École Polytechnique Fédérale de Lausanne

Implementation of Trading Strategies Based on Clustered Periods

Financial Big Data (FIN-525)

Student :
Youssef Belghmi
Hamza Morchid
Milos Novakovic

Teacher :
Damien Challet

January 26, 2025

Contents

1 Introduction

1.1 Abstract

Financial markets exhibit dynamic and complex behaviors, particularly at high frequencies where volatility, volume, and spread significantly influence trading decisions. This project aims to design and evaluate trading strategies tailored to different market conditions by leveraging data clustering techniques. By segmenting intraday trading periods into homogeneous groups based on their characteristics, the project seeks to optimize decision-making for each type of period.

The methodology involves analyzing key market dynamics, applying clustering algorithms to identify similar trading periods, and developing strategies aligned with the unique features of each cluster. This process encompasses data preparation, statistical modeling, clustering, and the implementation of tailored trading strategies.

This study demonstrates the potential of clustering-based strategies to significantly improve trading outcomes by capturing stable and repeatable market dynamics. It also highlights the value of integrating statistical and machine learning techniques into financial analysis, providing a scalable framework for optimizing trading decisions in complex and volatile environments.

1.2 Context

This project is part of the Financial Big Data course at EPFL, taught by Professor Damien Challet. The course focuses on acquiring, cleaning, and analyzing financial data, and applying these techniques to intraday data and investment strategies.

This report constitutes the written component of our project, providing a summary of the methodology, analysis, and results. The implementation details are available in the notebooks hosted on this repository: https://github.com/youssefbelghmi/Financial-Big-Data. Additionally, some key functions utilized in the notebooks are included in the Appendix section of this report.

2 Data Preparation and Exploration

2.1 Dataset Overview

This study examines intraday stock market data at the minute level of companies listed in the S&P 500 during 2010, offering a detailed view of short-term market dynamics. Leveraging its high temporal resolution, we aim to cluster market states, capture trends, and analyze subtle variations in market behavior.

The S&P 500, a leading index tracking 500 large U.S. companies across sectors like technology, healthcare, and finance, provides broad economic representation, making it ideal for industry-wide market analysis.

Our dataset, provided by our professor, is organized into Parquet files, each representing one trading day for a specific ticker. Spanning all trading days in 2010, these files offer comprehensive minute-by-minute stock observations, enabling precise analysis.

To better understand the structure of the dataset, let's examine an excerpt of the Parquet file for Agilent Technologies (Ticker: A) on the first trading day of the year :

	index	X.Open	X.High	X.Low	X.Close
0	2010-01-04 09:31:00-05:00	30.96	7.0	31.37	3.0
1	2010-01-04 09:32:00-05:00	31.29	19.0	31.18	2.0
2	2010-01-04 09:33:00-05:00	31.32	13.0	31.26	4.0
3	2010-01-04 09:34:00-05:00	31.23	11.0	31.22	3.0
4	2010-01-04 09:35:00-05:00	31.22	16.0	31.25	11.0
..	...	...	...	...	...

Figure 1: Parquet File of Intraday Stock Data for Agilent Technologies (Ticker: A)

Each row captures a stock's behavior at a specific minute, allowing precise tracking of intraday fluctuations and providing a detailed view of performance. While the column names may appear unclear at first, the dataset adheres to the Best Bid and Offer (BBO) format, which records the highest bid price and volume alongside the lowest ask price and volume. This structure is especially useful for analyzing market liquidity by offering detailed insights into supply and demand dynamics in real time.

Thus, this dataset is an excellent starting point for our project due to its suitability for analyzing short-term market dynamics and its relevance to clustering market states. It provides a strong foundation for understanding market behavior for several reasons:

- Granularity: The minute-level data provides an extremely detailed view of stock behavior, allowing us to explore short-term patterns and trends.
- Diversity: With 500 companies across various sectors, the dataset offers a rich and varied sample of market behaviors.
- Comprehensiveness: The inclusion of all trading days in 2010 ensures that our analysis captures seasonal effects and market anomalies.
- Relevance: As one of the most widely tracked indices, the S&P 500 is a cornerstone of financial analysis, making our findings highly applicable to real-world scenarios.

Although the dataset is already quite clean and rather well structured, some preparation is necessary to tailor it for our specific objective of clustering market states.

In the next section, we will outline the transformations and enhancements applied to the dataset to extract meaningful insights.

2.2 Feature Engineering

The dataset comprises minute-level intraday stock data for S&P 500 companies across all trading days in 2010, stored as individual Parquet files for each company and trading day, totaling 877.2 MB. To create continuous timelines, daily files were concatenated chronologically for each ticker, resulting in annual datasets per company, which serve as the foundation for analyzing both short-term fluctuations and long-term trends.

The initial column names in the dataset were somewhat ambiguous and were therefore adjusted to ensure consistency with financial terminology. Since the data adhered to the Best Bid and Offer (BBO) format, the columns X.Open, X.High, X.Low, and X.Close were renamed bid_price, bid_volume, ask_price, and ask_volume, respectively. This renaming aligns the column names with their actual financial interpretations.

The dataset initially included an index column that stored the timestamp of each observation. For greater clarity, we split this column into two separate components:

- **date:** Captures the specific trading day (e.g., 2010-01-04).

- **time:** Reflects the minute of the observation, formatted in Eastern Standard Time (UTC-5:00) to align with the trading hours of the New York Stock Exchange.

This transformation facilitates temporal analysis by separating the trading day from the time of observation.

While the dataset was relatively clean, certain preprocessing steps were necessary to ensure its quality:

- Handling Missing Values: Rows with missing data were removed to avoid introducing inaccuracies in subsequent analyses.
- Removing Invalid Rows: Rows where the bid_price exceeded the ask_price were eliminated, as such occurrences violate financial market principles.
- Focusing on Market Hours: Only rows from regular trading hours (09:30–16:00 EST) were retained to reduce noise from low-activity off-hour trading.

