Enhancing Machine Learning-Driven Portfolio Construction through Advanced Data Utilization and Methodologies

1. Executive Summary

This report provides a comprehensive analysis of three S&P 500 datasets (30-minute, daily, and weekly frequencies) and offers detailed recommendations for improving a machine learning (ML) based portfolio construction Jupyter notebook. The core objective is to enhance the notebook's predictive capabilities and portfolio optimization strategies by effectively integrating these multi-frequency datasets and adopting advanced financial machine learning techniques.

Key findings from the data analysis reveal a rich source of pre-calculated technical indicators across different timeframes, presenting significant opportunities for sophisticated feature engineering. The 30-minute data (SPX_30.csv) [1], daily data (SPX_1D.csv) [1], and weekly data (SPX_1W.csv) [1] offer a multi-scale view of market dynamics, which, if properly integrated, can lead to more robust and nuanced ML models.

The primary recommendations focus on several areas:

1. **Advanced Feature Engineering:** Leveraging the multi-timeframe nature of the provided data by creating aggregated features from higher frequency data (e.g., intraday volatility for daily models) and contextual features from lower frequency data (e.g., weekly trend regimes for daily models). This also includes guidance on interpreting and utilizing complex pre-calculated indicators like Gaussian Moving Averages, MA Crossovers, RSI, Bollinger Bands, and Divergence signals present in the datasets.
2. **Sophisticated Model Considerations:** Exploring the use of time-series specific models such as LSTMs, GARCH (for volatility forecasting), and Transformer networks for their ability to capture complex temporal dependencies. An introduction to Reinforcement Learning (RL) for direct policy optimization is also provided.
3. **Robust Backtesting and Validation:** Implementing rigorous backtesting methodologies like Walk-Forward Optimization and assessing the statistical significance of results to ensure strategy robustness and avoid overfitting.
4. **Advanced Portfolio Construction Techniques:** Moving beyond basic Mean-Variance Optimization to explore methods like Hierarchical Risk Parity (HRP), Black-Litterman model (integrating ML predictions as views), and CVaR optimization for improved risk management and diversification, while also

considering the crucial impact of transaction costs.

The successful implementation of these recommendations is anticipated to significantly elevate the sophistication and potential efficacy of the user's ML portfolio construction notebook, leading to more informed and potentially more profitable investment strategies.

2. Analysis of Provided S&P 500 Datasets (SPX_30.csv, SPX_1W.csv, SPX_1D.csv)

A thorough understanding of the provided datasets is paramount for their effective integration into any machine learning portfolio construction framework. This section details the structure, granularity, time coverage, and key technical indicators present in the SPX_30.csv, SPX_1D.csv, and SPX_1W.csv files.

2.1. Data Structure, Granularity, and Time Coverage

Each CSV file represents S&P 500 index data at different time granularities: 30-minute (SPX_30.csv) [1], daily (SPX_1D.csv) [1], and weekly (SPX_1W.csv).[1] All files share a common set of core data points: 'time' (Unix timestamp), 'open', 'high', 'low', 'close', and 'Volume'. Additionally, they contain a variety of pre-calculated technical indicators.

The 'time' column, representing Unix timestamps, has been converted to Coordinated Universal Time (UTC) to establish the precise historical range of each dataset. For example, the weekly data in SPX_1W.csv spans from June 1, 1963, to May 25, 2025 [1], offering a very long-term perspective. The daily data in SPX_1D.csv covers June 16, 2010, to May 29, 2025.[1] The 30-minute data in SPX_30.csv starts much later, from July 25, 2024, and extends to May 29, 2025, based on the provided snippets.[1] It's important to note that the end dates extending into 2025 suggest these datasets might include some forecasted or simulated data, or the export date was in the future relative to the last actual data point. For ML model training, using data up to the present (or the last available historical point) is standard practice.

The following table summarizes the key characteristics of each file based on the provided snippets:

Table 1: Summary of CSV File Characteristics

File Name	Timeframe	Start Time (UTC Example)	End Time (UTC Example)	Key Indicator Families Present (Examples)
SPX_30.csv	30-minute	July 25, 2024, 14:00:00	May 29, 2025, 16:00:00	Gaussian MAs, Short/Long MAs, Cross, RSI, RSI MA, Bollinger Bands, Regular Bullish/Bearish Divergence [1]
SPX_1D.csv	1-Day	June 16, 2010, 14:50:00	May 29, 2025, 14:50:00	Gaussian MAs, Short/Long MAs, Cross, RSI, RSI MA, Bollinger Bands, Regular Bullish/Bearish Divergence [1]
SPX_1W.csv	1-Week	June 1, 1963, 00:30:00	May 25, 2025, 00:30:00	Gaussian MAs, Short/Long MAs, Cross, RSI, RSI MA, Bollinger Bands, Regular Bullish/Bearish Divergence [1]

Note: The exact start/end times and indicator presence might vary across the full datasets; the table reflects information from the provided snippets.

The availability of data across these distinct intervals—intraday (30-minute), daily, and weekly—offers a powerful foundation for multi-scale market analysis. Shorter timeframes, such as the 30-minute data, are adept at capturing intraday volatility dynamics, short-term momentum shifts, and the immediate impact of news or events. Daily data provides a balanced view, suitable for identifying medium-term trends, swing trading opportunities, and making tactical asset allocation adjustments. The weekly data, with its extensive history, offers a strategic lens on long-term market phases, economic cycles, and overarching trend directions.

The varying historical depth of these datasets is a critical consideration. The weekly data extends significantly further back than the daily or 30-minute data. This

discrepancy necessitates careful thought during feature engineering and model training, particularly when combining features from different timeframes. One approach is to utilize the common overlapping period for all features, ensuring a consistent dataset length. Alternatively, one could use the full history available for each timeframe-specific feature. This latter approach, while potentially capturing longer-term dependencies, introduces complexity in handling features of different lengths, possibly requiring techniques like padding, truncation, or specialized model architectures that can accommodate variable-length input sequences. The choice will influence model design and its capacity to learn from historical patterns of varying durations.

2.2. Interpretation of Key Technical Indicator Columns for Feature Engineering

The provided CSV files contain numerous pre-calculated technical indicators. Understanding their meaning and typical interpretation is crucial for effective feature engineering.

Gaussian Moving Averages (GMA):
The columns labeled 'Gaussian Moving Average' (appearing three times in each file, e.g.1) represent Gaussian-weighted moving averages. Unlike Simple Moving Averages (SMAs) that assign equal weight to all data points in their lookback period, GMAs assign higher weights to more recent data points, with weights decreasing according to a Gaussian (bell-shaped) curve.2 This often results in a smoother moving average that is more responsive to significant price movements while filtering out minor fluctuations.2 The presence of three distinct GMA columns strongly suggests they are calculated with different parameters, likely varying lengths or sigma values (the standard deviation of the Gaussian curve, which controls the smoothness). These can be directly used as separate features, allowing a machine learning model to discern which trend speed or degree of smoothing is most predictive under different market conditions. If the notebook were to calculate its own moving averages, these pre-calculated GMAs could serve as valuable benchmarks or as components in constructing more complex features, such as the spread between different GMAs or the price's deviation from each GMA.
Simple Moving Averages (Short MA, Long MA) and Crossover ('Cross') Signals:
The 'Short MA' and 'Long MA' columns represent standard Simple Moving Averages with different lookback periods.1 The 'Cross' column is particularly interesting. TradingView's MA Cross indicator typically highlights the bar where a shorter-term MA crosses a longer-term MA.4 The data in SPX_30.csv shows that the 'Cross' column contains numerical values that align with price levels (e.g., 5469.210999999983 at timestamp 1722261600 1). This suggests that the 'Cross' column likely records the price level of the Short MA (or perhaps the closing price) at the exact bar where the crossover event occurs. A non-NaN value in this column signifies a crossover event on that bar.
This 'Cross' column provides a direct numerical value associated with a trading signal. It can

