**Final project: Modernizing Risk and Return Analysis with Machine Learning**

**Clark University School of Business**

**STAT 4650**

**MACHINE LEARNING**

**Spring 2025**

A logo with text and a book

AI-generated content may be incorrect.

**Group member(s)**:

**Kubra Banu**

**Joash Kawal**

**Aman**

[**Manasa Gitta**](https://canvas.clarku.edu/courses/15412/users/14069)

**Submitted to Professor YUE Gao** **on May 6th , 2025**

**INDEX**

1. **Abbreviations ................................................................................................ 3**
2. **Abstract……………………………………………………………………... 4**
3. **Introduction………………………………………………………………… 5**
4. **Literature Review .......................................................................................... 6-7**
5. **Gaps in Current Research ............................................................................. 7-8**
6. **Methodological Framework and Data Description ….................................. 8-11**
7. **Model Performance and Results .................................................................... 11-22**
8. **Future Research and Conclusion………………...…….……………………. 23**
9. **References……………………………………………………………………. 24-25**
10. **Appendix ........................................................................................................... 26-56**

**1. Abbreviations**

|  |  |
| --- | --- |
| **Abbreviation** | **Full Form** |
| **ML** | **Machine Learning** |
| **SVR** | **Support Vector Regression** |
| **OLS** | **Ordinary Least Squares** |
| **CAPM** | **Capital Asset Pricing Model** |
| **FF** | **Fama-French** |
| **RFS** | **Review of Financial Studies** |
| **JFE** | **Journal of Financial Economics** |
| **JFo** | **Journal of Forecasting** |
| **ESWA** | **Expert Systems with Applications** |
| **AI** | **Artificial Intelligence** |
| **ANN** | **Artificial Neural Network** |
| **NLP** | **Natural Language Processing** |
| **ETF** | **Exchange-Traded Fund** |
| **S&P** | **Standard & Poor’s** |
| **GDP** | **Gross Domestic Product** |
| **RF** | **Risk-Free rate** |
| **RFR** | **Risk-Free Rate** |

**2.Abstract**

**a. Motivation and Purpose of Paper**

 The research aims to develop an investment model based on machine learning (ML) data to transform current financial investment choices. The combination of market volatility and nonlinearity in financial variables and data expansion has rendered traditional financial models less effective. Machine learning provides an effective solution to these problems while enhancing return projections.

**b. Method Including Data**

The research uses financial data from Fama-French datasets and Yahoo Finance and Alpha Vantage which includes daily and monthly historical information. The analysis includes market-based metrics (CAPM and Fama-French), valuation considerations (P/E ratio), and technical indicators (RSI, SMA, and MACD). The K-means clustering algorithm groups similar stocks together while feature engineering generates return and beta value calculations.

**c. Analysis and Results**

The models employed consisted of Linear Regression, Random Forest, and XGBoost. Additionally, traditional models like the CAPM and the Fama-French 5-Factor Model were used. Fama-French surpassed CAPM by attaining a reduced RMSE and an increased R2. In spite of its potential, machine learning models encounter difficulties, particularly overfitting, especially when applied to smaller datasets.

**d. Conclusion**

Fama-French is reliable for financial predictions due to its economic clarity. ML models show potential, but they need additional data and a broader spectrum of features. The created model can help investors make knowledgeable choices by utilizing both conventional and contemporary techniques

**3. Introduction**

**a. Motivation of the Project in Detail**

Risk and return analysis is the pivot of financial modelling, hitherto dominated by linear factor models such as the Capital Asset Pricing Model (CAPM) and Fama-French factor models. While these models are revered for their intuitive simplicity and interpretability, they are dramatically restrictive in their capacity to describe the complexity of contemporary markets. These limitations, including linear assumptions and reliance on static variables, have created room for machine learning (ML) to emerge as a dynamic option. ML models, since they can learn nonlinear relationships and process large datasets, offer revolutionary potential for financial forecasting. Kelly and Xiu describe that the combination of ML with financial modelling is a "nascent literature" expanding quickly in the finance arena (Kelly and Xiu). Thus, the difference between ML models and traditional models offers fertile ground to cultivate risk-return analysis.

The extremely unpredictable financial markets no longer utilize the standard distributions found in traditional models. Investors now require advanced tools to assess large amounts of data from market performance, news sentiment, and social media interactions. Machine learning can help detect nonlinear patterns and trends that conventional models often missed out.

**b. Specific Problem Under Study**

The research notably employs Nvidia stock as a case study to explore if machine learning (ML) can surpass traditional models (CAPM, Fama-French) in forecasting stock returns and constructing optimal portfolios because of its volatility and extensive feature set.

**c. Why Studying Problem is Important**

Precise return predictions and portfolio optimization are essential for reducing risk and enhancing investment gains. In the era of big data, it is essential to verify whether machine learning (ML) can offer superior financial insights compared to conventional financial models.

**d. Research Questions**

* Can ML outperform traditional financial models in return prediction?
* How do ML models compare in terms of interpretability and accuracy?
* Is there a trade-off between model complexity and predictive power?

**4. Literature Review**

**1.Traditional Risk-Return Models: CAPM and Fama-French**

CAPM presumes expected returns are influenced by exposure to market risk (Sharpe). However, when financial anomalies persisted, the Fama-French 5-Factor model (FF5) emerged as an advanced substitute. FF5 extends size, value, profitability, and investment dimensions to offer increased explanatory power (Fama and French). Empirical evidence leans towards suggesting FF5 has more adjusted R² and smaller errors in predictions than CAPM, emphasizing its ability to detect multidimensional risks (Fama and French).

Even so, such models' inflexibility poses significant disadvantages. They are based on static relationships and frequently do not capture interaction effects or nonlinear patterns (Brogaard et al.). They also have low explanatory power, especially in predicting individual stock returns where market conditions change rapidly (Gu, Kelly, and Xiu). As a result, even though the conventional models continue to be pillars of financial economics, their flaws paved the way for ML-based approaches.

**2.Emergence of Machine Learning for Financial Modeling**

Relative to conventional approaches, machine learning offers the capability and flexibility to handle financial data's noisy, multidimensional nature. As developed by Henrique et al., ML is optimally positioned to find complex patterns, handle high-dimensional data, and offer predictive insights higher than those of traditional factor models (Henrique et al.). Ensemble methods like Random Forest and XGBoost and neural networks allow for the inclusion of multiple data sources and finding nonlinear relationships common in financial markets.

Interestingly, ML models are increasingly being utilized to augment factor-based models. Diallo et al. demonstrate how a Bayesian optimized Support Vector Regression (SVR) significantly performs better than standard linear factor models with 92–94% out-of-sample correlation in predicting returns across U.S. industry portfolios (Diallo et al.). Besides, Gu, Kelly, and Xiu's deep work identifies ML's ability to look through massive amounts of financial information and outperform traditional models by recognizing new unknown factors that enhance cross-sectional return forecasting (Gu, Kelly, and Xiu).

But ML is not always optimal. Its performance is highly context-dependent, sometimes requiring huge data to avoid overfitting. As in the case of small-sample studies, e.g., applying Random Forest or XGBoost to intraday stock returns, ML may be worse than simple models such as FF5 by overfitting and a lack of feature diversity (Slides - ML).

**3.Predictive Performance and Interpretability**

Even though ML is usually superior to traditional models in predictive accuracy, it is always a concern with respect to explainability. Traditional models provide clear economic intuition, each coefficient has a direct correspondence to a factor such that ML lacks (Brogaard et al.). Even when ML can identify primary return drivers, according to Kwon's research on ensemble ML models, its ranking of secondary factors may be unstable or misleading . This obscurity handicaps the application of ML in risk management and investment choice settings where explanation and accountability are of utmost importance.

To address this, recent studies employ explainable AI (XAI) techniques like SHAP values and partial dependence plots to estimate ML decisions in terms of money. Nevertheless, scholars such as Nazareth and Reddy observe that explainability in ML for finance is still in its early stages and is an important area of study (Nazareth and Reddy).

**4.Applications and Empirical Findings**

ML is still applied more in return prediction, volatility estimation, credit risk assessment, and portfolio optimization. For example, Khoa and Huynh demonstrate how SVR-enhanced FF models reduced RMSE in Vietnamese market predictions as compared to CAPM (Khoa and Huynh). Moreover, research by Gu, Kelly, and Xiu displays the application of ML in portfolio allocation through reducing return-driving features better than manually selected factors (Gu, Kelly, and Xiu).

However, traditional models still find uses in stress testing, scenario analysis, and theoretical consistency. ML is usually used alongside, rather than in addition to, traditional models. Brogaard et al. describe this relationship as complementary, wherein ML improves traditional models by refining prediction while still leveraging the interpretive framework of CAPM and FF5 (Brogaard et al.).

**5. Gaps in Current Research**

Despite advances, several critical gaps in current research remain:

**1.Theoretical Background:** ML's predictive potential is not based on financial economics theory. Kelly and Xiu propose the incorporation of ML-discovered patterns within asset pricing theory (Kelly and Xiu).

**2.Interpretability:** Even though explainable AI is gaining traction, findings for misleading feature importance emphasize the significance of advanced interpretability frameworks .

**3.Robustness:** ML models universally break down when market regimes shift. Future research must develop models that dynamically adjust without sacrificing reliability (Nazareth and Reddy).

**4.Validation:** Overfitting issues and lacking long-run out-of-sample performance persist. There needs more employment of strict validation procedures (Gu, Kelly, and Xiu).

**5.Alternative Data Integration:** Thorough incorporation of alternative data, sentiment, and macro data in ML-enhanced risk-return models is hardly a topic for investigation (Nazareth and Reddy).

**6.Case Study Rationale: Why Nvidia?**

Nvidia Corporation was chosen because it holds unique positions within the finance and tech markets today. As an AI and GPU pioneer, Nvidia is the prime example of technological convergence with finance. Its recent record has been incredible — crossing a market capitalization of $3.4 trillion and displaying huge volatility (Slides - ML). This volatility embodies not only market risk in the form of CAPM but also sectoral and speculative risk elements that might be more explained through ML models.

Nvidia's leadership in AI-powered tales, its breakaway expansion, and its wealth of data infrastructure (deep trading history, balance sheets, sector connectivity) make it a prime candidate. The company's position as an AI stock benchmark and its price-volatility behavior create an unparalleled laboratory on which to assess both traditional and ML-based models for predicting returns. Further, Nvidia offers a realistic setting under which the adaptability and fact-based observations of machine learning may be contrasted with the experimentally tested interpretability and parsimony of CAPM and FF5 (Slides - ML).

Thus, Nvidia was selected for this project not only on its relevance in the market but also because it offers an empirically rich setting in which to test the research question at the center of this analysis: Can machine learning fruitfully revise risk-return analysis?

**6. Methodological Framework and Data Description**

This project compares and discusses the usage and application of both theory-based financial models and machine learning models in the field of financial analytics. The models used herein are XGBoost, Random Forest, the Capital Asset Pricing Model (CAPM), and the Fama-French Three-Factor Model. They all possess distinct characteristics and applications that qualify them for various types of financial problems.

XGBoost is a robust machine learning model with decision tree ensembles. It is known to have good predictive capability, especially in structured data scenarios. XGBoost has wide usage in financial analysis to perform tasks such as credit risk scores, stock price forecasting, and fraud detection. It can handle large datasets as well as the missing values with ease, one of its biggest strengths. But it may be difficult to interpret and most frequently needs good parameter tuning to produce good results.

Random Forest is another ensemble learning method that works by the generating numerous decision trees and then combining their predictions. Random Forest is commonly used for classification and regression problems in finance, such as the loan default prediction, market movement classification, and financial fraud detection. Random Forest is valued for its stability and ability to reduce overfitting, making it a suitable model when working with noisy or complex data. While more interpretative than XGBoost, it is less transparent than traditional statistical models.

CAPM (Capital Asset Pricing Model) is a theoretical construct in the finance used to calculate the expected asset return from market risk. Simple and widely used for the calculation of cost of equity as well as measuring performance of an investment. CAPM is appreciated for its simplicity and use but is limited by the one risk factor assumption (market risk) and the assumption of perfect markets. It is more theoretical in character and less applicable to predictive use.

The Fama-French Three-Factor Model expands CAPM by adding two additional factors—size and value—in an attempt to better explain asset returns. It picks up more variation in stock return and is widely applied in academia and portfolio management. More precise in most cases, however, than CAPM is the Fama-French model, since it is also more complex and needs more data and still relies on a number of assumptions regarding investor behavior and market efficiency.

1. **Data description and source**

This analysis utilizes two major data sources: NVIDIA's daily stock data and the Fama–French five-factor model's daily factor returns.NVIDIA's adjusted closing prices data was obtained through Yahoo Finance from the yfinance Python library. The data range was from the 1st of January 2015 to the 31st of December 2024. The adjusted closing prices alone were utilized to calculate the excess returns and the daily log returns.

The second data set consists of the Fama–French five-factor daily returns, taken directly from the data library of Kenneth R. French. It contains the following factors:Excess market return (mkt\_rf): The excess return on a broad market portfolio (often the value-weighted return on all U.S. stocks) in excess of the risk-free rate. It is the reward investors receive for bearing market risk.

Size (smb – "small minus big"): Average differential return between small-cap and big-cap stocks. It represents the size factor that describes how smaller companies tend to outperform larger companies in the end.Value (hml - "high minus low"): The difference in average returns between high book-to-market value stocks and low book-to-market growth stocks. It measures the value premium in the returns to equities.

Profitability (rmw – “robust minus weak”): The difference in average return between firms that have high (robust) and low (weak) profitability. This captures the tendency for more profitable firms to have higher returns.Investment (cma - "conservative minus aggressive"): The spread of return between conservatively investing and aggressively investing firms. It indicates how investment intensity translates into the performance of a stock.

These factor returns are already in excess returns form, that is, adjusted for the same risk-free rate as in the case of the stock returns. The two datasets were matched on common trading dates to form a combined dataset for modeling. There is an observation for each trading day that contains NVIDIA’s excess return and five explanatory variables that are utilized in asset pricing studies.

1. **Variable description**

The final dataset includes the following variables utilized in the regression models

nvda\_exret: NVIDIA’s daily excess return, calculated as the log return minus the daily risk-free rate. This is the dependent (target) variable in all models.

mkt\_rf: Market excess return, which captures the overall market risk premium.

smb: Size factor, which reflects the differential in returns between small and big firms.

hml: Value factor that measures the difference in returns between value and growth securities.

RMW: Profitability factor that captures differences in returns on the basis of profitability.

CMA: Investment factor, a measurement of the return gap between conservative and aggressive investment companies.

These variables are typical elements of asset pricing models, the CAPM and the Fama–French five-factor models. The choice of variables allows examination of the extent to which these risk factors which are systematic explain the performance of NVIDIA's stock through time.

1. **Data Preprocessing**

Data preparation started by computing the daily log returns for the adjusted closing prices of NVIDIA. For trading day t, returns were calculated as:  
rₜ = ln(Pₜ / Pₜ₋₁),where Pₜ represents the adjusted close for day t and Pₜ₋₁ represents the adjusted close for the prior trading day.

To compare performance to a risk-free benchmark, the 10-year annual U.S. Treasury yield of 4.035% was converted to a daily rate by dividing by 252 trading days in a standard year:  
rfₜ = 0.04035 / 252.The excess daily return for NVIDIA was subsequently obtained by deducting the risk-free rate for the day from the log return:  
nvda\_exretₜ = rₜ - rfₜ.

We then acquired daily data for the five Fama–French factors—market excess return (mkt\_rf), size (smb), value (hml), profitability (rmw), and investment (cma)—from the Kenneth R. French data library. These series already provide excess returns above the risk-free rate so no adjustment was required.

The NVIDIA return and Fama–French factors were joined on common trading dates to yield a merged DataFrame in which a row corresponds to a trading day and includes the excess return for NVIDIA as well as the five factor values. Two model datasets were constructed: the first for the Capital Asset Pricing Model (CAPM), which only contains the predictor mkt\_rf, and the second for the complete Fama–French five-factors specification. Both datasets contain the target variable (nvda\_exret) and relevant predictors.

1. **Data Cleaning**

In the process of uniting NVIDIA's return data with the data for the Fama–French factors, a few dates—fewer than a decade—were found in which there were missing values in either the series of stock returns or in one of the series of the factors. These dates essentially occurred for the market holidays and therefore accounted for non-trading days.

Since these missing records accounted for fewer than 0.5% of the overall data and happened outside of the trading periods, these records were dropped from the data. This was done in order to have a clean and uniform series without gaps or misalignments.

For data integrity purposes, the date column was changed to the relevant datetime format using Python’s pandas library. Numerical columns of returns and factor values have been confirmed to be of the type float to enable precise statistical analysis.