These cleaning steps were essential for creating a robust and reliable dataset for analysis.

To enrich the dataset and capture additional insights, we added several new features, selected based on their relevance to financial analysis and their ability to improve our understanding of market dynamics:

- **mid_price:** The average of the bid_price and ask_price. This feature provides a central reference point for market prices, smoothing out bid-ask spreads.
- **order_density:** Reflects the total volume of bids and asks, indicating market liquidity at a given moment.
- **order_imbalance:** The difference between bid_volume and ask_volume. This feature helps identify supply and demand imbalances in the market.
- **spread:** The difference between ask_price and bid_price. This metric quantifies market inefficiency and liquidity costs.
- **vw_spread:** A volume-weighted version of the spread, offering a refined perspective on liquidity and price movement.
- **relative_spread:** The spread normalized by the mid_price, providing a scale-invariant measure of liquidity.
- **log_return:** The logarithmic return between consecutive mid_price values, capturing the percentage change in prices. This is a critical metric for analyzing stock volatility and trends.
- **volatility:** Measures the standard deviation of log_return over a specified window of 10 observations, offering insights into market risk and uncertainty.

These features enrich the dataset, enabling detailed analysis of price trends, liquidity, and volatility, key inputs for clustering market states.

The table below displays the first five rows of the finalized dataset for Agilent Technologies (Ticker: A), highlighting the engineered features that capture market behavior.

date	time	bid_price	bid_volume	ask_price	ask_volume	mid_price	order_density	order_imbalance	spread	vw_spread	relative_spread	log_return	volatility
2010-01-04	09:31:00-05:00	30.96	7.0	31.37	3.0	31.165	10.0	0.4	0.41000000000000014	4.100000000000001	0.013155783731750366		
2010-01-04	09:35:00-05:00	31.22	16.0	31.25	11.0	31.235	27.0	0.18518518518518517	0.030000000000001137	0.8100000000000307	0.0009604610212902557	0.0022435906847192272	
2010-01-04	09:37:00-05:00	31.3	24.0	31.32	4.0	31.310000000000002	28.0	0.7142857142857143	0.019999999999999574	0.5599999999999881	0.0006387735547748187	0.0023982743927812565	0.00010937789897411
2010-01-04	09:39:00-05:00	31.28	13.0	31.29	1.0	31.285	14.0	0.8571428571428571	0.009999999999999801	0.13999999999997215	0.0003196420009588624	-0.000798785887987612	0.0018028300298218 7
2010-01-04	09:41:00-05:00	31.27	19.0	31.28	6.0	31.275	25.0	0.52	0.010000000000001563	0.2500000000000391	0.0003197442046363 41	-0.000319693097352067	0.0016755217502183077

Figure 2: Finalized Dataset for Agilent Technologies (Ticker: A)

The table presents a minute-level snapshot for Agilent Technologies, with corresponding key features like bid_price, ask_price, and derived metrics that capture price trends, liquidity, and market fluctuations, providing a detailed view of stock behavior.

Before advancing to clustering and modeling, the next section evaluates the dataset's quality, structure, and completeness to ensure readiness for analysis.

2.3 Data Observation

The dataset contains minute-level intraday stock data for 401 S&P 500 companies in 2010, with data missing for 99 companies. Despite this, the remaining data provides sufficient coverage for financial analysis.

Each stock averages 69,546 observations, slightly below the expected 99,450 based on U.S. trading hours (390 minutes/day) and 252 trading days. The shortfall likely arises from market holidays, data gaps, or low trading activity. Daily observations average 283.68 per stock, indicating minor gaps that do not compromise the dataset's reliability.

To visualize temporal coverage, a heatmap was created with rows representing tickers and columns representing trading weeks, where color intensity reflects the weekly number of observations per ticker.

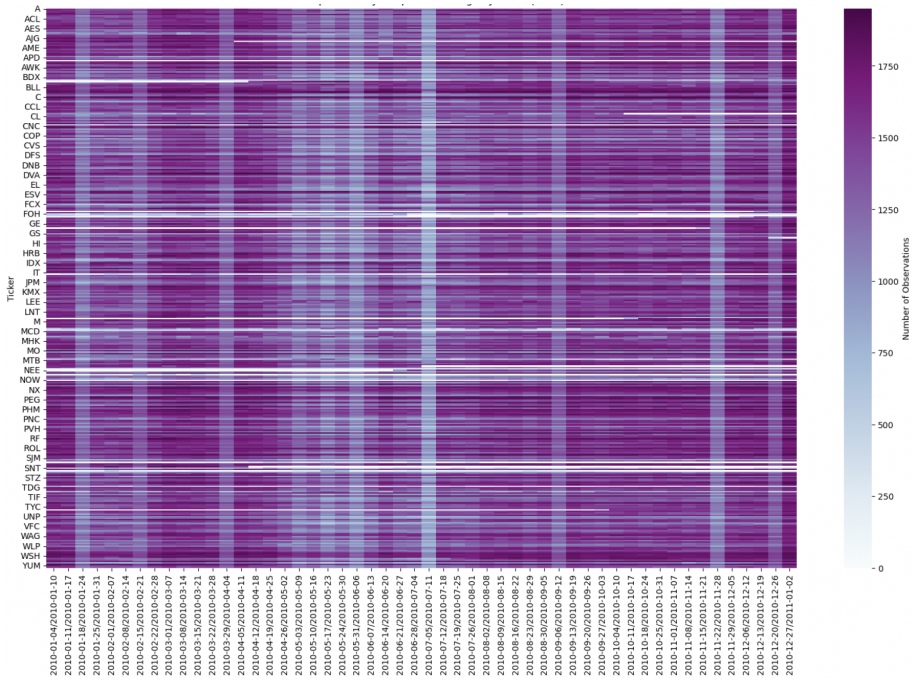

Figure 3: Heatmap of Weekly Temporal Coverage by Ticker during 2010

The heatmap reveals that most tickers maintain consistent and high coverage throughout the year, as indicated by the darker purple regions. However, some tickers exhibit

gaps (lighter or white areas) during certain weeks, likely due to market closures or low trading activity. Despite these gaps, the dataset remains robust and comprehensive for detailed financial analysis.

