Professor
Damien Challet

Semester Project

High-Frequency S&P100 Analysis: Data Processing, Correlation Clustering and Statistical Pairs Trading

Group I

Timothé Dard	340944	timothe.dard@epfl.ch
Marius Pécaut	330684	marius.pecaut@epfl.ch
Cyprien Tordo	345618	cyprien.tordo@epfl.ch

Introduction

This project demonstrates the application of big data techniques to high-frequency financial markets by implementing a complete pipeline from raw limit order book data to an executable pairs trading strategy. We process 23.4 GB of nanosecond-level S&P100 data from the 2008 financial crisis, transforming it into minute-frequency panels while addressing computational efficiency, missing values, and estimation noise. The workflow comprises four phases: (1) data preprocessing using Polars, (2) data formatting and missing value treatment, (3) asset grouping using both data-driven and benchmark clustering methods to identify statistically and economically related stocks, and (4) implementation of a rolling-window intra-cluster pairs trading strategy.

Our submission contains a `README.md` with set-up and execution guidelines to reproduce our results.

Contents

1 Data Presentation and Preprocessing **2**
1.1 Methodology and Data Preprocessing . . . 2
1.2 Filtering the SP100 files & Efficiency using Polars . . . 2

2 Data Formatting and NaN Values **4**
2.1 Panel-to-Table Transformation . . . 4
2.2 Missing Value Treatment . . . 4

3 Clustering Methods for Pair Trading **5**
3.1 Motivation . . . 5
3.2 Louvain Clustering . . . 6
3.3 Leiden Clustering . . . 7
3.4 Marsili–Giada . . . 7
3.5 Industry-Based Clustering . . . 7
3.6 Clustering Results and Method Selection . . . 8

4 Application to Pairs Trading **11**
4.1 Pairs Trading Methodology . . . 11
4.2 Implementation Statistics and Performance . . . 15

1 Data Presentation and Preprocessing

1.1 Methodology and Data Preprocessing

The subsequent project utilizes high-frequency trading data from the S&P100 index for the period from January 2nd 2004 to December 31st 2008. This data was provided by Pr. Damien Challet through a moodle Switchdrive link. The raw dataset, totaling approximatively 23.4 GB, comprises two primary subsets :

- **Best Bid and Offer (BBO) Data** : Nanosecond-stamped records of the prevailing best bid and ask prices along with their respective volumes.
- **Trade Data** : Nanosecond-stamped records of all the trades that occurred on a specific day for each ticker.

For the remaining of this report, the BBO dataset was prioritized as we think it provides a more granular view of market liquidity and price mechanisms.

1.2 Filtering the SP100 files & Efficiency using Polars

Processing nanosecond-level data over a five-year horizon presents significant computational challenges. It is the first time we work with such heavy data. To overcome this difficulty, a preprocessing pipeline was developed using the `Polars` library. By leveraging Polars' parallel execution capabilities, the data cleaning runtime was reduced from approximately 13 hours (using `Pandas`) to 4 minutes. This effiency gain was critical for maintaining a scalable iterative workflow without memory exhaustion and kernel crashes. As a result, the rest of the project is implemented using `Polars`.

1.2.1 Index Composition and Asset Filtering

Initial exploratory analysis revealed that the provided S&P100 dataset contained 85 unique equities rather than the nominal 100. Notable large-cap omissions included Apple (ticker : `AAPL`) or Microsoft (ticker : `MSFT`) which was the biggest SP500 market cap at the time. Furthermore, several assets exhibited structural changes or listing shifts during the observation window :

1. **Exits** : Alcoa (ticker : `AA`) exited the available data on August 27th 2007 (it did not exit the SP100 index but rather faced a significant corporate restructuring that made further data unavailable).
2. **Entrants** : Several tickers joined the index mid-sample, including the following tickers : `MA` entered in May 2006, `MS` entered in January 2006, `NOV` entered in March 2005, `PM` entered in March 2008, `V` entered in January 2005 and `DVN` entered in October 2004

In order to ensure a balanced panel and avoid survivorship or entry bias in the subsequent trading strategy, any asset without a continuous trading history across the full period was excluded. As a result, the 7 assets described above were discarded from the dataset.

1.2.2 Missing Data Analysis

A two-layer verification process was implemented to identify gaps in the trading recors. First, a global set of expected trading dates was established. Second, individual asset logs were cross-referenced agains this master set. Significant data gaps were identified in `DVN` (193 missing trading timestamps in 2004), `S` (98 missing trading timestamps in 2005) and `T` were 7 trading timestamps were missing in 2005.

In order to maximize data integrity for the pair trading stategy, the study period was narrowed to a 3-months window, spanning September 1, 2008, to December 31, 2008.

1.2.3 Processing Pipeline & Feature Engineering

To ensure computational tractability, each asset was processed through a standardized transformation pipeline. The objective of this stage was twofold : reduce the dimensionality of the raw dataset ('row reduction') and to engineer robust features for the subsequent trading strategy. The process is executed via the function `process_single_asset`, which utilizes a local data repository (which contained the raw files). The pipeline is as follows :

1. **Data Loading :** The raw parquet file is loaded
2. **Time standardization :** Timestamps are standardized by converting from string format to datetime objects while preserving millisecond precision. We further noticed that the data alternates between UTC-5:00 during winter and UTC-4:00 during summer, however we did not apply any special treatment to it as our analysis does not require it.
3. **Time filtering :** Filters the timestamps and only selects the desired interval (September 1st 2008 to December 31st 2008).
4. **Drop useless columns :** Keep only the columns of interest - `ticker`, `timestamp`, `bid-price`, `bid-volume` and `ask-volume` - i.e, we dropped `xltime` as it is redundant.
5. **Zero-volume rows handling :** We thoroughly inspected all columns for missing or invalid entries. Rows with `NaN` values in either columns were removed as they could distort further analyses. Additionally, rows with zero-volumes were excluded as they represent incomplete or meaningless transactions. This step ensures noise reduction and reliability of the data.
6. **Aggregating the data :** High-frequency data often contains multiple orderbook updates within a single timestamp (multiple trade at the same timestamp). In order to collapse these duplicates into a single representative price point without losing the informational content of the volume, we apply a **volume-weighted average price (VWAP)** aggregation. For any set of N updates at a given timestamp t, the price is defined as :

$$VWAP_t = \frac{\sum_{i=1}^{N}(P_{i,t} \cdot V_{i,t})}{\sum_{i=1}^{N} V_{i,t}}$$

   where $P_{i,t}$ and $V_{i,t}$ represent the price and volume of the i-th update respectively. This method provides a more accurate reflection of the 'equilibrium' price at that instant than the arithmetic mean.

