ChatGPT

Using Unsupervised Learning to Predict Stock Movement for Swing Trading

Identifying the Target Output (Market Phases & Patterns)

Unsupervised models won't directly output "buy" or "sell" signals with predicted prices; instead, they uncover **hidden market states or patterns** that you can interpret and use for swing trading decisions. Often the goal is to identify **market regimes** – for example, clustering time periods into *bullish* vs *bearish* phases or other repeatable patterns [1] [2]. By grouping days with similar characteristics, an unsupervised model might reveal that a certain cluster of conditions tends to precede an upward swing, while another cluster corresponds to downturns. For instance, a **Hidden Markov Model (HMM)** (a sequential unsupervised technique) can detect bull and bear market stages from price and volatility data [1]. Likewise, applying k-means clustering to candlestick patterns can segment market days into regimes, which **aids in developing entry/exit rules** for trading (e.g. only enter trades during a "bullish regime" cluster) [2]. In practice, you would use these clusters or states as your prediction output – e.g. *"current conditions belong to a bullish cluster, which historically leads to price rises."* This effectively turns an unsupervised pattern into a swing trading signal (such as identifying when the market is likely to swing upward or downward next).

Data Sources and Features to Include

To make the clustering meaningful, you'll want to feed the model a rich set of features capturing **technical, fundamental, and sentiment factors** – essentially *"all"* relevant data as you indicated:

- **Technical Indicators & Price Action:** Include common swing-trading technicals like moving averages, RSI, MACD, Bollinger Bands, ATR (volatility), etc., as well as raw price action features (recent returns, candlestick shape metrics like percent change from open to close, high/low ranges, etc.). These quantify the trend, momentum, and volatility state of the stock. For example, you might use the last N days' returns or indicator values as part of the feature vector to capture short-term patterns. Including such trend and volatility indicators provides the algorithm with potentially predictive information about market state [3].

- **Volume and Order Flow:** Features like average volume, volume spikes, volume relative to average, or indicators such as OBV (On-Balance Volume) can signal accumulation or distribution phases. High volume combined with price moves might distinguish explosive breakout days from quiet ones, which is useful in clustering market conditions.

- **News Sentiment & Headline Data:** Quantify news or social media sentiment for the stock or market. For instance, you could use a daily sentiment score (from news articles or Twitter) or counts of positive vs. negative news. Sentiment adds a fundamental context – e.g. clustering could separate days with bullish news momentum from days with fear or uncertainty. In fact, research has shown value in combining **financial news data with market data** for unsupervised pattern discovery [4]. A recent study clustered **Google News headlines and Yahoo Finance price data** together using PCA + k-means, and identified distinct market "conditions" – notably finding that most of the time the market stayed in a steady state, but one cluster (with a small fraction of days) corresponded to **atypical high-volatility conditions** [4]. This underscores how including sentiment/news can help the model discern *unusual* regimes (e.g. panic-driven volatility spikes) from normal days.

- **Fundamental Indicators:** For swing trading a single stock, daily fundamental changes are subtle, but you can include things like valuation metrics (P/E, P/B ratios), earnings surprises, or sector-level fundamental signals. These might not change daily, but they can help cluster the *type* of stock or the broader context (growth vs value environment, etc.). If you expand the scope to clustering **multiple stocks** together, then fundamental features become even more important – e.g. clustering stocks by fundamentals each quarter to find peer groups [5]. (For a single stock's short-term movements, fundamentals are secondary, but they can still provide context like "overvalued and crowded" vs "undervalued," which might influence its swing behavior.)

- **Macro and Others:** Don't hesitate to include broader factors like market index trends, sector performance, interest rates, or volatility index (VIX) levels as features. Swing trading a stock can be affected by these external variables (market regime or sector rotation). They can help the unsupervised model differentiate, say, "stock is rising but in the context of a bullish market vs. rising against a bearish market," which might form different clusters.

Data Preprocessing: With such diverse data, careful preprocessing is key. All features should be scaled to a comparable range so that no single feature (e.g. a huge raw volume number or a P/E ratio) dominates the distance calculations. Often, **log-transforming skewed features and using robust scalers or z-score normalization** is effective [5]. For example, you might log-scale volume or fundamental ratios to reduce skew, then standardize each feature. This was the approach in one quantitative strategy: they log-transformed and robust-scaled fundamental features before clustering to handle heavy-tailed distributions and outliers in financial data [5]. Additionally, align all data sources by timestamp (e.g. merge daily prices with that day's sentiment score and last known fundamentals). You may need to fill forward fundamental data (since it updates quarterly) or use week-average sentiment if news is too noisy day-to-day. The result of this step is a time-indexed dataset where each day (or each period you choose for swing trading, say each trading day) has a feature vector incorporating technicals, volume, sentiment, etc.

Full Pipeline: Unsupervised Swing Trading Strategy

Using the above data, here's a step-by-step pipeline for applying unsupervised learning to generate swing trade signals. We'll outline the full process – from data to trading decision – and suggest modeling techniques along the way:

1. **Data Collection & Preparation:** Gather historical data for your stock (price OHLC data, volume) and relevant features. For technicals, you can compute indicators from the price series. Collect daily news sentiment scores (e.g. via an API or sentiment analysis model on headlines) and fetch fundamental metrics (company financials or valuation ratios). Merge these into one timeline. Perform the necessary preprocessing: handle missing values, smooth noisy series if needed (e.g. a rolling average on sentiment), and scale features. At this stage, ensure features are on comparable scales – for instance, scale numeric features to zero-mean, unit-variance or use percentile scaling. This will prevent distance-based algorithms from being overly influenced by one feature. *Example:* If your feature set includes RSI, 5-day return %, trading volume, and a sentiment index, normalize each so that RSI ~ 50 and sentiment ~ 0 mean have similar weight. This creates a clean matrix of feature vectors for each day.

2. **Feature Engineering & Dimensionality Reduction:** It's often useful to expand your feature set with domain-specific insights and then possibly reduce complexity. You might engineer features like: recent momentum (e.g. 5-day and 10-day returns), volatility measures (e.g. std dev of returns or ATR over last 14 days), **trend indicators** (slope of a moving average), **oscillators** (RSI, stochastics), or pattern-based flags (e.g. "hammer" candlestick pattern indicator). Volume could be turned into features like volume surge (today's volume / 20-day average). For sentiment, you might include a 3-day moving average of news sentiment to capture persistent mood. Including a wide array is good, but too many highly correlated or noisy features can confuse clustering. This is where dimensionality reduction helps. You can apply **Principal Component Analysis (PCA)** to the feature matrix to compress it into a smaller number of principal components that explain most variance [6]. PCA will combine correlated features (like various moving averages or correlated fundamentals) into composite features, reducing noise. In the study that combined news and market data, PCA was used as a first stage to identify the most important dimensions of variation before clustering [4] [6]. Alternatively, more advanced methods like **autoencoders** (non-linear dimensionality reduction) could be used to learn a compact representation of the data. The goal is to distill the feature set so that the clustering algorithm can focus on a few informative dimensions (e.g. perhaps one dimension captures overall **trend strength**, another captures **sentiment**, another **volatility**). This step is optional but highly recommended when you have "all" the data – it prevents the curse of dimensionality and yields more robust clusters.

3. **Unsupervised Model Training (Clustering):** Now, feed the processed feature vectors into an unsupervised learning algorithm to **discover clusters or states**. A straightforward choice is **K-Means clustering**, which will partition days into *K* clusters based on similarity in the multi-dimensional feature space. You would need to decide on K (the number of clusters) – methods like the elbow plot or silhouette score can help find an appropriate K by testing how well-separated the clusters are. (In one fundamental-data clustering example, 11 clusters gave the best silhouette score, but robustness was seen across a range of K [5]; for your data, K might be, say, 3-6 if you expect regimes like "bullish," "bearish," "sideways," etc.) If you're unsure of K, you could use **hierarchical clustering** instead – it builds a dendrogram of clusters and you can choose a cut that yields a natural grouping. In fact, hierarchical clustering is noted as a good alternative because it **does not require specifying the number of clusters in advance** [7]. Other clustering algorithms can be applied here as well (we'll discuss choices in the next section). The output of this training step is that each historical time period (each day, in our framing) is assigned to a cluster – essentially labeling each day with an unsupervised regime ID.

4. **Cluster Interpretation & Regime Labeling:** After clustering, **analyze the characteristics of each cluster**. This is crucial for turning clusters into actionable signals. You'll want to answer: *What does each cluster represent in trading terms?* Look at the feature averages/medians per cluster: for example, Cluster A might have low RSI, low price momentum but extremely negative sentiment (perhaps indicating **oversold panic days**), whereas Cluster B has high momentum, high RSI, positive sentiment (a **strong uptrend** regime), and Cluster C might show high volatility and negative returns (a **sell-off** regime). You might also notice clusters that correspond to sideways consolidation (indicators neutral, low volatility). Importantly, check the **forward stock performance** associated with each cluster historically. Since we're doing swing trading, you could compute the average **next 5-day return** or next 10-day move for all instances of each cluster. This effectively evaluates whether the cluster has bullish or bearish implications. For example, one clustering experiment on daily candlesticks found that *one particular cluster of candle patterns had significantly higher next-day returns than others*, even outperforming the baseline market return [8]. That cluster (in their case, a certain type of small-down-day pattern) could then be used as a bullish signal. If you find a cluster that, say, historically yields +2% average return in the following week (with a decent win rate), that's a strong bullish cluster. On

the other hand, a cluster that precedes losses can signal caution or short opportunities. At this stage, you might assign intuitive labels to clusters – e.g. “Bullish Reversal Setup,” “Bearish Breakdown,” “Stable/Base State,” etc., based on their features and outcomes. This interpretation step ties the unsupervised results back to real trading logic.

5. **Trade Signal Generation:** Using the interpreted clusters, design your swing trading rules. One straightforward approach is **to trade based on cluster membership**. For instance, you may decide: “If today’s data lands in the **bullish cluster**, enter a long position (expecting an upswing over the next several days); if it’s in the bearish cluster, either short or stay in cash; if in a neutral/consolidation cluster, perhaps hold off on new trades.” Essentially, the cluster acts as a filter or trigger. This aligns with the idea of using clusters as **entry filters or regime filters to improve strategy profitability** [2]. Another usage is position sizing or risk management: you might stay in a trade but reduce exposure if the market transitions into a riskier cluster. If you opted for an HMM instead of static clustering, you would use the inferred hidden state probabilities similarly – e.g. if the HMM indicates a high probability of being in a “bull” state, you take or hold long positions, whereas an inferred “bear” state prompts you to tighten stops or go short. The unsupervised model essentially provides a context signal; you layer your trading action onto it. It’s wise to integrate this with a **trading strategy logic** – for example, you might combine cluster signals with a confirmation indicator. Say cluster X is bullish: you could still wait for price to break a technical level upward before entry, using the cluster as a confidence booster. Or cluster Y is bearish: you avoid buying breakouts when the market’s in that cluster. These are just examples – the key is that the unsupervised model narrows down *when* to act (during favorable regimes) and potentially *how* to act (e.g. aggressively in bullish regimes, defensively in bearish regimes).

6. **Backtesting and Refinement:** Before deploying this approach, backtest it on historical data. Walk through past data, labeling each day by cluster (you can do an *in-sample clustering* on a training period, then fix the cluster centroids and label a test period to simulate how it would work in real-time). See how a simple cluster-based strategy would have performed: e.g. buy at close on days classified as bullish-regime and sell after N days, etc., and compare to baseline performance. This will reveal if the clusters indeed carried predictive power (some clusters might turn out to be noise). It also helps in refining the pipeline: you may discover that certain features weren’t helpful (if clusters seem random with respect to returns, perhaps include different features or reduce noise further). You might try different numbers of clusters or different algorithms and evaluate which yields more usable groupings (for example, maybe **Gaussian Mixture clustering** finds more nuanced regimes than k-means, or maybe a **DBSCAN** better isolates outlier events – see next section for these alternatives). The backtest can guide you to tweak the look-back window for features (maybe using past 3 days vs 10 days in features improves regime detection) or to adjust how you generate signals (maybe cluster is best used as a filter combined with another strategy). **Iterate** on the clustering and feature set until you achieve a satisfactory balance of interpretability and performance in backtesting.

7. **Execution & Monitoring:** Once validated, integrate the unsupervised model into your trading routine. This means periodically updating the model with new data. For example, you might re-run the clustering every week or month to ensure the clusters reflect the latest market conditions (market patterns can evolve – unsupervised models should be updated to adapt). Some approaches re-cluster dynamically (e.g. Parker’s strategy re-trained clusters each quarter to account for new data [5]). Alternatively, with an HMM, you could continually feed new observations to update state probabilities on the fly. During live trading, each day you compute that day’s feature vector, feed it to the model (e.g. find which cluster centroid it’s closest to), and then apply your trading rule for that cluster. Keep monitoring performance and if you notice the

signals degrading, it may be time to retrain the model or re-examine your features – *unsupervised models can "drift" if the underlying market dynamics shift significantly*.

By following this pipeline, you end up with a framework where **the unsupervised algorithm detects the context (market state) and you make trading decisions informed by that context**. This can greatly enhance a swing trading strategy – for example, avoiding trades during choppy, unpredictable regimes and focusing on periods where the market has a clear pattern similar to past profitable swings [8].

Unsupervised Learning Techniques to Consider

Several unsupervised ML methods could be applied to this problem. Choosing the "best" depends on your specific data and goals, but here are some top candidates and how they fit in:

- **K-Means Clustering:** A popular choice for its simplicity. It partitions the data into *K* clusters by minimizing within-cluster distance. K-means tends to work well if you suspect your data naturally falls into a few distinct pattern groups (e.g. perhaps 3–5 market regimes). It's fast and easy to implement, but you must choose K and it assumes clusters are roughly spherical in feature space. Despite its simplicity, k-means has been successfully used in trading contexts – for example, to group similar days or stocks, aiding in strategy filters [2]. Keep in mind that financial data is noisy; k-means will **force a grouping even if patterns overlap**, which can sometimes create spurious clusters [9]. To mitigate that, you might run k-means multiple times or compare with other methods.

- **Hierarchical Clustering:** This method builds a hierarchy (tree) of clusters without needing a preset K. You can use **agglomerative clustering** to let all points start separate and then merge clusters stepwise. The result is a dendrogram you can cut at a chosen level to get clusters. The advantage is you can see the whole nesting structure – perhaps revealing that certain clusters are subdivisions of bigger regimes. It also allows you to decide the number of clusters *after* seeing the structure (perhaps based on a distance threshold). Hierarchical clustering can be computationally heavier on large data, but for daily stock data it's manageable. As noted earlier, it's a good alternative if you want the data to "speak for itself" on how many regimes exist [7]. You could even use hierarchical clustering to determine K for k-means (e.g. if a dendrogram shows 4 natural groupings, use k=4 in k-means).

- **Gaussian Mixture Models (GMM):** This is a probabilistic soft-clustering approach. