MSBR 70260

Machine Learning: Project Final Report

Revolutionizing Stock Market Predictions with Machine Learning Models

---

Introduction:

As keen investors and students of the stock market on a quest to harness the volatile nature of the financial market, we embarked on a project to develop a machine learning predictive model that allows us to estimate the chosen stock trends. Our attempted model, built to interpret complex signals from market indicators and the relationship between various stock statistics and metrics, can significantly influence the financial landscape, enabling an easier decision-making process for investors.

Our study is not just based on theoretical explorations, but we have also tried to integrate it with practical financial applications. Leveraging data-driven approaches, we have constructed a dataset with a sample of 21,315 from 5 major tech stocks - Apple, Amazon, Microsoft, Google, and Nvidia, drawing from January 3rd, 2007 to November 7th, 2023. This dataset, enriched with 25 variables and features like daily and future returns, SMA, SD, RSI, Bollinger Bands, Beta, and moving averages, was sourced using sophisticated R packages like Quantmod, TTR, and PerformanceAnalytics.

In the face of challenges like market noise and unexpected volatility, we have infused our model with robustness to withstand such anomalies. Conscious of the biases that could detract from our model's accuracy, we employ advanced methods to refine and validate our predictions. At the heart of our strategy lies feature engineering, ensuring that our model is both precise and adaptable to various financial scenarios. This is crucial in our mission to democratize access to powerful investment insights, once exclusive to large institutions.

Our final report is a compendium of our analytical journey, offering insights, visualizations, and a roadmap for future advancements in stock market predictions using machine learning. It is our contribution to the financial analytics domain aimed at empowering investors with state-of-the-art tools for navigating the stock market.

Abstract:

We embarked on a project to predict stock market trends using machine learning, focusing on returns and prices within the tech sector. Facing the complexities of the financial markets, our interdisciplinary approach merges data-driven analysis with advanced modeling techniques. Utilizing data from Yahoo Finance, our dataset spans from January 2007 to November 2023, includes five tech stocks, and consists of 21,315 observations. We've implemented regression models and neural networks to dissect Apple's stock behavior as a case study. We aim to make sophisticated investment tools accessible to individual investors and enrich financial analytics literature. Our study strives to empower informed investment decisions and a deeper comprehension of tech stock risks, advancing the accessibility of cutting-edge financial resources.

Related work (0.5 pages) - What previous work has been done in this area, and by whom? Remember to cite your sources. How is your approach different/an improvement on previous attempts to solve the problem?

Several innovative studies have made significant strides in the dynamic field of financial machine learning. In 2011, Bollen, Mao, and Zeng embarked on a novel approach, using sentiment analysis of Twitter data to forecast movements in the Dow Jones Industrial Average. Their method achieved a notable 87.6% accuracy, showcasing the unexpected value of social media sentiment in predicting market trends.

Krauss, Do, and Huck, in their 2017 study, turned to historical stock prices but through a modern lens, employing sophisticated machine learning techniques like Deep Neural Networks and Random Forests. Their model remarkably outperformed traditional methods, achieving an impressive Sharpe ratio above 4.

Zhang, Zohren, and Roberts, in 2020, adopted a focused approach to high-frequency trading data, utilizing Long Short-Term Memory (LSTM) networks. Their work effectively demonstrated the capability of LSTMs in analyzing the complexities of high-frequency trading data.

Heaton, Polson, and Witte's 2016 research combined traditional financial indicators with unconventional data sources, such as satellite imagery and social media content, using Convolutional Neural Networks. This approach underscored the potential of deep learning in merging diverse data sets for enhanced stock market predictions using sophisticated neural network architectures. However, the dynamic and often unpredictable nature of financial markets means that high performance on historical data does not necessarily translate to future success.

To conclude our exploration of previous research, we wanted to find a study that had similar conclusions as ours. We came across Johnson's 2023 Master's thesis from Harvard University, where the author compared multiple machine learning models in the context of financial market forecasting. Like us, Johnson concluded the document by stating the superiority of SVM models, which performed the best with high accuracy, ranging from 72% to 99%. Further, SVM achieved better precision and recall when compared to linear models. The author remarked on the previous accomplishments of SVM in classification problems, which aligns with the fact that our study is also a classification problem since we are predicting a Yes/No outcome based on our stocks' subsequent day movements (up or down). Johnson's study reinforces our conclusions while incentivizing us to test out other models in the future, especially LSTM.

Unfortunately, unlike our study (which has a great focus on finding the best predictor variables and metrics for stock prediction), most studies focus solely on model performance instead of revealing good predictors. Although revealing the inner workings of different models is crucial, more research is needed to identify the reliable features that can boost short-term stock price prediction.

Our approach has improved over previous attempts as we have thoughtfully incorporated many stock features and metrics tied to the broader stock market benchmark, like the S&P 500. We believe these features have the potential to improve predictive accuracy. As previously mentioned, our study is one of the few that tries to identify features with high predictive power while comparing model performance. Ultimately, our study goes above and beyond by ranking the features and variables with the most predictive power for each of the five tech stocks.

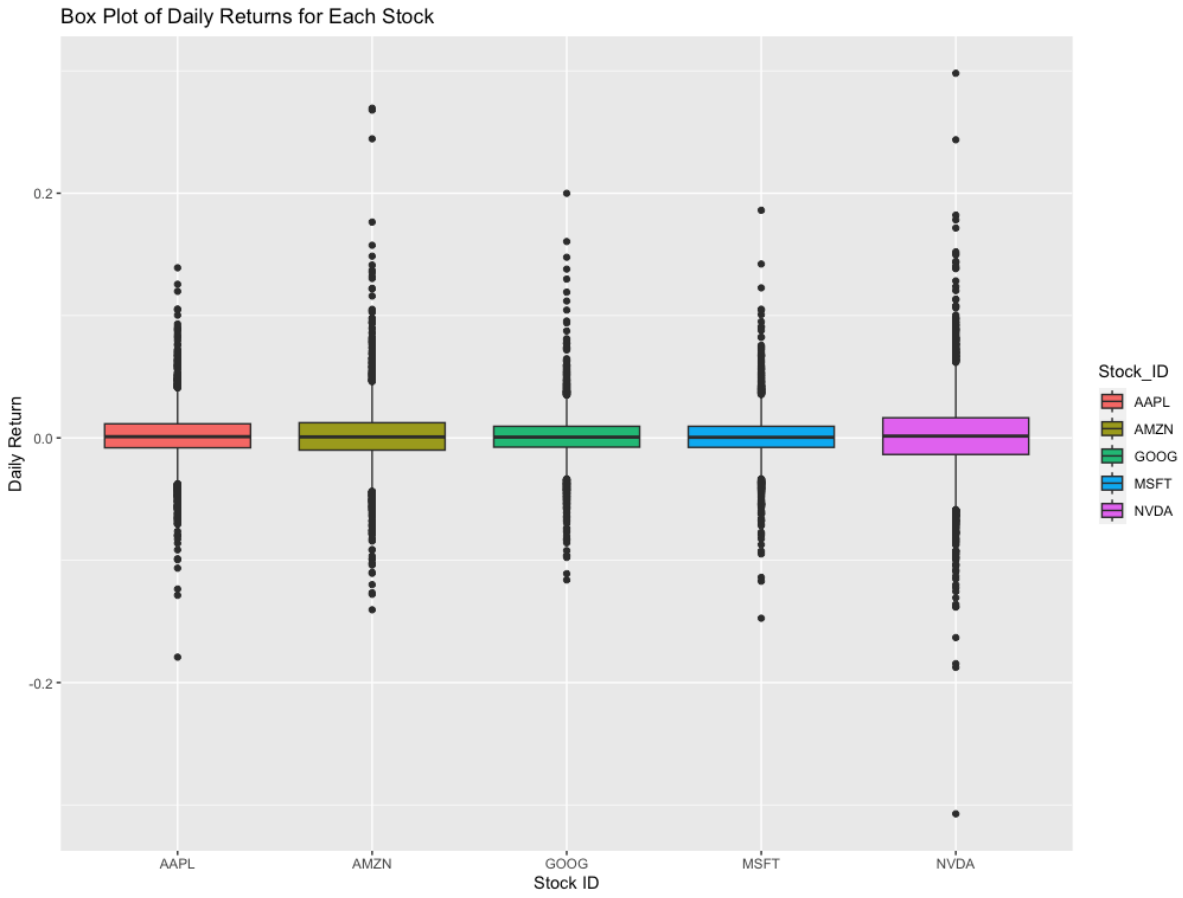
Data Description (0.5 - 1 pages) - Describe the dataset used for the problem, the pre-processing steps taken during the modeling process. How many samples and variables were used, how were these split amongst the training, test and validation sets. Try to include some summary graphs of your dataset.

We retrieved our dataset using the Quantmod, TTR, and PerformanceAnalytics packages. These R stock packages allowed us to retrieve data for five tech stocks: Apple, Amazon, Microsoft, Google, and Nvidia, using various functions. We retrieved daily data for each stock from January 3rd, 2007, to December 7th, 2023. For pre-processing, we cleaned the data and lagged it one (1) day using the "lead" function to predict the future return for each stock, which is our dependent variable. We chose one day as our accuracy would take a hit if the number exceeded one. The data that we retrieved for each stock included Date, daily return, future return, SMA (Simple Moving Average over seven (7) days), SMA.1(Simple Moving Average over 14 days), SMA.2 (Simple Moving Average over 30 days), sd14 (SD over 14 days), sd7 (SD over seven days), sd30 (SD over 30 days), RSI, Upper Bollinger Band (up), MAVG (Modified moving average), Lower Bollinger Bands (dn), EMA (Exponential Moving Average), VWAP (average price of a stock weighted by volume), MACD (Moving average convergence divergence), signal, close, beta, and OBV (On Balance Volume). Additionally, we included S&P 500 derived variables that would help our models' accuracy by considering the changes in the overall

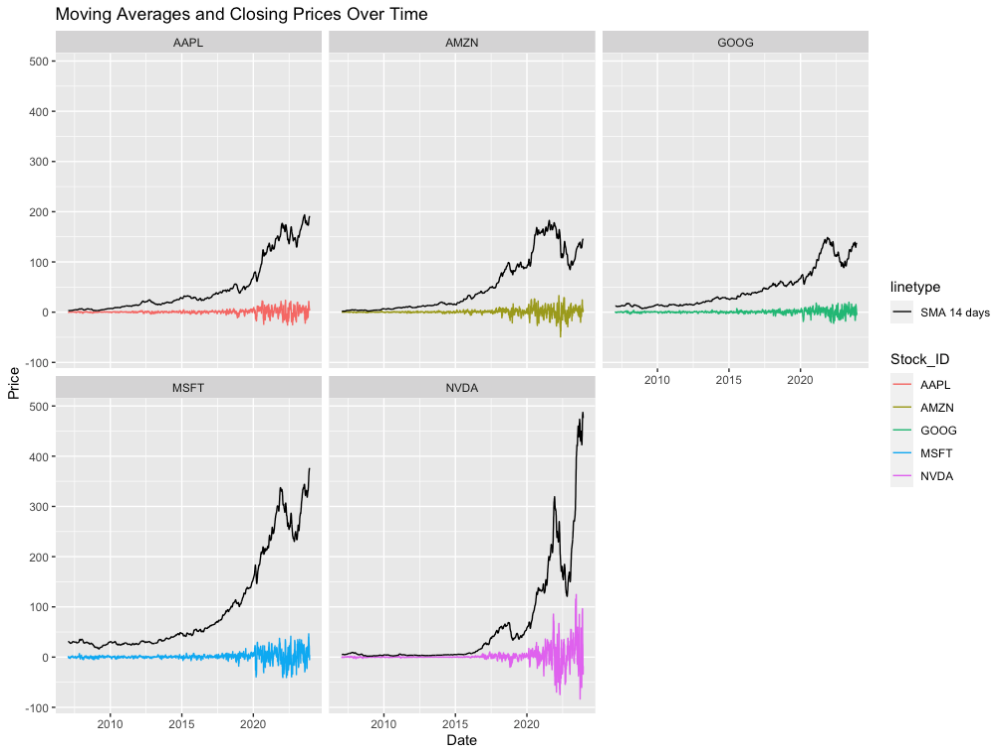
market. The S&P 500 variables include the daily return, the moving average over 14 days, the moving average over 30 days, and RSI. Overall, we have 21,315 samples and 25 variables.

We used a more unconventional approach to split the data amongst training and test sets. Instead of simply partitioning the data using the "sample" function, we used indexing and simple slicing to keep our time order intact. We used 80% of the data for training and 20% for test purposes for each of the five stocks. After splitting the data, we omitted NA's and dropped the date, stock_ID, and close columns, as we only had to work with numerical variables for regression. To predict accurately, we lagged future returns by one day using the lead function. We used the "xgb.cv" function from the regression tuning code in Module 7 to validate our hyperparameters since we conducted the hyperparameter tuning process using the test data.

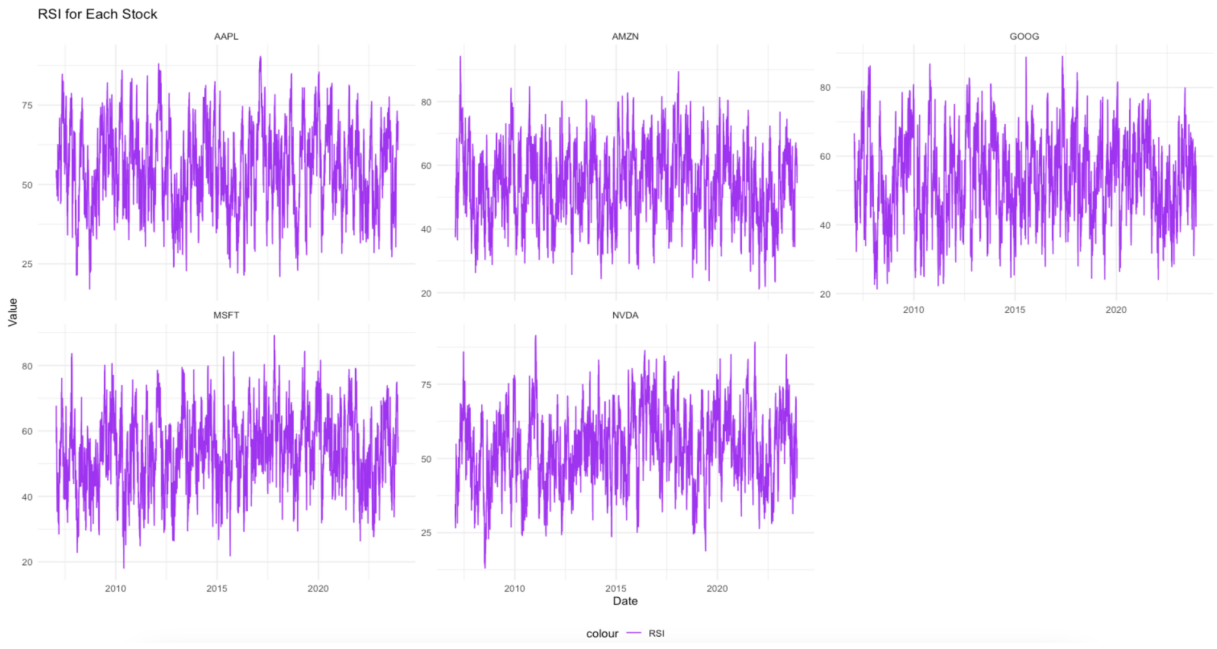
Summary graphs of our dataset:



The box plots of daily returns for each stock are an effective way to compare their volatility. A wider box and longer whiskers indicate higher volatility. NVIDIA seems to have the highest volatility, while Google appears to be the most stable stock. However, the differences are not significant.

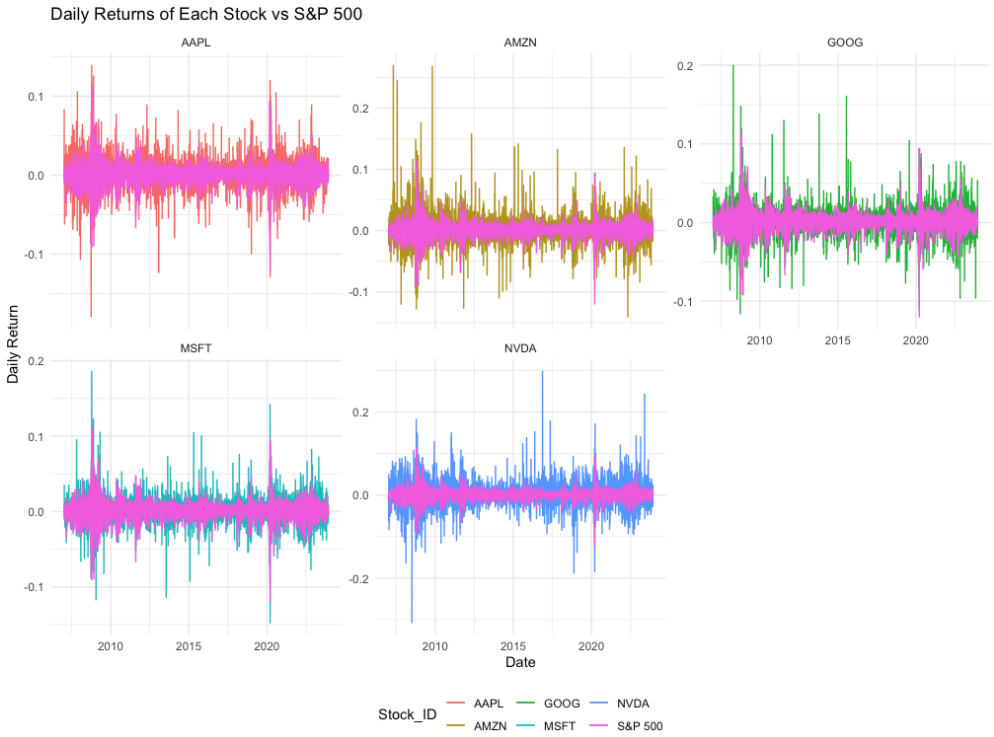


The facet time series plot for closing prices and moving averages (14 days) over time for each stock shows general trends by stock. Once again, we can highlight NVIDIA's volatility - followed by Microsoft - while Google is the more stable stock over the established period. From this chart, we can also conclude that all these tech stocks have had a considerable increase in price over the last ten years.

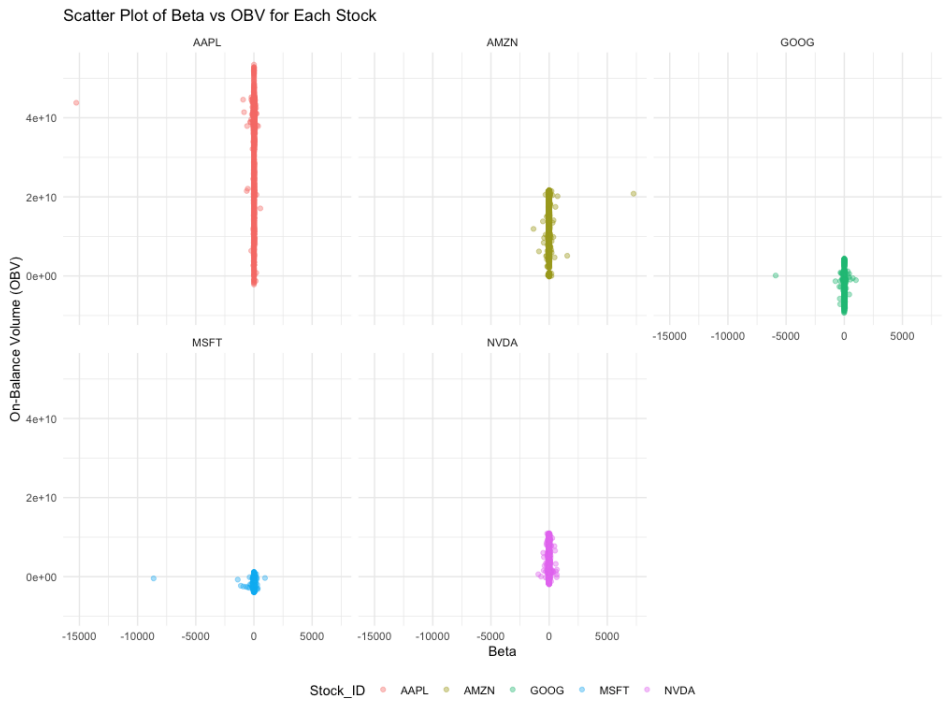


The graph above showcases the RSI variability for each stock. RSI is a momentum oscillator that measures the speed and change of price movements, and it usually oscillates between 0 and 100. The RSI is considered overbought for each stock above 70 and oversold when below 30.

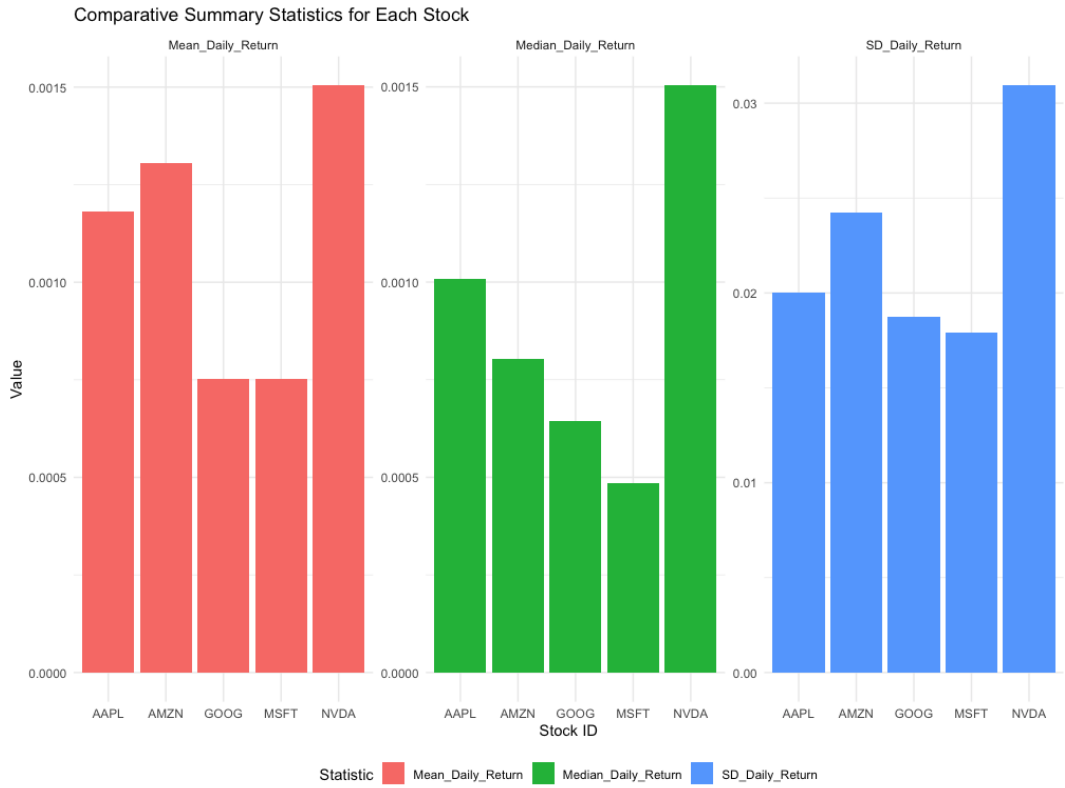
Plotting the Moving Average Convergence Divergence (MACD) and its signal line for each stock can provide valuable insights into its momentum and potential buy or sell signals.



We created the graph above to compare each stock's daily return against the S&P 500's general return metrics. It provides a clear view of how the returns of each stock align or diverge from the broader market trends.



By comparing the Beta and OBV (On-Balance Volume), we can visualize the relationship between market volatility (as measured by Beta) and trading volume (as measured by OBV) for each of our five tech stocks. Although not much is revealed about how stock volatility relates to trading volume, we can see the significant difference between trading volume for each stock, with Apple being the most traded and Microsoft the least.



Lastly, we wanted to visualize summary statistics for each stock's daily returns. Therefore, we constructed a comparative bar chart. Here, we can again point out NVIDIA's volatility and Microsoft's relative stability.

Methods (0.5 - 1 pages) - Provide a one to two paragraph description of the method used, its applicability to/suitability for the problem along with the methods strengths and limitations. I am looking to see your understanding of the method here.

Our initial approach used XGBoost, an excellent tool for capturing non-linear relationships. This characteristic is particularly advantageous in the world of stock market analysis. XGBoost stands out for its capability to manage a wide array of features and interactions, making it an ideal candidate for assessing our 25 variables. Additionally, its robustness in handling large datasets will ensure the seamless processing of our extensive 21,315-row dataset. We anticipate that XGBoost will be the model with the best accuracy. However, there is a risk of overfitting our data if not appropriately tuned. Furthermore, its sensitivity to noisy data, a frequent challenge in stock market datasets, necessitates careful handling.

The second method applied was linear regression. This is useful for understanding the impact of various features on stock returns when relationships are linear. This model is the most parsimonious, making it easy to implement and interpret. Having so many rows for each of the five tech stocks, linear regression shines for its fast computation. Although this model is straightforward, it assumes linearity in data, which is often not the case in stock markets. Our linear model is also prone to being affected by outliers, which do exist in our data. Ultimately, this model may oversimplify the complex relationships in our stock market data.

Next, we tried implementing random forest. Like XGBoost, this method was suitable for capturing complex and non-linear relationships in our stock market data; however, this method did not need extensive hyperparameter tuning. This model is good because it is robust to outliers and noisy data, which stock market data usually has. Since our stock dataset did not have categorical variables, we could not leverage the maximum capabilities of random forest. Like XGBoost, this was one of the most time-consuming and computationally expensive models. It was also more prone to overfitting than other models due to the inherent noise in the tech stock data, and its interpretability could have been clearer.

The fourth applied model was bagging. Since this method has a robust performance on outliers, it can improve the stability and accuracy of our algorithms when used in stock future return prediction. Bagging reduces variance and the risk of overfitting and works well with complex and noisy data, which might be the biggest advantage of this model over the other ones we have utilized. The main issue with this model is a need for more interpretability and transparency, which can be even worse when considering the already complicated nature of stock data. Lastly, it is relevant to mention the risk of one single predictor dominating the others when running this model.

We also tried implementing a Lasso regression. Due to its feature selection capability, the Lasso shines when dealing with high-dimensional data. Although our lower-dimensional dataset did not take full advantage of Lasso's feature selection capability, it was beneficial since it prevented high overfitting and handled the decently large number of derived features we have. Our Lasso did not suffer from much bias since we had more observations than features. The main issue with this model is that it is likely to miss important signals as the true relationship of our variables is highly complex.

For our last method, we wanted to try something new. So, we attempted an SVM (Support Vector Machines) model. SVM uses supervised learning models to solve problems. SVMs are used for Classification, Regression, and Outlier detection. Most notably, SVMs are particularly good at solving binary classification problems. This is apt for our use case as we tried to predict whether the stock would trend positively or negatively the next day. They are also effective in high-dimensional spaces. If we used the appropriate kernel, the SVM model would effectively capture the complex

relationships between our variables. Its generalization capabilities came in handy when assessing our various derived variables. We realized that SVM would struggle with a moderately large dataset like ours and its inherent noisy nature. Further, we found that choosing the right kernel was a challenging endeavor, so we opted to use the e1071 package to simplify our code and predictions.

Predicting future stock returns is challenging due to financial markets' complex and unpredictable nature. Our non-stationary and noisy data makes it difficult for our models to accurately capture and predict future trends - even if it's just one day out. We picked an array of features based on previous research and our knowledge of financial markets, yet feature engineering is an ongoing and ever-changing process. While some of our models achieved an accuracy that is a little more reliable than a coin toss, external factors like economic changes, political events, and psychosocial dynamics make stock market prediction extremely difficult. We are not the first to attempt this task and will not be the last.

Results (1-1.5 pages) - Describe your application of the method to your problem. What were the hyper parameters chosen? How was the model fit? What was the output of the method? How did the method perform on the training, test, and validation sets? What metrics were used to judge the method and provide a description of them (e.g. accuracy, sensitivity, AUC, etc.)? For classification problems you should include a confusion matrix of model accuracy while for regression problems you should provide an estimate of the error such as MSE. Present any challenges or issues in the project or model fitting process.

This section presents the findings from our application of the XGBoost model to the dataset. We aimed to optimize the model for the highest predictive accuracy by experimenting with various hyperparameters.

Model Configuration
We initially configured our XGBoost model with the following parameters, only testing on AAPL data:

Learning rate (eta): 0.05
Max depth: 3
Min child weight: 3
Gamma: 0
Subsample: 0.9
Colsample_bytree: 0.9

To explore the impact of the 'gamma' parameter on model performance, we tested additional configurations with gamma values set at 0.01 and 0.005.

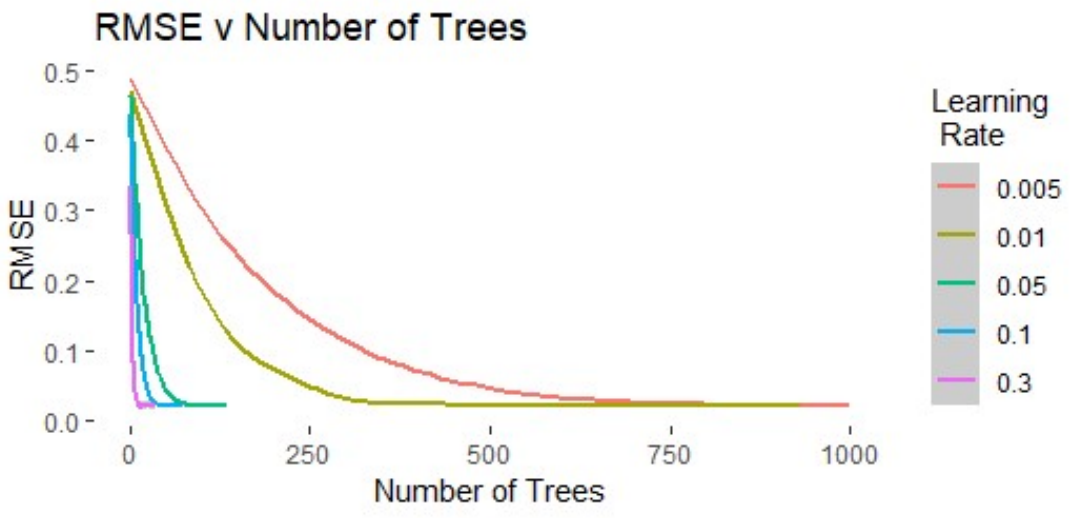
Looking at our RMSE v. Number of Trees chart, the goal was to pick a learning rate that minimizes the RMSE while also considering the model's efficiency (i.e., using a reasonably small number of trees).

From the plot, you can observe the following about the learning rates:

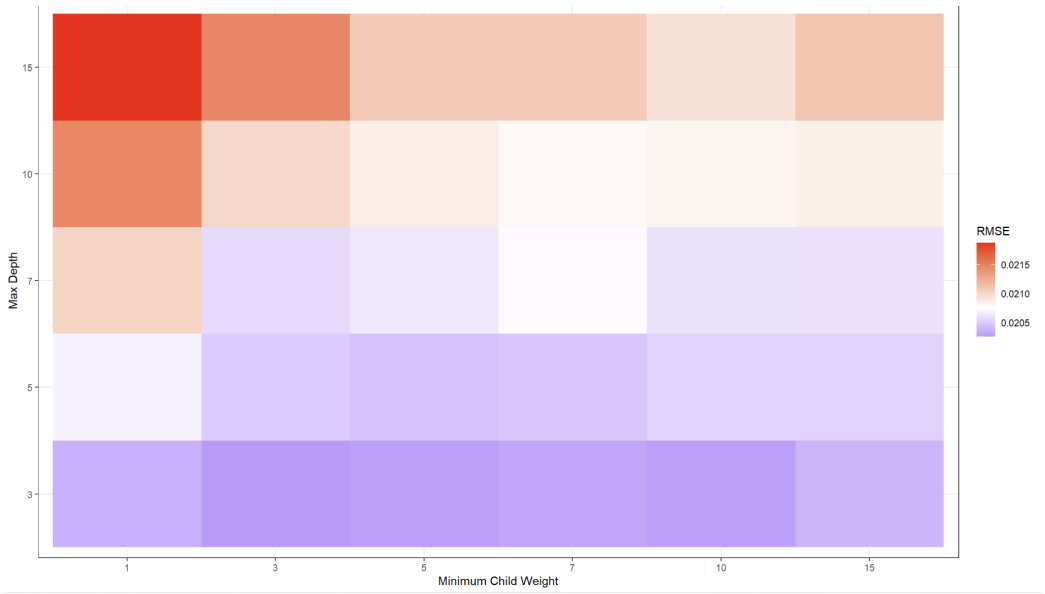
- Learning rate of 0.3 (purple line): This rate converges quickly, but the RMSE does not decrease much after the initial drop. It may be reaching a plateau early, indicating underfitting or insufficient trees.
- Learning rate of 0.1 (pink line): The RMSE decreases steadily and seems to stabilize after a certain number of trees. This learning rate is generally a good balance between convergence speed and performance.
- Learning rate of 0.05 (green line): The RMSE decreases and almost parallels the 0.1 learning rate after a higher number of trees, but it requires more trees to achieve a similar RMSE, which can increase the computational cost.

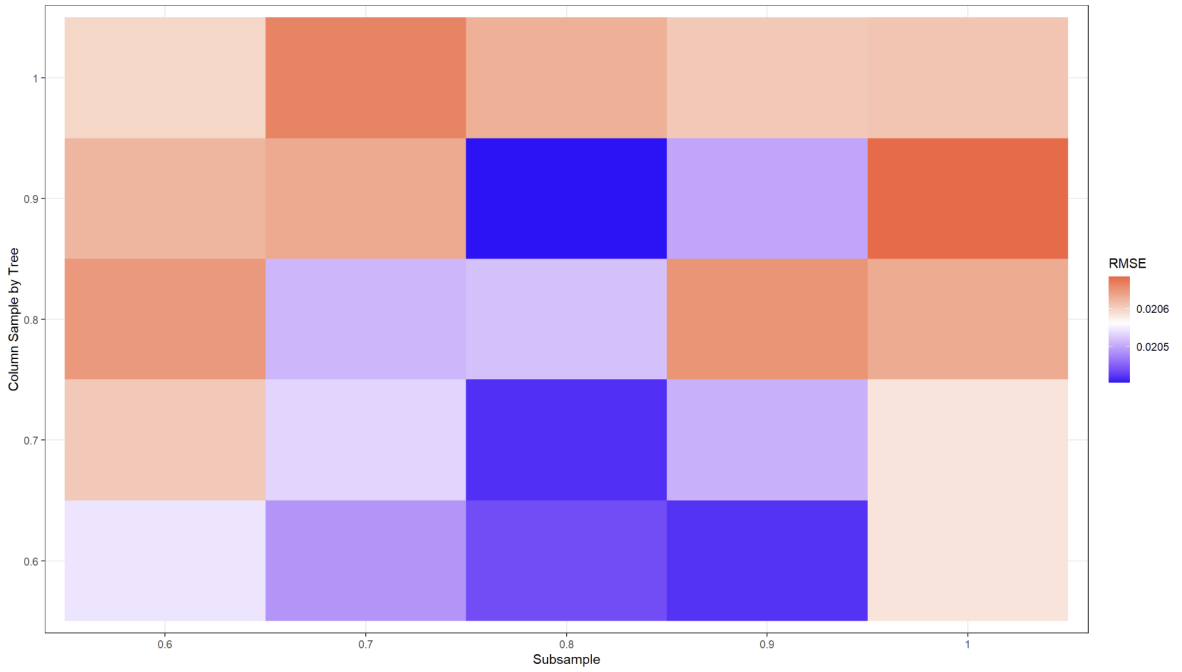
- Learning rate of 0.01 (orange line): This learning rate continues to decrease slowly, suggesting that it may continue to improve with even more trees, but it requires a lot of trees to reach an optimal RMSE, which can be computationally expensive.
- Learning rate of 0.005 (blue line): This rate shows the slowest convergence and would require a very large number of trees to reach an optimal RMSE, making it the least efficient choice among those plotted.

In choosing the best learning rate, you want to find the "sweet spot" where the RMSE is low, but the model also learns quickly (usually fewer trees). From the plot, the learning rate of 0.1 appears to be a good compromise, as it reaches a low RMSE quickly and stabilizes soon after that. However, it is also essential to consider the behavior beyond what's shown in the plot. Sometimes, a slightly **lower learning rate, like 0.05,** could result in a better RMSE, given enough trees, without overfitting. **Hence, we chose 0.05.**

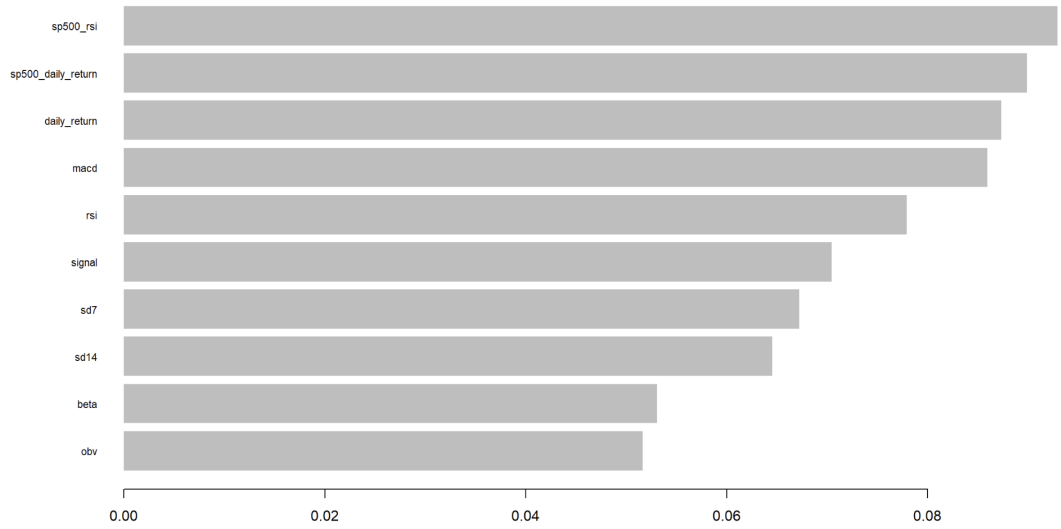


We chose our max depth, min child weight, subsample, and colsample_bytree using the below heatmaps. We chose the darkest purples and the deepest blues as they had the lowest RMSE.





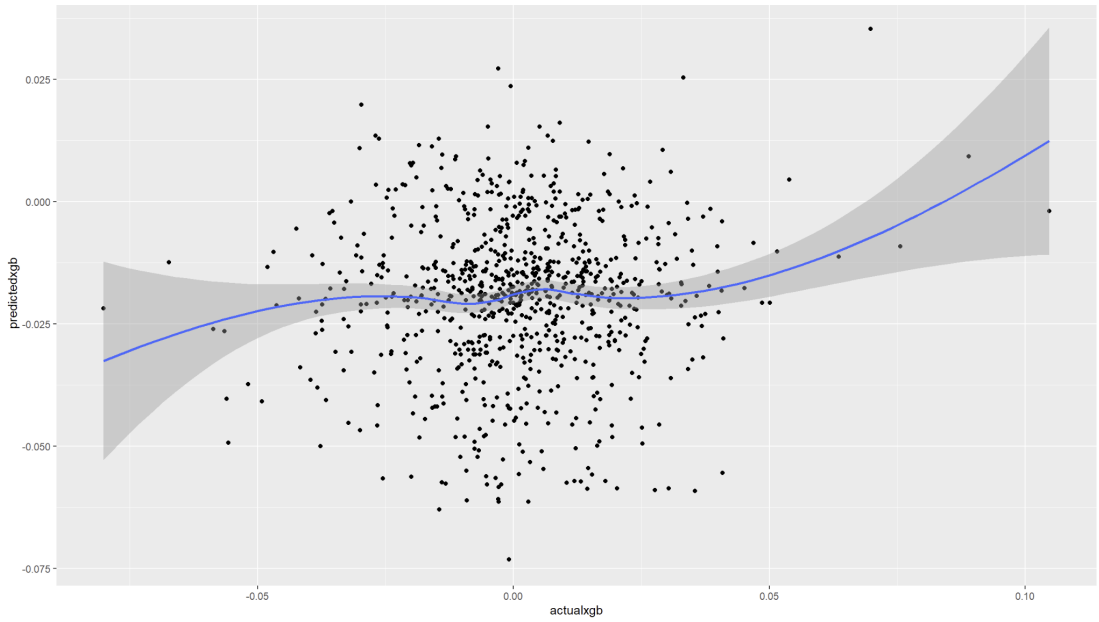
The output of the XGBoost showed the below feature importance and SHAP plot.



The RSI for the S&P 500, the S&P 500's daily return, and the AAPL's daily return.

We used the SHAP plot to identify what role different variables play in our XGBoost model, how they impact our model, and what value they have. The MACD and OBV came out on top.

We also looked at our predicted vs. actual values in the plot below. The model fit was not too good. Otherwise, we would have seen the points clustering around a 45-degree line.



Model Performance

The model's accuracy varied significantly with different gamma settings:

With gamma set to 0.02, the model achieved the highest accuracy of 53.43%.
A gamma value of 0.005 resulted in 49% accuracy.
The lowest accuracy, 47%, was observed when we set gamma to 0.

We also generated confusion matrices for each configuration to provide deeper insights into model performance.

<u>Hyperparameter Tuning and Confusion Matrix for gamma = 0.01 for AAPL</u>

eta = 0.05, max.depth = 3, min_child_weight = 3, gamma = 0.01, subsample = 0.8, colsample_bytree = 0.9

We did not use this gamma for everything because it would not output a SHAP plot. Anything above gamma = 0.03, and the xgb.imp function won't run. Hence, we stuck with gamma = 0, even though our accuracy fell using it. This was the tradeoff to obtain a better SHAP and XGB importance.

	Reference	
Prediction	Negative	Positive
Negative	192	185
Positive	211	258
Accuracy:	0.5319	
Sensitivity:	0.5824	
Specificity:	0.4764	

Hyperparameter Tuning and Confusion Matrix for gamma = 0.005 for AAPL
eta = 0.05, max.depth = 3, min_child_weight = 10, gamma = 0.005, subsample = 0.8, colsample_bytree = 0.9

	Reference	
Prediction	Negative	Positive
Negative	311	332
Positive	92	111
Accuracy:	0.4988	
Sensitivity:	0.2506	
Specificity:	0.7717	

Hyperparameter Tuning and Confusion Matrix for gamma = 0 for AAPL
eta = 0.3, max.depth = 3, min_child_weight = 3, gamma = 0, subsample = 0.8, colsample_bytree = 0.9

	Reference	
Prediction	Negative	Positive
Negative	401	439
Positive	2	4
Accuracy:	0.4787	
Sensitivity:	0.009029	
Specificity:	0.995037	

Hyperparameter Tuning and Confusion Matrix for gamma = 0.02 for AAPL
eta = 0.05, max.depth = 3, min_child_weight = 10, gamma = 0.02, subsample = 0.8, colsample_bytree = 0.9

	Reference	
Prediction	Negative	Positive
Negative	177	168
Positive	226	275
Accuracy:	0.5343	
Sensitivity:	0.6208	
Specificity:	0.4392	

This model is relatively balanced in being either too optimistic or too pessimistic. However, it shows mediocre accuracy, sensitivity, and specificity.

Please see the summary statistics here:

The accuracy of each model for each stock return prediction has been chosen as the key metric for model evaluation. It allows us to understand the model's overall performance before deeper analysis into other metrics, such as sensitivity, specificity, and errors.

Accuracy						
	Linear	xgboost	Random Forest	Bagging	Lasso	SVM
AAPL	-	0.4704	0.4882	0.4894	-	**0.5177**
AMZN	-	0.4976	0.4976	**0.5035**	-	0.4669
GOOG	-	0.4894	0.4752	0.4752	-	**0.5213**
MSFT	-	**0.5106**	0.4941	0.4929	-	0.5083
NVDA	-	0.4976	0.5000	0.5059	-	**0.5272**

The SVM model was the clear winner for most of our stocks, beating out a partially tuned XGBoost and Random Forest.

Sensitivity is crucial in assessing a model's ability to correctly identify positive cases, which is especially important in high-stakes scenarios like stock predictions for trading. It helps understand the balance between detecting true positives and the risk of missing them, which can inform necessary adjustments in model performance.

Sensitivity						
	Linear	xgboost	Random Forest	Bagging	Lasso	SVM
AAPL	-	0.0971	0.1986	0.1558	-	**0.9594**
AMZN	-	**0.7425**	0.5731	0.6079	-	0.7146
GOOG	-	0.1454	0.0067	0.0067	-	**0.7964**
MSFT	-	0.6682	0.0415	0.0461	-	**0.7535**
NVDA	-	0.7594	0.7748	0.7792	-	**0.9757**

Once again, SVM's sensitivity was highest for 4/5 stocks.

Specificity is important as it measures a model's ability to correctly identify negative cases, ensuring it doesn't over-diagnose or is not too risk-averse in risk management terms.

Random Forest and XGBoost had the highest specificity.

Specificity						
	Linear	xgboost	Random Forest	Bagging	Lasso	SVM
AAPL	-	**0.8809**	0.8065	0.8561	-	0.0323
AMZN	-	0.2434	**0.4193**	0.3952	-	0.2096
GOOG	-	0.8747	**1.0000**	1.0000	-	0.2130
MSFT	-	0.3447	**0.9709**	0.9636	-	0.2500
NVDA	-	**0.1959**	0.1832	0.1908	-	0.0102

RMSE (Root Mean Square Error), another key metric to evaluate model performance, provides a clear measure of the average magnitude of the model's prediction errors, reflecting both the variance and bias in the predictions.

SVM's RMSE was the lowest across all stocks.

RMSE						
	Linear	xgboost	Random Forest	Bagging	Lasso	SVM
AAPL	0.0226	0.0314	0.0285	0.0282	0.0444	**0.0190**
AMZN	0.0242	0.0271	**0.0248**	0.0251	0.0466	0.0250
GOOG	0.0204	0.0290	0.0315	0.0332	0.0421	**0.0199**
MSFT	0.0203	0.0213	0.0295	0.0291	0.1044	**0.0180**
NVDA	0.0368	0.0419	0.0457	0.0520	0.0737	**0.0333**

Model Fitting Process

Throughout the model fitting process, we encountered challenges primarily in balancing accuracy with computational efficiency. The RMSE plot was instrumental in guiding our choice of the learning rate, highlighting the trade-off between convergence speed and performance.

The other models tested were a linear model, bagging, lasso, SVM, and random forests. Please see the detailed results for each model in our appendix. We also tried neural networks that took too long to run and generated low accuracies.

Discussion (1-2 pages) - Discuss the results/findings coming from the model. What were the insights gained from the model? What actions can be taken off of the back of the project? Include suitable visualizations to support your conclusions.

Analysis of Results

The results indicate that the 'gamma' parameter significantly influences model accuracy, with a gamma of 0.02 yielding the best results. However, this improvement in accuracy comes with its trade-offs, notably in terms of model complexity and interpretability. With a gamma higher than 0.03, we could not generate variable importance charts or SHAP values. Hence, we stuck to 0.

The confusion matrices provide a nuanced view of model performance, revealing how the model fares in terms of false positives and false negatives.

Our predicted vs. actual plots show that our models could have fit the data better as we saw scattered points.

Insights and Implications

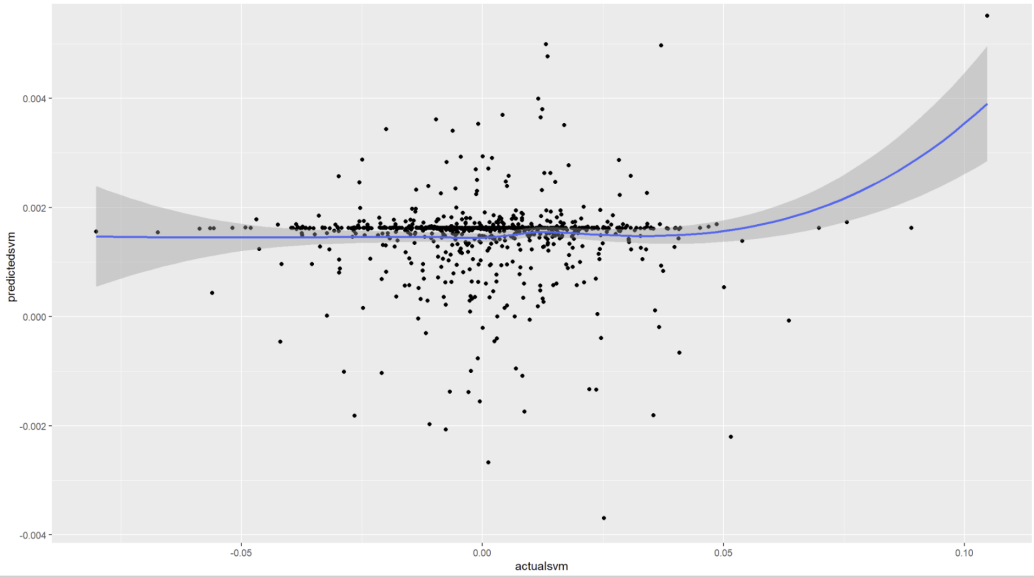
From the RMSE analysis, we observed that while lower learning rates require more trees, potentially increasing computational cost, they might offer more refined performance. The learning rate of 0.1, represented by the pink line in our plot, emerged as a balanced choice, showing a low RMSE without excessive trees.

Concluding Visualizations

We put together graphs demonstrating the best model's predictive capabilities for each stock to highlight our discoveries. The scatter plots show actual vs predicted values of future stock returns. The charts depict that, although achieving acceptable results, our models could not fully capture the 45-degree line we sought in these visualizations.

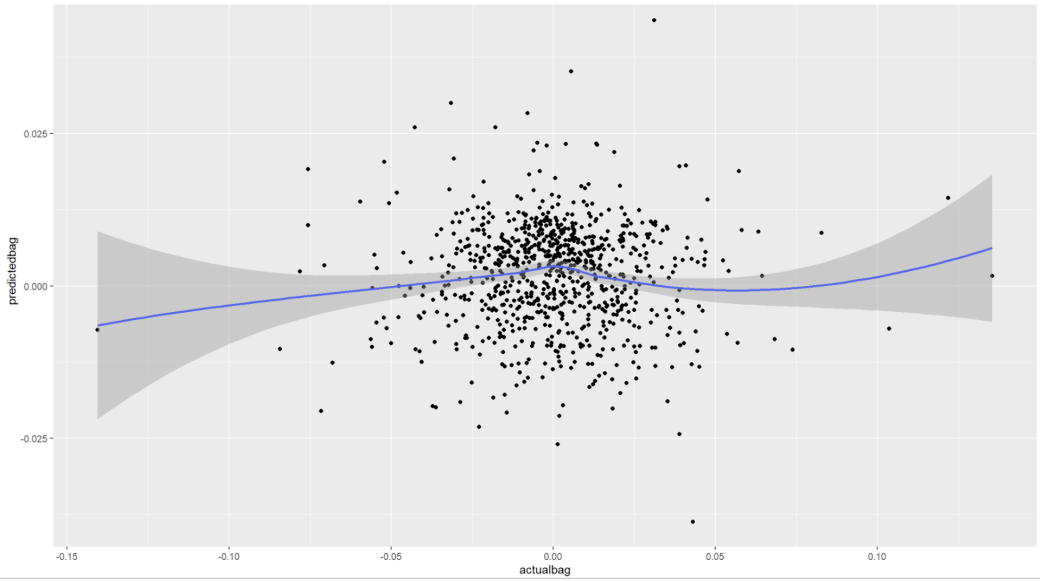
Findings: Apple Inc
NASDAQ: AAPL

SVM (51.77%)



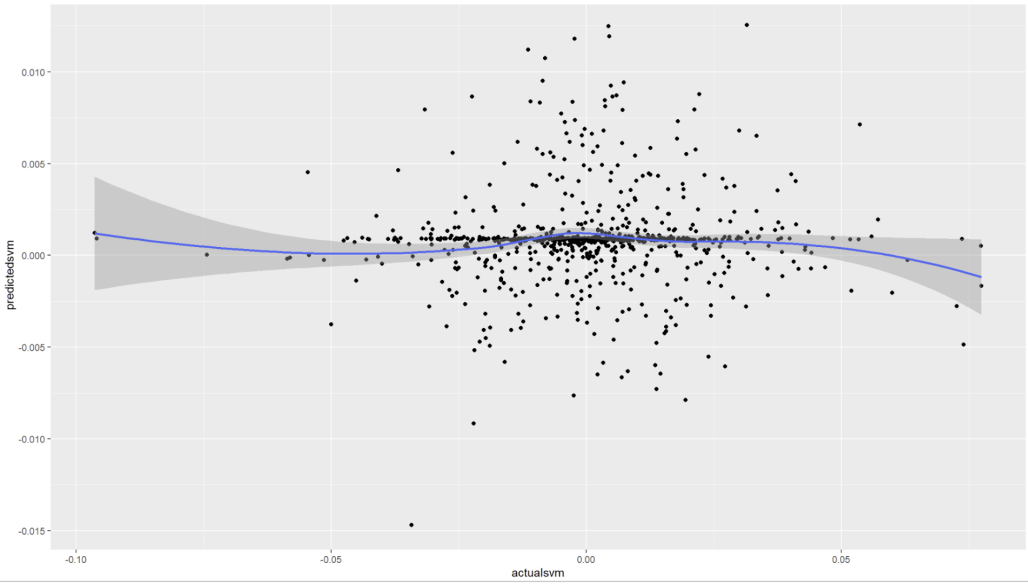
Findings: Amazon.com Inc
NASDAQ: AMZN

Bagging (50.35%)



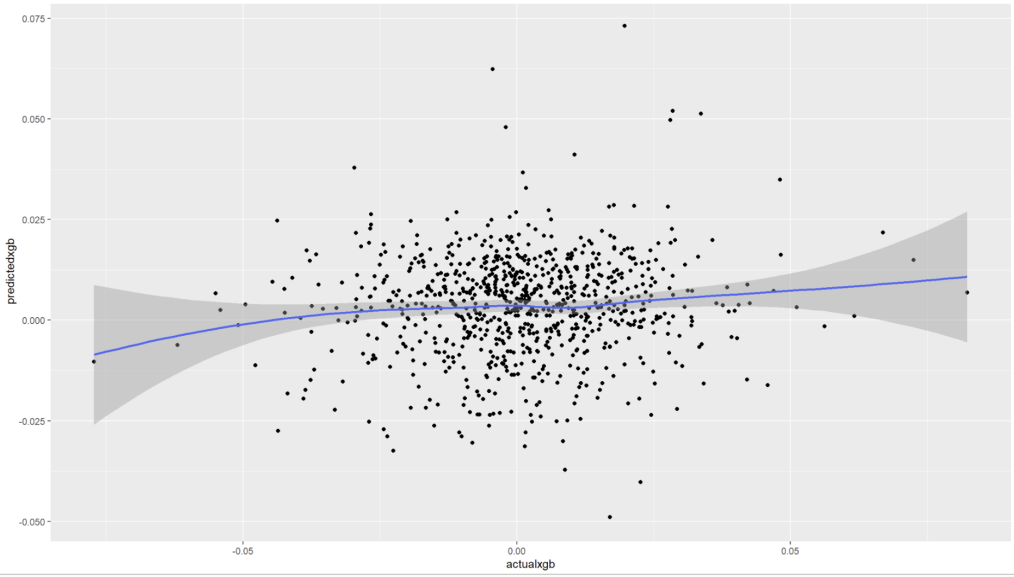
Findings: Alphabet Inc Class A
NASDAQ: GOOGL

SVM (52.13%)



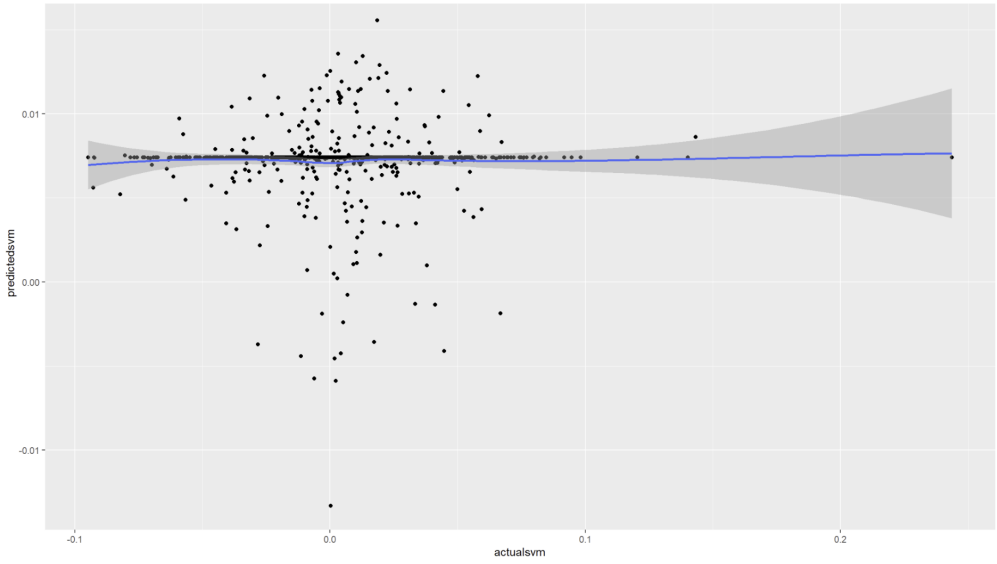
Findings: Microsoft Corp
NASDAQ: MSFT

XGBoost (51.06%)



Findings: NVIDIA Corp
NASDAQ: NVDA

SVM (52.72%)



Actions and Recommendations

Based on our findings, we recommend using a combination of the predictions of the models applied to develop a custom ensemble model based on XGBoost logic and parameter tuning. Additionally, future work could explore further optimization of the 'min_child_weight' and 'subsample' parameters, which we adjusted in our final model configuration but did not extensively experiment with. Finally, the predictions can be more accurate and sensitive if sentiment analysis focusing on Twitter and Bloomberg data is also performed.

