Advanced Programming 2025

Enhancing Cross-Sectional Momentum Strategies through Machine Learning and Volatility Targeting

Final Project Report

Francisco Manuel Lameiras dos Santos
francisco.lameirasdossantos@unil.ch
Student ID: 25415282

January 11, 2026

Abstract

Momentum is an anomaly of financial economics that has been heavily researched. However, portfolios built with this strategy suffer from drawdowns when there are market reversals. This analysis presents a machine-learning approach to Cross-Sectional Momentum, accompanied by dynamic volatility targeting. By using XGBoost and Random Forest modeling to predict and rank US equities, the main objective is to evaluate how well these models perform against linear baselines.

The idiosyncratic alpha of returns is isolated using a market-neutral forward target and the model is continuously validated using an expanding walk-forward window for the testing, between 2019 and 2024. The XGBoost and Random Forest produced volatility-targeted portfolios with average annual returns of 12.29% and 11.71% and Sharpe Ratios of 0.67 and 0.65 respectively, compared to just 2.6% and 0.14 for an OLS Linear Regression. Furthermore, the volatility targeting mechanism allowed for the reduction of crash risk, especially during 2020 with the COVID-19 market downfall and the 2022 market crash. This analysis suggests that Momentum-Based strategies can be optimized with the interpretation of non-linear signals and dynamic risk management.

Keywords: Algorithmic Trading, Cross-Sectional Momentum, XGBoost, Volatility Targeting, Walk-Forward Validation, S&P 500, Machine Learning.

Contents

1 Introduction

Cross-Sectional Momentum is one of the most robust anomalies of financial markets. An investment strategy based on this feature involves buying "winners" and selling "losers". What this means is that an investor would buy the securities that performed better recently and sell the ones that performed poorly, which in an extreme case could be done by short-selling the underperforming securities. It was first documented by Jegadeesh and Titman (1993) and it suggests that markets are not completely efficient since past prices may have some predictive information. However, traditional strategies are "naive" and rely mainly on linear rankings, causing massive drawdowns when financial markets crash (e.g., 2009, 2020).

A major fault of standard momentum strategies is not adapting to market fluctuations, as such the portfolio volatility stays unmanaged, this being the cause of big drawdowns. Additionally, a common problem with machine learning applications in finance is that there exists look-ahead bias, meaning that the models are tested with models that already incorporate information from the test data. Including illiquid securities for which trading is not very feasible is also a major problem, as transaction costs would most likely eliminate any alpha generated by the strategy.

This report presents an algorithmic strategy that improves traditional momentum with ensemble learning and volatility targeting. The specific goals are to isolate idiosyncratic alpha, by using a market-neutral target, to compare the performance of non-linear models, namely XGBoost and Random Forest against linear base models such as Lasso and linear regression. It is also relevant to mention that the strategy uses an expanding window walk-forward validation process, with the models being iteratively retrained in an increasing historical window. This allows to evaluate the analysis in different years and thus financial market scenarios.

2 Research Question & Literature Review

The optimization of momentum strategies through machine learning stems from three main points of financial literature: the cross-sectional momentum anomaly, volatility-based risk management, and non-linear asset pricing.

2.1 The Cross-Sectional Momentum Anomaly

Jegadeesh and Titman (1993) first documented that a portfolio with zero costs containing securities with recent winners outperforms recent losers over medium-term horizons, between 3 and 12 months. This "Cross-Sectional Momentum" (CSM) is distinct from "Time-Series Momentum" (Trend Following), as presented by Moskowitz, Ooi, and Pedersen (2012). Time-Series momentum analyzes absolute direction, which only concerns the question of "is the asset price going up?", whereas the Cross-Sectional Momentum focuses on how the asset performs comparatively to others. This analysis exploits CSM by ranking assets based on their standardized cumulative log-returns in different periods, namely, 1, 3, 6 and 12 months.

