# Multi-Model Analysis: Predicting Monthly Returns for S&P 500 Stocks

Hrishikesh Balakrishnan
Minaal Sheikh
Aditya Pawar

## Introduction

Predicting stock returns is a notoriously challenging task due to the complex and dynamic nature of financial markets. However, advancements in machine learning offer powerful tools to explore this challenge. This project leverages Python for automation and analysis, aiming to identify effective approaches for predicting monthly returns within the S&P 500 market.

The report details a multi-step approach:

- **Data Acquisition**: We will automate the process of scraping relevant data. This includes acquiring the list of S&P 500 companies from Wikipedia and downloading daily historical stock prices for each company from a source like Yahoo Finance.

- **Data Processing**: The collected data will be transformed to focus on monthly returns for the period 2010-2023. This timeframe provides a robust dataset for model training and evaluation.

- **Machine Learning Model Exploration**: We will train and compare the performance of several machine learning models for predicting next month's return. These models include linear regression with Ridge regularization for stability, random forest regressor for capturing complex relationships, MLPRegressor (a type of neural network) for non-linear patterns, and decision tree regressor for interpretability.

- **Model Evaluation**: By analyzing model performance metrics, we will identify the most effective approach for predicting monthly returns within the S&P 500.

To ensure comprehensive analysis, data will be extracted from diverse sources like Yahoo Finance, Wikipedia, and Edgar. This multifaceted approach allows for a more holistic understanding of the factors influencing stock returns.

## Data Sources

The project will leverage data from the following sources:

**S&P 500 Company List**: Acquired from Wikipedia, this comprehensive list provides the universe of companies for which we will collect stock price data.

**Daily Stock Prices**: Downloaded from Yahoo Finance. This robust source offers historical daily closing prices for a vast number of stocks.

**Edgar**: The Edgar database (Electronic Data Gathering, Analysis, and Retrieval system) maintained by the SEC (Securities and Exchange Commission) will be used to supplement the analysis. It will provide additional financial data on the companies.

The collected data will be transformed into a set of features that represent various aspects of a stock's performance and health.

### Calculated Features:

1. **Average Daily Returns Over A Month**: This metric captures the average daily price movement for a month.
2. **Standard Deviation Of Daily Returns Over A Month**: This indicates the volatility of the stock's price within a month.
3. **Sharpe Ratio**: This metric helps assess risk-adjusted return, considering both return and volatility.
4. **Skewness Of Daily Returns For A Month**: This measures the asymmetry of the daily return distribution, indicating if returns are skewed towards positive or negative movements.
5. **Monthly Return**: This represents the overall change in price for a month.
6. Cumulative Returns Over The Past 3, 6, 9, And 12 Months: This captures the total price appreciation over these timeframes.
7. **Stock's "Close" Price At The End Of A Month**: This is the closing price on the last trading day of the month.
8. **Average Of Dollar Trade Volume In A Month**: This reflects the average daily trading activity for the stock in terms of dollar volume.
9. **Standard Deviations Of Dollar Trade Volume In A Month**: This indicates the variability of the trading volume throughout the month.

**Additional Features from Edgar:**

- **Profitability**: Metrics like return on equity (ROE) or profit margin.
- **Liquidity**: Current ratio or quick ratio.
- **Activity**: Inventory turnover or receivable turnover.
- **Financial Leverage**: Debt-to-equity ratio.
- **Market Valuation**: Price-to-earnings (P/E) ratio, retrieved from financial news sources.

## Data Preprocessing

This section details the extensive data pre-processing steps undertaken prior to feeding the data into our machine learning models for stock return prediction. The process can be broken down into five key stages:

**1. Data Collection and Feature Extraction:**

- Source Identification: We began by acquiring a list of S&P 500 companies from a reliable source like Wikipedia.
- Historical Price Extraction: Next, historical stock price data for each company was obtained, potentially from a source like Edgar and Yahoo finance.

**2. Data Transformation:**

- Datetime Conversion: Dates within the data were converted into a consistent datetime format for efficient manipulation and analysis.
- Date Decomposition: Further processing involved splitting the dates into separate components for year, month, and day.

**3. Feature Engineering:**

Informative Feature Creation: From the daily data, several features with potential predictive power were derived, including:

- Average daily return over a month
- Standard deviation of daily returns
- Sharpe Ratio approximation (risk-adjusted return metric)
- Skewness of daily returns (measure of asymmetry)
- Monthly return
- Cumulative returns over various timeframes (3, 6, 9, and 12 months)
- End-of-month closing price
- Average and standard deviations of monthly dollar trade volume

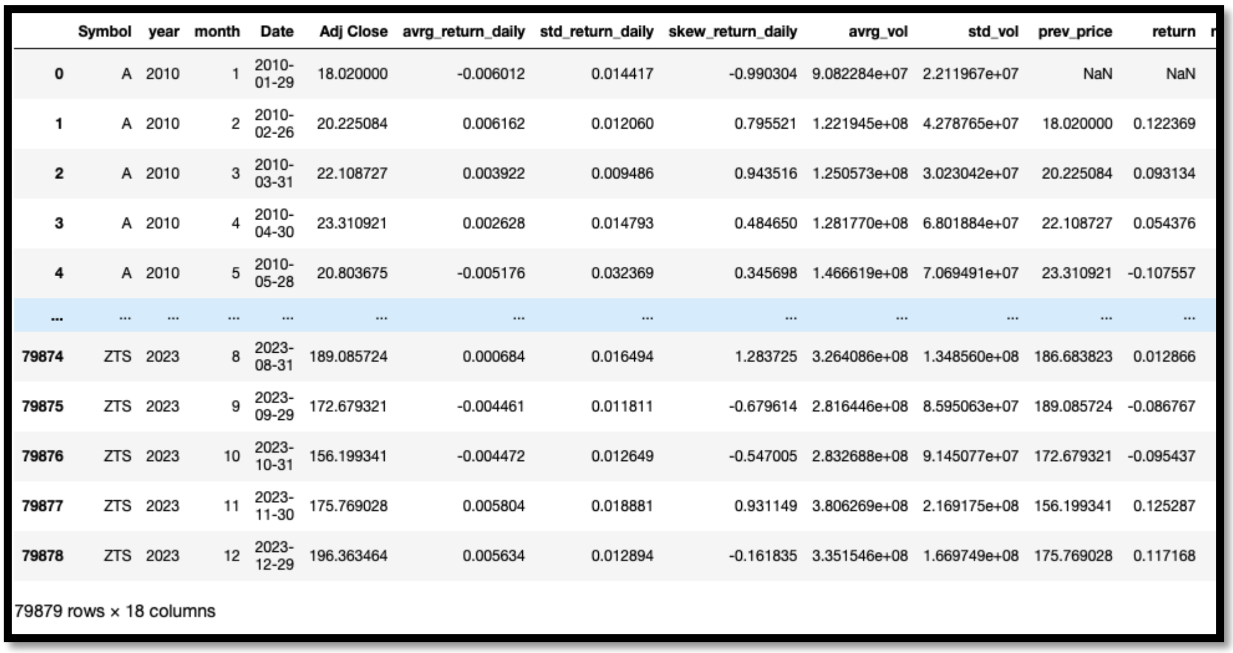
**4. Data Cleaning and Preparation:**

- Data Organization: The data was meticulously organized by sorting it based on a combination of stock symbol, year, month, and day.
- Timeframe Filtering: The data was then filtered to focus on the specific period of interest, from 2010 to 2023.
- Dimensionality Reduction: To achieve a monthly view and potentially reduce computational complexity, we employed dimensionality reduction techniques by grouping data by symbol, year, and month.
- Missing Value Imputation: Missing values within the data were addressed using a mean imputation strategy.

**5. Model Training Preparation:**

- Training-Testing Split: The pre-processed data was strategically divided into training (80%) and testing (20%) sets. A random state was used to ensure reproducibility of the split.
- Cross-Validation: To enhance model performance and prevent overfitting, 5-fold cross-validation was implemented within the training data using KFold.
- Normalization: Finally, the data was normalized using StandardScaler. This step ensures all features are on a similar scale, leading to improved model convergence during training.

By implementing this comprehensive data pre-processing pipeline, we ensured our machine learning models received high-quality data that is clean, consistent, and well-structured. This, in turn, contributes to optimal model performance in predicting future monthly stock returns.



