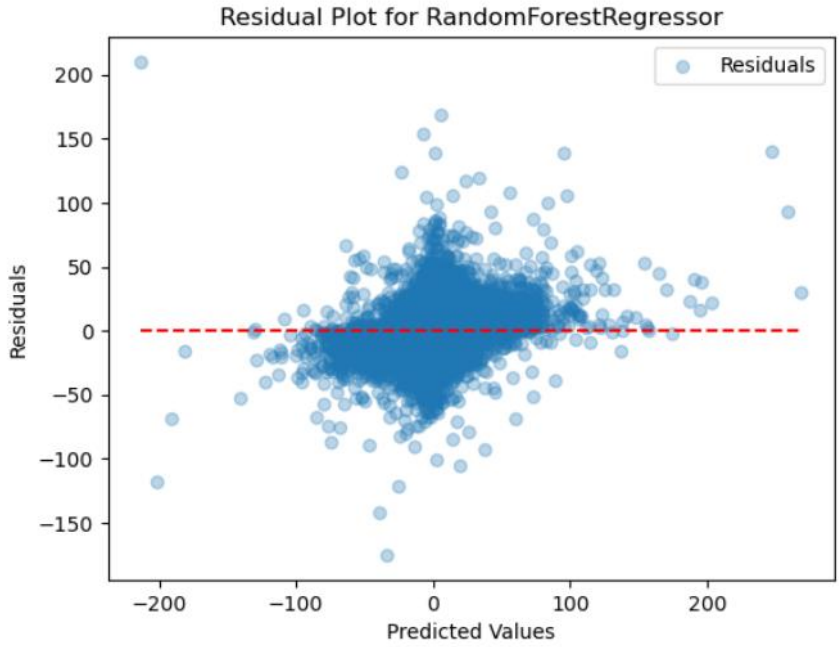
Outliers: Points that are far from the horizontal line are outliers. They can have a disproportionate impact on the model and may warrant further investigation.

Random Forest Regressor:

The scatter plot is more aligned with the y = x line, especially for values closer to zero, indicating better performance than the Linear Regression model.
The residual plot shows residuals more closely clustered around zero, implying more consistent predictions.
The metrics indicate a much lower MAE and RMSE, and an R^2 value substantially greater than zero, signifying a decent model fit.



RandomForestRegressor - MAE: 2.410769798979481, RMSE: 4.062756941843704, R^2: 0.814688115898921

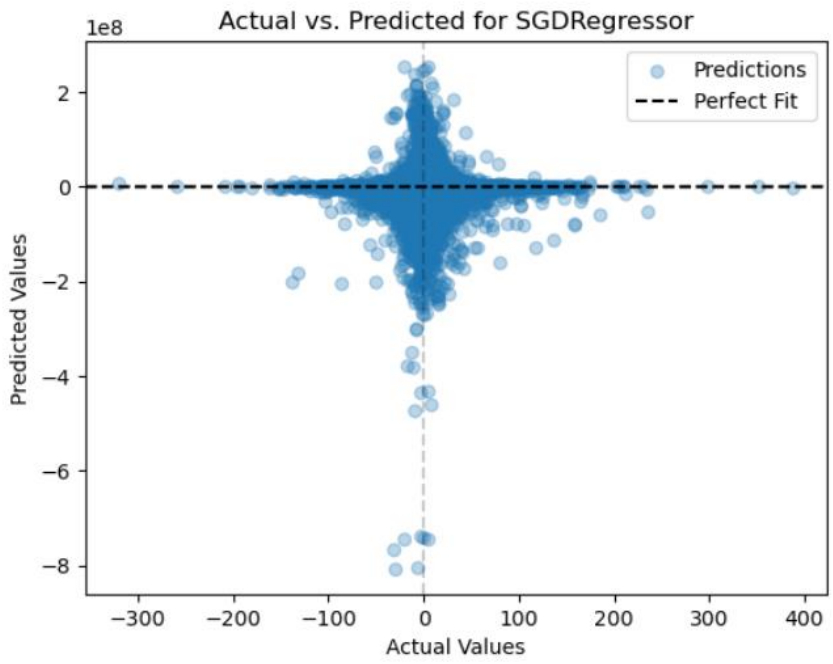


SGD Regressor:

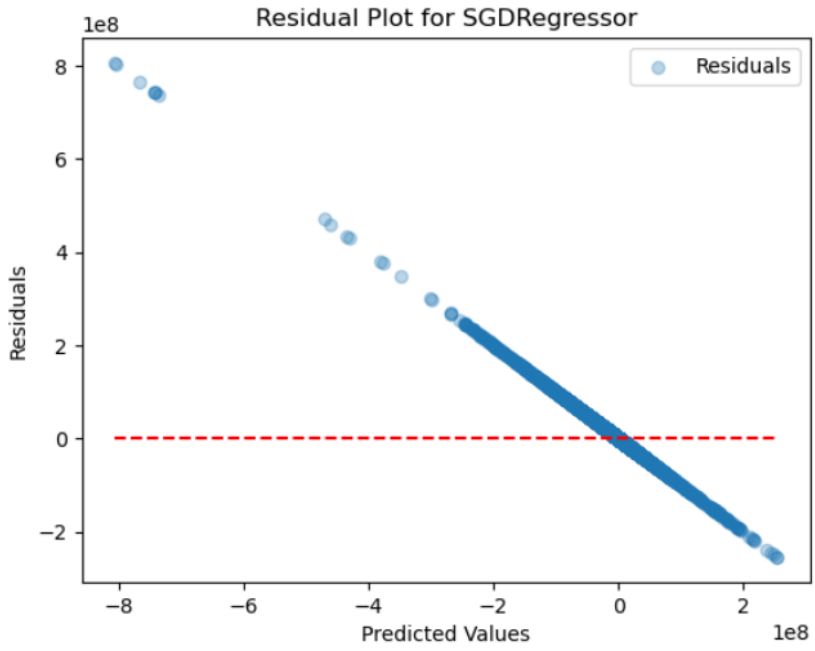
The scatter plot for the SGD Regressor is peculiar, with predictions forming a vertical line at x = 0, which suggests possible issues with the model's convergence or data scaling.

The residual plot for the SGD Regressor is not typical and suggests major issues, with a clear pattern in the residuals.

The metrics are extremely poor, with very high MAE and RMSE values, and a negative R^2, which usually indicates that the model is worse than a simple mean model.



SGDRegressor - MAE: 1764361.796215795, RMSE: 6237531.726917179, R^2: -436804521128.4966

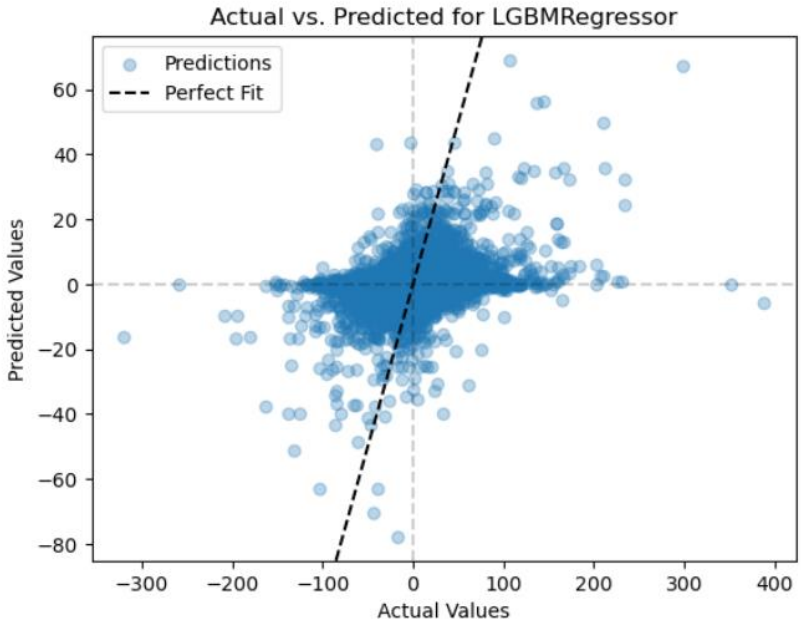


LGBM Regressor:

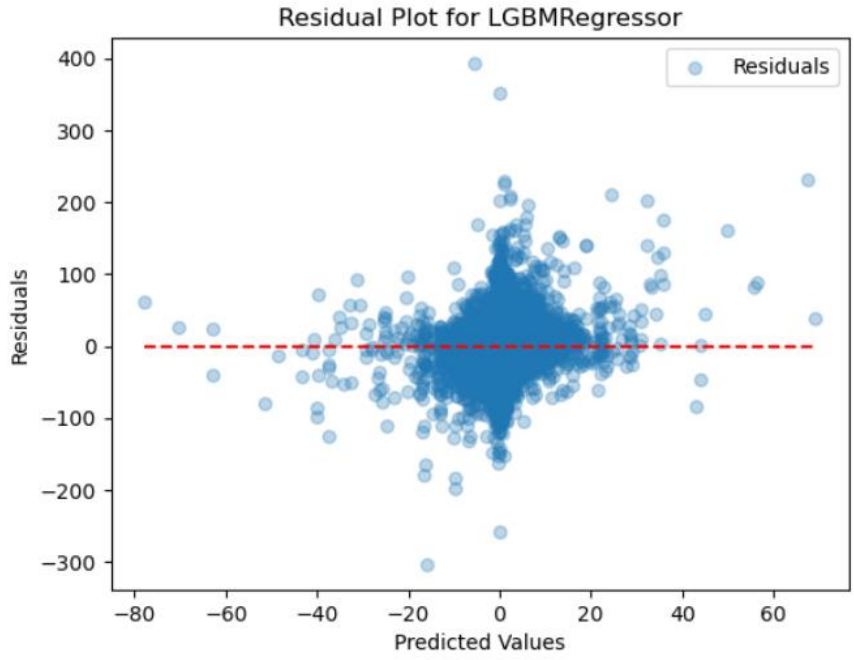
The scatter plot for the LGBM Regressor shows a reasonable alignment with the y = x line but with some dispersion.

The residual plot indicates a concentration of residuals around zero, although there is some spread which suggests room for improvement.

The metrics show a moderate MAE and RMSE, and an R^2 value that is positive but low, suggesting the model has some predictive power but could potentially be improved further.



LGBMRegressor - MAE: 6.372533132061587, RMSE: 9.35763089796269, R^2: 0.01690969921035501



In conclusion, the **Random Forest Regressor** is performing relatively well compared to the other models, with more accurate predictions and consistent residuals. The Linear Regression model and the LGBM Regressor show signs of predictive ability but with more significant errors and room for improvement. The SGD Regressor's performance is highly questionable and likely indicates a fundamental issue with the parameters or data handling.

Advancing with Random Forest: Enhanced Data Handling and Feature Engineering

As the project progressed, the Random Forest Regressor was further changed to improve its predictive accuracy and interpretability.