7. **Further characteristics :** Following aggregation, three key market microstructure variables were derived to characterize the trading environment for each asset and each timestep :
   (a) **Spread** : Calculated as : $S_t = P_{ask,t} - P_{bid,t}$ serving as a proxy for transaction costs and liquidty.
   (b) **Mid-price** : Calculated as : $M_t = \frac{P_{ask,t}+P_{bid,t}}{2}$.
   (c) **Volume Imbalance** (V_{imb}) : Defined by : $V_{imb,t} = \frac{V_{bid,t}-V_{ask,t}}{V_{bid,t}+V_{ask,t}}$,

This procedure yields 78 cleaned parquet files that we use in the subsequent phase to create a convenient panel data that we will use later on for estimating covariances matrices, perform clustering and implement a pair trading strategy.

1.2.4 Panel Data creation

In order to get a clean usable Dataframe, we use the function `def create_panel_data`. This function transforms the cleaned parquet files (one per asset we do not discard) into a unified long-format DataFrame where each row represents one asset at one point in time. Indeed, once all the files were cleaned, we observed that each stock was traded at different microseconds timestamps. When concate-

nating all files together, the result is a union of all time columns across all stocks, leading to a sparse dataframe containing a significant number of `NaN` values. Therefore, we decided to resample all cleaned files with 1 minute timestamps, ensuring a smooth aggregation and a limited amount of `NaN` values. The procedure is as follows :

1. **File loading & Filtering** : Locates the cleaned parquet files in the output directory (it is located in our local disk, files are too heavy to be pushed into the Git repo). The assets that have been flagged as "should be skipped" are excluded.
2. **Time resampling** : Applies VWAP aggregation in order to reduce data frequency and optimize computation process during the trading strategy. This is done by applying VWAP to the trades that occur within each minute for each of the assets.

On figure (1) below can be seen an abstract of the 5 first rows of the 1min panel data that will be subsequently used.

timestamp	ticker	ask-price	ask-volume	bid-price	bid-volume	spread	mid-price	volume_imbalance
datetime[µs]	str	f64	f64	f64	f64	f64	f64	f64
2008-09-02 13:30:00	"ABT"	58.537504	605.0	58.309401	484.0	0.228103	58.423452	-0.111111
2008-09-02 13:30:00	"ALL"	45.75773	163.0	45.637612	67.0	0.120118	45.697671	-0.417391
2008-09-02 13:30:00	"BAC"	33.0075	13393.0	32.955497	8656.0	0.052003	32.981498	-0.21484
2008-09-02 13:30:00	"BAX"	68.484787	328.0	68.272418	153.0	0.212368	68.378602	-0.363825
2008-09-02 13:30:00	"BK"	35.3655	280.0	35.302484	153.0	0.063016	35.333992	-0.293303

Figure 1: 1min panel data - Aggregated timestamps

2 Data Formatting and NaN Values

This short second phase transforms the previously created raw panel data into analysis-ready tabular formats while addressing missing values (regular challenge in high-frequency financial data).

2.1 Panel-to-Table Transformation

The initial dataset `panel_data_1min.parquet` contains observations in long format (timestamp, ticker, mid-prices, spreads...). We perform pivot operations to create wide-format matrices of dimension $\approx 33{,}500 \times 78$ (timestamps $\times$ assets). We do that for both stock mid-prices and stock bid-ask spreads. The `aggfunc="last"` handles intra-minute duplicates by retaining the final observation.

2.2 Missing Value Treatment

We use a heatmap (figure 2) to visualize the counts and proportions of missing values per assets per day. Importantly, we notice that the missing value proportions are homgeneous over all 78 assets and are of the order of 5% per day, with a maximum of approximately 12% December 4th, 2008.

We apply a two-stage approach:

1. **Forward-filling**: Missing values propagate from the last known observation via `df.ffill()`. This avoids look-ahead bias inherent in interpolation methods, as it uses only past information:

$$p_i(t) = \begin{cases} \text{observed} & \text{if available} \\ p_i(t-1) & \text{if NaN} \end{cases}$$

2. **First-day truncation**: Forward-filling cannot impute values at initial observations. We remove the first trading day (September 2, 2008) entirely, eliminating all first-entry NaN values while sacrificing $<0.3\%$ of data.

In our notebook, we reuse our heatmapping of missing values to verify complete missing value elimination post-treatment.

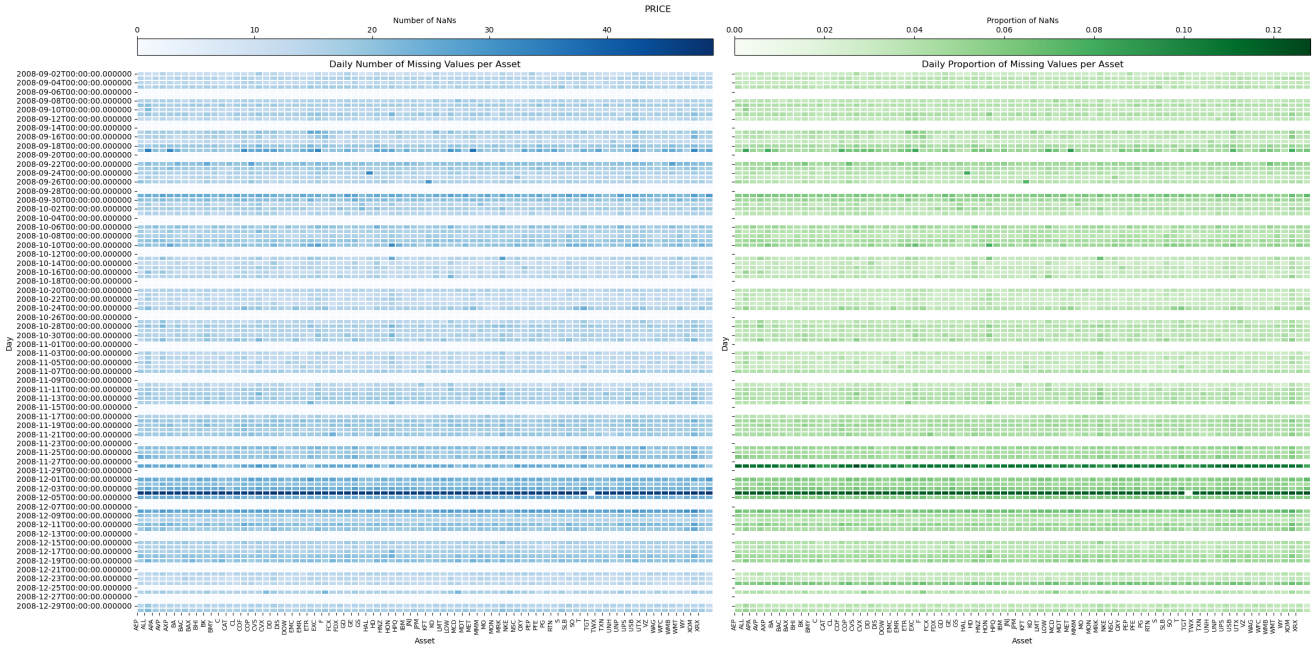

Figure 2: Missing values counts and proportions aggregated per day (stock prices)

Once msising valuse have been handled, we compute minute-level returns are computed as $r_i(t) = \frac{p_i(t) - p_i(t-1)}{p_i(t-1)}$ via `df_prices.pct_change()`.

Finally, our three new matrices are saved as CSV in the (external) data folder and ready to use for following phases:

- `stock_prices.csv`: Mid-prices
- `stock_returns.csv`: Percentage returns
- `stock_spreads.csv`: Bid-ask spreads

3 Clustering Methods for Pair Trading

3.1 Motivation

In statistical arbitrage and, in particular, pair trading strategies, the identification of economically and statistically related assets is an important preliminary step. Given a large universe of stocks and high-frequency intraday data (one-minute observations), the search space for potential trading pairs becomes very large. Clustering methods provide a way to reduce this dimensionality by grouping assets that shows similar behavior according to a chosen similarity measure.

In this project, clustering is used as a pre-filtering step: candidate pairs are only searched within the same cluster. The underlying assumption is that assets belonging to the same cluster are more likely to share common risk factors, display strong short-term co-movements and show stable mean-reverting relationships.

We consider four different clustering methods: Louvain, Leiden, Marsili-Giada and industry-based clustering.

For all clustering methods except the industry-based benchmark, we construct the input network using a Random Matrix Theory (RMT) filtering procedure.

Let $R \in \mathbb{R}^{T \times N}$ denote the matrix of intraday returns, where T is the number of time points and N the number of stocks. We define the empirical correlation matrix as

$$\mathbf{C} = \mathrm{corr}(R).$$

To remove estimation noise, we apply a spectral filtering motivated by Random Matrix Theory. Define the ratio

$$q = \frac{N}{T},$$

and the upper edge of the Marčenko–Pastur distribution,

$$\lambda_+ = (1 + \sqrt{q})^2 .$$

Let the eigen-decomposition of $\mathbf{C}$ be

$$\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top,$$

where $\{\lambda_i\}_{i=1}^N$ are the eigenvalues and $\{v_i\}_{i=1}^N$ the corresponding eigenvectors. The largest eigenvalue is typically associated with the market mode. To isolate sector-level dependencies, we construct a filtered "sector" correlation matrix by retaining eigenmodes that (i) lie above the RMT noise threshold λ_+ and (ii) exclude the dominant market eigenvalue:

$$\mathbf{C}_{\text{sector}} = \sum_{\substack{i:\,\lambda_i > \lambda_+ \\ i \neq i_{\max}}} \lambda_i \, v_i v_i^\top, \qquad i_{\max} = \arg\max_i \lambda_i.$$

Since the clustering algorithms require nonnegative edge weights, we take the elementwise absolute value of the filtered correlation matrix and remove self-loops:

$$\mathbf{W} = |\mathbf{C}_{\text{sector}}|, \qquad W_{ii} = 0 \quad \forall i.$$

The resulting matrix $\mathbf{W}$ is interpreted as the weighted adjacency matrix of an undirected graph $G = (V, E)$.

3.2 Louvain Clustering

The Louvain algorithm detects communities by greedily maximizing the modularity of the network partition. Starting from an initial configuration in which each node forms its own community, nodes are iteratively reassigned to neighboring communities whenever such a move yields a positive modularity gain.

Once no further local improvements are possible, each detected community is collapsed into a super-node and a new reduced network is constructed. The local optimization step is then applied again to this reduced network. These two phases are repeated until modularity converges.

3.2.1 Advantages and Limitations

The Louvain method is computationally efficient and scales well to large networks, making it suitable for high-frequency financial data. Combined with RMT-based correlation cleaning and market-mode removal, it is able to extract sectoral structures from noisy intraday returns.

However, the Louvain algorithm does not guarantee that detected communities are internally connected and may converge to suboptimal local maxima of the modularity objective. As a result, cluster stability can be sensitive to initialization and node ordering, which motivates the comparison with the Leiden method.