2.2 Volatility Targeting and Crash Mitigation

A big limitation of linear momentum strategy is negative skewness; Daniel and Moskowitz (2016) showed that momentum-based portfolios are prone to crash risk. What this means is that during market rebounds after bad performances of the financial market, assets with high beta that have had decreasing returns will go back to regular prices faster than assets with low beta that are recent "winners". Thus, a momentum strategy will cause a massive drawdown during this period. However, Barroso and Santa-Clara (2015) propose a solution by optimizing the portfolio with constant volatility targeting. What they found is that scaling the portfolio's leverage to be lower

when volatility is higher reduced lower tail risk of returns. This same approach is implemented in this analysis, targeting an annual volatility of 15%.

2.3 Non-Linearity in Asset Pricing

Traditional models such as the capital asset pricing model or Fama-French three-factor model assume a linear relationship between risk and returns. However, Gu, Kelly and Xiu (2020) provide evidence that the relationship is non-linear and high-dimensional. Their studies on U.S. equities showed that machine learning methods outperform linear regressions because they take into account the interaction effects between predictors. This analysis also draws from their research by comparing XGBoost and Random Forest models against Lasso and Simple linear regressions.

3 Methodology

3.1 Data Sourcing and Universe Definition

The universe of investment is composed by the constituents of the S&P 500 index. For this analysis, a static universe is estimated from an index composition available in Kaggle (`sp500_companies.csv`). The code extracts the approximately 500 tickers from Kaggle's file and then makes a request to the `yfinance` API, fetching daily Open, High, Low, Close, and Volume (OHLCV) for each ticker between January 2015 through December 2024.

Bias Consideration: Having that in mind, two forms of bias should be considered before starting the analysis of the simulation:

1. **Survivorship Bias:** Since the universe uses the *current* S&P 500 constituents, it does not account for companies that were delisted or went bankrupt during the sample period (e.g., "fallen angels"). As such, there might be a positive bias in the simulated returns as only companies that "survived" will be accounted for. This likely overestimates performance.
2. **Look-Ahead Bias:** This bias is eliminated with a 35-day Purge Buffer between training and testing sets, meaning that there is a 35-day gap between the two sets, avoiding any data leakage in the models.

Moreover, a liquidity filter is added to mitigate the impact on the strategy of trading illiquid stocks that maybe "survived" but were untradeable in the past. This is done at each rebalance date by calculating the 3-month average daily dollar volume (ADV). Therefore, the selection is restricted to the top 500 most liquid stocks of the available universe. Still, it does not fix the survivorship bias but it ensures the strategy only simulates on institutional-grade liquid assets and does not generate alpha from extremely illiquid ones.

3.2 Feature Engineering

The API used to download the data provides it in a "Wide" format where each ticker has its own column. Having that in mind, the code converts the CSV file to a "Long" format where there are approximately 500 rows for each ticker for one trading day and the same for the subsequent days. From this file another one is created adding momentum features based on cumulative log-returns. The use of this kind of returns instead of simple returns is due to their additive property over time.

For every stock i on day t, Momentum (M) is calculated over horizon H:

$$M_{i,t,H} = \sum_{k=0}^{H} \ln\left(\frac{P_{t-k}}{P_{t-k-1}}\right) \tag{1}$$

Having this equation in mind, four horizons are calculated:

- **Short-Term (Mom_{1m}):** a 21-day rolling sum, which is close to 1 month of trading, capturing short-term reversals of the stocks.
- **Intermediate (Mom_{3m}, Mom_{6m}):** 63-day and 126-day rolling sums, equivalent to approximately 3 and 6 months of trading days, respectively. These medium-length windows capture core momentum features.
- **Long-Term (Mom_{12m}):** 252-day rolling sum, close to 1 year of trading days and which captures structural trends of the assets.

Cross-Sectional Standardization: The raw returns vary with time and are heavily influenced by market regimes; for example, in 2020 the values are structurally higher than in 2017, as financial markets grow. To adjust the returns to be used in the machine learning models, Cross-Sectional Z-Scoring is applied. For each date t, a feature x is standardized for all stocks N:

$$z_{i,t} = \frac{x_{i,t} - \mu_t}{\sigma_t} \tag{2}$$

This adjustment will allow the model to train using relative rankings rather than absolute values. A stock has a good score only if it is better at the moment than others, as such it will mitigate the impact of the correlation of each asset to the market, their Beta.