We must also remember that we trained the models on US technology stock data. If we were to use the models on stocks from another sector or stocks from another country, performance may vary or suffer. This is a source of bias as we trained on X and are testing on Y. We could have also lagged for more than one day out and changed up the threshold of positive and negative from 0 to -0.05 to experiment with the cutoff.

Concluding Remarks

In conclusion, our application of the XGBoost model demonstrates [summarize key findings]. The balance between accuracy and computational efficiency remains critical in model tuning.

Conclusion

The highest accuracy was 53.43% with the XGBoost model for AAPL with a tuned Gamma parameter (0.02). However, with SVM and gamma = 0, NVDA's accuracy was greatest at 52.72%. In general, the stock price/return prediction is a tedious process due to the unpredictable behavior of different models and primarily due to the inherent unpredictability of stock markets. Moreover, this project's scope was limited by the number of variables and did not include sentiment analysis and utilization of more complex models. Nevertheless, the results are satisfactory because we could determine what models work correctly for our problem and what could be done in future work. The main takeaways from this project are:

- The **SVM outperformed** all other models for most stocks when looking at accuracy, sensitivity, and RMSE despite tuning the XGBoost.
- Financial markets cannot be fully predicted. Usually, it is possible to predict with around 55% accuracy, and some advanced models and methodologies models may achieve an accuracy of up to 80% given the right features and tuning.

- Model selection and tuning differ for each stock type and other financial instruments.
- When tuning parameters, a balanced approach must be found to achieve a satisfactory accuracy rate and avoid overfitting.

Future Work

We believe sentiment analysis on Twitter, Bloomberg, Financial Times, The Economist, and other finance and stock market-related magazines would primarily benefit the analysis. It would enhance the reliability of the models. Another aspect that would produce the accuracy and predictive power of the features would be utilizing more advanced machine learning models, such as LSTM and Bi-LSTM. These models are considered the most robust in handling sequential data, robustness to noise, and general adaptiveness.

Finally, we would have liked to build a custom ensemble model that took in predictions from all the models and bound them back to the dataset. So, the custom ensemble model would capture linear relationships from the linear model and the complexity from the XGBoost model with feature selection capabilities to build an ultimately superior model.

Ultimately, we have learned a valuable lesson about the efficient market hypothesis, a financial theory that states that asset prices reflect all available information. This means that it's not possible to beat the market consistently, and will stick to investing in the overall market to build long-term wealth.

Contributions (Does not count towards limit) - What work was done by each of the team members on the project?

Adi Mhatre: results summarization, presentation planning & design, overall project management
Vadym Zeinalov: data scraping, model selection, model tuning, planning
Mateo Acosta Loza: data wrangling, model/parameter tuning, visualizations, final presentation
Jay Sampat: data wrangling, model selection, debugging, communication

Bibliography (Does not count towards limit) - Citations for the references used.

1. *RPubs - Introduction to Yahoo Finance in R*. https://rpubs.com/kcin999/yahoo-finance. Accessed 16 Nov. 2023.

Quantmod: Quantitative Financial Modelling Framework. https://www.quantmod.com/. Accessed 16 Nov. 2023.

2. *'How to Download Stock Prices in R :: Coding Finance —'. How to Download Stock Prices in R, 2 Apr. 2018, https://www.codingfinance.com/post/2018-03-27-download-price/.*

3. *Yu, Ko Chiu. Chapter 7 Quantmod | Technical Analysis with R. bookdown.org, https://bookdown.org/kochiuyu/Technical-Analysis-with-R/quantmod.html. Accessed 16 Nov. 2023.*

Random Forest and XGBoost:

4. *Liaw, A., & Wiener, M. (2015). randomForest: Breiman and Cutler's Random Forests for Classification and Regression. https://cran.r-project.org/web/packages/randomForest/index.html. Accessed 2 Dec. 2023.*

5. *Johnson, Jaya. 2023. Machine Learning for Financial Market Forecasting. Master's thesis, Harvard University Division of Continuing Education.*

Bagging (Bootstrap Aggregating):

6. *Biau, G., & Scornet, E. (2016). A Random Forest Guided Tour. Test, 25(2), 197-227. Accessed 2 Dec. 2023. URL: https://link.springer.com/article/10.1007/s11749-016-0481-7*

Lasso Regression:

7. *Hastie, T., Tibshirani, R., & Wainwright, M. (2015). Statistical Learning with Sparsity: The Lasso and Generalizations. CRC Press. Accessed 2 Dec. 2023. URL: https://www.crcpress.com/Statistical-Learning-with-Sparsity-The-Lasso-and-Generalizations/Hastie-Tibshirani-Wainwright/p/book/9781498712163*

Linear Models:

8. *James, G., Witten, D., Hastie, T., & Tibshirani, R. (2015). An Introduction to Statistical Learning: with Applications in R. Springer Publishing Company, Incorporated. Accessed 2 Dec. 2023. URL: https://www.springer.com/gp/book/9781461471370*

Support Vector Machines (SVM):

9. *Steinwart, I., & Christmann, A. (2018). Support Vector Machines. Springer. Accessed 2 Dec. 2023. URL: https://www.springer.com/gp/book/9780387772417*

R-code (Does not count towards limit) - This should be provided separately R-script and should allow the findings and graphs in your project report to be recreated.

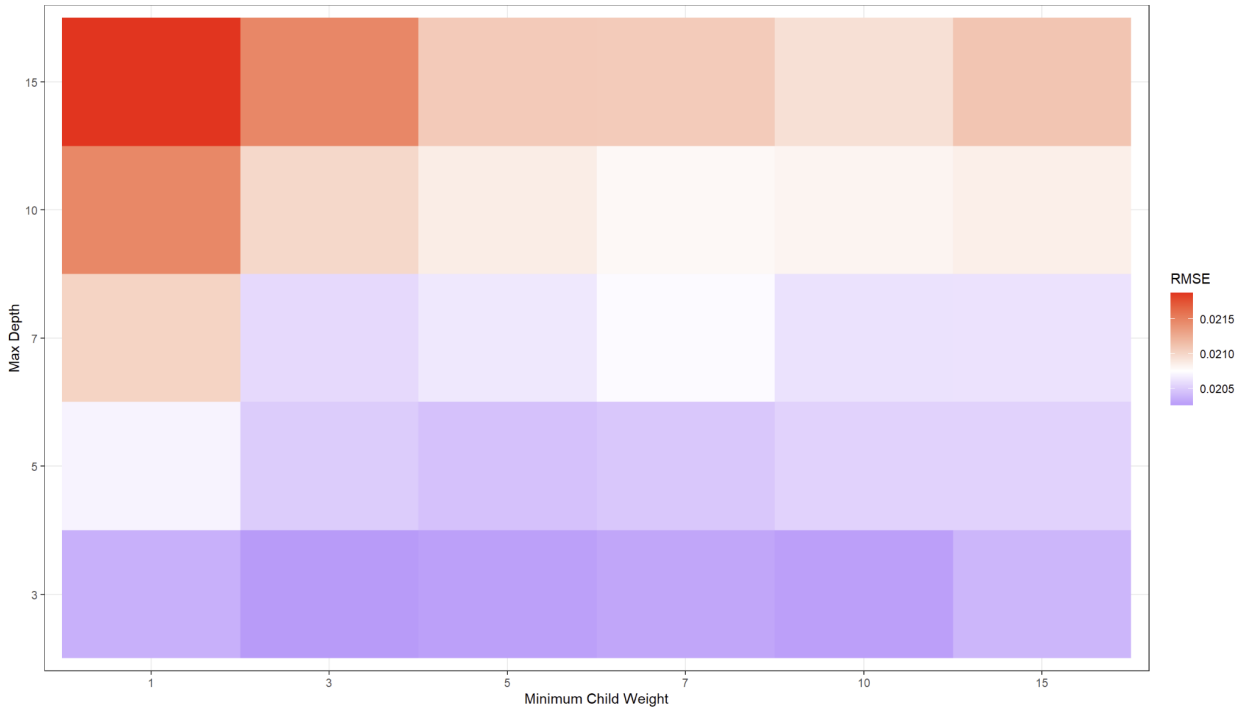
Please refer to separate R scripts attached with the submission on Canvas

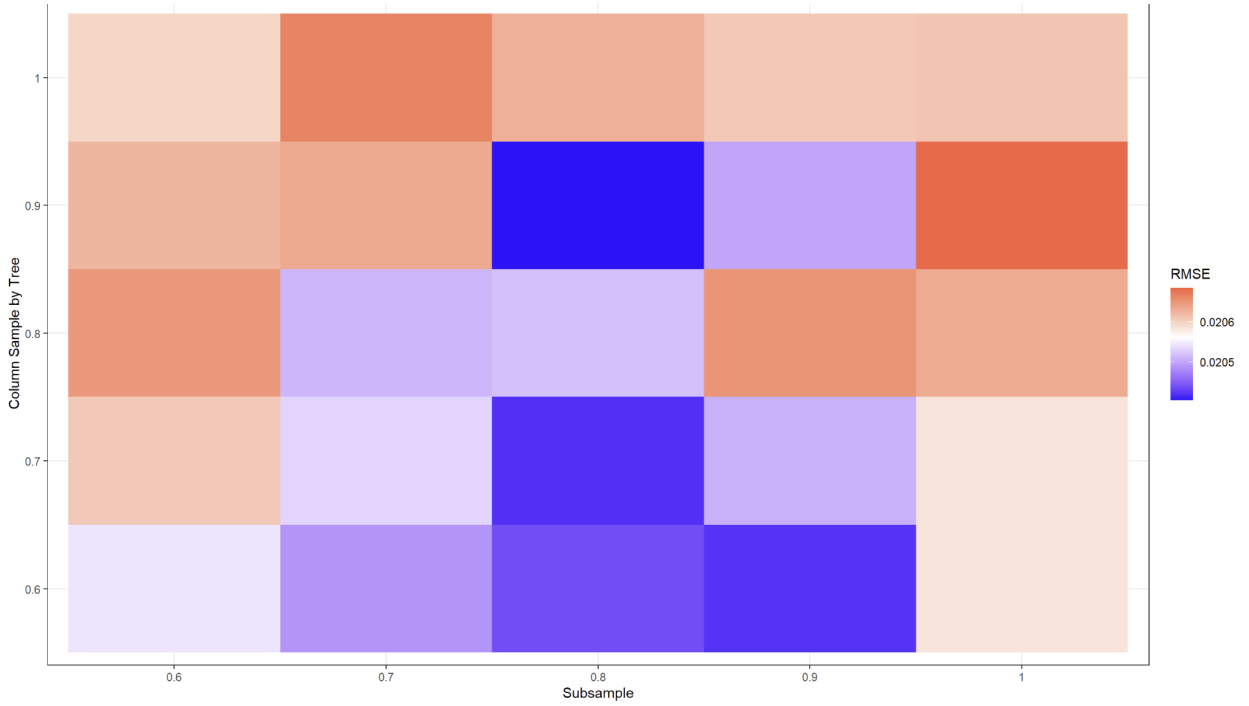
Appendix

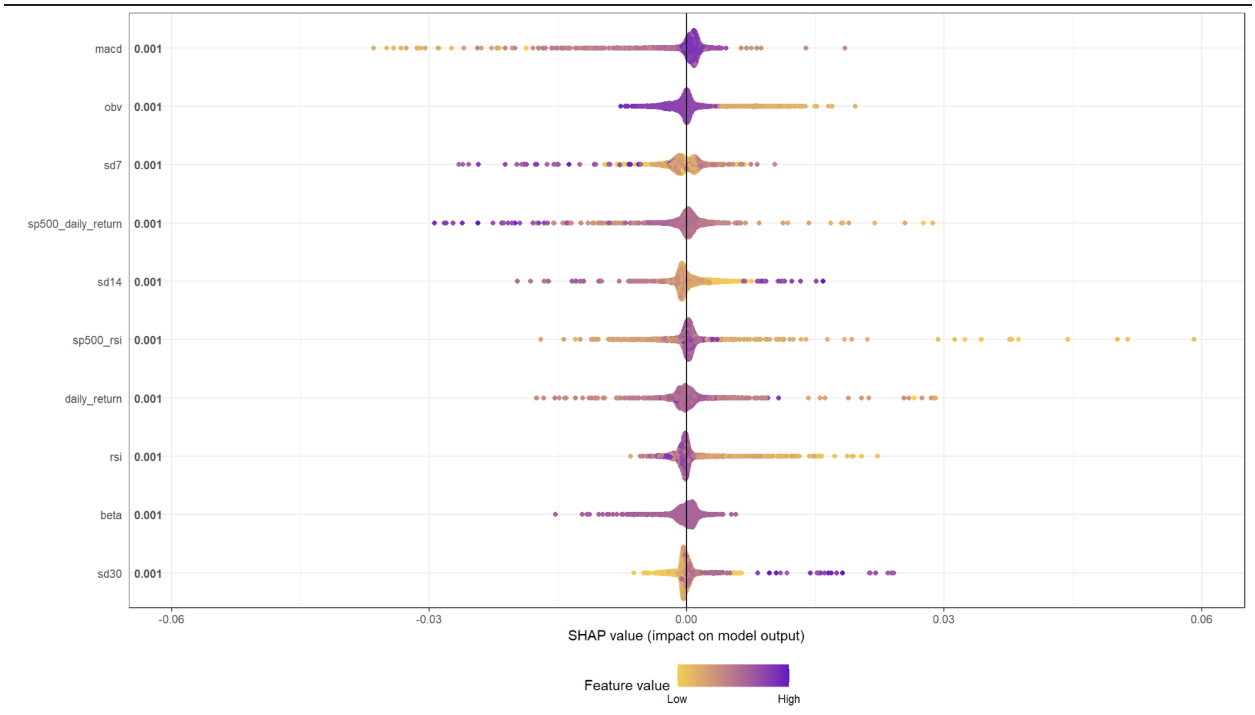
Final parameters used for all models:

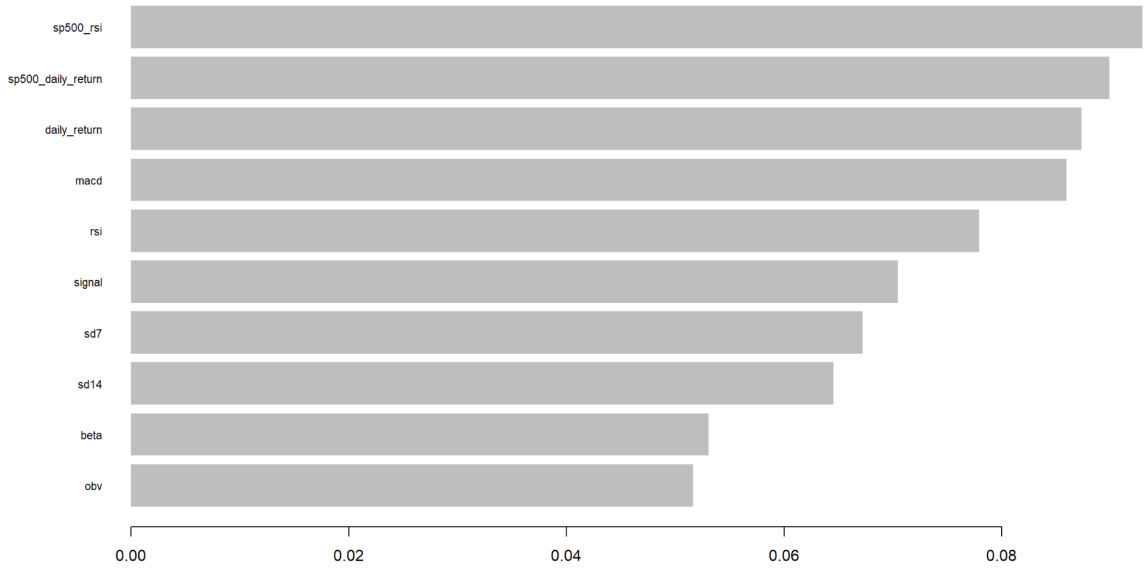
AAPL

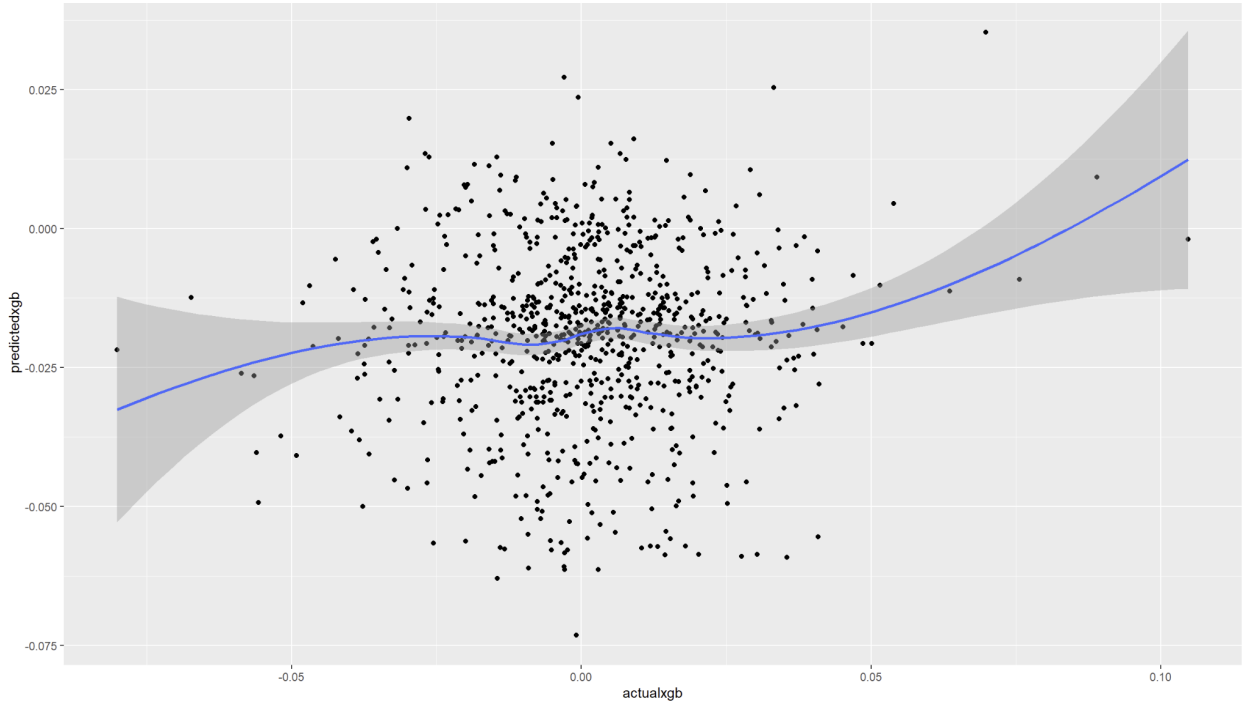
eta = 0.05,
# Set learning rate
max.depth = 3,
# Set max depth
min_child_weight = 10,
# Set minimum number of samples in node to split
gamma = 0,
# Set minimum loss reduction for split
subsample = 0.8,
# Set proportion of training data to use in tree
colsample_bytree = 0.9,
# Set number of variables to use in each tree

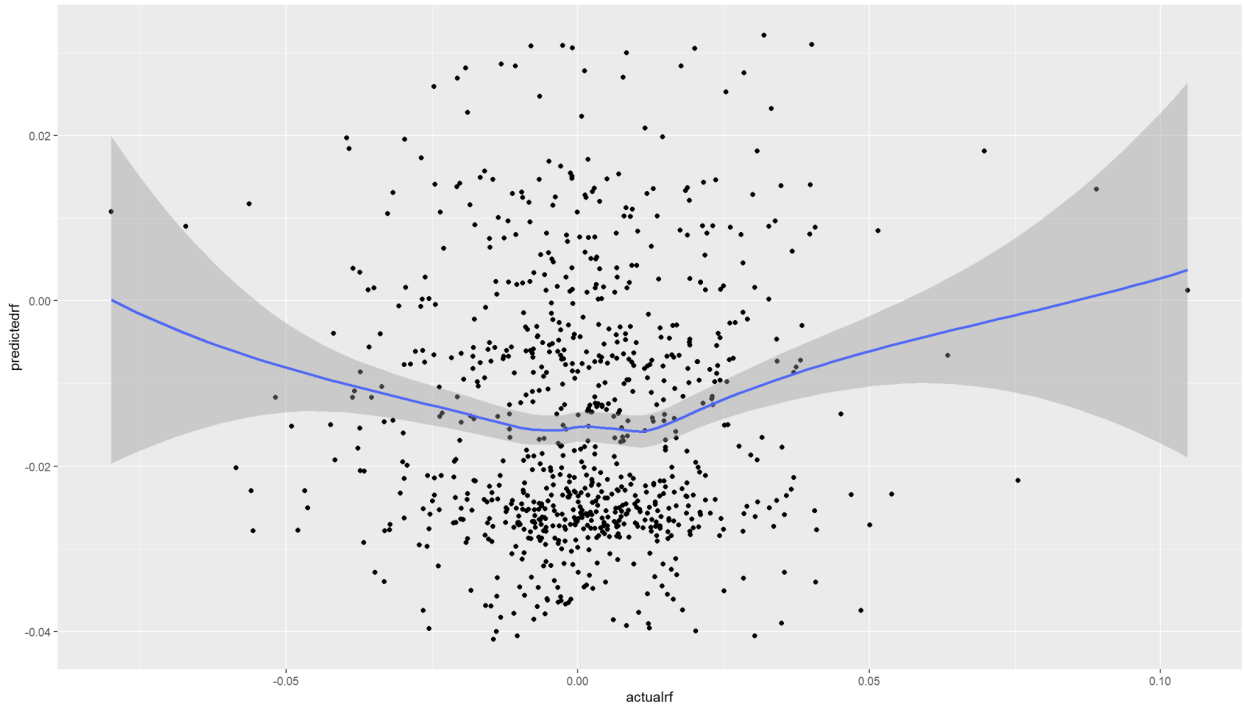


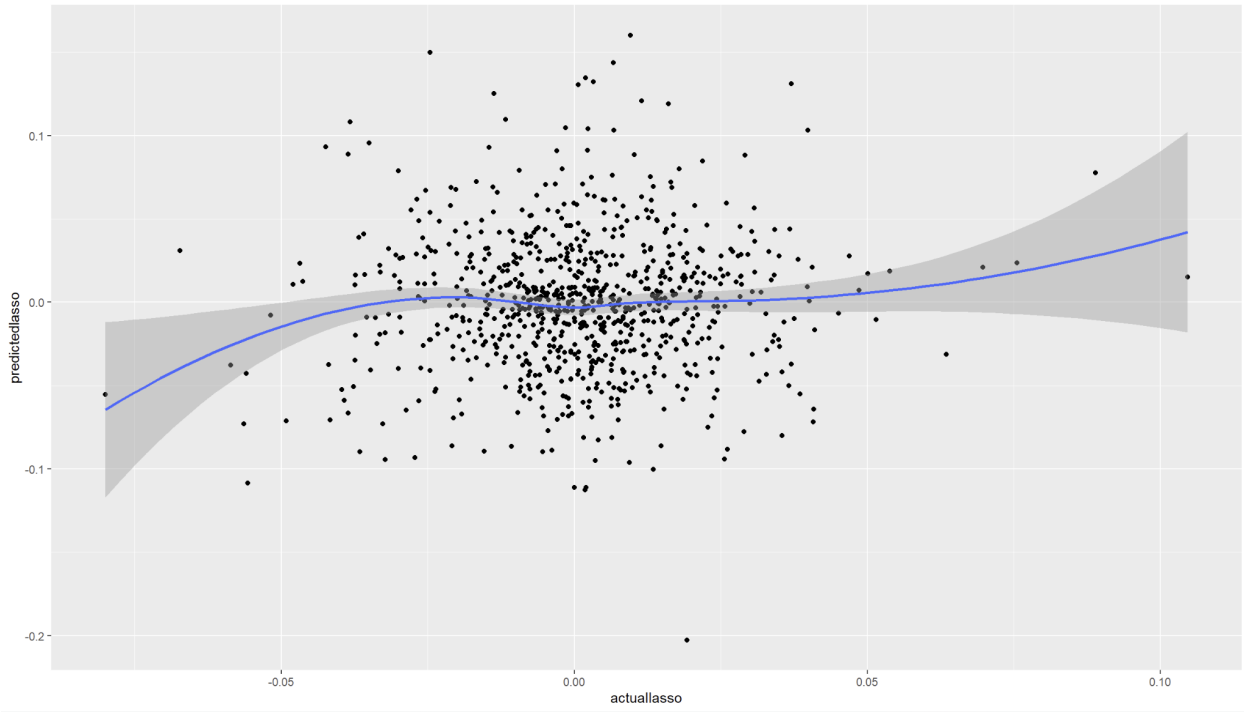

Column Sample by Tree
1
0.9
0.8
0.7
0.6
0.6
0.7
0.8
0.9
1
Subsample
RMSE
0.0206
0.0205