| | Symbol | year | month | Date | Adj Close | avrg_return_daily | std_return_daily | skew_return_daily | avrg_vol | std_vol | prev_price | return |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | A | 2010 | 1 | 2010-01-29 | 18.020000 | -0.006012 | 0.014417 | -0.990304 | 9.082284e+07 | 2.211967e+07 | NaN | NaN |
| **1** | A | 2010 | 2 | 2010-02-26 | 20.225084 | 0.006162 | 0.012060 | 0.795521 | 1.221945e+08 | 4.278765e+07 | 18.020000 | 0.122369 |
| **2** | A | 2010 | 3 | 2010-03-31 | 22.108727 | 0.003922 | 0.009486 | 0.943516 | 1.250573e+08 | 3.023042e+07 | 20.225084 | 0.093134 |
| **3** | A | 2010 | 4 | 2010-04-30 | 23.310921 | 0.002628 | 0.014793 | 0.484650 | 1.281770e+08 | 6.801884e+07 | 22.108727 | 0.054376 |
| **4** | A | 2010 | 5 | 2010-05-28 | 20.803675 | -0.005176 | 0.032369 | 0.345698 | 1.466619e+08 | 7.069491e+07 | 23.310921 | -0.107557 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **79874** | ZTS | 2023 | 8 | 2023-08-31 | 189.085724 | 0.000684 | 0.016494 | 1.283725 | 3.264086e+08 | 1.348560e+08 | 186.683823 | 0.012866 |
| **79875** | ZTS | 2023 | 9 | 2023-09-29 | 172.679321 | -0.004461 | 0.011811 | -0.679614 | 2.816446e+08 | 8.595063e+07 | 189.085724 | -0.086767 |
| **79876** | ZTS | 2023 | 10 | 2023-10-31 | 156.199341 | -0.004472 | 0.012649 | -0.547005 | 2.832688e+08 | 9.145077e+07 | 172.679321 | -0.095437 |
| **79877** | ZTS | 2023 | 11 | 2023-11-30 | 175.769028 | 0.005804 | 0.018881 | 0.931149 | 3.806269e+08 | 2.169175e+08 | 156.199341 | 0.125287 |
| **79878** | ZTS | 2023 | 12 | 2023-12-29 | 196.363464 | 0.005634 | 0.012894 | -0.161835 | 3.351546e+08 | 1.669749e+08 | 175.769028 | 0.117168 |

79879 rows × 18 columns

## Linear Regression with Ridge Regularization

**Ridge Regression with L2 Regularization**:

Linear regression establishes a linear relationship between a dependent variable (stock return in this case) and one or more independent variables (features engineered from historical data). However, it can be susceptible to overfitting, particularly when dealing with a high number of features. Ridge regression addresses this by introducing a penalty term that discourages large coefficient values in the model. This essentially reduces the model's complexity and helps to prevent it from fitting too closely to the training data at the expense of generalizability.

**Model Performance Analysis:**

The provided output presents the results of the ridge regression model applied to predict monthly stock returns. Here's a breakdown of the key metrics:

- **Best Alpha**: This value (100 in this case) represents the optimal regularization strength chosen for the model through a process of hyperparameter tuning.
- **Train R-squared (0.0164) vs Test R-squared (0.0089):** R-squared is a statistical measure that reflects the proportion of variance in the dependent variable explained by the independent variables. While the training R-squared suggests a weak positive correlation between the features and the predicted returns, the test R-squared is significantly lower. This indicates that the model performs better on the data it trained on but fails to generalize well to unseen test data. This is a classic sign of overfitting.
- **Train MSE (0.0071) vs Test MSE (0.0072):** Mean squared error (MSE) measures the average squared difference between the predicted and actual returns. Similar to R-squared, the training and test MSE values are very close. While the training MSE suggests a decent fit, the lack of improvement on unseen test data reinforces the overfitting concern.

**Overall Analysis**:

Based on the low test R-squared and similar training and test MSE values, the ridge regression model in this instance exhibits signs of overfitting. While it might capture some patterns in the training data, it likely fails to generalize well to new data, rendering its effectiveness in predicting future monthly stock returns questionable. It's essential to compare these results with other models, such as Lasso regression or more complex algorithms like neural networks, to determine the most suitable approach for this prediction task.

## Decision Tree Regressor

**Hyperparameter Tuning with GridSearchCV:**

Decision trees have several hyperparameters that can significantly impact their performance. GridSearchCV (or a similar technique) is a method commonly used to explore various combinations of these hyperparameters and identify the configuration that yields the best results on a validation set. In this case, the tuning process likely optimized the following parameters based on the provided values:

- **max_leaf_nodes (10):** This parameter controls the maximum depth of the tree, preventing it from becoming overly complex. By setting it to 10, the tuning process restricted the tree to a maximum depth of 10, potentially reducing the risk of overfitting.
- **min_samples_leaf (1):** This parameter specifies the minimum number of samples required at a leaf node, ensuring some level of stability in the model. Setting it to 1 ensures that each leaf node contains at least one data point, preventing empty leaves and potentially improving model stability.
- **min_samples_split (2):** This parameter defines the minimum number of samples required to split an internal node, helping to prevent overfitting. Setting it to 2 means that a node will only be split if it contains at least two data points, encouraging the creation of more informative splits.