Next, we computed descriptive statistics for key features, including bid_price, ask_price, mid_price, order_density, spread, relative_spread, order_imbalance, log_return, and volatility. These metrics provide insights into the distribution, variability, and potential outliers present in the dataset.

	count	mean	std	min	25%	50%	75%	max
bid_price	27887970.0	44.960639	49.269904	0.000000	22.310000	36.250000	55.100000	766.560000
ask_price	27887970.0	45.217599	469.504029	0.000000	22.320000	36.260000	55.120000	1000000.000000
mid_price	27887970.0	45.089119	238.602542	0.000000	22.315000	36.250000	55.110000	500038.250000
order_density	27887970.0	225.011197	2813.533661	0.000000	10.000000	21.000000	56.000000	566066.000000
order_imbalance	27887840.0	0.337533	0.375481	-1.000000	0.090909	0.384615	0.636364	1.000000
spread	27887970.0	0.256960	466.903011	0.000000	0.010000	0.010000	0.010000	999990.370000
relative_spread	27887649.0	0.000676	0.004272	0.000000	0.000151	0.000304	0.000553	2.000000
log_return	27887163.0	0.000003	0.007092	-10.787321	-0.000172	0.000000	0.000290	10.726298
volatility	27887027.0	0.000793	0.007390	0.000000	0.000304	0.000518	0.000886	5.830378

Figure 4: Descriptive Statistics for Key Features Across the Entire Dataset

Our preliminary analysis highlights the dataset's quality, granularity, and relevance for financial modeling while acknowledging natural market variability and outliers:

- Bid Price, Ask Price, and Mid Price: Averages around 45 suggest stable pricing, though extreme values indicate possible outliers.
- Order Density: High variability reflects diverse trading activity across timestamps.
- Order Imbalance: A mean of 0.33 aligns with typical bid-ask volume imbalances.
- Spread and Relative Spread: Low values point to efficient pricing and high liquidity.
- Log Return : Well-centered around 0, indicating minimal average price changes.
- Volatility: Generally low, reflecting market stability with occasional fluctuations.

These observations confirm the dataset's reliability and suitability for modeling, despite outliers consistent with expected financial market behavior. To further analyze data distribution and detect outliers, we visualized key features using histograms and boxplots.

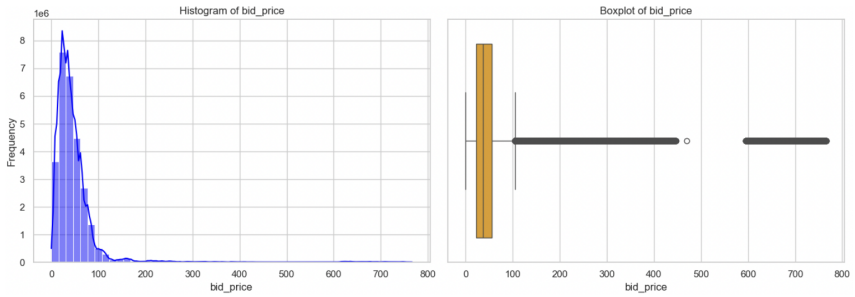

Figure 5: Histograms and Boxplots for Bid Price and Ask Price Distributions

The plots focus on bid_price and ask_price, showing skewed distributions and visible outliers, highlighted by long tails and extreme values in the boxplots. Results for other features are available in the observations.ipynb notebook.

Although these outliers reflect the dataset's variability, interpreting them requires caution as feature values depend on tickers. Uniform treatment may lead to misleading results, so we assess outliers within each ticker's context to preserve meaningful patterns.

2.4 Outlier Treatment

We used a Quantile Range (QR)-Based Outlier Detection method to identify and handle outliers. This approach leverages the interquartile range (IQR) to define bounds: $[Q1 - 1.5 \times \text{IQR}, Q3 + 1.5 \times \text{IQR}]$. To improve the robustness of this method and reduce false positives, we introduced a tolerance ratio T which extends the bounds as follows:

$$\text{Lower Bound} = Q1 - (1.5 + T) \times \text{IQR}$$

$$\text{Upper Bound} = Q3 + (1.5 + T) \times \text{IQR}$$

We set $T = 1.0$, to extend the IQR bounds, providing flexibility to retain data points that might otherwise be flagged as outliers due to minor deviations.

The method focused on key features, bid_price and ask_price, as their accuracy is critical to the validity of derived metrics. Features such as order_density were excluded, as they reflect trading dynamics rather than intrinsic stock properties, preserving insights into market behavior. The QR-Based Detection effectively flagged extreme values such as 0.0, 0.01, and 1,000,000.0 as anomalies but also mislabeled some realistic values near normal ranges as outliers.

Recognizing this limitation, we opted for a simpler cleaning process that focused only on removing rows where bid_price or ask_price displayed evidently erroneous values, such as 0.0, 0.01, or 1,000,000.0. This approach retained the dataset's inherent variability while eliminating the most obvious inaccuracies, achieving a balance between analytical richness and reliability.

	Ticker	Feature	Outliers Count	Outliers Percentage (%)	Outliers
0	AAP	bid_price	3	0.0	0.0
1	AAP	ask_price	3	0.0	0.0
2	ACN	bid_price	1	0.0	0.01
3	AIG	bid_price	2	0.0	0.0
4	AIG	ask_price	2	0.0	0.0
...	...	...	...	...	...

Figure 6: Summary of Outliers Removed Using Simplified Cleaning Method

A total of 686 outliers, anomalous data points which deviate from typical market behavior, were identified and removed from the dataset. The cleaned dataset is now well-prepared, offering a reliable foundation for subsequent analysis, and clustering tasks.

