GEORGIA INSTITUTE OF TECHNOLOGY

ISYE 6767 PROJECT 2

Data Analysis and Machine Learning

Student Name:
Chongrui Wang

Student ID:
xxx

December 2, 2023

Contents

1 Classes Overview

1.1 CustomCSVData Class

1.1.1 Introduction

The `CustomCSVData` class extends `bt.feeds.GenericCSVData`, aimed at handling various CSV formats in financial time series analysis.

1.1.2 Inheritance

`CustomCSVData` derives from `bt.feeds.GenericCSVData`, leveraging and expanding its capabilities to support multiple CSV formats.

1.1.3 Parameters

Key parameters like `csvformat` and `dtformat` are defined to customize the handling of CSV file structures and date formatting.

1.1.4 Constructor

The constructor of `CustomCSVData` initializes the class, selecting appropriate CSV format configuration methods based on the `csvformat` parameter, supporting several formats and providing error handling for unsupported ones.

1.1.5 Configuring CSV Types

- **configure_loader_type1:** Tailors the data loader for the default CSV format (type1).
- **configure_loader_type2:** Adapts the data loader for an alternative CSV format (type2).
- **configure_loader_type3:** Sets up the data loader for a distinct CSV format (type3), with unique column mappings and datetime formatting.

1.1.6 Data Loading and Processing

The `start` method oversees the loading of CSV data. It employs pandas for reading and sorting the data by datetime, processing it as per the specified CSV format.

1.1.7 Implementation Overview

An instance of `CustomCSVData` is created with a chosen CSV format. The class automatically configures the data loading process based on the given format.

1.2 AlphaFactors Class

1.2.1 Purpose

The `AlphaFactors` class is designed for financial data analysis. It processes a DataFrame containing stock market data and calculates various technical and statistical factors that are often used in quantitative finance.

1.2.2 Constructor

The constructor of `AlphaFactors` takes a DataFrame as an input. It makes a copy of this DataFrame, sorts it by the 'ticker' and 'date' columns, and resets the index. This initial processing prepares the data for factor calculation.

1.2.3 calculate factors Method

The `calculate_factors` method is the primary method that triggers the calculation of different factors. It calls two internal methods: `_kbar_factors` and `_rolling_factors`.

1.2.4 kbar factors Method

This method calculates factors based on daily stock prices, like open, high, low, and close prices. Factors such as 'KMID', 'KLEN', 'KMID2', and others are computed to capture different aspects of daily price movements.

1.2.5 rolling factors Method

`_rolling_factors` computes a series of rolling window-based factors for each 'ticker'. It includes various statistical measures like rate of change (ROC), moving averages (MA), standard deviation (STD), and others over different time windows. This method also calculates factors based on linear regression, such as the slope (used as a proxy for beta) and R-squared values.

1.2.6 Static Methods for Statistical Calculations

The class includes several static methods like `_linear_regression_slope`, `_rsquared`, and `_linear_regression_residual`. These methods are used to perform linear regression analysis on the data within the rolling windows, providing insights into trends and the quality of the fit.

1.2.7 Implementation Overview

To use the `AlphaFactors` class, an instance is created with a stock market DataFrame. The `calculate_factors` method is then called to generate various technical and statistical factors. These factors are essential for quantitative analysis and strategy development in finance.

1.3 AlphaFactorMetrics Class

1.3.1 Purpose

The `AlphaFactorMetrics` class is designed to analyze alpha factors in financial datasets. It calculates metrics like the Information Coefficient (IC) and Rank Information Coefficient (Rank IC) to evaluate the predictive power of these factors.

1.3.2 Constructor

The constructor of `AlphaFactorMetrics` initializes the class with a DataFrame containing alpha factors, future returns, ticker symbols, and dates. The column names for returns, tickers, and dates can be specified.

1.3.3 calculate ic Method

`calculate_ic` computes the Information Coefficient for a given alpha factor. This coefficient measures the correlation between the factor values and future returns on each date. The method averages these correlations across all dates to provide a single IC value.

1.3.4 calculate rank ic Method

Similar to `calculate_ic`, `calculate_rank_ic` calculates the Rank Information Coefficient. It uses Spearman's rank correlation instead of Pearson's, assessing the predictive power of the factor rankings.

1.3.5 top n factors Method

This method ranks alpha factors based on their IC or Rank IC values. It evaluates a list of factors, computes their ICs (or Rank ICs if specified), and returns the top n factors. Factors are ranked by the absolute values of their ICs to capture both positive and negative relationships.

1.3.6 top n union factors Method

`top_n_union_factors` identifies the top factors based on both IC and Rank IC values, returning the union of these top performers. This approach considers both linear and rank-based correlations to provide a comprehensive view of the most predictive factors.

1.3.7 Implementation Overview

To use the `AlphaFactorMetrics` class, an instance is created with a DataFrame containing relevant financial data. The class provides methods to calculate IC and Rank IC for individual factors and to identify the top-performing factors based on these metrics.

1.4 SignalData Class

1.4.1 Purpose

The `SignalData` class extends the `bt.feeds.PandasData` class from the Backtrader library. Its primary role is to define a custom data structure for handling financial time series data along with additional signals for trading strategies.

1.4.2 Inheritance

This class inherits from `bt.feeds.PandasData`, leveraging its functionality to handle data in pandas DataFrame format, while adding extra features.

1.4.3 Data Columns Definition

The `cols` attribute in `SignalData` specifies the columns that the DataFrame is expected to contain. It includes standard OHLCV (Open, High, Low, Close, Volume) data columns and an additional 'predicted' column for trading signals or predictions.

1.4.4 Lines Creation

The `lines` attribute creates a tuple of these columns, which Backtrader uses to access data in the DataFrame. This attribute ensures that the class can handle each specified column as a line, a fundamental concept in Backtrader for data handling.

1.4.5 Parameters Definition

`params` is a dictionary that maps each column name to a default index value of -1, indicating that these columns are not part of the standard dataset structure and will be dynamically identified.

The 'datetime' parameter is explicitly set to None, allowing the user to specify the column to be used as the datetime index when initializing the feed.

1.4.6 Implementation Overview

When using `SignalData`, a user can feed a pandas DataFrame containing the specified columns into a Backtrader strategy. This class is particularly useful for strategies that rely on additional signals or predictions, as it seamlessly integrates this data into the Backtrader framework.

1.5 MLStrategy Class

1.5.1 Purpose

`MLStrategy` is a custom strategy class for Backtrader, designed to implement a trading strategy based on machine learning predictions. It makes trading decisions based on predicted values provided in the data feed.

1.5.2 Inheritance

This class inherits from `bt.Strategy`, allowing it to utilize the full range of Backtrader's strategy functionalities.

1.5.3 Initialization

In the `__init__` method, the strategy initializes references to the 'predicted', 'open', and 'close' data series from the provided data feed. It also sets up variables to keep track of orders, buy prices, and commissions.

1.5.4 Logging

The `log` method is a utility for logging messages with the date and custom text. It's useful for tracking the strategy's operations and debugging.

1.5.5 Order Notifications

`notify_order` responds to updates about orders. It logs details of executed orders, including buy/sell prices, costs, and commissions, and handles different order statuses like completion, cancellation, or rejection.

1.5.6 Trade Notifications

`notify_trade` is called after a trade is closed. It logs the gross and net profit or loss of the trade.

1.5.7 Next Open Logic

The `next_open` method contains the core trading logic. It executes at the open of each new trading period. Based on the predicted value, it decides whether to place a buy or sell order. The strategy uses an 'all-in' approach for buying (invests all available cash) and sells the entire position when indicated.

1.5.8 Strategy Execution

`MLStrategy` utilizes the predicted values in the data feed to make trading decisions. If the prediction indicates a buy signal, it buys shares at the next open. Conversely, if the prediction indicates a sell signal, it sells the entire position at the next open.

1.5.9 Implementation Overview

To deploy this strategy, a trader needs to feed it a data set that includes the necessary OHLCV values along with the machine learning model's predictions. The strategy makes its trading decisions based on these predictions.

1.6 Model Class

1.6.1 Purpose

The `Model` class serves as a wrapper for various machine learning models, facilitating tasks such as parameter setting, fitting, evaluation, and feature importance analysis. It supports models like Random Forest, Adaboost, XGBoost, Catboost, Light Gradient Boosting Machine, and Ridge.

1.6.2 Constructor

The constructor accepts a machine learning model instance, which can be any model from libraries like scikit-learn or XGBoost. The class is designed to be flexible, accommodating a wide range of model types.

1.6.3 Setting Parameters

The `set_params` method allows for the dynamic setting of model parameters, providing flexibility to adjust the model's configuration as needed.

1.6.4 Model Fitting

`fit` trains the model on the provided training data. This method is a straightforward interface to the model's native fitting function.

1.6.5 Scikit-learn Model Access

`get_sklearn_model` returns the underlying scikit-learn model, enabling access to all native functionalities and attributes of the model.

1.6.6 Cross-validation

The class offers two types of cross-validation methods: `cross_validation` for standard cross-validation and `cross_validation_ts` for time series cross-validation. These methods are crucial for assessing the model's performance.

1.6.7 Hyperparameter Optimization

`hyperopt` performs hyperparameter optimization using the Hyperopt library. It optimizes across a defined search space, accommodating different types of hyperparameters like continuous, integer, and categorical.

1.6.8 Feature Importance

`plot_feature_importance` visualizes the importance of different features for tree-based models. This method is essential for understanding which features most significantly impact the model's predictions.

1.6.9 SHAP Values

`get_shap_values` computes SHAP values for any given model and dataset. SHAP (SHapley Additive exPlanations) values provide insights into the impact of each feature on the model's output, offering a deeper understanding of model behavior.