3.3 Predictive Models and Configuration

To test if momentum features contain non-linear interactions, this project uses four different model architectures, ranging from a simple linear regression to advanced gradient boosting. The code to build these models uses the `scikit-learn` and `xgboost` Python libraries.

3.3.1 Linear Baselines

One of the main goals of the analysis is to compare the machine learning model's performance to a naive baseline. This naive model is the Ordinary Least Squares (OLS). Still, a Lasso regression (L1 regularization) is used as a somewhat naive simulation, but which accounts for multicollinearity between the different horizons chosen (1, 3, 6 and 12 months).

- **Configuration:** We set the regularization strength $\alpha = 0.0001$. This low-penalty value was chosen to allow the model to perform soft feature selection, zeroing out redundant horizons while maintaining predictive signal.

3.3.2 Machine-Learning Models

The other important models of the strategy are tree-based ensembles, which are used with the purpose of finding non-linear regime shifts that linear regressions could not account for.

- **Random Forest:** This model was ran with 200 decision trees. It also uses bootstrap sampling to reduce variance and prevent overfitting of data.
- **XGBoost:** It is without doubt the model with the best performance. As such its results are the ones analyzed more in depth later on.
- **Hyperparameters:** The XGBoost parameters complexity is heavily restricted in order for the analysis to be comparable to models that use other samples.
  - **Max Depth = 3:** This small value forces the model to learn simple and high-level interactions, avoiding noise from the data.

- **Learning Rate = 0.1:** This standard rate ensures smooth convergence.
- **Estimators = 50:** Also a rather conservative number of boosting rounds to prevent overfitting.

4 Implementation

The project uses a modular pipeline architecture and it is built such that the data is the same and it can be reproducible. The execution of the code is linear, starting with ingestion of data, then machine learning models and finally a simulation of portfolio.

4.1 Layer 1: Data Ingestion (`data_loader.py`)

The first part of the pipeline is the `data_loader` module. Its function is to gather the historical data of returns to be ingested in the models later on. It gets an approximated ticker list for S&P 500 stocks, which is also saved in the `data` folder, allowing for changes in the investment universe. Tickers corresponding to duplicated equities such as "GOOG", "FOX", "NWS" were dropped to eliminate data noise. The code then uses the `yfinance` API to fetch daily OHLCV (Open, High, Low, Close, Volume) data.

Furthermore, to avoid errors from the API, the requests are split into different batches with approximately 50 tickers each and the `sleep` function from the `time` module is used to give an interval between the requests.

Finally, as previously mentioned this module changes the CSV data file from "Wide" Format to "Long" Format, which makes it ready to be trained and tested by the linear and machine learning models in subsequent modules.

4.2 Layer 2: The Orchestrator (`main.py`)

The `main.py` script is the main module for the project. It orchestrates the entire code, calling other modules. The code also has independent functions, preparing the data for the models, namely setting up the predictive target and the walk-forward loop. In what concerns the linear and ML model preparation, the file is split into three steps:

Step A: Feature Generation Call First the `calculate_momentum` module is called. This is where the raw prices are converted into log-returns for different rolling windows, 1, 3, 6 and 12 months. Then, in the `main.py` file the data is standardized with z-score normalization, ensuring that the momentum features represent relative rankings instead of absolute values.

```
# Target Engineering: Market-Neutral Forward Return
df['fwd_ret'] = df.groupby('ticker')['close'].shift(-21) / df['close'] - 1
df['target'] = df.groupby('date')['fwd_ret'].transform(lambda x: x - x.mean())

# Expanding Window Loop
for year in range(2019, 2025):
    test_start_date = pd.Timestamp(f'{year}-01-01')

    # PURGE PROTOCOL: End training 35 days before test starts
    # This prevents the model from seeing the target of the last training rows
    train_cutoff = test_start_date - pd.Timedelta(days=35)

    train_data = df[df['date'] < train_cutoff]
    test_data = df[(df['date'] >= test_start_date) &
                   (df['date'] < test_start_date + pd.DateOffset(years=1))]

    # Train Model (XGBoost)
    model.fit(train_data[features], train_data['target'])
```