Handling Null Values with Linear Interpolation

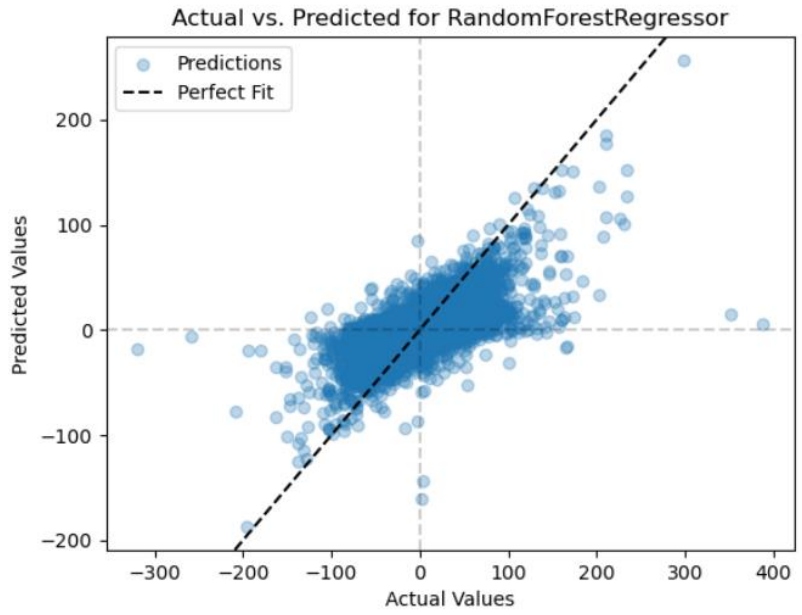
Null values present a common challenge in data analysis, particularly in financial time series where they can signify missing trades or quotes. To address this, linear interpolation was employed for the Random Forest model. This method assumes that the missing value can be linearly approximated using the available adjacent values, which is a reasonable assumption in financial markets where prices tend to change incrementally.

Introduction of New Features

With the null values addressed, attention turned to enriching the dataset with new features. Three new features were introduced:

- **Bid-Ask Spread**: This feature (**df['bid_ask_spread']**) represents the difference between the ask and bid prices. It's an essential indicator of market liquidity and transaction costs.
- **Median Imbalance Size by Stock ID**: To capture the typical order imbalance for each stock (**df['imbalance_size']**), the median value was imputed group-wise. This reflects the prevailing market sentiment towards the buying or selling pressure for each stock.
- **Cyclical Time Features**: Recognizing the intraday cyclical nature of trading, sine and cosine transformations (**df['sin_time']**, **df['cos_time']**) were applied to the seconds within a trading bucket. This helps the model capture patterns related to specific times during the trading day.

The addition of these features was driven by the underlying financial intuition and their potential impact on the target variable, which in this context is likely a proxy for short-term volatility or price movement.




```
RandomForestRegressor - MAE: 4.842499732104804, RMSE: 7.277327245285582, R²: 0.40542670549228266
```

Evaluation of Feature Importance

After retraining the Random Forest model with the new features, the importance of each feature was evaluated. The results underscored the significant role of **matched_size** and **reference_price**, which aligns with financial theory — larger transactions and deviations from the reference price can both be indicative of market-moving events.

```
In [21]: # Get feature importances
         importances = rf_model.feature_importances_

         # Summarize feature importances
         for i, feat in enumerate(features):
             print(f'Feature: {feat}, Importance: {importances[i]}')
```

```
Feature: stock_id, Importance: 0.11021939740688247
Feature: date_id, Importance: 0.13755947517706246
Feature: seconds_in_bucket, Importance: 0.03859653743247226
Feature: imbalance_size, Importance: 0.13041347739037581
Feature: imbalance_buy_sell_flag, Importance: 0.0029316180269366014
Feature: reference_price, Importance: 0.17349238868771077
Feature: matched_size, Importance: 0.17737390593596108
Feature: bid_ask_spread, Importance: 0.1565453956029102
Feature: sin_time, Importance: 0.035029998090269396
Feature: cos_time, Importance: 0.03783780624941892
```

Further Feature Engineering and Model Refinement

Building upon the insights from feature importance, further feature engineering was undertaken:

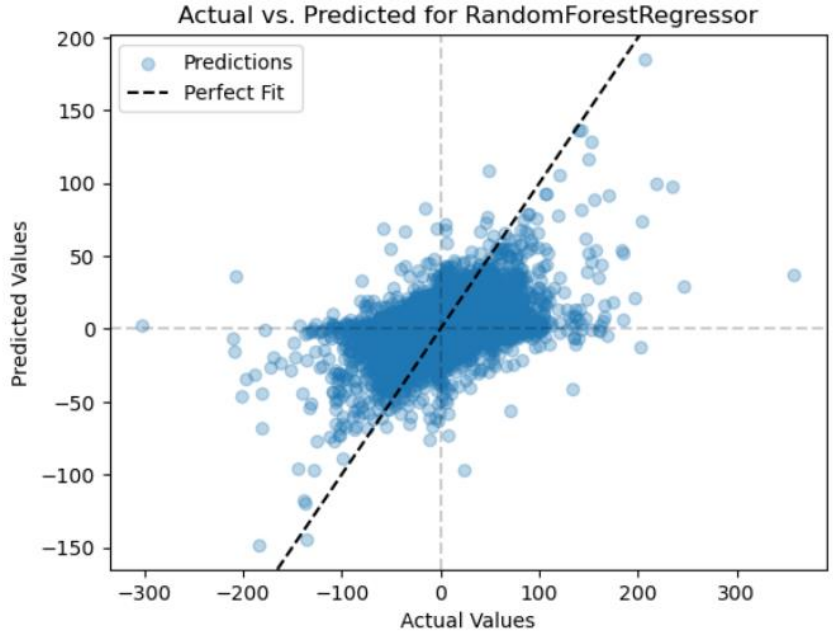
- **WAP Volatility**: A rolling standard deviation of the Weighted Average Price (WAP) over a 10-time bucket window was computed (**df['wap_volatility_10']**). Volatility is a cornerstone of financial metrics, often associated with risk and price efficiency.

- **Weighted Price Movement**: The product of **matched_size** and the change in WAP (**df['weighted_price_movement']**) was calculated. This feature aims to quantify the influence of trade size on price dynamics, potentially capturing significant market events.
- **Order Book Imbalance Ratio**: The ratio of order book imbalance size to matched size (**df['imbalance_ratio']**) was introduced. This ratio can signal which side of the market is more aggressive, thereby influencing price movement.

These newly engineered features are grounded in financial theory and are designed to provide the Random Forest model with nuanced inputs that more accurately reflect market mechanics.

Model Training with Updated Features

The model was then trained with an updated list of features, which included both the original and newly engineered ones. This comprehensive set of features aims to capture a wide array of market forces, from liquidity to sentiment, and from market timing to transaction impact.



RandomForestRegressor - MAE: 6.01203931545928, RMSE: 8.780120928259809, R^2: 0.12823144713053636

Conclusion: Assessing Model Performance with Feature Engineering

Despite the extensive feature engineering efforts and advanced data preprocessing techniques, the recent metrics indicate a deterioration in model performance. The latest iteration of the Random Forest Regressor, which incorporated newly engineered features such as **wap_volatility_10**, **weighted_price_movement**, and **imbalance_ratio**, yielded suboptimal results compared to the previous model configurations.

Evaluation of Enhanced Model

The updated model, which was expected to better capture the complexities of the financial market and provide improved predictions, resulted in the following metrics:

- Mean Absolute Error (MAE): 6.01203931545928

- Root Mean Squared Error (RMSE): 8.78012928259889
- R-Squared (R^2): 0.12823144713053636

These scores reflect a regression in predictive accuracy and explainability, contrary to the anticipated outcome. The R^2 value, a measure of how well the observed outcomes is replicated by the model, is comparatively low.

Return to Simpler Model Configurations

Considering these findings, the report concludes with a recommendation to revert to the initial Random Forest model, which utilized a straightforward approach to feature selection and median imputation for handling null values. The simpler model not only achieved better metrics but also provided a more interpretable framework for understanding the predictions.