**Model Performance:**

The provided results show the performance of the tuned decision tree model:

- **Training R-squared (0.0391) vs Test R-squared (0.0126):** While the training R-squared indicates a slightly stronger positive correlation compared to the ridge regression model, the test R-squared remains low. This suggests some improvement in generalizability but still highlights limitations in capturing the underlying relationships for prediction.
- **Training MSE (0.0069) vs Test MSE (0.0071):** Similar to the R-squared values, the training and test MSE show a slight improvement compared to the previous model. While the training MSE suggests a better fit, the small difference on unseen test data indicates a potential for overfitting.

**Analysis:**

The hyperparameter tuning process, focusing on the specific parameters mentioned (max_leaf_nodes, min_samples_leaf, and min_samples_split), appears to have yielded some improvement in the decision tree's generalizability compared to the ridge regression model. However, the test R-squared and MSE values remain relatively low, suggesting there's still room for improvement.

## Random Forest Regressor

Random forest regression is a powerful ensemble learning technique well-suited for tasks like predicting continuous values, such as monthly stock returns. It combines the predictions from multiple decision trees to create a more robust and generalizable model compared to a single decision tree.

**Hyperparameter Tuning Results:**

The tuning process, limited to 10 iterations due to computational constraints, identified the following optimal hyperparameter configuration:

- **max_depth = None:** This allows the individual trees within the forest to grow as deep as possible without restrictions, potentially capturing complex relationships in the data.
- **max_features = 'sqrt':** This setting uses the square root of the total number of features when selecting random subsets of features at each split within a tree. This helps to reduce overfitting and promotes diversity among the trees.
- **min_samples_leaf = 2:** This parameter ensures that each leaf node in the trees contains at least two data points, improving the stability of the model.
- **min_samples_split = 4:** This parameter requires a minimum of four data points to split an internal node in a tree, encouraging the creation of more informative splits within the forest.
- **n_estimators = 112:** The tuning process identified 112 as the optimal number of trees to include in the random forest. This ensemble size likely provides a good balance between model complexity and generalizability.

**Performance Analysis:**

The provided results show a mixed picture of the model's performance:

- **Training R-squared (0.7681):** This high value indicates a strong positive correlation between the features and predicted returns on the training data.
- **Test R-squared (0.0373):** However, the test R-squared is significantly lower, suggesting significant overfitting. While the model captures patterns in the training data, it fails to generalize well to unseen data.
- **Train MSE (0.0016) vs Test MSE (0.0070):** The training MSE is considerably lower than the test MSE, further highlighting the overfitting issue. The model performs well on the training data but struggles on unseen data.

While the random forest regressor achieved a low MSE on the test set, indicating a good fit for the training data, the low test R-squared raises concerns about overfitting. The limited number of tuning iterations and computational constraints restrict the generalizability conclusions. Further exploration with more iterations, increased resources, and alternative tuning techniques could yield a more robust model with improved prediction capabilities for unseen data.

## Neural Network – MLP Regressor

**Hyperparameter Tuning:**

The tuning process identified a specific configuration that optimized the model's learning and complexity:

- **Activation: 'tanh':** This activation function, applied in the hidden layer, introduces non-linearity into the network, allowing it to capture more complex relationships in the data compared to a linear model.
- **Alpha: 0.004781:** This hyperparameter controls the L-BFGS solver's regularization strength, helping to prevent overfitting by penalizing overly complex models.
- **Hidden Layer Sizes: (50**,): This setting indicates a single hidden layer with 50 neurons. This configuration balances model capacity with potential overfitting risks.
- **Learning Rate: 'adaptive':** This refers to an adaptive optimizer that adjusts the learning rate throughout training. This can help the model converge faster and potentially achieve better performance.
- **Solver: 'lbfgs':** The L-BFGS algorithm is used to optimize the neural network's weights and biases during training.

**Performance Analysis:**

- **Training R-squared (0.0379):** This value suggests a weak positive correlation between the features and predicted returns on the training data.
- **Test R-squared (0.0186):** While lower than the training R-squared, it indicates some ability to generalize to unseen data.
- **Train MSE (around 0.007):** The low training MSE suggests a good fit for the training data.
- **Test MSE (around 0.007):** The test MSE is similar to the training MSE, further indicating that the model performs reasonably well on unseen data.