Listing 1: Walk-Forward Validation with 35-Day Purge

Step B: Target Engineering With the momentum features ready, `main.py` also sets up the target variable to be inputted in the predictive models, which is defined as the Market-Neutral Forward Return.

```
# Shift Close price by -21 days to get future return
df['fwd_ret_1m'] = df.groupby('ticker')['close'].shift(-21) / df['close'] - 1

# Subtract the cross-sectional mean to isolate Alpha
df['fwd_ret_1m_neutral'] = df.groupby('date')['fwd_ret_1m'].transform(
    lambda x: x - x.mean()
)
```

Listing 2: Target Engineering in main.py

Step C: Walk-Forward Loop The orchestrator module also sets up the expanding window validation which is key to ensure the strategy is consistent across years. It iterates through each year from 2019 to 2024, and defines the training set as all the years before the one where the data is trained (e.g. in 2020 the training set is 2015-2019 and test in 2020). Then it adds the 35-day purge buffer to prevent look-ahead bias, trains the linear and ML models in the training set generating predictions for the test year for each iteration.

4.3 Layer 3: Execution and Risk (`backtest.py`)

This final module of the pipeline takes the predictions from the different models and builds portfolios throughout the period, which is the best way to visualize how well the strategy worked and assess the quality of the predictive models. The portfolios were built with the following logic:

1. **Ranking:** The stocks were sorted everyday according to the predicted alpha (excess return compared to the market average).
2. **Selection:** The top 10% assets are bought (Long) and the bottom 10% are sold (Short). To the long portfolio a weight of 1.0 is assigned and to the short only 0.5, again a conservative approach to avoid high leverage.
3. **Volatility Targeting:** Like mentioned before, this module calculates the 12-month realized volatility of the portfolio. After, it computes a scalar $S_t = \frac{15\%}{\sigma_{realized}}$ and if the market volatility spikes, S_t drops below 1.0, and the portfolio is de-levered in order to reduce crashes. This method draws directly from Barroso and Santa-Clara's (2015) approach to mitigate crash risk on a momentum-based portfolio.

5 Codebase & Reproducibility

The Python project is fully reproducible from the GitHub repository. The dependencies are present in a conda environment file to ensure version compatibility, specifically for xgboost and pandas.

How to Run the code:

1. The first step is to clone the repository by writing in the terminal: `git clone https://github.com/franciscomldsantos/DSAP_Project`
2. Then the dependencies need to be installed: `conda env create -f environment.yml`

3. (optional) The data loader can be run, but due to inconsistencies with `yfinance` API, the dataset is also provided inside the folder `data` which is downloaded inside a `zip` file. To request the data again, the following input can be used: `python src/data_loader.py` (Downloads sp500 data)
4. Momentum features are calculated: `python src/calculate_momentum.py`
5. The final step is to execute the pipeline with the orchestrator: `python main.py`

Key Dependencies:

- `pandas (2.0+)`: Essential in the feature engineering to calculate momentum base on the stock prices.
- `xgboost (1.7+)`: For the main Machine Learning Model
- `yfinance`: For fetching historical market data.

6 Results

6.1 The Setup

The results of the analysis are mostly drawn from the information provided by the backtest of the predictions. Nonetheless, before analyzing the results it is important to clarify the setup of the XGBoost model, which is the one with the best results and which are analyzed in more depth. The analysis was conducted in an environment using Python 3.10 and XGBoost model was configured with conservative hyperparameter, which are as follows:

Table 1: Model Hyperparameters

Parameter	**Value**	**Rationale**
Estimators	50	Reduces bias, while preventing overfitting
Learning Rate	0.1	Standard shrinkage for boosting
Max Depth	3	Gives more importance to meaningful interactions
Subsample	0.8	Stochastic training to improve generalization

6.2 Performance Evaluation

The performance of the models was backtested over the out-of-sample period from January 2019 to January 2025. In this period there was a lot of market volatility, such as the Covid-19 crash or the sharp increase of AI stocks, which allow to test the models under different financial market regimes.

Table 2: Backtest Performance Metrics Vol-Target (2019-2025)

Model	**Ann. Return**	**Sharpe Ratio**	**Max Drawdown**
XGBoost	**12.29%**	**0.67**	**-21.02%**
Random Forest	11.71%	0.65	-19.76%
Lasso	1.87%	0.1	-33.5%
Linear Regression	2.6%	0.14	-33.6%

The XGBoost model had much higher risk-adjusted returns compared to all baselines and to the random forest model. The linear models (Lasso and OLS) had a big drawdown correlation during 2022, suggesting that the models allocate too much weight to high-beta losers, which

are stocks that are heavily correlated to the market, but if the market rebounds, so does the high-beta loser causing a massive drawdown to the portfolio.

6.3 Visualizations: Model Comparison

The cumulative returns graph (Figure 1) illustrates the performance of the four models. It is clear that in 2022 the linear models returns stagnated, yet, the machine learning ones kept rising, with both the XGBoost and Random Forest exhibiting increasing returns. Between the two linear models (OLS and Lasso) their trend is very similar with Lasso slightly outperforming the OLS regression. Still, when looking at their drawdown profile, OLS has a greater crash risk in early years, although Lasso's drawdown is bigger in the following years. The drawdown profile of the ML models is much more controlled, never going below -25%.

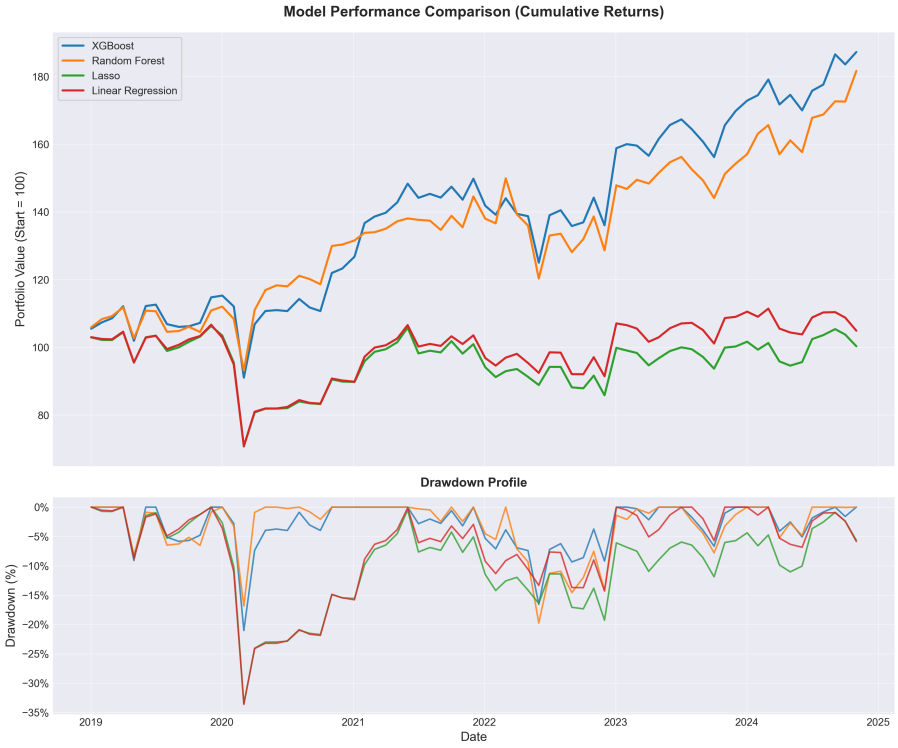

Figure 1: Cumulative Returns (Log Scale). The XGBoost strategy (Blue) and Random Forest (Yellow) outperform the linear models (Green and Red), showing that ML models have better predictive power for a momentum strategy.

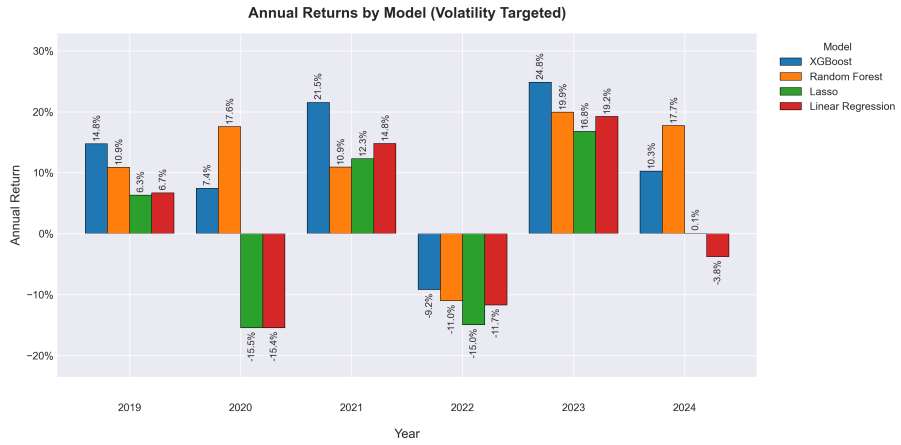

Figure 2: Annual Returns Breakdown. The XGBoost model (Blue) outperformed the benchmark in 2022, avoiding the large drawdowns seen in the market.

6.4 Visualizations: Consistency and Risk Management

The consistency of the strategy is demonstrated by the Rolling Sharpe Ratio as can be visualized in (Figure 3). Contrary to the linear baseline, which often has a negative sharpe ratio. The Machine Learning Models were able to keep positive risk-adjusted returns for about 85% of the period. From the graph it is also possible to assess that the Random Forest Model performed slightly better than XGBoost following the Covid-19 decrease and it handled the drawdowns after 2022 better.

Moreover, regarding the effectiveness of the volatility targeting mechanism, it can be seen that in Figure 4, that in periods with high volatility such as during the Covid-19 pandemic, the leverage factor fell to under 1.0 and the portfolio optimization mechanism automatically reduced leverage undertaken. This automatic process allows for a significant reduction of the maximum drawdown compared to an approach with constant leverage.

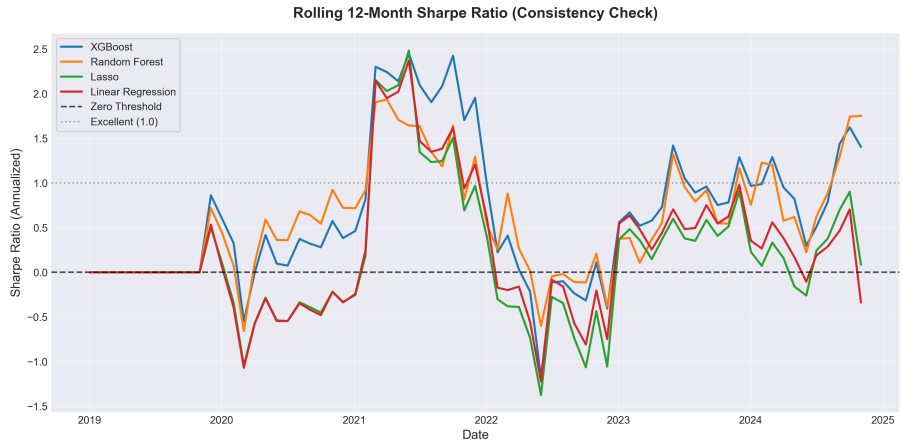

Figure 3: Rolling 12-Month Sharpe Ratio. It is generally positive, confirming consistency across different financial market scenarios.