be engineered into features such as "price relative to crossover level" (i.e., (close - Cross) / Cross) or used to define market regimes (e.g., a "post-golden cross" regime if the short MA crosses above the long MA). The absence of a value (NaN) indicates no crossover occurred, which is also informative. Models must be designed to handle these NaNs, perhaps through imputation (e.g., forward-filling the last cross value until a new cross occurs if the feature is intended to be "price since last cross") or by using an indicator variable for crossover events.
Relative Strength Index (RSI) and RSI-based Moving Averages:
The 'RSI' column provides the standard Relative Strength Index value, a momentum oscillator measuring the speed and change of price movements, commonly used to identify overbought (>70) or oversold (<30) conditions.1 The 'RSI-based MA' column is a moving average applied directly to the RSI values, offering a smoothed momentum signal. The difference between the raw RSI and its MA can itself be a valuable feature, akin to a MACD for momentum, signaling acceleration or deceleration in momentum strength. Furthermore, features can be engineered based on RSI levels (e.g., binary flags for RSI > 70 or RSI < 30) or its slope.
Bollinger Bands:
The 'Upper Bollinger Band' and 'Lower Bollinger Band' columns define a dynamic envelope around price.1 These bands are typically set K standard deviations above and below a central moving average (often an SMA, which might correspond to one of the existing MA columns or need to be calculated if not explicitly provided). Features such as Bollinger Bandwidth ((UpperBB - LowerBB) / MiddleBB) can quantify market volatility in a normalized way. Another powerful feature is %B ((Price - LowerBB) / (UpperBB - LowerBB)), which indicates the price's position relative to the bands, normalized between 0 and 1.
Regular Bullish/Bearish Divergence Signals and Labels:
Columns like 'Regular Bullish', 'Regular Bullish Label', 'Regular Bearish', and 'Regular Bearish Label' denote pre-identified divergence patterns.1 Regular divergence occurs when price makes a new extreme (lower low in a downtrend for bullish divergence, or higher high in an uptrend for bearish divergence) that is not confirmed by a corresponding new extreme in an oscillator (e.g., RSI makes a higher low for bullish divergence, or a lower high for bearish divergence).5 This signals a potential weakening of the current trend and a possible reversal. In the SPX_30.csv snippet 1, the 'Regular Bullish' and 'Regular Bearish' columns contain numerical values when a divergence is detected (e.g., 51.8757... for 'Regular Bullish' at timestamp 1721923200). This numerical value likely represents the value of the underlying oscillator (e.g., RSI) that formed the divergence. A higher RSI value during a bullish divergence (where price makes lower lows but RSI makes higher lows) could indicate stronger underlying buying pressure despite the price decline. The 'Label' columns are mostly NaN in the provided snippets; if they contain data, it might be the price level at which the divergence was confirmed or another related piece of information. These divergence signals are event-based features. The presence of a non-NaN value is a signal in itself. The magnitude of the value in the primary 'Regular Bullish/Bearish' columns can also be used as a feature, potentially indicating the strength of the divergence. NaNs (no divergence) should be handled appropriately, perhaps by imputing zero or using a binary indicator flag.

3. Strategic Enhancements for the ML Portfolio Construction

Notebook

To elevate the existing ML portfolio construction notebook, several strategic enhancements can be implemented, focusing on advanced feature engineering, sophisticated model considerations, robust backtesting, and improved model interpretability.

3.1. Advanced Feature Engineering Strategies

Effective feature engineering is often the most critical step in developing successful financial machine learning models.[6]

3.1.1. Leveraging Multi-Timeframe Data from Provided CSVs

The availability of S&P 500 data at 30-minute, daily, and weekly intervals [1] presents a significant opportunity to create a richer feature set. By combining information across these scales, models can capture a more holistic view of market dynamics.

- **Methodology for Combining Timeframes:**
  - **Upsampling Lower Frequency Data:** Weekly indicators (e.g., the direction of a weekly Gaussian Moving Average or whether the weekly RSI is above 50) can be forward-filled to apply to each day of the subsequent week. This allows daily models to incorporate longer-term trend context or regime information.[9] For instance, a daily trading signal might be filtered based on whether the weekly trend is aligned.
  - **Downsampling Higher Frequency Data:** Intraday (30-minute) data can be aggregated to create daily features. Examples include:
    - *Volatility Measures:* Calculating the standard deviation of 30-minute log returns over a trading day to represent daily realized volatility.
    - *Volume Profile:* Summing 30-minute volume to get total daily volume, or analyzing intraday volume patterns.
    - *Indicator Aggregation:* Averaging the 30-minute RSI values over a day, or counting the number of bullish/bearish 30-minute MA crossovers that occurred within a single trading day.
  - These aggregated or disaggregated features, when aligned to a common frequency (e.g., daily), provide the ML model with diverse signals. Features from longer timeframes can establish the broader market context (e.g., prevailing trend, volatility regime), while features from shorter timeframes can offer more granular information for timing entries, exits, or assessing short-term risk.[10] This hierarchical approach often leads to more robust and adaptive models.
  - It is crucial to ensure that the process of combining data from different

timeframes does not introduce lookahead bias. For example, when creating a daily feature from weekly data (like a weekly moving average), the value used for any given day should be based on weekly data available *up to the end of the preceding week*, or on indicators that are only finalized at the week's close. Similarly, when aggregating 30-minute data for a daily signal, all 30-minute data points used must strictly precede or be contemporaneous with the daily bar to which the feature is being assigned.

3.1.2. Techniques for Handling Initial NaN Values in Indicator Series

Technical indicators, by their nature (requiring a lookback period), will have NaN (Not a Number) values at the beginning of their time series. Properly addressing these initial NaNs is essential for model training.

- **Truncation:** The most straightforward method is to remove the initial rows where any feature contains a NaN value. This is often acceptable if the NaN period is short relative to the total dataset length and if the dataset is large enough to absorb the loss of initial data points.
- **Backward Fill (bfill):** This method fills NaN values with the first valid calculated value that appears later in the series. For initial NaNs in technical indicators, this means propagating the first non-NaN indicator value backward to fill the initial missing period.[11] This approach assumes that the indicator's value during its uncalculable initial phase was similar to its first observable value. This can be a reasonable approach if the indicator is expected to be somewhat stable initially or if preserving the length of the dataset is critical.
- **Zero or Arbitrary Constant Imputation:** Replacing NaNs with zero or another arbitrary constant (e.g., -1, 999) is generally discouraged for financial indicators unless zero is a meaningful baseline for that specific indicator.[11] Such imputation can introduce artificial levels or patterns that might mislead the ML model.
- **Model-Based Imputation:** More sophisticated methods, such as using an autoregressive model or Expectation-Maximization (EM) algorithms, can be employed to estimate missing values.[12] However, for initial NaNs caused by lookback periods, these might be overly complex unless the NaN period is very long or affects many crucial features.
- **Considerations for Financial Data:** The impact of NaN handling is more pronounced for shorter datasets or for indicators with very long lookback periods. The chosen method should be validated during backtesting to ensure it doesn't introduce biases or degrade model performance. Some machine learning models (e.g., certain tree-based algorithms like XGBoost) can handle NaN values natively or through specific encoding strategies, which might be preferable to imputation

if the "missingness" itself could be an informative signal, though this is less likely for initial NaNs due to indicator calculation windows.

3.1.3. Potential for Advanced Time-Series Features

Beyond standard technical indicators, more advanced feature engineering techniques can uncover deeper patterns in financial time series:

- **Fractional Differentiation:** Financial time series are often non-stationary. While standard integer differencing (e.g., taking first differences of prices to get returns) can achieve stationarity, it can also remove valuable long-term memory from the series. Fractional differentiation offers a method to achieve stationarity while preserving as much of this memory as possible by differencing by a positive real number d (where $0<d<1$).[13] Libraries like mlfinpy (an open-source alternative inspired by MlFinLab) provide implementations for calculating fractionally differentiated series.[18] The optimal d value is typically found by identifying the minimum differentiation that makes the series stationary (e.g., via an ADF test) while maximizing correlation with the original series.[14]
- **Wavelet Transforms:** Wavelet analysis can decompose a time series into different time-frequency components. This allows for the identification of localized patterns, transient events, and the separation of signal from noise across multiple resolutions.[22] Features derived from wavelet coefficients (e.g., energy at different scales) can capture market dynamics that are not apparent in the time domain alone. Python libraries like PyWavelets can be used for this purpose.

These advanced techniques can potentially improve model performance by providing richer, more informative features. However, they also introduce greater complexity and require careful parameter tuning. It is generally advisable to start with simpler, well-understood features and incrementally add complexity, validating the impact of new features through rigorous backtesting.

3.2. Sophisticated Model Considerations

The choice of machine learning model is pivotal. While simpler models have their place, financial time series often exhibit complex, non-linear dynamics that may be better captured by more sophisticated architectures.

3.2.1. Evaluating Suitability of Time-Series Specific Models

- **Recurrent Neural Networks (LSTMs/GRUs):** If the current notebook employs simpler models (e.g., linear regression, basic tree models), Long Short-Term

Memory (LSTM) networks and Gated Recurrent Units (GRUs) should be considered. These are types of Recurrent Neural Networks (RNNs) specifically designed to capture sequential dependencies and long-range patterns in time series data, which is characteristic of financial markets.[23] Python libraries like TensorFlow/Keras and PyTorch facilitate their implementation.