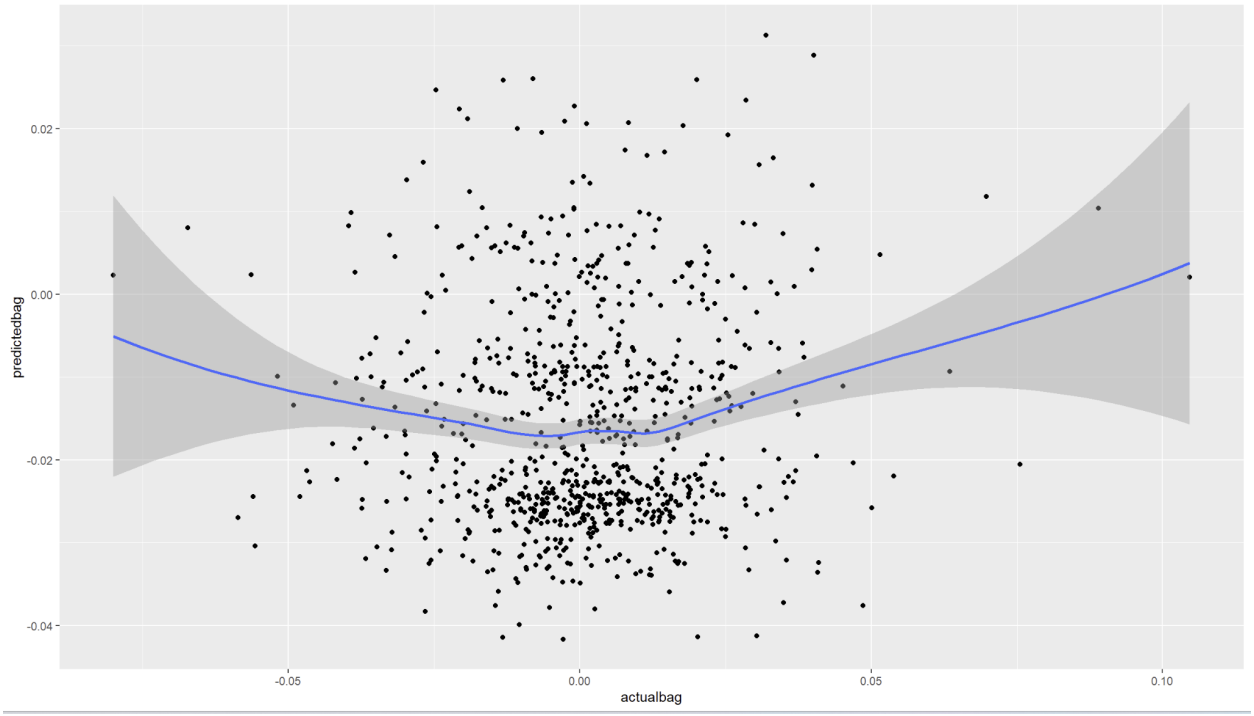

macd 0.001
obv 0.001
sd7 0.001
sp500_daily_return 0.001
sd14 0.001
sp500_rsi 0.001
daily_return 0.001
rsi 0.001
beta 0.001
sd30 0.001
-0.06
-0.03
0.00
0.03
0.06
SHAP value (impact on model output)
Feature value
Low
High

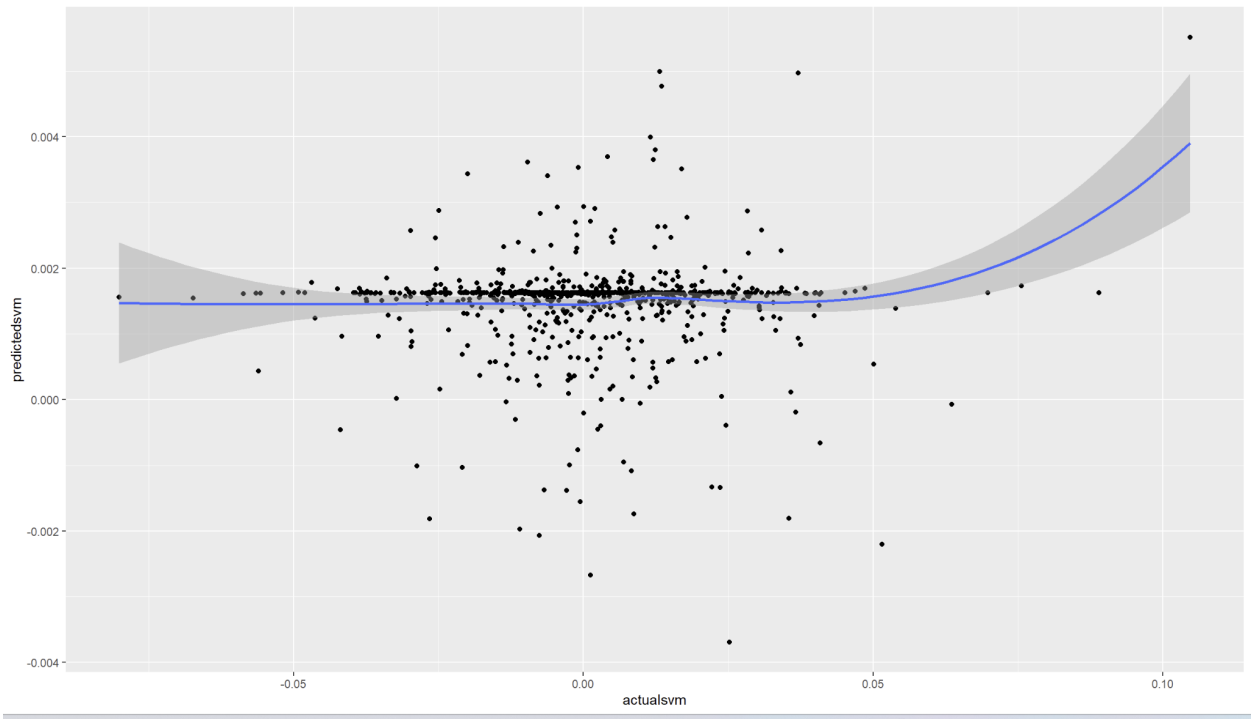

sp500_rsi
sp500_daily_return
daily_return
macd
rsi
signal
sd7
sd14
beta
obv
0.00
0.02
0.04
0.06
0.08

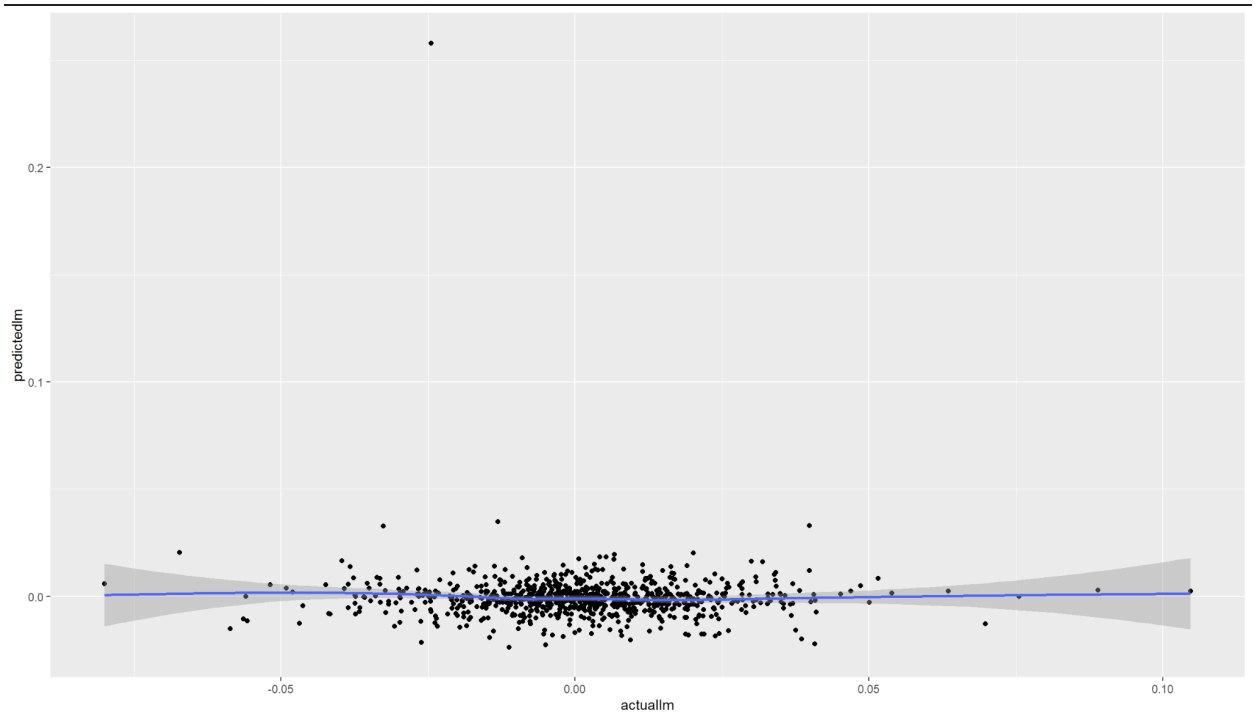

0.025
0.000
-0.025
-0.050
-0.075
predictedxgb
-0.05
0.00
0.05
0.10
actualxgb


predictedrf
0.02
0.00
-0.02
-0.04
-0.05
0.00
0.05
0.10
actualrf


predictedlasso
0.1
0.0
-0.1
-0.2
-0.05
0.00
0.05
0.10
actuallasso

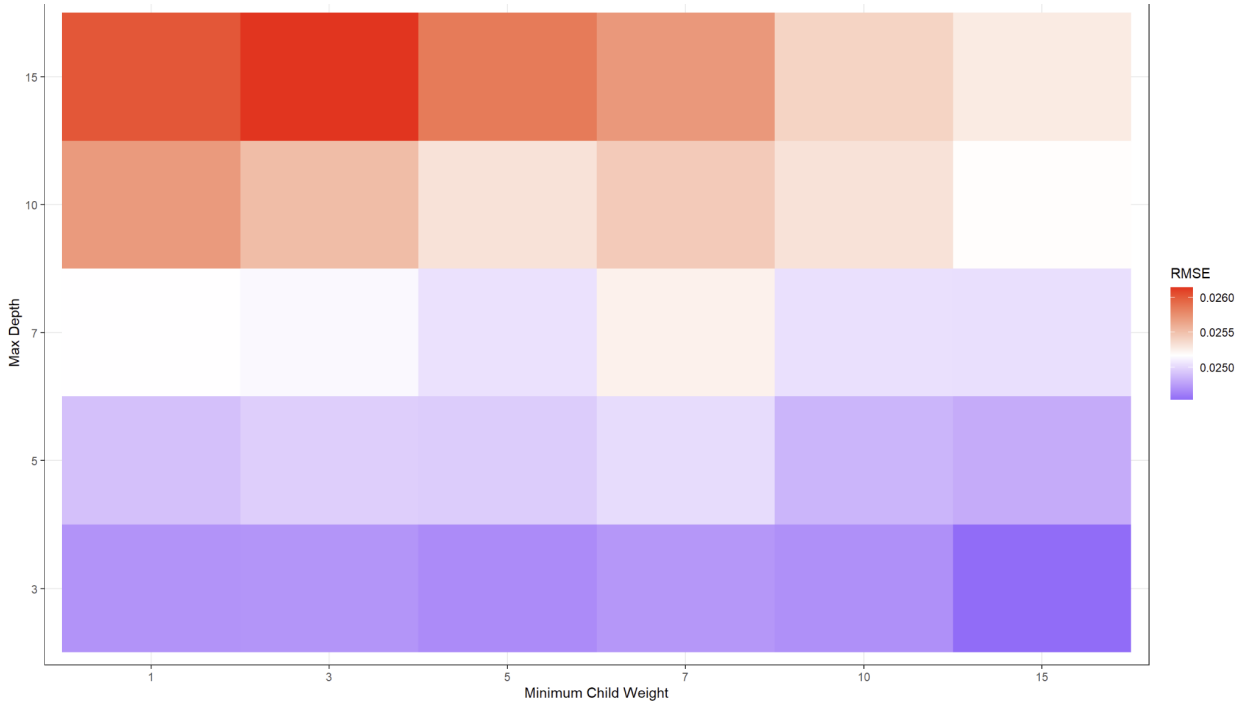

predictedbag
0.02
0.00
-0.02
-0.04
-0.05
0.00
0.05
0.10
actualbag

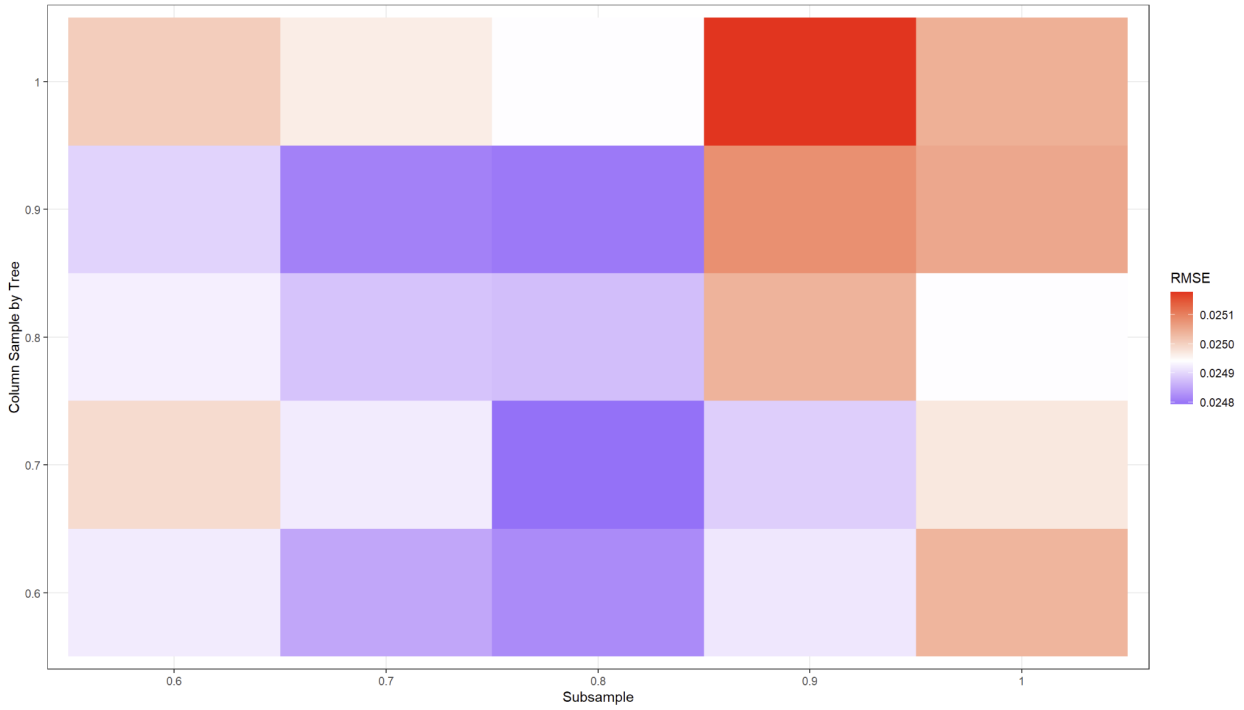

predictedsvm
0.004
0.002
0.000
-0.002
-0.004
-0.05
0.00
0.05
0.10
actualsvm

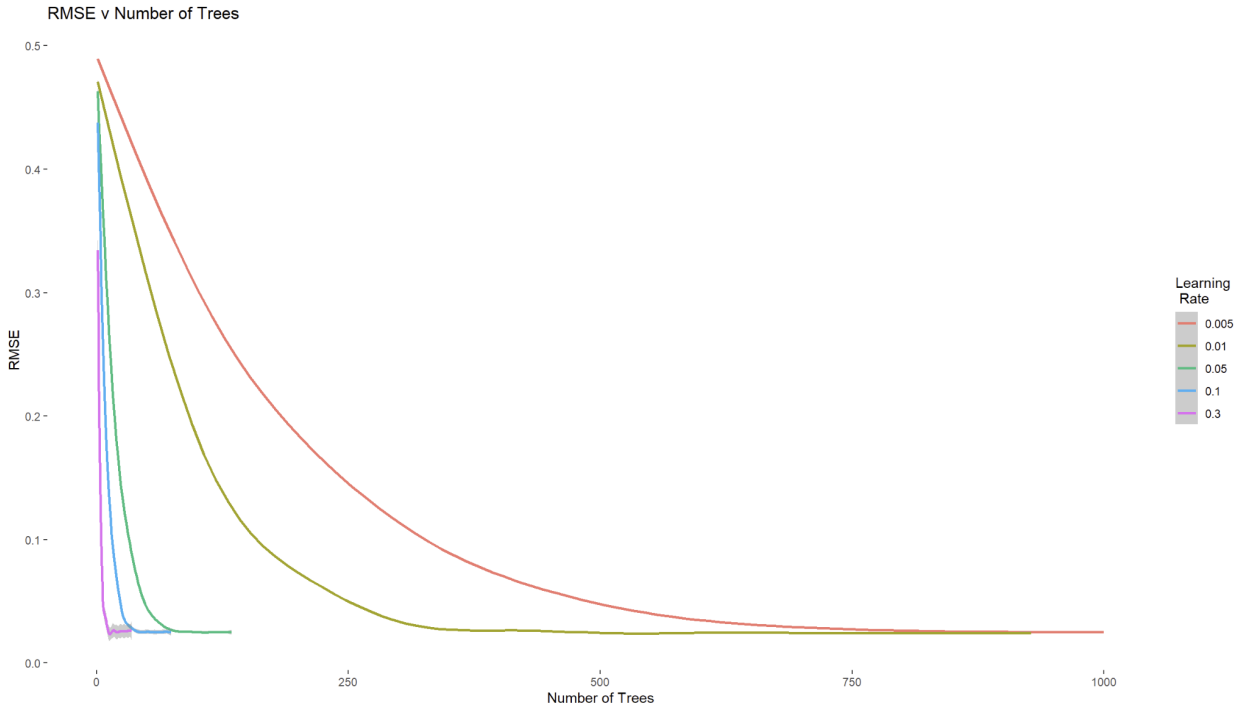

predictedlm
0.2
0.1
0.0
-0.05
0.00
0.05
0.10
actuallm

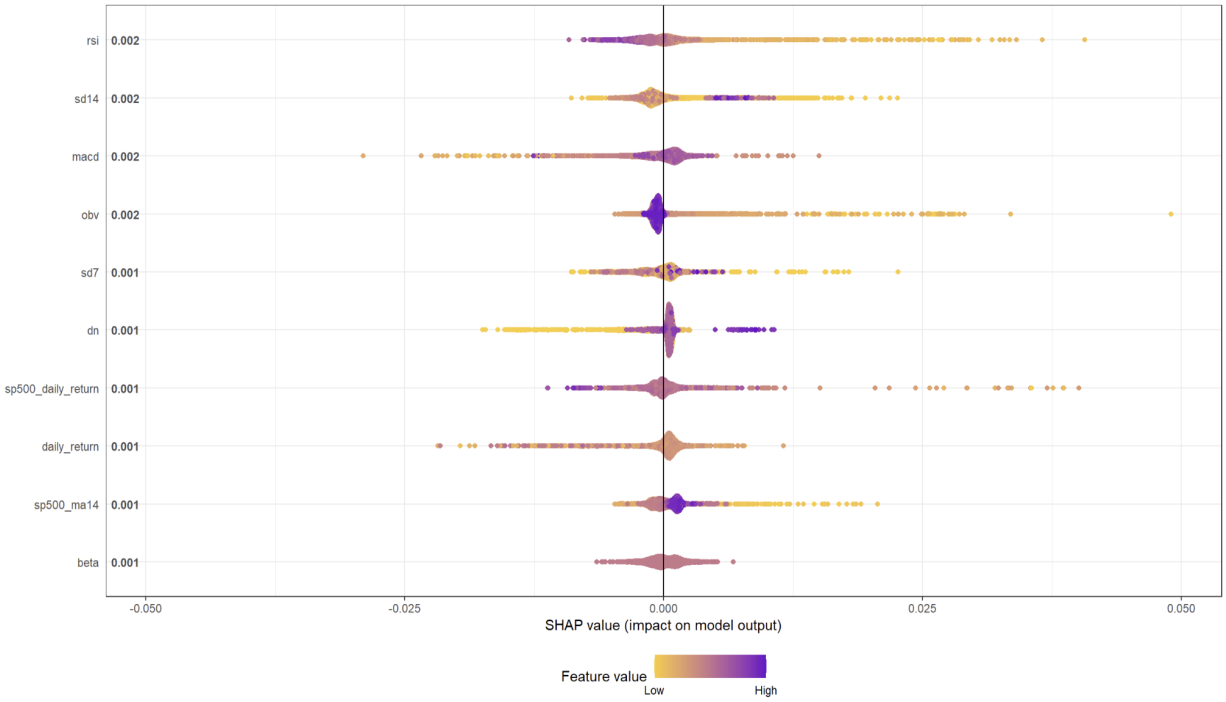
AMZN

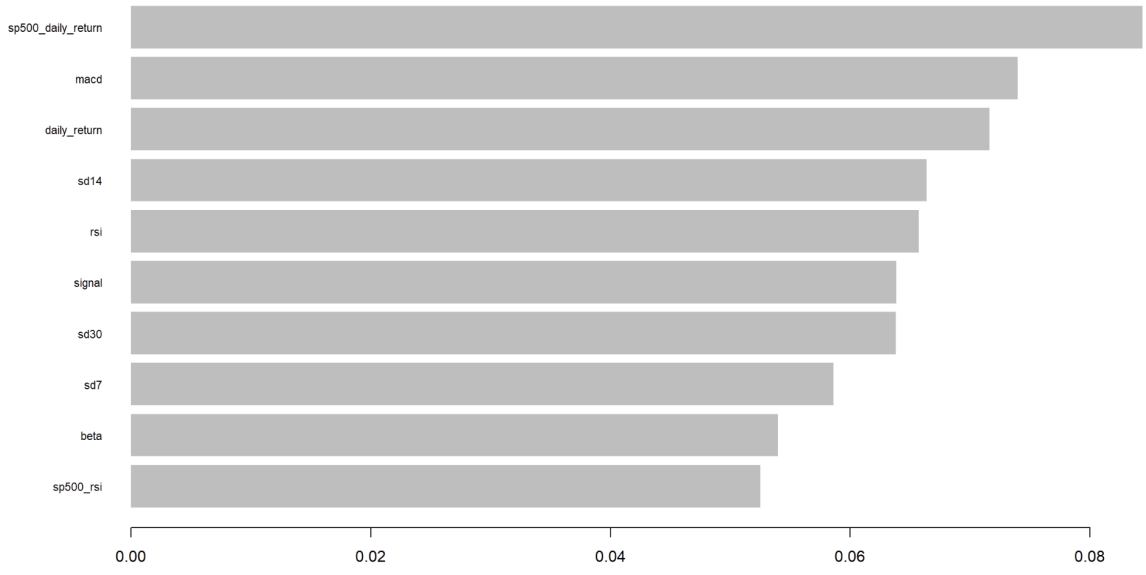
```
eta = 0.05,
 # maybe 0.3 as it learns quickest? or 0.1 as it is a happy compromise?
 # Set learning rate
 max.depth = 3,
 # Set max depth
 min_child_weight = 15,
 # Set minimum number of samples in node to split
 gamma = 0,
 # Set minimum loss reduction for split
 subsample = 0.8,
 # Set proportion of training data to use in tree
 colsample_bytree = 0.7,
 # Set number of variables to use in each tree
```

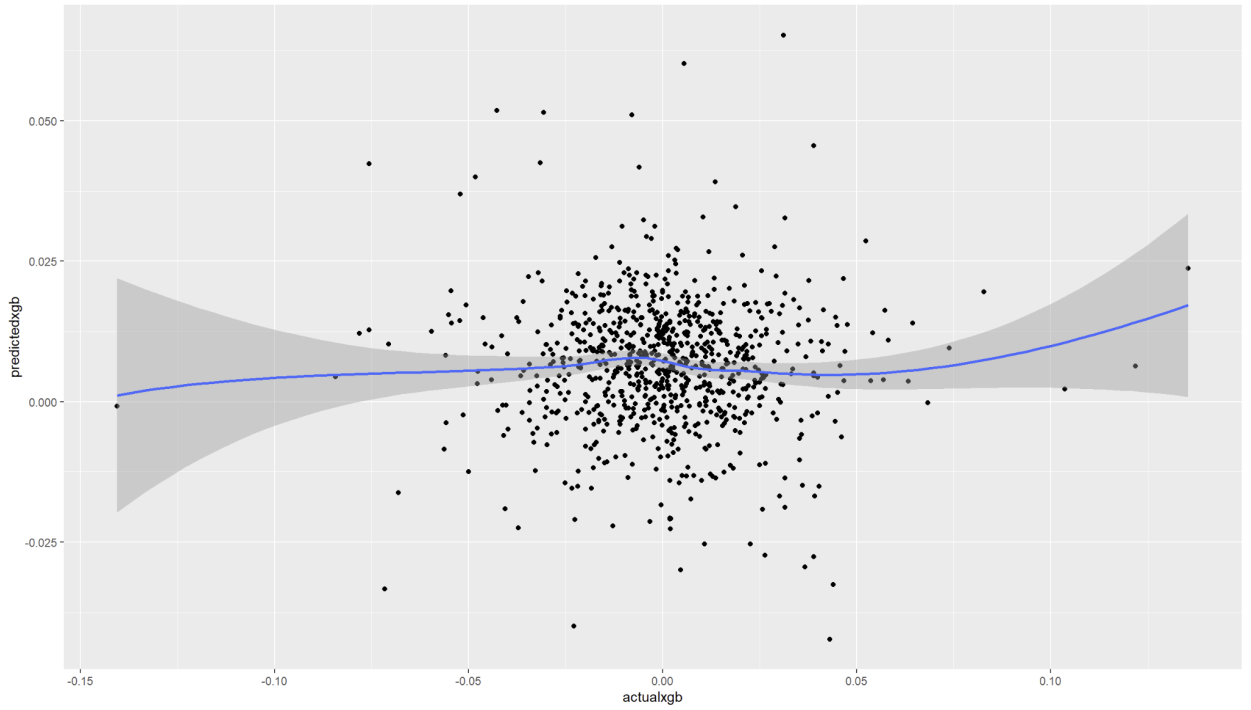


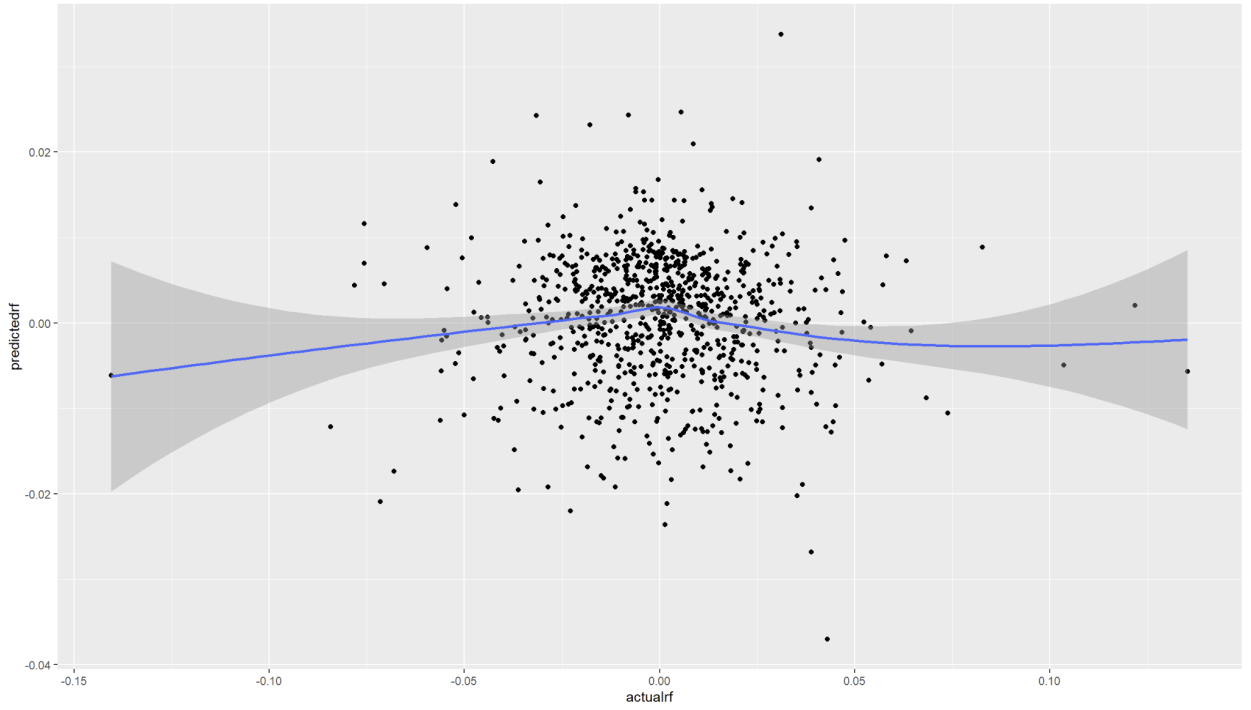

Column Sample by Tree
1
0.9
0.8
0.7
0.6
0.6
0.7
0.8
0.9
1
Subsample
RMSE
0.0251
0.0250
0.0249
0.0248