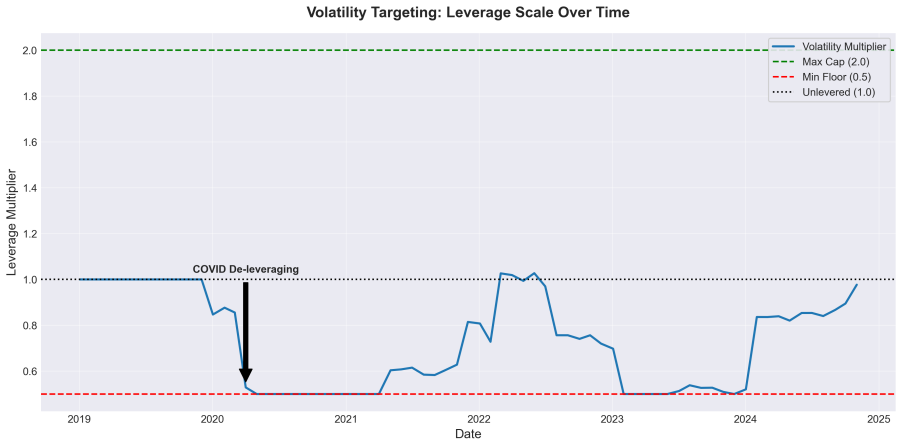

Figure 4: Dynamic Volatility Scaling. The leverage factor represented in the y axis goes below 1.0 during the March 2020 volatility spike to avoid a portfolio crash.

7 Conclusion

7.1 Summary

Based on previous literature on momentum-based investment strategies, this project builds an algorithmic trading strategy. By using an Expanding Window Walk-Forward Validation with a data purge, the integrity of results is kept. Our findings confirm that Machine Learning models is significantly more effective at capturing Cross-Sectional Momentum signals than the Linear Models, achieving an average Sharpe Ratio for the volatility targeted portfolios of approximately 0.67 (XGBoost) compared to an average of 0.14 (OLS Linear Regression).

7.2 Discussion

The significant difference between Linear Regression and Lasso models compared to the Machine Learning ones demonstrates that momentum is non-linear and is highly dependent on the financial market. In a linear model, higher momentum scores predict higher returns, but empirical evidence shows that when there is very extreme momentum, it can lead to reversals conditioned by changes in the market itself. The Machine Learning models were able to pick these features, learning that when the volatility is too high the predictions should be more conservative as there might be reversals.

7.3 Limitations & Future Work

The static S&P 500 universe introduces survivorship bias, which inflates the returns obtained in the backtest. Future work could use an historical accurate list of tickers and get data for all companies even the ones that were de-listed. This would of course require a more robust method for data download.

Also, while the strategy only uses liquid stocks it does not account for transaction costs, such as trade commissions. Since the strategy has monthly portfolio rebalancing, real-world net returns would be lower if these costs are implemented. Future work could incorporate transaction costs to provide more accurate results and executable strategies. Still, for the project's main goal, the setup is adequate, showing that Machine Learning Models provide better Momentum-Based portfolios compared to pure-momentum approaches.

References

1. Jegadeesh, N., & Titman, S. (1993). *Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency*. The Journal of Finance, 48(1), 65-91.
2. Daniel, K., & Moskowitz, T. J. (2016). *Momentum Crashes*. Journal of Financial Economics, 122(2), 221-247.
3. Barroso, P., & Santa-Clara, P. (2015). *Momentum Has Its Moments*. Journal of Financial Economics, 116(1), 111-120.
4. Gu, S., Kelly, B., & Xiu, D. (2020). *Empirical Asset Pricing via Machine Learning*. The Review of Financial Studies, 33(5), 2223-2273.
5. Moskowitz, T. J., Ooi, Y. H., & Pedersen, L. H. (2012). *Time Series Momentum*. Journal of Financial Economics, 104(2), 228-250.
6. Google. (2026). *Gemini Code Assist* - Software extension.
7. Francisco Santos. *DSAP_Project*. 2026. GitHub Repository. Available at: `https://github.com/franciscomldsantos/DSAP_Project`

A AI Tools Usage

According to the project guidelines, this section details the use of AI tools in its development.

- Gemini (Gemini 3 model) to help format the document in Latex format, structure the analysis in a logical way and correct orthographic errors.
- Gemini Code Assist within VSCODE for error debugging, explaining hyperparameter, complex concepts and for plotting graphs.

AI tools were used for LaTeX formatting, code debugging, and minor language corrections. All data processing, modeling decisions, backtesting, and result interpretation were implemented and verified by the author.