2.5 Prepared Dataset

We combined all 2010 intraday stock data into a single DataFrame, adding a ticker column to identify each company. This unified structure simplifies analysis and provides a solid foundation for further processing.

We also enriched the dataset with metadata from Wikipedia, including company names and industries based on the Global Industry Classification Standard (GICS), available in the List_S&P500_Compagnies.csv file, helping us better understand the dataset.

To highlight dataset diversity, we calculated average values for each feature by ticker, grouped them into bins (capped at 100+), and visualized the distributions with bar plots, all available in the observations.ipynb notebook.

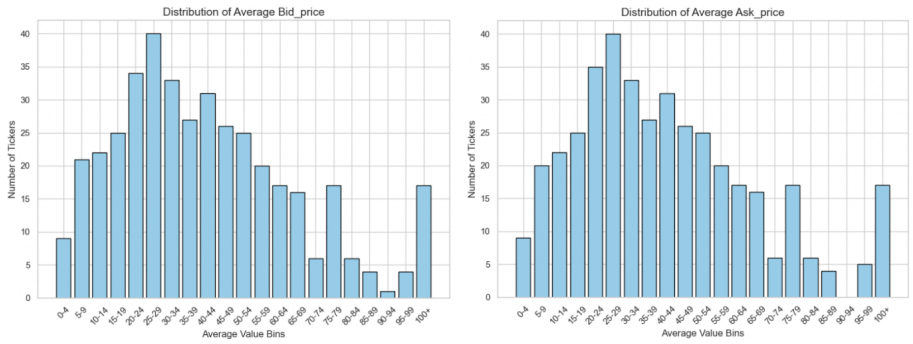

Figure 7: Distribution of Average Bid and Ask Prices Across Tickers

The distributions of average bid and ask prices illustrate the dataset's diversity and granularity. Most tickers cluster within specific price ranges, while others extend beyond 100, reflecting varying market valuations across companies. Trading volumes also show a concentration in lower ranges, with some tickers displaying significantly higher activity, emphasizing differences in market behavior and liquidity.

This level of detail provides a robust foundation for clustering market states and exploring financial trends. The dataset's granularity enables the study of short-term fluctuations, while its diversity supports broader analyses across industries and sectors. With this rich structure, we can effectively examine market dynamics and develop insights for advanced clustering tasks.

3 Market Clustering

3.1 Building the Log Return Matrix

To segment market periods into distinct states, we created a log-return matrix that captures short-term price movements across multiple stocks, forming the basis for correlation analysis and behavior pattern identification.

The dataset, containing minute-level observations for S&P 500 stocks in 2010, is aggregated into 5-minute intervals using the first available mid-price as a reference. This approach balances temporal resolution and data stability, minimizing noise while preserving meaningful patterns. Log returns are calculated as the difference in the logarithm of mid-prices between consecutive first available mid-price in each 5-minute windows.

To enhance data quality, we removed rows and columns with more than 75% missing values, because they do not provide enough information for meaningful analysis, reducing the average percentage of missing data from 6.52% to 3.85%. The final log-return matrix comprises rows representing 5-minute trading windows and columns corresponding to tickers, providing a solid basis for further analysis. Part of this matrix is shown below.

ticker	A	AA	AAP	ABC	ABD	ABT	ACE	ACL	ACN	ADI	...
timestamp_5min											
2010-01-04 09:35:00-05:00	0.002244	0.000305	0.003196	0.003240	0.011375	NaN	-0.005353	0.009064	0.007080	0.001886	...
2010-01-04 09:40:00-05:00	0.001280	0.002130	0.000859	0.004746	0.005970	0.004071	0.001192	NaN	0.002627	-0.000157	...
2010-01-04 09:45:00-05:00	-0.000160	0.000911	0.000613	0.002081	0.006592	0.002122	-0.005076	-0.002188	-0.000477	0.007044	...
2010-01-04 09:50:00-05:00	-0.000480	0.001820	0.000857	0.001511	-0.013228	0.001473	0.000898	0.000938	-0.000358	0.000312	...
2010-01-04 09:55:00-05:00	-0.001922	-0.001517	0.000122	-0.000566	-0.001332	-0.001381	-0.000998	-0.005926	-0.000836	-0.003436	...
...	...	...	...	...	...	...	...	...	...	...	...

Figure 8: Sample Log-Return Matrix with 5-Minute Intervals

With this refined matrix, we are now ready to calculate correlation matrices and apply clustering techniques to identify distinct market states and analyze their dynamics.

3.2 Louvain-Based Sliding Window Approach

To achieve this, we employed the Louvain community detection algorithm, well-suited for financial data where stock relationships evolve dynamically. This algorithm optimizes modularity Q, to identify coherent groups of time-periods, defined as:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

where A_{ij} is the edge weight, k_i and k_j are node degrees, m is the sum of edge weights, and $\delta(c_i, c_j)$ indicates cluster membership. Applied to the correlation matrix of the transposed log-return matrix, it identifies strongly correlated, distinct clusters of time-periods.

To capture temporal evolution, we adopted a sliding window approach, applying clustering to subsets of data to effectively identify dynamic market behaviors. Key aspects of this approach include:

- **Window Size:** Each window spans 100 rows of the log-return matrix. This size balances capturing short-term fluctuations with minimizing noise.
- **Non-Overlapping Windows:** Windows are treated independently to simplify analysis and ensure clear segmentation across time periods.

For each window, a correlation matrix of log-returns was computed, followed by eigenvalue decomposition to filter noise, retaining only eigenvalues accounting for at least 3.5% of the total variance. The resulting filtered matrix, emphasizing meaningful co-movements, was transformed into a weighted graph with nodes as time-periods and edges representing correlation strengths.

The Louvain algorithm was applied to these graphs, clustering time-periods into market states where they exhibit similar log-return behaviors. The algorithm's modularity optimization ensures internally cohesive clusters that remain distinct from one another.

3.3 Clustering Results

The Louvain-based sliding window clustering approach produced valuable insights into market segmentation. The figure below illustrates the number of clusters detected for each window throughout 2010.

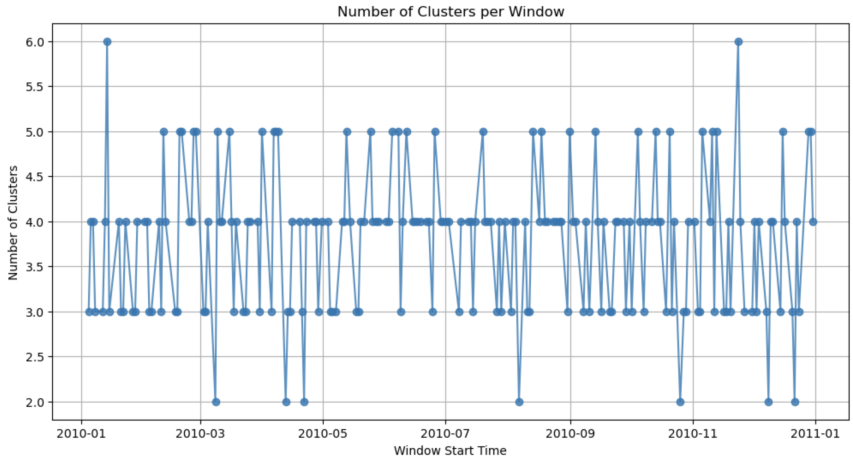

Figure 9: Number of Clusters per Time Window

On average, the analysis identified 3.78 clusters per window, indicating that, within each trading window, the market can be segmented into distinct groups of time periods with strongly correlated log-return behaviors.

To further assess the stationarity of market states, the Adjusted Rand Index (ARI) was computed to evaluate the similarity between clustering results in consecutive windows. The ARI distribution, presented in Figure 10, is concentrated around 0, indicating minimal similarity between successive clustering windows. This lack of consistency suggests that the market states are non-stationary, as cluster structures change dynamically over time.

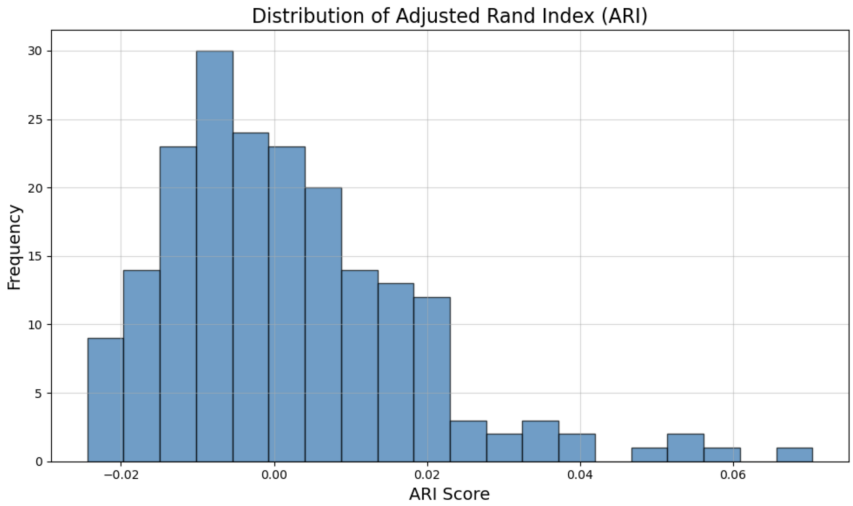

Figure 10: Distribution of Adjusted Rand Index (ARI) Across Clustering Windows

The results validate the effectiveness of the Louvain-based sliding window method in capturing meaningful market states by reflecting the non-stationarity of market behavior. The low ARI scores highlight the method's ability to detect dynamic changes and adapt to evolving market conditions over time.

4 Implementation of Trading Strategies

This section outlines three trading strategies based on market state clustering. Leveraging the behaviors identified using the Louvain algorithm, these strategies inform portfolio decisions and progressively integrate methodologies to enhance performance. The following subsections detail each strategy.

4.1 Market State-Based Strategy

The market state-based strategy seeks to identify recurring market conditions and leverage them to guide portfolio allocation. By analyzing historical patterns in stock behavior, it aims to predict future asset performance and generate actionable trading signals. The approach operates within predefined time windows, systematically examining market states and utilizing historical data from similar conditions to inform dynamic decision-making.

1. **Market State Identification:** The market state at the last timestamp of each window is determined based on the final cluster ($S_t = C_{\text{last}}$).
2. **Historical Data Retrieval:** Historical data from the same market state is used to isolate periods of similar behavior.
3. **Performance Computation:** The average log returns ($P_i = \text{Average}(R_{i,t+1})$) are calculated to predict asset performance.
4. **Portfolio Assignment:** Positions are assigned based on P_i: long ($P_i > 0$), short ($P_i < 0$), or hold ($P_i = 0$).

By repeating this process for each time window, the strategy generates dynamic and adaptive trading signals aligned with changing market conditions.

Figure 11: Cumulative Returns of Market State-Based Strategy over 2010

When applied to a log-return matrix, the Market State-Based Strategy was benchmarked against a Buy-and-Hold portfolio, which assumes equally weighted positions across all assets over time. The results, illustrated in figure above, reveal that while cumulative returns of the strategy were modest, it effectively demonstrated the viability of clustering-based approaches for portfolio optimization.

4.2 Momentum-Weighted Market State Strategy

The Momentum-Weighted Market State Strategy enhances the Market State-Based Strategy by integrating momentum-based weighting, dynamically adjusting portfolio allocations according to assets' historical performance within specific market states. This approach refines decision-making, prioritizing high-momentum assets to allocate capital more effectively. The process includes:

1. **Market State Identification:** The dominant cluster at the end of each window (S_t) is selected to guide portfolio adjustments.
2. **Performance Analysis:** Expected future performance (P_i) is calculated using historical returns from similar market states.
3. **Momentum Filtering:** Weights ($W_i = |P_i| / \sum |P_j|$), assign greater importance to high-momentum assets, ensuring that stocks with stronger momentum have a larger influence within the portfolio.
4. **Weighted Portfolio Construction:** Allocations ($D_i = W_i \cdot \text{Sign}(P_i)$) combine weights with directional signals.

This methodology provides a systematic framework to capitalize on historical market patterns, emphasizing stocks that have demonstrated consistent behavior under similar market states. By adjusting weights dynamically, the strategy improves its ability to respond to evolving market dynamics. The following figure compares the cumulative returns of the Momentum-Weighted Market State Strategy to those of the buy-and-hold portfolio over the evaluation period:

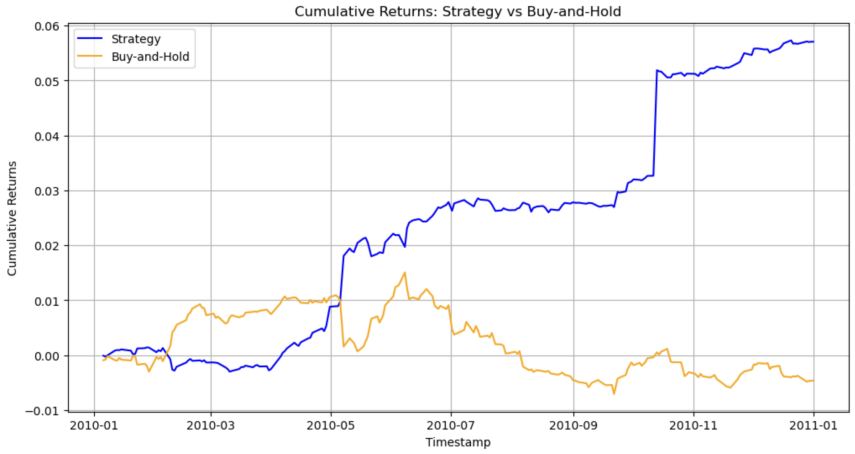

Figure 12: Cumulative Returns of Momentum-Weighted Strategy over 2010

The Momentum-Weighted Market State Strategy demonstrated a clear improvement over both the buy-and-hold benchmark and the previously implemented Market State-Based Strategy. By effectively enhancing portfolio performance, this approach underscores the importance of momentum filtering as a valuable addition to clustering-based trading strategies, paving the way for further refinements and optimizations in portfolio design.

4.3 Historical Performance-Weighted Strategy

The Historical Performance-Weighted Strategy combines clustering insights with historical performance to assign dynamic weights to assets based on their returns in similar market states. By emphasizing consistently high-performing stocks, it aims to enhance portfolio returns and achieve better risk-adjusted outcomes compared to simpler allocation methods. The strategy is as follows:

1. **Market State Identification:** As with prior strategies, $S_t = C_{\text{last}}$ defines the market state for each window.
2. **Historical Data Analysis:** For each identified market state S_t, the strategy analyzes all prior timestamps associated with the same state. Historical stock performance is then assessed using the log-return matrix for these periods.
3. **Weight Assignment:** Weights ($w_i = P_i / \sum |P_i|$) reflect normalized historical performance, ensuring diversification while emphasizing high-performing stocks.
4. **Portfolio Construction:** These weights are then used to construct a dynamically weighted portfolio. Stocks with higher weights receive more significant allocations, reflecting their historical performance.

The strategy's design incorporates a market-neutral approach by ensuring that the weights are balanced. This helps to mitigate the risk of overexposure to specific stocks or sectors. The Historical Performance-Weighted Strategy was then evaluated against the buy-and-hold benchmark, which assumes equal-weighted positions in all assets.

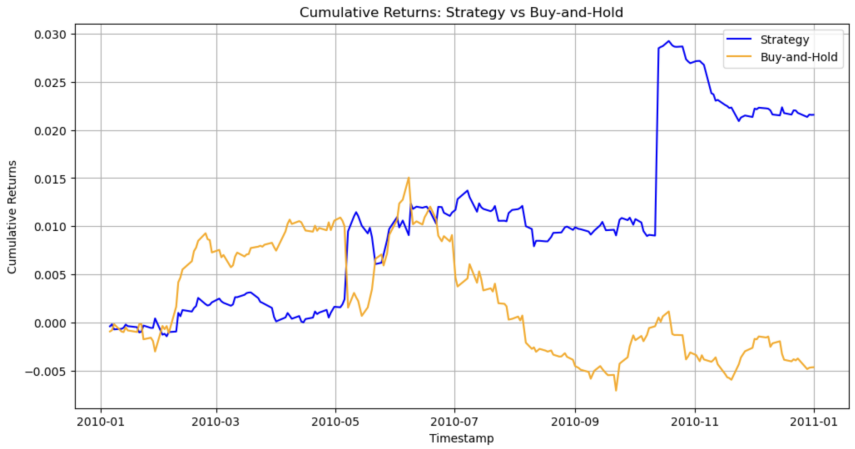

Figure 13: Cumulative Returns of Historical Performance-Weighted Strategy over 2010

The graph highlights the strategy's ability to deliver positive returns while maintaining stability, outperforming the buy-and-hold portfolio, which faced losses. However, the overall returns of the Historical Performance-Weighted Strategy remain moderate, indicating that while the strategy provides a more reliable performance than the buy-and-hold approach, its potential for higher profitability remains limited.

5 Conclusion

5.1 Analysis of Results

The results of the three strategies—Market State-Based, Momentum-Weighted Market State, and Historical Performance-Weighted—highlight varying levels of success in leveraging market states to optimize portfolio performance, as summarized in Figure 14.

	Market State Strat.	Momentum-Weighted Strat.	Performance-Weighted Strat.	Buy-and-Hold
Cumulative Return	0.68%	5.71%	2.16%	-0.46%
Sharpe Ratio	0.05	0.18	0.07	-0.02
Volatility	0.048	0.002	0.002	0.001

Figure 14: Summary of Performance Metrics for All Strategies Over 2010

The **Market State-Based Strategy** achieved a modest cumulative return of 0.68%, with a Sharpe ratio of 0.05 and high volatility (0.048). While it outperformed the buy-and-hold benchmark (-0.46%), its static approach and elevated risk limited its effectiveness.

The **Momentum-Weighted Strategy** performed best, achieving a 5.71% cumulative return, a Sharpe ratio of 0.18, and significantly reduced volatility (0.002). By dynamically adjusting weights using momentum filtering, this strategy prioritized high-performing stocks and ensured balanced allocations, outperforming all other strategies.

The **Historical Performance-Weighted Strategy** balanced stability and returns, achieving a 2.16% cumulative return with a Sharpe ratio of 0.07 and low volatility (0.002). However, its reliance on historical averages limited its adaptability and growth potential compared to the momentum-based approach.

These findings highlight the scalability and versatility of the proposed strategies in adapting to varying market conditions and portfolio objectives.

- The **Market State-Based Strategy** provides a simple baseline but lacks dynamic optimization, leading to modest returns and higher risk.
- The **Momentum-Weighted Strategy** emerges as the most effective, offering the highest returns and risk-adjusted performance through adaptive adjustments.
- The **Historical Performance-Weighted Strategy** offers stability but falls short in dynamic market adaptability.

Overall, the Momentum-Weighted Strategy stands out as the most robust, demonstrating the value of integrating clustering insights with adaptive mechanisms to optimize portfolio performance.

5.2 Final thoughts

This project explored trading strategies based on clustering-driven segmentation of market states to address challenges in analyzing high-frequency financial data. After ensuring data reliability, clustering algorithms were applied to identify recurring patterns, forming the basis for strategy development.

The strategies evolved iteratively. The Market State-Based Strategy served as a baseline, generating signals based on identified clusters. The Momentum-Weighted Strategy

prioritized high-performing assets using momentum, enhancing returns. The Historical Performance-Weighted Strategy leveraged historical averages for stability, offering refined portfolio allocations. Together, these strategies demonstrated the potential of clustering to create adaptive financial models.

Challenges included reliance on static clustering and past performance, limiting responsiveness to sudden market shifts. To improve adaptability, incorporating real-time data and predictive models, such as machine learning, is essential. Exploring Marsili-Giada clustering could provide a probabilistic alternative to Louvain clustering, potentially improving performance by better capturing nuanced patterns.

Additionally, computational demands due to the large dataset increased runtime and energy consumption. Optimizing algorithms with parallel processing, efficient data handling, or hardware acceleration could address this.

This project highlights the value of combining financial insights with data analytics. Future work should focus on real-time adaptability, broader datasets, novel optimization methods, and computational efficiency to refine strategies for dynamic market conditions.

6 Appendix

6.1 Appendix A: Data Processing Function

The following function is an excerpt of the preprocessing function originally implemented in the notebook `preprocessing.ipynb`. The full function, which is significantly longer, has been simplified here for clarity and conciseness.

This function processes Parquet files containing stock market data for S&P 500 companies. It cleans, standardizes, and structures the data by filtering trading hours, renaming columns, handling missing values, and calculating key metrics such as mid-price, spreads, and volatility. The resulting dataset is tailored for further financial analysis and modeling.

```
def preprocess_stocks_parquet_files(input_folder, output_folder, full_analysis,
    volatility_window=10):
    """
    Preprocess Parquet files for S&P500 tickers, including filtering, renaming, and
        feature calculation.

    Args:
        input_folder (str): Path to input Parquet files.
        output_folder (str): Path to save preprocessed CSV files.
        full_analysis (bool): Process all files if True; otherwise, only demo files.
        volatility_window (int): Window size for rolling volatility calculation.
    """
    os.makedirs(output_folder, exist_ok=True)
    tickers = sorted([d for d in os.listdir(input_folder) if
        os.path.isdir(os.path.join(input_folder, d))])

    for ticker in tickers:
        print(f"Processing ticker: {ticker}")
        parquet_files = [...]  # Filter Parquet files based on the input criteria
        if not parquet_files:
            print(f"No Parquet files for 2010 found for ticker: {ticker}")
            continue

        # Concatenate and preprocess Parquet files
        dataframes = [pd.read_parquet(file) for file in sorted(parquet_files)]
        full_dataframe = pd.concat(dataframes, ignore_index=True)

        # Extract and reformat 'date' and 'time'
        full_dataframe['index'] = pd.to_datetime(full_dataframe['index'],
            errors='coerce')
        full_dataframe['date'] = full_dataframe['index'].dt.date
        full_dataframe['time'] = full_dataframe['index'].dt.strftime('%H:%M:%S%z')
        full_dataframe = full_dataframe[
            (full_dataframe['time'] >= '09:30:00-05:00') & (full_dataframe['time'] <=
                '16:00:00-05:00')
        ]

        # Rename and calculate features
        full_dataframe.rename(
            columns={
                'X.Open': 'bid_price',
                'X.High': 'bid_volume',
                'X.Low': 'ask_price',
                'X.Close': 'ask_volume'
            },
            inplace=True
        )
        full_dataframe['mid_price'] = (full_dataframe['bid_price'] +
            full_dataframe['ask_price']) / 2
        full_dataframe['spread'] = full_dataframe['ask_price'] -
            full_dataframe['bid_price']
        full_dataframe['log_return'] = np.log(full_dataframe['mid_price'] /
            full_dataframe['mid_price'].shift(1))
        full_dataframe['volatility'] = full_dataframe['log_return'].rolling(
```

```
        window=volatility_window, min_periods=1).std()


        # Save preprocessed data
        full_dataframe.to_csv(os.path.join(output_folder, f"{ticker}_2010_cleaned.csv"),
            index=False)
        print(f"Preprocessed CSV saved for {ticker}.")
```

6.2 Appendix B: Clustering Function

The following function implements clustering on non-overlapping sliding windows within the log-return matrix. This function is also available in the `clustering.ipynb` notebook for reference.

```
def sliding_window_clustering_no_overlap(matrix, window_size):
    """
    Perform clustering on completely non-overlapping windows with minimal progress
        updates.

    Parameters:
    - matrix (pd.DataFrame): The log return matrix with a 'timestamp' column and stock
        returns as other columns.
    - window_size (int): The size of the window in minutes.

    Returns:
    - results (list of dict): A list where each element contains the clustering result
        for a window.
                              Each element is a dictionary with 'start_time', 'end_time',
                              'num_clusters', 'clusters_df', and 'ari_score' (compared
                                  to the previous window).
    """
    # Start by making sure the timestamp is in datetime format
    matrix['timestamp_5min'] = pd.to_datetime(matrix['timestamp_5min'], utc=True)
    results = []  # To store clustering results for each window
    num_rows = matrix.shape[0]
    total_windows = num_rows // window_size  # Total number of full windows

    print(f"Starting clustering for {total_windows} non-overlapping windows (window
        size: {window_size})...\n")
    previous_labels = None  # To store the cluster labels of the previous window

    for window_num, start in enumerate(range(0, num_rows, window_size), start=1):
        end = start + window_size
        if end > num_rows:  # Skip the last window if it doesn't fit the size
            break

        print(f"Processing window {window_num} / {total_windows}...")

        # Define the current window
        matrix_window = matrix.iloc[start:end]
        timestamps = matrix_window['timestamp_5min']
        matrix_window_numeric = matrix_window.drop(columns=['timestamp_5min'])

        # Compute the correlation matrix for time periods
        C = matrix_window_numeric.T.corr()
        eigvals, eigvecs = eig(C)

        # Retain significant eigenvalues
        threshold = 0.035 * sum(eigvals)
        filtered_eigvals = [val if val >= threshold else 0 for val in eigvals]

        # Reconstruct filtered correlation matrix
        C_filtered = eigvecs @ np.diag(filtered_eigvals) @ eigvecs.T

        # Convert to a graph and apply Louvain clustering
        C_graph = np.abs(C_filtered)
        graph = nx.from_numpy_array(C_graph)
```

```
        partition = community_louvain.best_partition(graph)
        num_clusters = len(set(partition.values()))

        # Map time periods to clusters
        clusters_df = pd.DataFrame({
            'Timestamp': timestamps,
            'Cluster': [partition[i] for i in range(len(matrix_window))]
        })

        # Compute ARI if this is not the first window
        current_labels = clusters_df['Cluster'].values
        ari_score = None
        if previous_labels is not None:
            ari_score = adjusted_rand_score(previous_labels, current_labels)

        # Store the result
        results.append({
            'start_time': timestamps.iloc[0],
            'end_time': timestamps.iloc[-1],
            'num_clusters': num_clusters,
            'clusters_df': clusters_df,
            'ari_score': ari_score  # ARI compared to the previous window
        })

        # Update the previous labels
        previous_labels = current_labels

    print("\nClustering completed for all windows!")

    return results
```

6.3 Appendix C: Market State-Based Strategy Function

The following function, extracted from the `first_strategy.ipynb` notebook, implements our initial market state-based strategy, which served as the foundation for the more advanced approach. This strategy leverages clustering results to analyze patterns in market states and make portfolio decisions.

```
def first_strategy(clustering_results, matrix, measure='average'):
    """
    Implement the Market State-Based Strategy.

    Parameters:
    - clustering_results (list of dict): Output from the sliding_window_clustering
        function.
    - matrix (pd.DataFrame): The log return matrix with a 'timestamp' column and stock
        returns as other columns.
    - measure (str): The performance measure to use ('average', 'median').

    Returns:
    - strategy_results (list of dict): Contains the market state and portfolio
        directions for each window.
    """
    strategy_results = []  # To store the results

    for result in clustering_results:
        # Extract clustering data for the current window
        clusters_df = result['clusters_df']
        start_time = result['start_time']
        end_time = result['end_time']

        # Determine the market state at t1 (last timestamp of the window)
        last_timestamp = clusters_df['Timestamp'].iloc[-1]
        market_state = clusters_df.loc[clusters_df['Timestamp'] == last_timestamp,
            'Cluster'].values[0]

```

```
        # Find all prior timestamps within [t0, t1 - 1] with the same market state
        previous_states = clusters_df[(clusters_df['Timestamp'] < last_timestamp) &
                                      (clusters_df['Cluster'] == market_state)]

        # Extract their indices from the log return matrix
        previous_indices = matrix[matrix['timestamp_5min'].isin(
        previous_states['Timestamp'])].index

        # Compute performance measures for the returns at t+1 for these timestamps
        if not previous_indices.empty:
            future_indices = previous_indices + 1  # Shift to t+1
            future_indices = future_indices[future_indices < len(matrix)]  # Ensure
                valid indices

            if measure == 'average':
                performance = matrix.iloc[future_indices, 1:].mean(axis=0)
            elif measure == 'median':
                performance = matrix.iloc[future_indices, 1:].median(axis=0)
            else:
                raise ValueError("Invalid performance measure")

            # Assign portfolio directions based on the performance measures
            directions = performance.apply(lambda x: '+' if x > 0 else ('-' if x < 0
                else '0'))
        else:
            # If no prior states exist, assign hold positions (no signal)
            directions = pd.Series('0', index=matrix.columns[1:])

        # Store results for this window
        strategy_results.append({
            'start_time': start_time,
            'end_time': end_time,
            'market_state': market_state,
            'directions': directions.to_dict()
        })

    return strategy_results
```