While the neural network regressor achieved a low Mean Squared Error (MSE) on the test data, indicating a good fit for unseen data, the analysis needs to address the limitations imposed by restricted training iterations and computational resources. Here's a breakdown:

Positives: The test MSE suggests the model can generalize to some extent and make reasonable predictions on new data. This is promising, especially considering the non-linear nature of stock market movements.
Concerns: The low training and test R-squared values indicate a weak positive correlation between the features and the predicted returns. This suggests the model might not be capturing the underlying relationships as effectively as desired.

Additionally, limited training iterations restrict our confidence in how well the model will perform on entirely new data in the future.

## Model Comparison

| Model | Training R-squared | Test R-squared | Training MSE | Test MSE |
|---|---|---|---|---|
| Linear Regressor | 0.0164 | 0.0089 | 0.0071 | 0.0072 |
| Decision Tree Regressor | 0.0391 | 0.0126 | 0.0069 | 0.0071 |
| Random Forest Regressor | 0.7681 | 0.0373 | 0.0016 | 0.0070 |
| Neural Network Regressor | 0.0379 | 0.0186 | ~0.007 | ~0.007 |

The Random Forest Regressor outperforms other models based on the provided metrics, exhibiting higher R-squared values and lower mean squared errors (MSE - average squared difference between predicted and actual returns). R-squared values closer to 1 indicate a stronger relationship between the features and the predicted returns, while lower MSE suggests better alignment between predictions and actual values.

However, it's crucial to consider that the Random Forest's hyperparameters might have been limited due to the complexity and size of the dataset, as well as computational constraints. Random Forests, while powerful ensemble methods, can become computationally expensive, especially with large datasets and intricate hyperparameter grids. Therefore, the reported performance metrics might represent only a subset of potential configurations explored, constrained by computational resources.

Despite these limitations, the Random Forest Regressor still demonstrates superior predictive capabilities compared to simpler models like Linear Regression and Decision Trees. This suggests that Random Forests' ability to average predictions from multiple trees helps address overfitting issues common in decision trees, leading to potentially more robust and generalizable models.

Further exploration of alternative models like Support Vector Machines (SVMs) or Gradient Boosting could be beneficial. Additionally, analyzing feature importances from the Random Forest model can help identify the most influential features for predicting stock returns, potentially leading to improved model performance with a reduced feature set.

Given the current constraints, the Random Forest Regressor remains the best-performing option among the evaluated models. However, with more computational power and optimization opportunities, it and potentially other models could yield even better results for predicting monthly stock returns. It's important to remember that predicting the stock market remains a challenging task, and even the best performing model will likely have limitations.

## Feature Engineering with Random Forest Regressor

The analysis explores the creation of new features by interacting all combinations of two existing features (feature_1 * feature_2, feature_1 * feature_3, ...) and calculating the squared value of each feature (feature_1^2, feature_2^2, ...). This approach aims to capture potential non-linear relationships between features that might be missed by the original features alone. It can lead to a richer feature space that the Random Forest model can leverage for improved prediction accuracy.

**Hyperparameter Tuning:**
The tuning process identified a specific configuration that optimizes the model's learning and complexity:

- **Max Depth: None:** This setting allows the individual trees in the forest to grow as deep as necessary to capture complex patterns within the data, especially with the addition of new interaction and squared features.
- **Max Features: 'sqrt':** This continues to consider the square root of the total features (including the newly created ones) for each split in the decision trees. This helps manage the increased feature space while potentially preventing overfitting.
- **Min Samples Leaf: 2:** This parameter ensures that each leaf node in the trees contains at least two data points, improving the model's stability and preventing overly specific decision rules based on very few data points.
- **Min Samples Split: 4:** This parameter requires a minimum of four data points to split a node in the decision trees, again helping to prevent overfitting by ensuring sufficient data at each split point within the trees.
- **Number of Estimators: 53:** This hyperparameter tuning identified 53 as the optimal number of decision trees to include in the random forest. This number balances model complexity with the potential for overfitting.

**Performance Analysis:**

- **Training R-squared: 0.8060:** This high value indicates a strong positive correlation between the features (including the newly engineered ones) and the predicted returns on the training data.
- **Training MSE: 0.0014:** This low value suggests a good fit for the training data.
- **Test R-squared: 0.0463:** This value remains lower than the training R-squared, indicating some limitations in generalizability to unseen data. The model might still be overfitting to some extent.
- **Test MSE: 0.0069:** The test MSE remains similar to the previous model, suggesting potential overfitting.