Further, a check was made for the presence of extreme outliers in the excess return series of NVIDIA. Observations more than ±5 standard deviation away from the mean were taken into consideration but none were found. Consequently, all the valid observations were kept for the next analysis.

1. **Handling of Missing Values**

Missing values were handled in the data cleaning process. All the missing entries that were found reflected non-trading days, e.g., holidays or weekends when the price data of NVIDIA or the value for any of the Fama–French factors was missing.

Because there were fewer than a handful of these cases and because they happened to occur on non-trading days, no imputation was done. Rather, these rows in the data were deleted altogether. This allowed the time series alignment between NVIDIA's excess returns and the factor data to be maintained without the insertion of artificially made values.

The last data set includes a full series of observations for the exact matching between NVIDIA's excess returns and Fama–French factors for all trading days within the sample duration between the dates January 1st, 2015, and December 31st, 2024.

**7. Model Performance and Results**

1. **EDA**

To determine the nature and interrelations present in the data, a thorough exploratory data analysis was performed on the excess returns of NVIDIA and the variables of the Fama–French five-factor model. This included calculating summary statistics, exploring the return distribution, and assessing pairwise correlations.

***1.Summary Statistics***

Descriptive statistics for the excess returns of NVIDIA (exs\_rtn\_nvda) and the five Fama–French factors—market excess return (MKT\_prm), size (SMB), value (HML), profitability (RMW), and investment.

The mean excess return for NVIDIA is about -0.0039 and the standard deviation of 0.0314 reveals relatively high variability in returns in relation to the factor variables. Between the factors, market premium (MKT\_prm) has a low mean value of 0.0003, along with a much lower standard deviation value of 0.0112 that reflects a broad market benchmark.

These factors have higher dispersion levels, i.e., standard deviation of 0.6896 for SMB and 0.9010 for HML, indicating that the size and value premiums have higher variability.RMW and CMA vary considerably and are skewed to a moderate degree towards positive returns.

A screenshot of a computer

AI-generated content may be incorrect.

***Figure: Summary Statistics***

***2.Distribution of Returns***

NVIDIA's excess returns' distribution is plotted in the histogram below. The returns look roughly normally distributed but slightly skewed and leptokurtic, indicating the occurrence of infrequent huge price movements.

**A graph of a tall graph

AI-generated content may be incorrect.**

***Figure: Distribution chart***

***3.Scatter Matrix and Relationships***

A pair plot was made to represent the correlations between the five Fama–French factors and NVIDIA's excess returns. Interestingly:

exs\_rtn\_nvda has the strongest positive relationship with MKT\_prm as expected according to asset pricing theory.The other factors have no or only weak visual relationships to NVIDIA’s returns, though there are some positive relationships between the factor variables themselves (for example, between CMA and HML, or between RMW and HML).

**A group of blue dots

AI-generated content may be incorrect.**

***Figure: Pairplot***

***4.Correlation analysis***

A correlation heatmap was also developed to explore the linear relationships between all variables in more detail. Some insights are:

A positive correlation strength of 0.62 between NVIDIA's excess return and the market factor (MKT\_prm), which reflects that movements in the market impact strongly on NVIDIA's performance. There are observed to be weak or slightly negative correlations between the returns of NVIDIA and other variables like HML (-0.30), RMW (-0.19), and CMA (-0.37). Significant positive correlations between the factor variables include CMA and HML (0.58), as well as HML and RMW (0.32), although these might affect multicollinearity in regression.

**A diagram of a heatmap

AI-generated content may be incorrect.**

***Figure: Heatmap***

***5.Time Series Comparison: NVIDIA vs. Market***

The following chart displays the excess daily returns for NVIDIA (blue) and the market excess return (MKT\_prm) (red) for the entire sample. It shows:

* The persistently elevated volatility in NVIDIA's returns compared to the market.
* Several notable spikes in NVIDIA’s returns, particularly during earnings releases and macroeconomic events (e.g., mid-2020 and mid-2023).
* Although both series have a similar directional bias in places, NVIDIA has a larger amplitude in accordance with a high-risk, high-growth stock.

**A graph with a red and blue line

AI-generated content may be incorrect.**

***Figure: Line Chart***

1. **Hypothesis Testing**

The primary research question for this study is:

Can machine learning models that are more sophisticated overcome conventional asset pricing models including the Capital Asset Pricing Model (CAPM) and the Fama–French five-factor model in explaining the excess returns of NVIDIA?

To meet this end, the following hypotheses were developed:

Null Hypothesis: There are no differences between CAPM and Fama–French traditional linear models in outperforming in the prediction or explanation of NVIDIA's excess returns using advanced machine models.

Alternative Hypothesis (H₁): More sophisticated machine-learning-based models do perform better than traditional linear models in explaining or predicting NVIDIA's excess returns.

The test framework consists of comparing the forecasting accuracy of standard models—CAPM as well as the Fama–French five-factor model—with that of chosen machine learning algorithms. The models are tested using the same metrics i.e. Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R²) on a holdout test sample.

We will reject the null hypothesis in favor of the alternative hypothesis if machine learning models prove to have statistically and practically better performance in explaining the variation in NVIDIA’s excess returns.

1. **Modeling**

To analyze the behavior of the excess return of NVIDIA and verify whether the advanced ML models are superior to the traditional asset pricing methods, a variety of models were tested. We started with classical linear models based on financial theory, namely CAPM and FamaFrench five-factor model, and accelerated to machine learning ones like XGBoost and Random Forests, trained on the very same financial predictors. Each model was trained on 80% of the data and tested on the remaining 20% data to maintain generalizability and prevent look-ahead bias.

CAPM (Linear Regression)

We consider the CAPM model as a benchmark for this study, which suggests that the excess return of a stock is attributable to its sensitivity to the market premium only. The model is expressed as:

*nvda\_exret = α + β1 ⋅ MKT\_prm + ε*

* Alpha (α): 0.00197
* Beta (β₁): 1.7356

where nvda\_exret is the daily excess return on NVIDIA, and MKT\_prm is the excess return on the market portfolio. The estimation of NVIDIA's sensitivity to market is measured by the presence of coefficient β1 and α gauges any consistent abnormal return unexplained by market.

Performing the CAPM regression across the entire sample yielded an alpha (α) of 0.00197 and a beta (β1) of 1.7356, suggesting that the returns from NVIDIA are very much responsive to that from the market – nearly 74% more volatile than that of the market. The R² is 0.405 which implied that the model could make up the 40.5% of variation in the daily excess returns of NVIDIA.

By 80/20 train-test split, the model was able to obtain test MSE of 0.000681 and test R² of 0.3243 with virtually the same test adjusted R² of 0.3229. Such results confirm that the CAPM exhibits some systematic behavior, but a significant amount of variability remains unexplained.

Fama–French Five-Factor Model (Linear Regression)

The Fama–French model is an extension of CAPM that adds four extra risk factors: size (SMB), value (HML), profitability (RMW), and investment (CMA). The model takes the form:

*nvda\_exret = α + β1 ⋅ MKT\_prm + β2 ⋅ SMB + β3 ⋅ HML + β4 ⋅ RMW + β5 ⋅ CMA + ε*

The coefficients estimated from the full-sample regression, are:

* Alpha (α): -0.00454
* β₁ (MKT\_prm): 1.6305
* β₂ (SMB): 0.00153
* β₃ (HML): -0.00840
* β₄ (RMW): -0.00014
* β₅ (CMA): -0.00496

The adjusted R² of the model is 0.4627, which suggests a decent improvement in explanatory power compared to CAPM. But it fared poorly on the holdout set: MSE = 0.000851, R² = 0.1651, and adjusted R² = 0.1567. This drop suggests possible overfitting or less relevance of some of the Fama–French factors for NVIDIA.

***1.CAPM with XGBoost***

Nonlinearity relationships were studied by running the CAPM model based on XGBoost, which is a boosting algorithm ensemble known for its great predictive power. Only the MKT\_prm variable was input as a control, to allow the model to remain analogous to the linear CAPM. The hyperparameter tuning was done using the grid search and the best configuration used:

* learning\_rate: 0.1
* max\_depth: 2
* n\_estimators: 50
* subsample: 0.8

The adjusted model reached MSE, RMSE and R² of 0.000676, 0.02600 (R² = 0.3290) in the test set. These statistics were not far better than the linear CAPM, suggesting that the XGBoost model did manage to identify a little bit of nonlinearity in the data, but not much.