- **GARCH Models for Volatility Forecasting:** Generalized Autoregressive Conditional Heteroskedasticity (GARCH) models are the standard for forecasting volatility in financial time series.[26] Volatility forecasts are crucial inputs for risk management (e.g., Value-at-Risk, CVaR calculations) and for certain portfolio optimization strategies like risk parity. The arch library in Python is a popular choice for implementing GARCH and its variants (e.g., EGARCH, GJR-GARCH).[28] The outputs of GARCH models (e.g., predicted conditional variance) can also serve as valuable input features for other machine learning models that predict returns or make allocation decisions.
- **Transformer Models:** Originally developed for Natural Language Processing (NLP), Transformer models have shown significant promise for time series forecasting, including financial applications.[30] Their key advantage is the self-attention mechanism, which allows them to weigh the importance of different past observations and capture very long-range dependencies more effectively than traditional RNNs.[30] Python libraries like Hugging Face Transformers (transformers) and PyTorch Forecasting (pytorch-forecasting, which includes implementations like the Temporal Fusion Transformer) can be used.[33] However, Transformers typically require large datasets for effective training, can be computationally intensive, and their "black-box" nature can make interpretation challenging.[30]
- The choice of model should align with the specific forecasting task (e.g., point prediction of returns, probability of an upward move, volatility estimation, regime identification) and the characteristics of the data. It's often beneficial to experiment with multiple model architectures. Furthermore, integrating predictions or outputs from different model types can create powerful hybrid systems. For example, a GARCH model might forecast daily volatility, which then becomes an input feature to an LSTM or Transformer model tasked with predicting daily returns.

3.2.2. Introduction to Reinforcement Learning (RL) Approaches

Reinforcement Learning offers a distinct paradigm for portfolio construction. Instead of predicting returns or risk factors which are then fed into a separate optimization model, RL agents learn an optimal *policy* (a mapping from market states to actions, such as asset allocations) directly by interacting with a simulated market environment

and receiving rewards or penalties based on their decisions.[25]

- **Common RL Algorithms:** Algorithms like Deep Q-Networks (DQN), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC) have been applied to financial trading and portfolio management.[38]
- **Python Libraries for Financial RL:**
  - **FinRL:** An open-source library specifically designed for quantitative finance, providing a framework for market environments, DRL agents, and applications like stock trading and portfolio allocation.[42] It supports integration with popular RL libraries like Stable Baselines3.
  - **TensorTrade:** Another open-source Python framework for building, training, and deploying trading algorithms using reinforcement learning. It emphasizes modularity and composability.[52]
- RL directly addresses the sequential decision-making nature of portfolio management. The agent learns to optimize a cumulative reward function (e.g., Sharpe ratio, total portfolio return, or a utility function incorporating risk aversion) over time. This can bypass the often difficult step of explicitly forecasting returns, which is a major challenge in traditional quantitative finance.
- However, implementing RL strategies is generally more complex than supervised learning approaches. It requires careful design of the state space (how the market environment is represented to the agent), action space (what actions the agent can take, e.g., reallocating weights), and the reward function (which guides the agent's learning). Backtesting RL agents also necessitates specialized considerations to ensure that the simulated environment accurately reflects real-world market dynamics, including transaction costs and market impact.

3.3. Robust Backtesting and Performance Validation

A robust backtesting framework is essential to assess the viability of any trading strategy and to avoid common pitfalls like overfitting and lookahead bias.[61]

3.3.1. Implementing Time-Series Cross-Validation and Walk-Forward Optimization

Standard k-fold cross-validation is often inappropriate for time series data because it shuffles data, thereby destroying temporal dependencies and potentially allowing future information to leak into the training set for past predictions.

- **Time-Series Cross-Validation (Expanding or Rolling Window):**
  - **Expanding Window:** The training set grows progressively, always starting from the beginning of the dataset, while the test set is a fixed window following the training set.

    - **Rolling Window (Walk-Forward Optimization - WFO):** Both the training and test sets are fixed-size windows that slide forward through time.[62] The model is retrained on each new training window and tested on the subsequent out-of-sample window.
- **Walk-Forward Optimization (WFO):** WFO is a particularly robust method for financial strategy backtesting.[62] It simulates how a strategy would realistically be deployed and periodically re-calibrated over time. By cycling through multiple optimization-validation periods, WFO provides a more reliable estimate of live performance and helps to assess the strategy's adaptability to changing market conditions.[62] For example, one might train the model on data from 2010-2015 and test on 2016, then roll forward to train on 2011-2016 and test on 2017, and so on.
- The choice of window lengths for in-sample training and out-of-sample testing in WFO is a critical hyperparameter. These lengths can significantly affect the backtesting results and should ideally be chosen based on the assumed stationarity of market regimes or tested for sensitivity. WFO helps mitigate overfitting by repeatedly testing the strategy on unseen data segments.[62]

3.3.2. Assessing Statistical Significance of Backtesting Results

A high Sharpe ratio or impressive cumulative return in a single backtest might be the result of luck or overfitting rather than a genuine strategic edge. Statistical significance testing helps quantify the confidence in a strategy's performance.

- **Bootstrapping:** This technique involves resampling from the historical returns (with replacement) to create many synthetic return series. The strategy's performance metric (e.g., Sharpe ratio) is calculated for each synthetic series, generating a distribution of the metric. This distribution can be used to estimate confidence intervals and p-values for the observed Sharpe ratio.[64] The arch library in Python provides tools for stationary bootstrapping, which is appropriate for time series data.[64]
- **Ledoit & Wolf Test for Comparing Sharpe Ratios:** While primarily for comparing the Sharpe ratios of two different strategies, the underlying statistical principles can inform about the uncertainty of a single Sharpe ratio.[65] Their work highlights the importance of accounting for non-normal and serially correlated returns when assessing Sharpe ratio statistics.
- **Addressing Common Pitfalls:** It's crucial to be aware of common backtesting pitfalls such as data snooping bias (testing too many variations on the same data), overfitting, unrealistic assumptions about transaction costs or liquidity, and ignoring changing market conditions.[61] While statistical significance tests are valuable, they should be complemented by a comprehensive out-of-sample

validation regime like WFO and sensitivity analysis. Libraries like QuantStats offer a wide array of performance metrics [67], but direct p-value calculation for Sharpe ratios might require custom implementations or specialized functions from other libraries like arch.

3.4. Enhancing Model Interpretability with XAI Techniques

For complex machine learning models, particularly in finance where decisions have significant monetary consequences, understanding *why* a model makes certain predictions or allocation choices is crucial. Explainable AI (XAI) techniques provide this insight.

- **SHAP (SHapley Additive exPlanations):** SHAP is a game theory-based approach that assigns an importance value (SHAP value) to each feature for a particular prediction, indicating its contribution to the model's output relative to a baseline.[69] SHAP values can provide both local explanations (for individual predictions) and global feature importance (by aggregating SHAP values across many predictions).
- **LIME (Local Interpretable Model-agnostic Explanations):** LIME explains the predictions of any classifier or regressor by approximating it locally with an interpretable model (e.g., a linear model or a decision tree).[69]
- **Application in Portfolio Construction:** In the context of portfolio construction, XAI can help identify which input features (e.g., specific technical indicators, macroeconomic variables) are most influential in the model's forecasts of returns, risk, or its direct allocation decisions. This is vital for building trust in the model, debugging unexpected behavior, and ensuring the model has learned sensible financial relationships rather than spurious correlations.
- For instance, if a model consistently assigns high importance to a particular moving average crossover when predicting a market upturn, this provides a degree of validation. Conversely, if seemingly irrelevant features show high importance, it may indicate overfitting or data leakage. XAI can also reveal how feature importance changes across different market regimes, offering deeper insights into the model's adaptability. Python libraries for SHAP and LIME are readily available (e.g., shap, lime).

4. Integrating and Utilizing TradingView CSV Data

The provided CSV files, exported from TradingView, contain a wealth of information that can be systematically processed and engineered into features for the machine learning portfolio construction notebook.

4.1. Recommended Data Loading and Preprocessing Pipeline

A robust data pipeline is the first step towards effectively utilizing the provided CSVs.

1. **Loading Data:** Use the pandas library to load each CSV file (SPX_30.csv, SPX_1D.csv, SPX_1W.csv) into separate DataFrames.
   Python
   ```
   import pandas as pd
   # Example for one file
   # df_30m = pd.read_csv('SPX_30.csv')
   ```

2. **Time Conversion and Indexing:** The 'time' column contains Unix timestamps. Convert this column to datetime objects and set it as the DataFrame index. This facilitates time-series operations.
   Python
   ```
   # df_30m['time'] = pd.to_datetime(df_30m['time'], unit='s', utc=True)
   # df_30m = df_30m.set_index('time')
   ```

3. **Handling Duplicate Column Names:** The CSV snippets show multiple columns named "Gaussian Moving Average" and potentially "RSI".[1] These likely represent the same indicator with different parameter settings. It's crucial to rename these columns to be unique for the ML model to treat them as distinct features. A systematic approach is to append a suffix (e.g., _1, _2, _3 or, if parameters are known/inferred, like _short, _medium, _long).
   Python
   ```
   # Example for GMA columns if three are present
   # gma_cols = [col for col in df_30m.columns if 'Gaussian Moving Average' in col]
   # rename_map = {old_col: f'GMA_{i+1}' for i, old_col in enumerate(gma_cols)}
   # df_30m = df_30m.rename(columns=rename_map)
   ```

4. **Data Cleaning - Missing Values:**
   - **Initial NaNs:** As discussed in Section 3.1.2, technical indicators will have NaNs at the beginning. Decide on a strategy: truncation (simplest if data is ample) or backward fill (bfill) if the initial period needs to be preserved and the first valid value is a reasonable proxy.
     Python
     ```
     # Option 1: Truncate rows with any NaN (often done after all feature engineering)
     # df_processed = df_features.dropna()
     # Option 2: Backward fill specific indicator columns if NaNs are only at the start
     # for col in indicator_columns_with_initial_nans:
     #     df_30m[col] = df_30m[col].bfill()
     ```

   - **Other NaNs:** Inspect data for other sporadic NaNs. The PlotCandle (...)

columns, for instance, often contain many NaNs [1] and might be less useful or require careful handling if used. Columns like 'Cross', 'Regular Bullish', 'Regular Bearish', and their 'Label' counterparts will naturally have many NaNs, as these represent specific events rather than continuous series. For these, NaNs mean "event did not occur."

5. **Data Type Verification:** Ensure all numerical columns are of a float or integer type as appropriate.
6. **Alignment for Multi-Timeframe Analysis:** If features from different timeframes are to be combined, resample and align them to a common frequency (e.g., daily) as detailed in Section 4.3.

4.2. Practical Feature Engineering from CSV Indicators

The pre-calculated indicators in the CSVs can be used directly or transformed into more potent features. The goal is to extract signals that are more robust, normalized, or capture interactions.

Table 2: Example Engineered Features from TradingView CSV Data

Original CSV Column(s)	Engineered Feature Name	Calculation/Logic Example	Rationale/Potential Signal
close, GMA_1 (renamed Gaussian Moving Average)	Price_Dev_GMA1	(close - GMA_1) / GMA_1	Normalized deviation from the first Gaussian Moving Average; measures how far price is from its trend.
Short MA, Long MA	MA_Spread_Normalized	(Short MA - Long MA) / Long MA	Normalized difference between short and long MAs; indicates trend strength and direction.
RSI, RSI-based MA	RSI_Momentum_Osc	RSI - RSI-based MA	Oscillator for RSI itself; signals accelerating/decelerating momentum.
Upper Bollinger Band, Lower Bollinger	BB_Width_Norm	(Upper Bollinger Band - Lower	Bollinger Bandwidth normalized by price;

Band, close		Bollinger Band) / close	a measure of relative volatility.
close, Upper Bollinger Band, Lower Bollinger Band	Percent_B	(close - Lower Bollinger Band) / (Upper Bollinger Band - Lower Bollinger Band)	Price position relative to Bollinger Bands (0-1 scale); indicates overextension or proximity to band edges.
Regular Bullish (oscillator value at divergence)	Bull_Divergence_Active	Binary: 1 if Regular Bullish is not NaN, 0 otherwise.	Flag indicating the presence of a bullish divergence signal. The actual oscillator value can also be used if scaled.
Regular Bearish (oscillator value at divergence)	Bear_Divergence_Strength	Regular Bearish (if not NaN, else 0 or other imputation). Potentially scale/normalize.	Magnitude of the oscillator during a bearish divergence; may indicate signal strength.
Cross (price level of MA crossover), close	Price_Dist_From_Cross	(close - Cross.ffill()) / Cross.ffill() (forward-fill Cross value until next cross)	Normalized distance of current price from the level of the last MA crossover; measures momentum since the last signal.
Volume	Volume_MA_Ratio	Volume / Volume.rolling(window=20).mean()	Volume relative to its moving average; identifies volume spikes or unusual activity.
high, low, close (from 30-min data, aggregated daily)	Daily_Realized_Vol_30m	np.std(log_returns_30m_for_the_day)	Daily volatility calculated from intraday (30-minute) price movements.
RSI (from weekly data, forward-filled to	Weekly_RSI_Regime	Binary: 1 if Weekly_RSI_ffill > 50, 0 if Weekly_RSI_ffill <	Daily feature indicating the prevailing weekly

daily)		50, else 0.5 (neutral)	momentum regime.

These engineered features aim to provide the ML model with more refined signals. For instance, normalizing price deviations or bandwidths makes features comparable across different price levels and time periods. Event-based features like Bull_Divergence_Active transform sporadic indicator signals into a continuous input the model can learn from. The interaction between different indicators can also yield powerful features. For example, a feature could flag instances where an RSI divergence occurs while the price is simultaneously touching an outer Bollinger Band and a key moving average is sloping in the direction of the divergence. Such multi-condition features, though potentially rarer, can represent high-conviction trading setups.

4.3. Strategies for Incorporating Multi-Frequency Data into the Existing ML Workflow

If the existing Jupyter notebook primarily operates on a single frequency (e.g., daily predictions), the data from SPX_30.csv (30-minute) and SPX_1W.csv (weekly) needs to be transformed and aligned to this primary frequency.

Assuming a Daily Model Frequency:

- **Utilizing SPX_1D.csv (Daily Data):** Features derived from this file are native to the model's frequency. This includes daily OHLCV, and daily calculations of the provided indicators (GMAs, MAs, RSI, Bollinger Bands, Divergences).
- **Integrating SPX_30.csv (30-Minute Data):** Aggregate intraday data to create daily features. This allows the daily model to benefit from intraday dynamics.
    - *Volatility:* Calculate daily realized volatility using the standard deviation of 30-minute log returns within each trading day. High intraday volatility can signal increased risk or potential breakouts/breakdowns.
    - *Volume:* Sum 30-minute volume to get total daily volume. Analyze intraday volume profiles (e.g., volume during the first/last hour) as separate daily features if relevant.
    - *Indicator Aggregation:*
        - Calculate the average, min, or max 30-minute RSI value over the day.
        - Count the number of bullish or bearish 30-minute MA crossovers that occurred within a single day.
        - Flag days where a 30-minute divergence signal was observed.
    - *Price Range:* Calculate the daily high-low range based on 30-minute data, which might be more precise than the daily OHLC if the daily snapshot is at a

fixed time.

- **Integrating SPX_1W.csv (Weekly Data):** Disaggregate or carry forward weekly signals to provide longer-term context to the daily model.
  - *Trend Context:* Determine the direction of a key weekly moving average (e.g., is the weekly close above its 20-week GMA?). Forward-fill this binary or categorical feature to all trading days within that week.
  - *Momentum Regime:* Use the weekly RSI level (e.g., RSI > 55 for bullish regime, RSI < 45 for bearish regime) and forward-fill it as a daily feature.
  - *Volatility Context:* The weekly Average True Range (ATR) or Bollinger Bandwidth can be forward-filled to indicate the prevailing weekly volatility environment.

Alignment and Merging:
After features are created at the desired common frequency (e.g., daily), they need to be merged into a single DataFrame. This is typically done by aligning the datetime indices.

Python

```
# Pseudo-code example for daily model
# df_daily_features = process_spx_1d(spx_1d_raw)
# df_30m_agg_daily = aggregate_30m_to_daily(spx_30m_raw)
# df_1w_disagg_daily = disaggregate_1w_to_daily(spx_1w_raw)

# final_features = df_daily_features.join(df_30m_agg_daily, how='left')
# final_features = final_features.join(df_1w_disagg_daily, how='left')
# final_features = final_features.dropna() # Or handle NaNs from joins/aggregation
```

The choice of aggregation (e.g., mean, sum, std dev) or disaggregation (e.g., forward fill) method depends heavily on the nature of the indicator. For instance, forward-filling a weekly trend signal is logical as a trend is assumed to persist through the week. However, averaging weekly price changes to derive a daily price change would be inappropriate. Careful consideration must be given to avoid introducing lookahead bias, especially when using data from a lower frequency (like weekly) in a higher frequency model (like daily). The weekly feature for a given day should only use information that was available at the close of the *previous* week or be based on indicators that are only fixed at the week's end.

5. Advanced Portfolio Construction Methodologies

While the Jupyter notebook's current portfolio construction method is not detailed,

moving beyond basic Mean-Variance Optimization (MVO) can often lead to more robust and practical portfolio allocations, especially when dealing with the complexities and estimation errors inherent in financial markets.

5.1. Review of Modern Alternatives to Basic MVO

Several advanced techniques address the limitations of traditional MVO, such as its sensitivity to input errors (expected returns and covariances) and tendency to produce concentrated portfolios.

- Hierarchical Risk Parity (HRP):
  Developed by Marcos Lopez de Prado, HRP is a portfolio allocation method that utilizes hierarchical clustering to group assets based on their correlation structure.73 It then applies risk parity principles within and across these clusters. A key advantage is that HRP does not require the inversion of the covariance matrix, making it more robust to estimation errors in this matrix, especially when the number of assets is large relative to the length of the time series.74 It tends to produce more diversified portfolios.
  - **Python Libraries:** PyPortfolioOpt offers the HRPOpt class [74], and skfolio also provides HRP implementations.[75]
- Black-Litterman Model:
  The Black-Litterman model provides a Bayesian framework for combining an investor's subjective views on expected returns with market equilibrium returns (the prior) to generate a blended, posterior set of expected returns.77 These posterior returns can then be used in a standard MVO framework.
  - **Integration with ML:** Machine learning model predictions (e.g., forecasted returns for specific assets or sectors) can be naturally incorporated as the "investor views" (Q) in the Black-Litterman model.[77] The confidence in these ML predictions can inform the uncertainty matrix (Ω). This provides a structured way to blend data-driven forecasts with market-implied priors, potentially leading to more stable and intuitive allocations.
  - **Python Libraries:** PyPortfolioOpt has a BlackLittermanModel class.[77] skfolio [75] and Riskfolio-Lib [80] also support Black-Litterman and its variants.
- CVaR (Conditional Value-at-Risk) Optimization:
  CVaR, also known as Expected Shortfall, measures the expected loss given that the loss exceeds the Value-at-Risk (VaR) at a certain confidence level. Optimizing portfolios based on minimizing CVaR (subject to a minimum expected return) or maximizing return (subject to a CVaR constraint) focuses on managing tail risk, which is often overlooked by variance-based measures.81
  - **Integration with ML:** If the ML model can generate probabilistic forecasts of

returns or a distribution of potential future asset price scenarios (e.g., via Monte Carlo simulation based on ML-predicted parameters), these can be used to estimate the CVaR of different portfolio allocations. This allows for direct optimization of tail risk based on the model's predictive distribution.
    - **Python Libraries:** PyPortfolioOpt includes an EfficientCVaR class.[84] skfolio offers DistributionallyRobustCVaR and allows CVaR as a risk measure in its MeanRisk optimizer.[75] Riskfolio-Lib supports CVaR and many other advanced risk measures across various optimization models.[80]

The choice of optimization method should align with the specific strengths of the ML model and the investor's risk preferences. If the ML model is particularly adept at identifying assets with lower tail risk or predicting scenarios leading to large losses, CVaR optimization becomes highly relevant. If the goal is robust diversification based on covariance structure, HRP is a strong candidate. If blending specific ML-driven return forecasts with market equilibrium is desired, Black-Litterman offers a formal framework.

5.2. Incorporating Transaction Costs and Market Impact

Realistic portfolio optimization must account for transaction costs, which include brokerage fees, bid-ask spreads, and market impact (the adverse price movement caused by the trade itself).[92] Ignoring these costs can lead to theoretically optimal portfolios that are unprofitable or infeasible in practice.

- **Impact on Portfolio Turnover:** Including transaction costs in the optimization objective (typically as a penalty term) or as constraints (e.g., turnover limits) naturally leads the optimizer to favor portfolios with lower turnover. This is because the expected benefit from rebalancing must outweigh the cost of trading.[92]
- **Modeling Costs:**
    - **Linear Costs:** Proportional costs (e.g., a percentage of trade value for commissions or bid-ask spread) are relatively straightforward to model.
    - **Non-Linear Costs:** Market impact is often non-linear, increasing more than proportionally with trade size.[94] Modeling this requires more sophisticated optimization techniques but leads to more realistic outcomes.
- **Python Libraries:**
    - Cvxportfolio: This library is specifically designed for portfolio optimization with a strong focus on realistic cost modeling, including various types of transaction and holding costs.[95] It allows users to define custom cost functions that can be incorporated into single-period or multi-period optimization problems.

- skfolio: Also includes features for incorporating transaction costs and management fees into the optimization process.[75]
- PyPortfolioOpt: While primarily focused on mean-variance and related models, it can be extended with custom objective terms or constraints to approximate transaction costs.

By explicitly accounting for trading frictions, the resulting portfolios are more likely to achieve their targeted risk-return profiles in live trading. This is particularly important for strategies that might otherwise generate frequent trading signals.

6. Detailed Recommendations and Implementation Roadmap

This section synthesizes the analyses and discussions into a prioritized and actionable roadmap for enhancing the ML portfolio construction notebook and effectively utilizing the provided TradingView S&P 500 datasets.

6.1. Prioritized List of Improvements for the Jupyter Notebook

The following improvements are suggested, ordered by a general progression from foundational data handling to more advanced modeling and optimization:

1. **Robust Data Preprocessing for CSVs:**
   - Implement systematic loading of all three CSV files (SPX_30.csv, SPX_1D.csv, SPX_1W.csv).
   - Convert 'time' columns to UTC datetime objects and set as index.
   - Rename duplicate indicator columns (e.g., 'Gaussian Moving Average', 'RSI') to ensure uniqueness (e.g., GMA_1, GMA_2, RSI_1).
   - Develop a clear strategy for handling initial NaN values in technical indicator series (e.g., truncation or backward fill, justified by analysis). Address other NaNs based on their meaning (e.g., for event columns like 'Cross' or 'Regular Bullish', NaN means no event).
2. **Enhanced Feature Engineering:**
   - Implement the creation of derived features from the raw indicator columns as outlined in Table 2 (e.g., Price_Dev_GMA1, MA_Spread_Normalized, BB_Width_Norm, Percent_B, event flags for divergences/crossovers).
   - Develop and integrate multi-timeframe features. For a daily model:
     - Aggregate 30-minute data (e.g., daily realized volatility from 30-min returns, daily sums/averages of 30-min volume/RSI).
     - Disaggregate/forward-fill weekly data (e.g., weekly trend direction, weekly RSI regime).
   - Consider experimenting with fractional differentiation for key price or

indicator series to achieve stationarity while preserving memory, using libraries like mlfinpy.

3. **Implement Robust Backtesting Framework:**
    - Transition from simple train-test splits to Time-Series Cross-Validation, specifically Walk-Forward Optimization (WFO), to get a more realistic assessment of out-of-sample performance.[62]
    - Incorporate statistical significance testing for key performance metrics like the Sharpe ratio (e.g., using bootstrapping with arch.bootstrap [64]) to assess if performance is due to skill rather than luck.
    - Explicitly account for estimated transaction costs and slippage in backtest performance calculations.[61]
4. **Refine or Diversify Model Selection:**
    - If using basic ML models, evaluate time-series specific architectures like LSTMs/GRUs or GARCH models (especially if volatility is a key prediction target or risk input).[23]
    - For more advanced exploration, consider Transformer models, being mindful of their data and computational requirements.[30]
    - Explore Explainable AI (XAI) techniques like SHAP or LIME to understand feature importance and model decision-making, regardless of the model used.[69]
5. **Explore Advanced Portfolio Optimization Techniques:**
    - If currently using basic MVO, experiment with alternatives like Hierarchical Risk Parity (HRP) for robust diversification [73] or CVaR optimization for tail risk management.[81]
    - Consider the Black-Litterman model to formally blend ML-generated return forecasts (as views) with market-implied priors.[77]
    - Integrate transaction cost models directly into the optimization objective or as constraints using libraries like Cvxportfolio or skfolio.[75]

6.2. Step-by-Step Guidance on Integrating and Using the CSV Data

The following provides a conceptual Python-based workflow for integrating the provided data:

Python

```
import pandas as pd
```

```
import numpy as np

# --- 1. Load and Initial Preprocess Data ---
def load_and_preprocess(file_path, timeframe_suffix):
    df = pd.read_csv(file_path)
    df['time'] = pd.to_datetime(df['time'], unit='s', utc=True)
    df = df.set_index('time')

    # Rename potentially duplicated columns (example for GMA)
    gma_cols = [col for col in df.columns if 'Gaussian Moving Average' in col]
    if len(gma_cols) > 1: # Check if there are indeed multiple GMA columns
        rename_map = {old_col: f'GMA_{i+1}_{timeframe_suffix}' for i, old_col in
enumerate(gma_cols)}
        df = df.rename(columns=rename_map)
    elif len(gma_cols) == 1: # Only one GMA column
         df = df.rename(columns={gma_cols: f'GMA_{timeframe_suffix}'})


    # Rename other potentially duplicated indicator columns similarly (RSI, etc.)
    # Example for RSI
    rsi_cols = # Avoid exact 'RSI' if it's unique
    if len(rsi_cols) > 1:
        rename_map_rsi = {old_col: f'RSI_{i+1}_{timeframe_suffix}' for i, old_col in
enumerate(rsi_cols)}
        df = df.rename(columns=rename_map_rsi)
    elif len(rsi_cols) == 1 and rsi_cols!= 'RSI': # If it's like 'RSI.1'
        df = df.rename(columns={rsi_cols: f'RSI_1_{timeframe_suffix}'})

    # Ensure standard columns like 'RSI', 'Short MA', etc., also get a suffix if not already unique
    standard_cols_to_suffix =
    rename_map_std = {col: f'{col}_{timeframe_suffix}' for col in standard_cols_to_suffix if col
in df.columns}
    df = df.rename(columns=rename_map_std)

    # Select only relevant columns (OHLCV + known indicators)
    # This step depends on the final list of columns after unique renaming
    # For now, let's assume all columns are potentially useful after renaming

    return df
```

```
df_30m = load_and_preprocess('SPX_30.csv', '30m')
df_1d = load_and_preprocess('SPX_1D.csv', '1d')
df_1w = load_and_preprocess('SPX_1W.csv', '1w')

# --- 2. Feature Engineering (Examples) ---
# Example: Daily Realized Volatility from 30-min data
if not df_30m.empty:
    df_30m[f'log_ret_30m'] = np.log(df_30m[f'close_30m'] / df_30m[f'close_30m'].shift(1))
    daily_vol_30m =
df_30m[f'log_ret_30m'].resample('D').std().rename('Daily_Realized_Vol_30m')
else:
    daily_vol_30m = pd.Series(name='Daily_Realized_Vol_30m')


# Example: Weekly Trend Regime forward-filled to daily
if not df_1w.empty and f'GMA_1_1w' in df_1w.columns and f'close_1w' in df_1w.columns: #
Check if columns exist
    df_1w = (df_1w[f'close_1w'] > df_1w[f'GMA_1_1w']).astype(int)
    # Align weekly data to daily (forward fill)
    # Ensure df_1d index is available for reindexing
    if not df_1d.empty:
        weekly_features_daily = df_1w].reindex(df_1d.index, method='ffill')
    else: # If df_1d is empty, create an empty series for weekly_features_daily
        weekly_features_daily = pd.DataFrame(index=df_1d.index, columns=)

else: # If necessary columns are not in df_1w or df_1w is empty
    weekly_features_daily = pd.DataFrame(index=df_1d.index, columns=)


# --- 3. Combine Features to Daily Frequency ---
# Start with daily data as the base
if not df_1d.empty:
    final_df = df_1d.copy()
    # Join aggregated 30-min features
    if not daily_vol_30m.empty:
        final_df = final_df.join(daily_vol_30m, how='left')
    # Join disaggregated weekly features
    if not weekly_features_daily.empty:
        final_df = final_df.join(weekly_features_daily, how='left')
```

```
else: # Handle case where df_1d might be empty or not loaded as expected
    # Attempt to create a base index from other dataframes if possible, or raise an error
    if not daily_vol_30m.empty:
        final_df = pd.DataFrame(index=daily_vol_30m.index)
        final_df = final_df.join(daily_vol_30m, how='left')
        if not weekly_features_daily.empty:
            final_df = final_df.join(weekly_features_daily.reindex(final_df.index,
method='ffill'), how='left')
    elif not weekly_features_daily.empty:
        final_df = weekly_features_daily.copy()
    else:
        # Or handle error: no base data to merge onto
        print("Error: No base daily data to merge features onto.")
        final_df = pd.DataFrame()


# --- 4. Handle Initial NaNs for all features ---
# Example: Backward fill for indicators that have calculation-induced NaNs at the start
# This should be done carefully, identifying which columns need it.
# For simplicity, let's assume all columns in final_df might have initial NaNs
if not final_df.empty:
    for col in final_df.columns:
        # A more targeted bfill would be better, only on actual indicator columns
        # that are known to have startup NaNs.
        final_df[col] = final_df[col].bfill()

    # Drop any remaining NaNs (e.g., if bfill couldn't fill everything due to full NaN columns at start)
    final_df = final_df.dropna()

# --- 5. Further Feature Engineering on final_df (as per Table 2) ---
# Example: Price deviation from its daily GMA
if f'close_1d' in final_df.columns and f'GMA_1_1d' in final_df.columns:
    final_df = (final_df[f'close_1d'] - final_df[f'GMA_1_1d']) / final_df[f'GMA_1_1d']

#... add other engineered features...

# Display some info
# print(final_df.head())
# print(final_df.info())
```

Note on load_and_preprocess function: The renaming logic for duplicated columns like

'Gaussian Moving Average' and 'RSI' has been made more robust. It now checks if multiple such columns exist before attempting to rename them with suffixes, and also handles the base case where only one such column might exist (e.g. 'GMA_1d' if only one GMA column was in the original daily CSV). Standard OHLCV and indicator columns are also suffixed to avoid clashes when joining dataframes from different timeframes.
Note on final_df creation and joining: Added checks for empty DataFrames before joining and a basic strategy to establish final_df if df_1d is empty but other aggregated/disaggregated data exists. A more robust solution might involve defining a master date range.
Note on bfill: The bfill for initial NaNs is applied broadly. In a production system, this should be more targeted to columns known to have leading NaNs due to indicator calculation windows.
This pseudo-code outlines the key steps. The actual implementation will require careful handling of column names (as they appear in the full CSVs), specific choices for NaN imputation, and detailed logic for each engineered feature.

6.3. Suggestions for Further Research and Development

Once the core improvements are implemented, further avenues for research and development include:

- **Alternative Data Integration:** Explore incorporating alternative datasets (e.g., sentiment data from news/social media, macroeconomic releases, satellite imagery for commodity-related assets if applicable) to capture different sources of alpha.[97] This often requires specialized data sourcing and feature engineering.
- **Advanced Model Architectures:**
    - If Transformers are explored, investigate architectures specifically designed or adapted for financial time series, such as those mentioned in research papers that deal with financial data's unique characteristics (e.g., non-stationarity, noise).[30]
    - For Reinforcement Learning, delve deeper into environment design, reward shaping, and agent hyperparameter tuning using libraries like FinRL or TensorTrade, focusing on portfolio allocation tasks.[42]
- **Dynamic Model Updating and Regime Switching:** Develop mechanisms for the ML models to adapt to changing market regimes. This could involve retraining models more frequently based on WFO principles, or explicitly modeling regimes and using different models or parameters for different detected regimes.
- **Hyperparameter Optimization:** Employ sophisticated hyperparameter optimization techniques like Bayesian Optimization, which can be more efficient than grid search or random search for finding optimal model parameters.[99] Libraries like scikit-optimize or Optuna can facilitate this.
- **Custom XAI Visualizations:** Develop bespoke visualizations for SHAP or LIME outputs tailored to financial portfolio context, making it easier to understand

feature contributions to allocation decisions or return predictions over time.
- **Portfolio Construction with Non-linear Costs:** For highly active strategies or large portfolios, investigate optimization models that incorporate non-linear transaction costs (market impact) using specialized solvers or libraries like Cvxportfolio.[94]

By systematically addressing these areas, the Jupyter notebook can evolve into a more powerful, robust, and insightful tool for machine learning-driven portfolio construction. The provided datasets offer a solid foundation for many of these enhancements, particularly in multi-timeframe feature engineering and the application of diverse ML models.

7. Conclusion

The effective integration of multi-frequency financial data and the adoption of advanced machine learning and portfolio construction methodologies can significantly enhance the capabilities of the user's Jupyter notebook. The provided S&P 500 datasets (30-minute, daily, and weekly) offer a rich tapestry of information, from intraday volatility patterns to long-term trend regimes. By strategically engineering features that capture these multi-scale dynamics—such as daily realized volatility from 30-minute data and weekly trend context for daily models—the predictive power of the underlying machine learning models can be substantially improved.

Key recommendations include a meticulous approach to data preprocessing, ensuring correct handling of timestamps and unique feature identification from the TradingView CSVs. The interpretation and utilization of pre-calculated indicators like Gaussian Moving Averages, MA Crossovers, RSI, Bollinger Bands, and Divergence signals should form the basis for a sophisticated feature set. Moving beyond raw indicator values to derived features (e.g., spreads, ratios, normalized deviations, event flags) is crucial.

From a modeling perspective, exploring time-series specific architectures such as LSTMs, GARCH models (for volatility), and potentially Transformer networks can offer advantages over simpler models. Reinforcement Learning presents a paradigm shift by learning direct allocation policies. Regardless of the model, robust backtesting through Walk-Forward Optimization and statistical significance testing of results are paramount to build confidence and avoid overfitting.

Finally, the portfolio construction process itself can be refined by considering alternatives to basic Mean-Variance Optimization. Techniques like Hierarchical Risk

Parity, the Black-Litterman model (which can elegantly incorporate ML-derived views), and CVaR optimization (for explicit tail-risk management) offer more nuanced approaches. Crucially, the incorporation of realistic transaction costs into the optimization process is essential for translating theoretical performance into practical, implementable strategies.

By systematically implementing these enhancements, the user can develop a more sophisticated, robust, and potentially more profitable machine learning-driven portfolio construction framework. The journey involves iterative experimentation, rigorous validation, and a continuous quest for deeper insights from the data.

Works cited

1. SPX_1W.csv
2. Gaussian SWMA For Loop — Indicator by Coff3eG - TradingView, accessed June 1, 2025, https://www.tradingview.com/script/WgqYPyvU-Gaussian-SWMA-For-Loop/
3. Gaussian — Indicators and Strategies — TradingView — India India, accessed June 1, 2025, https://in.tradingview.com/scripts/gaussian/
4. MA Cross — TradingView, accessed June 1, 2025, https://www.tradingview.com/support/solutions/43000599879-ma-cross/
5. What is Divergence? for BINANCE:BTCUSDT by Crypto4light ..., accessed June 1, 2025, https://www.tradingview.com/chart/BTCUSDT/0megIXsB-What-is-Divergence/
6. Feature engineering for time-series data - Statsig, accessed June 1, 2025, https://www.statsig.com/perspectives/feature-engineering-timeseries
7. Feature Engineering | Databricks, accessed June 1, 2025, https://www.databricks.com/glossary/feature-engineering
8. Feature Engineering For Financial Data - FasterCapital, accessed June 1, 2025, https://fastercapital.com/topics/feature-engineering-for-financial-data.html
9. Feature Engineering With Different Time Frame Data : r/quant - Reddit, accessed June 1, 2025, https://www.reddit.com/r/quant/comments/1avc6jd/feature_engineering_with_different_time_frame_data/
10. Mastering Real-Time Feature Engineering in Machine Learning | JFrog ML - Qwak, accessed June 1, 2025, https://www.qwak.com/post/real-time-feature-engineering
11. Top 4 Techniques for Handling Missing Values in Machine Learning, accessed June 1, 2025, https://blog.paperspace.com/top-4-techniques-for-handling-the-missing-values-in-machine-learning/
12. Missing Values Handling for Machine Learning Portfolios - arXiv, accessed June 1, 2025, https://arxiv.org/html/2207.13071v6
13. risklab.ai, accessed June 1, 2025,

https://risklab.ai/research/financial-data-science/fractional_differentiation#:~:text=In%20summary%2C%20fractional%20differentiation%20offers,machine%20learning%20models%20in%20finance.
14. Fractionally Differentiated - Mlfin.py, accessed June 1, 2025, https://mlfinpy.readthedocs.io/en/latest/FractionalDifferentiated.html
15. Fractional Differentiation and Memory | RiskLab AI, accessed June 1, 2025, https://www.risklab.ai/research/financial-data-science/fractional_differentiation
16. User Guide - Mlfin.py - Read the Docs, accessed June 1, 2025, https://mlfinpy.readthedocs.io/en/latest/UserGuide.html
17. Book2-Chapter5-Fractionally Differentiated Features - YouTube, accessed June 1, 2025, https://www.youtube.com/watch?v=acyAhc7B7ml
18. View this page - Mlfin.py, accessed June 1, 2025, https://mlfinpy.readthedocs.io/en/latest/_sources/index.rst.txt
19. Mlfin.py, accessed June 1, 2025, https://mlfinpy.readthedocs.io/
20. mlfinlab/mlfinlab/features/fracdiff.py at master · hudson-and-thames/mlfinlab - GitHub, accessed June 1, 2025, https://github.com/hudson-and-thames/mlfinlab/blob/master/mlfinlab/features/fracdiff.py
21. Machine Learning Trading Essentials (Part 2): Fractionally differentiated features, Filtering, and Labelling - Hudson & Thames, accessed June 1, 2025, https://hudsonthames.org/machine-learning-trading-essentials-part-2-fractionally-differentiated-features-filtering-and-labelling/
22. Wavelet Transform Essentials: Your Step-by-Step Guide for Data Science, accessed June 1, 2025, https://www.numberanalytics.com/blog/wavelet-transform-essentials-guide
23. Portfolio Construction Based on LSTM RNN and Black-Litterman Model: Evidence from Yahoo Finance - SciTePress, accessed June 1, 2025, https://www.scitepress.org/Papers/2024/132254/132254.pdf
24. Portfolio Optimization Using LSTM for Five Selected Stocks | Advances in Economics, Management and Political Sciences, accessed June 1, 2025, https://www.ewadirect.com/proceedings/aemps/article/view/15452
25. Reinforcement Learning Framework for Quantitative Trading - arXiv, accessed June 1, 2025, https://arxiv.org/html/2411.07585v1
26. GARCH Model: Definition and Uses in Statistics - Investopedia, accessed June 1, 2025, https://www.investopedia.com/terms/g/garch.asp
27. Dipartimento di Economia e Finanza Cattedra di Empirical Finance Multivariate GARCH and Portfolio Optimization, accessed June 1, 2025, https://tesi.luiss.it/36218/1/724101_BRARDINONI_EDOARDO.pdf
28. A Practical EGARCH Implementation in R and Python, accessed June 1, 2025, https://www.numberanalytics.com/blog/practical-egarch-implementation-r-python
29. How to implement GARCH models in Python, accessed June 1, 2025, https://campus.datacamp.com/courses/garch-models-in-python/garch-model-fundamentals?ex=9
30. (PDF) Financial Time Series Analysis with Transformer Models, accessed June 1,

2025, https://www.researchgate.net/publication/387524930_Financial_Time_Series_Analysis_with_Transformer_Models
31. Deep Learning in Quantitative Finance: Transformer Networks for ..., accessed June 1, 2025, https://blogs.mathworks.com/finance/2024/02/02/deep-learning-in-quantitative-finance-transformer-networks-for-time-series-prediction/
32. Exploring Mastering Transformers For Time Series Forecasting, accessed June 1, 2025, https://www.dhiwise.com/post/mastering-transformers-for-time-series-forecasting
33. TheFinAI/finma-7b-full - Hugging Face, accessed June 1, 2025, https://huggingface.co/TheFinAI/finma-7b-full
34. 9 Best Python Natural Language Processing (NLP) Libraries - Sunscrapers, accessed June 1, 2025, https://sunscrapers.com/blog/9-best-python-natural-language-processing-nlp/
35. Time series forecasting with PyTorch - GitHub, accessed June 1, 2025, https://github.com/sktime/pytorch-forecasting
36. Time Series Prediction with Hugging Face Transformers - wellsr.com, accessed June 1, 2025, https://wellsr.com/python/time-series-prediction-with-hugging-face-transformer/
37. [2503.04143] MTS: A Deep Reinforcement Learning Portfolio Management Framework with Time-Awareness and Short-Selling - arXiv, accessed June 1, 2025, https://arxiv.org/abs/2503.04143
38. A Beginner's Guide to Deep Reinforcement Learning - GeeksforGeeks, accessed June 1, 2025, https://www.geeksforgeeks.org/a-beginners-guide-to-deep-reinforcement-learning/
39. Deep Q networks in Python: Mastering reinforcement learning - BytePlus, accessed June 1, 2025, https://www.byteplus.com/en/topic/514188
40. Essential Guide to Actor Critic Algorithms in RL - Number Analytics, accessed June 1, 2025, https://www.numberanalytics.com/blog/essential-guide-actor-critic-algorithms-rl
41. Portfolio Optimization using Deep Reinforcement Learning models - Lund University Publications, accessed June 1, 2025, https://lup.lub.lu.se/student-papers/record/9178260/file/9178261.pdf
42. utyug/FinRL-Library: A Deep Reinforcement Learning Library for Automated Trading in Quantitative Finance. NeurIPS 2020. Please star. - GitHub, accessed June 1, 2025, https://github.com/utyug/FinRL-Library
43. [2011.09607] FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance - arXiv, accessed June 1, 2025, https://arxiv.org/abs/2011.09607
44. FinRL-Tutorials/4-Optimization/FinRL_HyperparameterTuning_Optuna.ipynb at master, accessed June 1, 2025,

https://github.com/AI4Finance-Foundation/FinRL-Tutorials/blob/master/4-Optimization/FinRL_HyperparameterTuning_Optuna.ipynb
45. FinRL-Tutorials/1-Introduction/Stock_NeurIPS2018_SB3.ipynb at master - GitHub, accessed June 1, 2025, https://github.com/AI4Finance-Foundation/FinRL-Tutorials/blob/master/1-Introduction/Stock_NeurIPS2018_SB3.ipynb
46. FinRL-Tutorials/5-Others/tutorial_multistock_variant_2.ipynb at master - GitHub, accessed June 1, 2025, https://github.com/AI4Finance-Foundation/FinRL-Tutorials/blob/master/5-Others/tutorial_multistock_variant_2.ipynb
47. FinRL-Tutorials/1-Introduction/FinRL_PortfolioAllocation_NeurIPS_2020.py at master, accessed June 1, 2025, https://github.com/AI4Finance-Foundation/FinRL-Tutorials/blob/master/1-Introduction/FinRL_PortfolioAllocation_NeurIPS_2020.py
48. Using Reinforcement Learning for Stock Trading with FinRL - Finding Theta, accessed June 1, 2025, https://www.findingtheta.com/blog/using-reinforcement-learning-for-stock-trading-with-finrl
49. FinRL/examples/Stock_NeurIPS2018_SB3.ipynb at master · AI4Finance-Foundation/FinRL - GitHub, accessed June 1, 2025, https://github.com/AI4Finance-Foundation/FinRL/blob/master/examples/Stock_NeurIPS2018_SB3.ipynb
50. AI4Finance-Foundation/FinRL: FinRL®: Financial ... - GitHub, accessed June 1, 2025, https://github.com/AI4Finance-Foundation/FinRL
51. Welcome to FinRL Library! — FinRL 0.3.1 documentation, accessed June 1, 2025, https://finrl.readthedocs.io/en/latest/
52. tensortrade-org - GitHub, accessed June 1, 2025, https://github.com/tensortrade-org
53. TensorTrade — TensorTrade 1.0.4-dev1 documentation, accessed June 1, 2025, http://tensortrade.org/
54. Training RL models for financial trading using TensorTrade on Amazon SageMaker Studio, accessed June 1, 2025, https://www.youtube.com/watch?v=Bfh6lyu9ca8
55. tensortrade-org/tensortrade: An open source reinforcement learning framework for training, evaluating, and deploying robust trading agents. - GitHub, accessed June 1, 2025, https://github.com/tensortrade-org/tensortrade
56. Portfolio Optimization Examples Using Financial Toolbox - MATLAB & Simulink - MathWorks, accessed June 1, 2025, https://www.mathworks.com/help/finance/portfolio-optimization-examples.html
57. Harnessing Volatility Targeting in Multi-Asset Portfolios - Research Affiliates, accessed June 1, 2025, https://www.researchaffiliates.com/publications/articles/1014-harnessing-volatility-targeting
58. Master Multi-Asset Portfolio Management with Python | Algo Trading Strategy - YouTube, accessed June 1, 2025, https://www.youtube.com/watch?v=USonmkLSbds

59. pawelkn/btester: Python framework optimized for running backtests on multiple asset portfolios - GitHub, accessed June 1, 2025, https://github.com/pawelkn/btester
60. accessed December 31, 1969, https://tensortrade.org/
61. Risks and Limitations of Backtesting | TrendSpider Learning Center, accessed June 1, 2025, https://trendspider.com/learning-center/risks-and-limitations-of-backtesting/
62. Walk-Forward Optimization: How It Works, Its Limitations, and Backtesting Implementation, accessed June 1, 2025, https://blog.quantinsti.com/walk-forward-optimization-introduction/
63. How is Cross-validation Used in Finance? - BytePlus, accessed June 1, 2025, https://www.byteplus.com/en/topic/475118
64. Bootstrap Examples - Sharpe Ratio - ARCH, accessed June 1, 2025, https://arch.readthedocs.io/en/latest/bootstrap/bootstrap_examples.html
65. Robust Performance Hypothesis Testing With the Variance - Department of Economics - University of Zurich, accessed June 1, 2025, https://www.econ.uzh.ch/dam/jcr:ffffffff-961c-1dd9-0000-000044710066/RobustVariance.pdf
66. Robust performance hypothesis testing with the Sharpe ratio, accessed June 1, 2025, http://www.ledoit.net/jef_2008pdf.pdf
67. Data Science for Financial Markets - Kaggle, accessed June 1, 2025, https://www.kaggle.com/code/lusfernandotorres/data-science-for-financial-markets
68. quantstats - Codesandbox, accessed June 1, 2025, http://codesandbox.io/p/github/ranaroussi/quantstats
69. LIME vs SHAP: A Comparative Analysis of Interpretability Tools - MarkovML, accessed June 1, 2025, https://www.markovml.com/blog/lime-vs-shap
70. An Introduction to SHAP Values and Machine Learning Interpretability - DataCamp, accessed June 1, 2025, https://www.datacamp.com/tutorial/introduction-to-shap-values-machine-learning-interpretability
71. Demystifying AI Decisions: A Comprehensive Guide to Explainable AI with LIME and SHAP, accessed June 1, 2025, https://www.cohorte.co/blog/demystifying-ai-decisions-a-comprehensive-guide-to-explainable-ai-with-lime-and-shap
72. Explainable AI for credit card fraud detection: Bridging the gap between accuracy and interpretability - | World Journal of Advanced Research and Reviews, accessed June 1, 2025, https://journalwjarr.com/sites/default/files/fulltext_pdf/WJARR-2025-0492.pdf
73. Portfolio Optimization with Python: Hierarchical Risk Parity - Yang Wu, accessed June 1, 2025, https://kenwuyang.com/posts/2024_10_20_portfolio_optimization_with_python_hierarchical_risk_parity/
74. Other Optimizers — PyPortfolioOpt 1.5.4 documentation, accessed June 1, 2025, https://pyportfolioopt.readthedocs.io/en/latest/OtherOptimizers.html

75. skfolio — skfolio 0.9.1 documentation, accessed June 1, 2025, https://skfolio.org/
76. skfolio/skfolio: Python library for portfolio optimization built on top of scikit-learn - GitHub, accessed June 1, 2025, https://github.com/skfolio/skfolio
77. Black-Litterman Allocation — PyPortfolioOpt 1.5.4 documentation, accessed June 1, 2025, https://pyportfolioopt.readthedocs.io/en/latest/BlackLitterman.html
78. Black-Litterman Asset Allocation Model - Portfolio Visualizer, accessed June 1, 2025, https://www.portfoliovisualizer.com/black-litterman-model
79. PyPortfolioOpt/docs/BlackLitterman.rst at master - GitHub, accessed June 1, 2025, https://github.com/robertmartin8/PyPortfolioOpt/blob/master/docs/BlackLitterman.rst
80. Riskfolio-Lib 7.0, accessed June 1, 2025, https://riskfolio-lib.readthedocs.io/en/latest/index.html
81. RM-CVaR: Regularized Multiple β-CVaR Portfolio - IJCAI, accessed June 1, 2025, https://www.ijcai.org/proceedings/2020/0629.pdf
82. Bayesian Optimization for CVaR-based portfolio optimization - arXiv, accessed June 1, 2025, https://arxiv.org/html/2503.17737v1
83. Use CVaR Daily: Guide to Portfolio Risk Management, accessed June 1, 2025, https://www.numberanalytics.com/blog/use-cvar-daily-portfolio-risk-management
84. PyPortfolioOpt/docs/GeneralEfficientFrontier.rst at master - GitHub, accessed June 1, 2025, https://github.com/robertmartin8/PyPortfolioOpt/blob/master/docs/GeneralEfficientFrontier.rst
85. robertmartin8/PyPortfolioOpt: Financial portfolio optimisation in python, including classical efficient frontier, Black-Litterman, Hierarchical Risk Parity - GitHub, accessed June 1, 2025, https://github.com/robertmartin8/PyPortfolioOpt
86. skfolio.optimization.convex._distributionally_robust的源代码, accessed June 1, 2025, https://www.aidoczh.com/skfolio/_modules/skfolio/optimization/convex/_distributionally_robust.html
87. API Reference - skfolio documentation, accessed June 1, 2025, https://skfolio.org/api.html
88. Riskfolio-Lib 7.0, accessed June 1, 2025, https://riskfolio-lib.readthedocs.io/
89. Riskfolio-Lib 0.2.0.2 - PyPI, accessed June 1, 2025, https://pypi.org/project/Riskfolio-Lib/0.2.0.2/
90. Parameters Estimation - Riskfolio-Lib 7.0, accessed June 1, 2025, https://riskfolio-lib.readthedocs.io/en/latest/parameters.html
91. Reports Functions - Riskfolio-Lib 7.0, accessed June 1, 2025, https://riskfolio-lib.readthedocs.io/en/latest/reports.html
92. IN CASE YOU MISSED IT: Machine Learning and the Implementable efficient frontier, accessed June 1, 2025, https://www.inquire-europe.org/news/in-case-you-missed-it-machine-learning-and-the-implementable-efficient-frontier/
93. Portfolio Optimization Problems with Transaction Costs - DiVA portal, accessed

June 1, 2025, https://www.diva-portal.org/smash/get/diva2:1776878/FULLTEXT01.pdf
94. Case Study: Portfolio Optimization with Nonlinear Transaction Costs - Stan Uryasev, accessed June 1, 2025, https://uryasev.ams.stonybrook.edu/research/testproblems/financial_engineering/portfolio-optimization-with-nonlinear-transaction-costs/
95. Cvxportfolio Documentation — Cvxportfolio 1.5.0 documentation, accessed June 1, 2025, https://www.cvxportfolio.com/
96. Cost models — Cvxportfolio 1.5.0 documentation, accessed June 1, 2025, https://www.cvxportfolio.com/en/stable/costs.html
97. AI, Automation, and Alpha: Alternative Data Trends 2025 - Kadoa, accessed June 1, 2025, https://www.kadoa.com/blog/alternative-data-trends
98. Exabel's 2025 Alternative Data Research Report is Available Now, accessed June 1, 2025, https://www.exabel.com/blog/exabels-2025-alternative-data-research-report-is-available-now/
99. Boosting Hyperparameter Tuning Efficiency Using Bayesian Optimization Techniques, accessed June 1, 2025, https://www.numberanalytics.com/blog/boosting-hyperparameter-tuning-efficiency-using-bayesian-optimization-techniques
100. Intro to MLOps: Hyperparameter Tuning - Weights & Biases - Wandb, accessed June 1, 2025, https://wandb.ai/site/articles/intro-to-mlops-hyperparameter-tuning/