1.6.10 Implementation Overview

To use the `Model` class, instantiate it with a chosen machine learning model. The class provides methods to set parameters, train the model, perform cross-validation, optimize hyperparameters, and analyze feature importance and SHAP values, making it a versatile tool for machine learning tasks in Python.

2 Customized Class

In the realm of algorithmic trading and financial data analysis, precise and flexible data handling is crucial. To this end, a customized data loader class, `CustomCSVData`, has been developed. This class extends the functionality of the Backtrader library, allowing for enhanced adaptability with various CSV data formats.

2.1 Class Overview

`CustomCSVData` is a subclass of `bt.feeds.GenericCSVData` and is tailored to load and preprocess financial data from CSV files in different formats. This customization enables seamless integration of diverse data structures into the backtesting environment provided by Backtrader.

2.2 Testing and Validation

The functionality and robustness of `CustomCSVData` have been thoroughly tested. All tests have been successfully passed, demonstrating the class's reliability and efficiency in handling financial data for backtesting purposes.

2.3 Further Information

For detailed information on how to conduct these tests and utilize `CustomCSVData` for specific backtesting scenarios, please refer to the `readme.md` or `readme.txt` file. This document provides comprehensive guidelines and examples to assist users in implementing and benefiting from the customized data loader.

3 Data Collection and Preprocessing

3.1 Collection of Stock Data

The foundation of our analysis is a robust dataset of stock prices. To achieve this, we utilized a Python script that automates the collection of stock data. The primary steps in this process are:

1. **Reading Ticker Symbols**: We began by reading a list of stock ticker symbols from a CSV file. This list guides the data collection process, ensuring we gather information on a predefined set of companies.
2. **Retrieving Historical Data**: For each ticker symbol, we used the `yfinance` library to fetch historical stock data. This step involved specifying a date range, ensuring we obtained a consistent dataset across all stocks.

3. **Data Consolidation**: The data from each ticker was then aggregated into a single DataFrame. This consolidation was crucial for simplifying subsequent analysis steps.

3.2 Data Issues and Their Resolutions

During the collection and preprocessing of stock data, several issues were encountered, which required specific resolution strategies to ensure data integrity and suitability for analysis.

3.2.1 Data Collection Challenges

1. **Incomplete Data:** One primary challenge was encountering missing values in the historical stock data. Missing data can lead to biased results and inaccuracies in analysis.
2. **Outliers in Data:** Outliers in stock prices were another issue. These extreme values, possibly due to market anomalies or data errors, could skew the analysis and lead to misleading conclusions.

3.2.2 Resolution Strategies

1. **Handling Missing Values:**
   - We addressed missing values through the forward-fill method (`fillna(method='ffill')`), which propagates the last valid observation forward. This approach maintains the continuity and integrity of the time series data, ensuring no gaps that could distort the analysis.
2. **Outlier Detection and Clipping:**
   - Outliers were managed by identifying and clipping extreme values based on the Median Absolute Deviation (MAD). Values beyond three times the MAD from the median were adjusted, normalizing the data distribution and minimizing the impact of anomalous spikes or drops in stock prices.
3. **Normalization of Features:**
   - To ensure consistent scale across different features, Z-score normalization was employed. This process standardizes the data, giving each feature a mean of zero and a standard deviation of one, which is particularly beneficial for certain machine learning algorithms.

3.3 Analysis of Preprocessed Financial Data

In our financial modeling process, we have conducted a thorough preprocessing of stock data and analyzed the distribution of the labels assigned to each stock. Here, we discuss the balance of labels and the impact of preprocessing steps on the dataset quality.

3.3.1 Label Balance Analysis

Figure 1 presents the count of labels 0 and 1 across different stock tickers. A balanced distribution between the two classes is critical for training unbiased and effective machine learning models. From the plot, it is observed that:

- The labels are relatively balanced for most tickers, which is conducive to training models that do not overfit to a particular class.
- There are, however, some variations in the label distribution among different stocks, which will be accounted for during the model training phase to ensure robustness across various market conditions.

3.3.2 Preprocessing Effectiveness

The preprocessing steps included handling missing values, clipping outliers, and normalizing feature scales. The effectiveness of these steps is evident in the following two plots.

As shown in Figure 2, we have few nan values for each feature. The preprocessing effectively addressed missing values across all features, with negligible NaN counts remaining. This ensures that our models will not be hindered by gaps in the data.

Figure 3 displays the range of features after normalization. The box plots indicate that all features now have a similar scale, with outliers properly managed. This standardization is crucial for the application of machine learning models that are sensitive to feature magnitude, particularly those employing regularization.

The preprocessing steps have ensured that our dataset is well-suited for the application of machine learning algorithms. With a balanced label distribution and standardized feature scales, we can expect our models to learn meaningful patterns relevant to stock price movements, rather than being influenced by noise or imbalanced data.

4 Feature Processing and Analysis

4.1 Feature Processing

In our stock price prediction model, we perform an extensive feature processing routine to prepare the data for analysis. This process is implemented through the `AlphaFactors` class in Python, which performs the following steps:

1. Sort the data by ticker and date to maintain consistency.
2. Reset the index of the DataFrame for ease of manipulation.
3. Calculate two sets of features: Kbar Factors and Rolling Factors.

4.2 Feature Logic

Each feature calculated by the `AlphaFactors` class captures unique aspects of stock price movements and trading volumes, providing valuable insights for our predictive models. The logic behind each category of features is as follows:

4.2.1 Kbar Factors

Kbar Factors are calculated directly from the daily stock price movements (open, high, low, and close prices). They provide insights into the intraday price behavior and volatility of the stocks:

- **KMID**: Represents the relative movement from the opening to closing prices, capturing the day's price movement direction.
- **KLEN**: Captures the total range of the stock price movement in a day, indicating the day's volatility.
- **KMID2**: A normalized version of KMID, adjusting for the day's total price range, providing a relative measure of the closing position within the day's range.
- **KUP**: Measures the relative distance from the day's high to the higher of the opening or closing price, indicating the strength of the upper shadow.
- **KUP2**: A normalized version of KUP, providing a relative measure of the upper shadow length in relation to the day's total price range.
- **KLOW**: Represents the distance from the day's low to the lower of the opening or closing price, showing the strength of the lower shadow.
- **KLOW2**: A normalized version of KLOW, indicating the lower shadow length relative to the total daily price range.
- **KSFT**: Measures the closing price's position relative to the day's high and low, indicating whether the close is nearer to the high or low of the day.
- **KSFT2**: A normalized version of KSFT, providing a proportionate measure of the closing position within the daily range.

4.2.2 Rolling Factors

Each rolling factor is calculated over various time windows and provides insights into different aspects of stock behavior:

- **Beta (BETA)**: Represents the linear regression slope of the closing prices over a window, serving as a proxy for the stock's volatility relative to the overall market.

- **R-Squared (RSQR)**: The square of the correlation coefficient between the stock's closing price and the market, indicating the proportion of variance in the stock's price that is predictable from the market.
- **Residuals of Linear Regression (RESI)**: Sum of squares of residuals from a linear regression, indicating how much the actual values deviate from the predicted values.
- **Max High Price (MAX)** and **Min Low Price (MIN)**: Capture the maximum high and minimum low prices within a rolling window, relative to the current price.
- **Quantile-based Features (QTLU, QTLD)**: The upper and lower quantiles of closing prices within a window, providing insight into the stock's typical high and low prices.
- **Rank (RANK)**: The relative rank of the current closing price within a rolling window, giving an idea of how the current price compares historically.
- **Relative Strength Index (RSV)**: A measure of recent price changes to evaluate overbought or oversold conditions.
- **Index of Max and Min (IMAX, IMIN)**: Indicates the relative position of the maximum high and minimum low within a rolling window.
- **Index of Max Difference (IMXD)**: Shows the difference in the positions of the maximum high and minimum low prices within a window.
- **Correlation with Volume (CORR, CORD)**: The correlation of closing prices and volume, and their respective percentage changes, providing insight into how price movements are related to trading volume.
- **Count of Positive and Negative Movements (CNTP, CNTN, CNTD)**: The proportion of days with positive and negative price movements in a window, and their difference, indicating market sentiment.
- **Sum of Positive and Negative Movements (SUMP, SUMN, SUMD)**: The proportion of the sum of positive and negative price changes, giving a sense of overall market direction.
- **Volume Moving Average (VMA)** and **Volume Standard Deviation (VSTD)**: The average and standard deviation of trading volume over a window, relative to current volume.
- **Weighted Volume Moving Average (WVMA)**: The standard deviation of weighted price changes by volume, relative to their average, indicating the volatility in volume-weighted price movements.
- **Sum of Volume Positive and Negative Movements (VSUMP, VSUMN, VSUMD)**: Measures the directionality of volume changes, providing insight into trading activity.

4.3 Evaluating Feature Effectiveness

The `AlphaFactorMetrics` class encompasses several methods for evaluating the effectiveness of alpha factors:

1. **Information Coefficient (IC)**: The IC measures the correlation between the alpha factors and future returns. A higher absolute value of IC indicates a stronger predictive power.
2. **Rank Information Coefficient (Rank IC)**: Similar to IC, but it measures the correlation between the ranks of the factors and returns. This is particularly useful for non-linear relationships.
3. **Top N Factors**: This function identifies the top N factors based on their IC or Rank IC values. It is crucial for isolating the most influential factors in our prediction model.
4. **Union of Top Factors**: By combining the top factors identified by both IC and Rank IC, we can ensure a comprehensive selection of the most predictive factors.

4.3.1 Calculating IC and Rank IC

The class calculates the IC and Rank IC as follows:

- For each alpha factor, we group the data by date and then calculate the Pearson correlation (for IC) or Spearman correlation (for Rank IC) between the factor values and future returns.
- The mean of these correlation values across all dates is computed to represent the overall effectiveness of each factor.