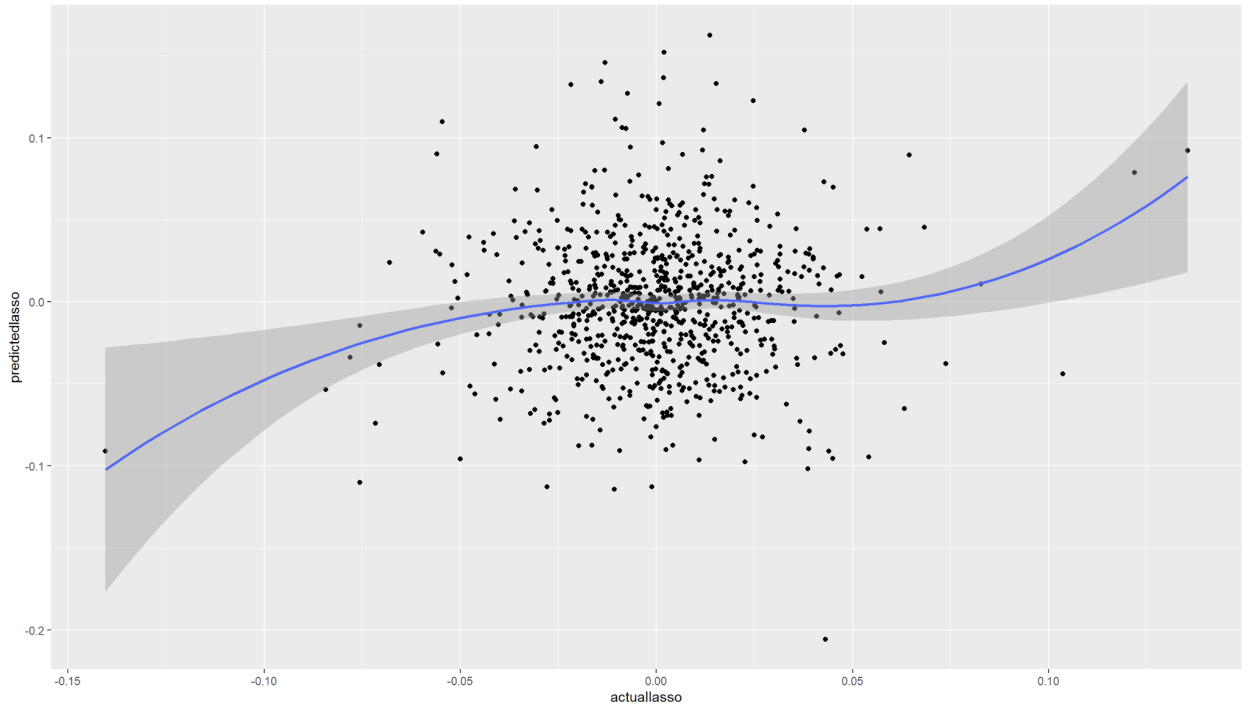

RMSE v Number of Trees
RMSE
0.5
0.4
0.3
0.2
0.1
0.0
0
250
500
750
1000
Number of Trees
Learning Rate
0.005
0.01
0.05
0.1
0.3

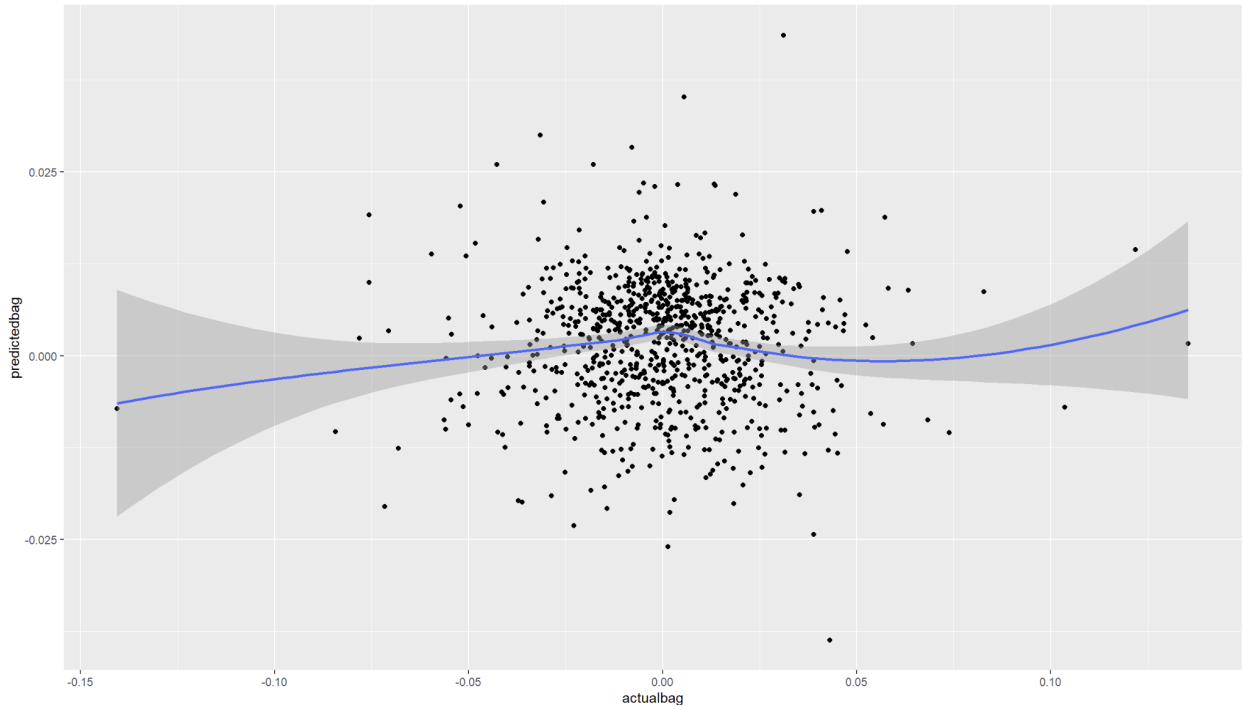

rsi 0.002
sd14 0.002
macd 0.002
obv 0.002
sd7 0.001
dn 0.001
sp500_daily_return 0.001
daily_return 0.001
sp500_ma14 0.001
beta 0.001
-0.050
-0.025
0.000
0.025
0.050
SHAP value (impact on model output)
Feature value
Low
High

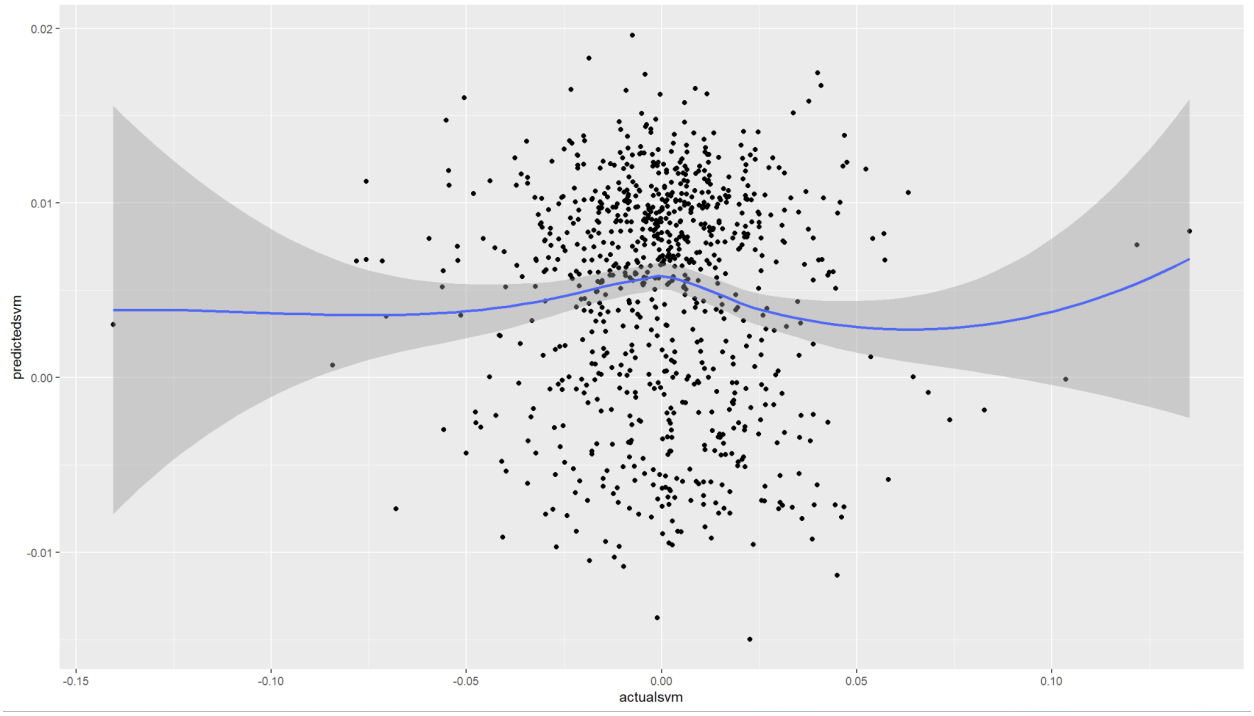

sp500_daily_return
macd
daily_return
sd14
rsi
signal
sd30
sd7
beta
sp500_rsi
0.00
0.02
0.04
0.06
0.08

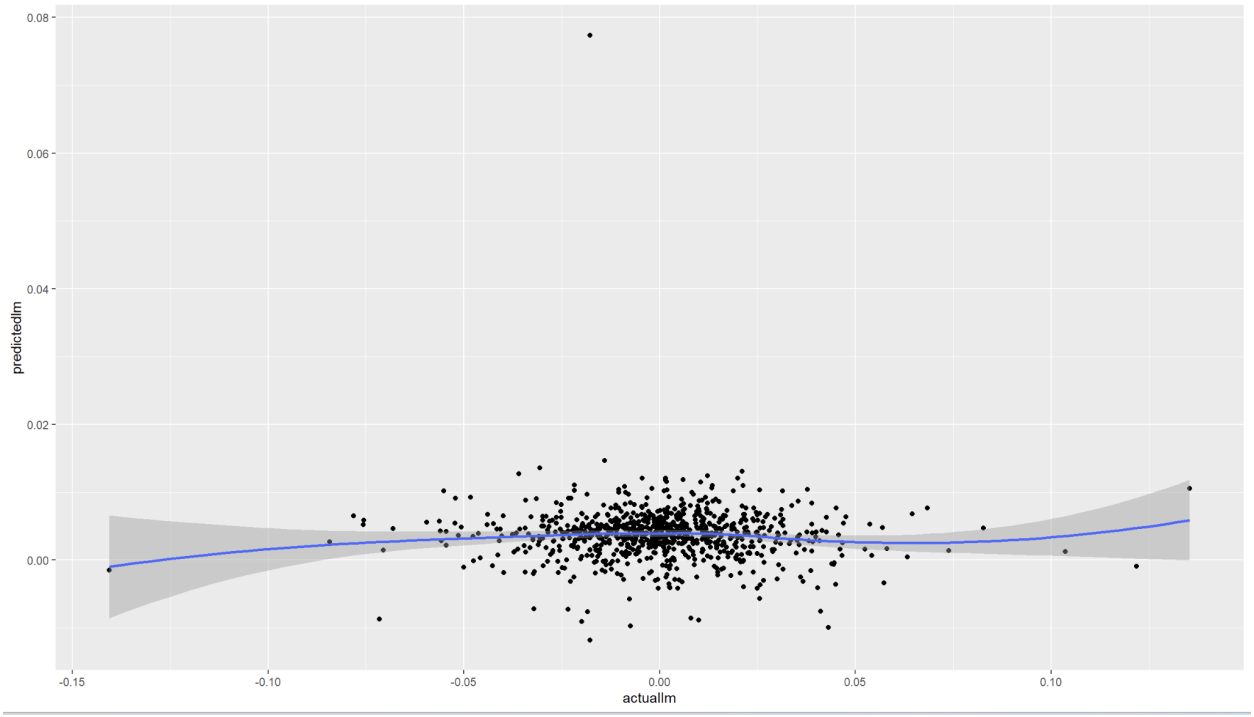

predictedxgb
0.050
0.025
0.000
-0.025
-0.15
-0.10
-0.05
0.00
0.05
0.10
actualxgb


predictedrf
0.02
0.00
-0.02
-0.04
-0.15
-0.10
-0.05
0.00
0.05
0.10
actualrf


predictedlasso
0.1
0.0
-0.1
-0.2
-0.15
-0.10
-0.05
0.00
0.05
0.10
actuallasso

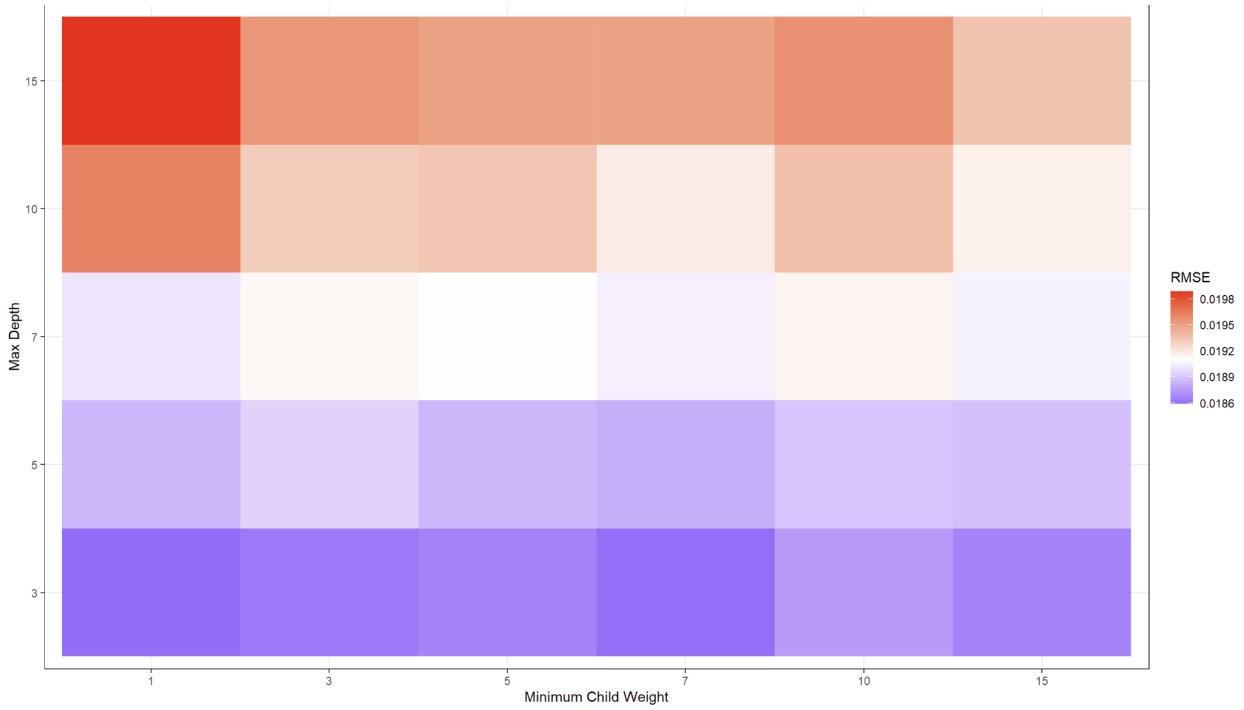

predictedbag
0.025
0.000
-0.025
-0.15
-0.10
-0.05
0.00
0.05
0.10
actualbag

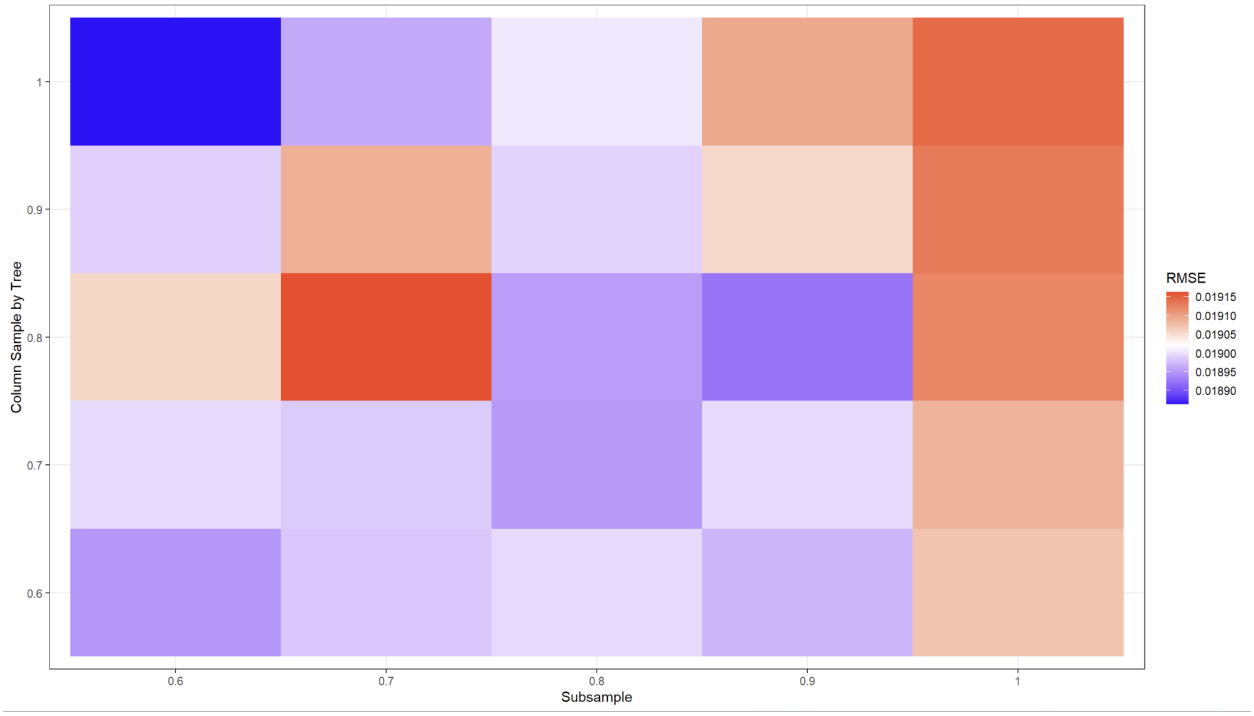

0.02
0.01
0.00
-0.01
predictedsvm
-0.15
-0.10
-0.05
0.00
0.05
0.10
actualsvm

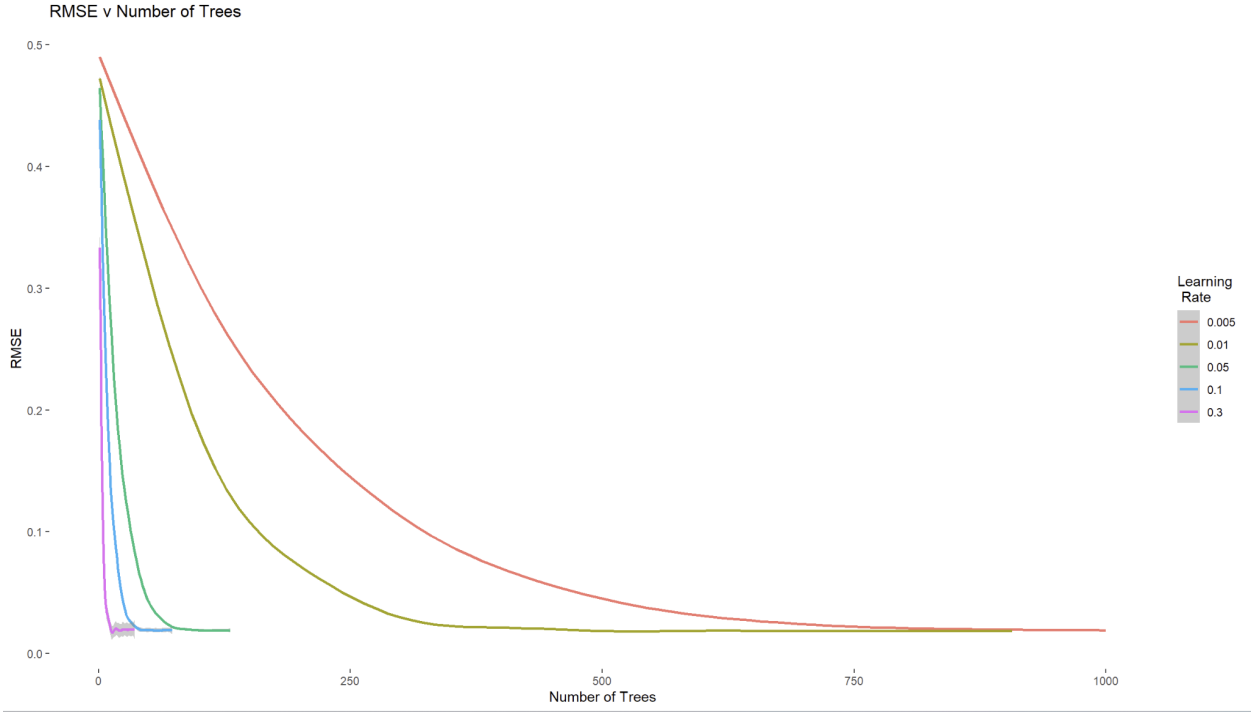

0.08
0.06
0.04
0.02
0.00
predictedlm
-0.15
-0.10
-0.05
0.00
0.05
0.10
actuallm

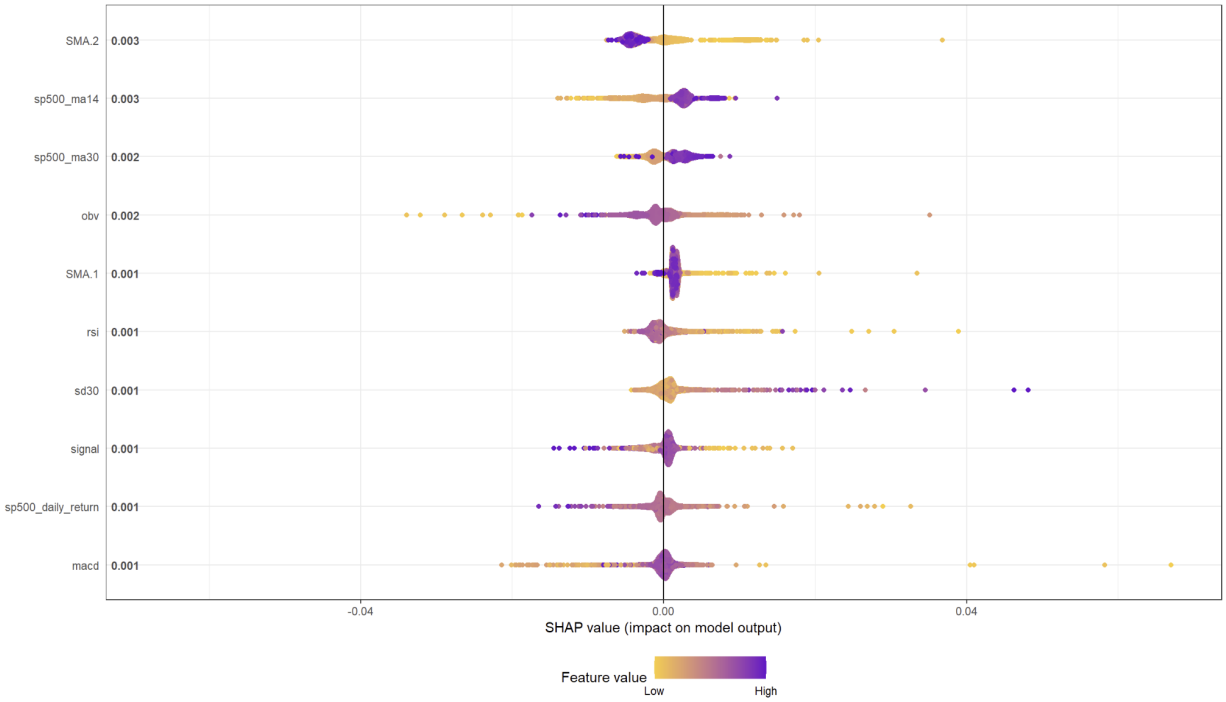
GOOG

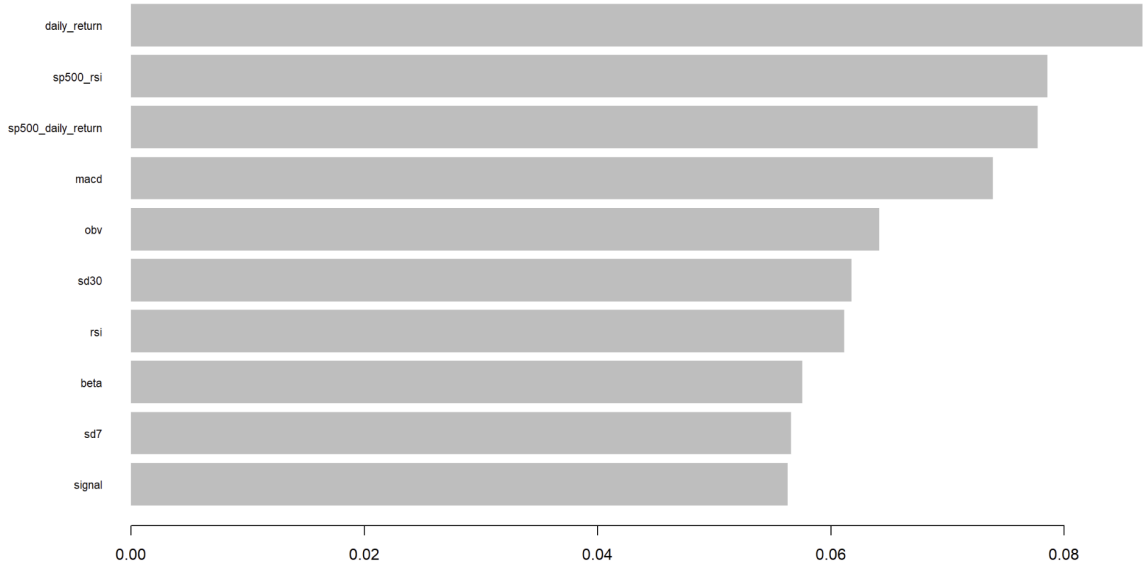
```
eta = 0.05,
 # maybe 0.3 as it learns quickest? or 0.1 as it is a happy compromise?
 # Set learning rate
 max.depth = 3,
 # Set max depth
 min_child_weight = 1,
 # Set minimum number of samples in node to split
 gamma = 0,
 # Set minimum loss reduction for split
 subsample = 0.6,
 # Set proportion of training data to use in tree
 colsample_bytree = 1,
 # Set number of variables to use in each tree
```