3.3 Leiden Clustering

Given the weighted adjacency matrix $\mathbf{W}$, the Leiden algorithm detects communities by iteratively optimizing a modularity-based objective while enforcing that communities remain internally well connected. Compared to Louvain, Leiden includes an explicit refinement step that prevents disconnected or weakly connected communities, leading to more reliable partitions on financial networks.

The output of the algorithm is a partition vector assigning each ticker to a cluster. These clusters are subsequently used as a pre-filtering step for pair selection, restricting candidate pairs to stocks belonging to the same detected community.

3.3.1 Relevance for Pair Trading

This pipeline aims to produce clusters driven by persistent sectoral co-movements rather than by high-frequency noise or the common market component. By focusing pair-search within these clusters, we reduce the combinatorial search space and increase the likelihood of finding economically coherent and statistically stable mean-reverting relationships.

3.4 Marsili–Giada

In addition to modularity-based community detection (Louvain/Leiden), we consider a model-based clustering method introduced by Giada and Marsili. The approach assumes that stocks in the same cluster share a common latent component, which induces higher within-cluster correlation. Clusters are obtained through an agglomerative (bottom-up) merging procedure that maximizes a likelihood criterion defined directly from the correlation matrix.

As for the graph-based approaches, we first clean the empirical correlation matrix using a Random Matrix Theory (RMT) filter and remove the market mode, in order to reduce the influence of noise and the global common factor.

3.4.1 Advantages and Limitations

A key advantage of the Marsili–Giada approach is that it is model-based: it clusters assets by directly optimizing a likelihood criterion grounded in a latent-factor interpretation of within-cluster correlation. It also does not require specifying the number of clusters a priori.

However, the procedure is greedy and can be computationally expensive for larger universes because it evaluates many candidate merges. In addition, its behavior depends on the quality of the correlation estimate; in particular, RMT cleaning and market-mode removal are crucial to avoid clusters being dominated by noise or by the global market component.

3.5 Industry-Based Clustering

In addition to data-driven clustering methods, we consider an industry-based clustering scheme as a benchmark. In this approach, stocks are grouped according to predefined industry classifications rather than inferred statistical relationships. This method reflects common practice in financial analysis, where assets are often compared within the same industry due to shared business models, regulatory environments, and exposure to similar risk factors.

Unlike the previous clustering methods, the industry-based clustering does not rely on intraday return data or correlation estimates and therefore provides a useful reference point to assess the added value of more sophisticated, data-driven approaches.

3.6 Clustering Results and Method Selection

3.6.1 Summary Statistics Across Methods

Table 1 reports the basic statistics of each clustering output (number of clusters and cluster-size dispersion). Leiden and Louvain yield a small number of medium-to-large clusters which is desirable to reduce the pair-search space while keeping enough candidates per cluster. Marsili–Giada produces a very large number of tiny clusters (mostly pairs), effectively acting closer to a pair selection mechanism than a coarse pre-filtering step. Industry clustering yields a moderate number of clusters but includes very small groups (including a singleton), which can be limiting for systematic pair search.

Table 1: Cluster statistics comparison across methods.

Method	N_{clusters}	Largest	Smallest	Avg size	Std size
Leiden	4	25	9	19.5000	7.1414
Louvain	5	22	9	15.6000	6.1887
Marsili–Giada	38	3	2	2.0526	0.2263
Industry	14	12	1	5.5714	3.5020

3.6.2 Within-Cluster Correlation Analysis

A key criterion for pair trading pre-selection is that clusters should group assets that exhibit strong dependence, i.e., high within-cluster correlation. Tables 1 report within-cluster correlation statistics computed from the (cleaned) correlation matrix.

Leiden: Leiden produces 4 relatively large clusters with a fairly uniform average within-cluster correlation (from 0.368 to 0.436). The overall average within-cluster correlation is 0.4030 (Table 2).

Table 2: Leiden: within-cluster correlation statistics.

cluster_id	n	avg	min	max	std
0	25	0.435686	0.154987	0.673119	0.104464
2	22	0.433159	0.265727	0.664249	0.064856
3	9	0.374924	0.258469	0.505085	0.053804
1	22	0.368209	0.082530	0.570414	0.102504
Overall average within-cluster correlation: **0.4030**					
Best Leiden cluster (by average): **Cluster 0** with **0.4357**					

Louvain: Louvain achieves a higher overall average within-cluster correlation of 0.4273 and identifies one particularly cohesive cluster (cluster 3) with average correlation 0.5387 (Table 3). This is consistent with the industry-composition plot, where one cluster appears almost sector-pure (notably Energy).

Table 3: Louvain: within-cluster correlation statistics.

cluster_id	n	avg	min	max	std
3	9	0.538737	0.429629	0.671660	0.054335
0	22	0.433159	0.265727	0.664249	0.064856
4	20	0.409690	0.108433	0.673119	0.137785
1	18	0.379777	0.201733	0.570414	0.083671
2	9	0.374924	0.258469	0.505085	0.053804
Overall average within-cluster correlation: **0.4273**					
Best Louvain cluster (by average): **Cluster 3** with **0.5387**					

Marsili–Giada: Marsili–Giada yields 38 clusters, almost all of size 2. The reported within-cluster correlation for size-2 clusters is simply the correlation of that pair, which makes the method behave like a direct pair builder. The overall average within-cluster correlation is 0.3897, and the best pair-cluster reaches 0.5630 (Table 4).

Table 4: Marsili–Giada: within-cluster correlation statistics.

cluster_id	n	avg	min	max	std
19	2	0.5630	0.5630	0.5630	0.0000
2	2	0.4983	0.4983	0.4983	0.0000
55	2	0.4941	0.4941	0.4941	0.0000
⋮					
34	2	0.3189	0.3189	0.3189	0.0000
49	2	0.3146	0.3146	0.3146	0.0000
31	2	0.3098	0.3098	0.3098	0.0000
26	2	0.1560	0.1560	0.1560	0.0000
48	2	0.1499	0.1499	0.1499	0.0000
Overall average within-cluster correlation: **0.3897**					
Best cluster (by average): **Cluster 19** (**0.5630**)					

Industry baseline: Industry clustering achieves the highest overall within-cluster correlation (0.4701), reflecting that firms in the same sector often co-move. Results are summarized in Table 5.

Table 5: Industry: within-cluster correlation statistics

cluster_id	n	avg	min	max	std
11	2	0.6477	0.6477	0.6477	0.0000
8	2	0.5641	0.5641	0.5641	0.0000
13	4	0.5462	0.5224	0.5704	0.0187
2	10	0.5265	0.3788	0.6717	0.0585
⋮					
4	3	0.3589	0.3309	0.3753	0.0199
7	3	0.3306	0.3058	0.3782	0.0337
Overall average within-cluster correlation: **0.4701**					
Best cluster (by average): **Cluster 11** (**0.6477**)					

3.6.3 Economic Coherence

Figure 3 reports the stacked bar plots of industry composition by detected cluster. Both Leiden and Louvain recover economically meaningful structures. In particular, Louvain isolates at least one cluster that is nearly industry-pure (Energy), while Leiden produces larger clusters that mix multiple industries more frequently. This suggests that Louvain provides a finer segmentation of sectoral co-movement at the intraday horizon.

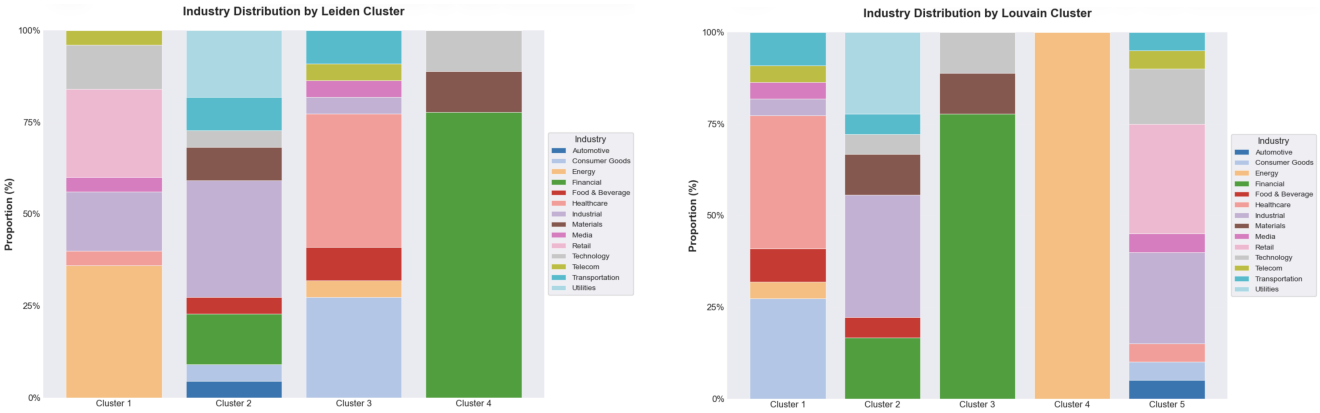

Figure 3: Industry composition by Leiden (left) and Louvain (right) clusters.

3.6.4 Choice of Clustering Method for Pair Trading

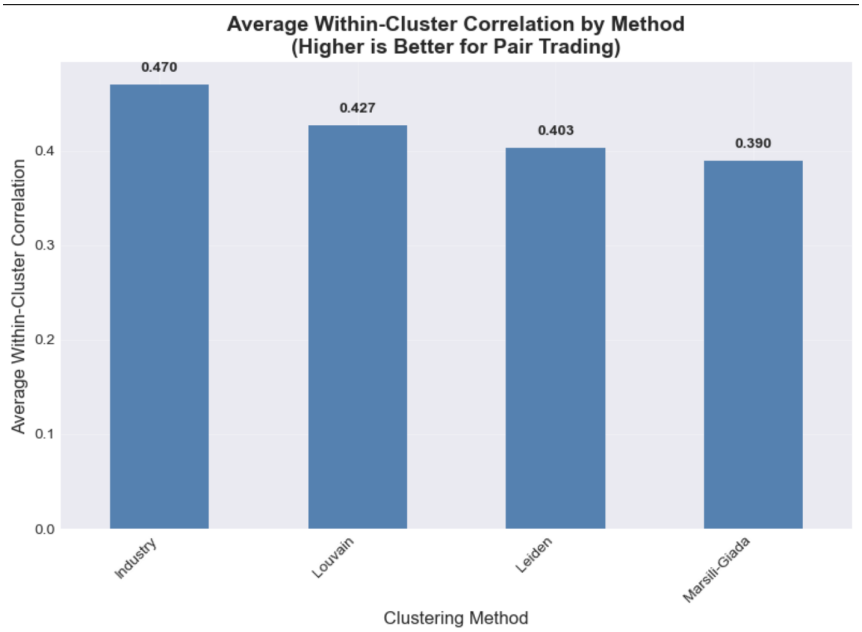

Figure 4: Average within-cluster correlation across clustering methods

Figure 4 summarizes the average within-cluster correlation across methods

For pair trading, the clustering step is used as a pre-filter to restrict the universe of candidate pairs. A good clustering method should therefore: (i) yield sufficiently large clusters to allow multiple candidate pairs, (ii) maximize within-cluster dependence (to increase the probability of strong relationships), (iii) remain economically interpretable and stable.

Based on the reported metrics:

- **Marsili–Giada** produces 38 clusters of size mostly 2, which is too granular for a pre-filtering step

and behaves closer to direct pair formation. This may overly restrict the search and risks missing useful pairs that do not appear as the single best merge outcome.

- **Industry clustering** Industry-based clustering achieves a high average within-cluster correlation, reflecting the fact that firms within the same sector often share common long-term risk exposures. However, industry classifications are static and designed to capture similarities in business models and fundamentals rather than short-horizon return dynamics. At the intraday frequency considered in this study, co-movements are driven primarily by liquidity effects, order-flow interactions, and transient market microstructure factors that are not necessarily aligned with industry boundaries.
- **Leiden** provides a robust partition into 4 large clusters but achieves a lower overall within-cluster correlation (0.4030), indicating that clusters may be too coarse for intraday pair selection.
- **Louvain** offers the best trade-off in our setting: it keeps cluster sizes large enough for systematic pair search (avg size 15.6), while achieving a higher overall within-cluster correlation (0.4273) and identifying a highly cohesive cluster (avg 0.5387). The industry-composition plot also suggests stronger sectoral purity in at least one cluster.

Therefore, we select Louvain clustering as the primary pre-filtering method for downstream pair trading. Industry clustering could be retained as a baseline benchmark, and Marsili–Giada is best interpreted as an alternative approach closer to direct pair proposal rather than clustering.

3.6.5 Remarks on Look-Ahead Bias

The clustering results presented here are intended for exploratory and comparative purposes only. Clusters are computed using the full sample to assess structural properties, stability, and economic interpretability across methods, and therefore rely on information not available in real time.

In the actual pair trading implementation, look-ahead bias is avoided by recomputing clusters on a rolling-window basis using only past data. At each rebalancing date, pair selection is restricted to stocks within the same contemporaneously estimated cluster.

Accordingly, this section serves to motivate the choice of clustering methodology rather than to define a fixed clustering for trading. The final strategy strictly follows a forward-looking framework consistent with realistic trading constraints.

4 Application to Pairs Trading

As a final phase of our project, we decided to implement a pairs trading strategy. This phase serves as an exploration of how clustering and correlation analysis can be leveraged in a financial context, rather than an attempt to construct an optimally profitable trading system. The emphasis remains on demonstrating proficiency in handling large-scale, high-frequency financial data by processing minute-frequency stock prices for 78 assets over a four-month period (rather than on achieving superior financial returns). Nevertheless, the strategy's eventual performance proved quite satisfactory, validating the computational approach.

4.1 Pairs Trading Methodology

Pairs trading is a market-neutral statistical arbitrage strategy that exploits temporary divergences between historically correlated assets. The fundamental premise is that when two stocks with strong co-movement experience an abnormal divergence in their returns, this divergence will eventually revert to the mean, creating a profit opportunity. The strategy involves taking opposing positions in the two assets (long one, short the other) when divergence exceeds a threshold, then closing these positions upon convergence. Our implementation integrates clustering analysis from Phase 3 to identify candidate pairs within groups of related stocks, thereby increasing the likelihood of genuine statistical relationships rather than spurious correlations. The strategy operates under the assumption that assets within the

same cluster share underlying economic factors that drive their long-term co-movement.

4.1.1 Rolling Window Methodology

To eliminate look-ahead bias, the strategy operates on a rolling window basis with 300-timestamp cycles. At each timestamp t where $t \bmod 300 = 0$, we perform the following procedure:

1. **Clustering Phase and Pair Construction**: Using the previous 300 timestamps of return data (from $t-300$ to $t-1$), we apply the Louvain clustering algorithm developed in Phase 3 to partition the 78 assets into clusters based on their correlation structure. We construct all possible intra-cluster pairs from the identified clusters.

2. **Trading Phase**: These filtered pairs are then actively traded for the subsequent 300 timestamps (from t to $t+299$). At timestamp $t+300$, all open positions are forcibly closed, and the cycle repeats with fresh clustering.

This approach ensures that clustering decisions rely exclusively on historical data available at decision time, while the 300-timestamp window (approximately 5 hours of minute-frequency data) balances statistical power against market regime stability.

4.1.2 Trading Logic

Entry Signal

For each pair (Stock A, Stock B), we continuously monitor the return differential with a lookback window of 50 timestamps. At each timestamp t during the trading phase, we calculate:

- Return differential: $r_{\text{diff}}(t) = r_A(t) - r_B(t)$
- Rolling volatility: $\sigma(t) = \text{std}\left(r_{\text{diff}}[t-50:t]\right)$
- Spread cost: $\text{spread\_cost}(t) = \frac{\xi_A(t)}{2 \cdot p_A(t)} + \frac{\xi_B(t)}{2 \cdot p_B(t)}$

where $\xi_i(t)$ denotes the bid-ask spread and $p_i(t)$ the mid-price of asset i at time t.

Entry Condition

A position is entered when:

$$|r_{\text{diff}}(t)| > \lambda_{\text{in}} \cdot \sigma(t) + \text{spread\_cost}(t)$$

with $\lambda_{\text{in}} = 3$.

When $r_{\text{diff}}(t) > 0$, we short Stock A and long Stock B (inverted position). When $r_{\text{diff}}(t) < 0$, we long Stock A and short Stock B (standard position). Each leg consists of ± 1 unit, ensuring zero-cost strategy.

Exit Mechanisms

Once a position is entered at time t_{in} with initial cumulative return differential R_{in}, we track the cumulative return differential $R_{\text{cum}}(t)$ and exit under any of three conditions:

Exit Condition 1: Profitable Convergence

- For inverted positions: $R_{\text{cum}}(t) < \lambda_{\text{out}} \cdot R_{\text{in}}$
- For standard positions: $R_{\text{cum}}(t) > \lambda_{\text{out}} \cdot R_{\text{in}}$

with $\lambda_{\text{out}} = 0.5$.

Exit Condition 2: Emergency Exit (Drift Detection)

- For inverted positions: $R_{\text{cum}}(t) > \lambda_{\text{emergency}} \cdot R_{\text{in}}$
- For standard positions: $R_{\text{cum}}(t) < \lambda_{\text{emergency}} \cdot R_{\text{in}}$

with $\lambda_{\text{emergency}} = 3$.