***2.CAPM with Random Forest***

Analogous experiment was carried out with Random Forest Regressor using the market factor only. Grid search tuning chose a maximum depth of 4 and 'sqrt' as max features. The model achieved MSE 0.000678, RMSE 0.02604, and R² 0.3270, fitting very well that of XGBoost. These findings also give additional evidence that even flexible ML models make only incremental gains when using a lone feature such as MKT\_prm.

***3.Fama–French with XGBoost***

The complete set of five Fama–French predictors were then entered into an optimized XGBoost model. Even with provision of a larger input space, performance was worse. The optimal model used:

* learning\_rate: 0.1
* max\_depth: 3
* n\_estimators: 50
* subsample: 1.0

The test set evaluation also returned an MSE of 0.000866, an RMSE of 0.02942, and an R² of 0.1506. This is substantially less than the linear and XGBoost CAPM models, indicating that the extra factors could have introduced some noise as opposed to additional signal in a nonlinear framework.

A graph with blue bars

AI-generated content may be incorrect.

***Figure : XGBoost Feature Importance***

***4.Fama–French with Random Forest***

Finally, the Fama–French model was calibrated using Random Forest with hyperparameter optimization for maximum depth of 4 and square root max feature. Testing performance was subpar as well: MSE 0.000880, RMSE 0.02967, and R² 0.1362. This renders it the weakest model in the context of explaining power, which is indicative of the fact that the Random Forest failed to pick up significant interactions among the FF factors in this context.

1. **Performance of the Model**This section illustrates a comparison of all the models that have been used for forecasting NVIDIA’s excess return data. Each model was trained on 80% of the data and tested on the remaining 20% across the same performance metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared (R²), and Adjusted R-squared.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **MODELS​** | **CAPM​** | | | **Fama French​** | | |
|  |
| **METRICS ​** | **MSE​** | **RMSE​** | **Adjusted R2​** | **MSE​** | **RMSE​** | **Adjusted R2​** |  |
|  |
| **Regression Model​** | **0.00056​** | **0.023619​** | **0.40469​** | **0.00053​** | **0.02302​** | **0.46167​** |  |
|  |
| **XGBoost​** | **0.00068​** | **0.02600​** | **0.32770​** | **0.00087​** | **0.02942​** | **0.14205​** |  |
|  |
| **Random Forest​** | **0.00087​** | **0.02951​** | **0.13406​** | **0.00088​** | **0.13623​** | **0.12754​** |  |
|  |

***Table: Models and metrics***

In both CAPM and Fama–French specifications, linear regressions performed better than machine learning methods. Fama–French linear regression performed the best with the smallest MSE (0.00053), smallest RMSE (0.02302), and highest Adjusted R² (0.4617), as it explained about 46.2% of NVIDIA's excess return variability.The CAPM regression lagged behind at a high Adjusted R² of 0.4047.By way of comparison, machine learning algorithms—most significantly Random Forest—fared poorly. Random Forest and XGBoost both exhibited higher rates of error and lower explanatory strength. In particular, Random Forest's Adjusted R² values were below 0.15 in both architectures even amidst their nonlinear flexibility, indicating weak generalizability to unseen data.   
  
These differences highlight the strength of theory-based econometric models, specifically as they relate to a stock such as NVIDIA whose returns are undoubtedly heavily correlated to linear relationships with well-established finance risk factors.

**e) Interpretation of Results and Recommendations**

The results indicate superior performance of sophisticated machine learning approaches relative to conventional linear regression methods under both CAPM and Fama–French architectures. Fama–French linear model provided the most predictive accuracy with the smallest test-set MSE value of 0.00053, RMSE value of 0.02302, and highest adjusted R² of 0.4617, which means about 46.2% of the excess return variability in NVIDIA was accounted for by its market, size, value, profitability, and investment risk factors exposures.

The CAPM model of regression was likewise very strong with an adjusted R² of 0.4047, suggesting even one factor, the market premium, explained very much of the variability of return. This demonstrates the efficacy of theoretically derived, simple models for financial prediction.

Contrastingly, machine learning algorithms like XGBoost and Random Forest, even after cross-validation tuning, offered lower explanatory power as well as increased rates of error. Their adjusted R² values were 0.13 to 0.32, and were worst for Random Forest models, which failed to generalize both in CAPM and Fama–French environments.

These findings suggest that in formal low-dimensional data settings in finance, linear models may not be outperformed by machine learning, especially where the relationship between return and the predictors is largely linear.

**i.Recommendations:**

1. Act as the standard for linear methods: Historical finance models such as CAPM and Fama–French remain very helpful and easy to interpret to forecast asset-level returns.

2. Use machine learning responsibly: Apply non-linear models sparingly where there is high-dimensional, noisy, or unstructured data such as news sentiment data or alternative data sets.

3. Before algorithmic complexity, emphasize feature extension. Addition of the correct predictors could have more effect than model architectures modifications.

4.Machine learning techniques must supplement, not supplant, linear modeling for asset pricing, especially where the research objective is to interpret coefficients (such as beta estimates).

**ii.Connection to Research Motivation**

The introduction posed the research question:

*Yes, advanced machine learning approaches may predict and explain NVIDIA's excess returns better than such standard finance designs as CAPM and the Fama–French model.*  
*This research offers a straightforward, fact-based response: No—at least in precisely this case. It shows that asset pricing theory-rooted conventional models performed better and were more accurate than data-driven machine learning methods in modeling the returns of NVIDIA. This confirms the notion that finance theory is still very relevant in modeling return behavior, even with sophisticated computational power at our disposal.*  
*In addition, the research is consistent with earlier works arguing machine learning's performance benefit is contingent on data complexity, dimensionality, and feature richness. In the current situation, where predictors were restricted to known finance factors, machine learning models failed to produce better out-of-sample predictive accuracy.*

**iii.Limitations of the Study**

While the study provides meaningful insights, it is subject to several limitations:

1. Single-stock focus: The analysis was limited to NVIDIA. Results may not generalize to other firms, especially those in different sectors or with different volatility profiles.
2. Daily frequency data: High-frequency data can be noisy and heavily affected by market microstructure effects, which may obscure the effectiveness of some modeling approaches.
3. Factor-only predictors: The models used only structured, traditional risk factors. The exclusion of alternative data sources, macroeconomic indicators, or sentimental data may have limited the machine learning models' ability to outperform.
4. No transaction cost or real-world trading considerations: The models are evaluated purely by statistical fit, not profitability or trading viability.

**8. Future Research and Conclusions**

Future studies may build upon this analysis in various significant manners:  
1. Implement the models to wider cross-sections of stocks across sector and size classes to establish if findings are in various market regimes.  
2. Use additional data types like earnings reports, forecast data from analysts, social media sentiment data, or macroeconomic data to provide richer feature sets to machine learning algorithms.  
3. Examine hybrid approaches where linear and nonlinear elements are mixed—for instance, applying linear regression to estimate betas but using machine learning to fit the residuals or nonlinear adjustments.  
4. Evaluate model resilience across various time horizons, such as during volatile market conditions or during crises to determine temporal consistency.

**Conclusion:**  
In summary, the study delivers the message that standard linear asset price models—particularly the Fama–French five-factor model—continue to serve as strong tools for predicting returns even when tested against the latest machine learning algorithms. Although machine learning is highly flexible and scalable, its potential was not achieved in this low-dimensional structured data environment. These results confirm the value of the existing financial theory and indicate that model choice should focus both on interpretability and empirical performance according to data properties and the purposes of the study.

**References**

1. Brogaard, J., & Zareei, A. (2022). Machine learning and the stock market. *Journal of Financial and Quantitative Analysis, 58*(4), 1431–1472. <https://www.cambridge.org/core/journals/journal-of-financial-and-quantitative-analysis/article/machine-learning-and-the-stock-market/F54ECC9DA40067B35261B32691D1DDA2>
2. Diallo, S., Fall, M., & Guo, X. (2023). A Bayesian optimization approach to SVR for Fama-French factor estimation. *Journal of Forecasting*. <https://www.researchgate.net/publication/368511949_FAMA-FRENCH_THREE_vs_FIVE_WHICH_MODEL_IS_BETTER_A_MACHINE_LEARNING_APPROACH>
3. Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics, 116*(1), 1–22. <https://www.sciencedirect.com/science/article/pii/S0304405X14002323>
4. Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *Review of Financial Studies, 33*(5), 2223–2277. <https://academic.oup.com/rfs/article/33/5/2223/5758276>
5. Gu, S., Kelly, B., & Xiu, D. (2020). Machine learning techniques and asset pricing. *Foundations and Trends® in Finance, 14*(3), 157–294. <https://www.nowpublishers.com/article/Details/FIN-064>
6. Henrique, B. M. P., Sobreiro, V. A., & Kimura, H. (2019). Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications, 124*, 226–251. <https://www.sciencedirect.com/science/article/abs/pii/S095741741930017X>
7. Kelly, B., & Xiu, D. (2023). Machine learning and asset pricing. *Annual Review of Financial Economics, 15*, 1–26. <https://www.annualreviews.org/content/journals/10.1146/annurev-financial-030421-113113>
8. Khoa, B. V., & Huynh, T. N. (2023). Improving Fama–French models using support vector regression: Evidence from Vietnam. *Journal of Forecasting, 42*(5), 987–1007. <https://onlinelibrary.wiley.com/doi/abs/10.1002/for.2970>
9. Khoa, B. V., & Huynh, T. N. (2023). Effectiveness of OLS and SVR in return prediction: Fama-French three-factor model and CAPM framework. *Journal of Forecasting*. <https://www.researchgate.net/publication/369916082_Effectiveness_of_OLS_and_SVR_in_Return_Prediction_Fama-French_Three-factor_Model_and_CAPM_Framework>
10. Nazareth, B., & Reddy, B. (2023). A systematic literature review of machine learning in finance. *Expert Systems with Applications, 214*, 119640. <https://www.researchgate.net/publication/368256380_Financial_applications_of_machine_learning_a_literature_review>
11. Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance, 19*(3), 425–442. <https://onlinelibrary.wiley.com/doi/10.1111/j.1540-6261.1964.tb02865.x>
12. Boussaada, R., Brihmat, K., Boussaada, A., & Djelassi, M. (2021). Predicting stock market movements using machine learning techniques: The case of European markets. *Computational Economics, 58*, 1183–1206. <https://link.springer.com/article/10.1007/S10614-021-10094-W>
13. Slides – ML. (2025). *Machine learning vs traditional models for Nvidia stock return prediction* [Internal project presentation]. <https://sciendo.com/pdf/10.2478/ceej-2021-0004>

**Appendix / Appendices**

**FULL CODE**

**CAPM Model**

Importing files

from statsmodels.regression.rolling import RollingOLS

import pandas\_datareader.data as web

import matplotlib.pyplot as plt

import statsmodels.api as sm

import seaborn as sns

import pandas as pd

import numpy as np

import datetime as dt

import yfinance as yf

import pandas\_ta

import warnings

warnings.filterwarnings('ignore')

Downloading the data

sp500 = pd.read\_html('https://en.wikipedia.org/wiki/List\_of\_S%26P\_500\_companies')[0]

sp500['Symbol'] = sp500['Symbol'].str.replace('.', '-')

symbols\_list = sp500['Symbol'].unique().tolist()

symbols\_list.append('^GSPC')

start\_date = '2024-04-01'

end\_date = '2024-12-31'

start\_date = pd.to\_datetime(end\_date)-pd.DateOffset(365\*10)

df = yf.download(tickers=[symbols\_list,

start=start\_date,

end=end\_date,auto\_adjust =False,threads=True).stack()

df.index.names = ['date', 'ticker']

df.columns = df.columns.str.lower()

df.head()

# Risk-free rate (10-year U.S. Treasury yield as a percentage, e.g., 4.305%)

risk\_free\_rate = 0.04035/252 # Convert annual rate to daily rate

risk\_free\_rate

df2 = df.reset\_index()[['date', 'ticker', 'close']]

df2.head()

df2 = df2[(df2['ticker'] == 'NVDA') | (df2['ticker'] == '^GSPC') ]

df2.head()

Stock\_close = df2.pivot(index = 'date', columns = 'ticker',values = 'close')

Stock\_close.head()

Stock\_close = Stock\_close.pct\_change().dropna()

Stock\_close = Stock\_close.rename(columns={'^GSPC': 'GSPC'})

Stock\_close = Stock\_close.reset\_index()

Stock\_close\_capm = Stock\_close

Stock\_close.info()

Stock\_close\_capm['exs\_rtn\_nvda'] = Stock\_close\_capm['NVDA'] -risk\_free\_rate

Stock\_close\_capm['MKT\_prm'] = Stock\_close\_capm['GSPC'] -risk\_free\_rate

Stock\_close\_capm = Stock\_close\_capm[['date','exs\_rtn\_nvda','MKT\_prm']]

Stock\_close\_capm.head()

Stock\_close\_capm = Stock\_close\_capm.set\_index('date')

Stock\_close\_capm.info()

Stock\_close\_capm.head()

Plots

import plotly.graph\_objects as go

fig = go.Figure()

fig.add\_trace(go.Scatter(x=Stock\_close\_capm.index, y=Stock\_close\_capm['exs\_rtn\_nvda'],

mode='lines', name='Nvidia'))

fig.add\_trace(go.Scatter(x=Stock\_close\_capm.index, y=Stock\_close\_capm['MKT\_prm'],

mode='lines', name='Market Return'))

fig.update\_layout(title='Nvidia vs. Market',

xaxis\_title='Date',

yaxis\_title='Return',

template='plotly\_white')

fig.show()

import matplotlib.pyplot as plt

# Use the index as date (already sorted)

dates = Stock\_close\_capm.index

returns = Stock\_close\_capm['exs\_rtn\_nvda']

# Plot scatter

plt.figure(figsize=(12, 6))

plt.scatter(dates, returns, color='blue', alpha=0.6, label="Daily Excess Return")

# Optionally: add trend line using polyfit (convert dates to numbers)

import matplotlib.dates as mdates

date\_nums = mdates.date2num(dates)

m, c = np.polyfit(date\_nums, returns, 1)

plt.plot(dates, m \* date\_nums + c, color='red', label=f'Trend line')

# Title and labels

plt.title('NVIDIA Excess Return Over Time', fontsize=14)

plt.xlabel('Date')

plt.ylabel('Excess Return')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

Scatter Plots

import numpy as np

import matplotlib.pyplot as plt

# Fit a line: y = mx + c

m, c = np.polyfit(Stock\_close\_capm['exs\_rtn\_nvda'], Stock\_close\_capm['MKT\_prm'], 1)

# Plot scatter

plt.figure(figsize=(8, 6))

plt.scatter(Stock\_close\_capm['exs\_rtn\_nvda'], Stock\_close\_capm['MKT\_prm'], color='blue', alpha=1, label="Data points")

# Plot fitted line

x\_vals = np.array(plt.gca().get\_xlim()) # Get x limits from the plot

y\_vals = m \* x\_vals + c

plt.plot(x\_vals, y\_vals, color='red', label=f'Fit: y = {m:.2f}x + {c:.2f}')

# Titles and labels

plt.title('Scatter Plot of Nvidia Returns vs S&P 500 Returns', fontsize=14)

plt.xlim(-0.09,0.15)

plt.ylim(-0.09,0.15)

plt.xlabel('S&P 500 Returns in %', fontsize=12)

plt.ylabel('Nvidia Returns in %', fontsize=12)

plt.legend()

plt.grid(True)

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Drop missing values if any

plt.figure(figsize=(10, 6))

sns.histplot(Stock\_close\_capm['exs\_rtn\_nvda'], bins=50, kde=True, color='steelblue')

plt.title('Histogram of Stock Returns with NVDA')

plt.xlabel('Daily Return')

plt.ylabel('Density')

plt.grid(True)

plt.show()

pairplot = sns.pairplot(Stock\_close\_capm)

pairplot

import statsmodels.formula.api as smf

import pandas as pd

# # Fit the regression model

reg\_nvda = smf.ols('exs\_rtn\_nvda ~ MKT\_prm', data=Stock\_close\_capm).fit()

# # Print the summary

print(reg\_nvda.summary())

alpha = reg\_nvda.params['Intercept']

beta = reg\_nvda.params['MKT\_prm']

r\_squared\_capm = reg\_nvda.rsquared

adj\_r\_squared\_capm = reg\_nvda.rsquared\_adj

print(f"Alpha (α): {alpha}")

print(f"Beta (β): {beta}")

print(f"R-squared: {r\_squared\_capm}")

print(f"Adjusted R-squared: {adj\_r\_squared\_capm}")

# Example: Use CAPM equation to compute expected return

# Expected return = α + β \* market\_premium

# You can replace `market\_premium\_value` with an actual value

# market\_premium\_value = (1+Stock\_close\_capm['MKT\_prm'].mean())\*\*252-1 # Example market premium (6%)

# expected\_return = alpha + beta \* market\_premium\_value

# print(f"Expected Return: {expected\_return\*100:.4f}")

import numpy as np

# Get the residuals from the regression

residuals = reg\_nvda.resid

# Mean Squared Error

mse = np.mean(residuals\*\*2)

# Root Mean Squared Error

rmse = np.sqrt(mse)

print(f"Mean Squared Error (MSE): {mse:.6f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.6f}")

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

# Create polynomial features (degrees 2 and 3)

poly = PolynomialFeatures(degree=2)

X\_poly = poly.fit\_transform(Stock\_close\_capm[['MKT\_prm']])

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, Stock\_close\_capm['exs\_rtn\_nvda'], test\_size=0.2, random\_state=42)

# Fit the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predictions

y\_pred = model.predict(X\_test)

# Evaluate performance (use RMSE, R-squared, etc.)

from sklearn.metrics import mean\_squared\_error

import numpy as np

# RMSE calculation

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"RMSE for Polynomial Regression (Degree 2): {rmse}")

from sklearn.model\_selection import cross\_val\_score

# Cross-validation for degree 2 polynomial

cv\_scores = cross\_val\_score(model, X\_poly, Stock\_close\_capm['exs\_rtn\_nvda'], cv=5, scoring='neg\_mean\_squared\_error')

mean\_cv\_score = np.mean(cv\_scores)

print(f"Mean CV Error (Degree 2 Polynomial): {mean\_cv\_score}")

# Cross-validation MSE (make positive)

cv\_scores = cross\_val\_score(model, X\_poly, Stock\_close\_capm['exs\_rtn\_nvda'], cv=5, scoring='neg\_mean\_squared\_error')

mean\_cv\_mse = -np.mean(cv\_scores)

print(f"Mean CV MSE (Degree 2 Polynomial): {mean\_cv\_mse:.6f}")

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error

X = Stock\_close\_capm[['MKT\_prm']]

y = Stock\_close\_capm['exs\_rtn\_nvda']

max\_degree = 10

train\_errors = []

val\_errors = []

for degree in range(1, max\_degree+1):

poly = PolynomialFeatures(degree=degree)

X\_poly = poly.fit\_transform(X)

model = LinearRegression()

mse\_train = cross\_val\_score(model, X\_poly, y, cv=5, scoring='neg\_mean\_squared\_error')

train\_errors.append(-np.mean(mse\_train))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, y, test\_size=0.2, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

val\_errors.append(mean\_squared\_error(y\_test, y\_pred))

plt.figure(figsize=(10,6))

plt.plot(range(1, max\_degree+1), train\_errors, label='Train Error', marker='o')

plt.plot(range(1, max\_degree+1), val\_errors, label='Validation Error', marker='o')

plt.title('Bias-Variance Tradeoff')

plt.xlabel('Polynomial Degree')

plt.ylabel('Mean Squared Error')

plt.legend()

plt.grid(True)

plt.show()

best\_degree = np.argmin(val\_errors) + 1

print(f"The best polynomial degree is: {best\_degree}")

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Prepare the data

X = Stock\_close\_capm[['MKT\_prm']]

y = Stock\_close\_capm['exs\_rtn\_nvda']

Stock\_close\_capm = Stock\_close\_capm.sort\_index()

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

from xgboost import XGBRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

xgb\_capm = XGBRegressor(objective='reg:squarederror', random\_state=42, n\_estimators=100)

xgb\_capm.fit(X\_train, y\_train)

# Step 4: Predict + Evaluate

y\_pred = xgb\_capm.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

n, p = X\_test.shape

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

# Step 5: Output results

print("XGBoost CAPM Performance:")

print(f"MSE: {mse}")

print(f"RMSE: {rmse}")

print(f"R²: {r2}")

print(f"Adjusted R²: {adj\_r2}")

from xgboost import XGBRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

import statsmodels.api as sm

import statsmodels.formula.api as smf

# Step 1: CAPM XGBoost

xgb\_capm = XGBRegressor(objective='reg:squarederror', random\_state=42, n\_estimators=100)

xgb\_capm.fit(X\_train, y\_train)

y\_pred\_xgb = xgb\_capm.predict(X\_test)

# Evaluate XGBoost

mse\_xgb = mean\_squared\_error(y\_test, y\_pred\_xgb)

r2\_xgb = r2\_score(y\_test, y\_pred\_xgb)

adj\_r2\_xgb = 1 - (1 - r2\_xgb) \* (len(y\_test) - 1) / (len(y\_test) - X\_test.shape[1] - 1)

print("\n XGBoost CAPM Performance:")

print(f"MSE: {mse\_xgb}")

print(f"R²: {r2\_xgb}")

print(f"Adjusted R²: {adj\_r2\_xgb}")

# Step 2: CAPM Linear Regression (OLS)

reg\_capm = smf.ols('exs\_rtn\_nvda ~ MKT\_prm', data=Stock\_close\_capm.iloc[:train\_size]).fit()

X\_test\_sm = sm.add\_constant(X\_test)

y\_pred\_ols = reg\_capm.predict(X\_test\_sm)

# Evaluate OLS

mse\_ols = mean\_squared\_error(y\_test, y\_pred\_ols)

r2\_ols = r2\_score(y\_test, y\_pred\_ols)

adj\_r2\_ols = 1 - (1 - r2\_ols) \* (len(y\_test) - 1) / (len(y\_test) - X\_test.shape[1] - 1)

print("\n CAPM Linear Regression on Test Data:")

print(f"MSE: {mse\_ols}")

print(f"R²: {r2\_ols}")

print(f"Adjusted R²: {adj\_r2\_ols}")

------------------------------------------------------------------------------------------------------------------from xgboost import XGBRegressor

from sklearn.model\_selection import GridSearchCV

# Define base model

xgb = XGBRegressor(objective='reg:squarederror', random\_state=42)

# Define search grid

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [2, 3, 4, 5],

'learning\_rate': [0.01, 0.1, 0.3],

'subsample': [0.8, 1.0]

}

# GridSearchCV

grid\_search = GridSearchCV(

estimator=xgb,

param\_grid=param\_grid,

scoring='neg\_mean\_squared\_error',

cv=5,

verbose=1,

n\_jobs=-1

)

# Fit on training data

grid\_search.fit(X\_train, y\_train)

# Best model

best\_xgb = grid\_search.best\_estimator\_

# Predict on test set

y\_pred = best\_xgb.predict(X\_test)

# Evaluate

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

n, p = X\_test.shape

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

rmse = mse \*\* 0.5

print("\n Tuned XGBoost CAPM Performance:")

print(f"Best params: {grid\_search.best\_params\_}")

print(f"MSE: {mse}")

print(f"RMSE: {rmse}")

print(f"R²: {r2}")

print(f"Adjusted R²: {adj\_r2}")

df = Stock\_close\_capm.copy()

# Create new features

df['MKT\_prm\_squared'] = df['MKT\_prm'] \*\* 2

df['MKT\_prm\_lag1'] = df['MKT\_prm'].shift(1)

df['VOLATILITY\_5d'] = df['MKT\_prm'].rolling(window=5).std()

df['MOMENTUM\_10d'] = df['MKT\_prm'].pct\_change(periods=10)

# Drop NaN rows from rolling and lag

df = df.dropna()

# Define features and target

X = df[['MKT\_prm', 'MKT\_prm\_squared', 'MKT\_prm\_lag1', 'VOLATILITY\_5d', 'MOMENTUM\_10d']]

y = df['exs\_rtn\_nvda']

# Lag features

df['MKT\_prm\_lag1'] = df['MKT\_prm'].shift(1)

df['MKT\_prm\_lag2'] = df['MKT\_prm'].shift(2)

df['MKT\_prm\_lag3'] = df['MKT\_prm'].shift(3)

df['exs\_rtn\_nvda\_lag1'] = df['exs\_rtn\_nvda'].shift(1)

df['exs\_rtn\_nvda\_lag5'] = df['exs\_rtn\_nvda'].shift(5)

# Rolling stats on MKT\_prm

df['rolling\_mean\_5d'] = df['MKT\_prm'].rolling(window=5).mean()

df['rolling\_std\_5d'] = df['MKT\_prm'].rolling(window=5).std()

# Rolling stats on exs\_rtn\_nvda

df['rolling\_mean\_nvda\_5d'] = df['exs\_rtn\_nvda'].rolling(window=5).mean()

df['rolling\_std\_nvda\_5d'] = df['exs\_rtn\_nvda'].rolling(window=5).std()

# Nonlinear term

df['MKT\_prm\_squared'] = df['MKT\_prm'] \*\* 2

# Drop rows with NaNs from rolling/lags

df = df.dropna()

# Final feature list

X = df[[

'MKT\_prm',

'MKT\_prm\_squared',

'MKT\_prm\_lag1',

'MKT\_prm\_lag2',

'MKT\_prm\_lag3',

'exs\_rtn\_nvda\_lag1',

'exs\_rtn\_nvda\_lag5',

'rolling\_mean\_5d',

'rolling\_std\_5d',

'rolling\_mean\_nvda\_5d',

'rolling\_std\_nvda\_5d'

]]

y = df['exs\_rtn\_nvda']

split\_index = int(len(X) \* 0.8)

X\_train, X\_test = X.iloc[:split\_index], X.iloc[split\_index:]

y\_train, y\_test = y.iloc[:split\_index], y.iloc[split\_index:]

from xgboost import XGBRegressor

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_squared\_error, r2\_score

param\_grid = {

'n\_estimators': [50, 100],

'max\_depth': [2, 3, 4],

'learning\_rate': [0.01, 0.1],

'subsample': [0.8, 1.0]

}

xgb = XGBRegressor(objective='reg:squarederror', random\_state=42)

grid = GridSearchCV(xgb, param\_grid=param\_grid, scoring='neg\_mean\_squared\_error', cv=5, verbose=1, n\_jobs=-1)

grid.fit(X\_train, y\_train)

best\_xgb = grid.best\_estimator\_

y\_pred = best\_xgb.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = mse \*\* 0.5

r2 = r2\_score(y\_test, y\_pred)

n, p = X\_test.shape

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

print("\n📊 Tuned XGBoost with Engineered CAPM Features:")

print(f"Best Params: {grid.best\_params\_}")

print(f"MSE: {mse}")

print(f"RMSE: {rmse}")

print(f"R²: {r2}")

print(f"Adjusted R²: {adj\_r2}")

## Important features of XGBoost

from xgboost import plot\_importance

import matplotlib.pyplot as plt

plot\_importance(best\_xgb)

plt.title("XGBoost Feature Importance")

plt.show()

----------------------------------------------------------------------------------------------------------------

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Prepare your data

X = Stock\_close\_capm[['MKT\_prm']]

y = Stock\_close\_capm['exs\_rtn\_nvda']

# Step 2: Time-aware split

split\_index = int(len(X) \* 0.8)

X\_train, X\_test = X.iloc[:split\_index], X.iloc[split\_index:]

y\_train, y\_test = y.iloc[:split\_index], y.iloc[split\_index:]

# Step 3: Fit Random Forest model

rf = RandomForestRegressor(n\_estimators=1000, random\_state=42)

rf.fit(X\_train, y\_train)

# Step 4: Predict and evaluate

y\_pred\_rf = rf.predict(X\_test)

mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

rmse\_rf = mse\_rf \*\* 0.5

r2\_rf = r2\_score(y\_test, y\_pred\_rf)

n, p = X\_test.shape

adj\_r2\_rf = 1 - (1 - r2\_rf) \* (n - 1) / (n - p - 1)

# Step 5: Print results

print("📊 Random Forest (CAPM):")

print(f"MSE: {mse\_rf}")

print(f"RMSE: {rmse\_rf}")

print(f"R²: {r2\_rf}")

print(f"Adjusted R²: {adj\_r2\_rf}")

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

import numpy as np

# Data setup (use original or smoothed version)

X = Stock\_close\_capm[['MKT\_prm']]

y = Stock\_close\_capm['exs\_rtn\_nvda']

# Time-aware split

split\_index = int(len(X) \* 0.8)

X\_train, X\_test = X.iloc[:split\_index], X.iloc[split\_index:]

y\_train, y\_test = y.iloc[:split\_index], y.iloc[split\_index:]

# Grid of parameters to tune

param\_grid = {

'max\_depth': [2, 4, 6, 8, 10, None],

'max\_features': ['auto', 'sqrt', 'log2']

}

# Model setup

rf = RandomForestRegressor(n\_estimators=1000, random\_state=42)

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=3, scoring='neg\_mean\_squared\_error', n\_jobs=-1)

# Fit

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

# Predict

y\_pred = best\_model.predict(X\_test)

# Evaluate

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

n, p = X\_test.shape

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

# Output

print("📊 Tuned Random Forest:")

print(f"Best max\_depth: {best\_model.max\_depth}")

print(f"Best max\_features: {best\_model.max\_features}")

print(f"MSE: {mse}")

print(f"RMSE: {rmse}")

print(f"R²: {r2}")

print(f"Adjusted R²: {adj\_r2}")

# 5 Factor Fama french model

url = '<https://github.com/Amanjhagta/Data-Science/blob/main/Data/fama-french/F-F_Research_Data_5_Factors_2x3_daily.CSV?raw=True>'

factor\_data = pd.read\_csv(url)

factor\_data['date'] = pd.to\_datetime(factor\_data['date'], format='%Y%m%d')

factor\_data.head()

Stock\_close\_FF = pd.merge(Stock\_close, factor\_data, on="date", how="left")

Stock\_close\_FF['exs\_rtn\_nvda'] = Stock\_close\_FF['NVDA'] - Stock\_close\_FF['RF']

Stock\_close\_FF = Stock\_close\_FF.rename(columns={'Mkt-RF': 'MKT\_prmf'})

Stock\_close\_FF.head()

Stock\_close\_FF\_fltr = Stock\_close\_FF[['date','exs\_rtn\_nvda','MKT\_prm','SMB','HML','RMW','CMA']]

# Stock\_close\_FF\_fltr = Stock\_close\_FF\_fltr.rename(columns={'Mkt-RF': 'MKT\_prm'})

Stock\_close\_FF\_fltr.head()

Stock\_close\_FF\_fltr = Stock\_close\_FF\_fltr.set\_index('date')

Stock\_close\_FF\_fltr.head()

Stock\_close\_FF\_fltr.info()

pairplot\_ff = sns.pairplot(Stock\_close\_FF\_fltr)

pairplot\_ff

plt.figure(figsize=(8, 5))

# Draw the heatmap

sns.heatmap(correlation, annot=True, cmap='coolwarm', center=0, linewidths=0.5, fmt=".2f")

# # Add a title

plt.title('Correlation Matrix Heatmap', fontsize=16)

# Show the plot

plt.show()

Stock\_close\_FF\_fltr.plot(y='exs\_rtn\_nvda', title='Excess Return of NVDA',figsize=(12, 8))

Stock\_close\_FF\_fltr[['MKT\_prm', 'SMB', 'HML', 'RMW', 'CMA']].plot(subplots=True, figsize=(12, 8))

import matplotlib.pyplot as plt

# Assuming you already have Stock\_close\_FF\_fltr loaded

plt.figure(figsize=(12, 6))

Stock\_close\_FF\_fltr.boxplot()

plt.title("Box Plot of Fama-French Factors and Excess Return (NVDA)")

plt.ylabel("Return")

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Drop missing values if any

plt.figure(figsize=(10, 6))

sns.histplot(Stock\_close\_FF\_fltr['exs\_rtn\_nvda'], bins=50, kde=True, color='steelblue')

plt.title('Histogram of Stock Returns with NVDA')

plt.xlabel('Daily Return')

plt.ylabel('Density')

plt.grid(True)

plt.show()

# Fit the regression model

reg\_nvda\_ff = smf.ols('exs\_rtn\_nvda ~ MKT\_prm+SMB+HML+RMW+CMA', data=Stock\_close\_FF\_fltr).fit()

# Print the summary

print(reg\_nvda\_ff.summary())

# Extracting coefficients (alpha and betas for each factor)

alpha\_ff = reg\_nvda\_ff.params['Intercept']

beta\_market = reg\_nvda\_ff.params['MKT\_prm']

beta\_smb = reg\_nvda\_ff.params['SMB']

beta\_hml = reg\_nvda\_ff.params['HML']

beta\_rmw = reg\_nvda\_ff.params['RMW']

beta\_cma = reg\_nvda\_ff.params['CMA']

# Extract R-squared and Adjusted R-squared

r\_squared\_ff = reg\_nvda\_ff.rsquared

adj\_r\_squared\_ff = reg\_nvda\_ff.rsquared\_adj

# Print results

print(f"Alpha (α): {alpha\_ff}")

print(f"Beta (Market) (β\_MKT): {beta\_market}")

print(f"Beta (SMB) (β\_SMB): {beta\_smb}")

print(f"Beta (HML) (β\_HML): {beta\_hml}")

print(f"Beta (RMW) (β\_RMW): {beta\_rmw}")

print(f"Beta (CMA) (β\_CMA): {beta\_cma}")

print(f"R-squared: {r\_squared\_ff}")

print(f"Adjusted R-squared: {adj\_r\_squared\_ff}")

# # Optional: Compute expected return based on a given market premium

# market\_premium\_value\_ff = Stock\_close\_FF\_fltr['MKT\_prm'].mean() # Example market premium (6%)

# smb\_value = Stock\_close\_FF\_fltr['SMB'].mean() # Example SMB factor

# hml\_value = Stock\_close\_FF\_fltr['HML'].mean() # Example HML factor

# rmw\_value = Stock\_close\_FF\_fltr['RMW'].mean() # Example RMW factor

# cma\_value = Stock\_close\_FF\_fltr['CMA'].mean() # Example CMA factor

# # Compute expected return using the Fama-French 5-factor model equation

# expected\_return\_ff = (alpha\_ff +

# beta\_market \* market\_premium\_value\_ff +

# beta\_smb \* smb\_value +

# beta\_hml \* hml\_value +

# beta\_rmw \* rmw\_value+

# beta\_cma \* cma\_value)

# print(f"Expected Return (FF5): {expected\_return\_ff:.4f}")

from sklearn.metrics import mean\_squared\_error

import numpy as np

# Actual and predicted values

y\_actual = Stock\_close\_FF\_fltr['exs\_rtn\_nvda']

y\_pred\_ff = reg\_nvda\_ff.fittedvalues

# Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)

mse\_ff = mean\_squared\_error(y\_actual, y\_pred\_ff)

rmse\_ff = np.sqrt(mse\_ff)

print(f"Fama-French MSE: {mse\_ff:.6f}")

print(f"Fama-French RMSE: {rmse\_ff:.6f}")

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import cross\_val\_score

import numpy as np

import matplotlib.pyplot as plt

# Define X and y

X = Stock\_close\_FF\_fltr[['MKT\_prm', 'SMB', 'HML', 'RMW', 'CMA']]

y = Stock\_close\_FF\_fltr['exs\_rtn\_nvda']

# Range of degrees to test

degrees = range(1, 6)

mean\_mse\_scores = []

# Loop through each degree

for degree in degrees:

poly = PolynomialFeatures(degree=degree, include\_bias=False)

X\_poly = poly.fit\_transform(X)

model = LinearRegression()

mse\_scores = cross\_val\_score(model, X\_poly, y,

scoring='neg\_mean\_squared\_error', cv=30)

mean\_mse = -np.mean(mse\_scores) # Convert to positive MSE

mean\_mse\_scores.append(mean\_mse)

print(f"Degree {degree} - Mean CV MSE: {mean\_mse:.6f}")

# Plot the results

plt.plot(degrees, mean\_mse\_scores, marker='o')

plt.title("Polynomial Degree vs CV MSE")

plt.xlabel("Polynomial Degree")

plt.ylabel("Mean Cross-Validated MSE")

plt.grid(True)

plt.show()

# Best degree

best\_degree = degrees[np.argmin(mean\_mse\_scores)]

print(f"\n Best Polynomial Degree: {best\_degree}")

# Step 1: Ensure data is sorted by date

Stock\_close\_FF\_fltr = Stock\_close\_FF\_fltr.sort\_index()

# Step 2: Define features and target

X = Stock\_close\_FF\_fltr[['MKT\_prm', 'SMB', 'HML', 'RMW', 'CMA']]

y = Stock\_close\_FF\_fltr['exs\_rtn\_nvda']

# Step 3: Time-aware split (80% train, 20% test)

split\_index = int(len(X) \* 0.8)

X\_train = X.iloc[:split\_index]

X\_test = X.iloc[split\_index:]

y\_train = y.iloc[:split\_index]

y\_test = y.iloc[split\_index:]

import statsmodels.api as sm

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Use the same formula, but fit on training data only

formula = 'exs\_rtn\_nvda ~ MKT\_prm + SMB + HML + RMW + CMA'

reg\_nvda\_ff\_train = smf.ols(formula=formula, data=Stock\_close\_FF\_fltr.iloc[:split\_index]).fit()

# Step 2: Predict on test set manually using model.predict

X\_test\_sm = sm.add\_constant(X\_test) # statsmodels expects intercept

y\_pred\_ff = reg\_nvda\_ff\_train.predict(X\_test\_sm)

# Step 3: Evaluate performance

mse = mean\_squared\_error(y\_test, y\_pred\_ff)

r2 = r2\_score(y\_test, y\_pred\_ff)

n = len(y\_test)

p = X\_test.shape[1]

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

# Step 4: Print results

print(" Fama-French OLS (Test Data):")

print(f"MSE: {mse}")

print(f"R²: {r2}")

print(f"Adjusted R²: {adj\_r2}")

import matplotlib.pyplot as plt

# Make sure the test set index (date) is preserved

y\_test\_plot = y\_test.copy()

y\_pred\_plot = pd.Series(y\_pred\_ff, index=y\_test.index)

# Plot actual vs predicted

plt.figure(figsize=(14, 6))

plt.plot(y\_test\_plot.index, y\_test\_plot, label='Actual Return', alpha=0.7)

plt.plot(y\_pred\_plot.index, y\_pred\_plot, label='Predicted Return (Fama-French OLS)', alpha=0.7)

plt.xlabel("Date")

plt.ylabel("Excess Return (NVDA)")

plt.title("Actual vs Predicted Excess Returns (Test Set)")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Convert 'Fiscal Year' to datetime (we'll assume it's Jan 1 of each year)

dfn['Fiscal Year'] = pd.to\_datetime(dfn['Fiscal Year'], format='%Y')

# Set it as index (optional but good for time series plotting)

dfn.set\_index('Fiscal Year', inplace=True)

# Plot

plt.figure(figsize=(10, 6))

plt.plot(dfn.index, dfn['Revenue (USD)'], label='Revenue (USD)', marker='o')

plt.plot(dfn.index, dfn['Operating Income (USD)'], label='Operating Income (USD)', marker='o')

plt.plot(dfn.index, dfn['Net Income (USD)'], label='Net Income (USD)', marker='o')

plt.title('Company Financials Over Time')

plt.xlabel('Fiscal Year')

plt.ylabel('USD (Billions)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

param\_grid = {

'n\_estimators': [50, 100],

'max\_depth': [2, 3, 4],

'learning\_rate': [0.01, 0.1],

'subsample': [0.8, 1.0]

}

xgb = XGBRegressor(objective='reg:squarederror', random\_state=42)

grid = GridSearchCV(xgb, param\_grid=param\_grid, scoring='neg\_mean\_squared\_error', cv=5, verbose=1, n\_jobs=-1)

grid.fit(X\_train, y\_train)

best\_xgb = grid.best\_estimator\_

# Step 4: Predict and evaluate

y\_pred = best\_xgb.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = mse \*\* 0.5

r2 = r2\_score(y\_test, y\_pred)

n, p = X\_test.shape

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

# Step 5: Output

print("\nXGBoost Fama-French Model (Tuned):")

print(f"Best Params: {grid.best\_params\_}")

print(f"MSE: {mse}")

print(f"RMSE: {rmse}")

print(f"R²: {r2}")

print(f"Adjusted R²: {adj\_r2}")

from xgboost import plot\_importance

plt.figure(figsize=(10, 5))

plot\_importance(best\_xgb, max\_num\_features=10, importance\_type='gain')

plt.title("XGBoost Feature Importance (Gain)")

plt.tight\_layout()

plt.show()

# Step 3: Train Random Forest

rf\_ff = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_ff.fit(X\_train, y\_train)

# Step 4: Predict and evaluate

y\_pred\_rf\_ff = rf\_ff.predict(X\_test)

mse\_rf\_ff = mean\_squared\_error(y\_test, y\_pred\_rf\_ff)

rmse\_rf\_ff = mse\_rf\_ff \*\* 0.5

r2\_rf\_ff = r2\_score(y\_test, y\_pred\_rf\_ff)

n, p = X\_test.shape

adj\_r2\_rf\_ff = 1 - (1 - r2\_rf\_ff) \* (n - 1) / (n - p - 1)

# Step 5: Output

print("\n📊 Random Forest (Fama-French Model):")

print(f"MSE: {mse\_rf\_ff}")

print(f"RMSE: {rmse\_rf\_ff}")

print(f"R²: {r2\_rf\_ff}")

print(f"Adjusted R²: {adj\_r2\_rf\_ff}")