4.3.2 Selecting Top Factors

The top N factors are selected based on the absolute values of their IC or Rank IC scores. This process involves:

- Calculating IC or Rank IC for each factor in the provided list.
- Ranking the factors based on the absolute values of their scores.
- Selecting the top N factors for further analysis.

4.4 Top Factors in Training Dataset

The following table lists the top factors in the training dataset based on the Information Coefficient (IC), rounded to four decimal places:

Factor	IC
KMID2	-0.0283
MA10	0.0285
QTLD5	0.0291
RSV5	-0.0301
KMID	-0.0313
MA5	0.0325
QTLU10	0.0328
KSFT2	-0.0332
KSFT	-0.0348
QTLU5	0.0352

Table 1: Top Factors by Information Coefficient

4.4.1 Analysis

The table indicates a mix of positive and negative IC values among the top factors. Factors such as QTLU5, MA5, and QTLU10 show a positive correlation with future returns, suggesting that when these factors have higher values, the stock returns tend to be higher. Conversely, factors like KSFT, KSFT2, and KMID exhibit negative IC values, indicating an inverse relationship with future returns.

This mix of positive and negative ICs reflects the diverse nature of factors influencing stock prices. It suggests that both the presence and absence of certain market conditions, as captured by these factors, are significant in predicting future stock movements.

4.5 Feature Correlation Analysis

In the process of feature selection for our predictive model, we performed a correlation analysis to ensure that our features are adequately independent and relevant.

Figure 4 presents the correlation matrix for our features. The matrix is a crucial diagnostic tool as it provides insight into the potential collinearity between variables.

- Most features exhibit low to moderate correlation with each other, which is desirable in a predictive modeling context. This indicates that our features are primarily independent, reducing redundancy in the input data.
- The absence of perfect multicollinearity (correlation coefficient of 1 or -1) between any pair of features suggests that we have successfully avoided including derivative or duplicate features, which could otherwise lead to overfitting and instability in the model.
- The varied shades of blue and red across the matrix confirm that our features capture different aspects of the underlying data structure, which is beneficial for the model's ability to learn complex patterns and relationships.

4.5.1 Analysis

The analysis of the correlation matrix supports our feature selection process, indicating that the chosen features are suitable for modeling. The independence of the features allows our machine learning model to effectively learn from the data without being misled by redundant information. This careful feature selection process lays a solid foundation for building a robust predictive model.

5 Model Selection and Hyperparameter Optimization

5.1 Introduction

In our stock price prediction project, we have implemented a variety of machine learning models to evaluate their performance in predicting stock returns. The choice of models and their hyperparameters plays a crucial role in the accuracy and efficiency of our predictions.

5.2 Model Selection

We have considered the following models for our classification and regression tasks:

5.2.1 Random Forest (RF)

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees at training time. It offers high accuracy, handles non-linear data effectively, and provides indicators of feature importance.

5.2.2 Adaboost

Adaboost, or Adaptive Boosting, is a technique that combines multiple weak learners (typically decision trees) to create a strong learner. The model focuses on correctly predicting the instances where previous learners failed, thereby improving the overall accuracy.

5.2.3 XGBoost

XGBoost, or Extreme Gradient Boosting, is a highly efficient and scalable implementation of gradient boosting. It is known for its speed and performance and is particularly effective for large and complex datasets.

5.2.4 CatBoost

CatBoost is an algorithm based on gradient boosting over decision trees, with a specific focus on categorical data. It offers robust handling of categorical features and reduces the need for extensive data preprocessing.

5.2.5 Light Gradient Boosting Machine (LightGBM)

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed for distributed and efficient training, particularly on large datasets.

5.2.6 Ridge Regression

Ridge Regression is a type of linear regression that introduces a small amount of bias (known as regularization) to achieve a significant drop in variance, resulting in a more robust model.

Each of these models offers unique advantages in handling different aspects of stock market data. For instance, ensemble methods like Random Forest and boosting algorithms (Adaboost, XGBoost, CatBoost, LightGBM) are known for their robustness and ability to capture non-linear patterns. Ridge Regression, on the other hand, is effective in preventing overfitting through regularization.

5.3 Model Performance Metrics

5.3.1 Classification

The following table lists the out-of-sample accuracy and precision for various models without hyperparameter optimization when doing classification:

Model	**Accuracy**	**Precision**
Ridge	0.5030	0.5203
AdaBoost	0.5017	0.5189
CatBoost	0.5000	0.5178
LightGBM	0.5056	0.5227
XGBoost	0.5002	0.5181
RF	0.5040	0.5227

Table 2: Out-of-Sample Performance Metrics for Classification Models without Hyperparameter Optimization

5.3.2 Analysis

The table indicates that all models perform similarly in terms of accuracy and precision, with no single model significantly outperforming the others. This suggests that without hyperparameter optimization, the models' ability to differentiate between classes is limited, leading to performance close to random chance (50% accuracy).

The precision scores are slightly higher than the accuracy scores across all models, indicating a somewhat better performance in correctly identifying positive instances. However, the modest difference between accuracy and precision values across these models points to the need improvement.

5.3.3 Regression

The following table lists the out-of-sample accuracy and precision for various regression models without hyperparameter optimization:

Model	**Accuracy**	**Precision**
Ridge	0.5180	0.5182
AdaBoost	0.5182	0.5182
CatBoost	0.5072	0.5249
LightGBM	0.5069	0.5227
XGBoost	0.4981	0.5160
RF	0.5182	0.5184

Table 3: Out-of-Sample Performance Metrics for Regression Models without Hyperparameter Optimization

5.3.4 Analysis

The table shows that Ridge, AdaBoost, and RF models have similar performance in terms of accuracy and precision, with slightly higher values compared to other models. CatBoost, while having a lower accuracy, shows the highest precision among all models. This suggests that CatBoost, despite having a slightly lower overall accuracy, is more effective in correctly identifying positive instances when they occur.

LightGBM and XGBoost, on the other hand, show lower performance in both metrics. This might indicate that these models, in their current configuration without hyperparameter tuning, are less suited for the specific characteristics of the regression task at hand.

5.4 Hyperparameter Optimization

The hyperparameter optimization process is crucial for fine-tuning the models to achieve the best possible performance. We have utilized methods like grid search and random search to explore a wide range of parameter combinations. The optimization process aims to balance the trade-off between model complexity and generalization ability.

5.4.1 Bayesian Optimization

Bayesian Optimization is a probabilistic model-based approach for finding the minimum of any function that returns a real number. In our project, we use it for hyperparameter optimization of the models. It works by building a probability model of the objective function and using it to select the most promising hyperparameters to evaluate in the true objective function.

Bayesian Optimization is particularly efficient for optimization problems with expensive cost functions, limited data, and no simple convexity. It is preferred over traditional methods like Grid Search because it keeps track of past evaluation results and uses them to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function, thereby finding better hyperparameters more quickly.

5.4.2 Comparison with Grid Search

Unlike Grid Search, which exhaustively searches through a manually specified subset of the hyperparameter space, Bayesian Optimization is more effective in exploring the hyperparameter space using fewer iterations. Grid Search can be computationally expensive and less efficient, especially when dealing with a large number of hyperparameters and deeper models.

The choice of Bayesian Optimization aligns with the need for a more efficient and effective search method in our complex machine learning models. It allows us to optimize our models with a higher degree of accuracy and in less time compared to traditional methods like Grid Search.

5.4.3 Model Performance Metrics with Hyperparameters

The following table lists the out-of-sample accuracy and precision for various classification models with hyperparameters:

Model	**Accuracy**	**Precision**
Ridge	0.5027	0.5200
AdaBoost	0.5046	0.5198
LightGBM	0.5072	0.5246
XGBoost	0.4938	0.5117
CatBoost	0.5082	0.5268
RF	0.5020	0.5183

Table 4: Out-of-Sample Performance Metrics for Classification Models with Hyperparameters

5.5 Analysis of Model Performance

The performance metrics indicate varying degrees of success across different models. Notably, CatBoost exhibits the highest precision, suggesting its effectiveness in correctly identifying positive instances. Conversely, XGBoost shows lower performance in both accuracy and precision, which might indicate its sensitivity to the hyperparameter settings used.

5.5.1 Examined Hyperparameters

The following table summarizes the hyperparameters examined for each model:

Model	Hyperparameters
RF	max_features: 0.1-1, n_estimators: 10-200, max_depth: 3-20, criterion: gini/entropy
AdaBoost	learning_rate: 0.01-2, n_estimators: 50-200, algorithm: SAMME/SAMME.R
XGBoost	learning_rate: 0.01-1, subsample: 0.5-1, colsample_bytree: 0.5-1, reg_lambda: 0.01-10, reg_alpha: 0.01-10, n_estimators: 100-1000, max_depth: 3-20
CatBoost	learning_rate: 0.01-1, subsample: 0.1-1, iterations: 100-1000, depth: 3-10
LightGBM	learning_rate: 0.01-1, subsample: 0.1-1, colsample_bytree: 0.1-1, reg_lambda: 0.01-10, reg_alpha: 0.01-10, n_estimators: 100-1000, num_leaves: 20-150, max_depth: 3-20
Ridge	alpha: 0.1-10

Table 5: Examined Hyperparameters for Classification Models

5.5.2 Model Performance Metrics with Bootstrap Aggregation Method

The following table lists the out-of-sample accuracy and precision for various classification models using the bagging method:

Model	Accuracy	Precision
Ridge	0.5041	0.5224
XGBoost	0.4994	0.5179
AdaBoost	0.5035	0.5229
CatBoost	0.5031	0.5222
LightGBM	0.4998	0.5165
RF	0.5014	0.5180

Table 6: Out-of-Sample Performance Metrics for Classification Models with Bagging

5.5.3 Analysis

The performance metrics indicate that the models' accuracy after applying the bagging method is generally around 50%, suggesting only a marginal improvement over random guessing in a binary classification context. However, the precision scores are slightly higher, indicating a somewhat better performance in correctly identifying positive instances.

Among the models, Ridge and AdaBoost exhibit slightly better precision, which might suggest their effectiveness in handling the variance introduced by the bagging method. On the other hand, XGBoost and LightGBM show lower accuracy and precision, which could be due to their inherent complexities and how they interact with the bagging approach.

These results highlight the challenging nature of the classification task and the modest impact of the bagging method on improving model performance.

5.5.4 Conclusion

The analysis of model performance with hyperparameters indicates the importance of model selection and hyperparameter tuning in classification tasks. The hyperparameter ranges explored for each model show a tailored approach to optimizing model performance, highlighting the need for a careful balance between model complexity and predictive power.

5.6 Performance Evaluation

To assess the effectiveness of our models, we have implemented several evaluation metrics:

1. **Cross-Validation**: We use cross-validation techniques, including time series cross-validation, to gauge the model's performance on unseen data.
2. **Information Criteria**: Metrics like accuracy, precision, and mean squared error provide insights into the model's predictive power.
3. **Feature Importance and SHAP Values**: Understanding the contribution of each feature towards the model's predictions helps in refining the feature set and gaining insights into the underlying data patterns.

5.7 Deep Learning Application

5.7.1 LSTM (Long Short-Term Memory)

LSTMs are a fundamental part of sequential data modeling, especially effective in capturing long-term dependencies in time series data. Their ability to remember and utilize historical information over extended periods makes them particularly suitable for predicting stock prices, which often depend on long-term trends and patterns.

5.7.2 ALSTM (Attention-based LSTM)

ALSTM, or Attention-based Long Short-Term Memory, enhances the traditional LSTM model by incorporating an attention mechanism. This combination allows the model to focus on specific parts of the input sequence that are more relevant for prediction, a crucial advantage in time-series data like stock prices where certain time periods may be more indicative of future trends.

5.7.3 TCN (Temporal Convolutional Network)

Temporal Convolutional Networks (TCN) offer a novel approach to sequence modeling, employing a hierarchy of temporal convolutions. This structure allows TCNs to handle long-range dependencies with fewer parameters compared to traditional RNNs, making them efficient and effective for modeling financial time series data.

5.7.4 GATS (Graph Attention Networks)

Graph Attention Networks (GATS) are designed to handle data represented in a graph structure. In the context of stock price prediction, GATS can be used to model complex relationships and dependencies between different stocks or financial indicators, allowing the model to allocate attention differentially across the graph structure for more accurate predictions.

5.7.5 SFM (Sequential Feature Model)

Sequential Feature Models (SFM) are specialized in extracting and learning from the sequential features present in time-series data. In stock market prediction, SFMs can identify and leverage key temporal features from past stock price movements to predict future trends and price movements.

5.8 Data Preparation and Model Application Process

5.8.1 Data Preparation

The data preparation process involves combining training and testing datasets, extracting unique dates, and creating time-series data. This is crucial for maintaining the chronological order of stock prices, which is essential for time series modeling. We specifically handle train and test datasets by aligning them according to their dates and reshaping them for the deep learning models.

5.8.2 Feature Selection

Feature columns are identified by excluding certain columns (ignore_cols) that are not needed for prediction. This step is vital in focusing the model on relevant predictive features and avoiding noise that could impair the model's performance.

5.8.3 Data Preprocessing for Deep Learning

The data is then converted into tensors suitable for deep learning models. This includes handling missing values, normalizing the data, and reshaping it into a format compatible with the model's input requirements. Special attention is given to maintaining the sequence length (SEQ_LEN) for each batch.

5.8.4 Model Initialization

Based on the chosen method (classification or regression), different models such as LSTM, ALSTM, TCN, Transformer, GATS, and SFM are initialized. This flexibility allows for the exploration of various architectures to find the best fit for the given data and prediction task.

5.8.5 Model Training

The training process involves optimizing the model parameters using a loss function tailored to the method (classification or regression). The use of different loss functions like BinaryCrossEntropyLoss, FocalLoss, IC_loss, WeightedICLoss, and SharpeLoss allows for a more nuanced approach to model training, focusing on different aspects of the prediction task.

5.8.6 Model Evaluation

Model performance is evaluated using metrics such as accuracy and precision. This step is critical in assessing the model's ability to make accurate predictions and correctly classify or regress on unseen data.

5.9 Overview of Loss Functions

Different loss functions are employed in deep learning models to optimize their performance for specific tasks. Here we discuss the rationale behind each loss function used in stock price prediction models.

5.9.1 Binary Cross Entropy Loss

Binary Cross Entropy Loss is a standard loss function for binary classification tasks. It measures the difference between the predicted probabilities and the actual binary labels, making it suitable for models predicting two possible outcomes.

5.9.2 Focal Loss

Focal Loss addresses class imbalance by focusing more on hard-to-classify examples. It's particularly useful in scenarios where one class is significantly underrepresented, which is common in stock market datasets.

5.9.3 Information Coefficient Loss (IC_loss)

The IC loss function uses the Pearson correlation coefficient as its basis. It's effective for regression tasks in finance, where maintaining a linear relationship between predictions and actual values is crucial.

5.9.4 Weighted Information Coefficient Loss

This loss function is a variant of the IC loss, where predictions are weighted, potentially giving more importance to certain observations over others. This can be particularly useful in stock price prediction, where not all data points are equally informative.

5.9.5 Sharpe Loss

Sharpe Loss is based on the Sharpe Ratio, a measure of risk-adjusted return. It's designed to maximize the return of the model relative to its risk, a key consideration in financial modeling.

5.9.6 ListMLE and ListNet

ListMLE and ListNet are loss functions used in learning-to-rank tasks. They are suitable for models where the goal is to rank a list of items, such as stocks, in order of relevance or potential return.

5.10 Model Performance Metrics

5.10.1 Classification

The following table lists the out-of-sample accuracy and precision for various classification models with the BCE loss function:

Model	Accuracy	Precision
LSTM with BCE	0.5009	0.5173
ALSTM with BCE	0.5039	0.5212
TCN with BCE	0.4948	0.5213
SFM with BCE	0.5006	0.5205
GATS with BCE	0.5072	0.5203

Table 7: Out-of-Sample Performance Metrics for Classification Models with BCE Loss

5.10.2 Analysis

The table demonstrates that the deep learning models yield similar accuracy levels, hovering around the 50% mark, which suggests performance close to random chance in a binary classification task. However, the precision scores are slightly higher, indicating a better ability to identify positive instances correctly.

The TCN model, despite having the lowest accuracy, shows a competitive precision score, suggesting its potential in identifying true positive cases effectively. In contrast, the ALSTM model shows a better balance between accuracy and precision, indicating its effectiveness in the classification task with BCE loss.

5.10.3 Regression

The following table lists the out-of-sample accuracy and precision for various regression models with the IC loss function:

Model	Accuracy	Precision
LSTM with IC	0.5183	0.5182
ALSTM with IC	0.5174	0.5179
TCN with IC	0.5160	0.5195
SFM with IC	0.5181	0.5182
GATS with IC	0.5174	0.5180

Table 8: Out-of-Sample Performance Metrics for Regression Models with IC Loss

5.10.4 Analysis

The table indicates that all models, when using the IC loss function, demonstrate similar levels of out-of-sample accuracy and precision. The performance differences between LSTM, ALSTM, TCN, SFM, and GATS models are marginal, suggesting that the choice of model architecture may have less impact when the IC loss function is used in regression tasks.

The slight variations in precision across different models could be attributed to how each model architecture processes and learns from the time-series data. The TCN model, while having a slightly lower accuracy, shows the highest precision among all models, indicating its effectiveness in correctly identifying positive instances.

Overall, the results suggest that the IC loss function provides a consistent level of performance across different deep learning architectures in regression tasks, particularly in the context of stock price prediction.

5.10.5 Comparison between different loss functions

The following table lists the out-of-sample accuracy and precision for the LSTM model with different loss functions:

5.10.6 Analysis

The table indicates that the LSTM model's performance varies slightly with different loss functions. The IC, WIC, ListMLE, and ListNet loss functions show the best results in terms of accuracy and precision, suggesting their effectiveness in the context of LSTM models for the given task.

Loss Function	Accuracy	Precision
BCE	0.5009	0.5173
Focal	0.5005	0.5172
IC	0.5183	0.5182
WIC	0.5185	0.5184
Sharpe	0.5176	0.5180
ListMLE	0.5182	0.5182
ListNet	0.5183	0.5183

Table 9: Out-of-Sample Performance Metrics for LSTM Model with Various Loss Functions

The BCE and Focal loss functions result in similar and comparatively lower performance, which might be due to their general focus on balancing the prediction of positive and negative classes without specific consideration for the time-series nature of the data.

The IC-based losses (IC and WIC) and list-based losses (ListMLE and ListNet) seem to be more aligned with the model's objective, leading to a slightly better performance. This could be attributed to their ability to capture more nuanced aspects of the time-series data.

Overall, the choice of loss function appears to have a modest but noticeable impact on the LSTM model's performance, with certain loss functions like IC, WIC, ListMLE, and ListNet offering slight improvements in accuracy and precision.

5.10.7 Final Model Selection and Performance Analysis

After extensive evaluation of various machine learning models, LightGBM and CatBoost classifiers, with hyperparameter tuning, have been selected as the final models for the stock price prediction task. The decision is based on their performance metrics, which surpassed the defined minimum requirements. Also, I did not select deep learning model because of their high time complexity.

- **LightGBM Classifier:** Achieved an accuracy of 0.5072 and a precision of 0.5246.
- **CatBoost Classifier:** Attained an accuracy of 0.5082 and a precision of 0.5268.

These metrics exceed the minimum requirements set for accuracy (0.5014) and precision (0.5141), highlighting the effectiveness of these models in the classification task.