Exit Condition 3: Patience Timeout

A patience counter c is incremented when the position deteriorates:

- For inverted positions: $c \leftarrow c + 1$ if $R_{\text{cum}}(t) \geq R_{\text{cum}}(t-1)$, else $c \leftarrow 0$
- For standard positions: $c \leftarrow c + 1$ if $R_{\text{cum}}(t) \leq R_{\text{cum}}(t-1)$, else $c \leftarrow 0$

Exit when $c \geq c_{\max}$ with $c_{\max} = 120$.

The following heatmap (figure 5) represents positions taken in the 78 different assets over the entire trading period. Negative positions (blue cells) correspond to shorting units of the asset, positive positions (red cells) correspond to longing units of the asset. Note the presence of a 79th column (at the right extremity) which is the row-wise aggregation of positions (sum), providing a great sanity check that the sum of positions is always zero (proving that this is a zero-cost strategy before applying transaction costs).

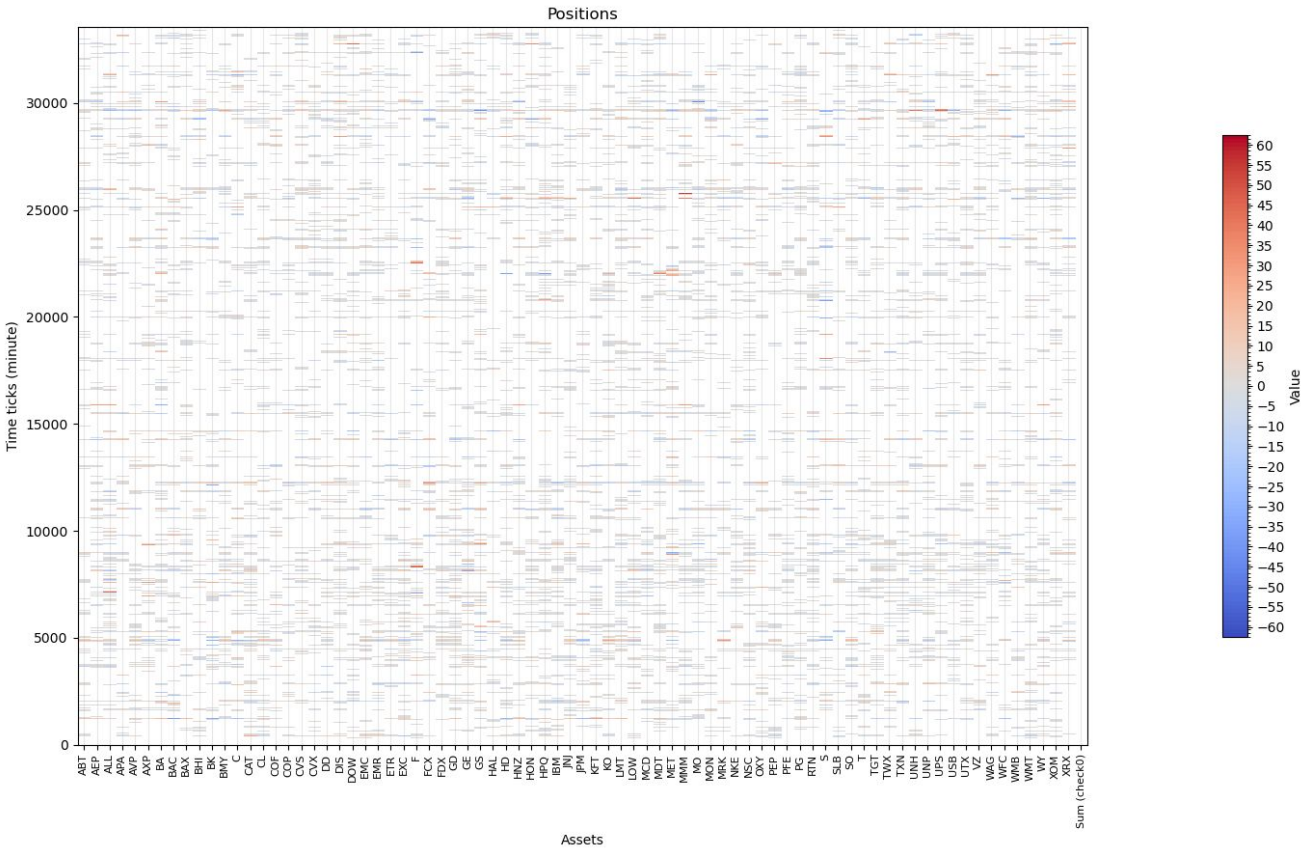

Figure 5: Positions in each asset over time. The sum of positions equals O at each time.

4.1.3 Transaction Cost Modeling

Rather than applying a fixed percentage cost, we model transaction costs based on the bid-ask spread. When a position in asset i changes by Δ_i units at time t, the transaction cost is:

Transaction cost

$$\text{cost}_i(t) = |\Delta_i| \cdot \frac{\xi_i(t)}{2 \cdot p_i(t)}$$

where $\xi_i(t)$ is the bid-ask spread of asset i at time t, and $p_i(t)$ is the mid-price. The factor of $\frac{1}{2}$ reflects the cost of crossing the spread once (either buying at ask or selling at bid). Total transaction costs are computed by summing across all assets with position changes:

$$\text{total\_cost}(t) = \sum_{i=1}^{N} \text{cost}_i(t) = \sum_{i=1}^{N} |\Delta_i(t)| \cdot \frac{\xi_i(t)}{2 \cdot p_i(t)}$$

This realistic modeling captures the heterogeneous liquidity across assets and the elevated spreads characteristic of the 2008 crisis period in our dataset.

4.1.4 Position Aggregation and PnL Calculation

At each timestamp, positions from all active pairs are aggregated into a position vector across all 78 assets. The cumulative PnL is then computed as:

$$\text{PnL}(t) = \sum_{\tau=1}^{t} [\text{position}(\tau - 1) \cdot \text{returns}(\tau)] - \text{total\_cost}(\tau)$$

where positions from the previous timestamp earn returns during the current period, and transaction costs are deducted when positions change. This lag structure correctly models the fact that trading decisions made at time *t* cannot earn returns until time *t+1*.

We finally plot the cumulative PnL over the trading period (figure 6).

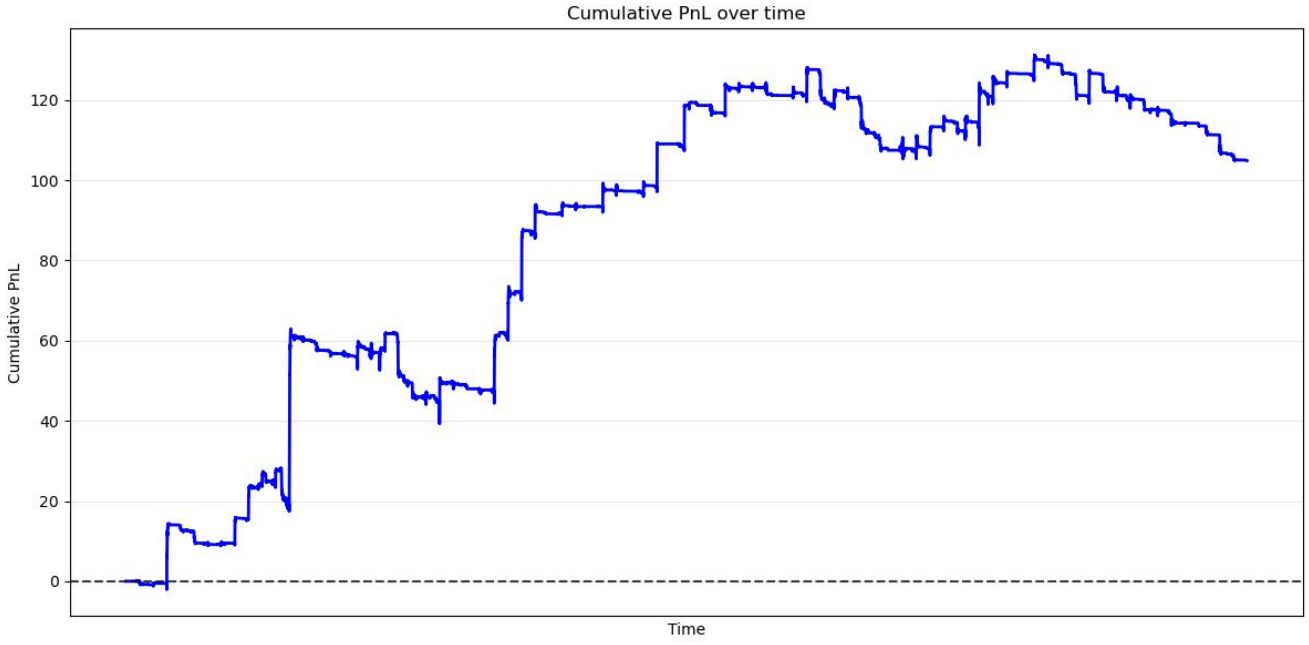

Figure 6: PnL of strategy

4.2 Implementation Statistics and Performance

4.2.1 Computational Performance

The strategy processed approximately 33,500 minute-frequency ticks, 111 complete clustering/trading cycles, with an average of 683 pairs per window resulting in a total of 22,743,900 pair-timestamp evaluation. The complete strategy execution required **approximately 30 minutes** on standard hardware. This represents a reasonable computational burden given the data scale. The majority of computational time was spent in the nested loop structure evaluating each pair at each timestamp within each rolling window.

4.2.2 Opportunities for Performance Optimization

While the current implementation achieves acceptable performance, several optimizations could dramatically reduce execution time for larger-scale applications. We could for example vectorize computations: the current implementation uses Python loops to iterate through pairs and timestamps. Replacing these with NumPy vectorized operations for calculating rolling statistics, spreads, and position updates could yield 10 to 100 times speedups, as vectorized operations leverage optimized C libraries. Even better, we could rely on parallel processing: pairs within a trading window are completely independent (one pair's positions do not affect another's). This parallel structure could be distributed across multiple CPU cores or GPUs. With modern multi-core processors, parallelizing across 8-16 cores could reduce execution time proportionally.

Conclusion

This project provided hands-on exposure to high-frequency financial data, offering empirical insights into how market microstructure shapes trading dynamics within the limit order book. It closely mirrored industry-level practices, thereby bridging the gap between theoretical coursework and practitioner-oriented data workflows.

Despite the considerable technical difficulty of the task, we began with a raw dataset of previously unseen scale and complexity and systematically addressed both structural and practical challenges inherent to nanosecond-level order book data. The adoption of the `Polars` library proved decisive, enabling large-scale data manipulation with substantial improvements in runtime and memory stability, ultimately making iterative experimentation feasible. Further work could investigate how temporal aggregation, as implemented in this project, affects return dynamics and potentially biases covariance matrix estimates. In addition, a systematic analysis of resampling granularity (e.g., 1-minute versus 5- or 10-minute intervals) could assess its impact on data sparsity, measured through the proportion of missing values, as well as on the performance and stability of an illustrative trading strategy.

A central contribution of this project lies in the use of clustering as a structural tool to organize a large cross-section of assets and make intraday pair trading computationally and economically tractable. By comparing several clustering approaches, we showed that data-driven methods, particularly Louvain, effectively capture short-horizon co-movements beyond static industry classifications, while highlighting trade-offs between cluster granularity and dependence strength. Nevertheless, clustering remains sensitive to estimation noise and window choice, suggesting that more adaptive or time-varying clustering schemes could further enhance robustness.

Finally, the pairs trading implementation successfully demonstrates the application of clustering and correlation analysis techniques to a practical financial problem, while maintaining rigorous focus on efficient large-scale data handling. The strategy's 30-minute execution time for processing 78 assets at minute frequency over four months supports its primary objective: showcasing proficiency in managing and processing high-frequency financial big data within a meaningful application context.