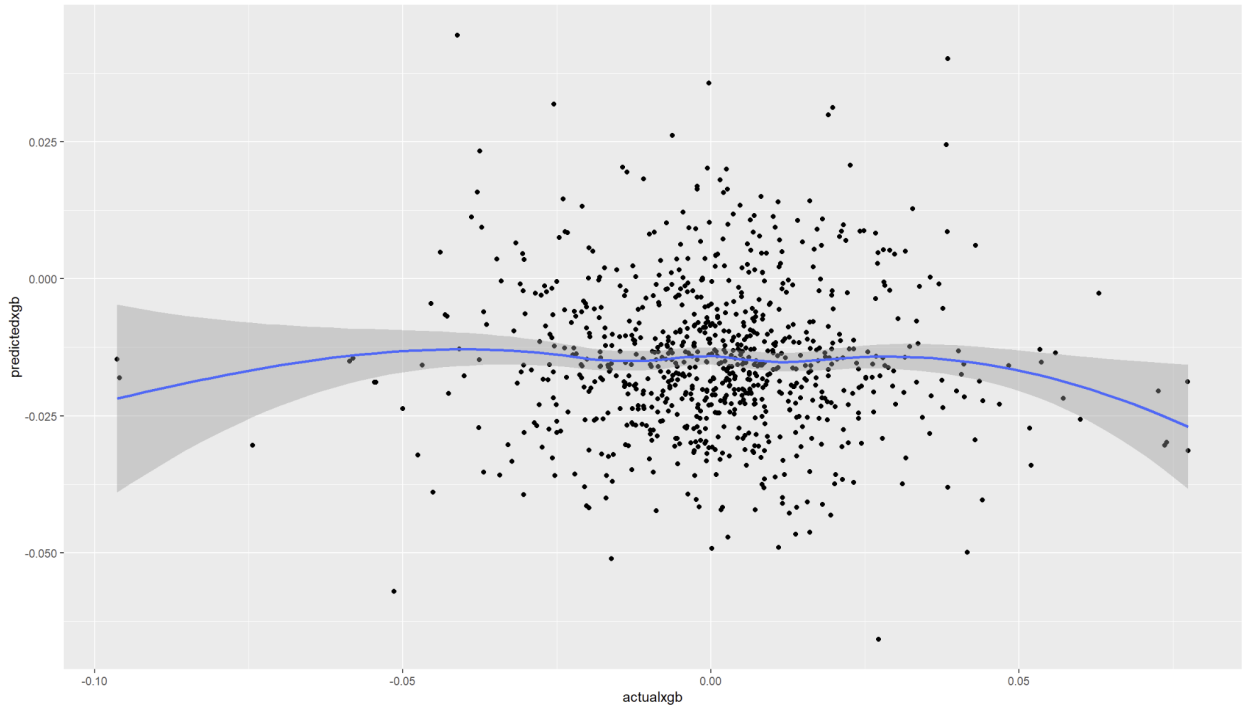


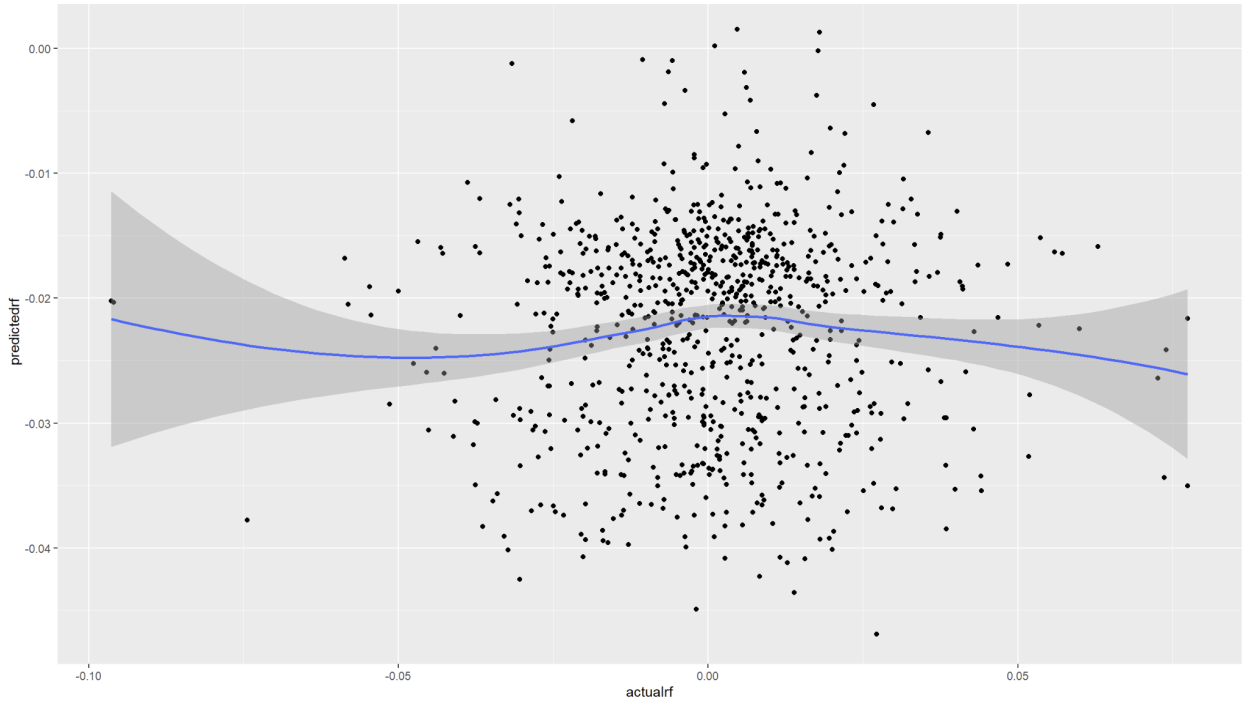

Column Sample by Tree
1
0.9
0.8
0.7
0.6
0.6
0.7
0.8
0.9
1
Subsample
RMSE
0.01915
0.01910
0.01905
0.01900
0.01895
0.01890

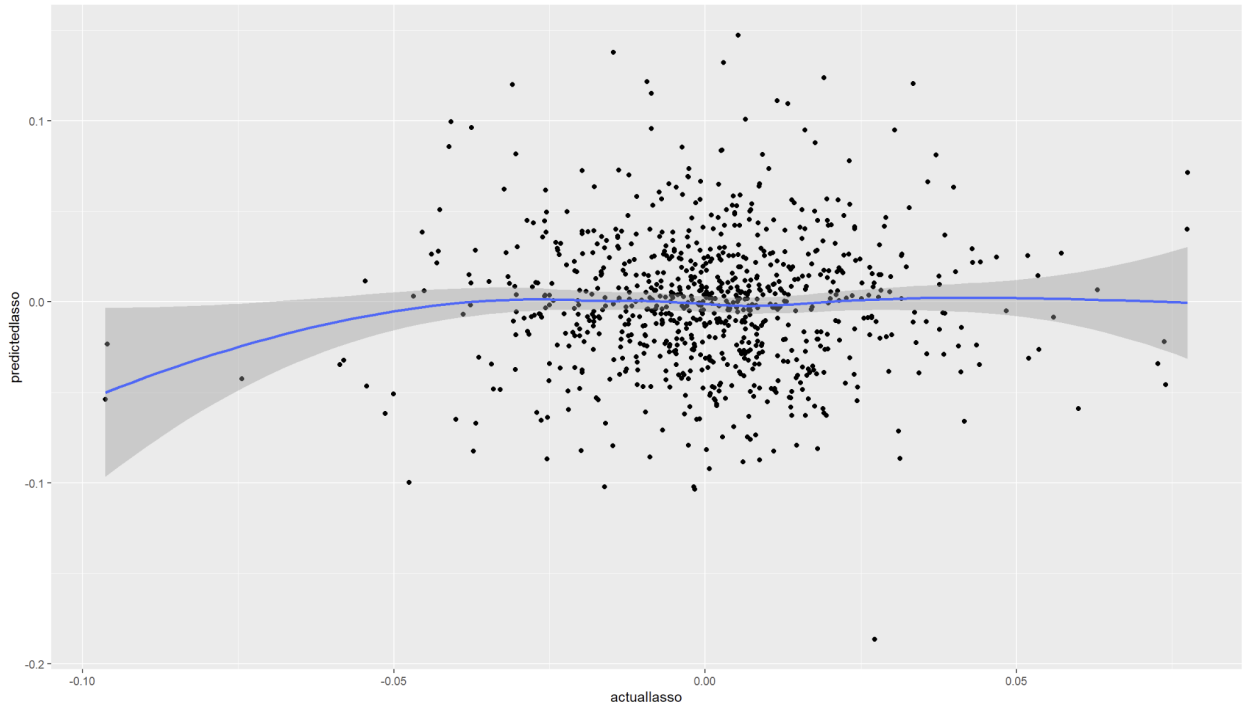

RMSE v Number of Trees
RMSE
0.5
0.4
0.3
0.2
0.1
0.0
0
250
500
750
1000
Number of Trees
Learning Rate
0.005
0.01
0.05
0.1
0.3

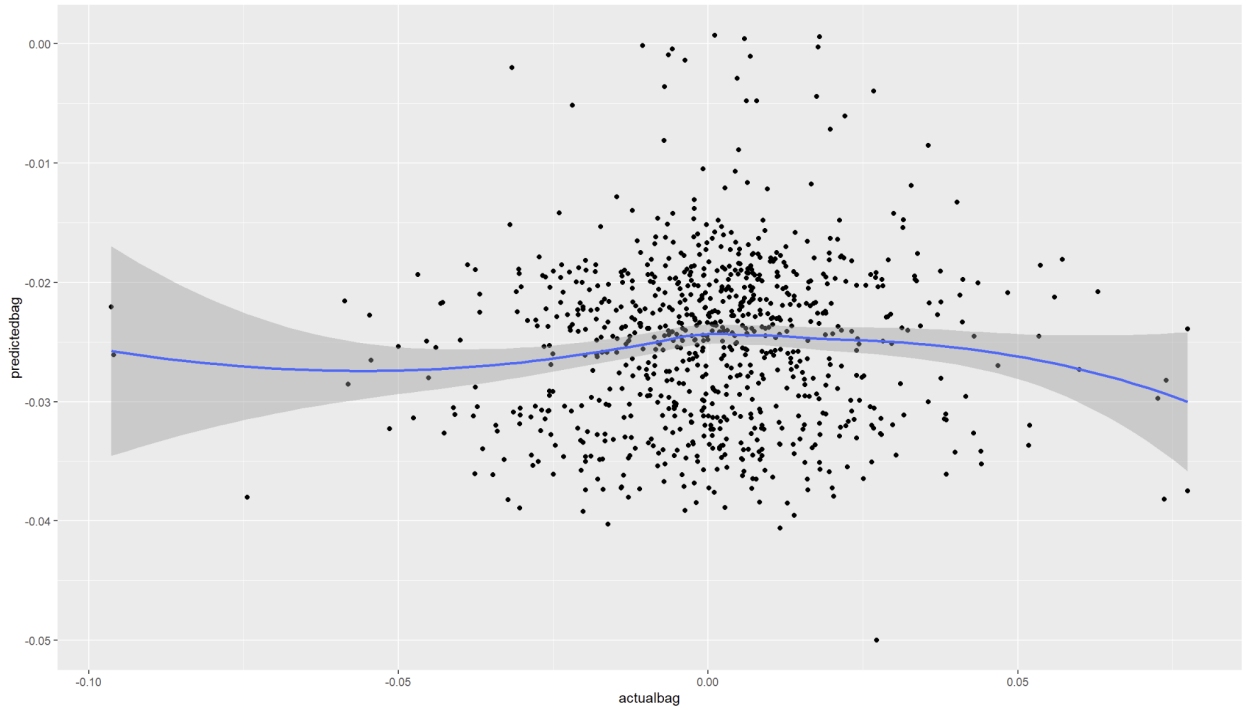

SMA.2 0.003
sp500_ma14 0.003
sp500_ma30 0.002
obv 0.002
SMA.1 0.001
rsi 0.001
sd30 0.001
signal 0.001
sp500_daily_return 0.001
macd 0.001
-0.04
0.00
0.04
SHAP value (impact on model output)
Feature value
Low
High

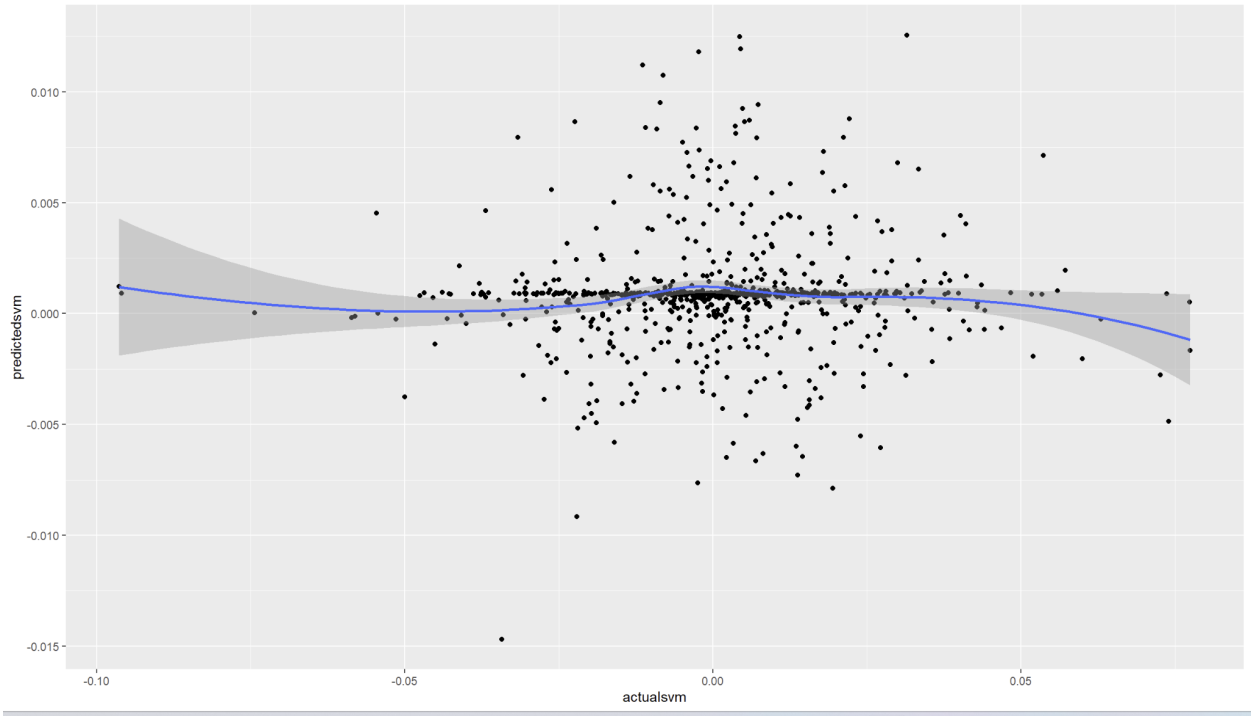

daily_return
sp500_rsi
sp500_daily_return
macd
obv
sd30
rsi
beta
sd7
signal
0.00
0.02
0.04
0.06
0.08

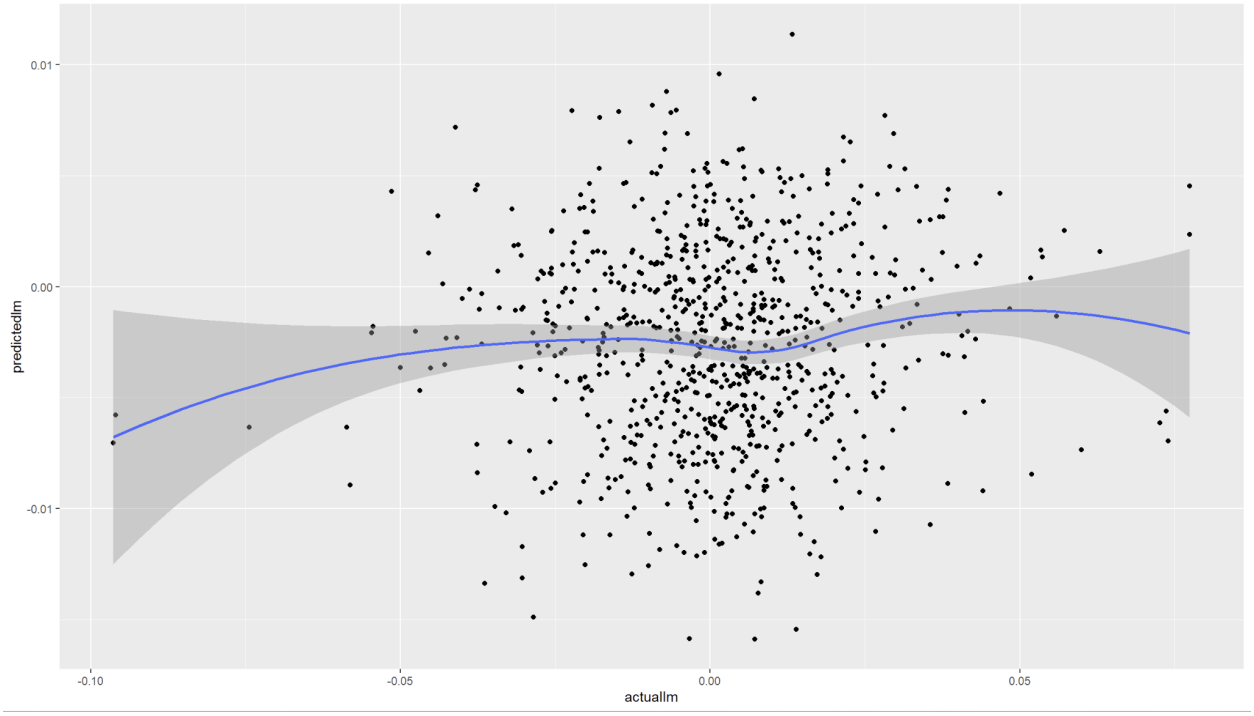

predictedxgb
0.025
0.000
-0.025
-0.050
-0.10
-0.05
0.00
0.05
actualxgb


predictedrf
0.00
-0.01
-0.02
-0.03
-0.04
-0.10
-0.05
0.00
0.05
actualrf


predictedlasso
0.1
0.0
-0.1
-0.2
-0.10
-0.05
0.00
0.05
actuallasso

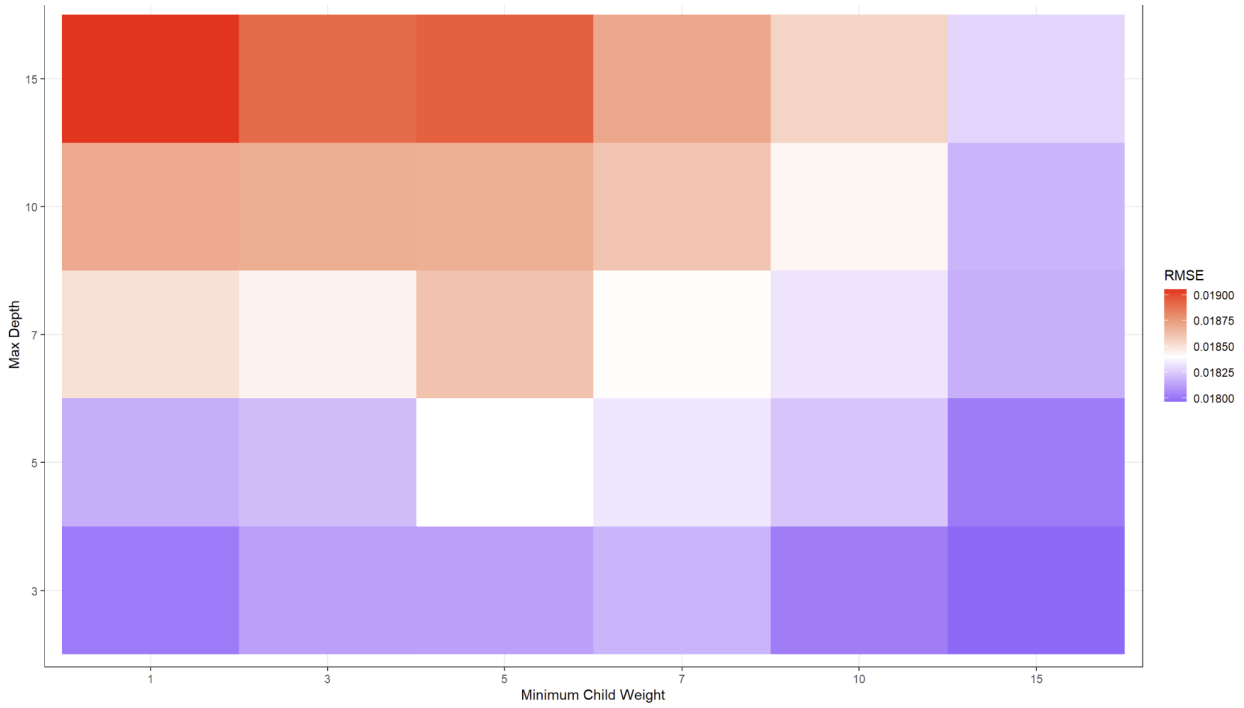

predictedbag
0.00
-0.01
-0.02
-0.03
-0.04
-0.05
-0.10
-0.05
0.00
0.05
actualbag

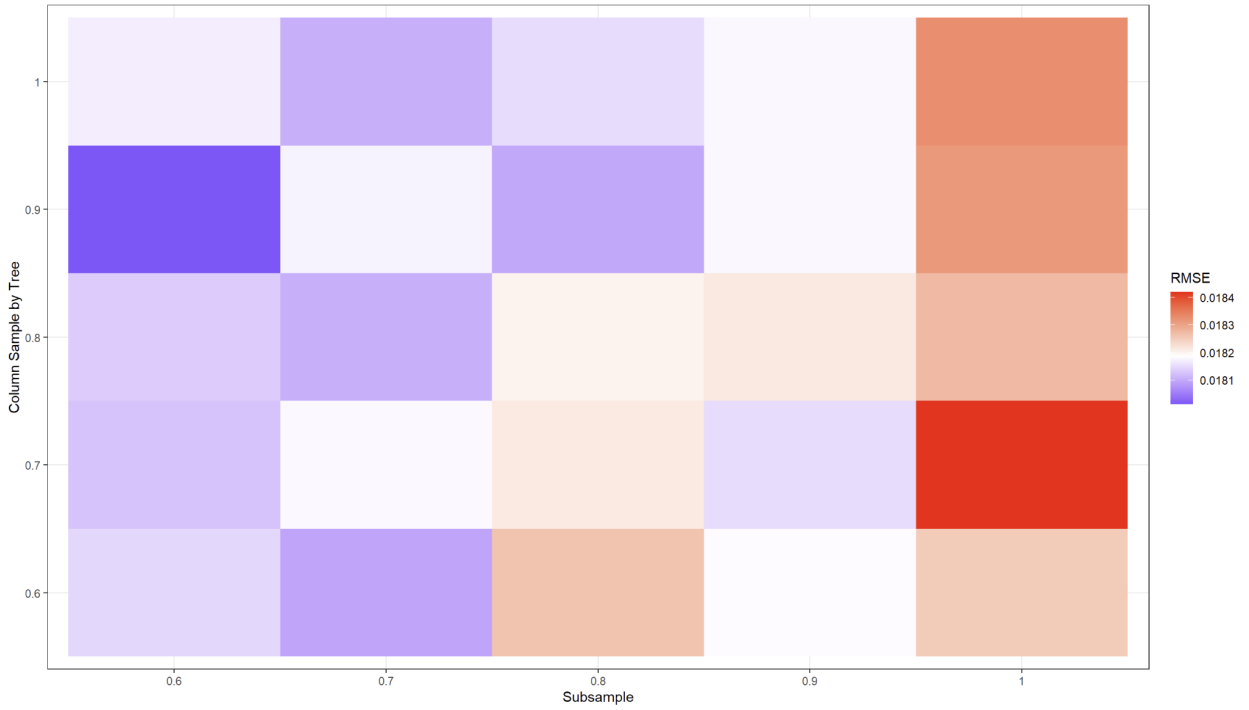

0.010
0.005
0.000
-0.005
-0.010
-0.015
predictedsvm
-0.10
-0.05
0.00
0.05
actualsvm