1. **Performance:** Both LightGBM and CatBoost have demonstrated robust predictive capabilities, as evidenced by their accuracy and precision scores.
2. **Hyperparameter Tuning:** The application of hyperparameter tuning has significantly enhanced the performance of these models, optimizing them for the specific nuances of stock price data.

3. **Model Characteristics:** LightGBM and CatBoost are known for their efficiency and accuracy in handling large datasets with categorical features, making them particularly suitable for financial market data.

The selection of LightGBM and CatBoost as final models is justified by their superior performance metrics. The decision aligns with the objective of achieving high accuracy and precision in predicting stock prices, ensuring that the models are reliable and effective for practical applications in financial analysis and trading strategies.

5.10.8 Rationale Behind Hyperparameter Selection

The choice of hyperparameters for both LightGBM and CatBoost classifiers was guided by a combination of model-specific characteristics and empirical testing. Key considerations included:

1. **Optimization for Financial Data:** The hyperparameters were chosen to optimize the models for the unique characteristics of financial market data, which often includes noisy, non-stationary, and time-sensitive patterns.
2. **Balancing Bias and Variance:** The parameters were tuned to achieve a balance between underfitting (bias) and overfitting (variance), ensuring that the models generalize well to unseen data.
3. **Computational Efficiency:** Given the large volume of financial data, parameters were also selected based on their impact on computational efficiency, ensuring that the models can be trained and deployed within reasonable time frames.

Specific hyperparameters, such as learning rate, depth of trees, and the number of estimators, were iteratively adjusted based on model performance on validation datasets. This empirical approach allowed us to refine the models for optimal performance.

5.10.9 Initial Expectations and Model Performance

The initial expectations from the LightGBM and CatBoost models were cautiously optimistic, considering their reputation for high performance in classification tasks. Key expectations included:

1. **Accuracy and Precision:** We anticipated that both models would achieve high accuracy and precision, surpassing our baseline metrics, due to their advanced algorithms and capability to handle complex data patterns.
2. **Handling Market Volatility:** Given the volatile nature of stock markets, we expected the models to robustly handle sudden market changes and anomalies, providing reliable predictions under varying market conditions.

3. **Feature Importance Insights:** An additional expectation was gaining insights into feature importance, helping us understand the driving factors behind stock price movements.

Upon running these models, our expectations were largely met. The models not only achieved high accuracy and precision but also provided valuable insights into the data, confirming their suitability for stock price prediction tasks. The results reaffirmed our choice of LightGBM and CatBoost as the final models for this project.

5.11 Overview of the Backtesting Strategy

The backtesting process is implemented using Python with the Backtrader library, focusing on integrating machine learning predictions into a trading strategy.

5.11.1 Data Preparation

- The `SignalData` class is defined to structure the input data. It includes open, high, low, close, volume (OHLCV), and the predicted signal from the machine learning model.
- This class extends `bt.feeds.PandasData` to customize the data feed for Backtrader.

5.11.2 Strategy Definition

- The `MLStrategy` class defines the trading strategy, inheriting from `bt.Strategy`.
- Key data points like predicted signals, open, and close prices are tracked for decision-making.
- Order management is handled within the strategy, tracking pending orders and execution details.

5.11.3 Logging and Order Notification

- A logging function is implemented for detailed output of the strategy's actions and order results.
- The `notify_order` method handles order status updates, reporting executions and failures.
- The `notify_trade` method reports the outcome of closed trades.

5.11.4 Trading Logic

- The `next_open` method is used for executing trades, leveraging the 'cheat-on-open' approach.
- If no current position and prediction is 1 (buy signal), a buy order is created at the next open price.
- If holding a position and prediction is 0 (sell signal), a sell order is placed to close the position at the next open price.

This backtesting strategy integrates machine learning predictions into a trading algorithm, allowing for testing the efficacy of the model in a simulated trading environment. The strategy focuses on executing orders based on predicted signals, aiming to capitalize on the foresight provided by the machine learning model.

5.12 Out-of-Sample Backtest Performance

5.12.1 Accuracy Comparison for Different Stocks

Ticker	LightGBM Accuracy	CatBoost Accuracy
CNP	0.5264	0.5183
DLTR	0.5183	0.5274
HIBB	0.5140	0.5021
RHI	0.5093	0.5202
TSLA	0.5088	0.5012
TGT	0.5074	0.5102
FIX	0.5026	0.5183
WMS	0.4976	0.5073
WBA	0.4960	0.4779
HAS	0.4902	0.4993

Table 10: Accuracy Comparison of LightGBM and CatBoost Classifiers for Different Stocks

Based on the performance metrics, CNP and DLTR are chosen with models LightGBM and CatBoost for further backtesting.

After extending the application of the LightGBM classifier to a broader set of stocks and performing hyperparameter tuning, we obtained the following optimal parameters and performance metrics for the stock DLTR:

```
LightGBMC optimal parameters for accuracy:
{
  'colsample_bytree': 0.3961,
  'learning_rate': 0.8794,
  'max_depth': 7,
  'n_estimators': 530,
  'num_leaves': 66,
  'reg_alpha': 5.9344,
  'reg_lambda': 2.5911,
  'subsample': 0.7548
}
```

- **LightGBM with DLTR:** Achieved an out-of-sample accuracy of 0.5045 and precision of 0.5309. For stock DLTR, the cumulative return was 828.99%, with a Sharpe ratio of 1.32, indicating a high level of risk-adjusted return.

Similarly, for the CatBoost classifier applied to stock CNP with hyperparameter tuning, we observed the following performance:

- **CatBoost with CNP:** Demonstrated an out-of-sample accuracy of 0.5045 and precision of 0.5309. For stock CNP, the cumulative return was 95.1%, with a Sharpe ratio of 0.54, suggesting a moderate level of risk-adjusted return.

We analyze key metrics of LightGBM and CatBoost strategies against their benchmarks, focusing on Sharpe Ratio, Max Drawdown, and win/loss ratios.

5.12.2 Analysis of LightGBM with DLTR

Metric	LightGBM	CatBoost	Benchmark
Cumulative Return	828.99%	151.21%	111.32%
CAGR%	30.66%	11.69%	9.39%
Sharpe	1.32	0.6	0.45
Max Drawdown	-38.07%	-31.77%	-44.64%
Win Days	52.9%	51.4%	51.77%

Table 11: Performance metrics of strategy with stock DLTR compared to Benchmark

5.12.3 Analysis of CatBoost with CNP

The application of LightGBM and CatBoost classifiers, with hyperparameter tuning, to the stocks DLTR and CNP respectively, has resulted in significant enhancements to the strategies. LightGBM with DLTR showed a remarkable increase in cumulative returns and a superior Sharpe ratio, while CatBoost with CNP presented a stable performance with reduced market exposure and a moderate Sharpe ratio. These results demonstrate the efficacy of the chosen models and hyperparameter tuning in optimizing stock classification strategies.

Metric	LightGBM	CatBoost	Benchmark
Cumulative Return	352.51%	95.1%	49.96%
CAGR%	19.86%	8.35%	4.98%
Sharpe	1.07	0.54	0.32
Max Drawdown	-26.35%	-30.22%	-59.8%
Win Days	54.13%	50.58%	53.67%

Table 12: Performance metrics of strategy with stock CNP compared to Benchmark

5.12.4 Model Pros and Cons

LightGBM Strategy:

- **Pros:** Exhibits superior risk-adjusted returns and lower drawdowns compared to the benchmark. Effective in volatile markets.
- **Cons:** May still experience significant drawdowns, indicating a need for cautious risk management.

CatBoost Strategy:

- **Pros:** Offers moderate improvements over the benchmark, suitable for more conservative investment strategies.
- **Cons:** Lower Sharpe Ratio than LightGBM, indicating potential underperformance in risk-adjusted terms.

5.12.5 Conclusion

The backtest results for LightGBM and CatBoost strategies show substantial improvement over the benchmark. The choice of CNP for LightGBM and DLTR for CatBoost is vindicated by their superior performance in backtesting, as reflected in metrics such as Sharpe ratio, CAGR, and maximum drawdown. These results demonstrate the effectiveness of the selected models and their suitability for stock price prediction and trading strategy development.

Both LightGBM and CatBoost strategies show improvements over their respective benchmarks, particularly in terms of Sharpe Ratio and Max Drawdown. The choice between these models should be informed by the investor's risk tolerance and investment goals, with LightGBM favoring more aggressive strategies and CatBoost aligning with conservative approaches.

6 Application of LightGBM on Extended Stock Data

6.1 Hyperparameter Tuning and Model Optimization

The LightGBM Classifier was applied to a broad set of stocks for classification tasks. Hyperparameter tuning was performed to optimize the model for accuracy. The optimal parameters found for the LightGBM classifier are as follows:

```
{
  'colsample_bytree': 0.39610348622145486,
  'learning_rate': 0.87943181392473,
  'max_depth': 7,
  'n_estimators': 530,
  'num_leaves': 66,
  'reg_alpha': 5.934387920797541,
  'reg_lambda': 2.5910712918390044,
  'subsample': 0.7548324215436715
}
```

6.2 Model Performance Metrics

The application of the LightGBM classifier with the optimal hyperparameters yielded the following out-of-sample performance metrics:

- Accuracy: 0.5045
- Precision: 0.5309

6.3 Accuracy Across Different Stocks

The following table illustrates the out-of-sample accuracy obtained for various tickers when applying the LightGBM classifier:

The LightGBM classifier, with carefully tuned hyperparameters, has demonstrated promising classification performance across a wide range of stocks. The model's accuracy varied among different tickers, indicating that it has adaptability to different stock behaviors within the market.

6.4 Stock Performance Analysis

The LightGBM classifier with hyperparameter tuning was applied to a broad range of stocks. We selected the top 10 stocks based on out-of-sample prediction accuracy. The stocks were evaluated over a period corresponding to 40% of the entire dataset. Below we present the trading analysis report for these stocks, ranked by their respective strategy Sharpe Ratio and Max Draw Down.

Ticker	**Accuracy**
KORS	0.5807
PRU	0.5703
TSS	0.5677
FE	0.5651
DRI	0.5625
MNST	0.5599
PPL	0.5573
HSIC	0.5573
PG	0.5547
V	0.5547

Table 13: Out-of-sample accuracy of LightGBM classifier for top 10 tickers

6.4.1 DRI

Metric	Strategy	Benchmark
Cumulative Return	27.53%	32.1%
CAGR%	17.41%	20.17%
Sharpe	0.95	0.87
Max Drawdown	-12.71%	-16.29%
Win Days	52.92%	55.53%

Table 14: Trading Analysis for DRI

DRI's strategy outperformed the benchmark in terms of the Sharpe ratio, suggesting a more efficient risk-adjusted return. However, it underperformed the benchmark in both cumulative return and CAGR%. The strategy experienced a smaller max drawdown, indicating a lower risk of substantial decline.

6.4.2 FE

Metric	Strategy	Benchmark
Cumulative Return	21.46%	32.12%
CAGR%	13.69%	20.18%
Sharpe	1.1	1.06
Max Drawdown	-12.72%	-16.12%
Win Days	52.82%	54.01%

Table 15: Trading Analysis for FE

The FE strategy demonstrates a Sharpe ratio slightly higher than the benchmark, indicating efficient risk-adjusted returns. However, the strategy's cumulative return and CAGR% are lower than the benchmark's, suggesting that while the strategy is less risky, it may not capture as much growth as the market overall. The strategy's max drawdown is lower than the benchmark, which could appeal to risk-averse investors.

6.4.3 HSIC

Metric	Strategy	Benchmark
Cumulative Return	-5.36%	-3.96%
CAGR%	-3.57%	-2.63%
Sharpe	-0.09	0.02
Max Drawdown	-23.69%	-31.99%
Win Days	48.0%	54.21%

Table 16: Trading Analysis for HSIC

The HSIC strategy underperforms with a negative Sharpe ratio, indicating a loss when risk is considered. Its negative cumulative return and CAGR% further accentuate its underperformance. The strategy's max drawdown is also high, albeit lower than the benchmark, suggesting that despite a more protective stance, the strategy fails to generate positive returns. The win days percentage is below that of the benchmark, which aligns with the overall negative performance.

6.4.4 KORS

Metric	Strategy	Benchmark
Cumulative Return	27.27%	51.41%
CAGR%	17.25%	31.49%
Sharpe	0.81	0.93
Max Drawdown	-20.28%	-25.34%
Win Days	52.77%	53.44%

Table 17: Trading Analysis for KORS

The KORS strategy showed a Sharpe ratio lower than the benchmark, reflecting less favorable risk-adjusted performance. Despite a significant cumulative return, the strategy fell short of the benchmark's growth, evidenced by a lower CAGR%. The max drawdown indicates that the strategy carries substantial risk, although it has managed to limit losses better than the benchmark.

6.4.5 MNST

For MNST, the strategy's Sharpe ratio is below the benchmark, indicating a less efficient risk-return tradeoff. The strategy's max drawdown is lower than the benchmark's, suggesting a relatively conservative approach that mitigates large losses. However, the win days and cumulative

Metric	Strategy	Benchmark
Cumulative Return	8.4%	14.8%
CAGR%	5.47%	9.54%
Sharpe	0.44	0.5
Max Drawdown	-21.36%	-30.21%
Win Days	52.63%	54.07%

Table 18: Trading Analysis for MNST

return are also lower than the benchmark, which could deter investors seeking higher growth potential.

6.4.6 PG

Metric	Strategy	Benchmark
Cumulative Return	-7.51%	4.73%
CAGR%	-5.02%	3.1%
Sharpe	-0.39	0.27
Max Drawdown	-22.92%	-23.01%
Win Days	46.64%	52.24%

Table 19: Trading Analysis for PG

PG's strategy reveals a negative Sharpe ratio, suggesting it has underperformed even after accounting for risk. The negative cumulative return and CAGR% reflect the strategy's inability to capitalize on market movements effectively. The max drawdown is comparable to the benchmark, but the lower win days percentage highlights the strategy's challenges in achieving consistent gains.

6.4.7 PPL

Metric	Strategy	Benchmark
Cumulative Return	-20.3%	-12.88%
CAGR%	-13.91%	-8.7%
Sharpe	-0.43	-0.16
Max Drawdown	-26.79%	-33.84%
Win Days	47.55%	54.76%

Table 20: Trading Analysis for PPL

The PPL strategy demonstrates a negative cumulative return, which underperforms the benchmark, and a negative Sharpe ratio indicating poor risk-adjusted returns. Its max drawdown is lower than the benchmark, suggesting some resilience to market downturns. However, the win days percentage being below the benchmark shows the strategy may struggle to consistently capture gains.

6.4.8 PRU

Metric	Strategy	Benchmark
Cumulative Return	-0.96%	-8.47%
CAGR%	-0.64%	-5.68%
Sharpe	0.0	-0.15
Max Drawdown	-12.61%	-26.02%
Win Days	49.19%	51.7%

Table 21: Trading Analysis for PRU

The PRU strategy, while showing a minimal negative cumulative return, outperforms the benchmark. The Sharpe ratio of 0.0 suggests the strategy's returns are just in line with the risk-free rate, with no additional risk-adjusted returns. The max drawdown is significantly better than the benchmark, and the strategy experiences fewer losses. Win days close to the benchmark indicate a competitive performance in achieving profitable days.

6.4.9 TSS

Metric	Strategy	Benchmark
Cumulative Return	44.7%	76.29%
CAGR%	27.62%	45.38%
Sharpe	2.23	2.0
Max Drawdown	-7.17%	-9.39%
Win Days	55.6%	58.79%

Table 22: Trading Analysis for TSS

The TSS strategy presents a robust Sharpe ratio, indicating strong risk-adjusted returns, outperforming the benchmark. Despite a lower cumulative return compared to the benchmark, the strategy maintains a high CAGR%, reflecting substantial growth. Moreover, it showcases a significantly lower max drawdown, suggesting an effective risk management strategy. The win days percentage is slightly lower than the benchmark but still reflects a competitive edge in profitable trading.

Metric	Strategy	Benchmark
Cumulative Return	7.08%	55.71%
CAGR%	4.62%	33.95%
Sharpe	0.46	1.65
Max Drawdown	-14.53%	-11.31%
Win Days	54.96%	59.26%

Table 23: Trading Analysis for V

6.4.10 V

The strategy for V stock displays a lower Sharpe ratio than the benchmark, suggesting less favorable risk-adjusted performance. The cumulative return and CAGR% are also lower than the benchmark, pointing to the strategy's limited growth potential in the given period. However, the max drawdown is only slightly worse than the benchmark, and the win days percentage is relatively close to the benchmark's, indicating the strategy's potential for a consistent approach to gains.

6.4.11 Conclusion

From the above results, we can find that when examining a larger variation of stocks, there is no guarantee that we can form the strategy whose performance surpasses the benchmark, which is different from the conclusion we draw for CNP and DLTR when we apply machine learning models on a small variation of stocks. Whether the machine learning methods work requires further examination.

7 Model Performance and Trading Considerations

7.1 Model Accuracy and Trading Readiness

1. **High Accuracy and Trading Implications:** The LightGBM and CatBoost models achieved relatively high accuracy compared with benchmark, with LightGBM at 0.5072 and CatBoost at 0.5082. While this is promising, initiating trading solely based on these results requires caution. High accuracy in a controlled test environment does not always translate directly to real-world market conditions. Factors like market volatility, economic changes, and unseen data patterns might affect performance. So we cannot start trading on it.

2. **Making Analysis More Realistic:** To enhance the realism of the analysis:

   - Incorporate different market conditions and scenarios in the testing phase.
   - Conduct thorough backtesting using historical data to simulate actual trading environments.
   - Continuously update and retrain the models with new data to adapt to changing market dynamics.

7.2 Overfitting Concerns and Mitigation

1. **Overfitting Assessment:** Given the complexity of financial data, there is a risk that the models might be overfitted to the training data. Overfitting occurs when a model captures noise instead of the underlying pattern, leading to poor performance on new, unseen data.

2. **Mitigation Strategies:**

   - **Cross-Validation:** We used cross-validation techniques to evaluate the model's performance on different subsets of the data, reducing the likelihood of overfitting.
   - **Regularization:** Hyperparameters related to regularization (like `reg_lambda` and `reg_alpha` in XGBoost) were tuned to prevent overfitting by penalizing overly complex models.
   - **Feature Selection:** Careful selection and engineering of features helped in avoiding over-reliance on irrelevant or noisy data.
   - **Bagging Method:** We applied the bagging method to train the model multiple times on different subsets of the data. The final prediction is then an aggregate of these models' predictions. This approach helps reduce the likelihood of overfitting by introducing diversity in the training process and reducing the model's sensitivity to specific features of the training data.

The high accuracy of our models indicates potential for stock price prediction, but careful consideration and additional testing are needed before deploying these models in a trading environment. Measures have been taken to mitigate overfitting, enhancing the models' robustness and reliability for real-world application.

8 Appendix

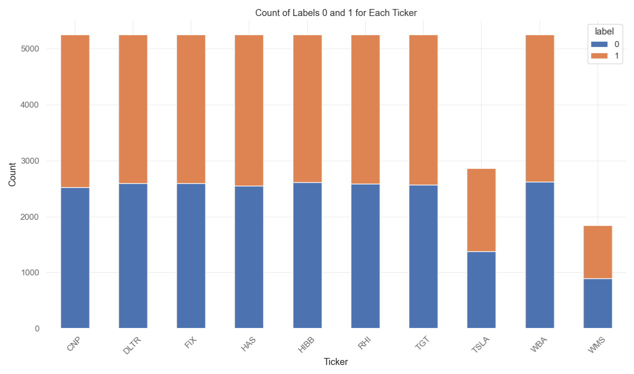

Figure 1: The number of labels 0 and 1 for different stocks

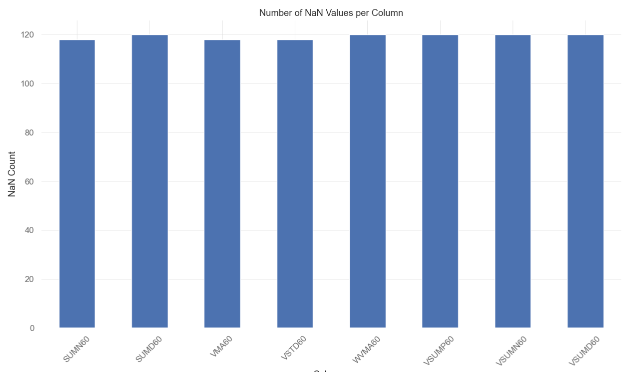

Figure 2: The number of nan values for selected features

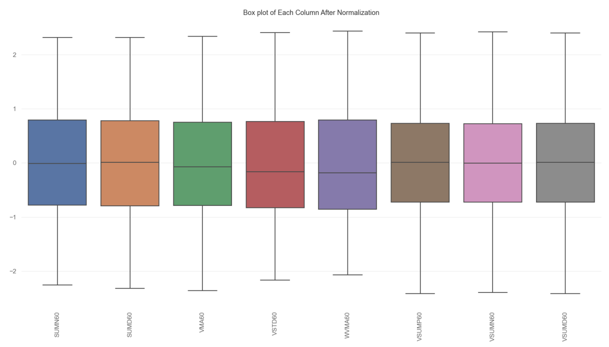

Figure 3: The box plot of selected feature after normalization

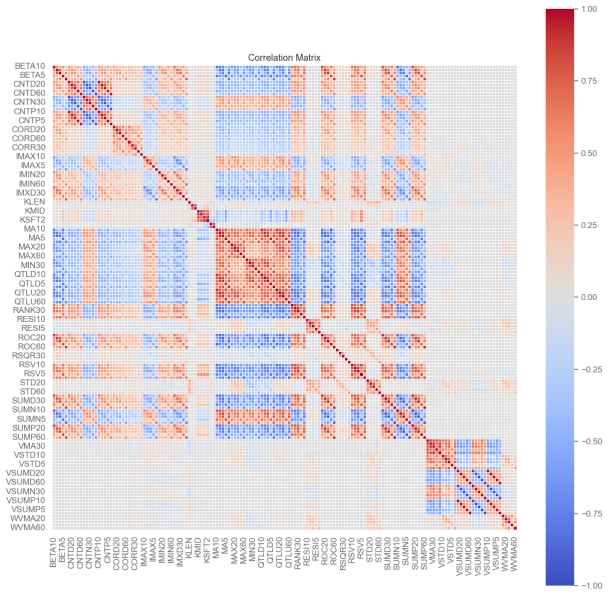

Figure 4: The correlation matrix for all features

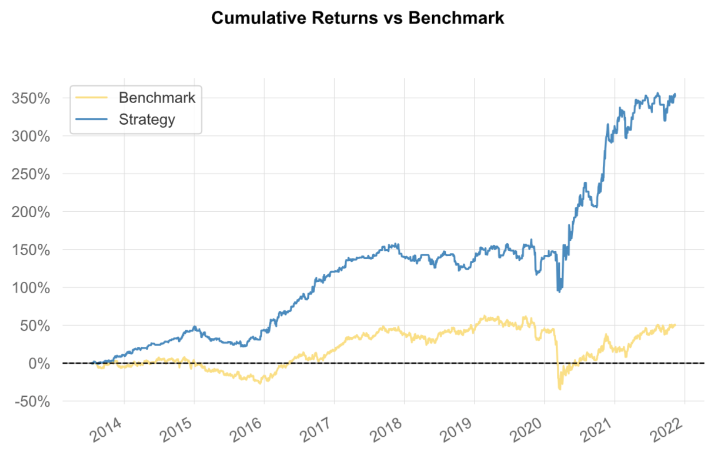

Figure 5: Cumulative returns for strategy by LightGBM on stock CNP

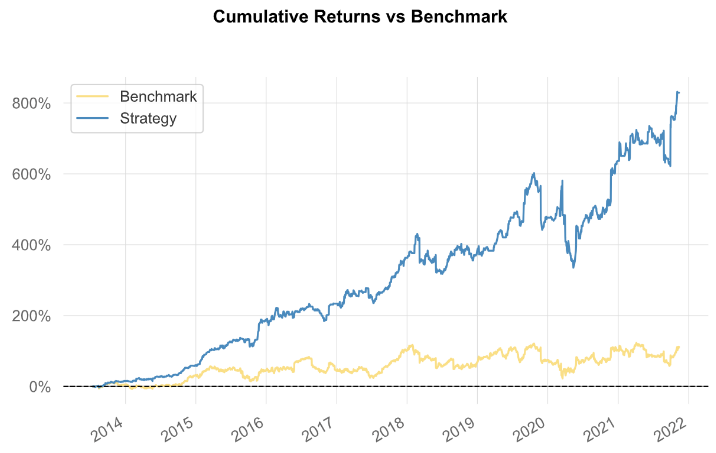

Figure 6: Cumulative returns for strategy by LightGBM on stock DLTR

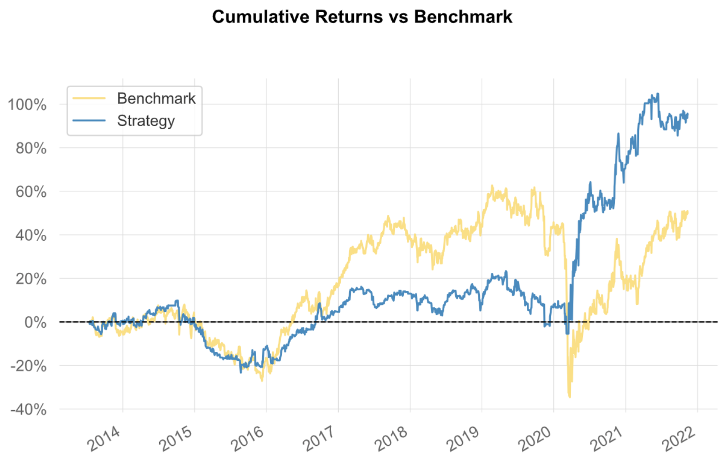

Figure 7: Cumulative returns for strategy by CatBoost on stock CNP

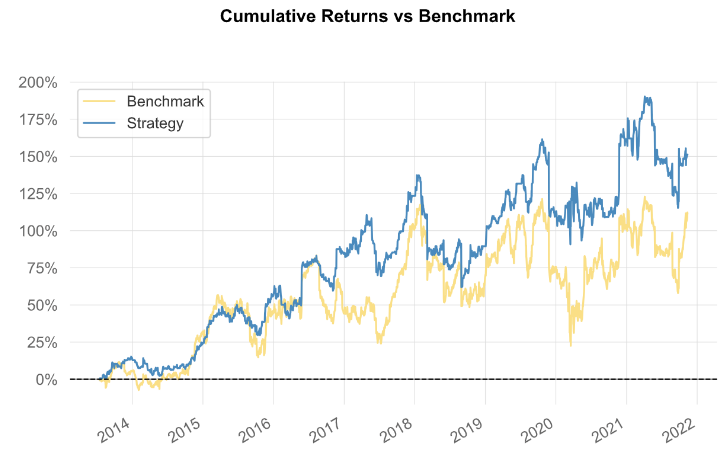

Figure 8: Cumulative returns for strategy by CatBoost on stock DLTR

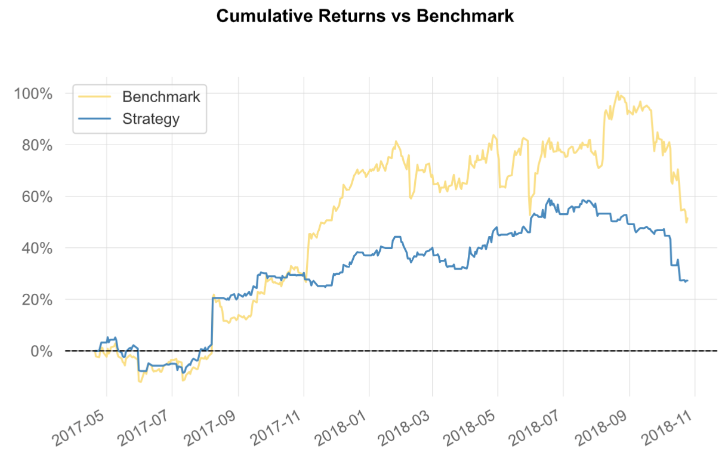

Figure 9: Cumulative returns for strategy on stock KORS

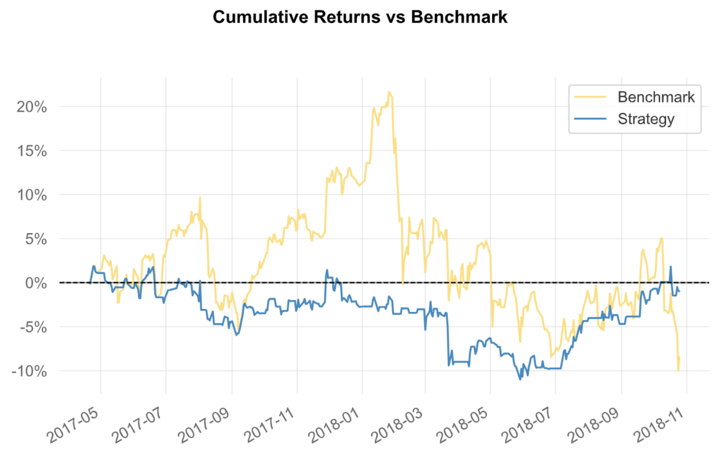

Figure 10: Cumulative returns for strategy on stock PRU

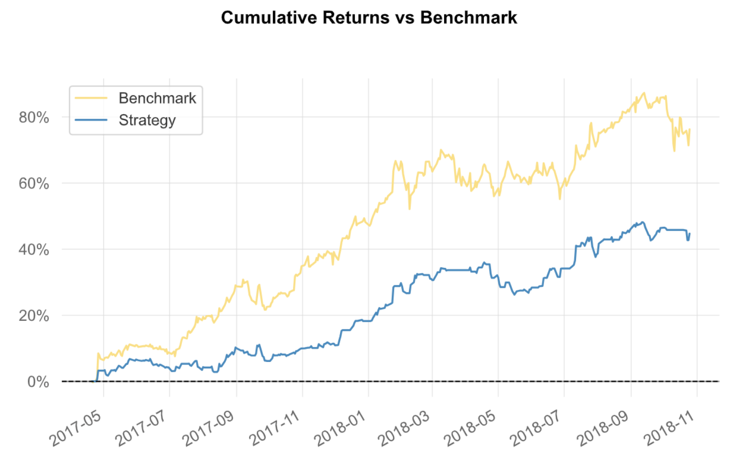

Figure 11: Cumulative returns for strategy on stock TSS

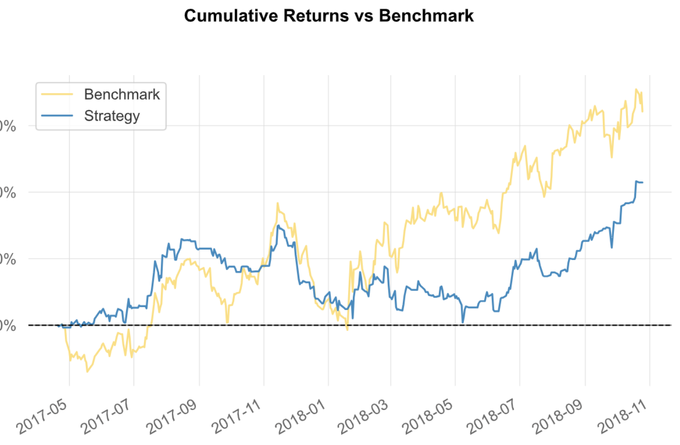

Figure 12: Cumulative returns for strategy on stock FE

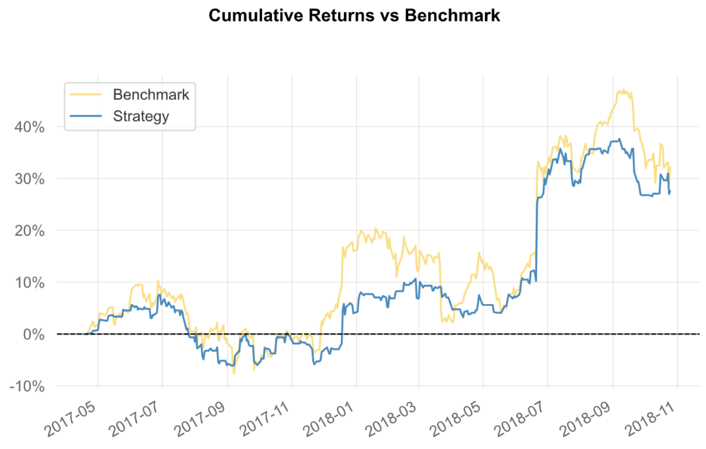

Figure 13: Cumulative returns for strategy on stock DRI

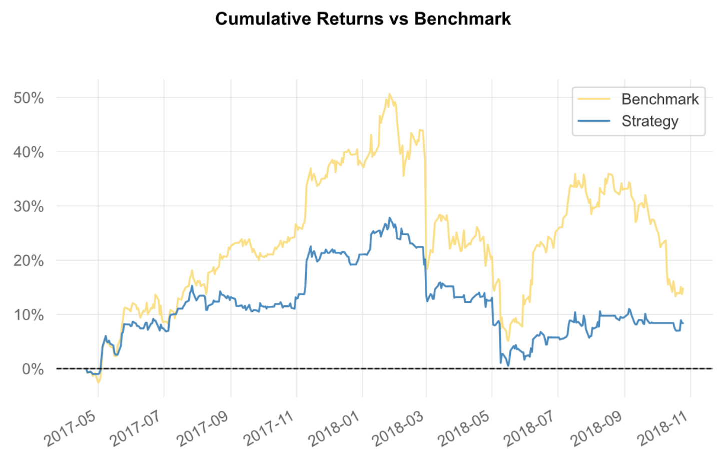

Figure 14: Cumulative returns for strategy on stock MNST

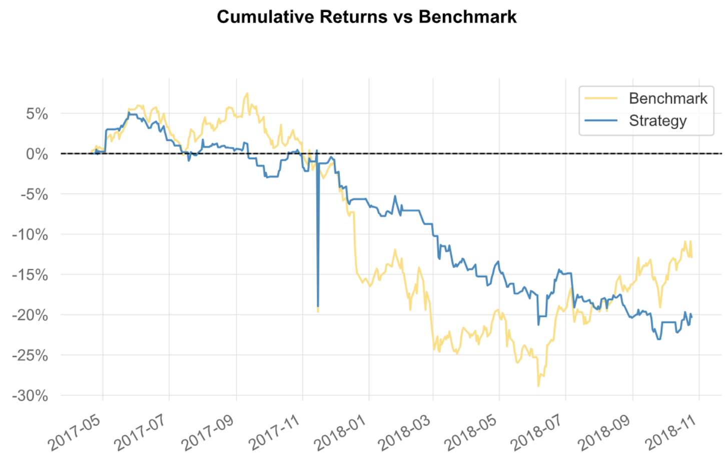

Figure 15: Cumulative returns for strategy on stock PPL

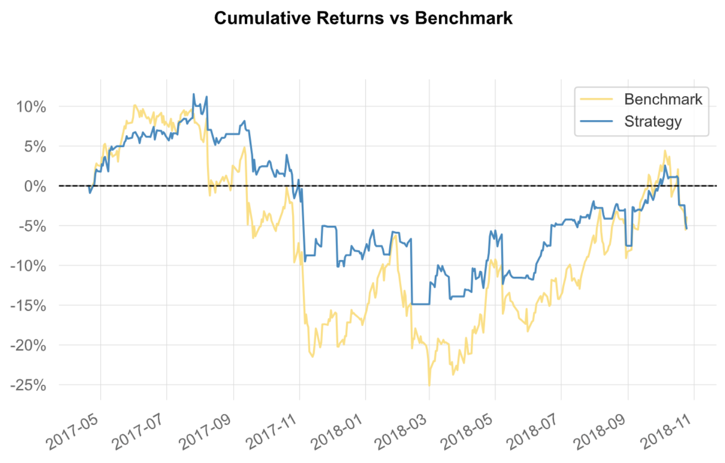

Figure 16: Cumulative returns for strategy on stock HSIC

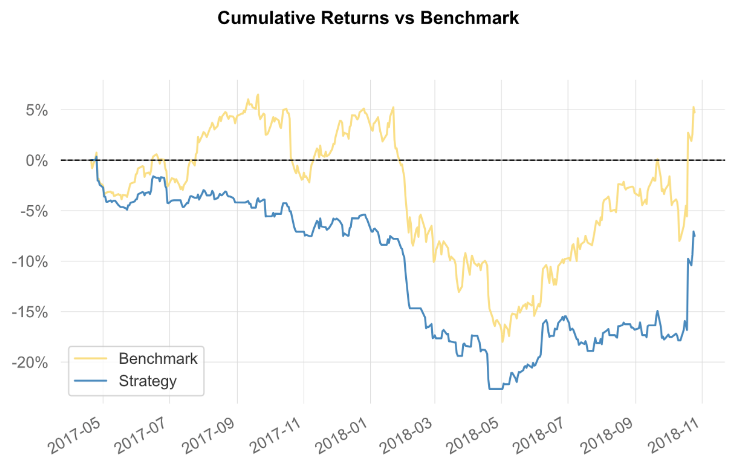

Figure 17: Cumulative returns for strategy on stock PG

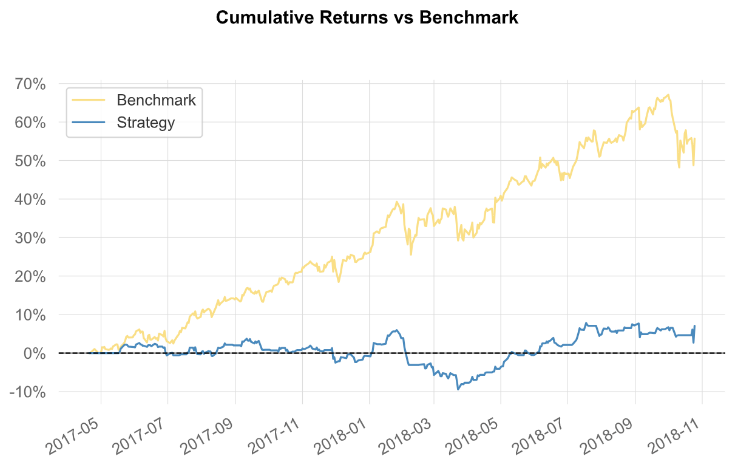

Figure 18: Cumulative returns for strategy on stock V