While the inclusion of interaction and squared features along with hyperparameter tuning improved the model's performance on the training data (higher training R-squared), generalizability to unseen data remains a challenge. This is evident from the lower test R-squared and similar test MSE, suggesting potential overfitting.

The model might be capturing patterns specific to the training data that don't generalize well to unseen data. This is a common limitation of complex models, especially when dealing with financial data like stock returns.

## Edgar Data Acquisition

We begin by scraping the S&P 500 company list from Wikipedia. Each company's symbol is then passed to the Edgar Company Ticker API. This API retrieves the corresponding Central Index Key (CIK) for each company. The CIK is a unique identifier used by Edgar to track filings for publicly traded companies.

We utilize the Edgar Company Facts API to extract relevant financial data for each company identified through the CIKs. This API provides access to a wealth of information, allowing us to calculate several key financial metrics that will serve as additional features in our analysis.

The data retrieved from Edgar's Company Facts API encompasses various financial categories. We leverage these categories to calculate the following features:

- **Profitability** (ROE) calculated from Net Income/Loss and Assets.
- **Liquidity** ratios (current ratio, quick ratio) calculated from Current Assets and Liabilities.
- **Activity** ratios (inventory turnover, receivable turnover) calculated from Revenues and Assets.
- **Financial Leverage** (debt-to-equity ratio) calculated from Assets and Liabilities.
- **Market Valuation** (P/E ratio) potentially calculated from Earnings Per Share and Common Stock Value.

All the features calculated from Edgar's Company Facts API are combined into a new dataframe, ensuring proper association with each company using the Central Index Key (CIK) and ticker symbol.

This newly created dataframe containing the fundamental features is then merged with the historical monthly stock price data obtained from Yahoo Finance for the S&P 500 companies.

The time frame for this merged dataset encompasses the period from the beginning of 2010 to the end of 2023.

## Performance

Hyperparameter Tuning with Enhanced Features:
This analysis builds upon the previous exploration by incorporating interaction and squared features, potentially enriching the model's ability to capture complex relationships within the data. However, due to resource limitations and the growing data complexity, we weren't able to fully explore the model's potential with these new features.

The hyperparameter tuning process identified a promising configuration:

- **Max Depth: None:** Allows trees to grow freely, capturing intricate data patterns.

- **Max Features: 'sqrt':** Considers the square root of total features (including new ones) for splits, managing the increased feature space and potentially preventing overfitting.
- **Min Samples Leaf: 2:** Ensures at least two data points per leaf node, improving stability and avoiding over specific rules.
- **Min Samples Split: 4:** Requires a minimum of four data points for node splits, again preventing overfitting.
- **Number of Estimators: 49:** Balances model complexity with overfitting risk by using 49 decision trees.

**Performance Analysis**:

- **Training R-squared: 0.8605:** This high value suggests a strong positive correlation between features (including new ones) and predicted returns on the training data. There's a potential improvement compared to the previous model due to the richer feature set.
- **Training MSE: 0.0014:** This low value indicates a good fit for the training data. Similar to training R-squared, the new features might lead to a lower MSE compared to the previous model.
- **Test R-squared: 0.0470:** This value remains lower than the training R-squared, highlighting limitations in generalizability to unseen data. The model might still be overfitting to some extent.
- **Test MSE: 0.0069:** The test MSE remains similar to the previous model, suggesting potential overfitting despite the addition of new features.

While incorporating interaction and squared features along with hyperparameter tuning shows promise on the training data, resource constraints limited a full exploration of this approach. The lower test R-squared compared to training R-squared indicates a need to address overfitting and improve generalizability.

## Conclusion

Our exploration of Random Forest hyperparameter tuning and feature engineering yielded promising results. The introduction of interaction and squared features, coupled with optimized hyperparameters, led to a model with a strong positive correlation between features and predicted returns on the training data. However, the key challenge of generalizability to unseen data persists. The model still struggles on test data, suggesting potential overfitting.

Moving forward, addressing resource constraints and data complexity will be crucial. Techniques like dimensionality reduction and regularization can help manage the feature space and improve generalizability. Additionally, exploring alternative machine learning models may be beneficial for this specific task.

By continuing to refine our approach and potentially acquiring more resources, we can develop a robust model that not only performs well on the training data but also generalizes effectively to predict future stock returns with greater accuracy. This will ultimately lead to a more insightful analysis of factors influencing stock performance.