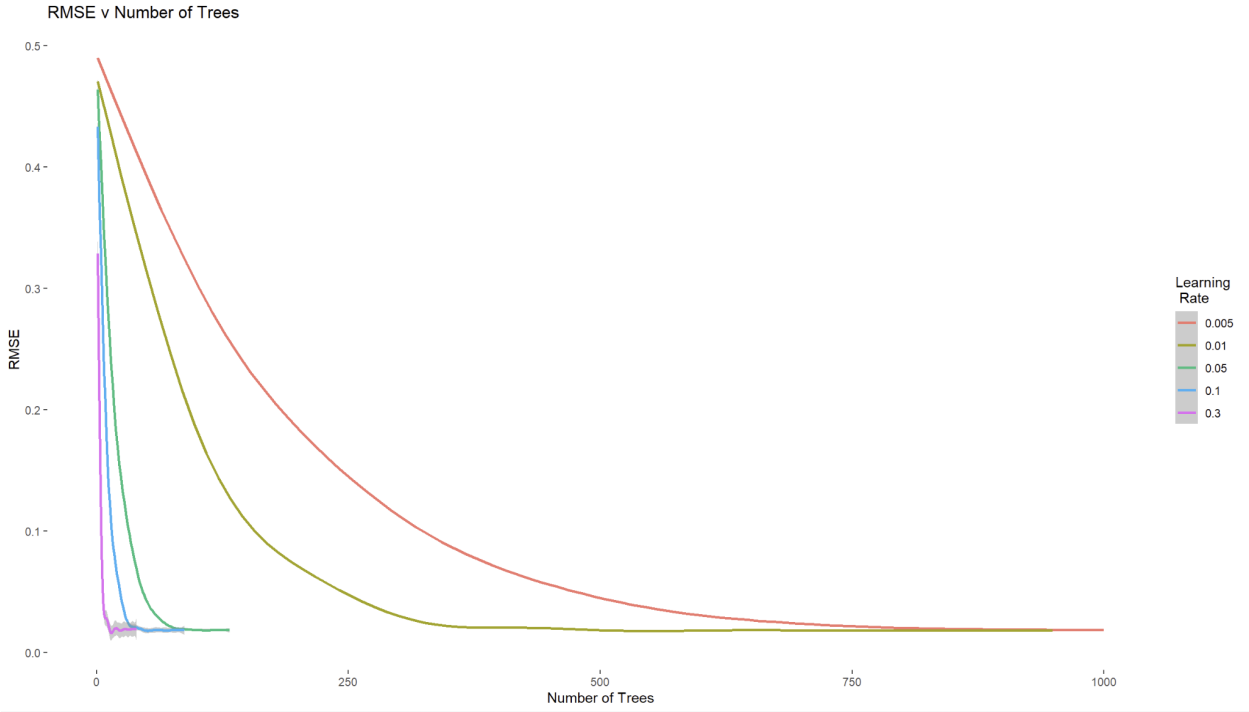

0.01
0.00
-0.01
predictedlm
-0.10
-0.05
0.00
0.05
actuallm

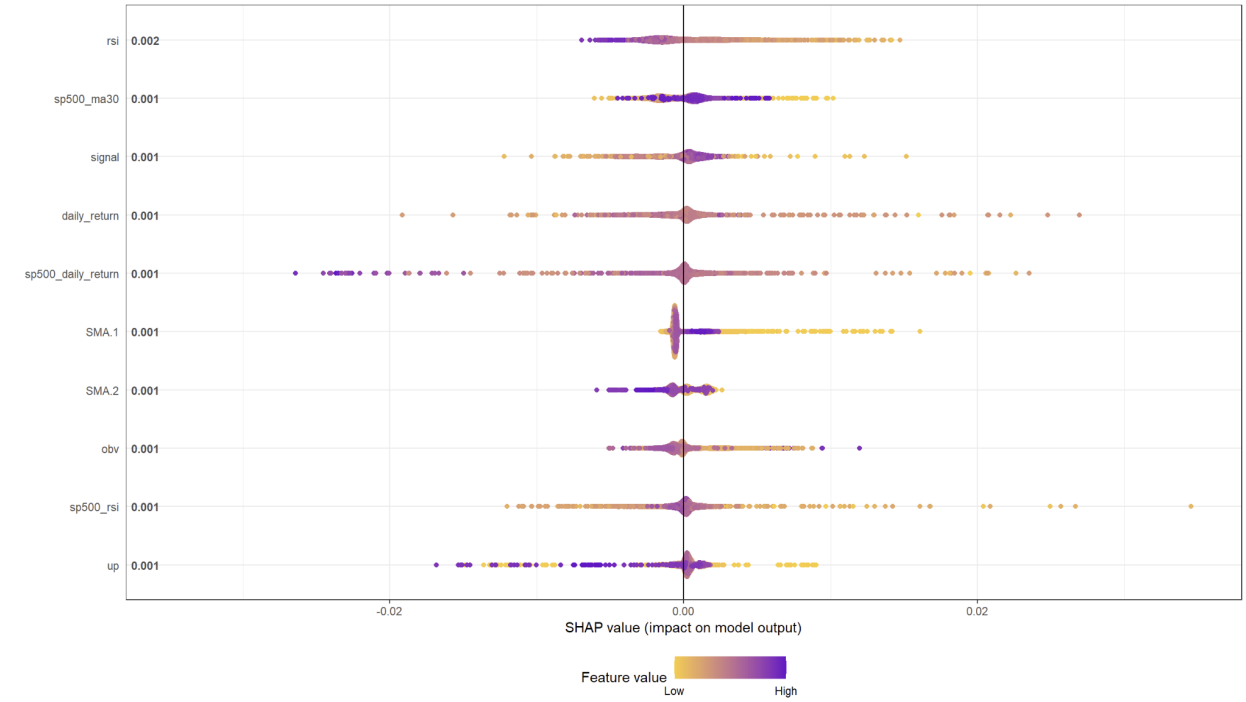
MSFT

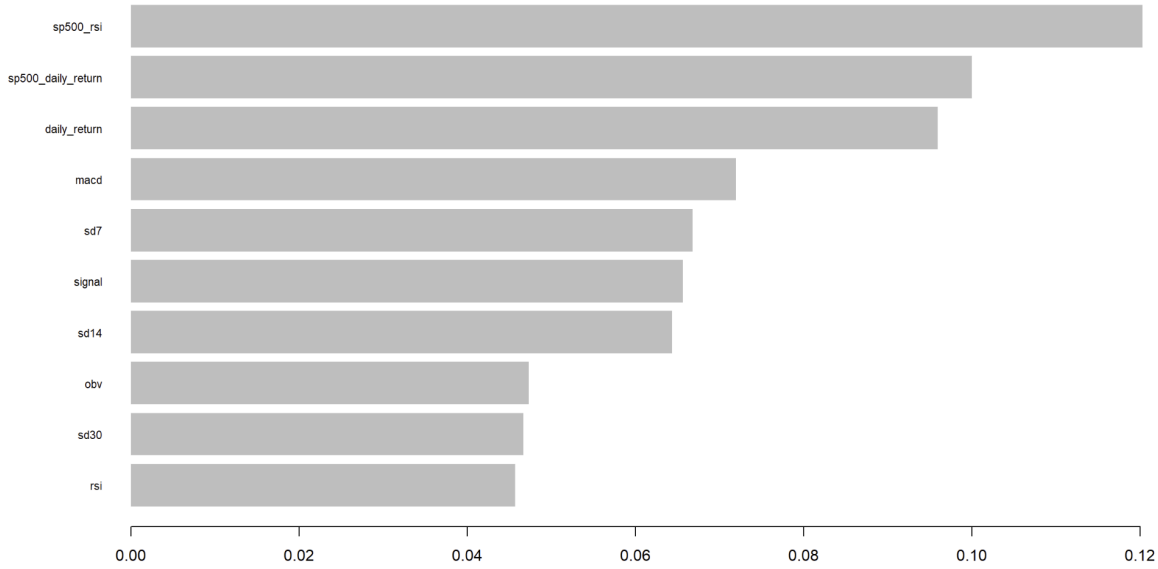
```
 eta = 0.05,
 # maybe 0.3 as it learns quickest? or 0.1 as it is a happy compromise?
 # Set learning rate
 max.depth = 3,
 # Set max depth
 min_child_weight = 15,
 # Set minimum number of samples in node to split
 gamma = 0,
 # Set minimum loss reduction for split
 subsample = 0.6,
 # Set proportion of training data to use in tree
 colsample_bytree = 0.9,
 # Set number of variables to use in each tree
```

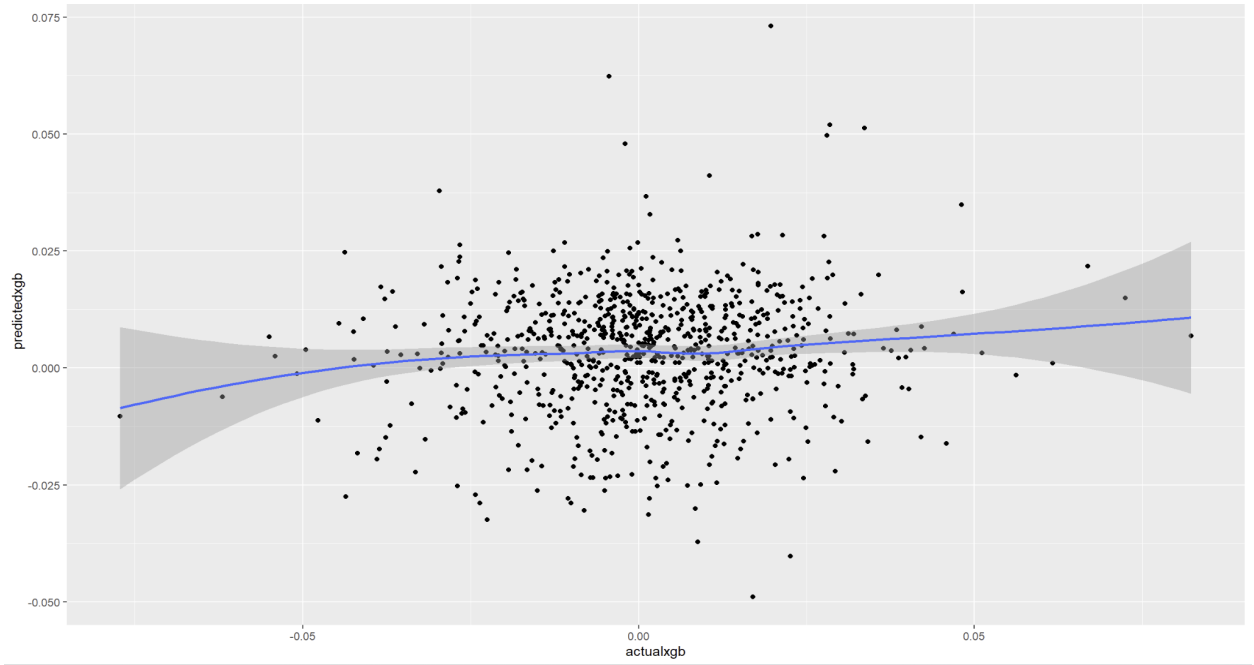


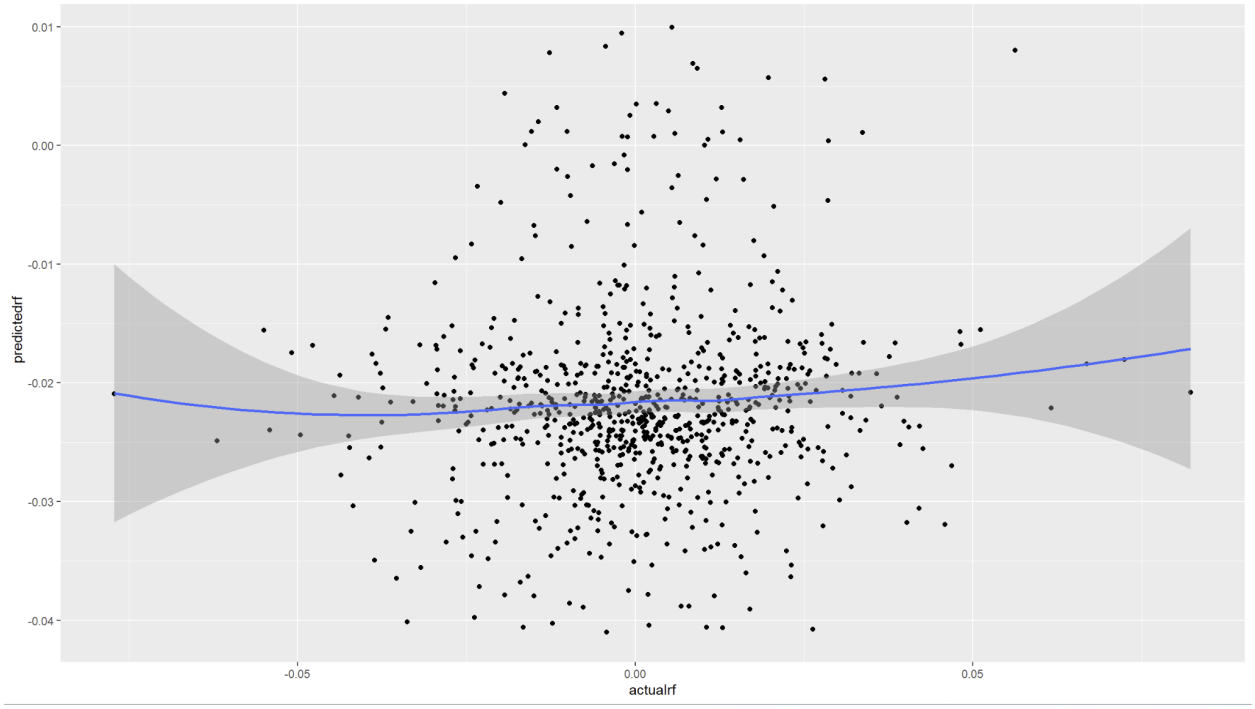

Column Sample by Tree
1
0.9
0.8
0.7
0.6
0.6
0.7
0.8
0.9
1
Subsample
RMSE
0.0184
0.0183
0.0182
0.0181

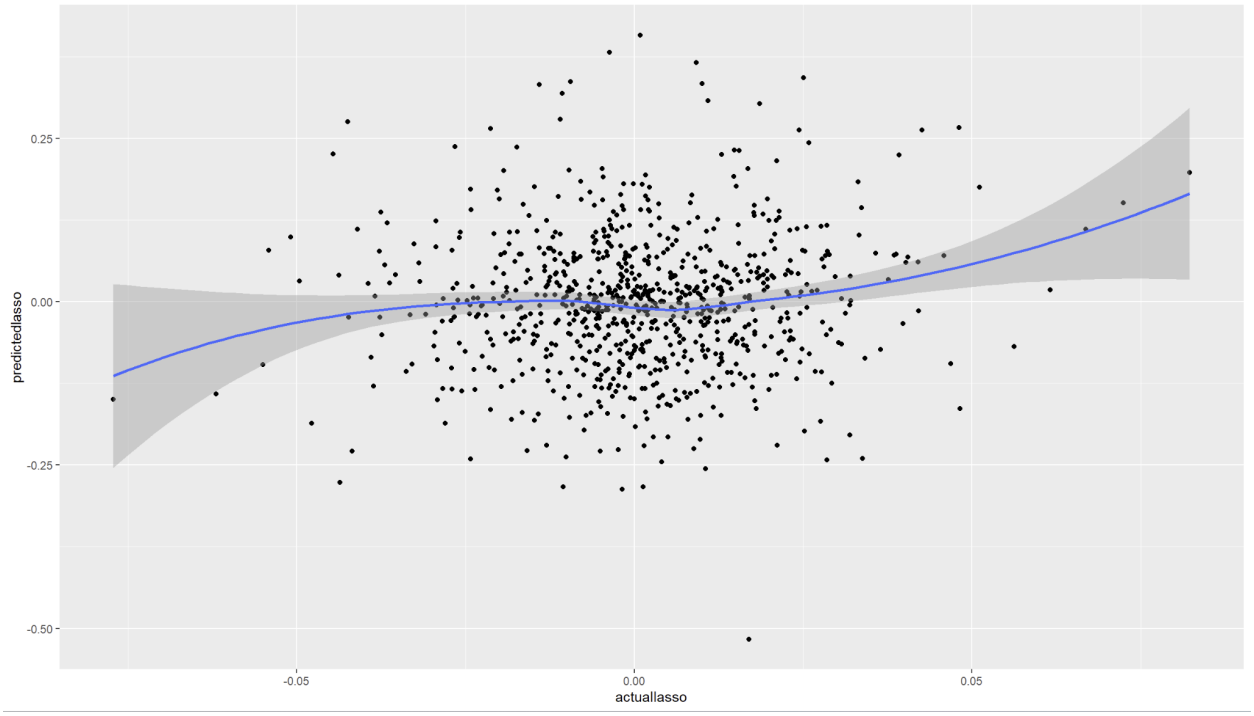

RMSE v Number of Trees
RMSE
0.5
0.4
0.3
0.2
0.1
0.0
0
250
500
750
1000
Number of Trees
Learning Rate
0.005
0.01
0.05
0.1
0.3

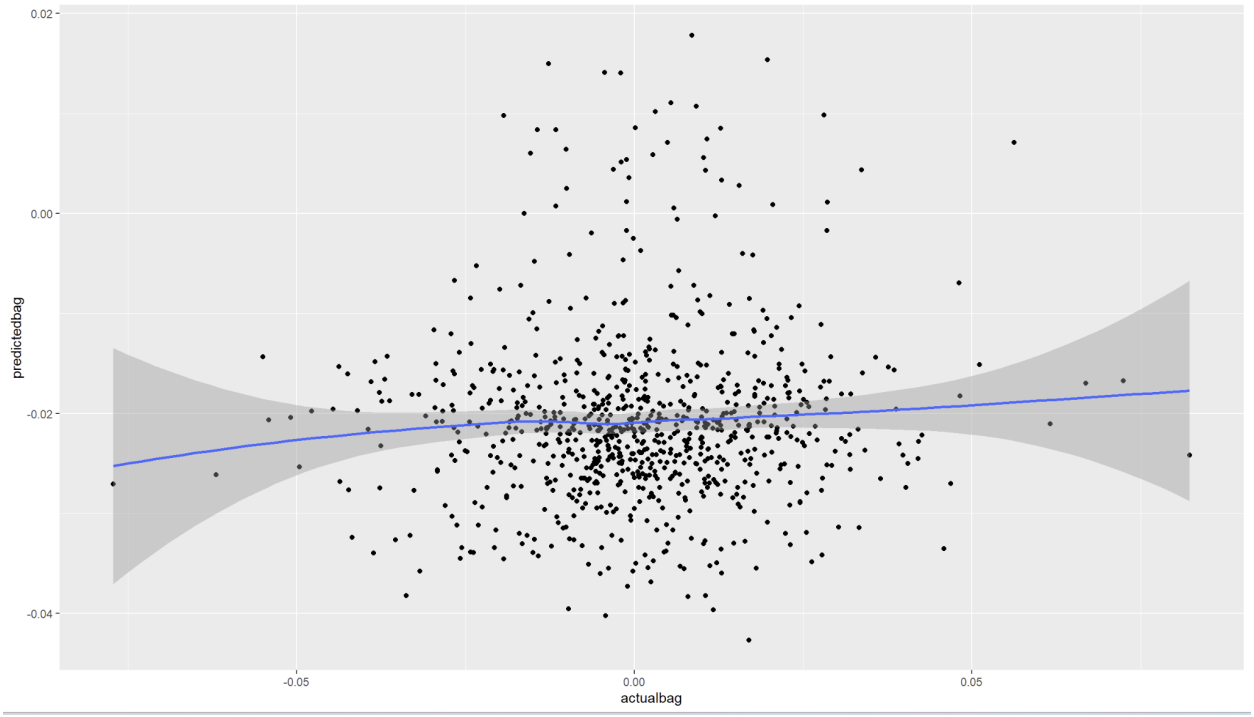

rsi 0.002
sp500_ma30 0.001
signal 0.001
daily_return 0.001
sp500_daily_return 0.001
SMA.1 0.001
SMA.2 0.001
obv 0.001
sp500_rsi 0.001
up 0.001
-0.02
0.00
0.02
SHAP value (impact on model output)
Feature value
Low
High

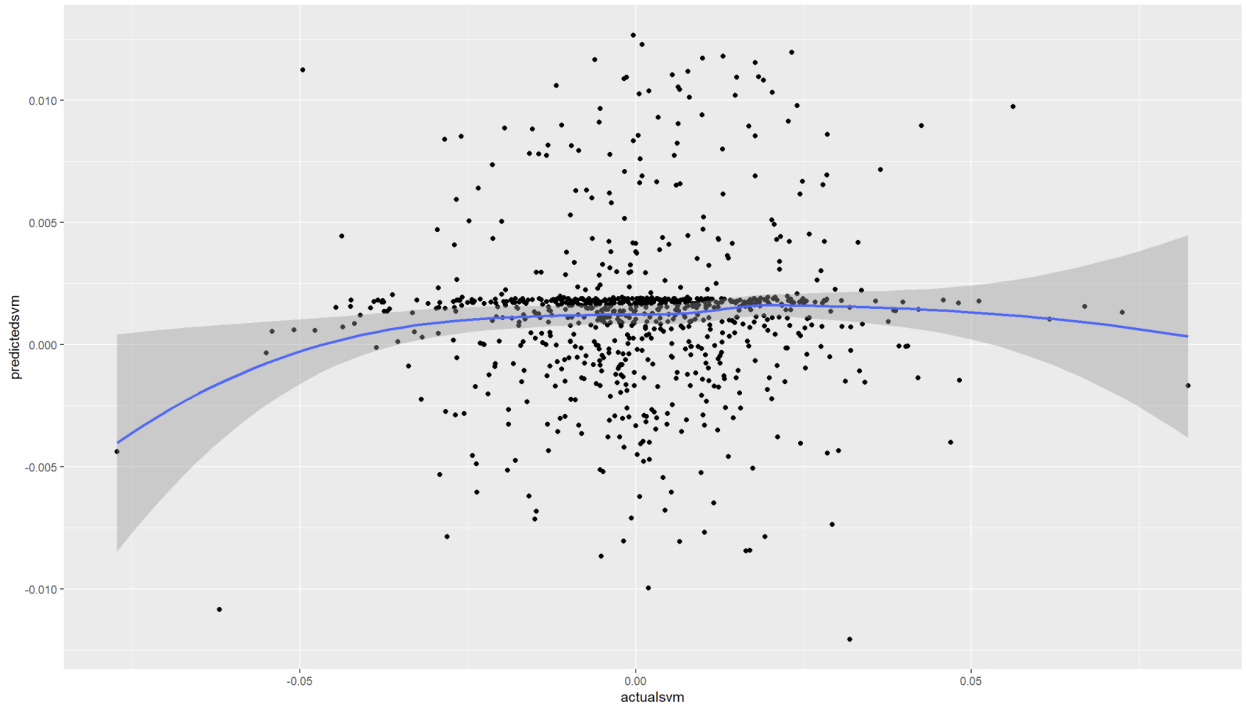

sp500_rsi
sp500_daily_return
daily_return
macd
sd7
signal
sd14
obv
sd30
rsi
0.00
0.02
0.04
0.06
0.08
0.10
0.12

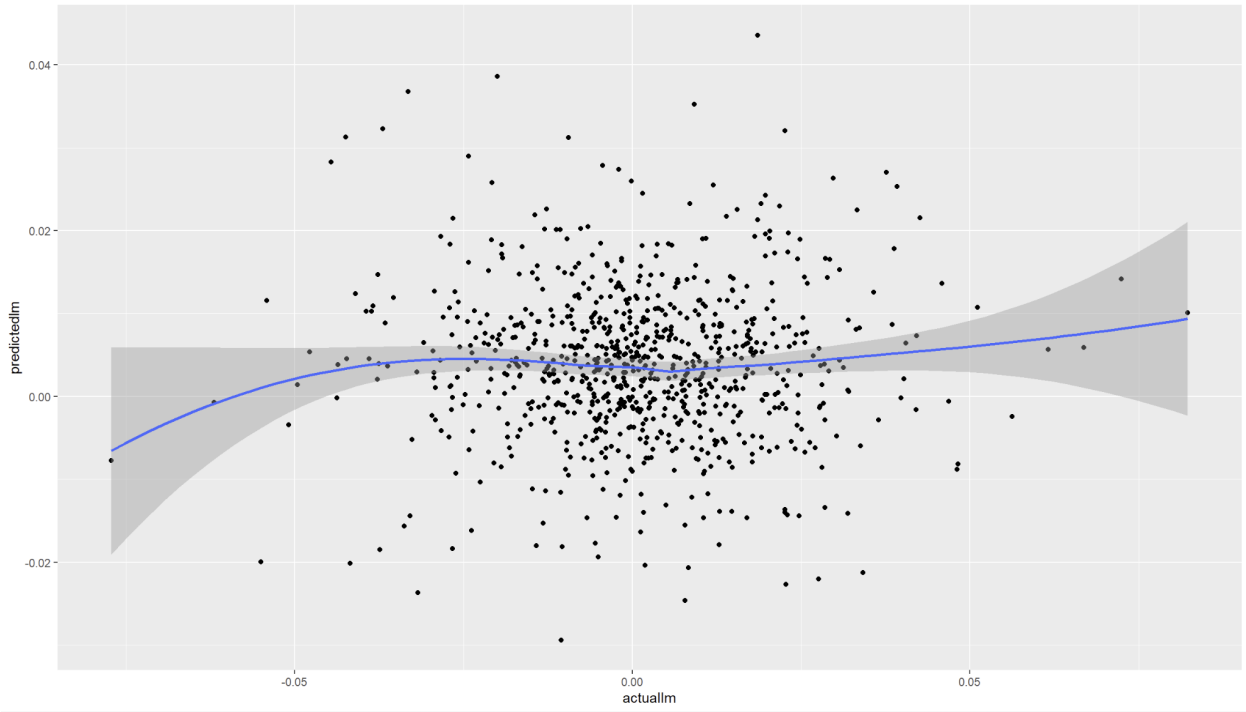

0.075
0.050
0.025
0.000
-0.025
-0.050
predictedxgb
-0.05
0.00
0.05
actualxgb


0.01
0.00
-0.01
-0.02
-0.03
-0.04
predictedrf
-0.05
0.00
0.05
actualrf


predictedlasso
0.25
0.00
-0.25
-0.50
-0.05
0.00
0.05
actuallasso

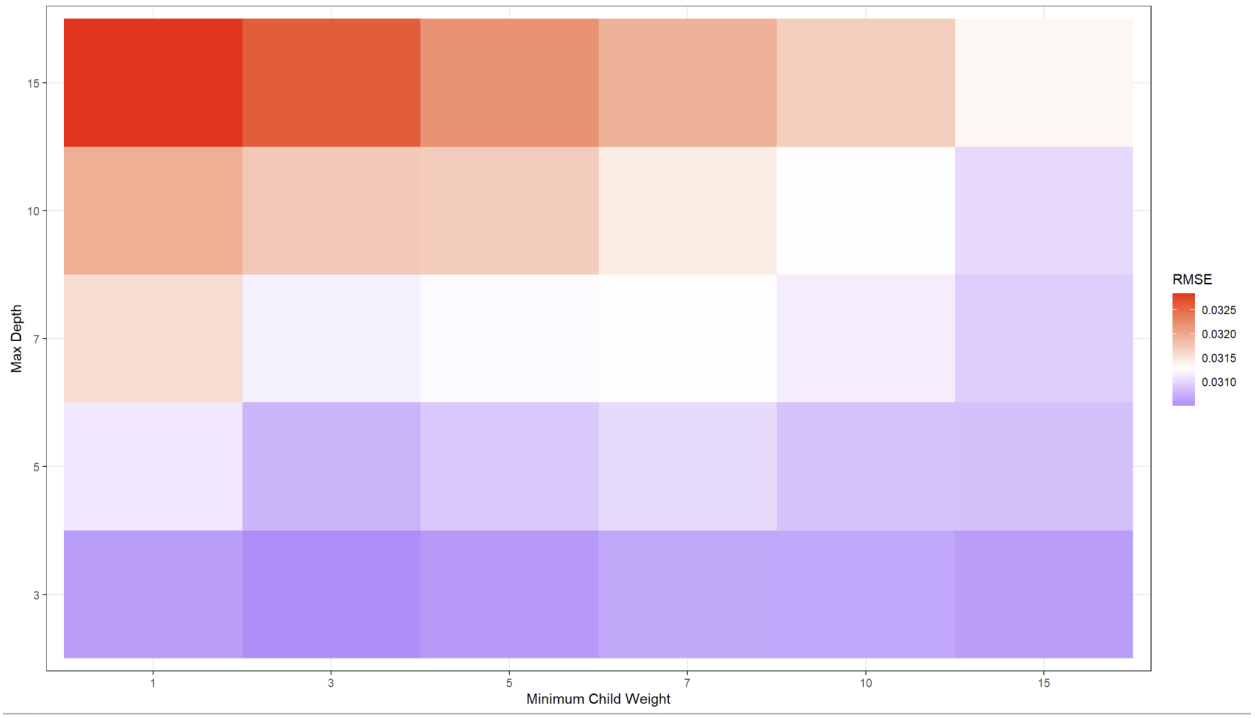

predictedbag
0.02
0.00
-0.02
-0.04
-0.05
0.00
0.05
actualbag

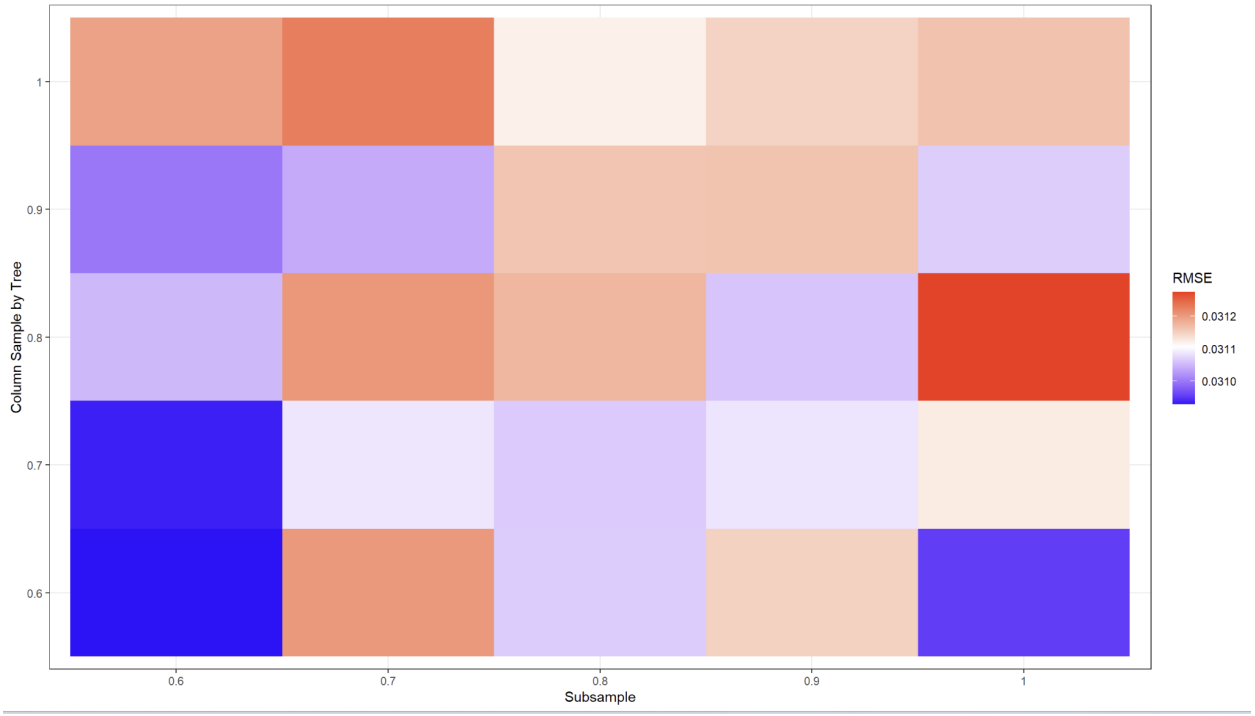

0.010
0.005
0.000
-0.005
-0.010
predictedsvm
-0.05
0.00
0.05
actualsvm

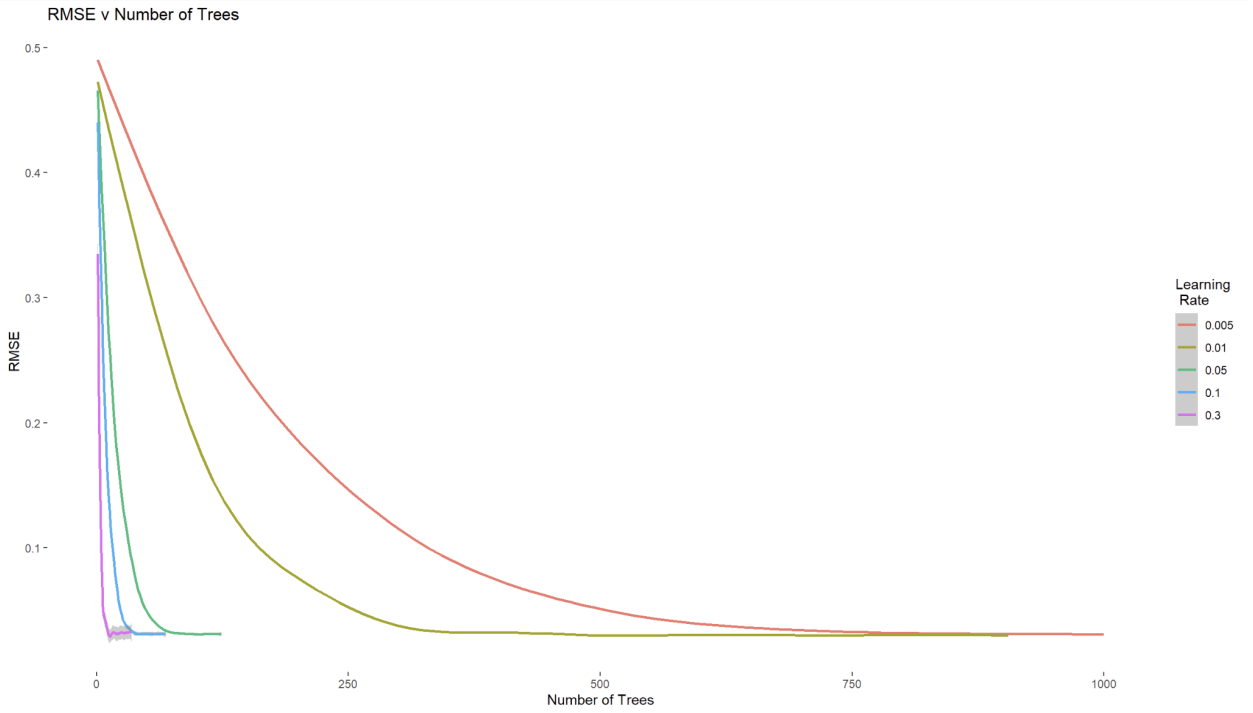

0.04
0.02
0.00
-0.02
predictedlm
-0.05
0.00
0.05
actuallm

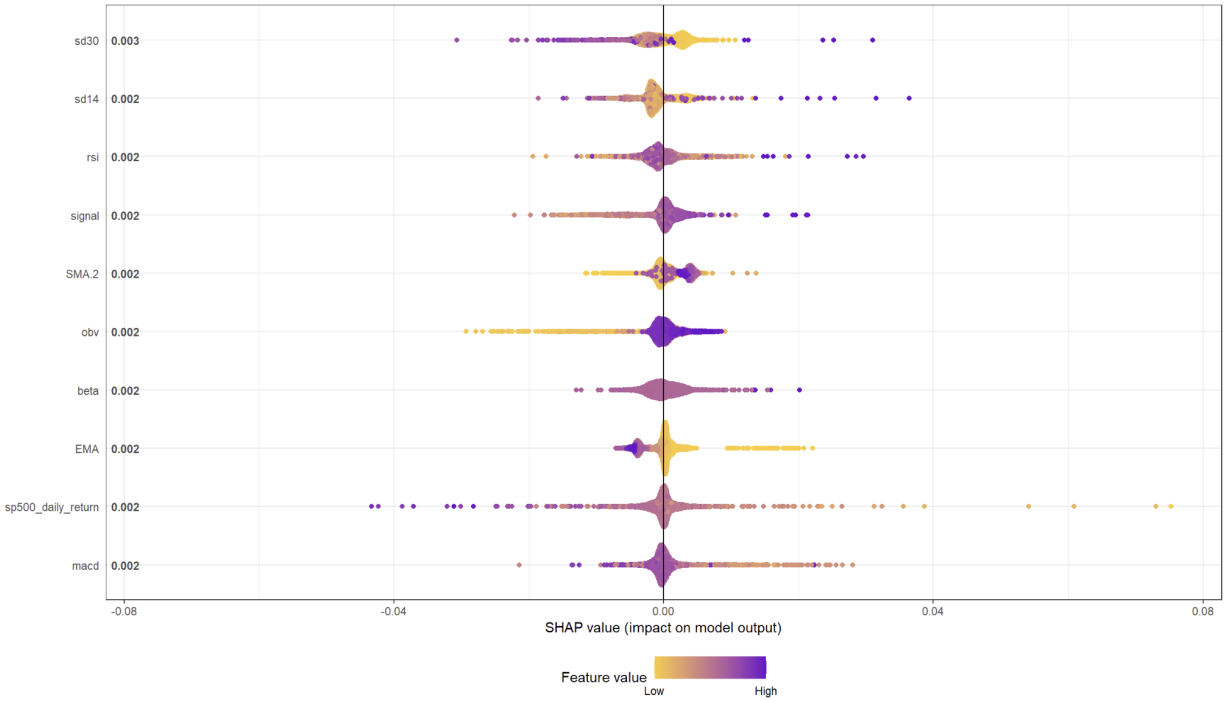
NVDA

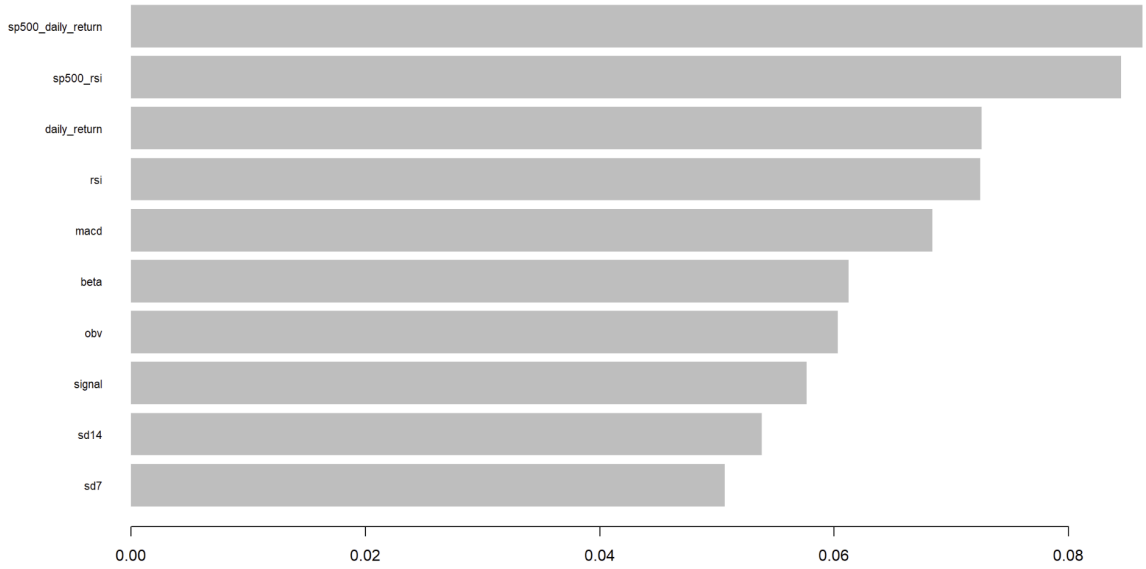
```
eta = 0.05,
 # maybe 0.3 as it learns quickest? or 0.1 as it is a happy compromise?
 # Set learning rate
 max.depth = 3,
 # Set max depth
 min_child_weight = 3,
 # Set minimum number of samples in node to split
 gamma = 0,
 # Set minimum loss reduction for split
 subsample = 0.6,
 # Set proportion of training data to use in tree
 colsample_bytree = 0.6,
 # Set number of variables to use in each tree
```

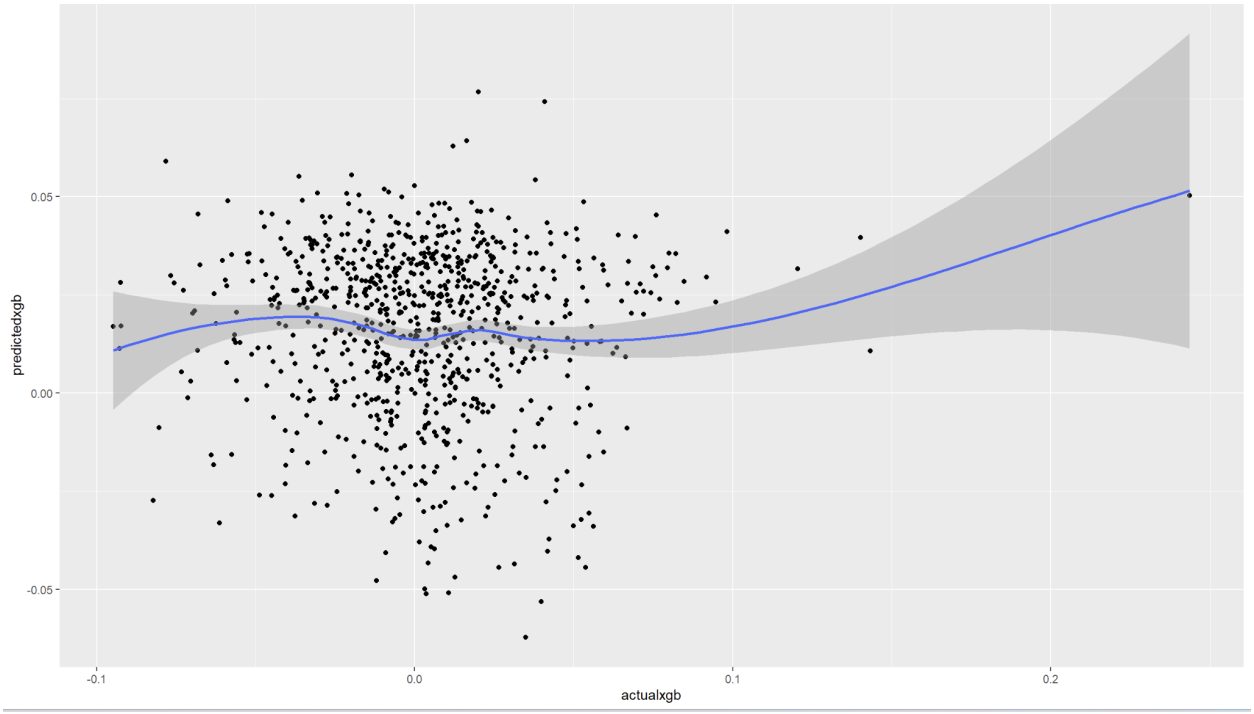


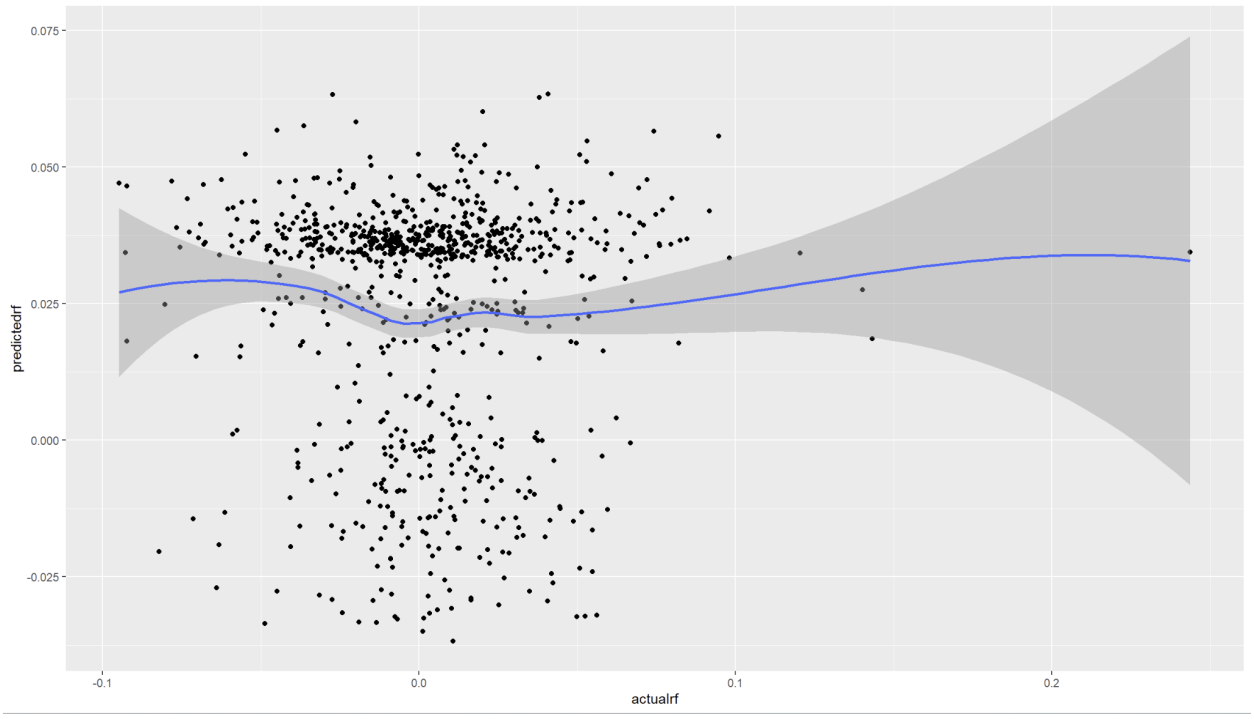

Column Sample by Tree
1
0.9
0.8
0.7
0.6
0.6
0.7
0.8
0.9
1
Subsample
RMSE
0.0312
0.0311
0.0310

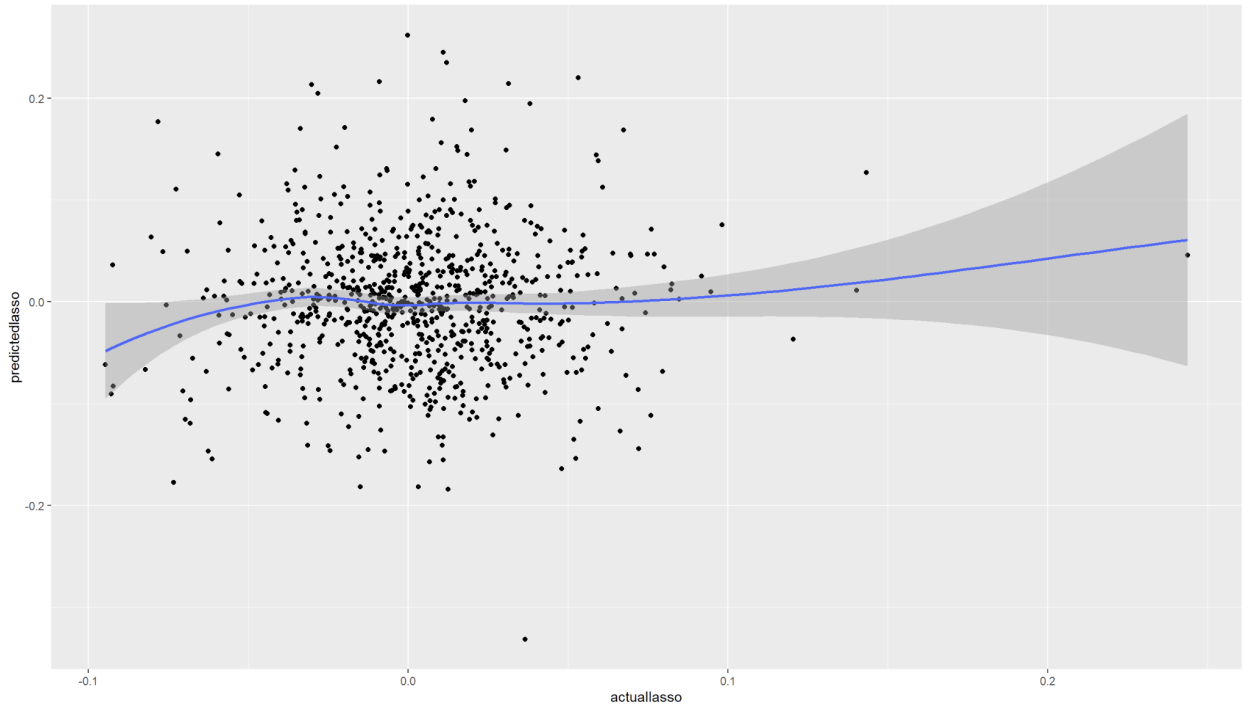

RMSE v Number of Trees
RMSE
0.5
0.4
0.3
0.2
0.1
0
250
500
750
1000
Number of Trees
Learning Rate
0.005
0.01
0.05
0.1
0.3

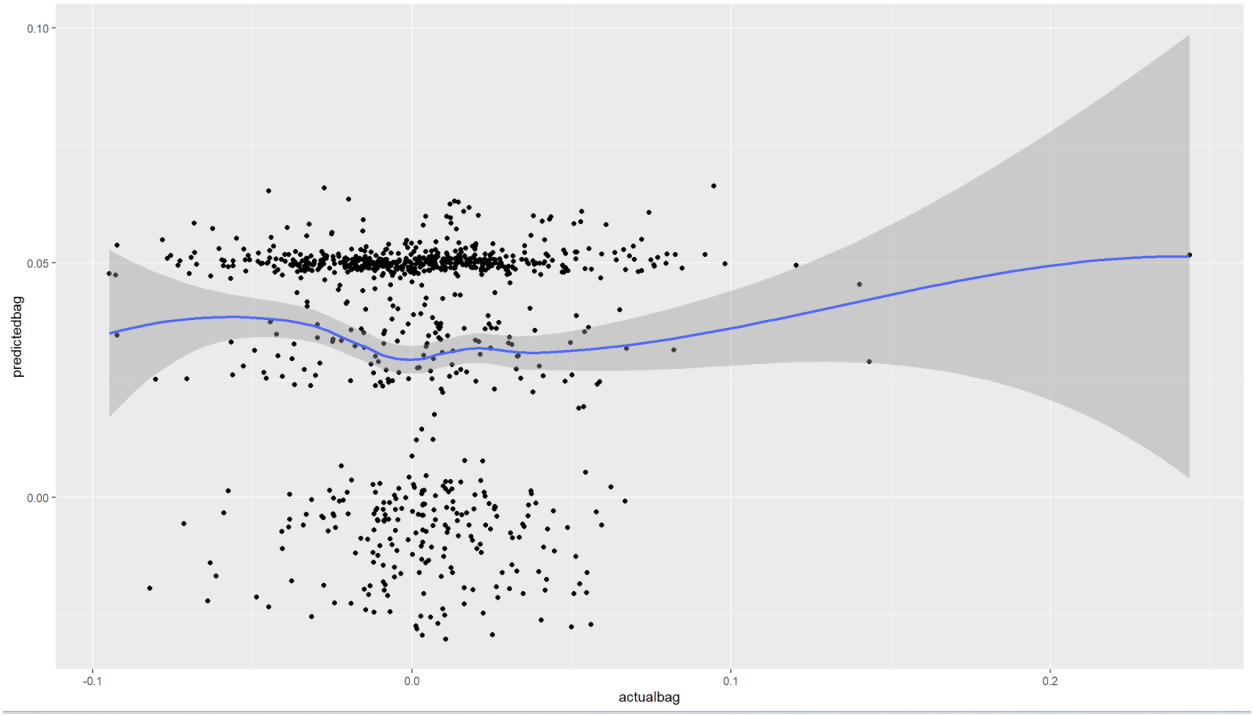

sd30 0.003
sd14 0.002
rsi 0.002
signal 0.002
SMA.2 0.002
obv 0.002
beta 0.002
EMA 0.002
sp500_daily_return 0.002
macd 0.002
-0.08
-0.04
0.00
0.04
0.08
SHAP value (impact on model output)
Feature value
Low
High

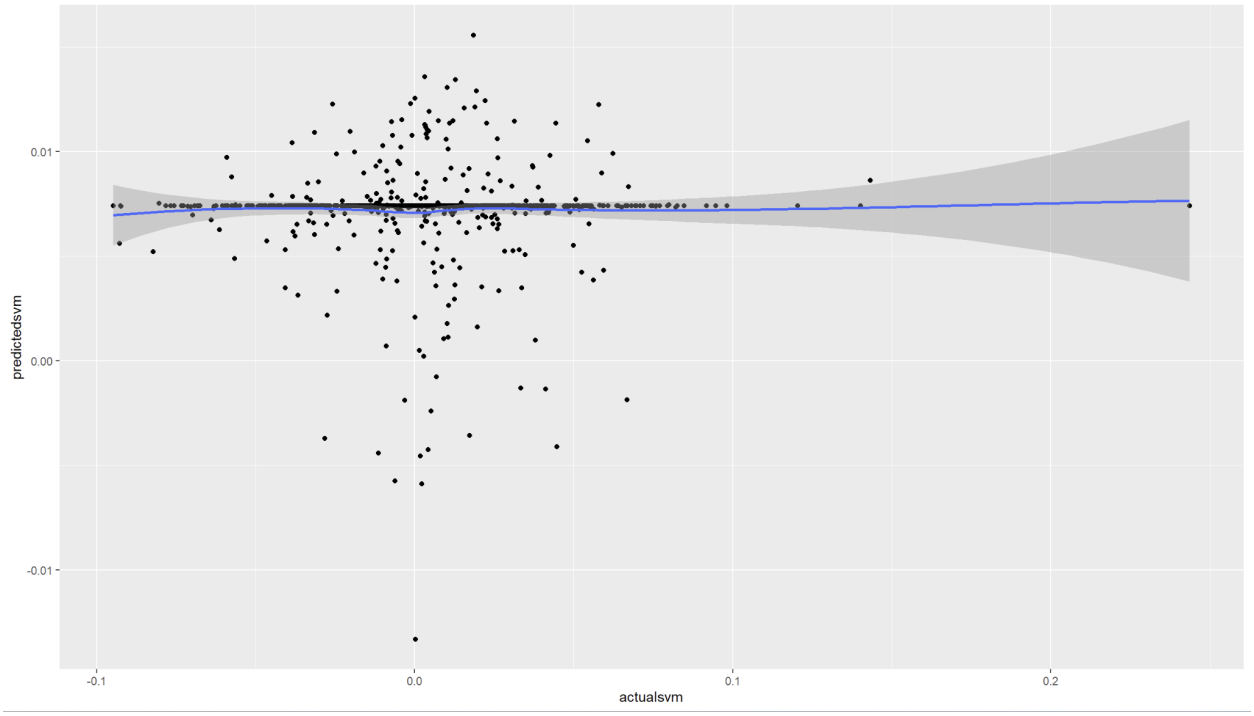

sp500_daily_return
sp500_rsi
daily_return
rsi
macd
beta
obv
signal
sd14
sd7
0.00
0.02
0.04
0.06
0.08

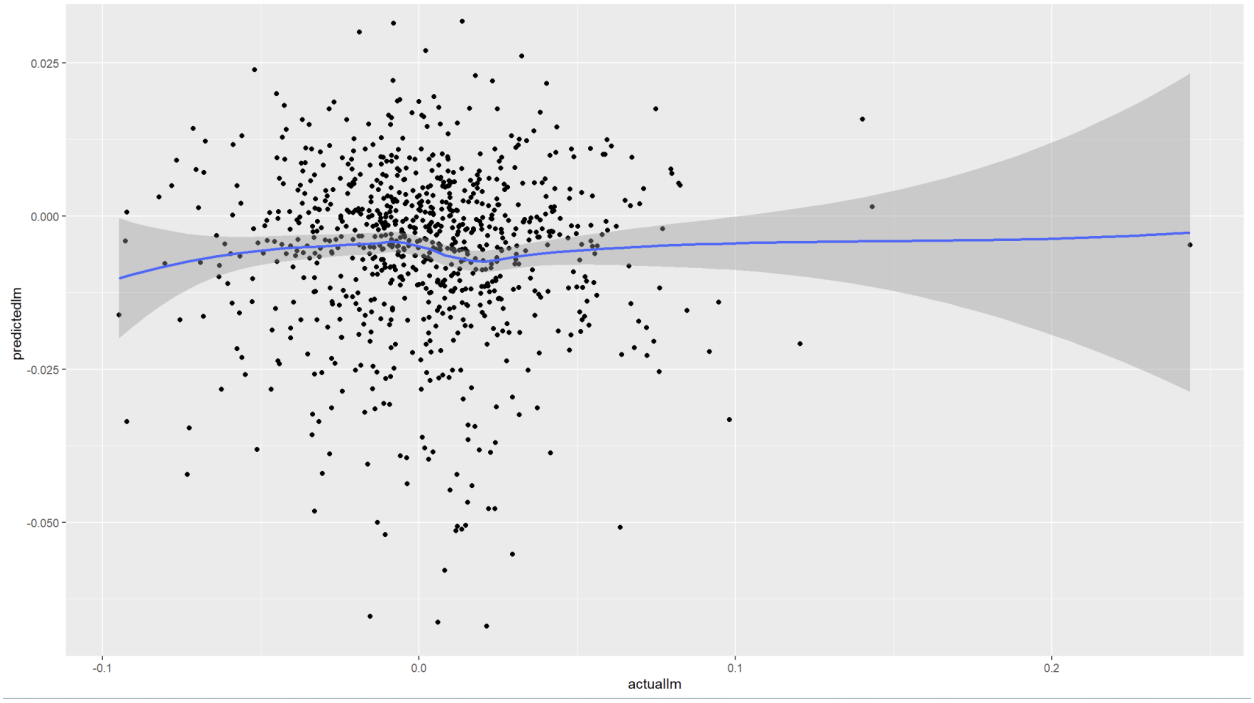

predictedxgb
0.05
0.00
-0.05
-0.1
0.0
0.1
0.2
actualxgb


predictedrf
0.075
0.050
0.025
0.000
-0.025
-0.1
0.0
0.1
0.2
actualrf


predictedlasso
0.2
0.0
-0.2
-0.1
0.0
0.1
0.2
actuallasso


predictedbag
0.10
0.05
0.00
-0.1
0.0
0.1
0.2
actualbag


predictedsvm
0.01
0.00
-0.01
-0.1
0.0
0.1
0.2
actualsvm


predictedlm
0.025
0.000
-0.025
-0.050
-0.1
0.0
0.1
0.2
actuallm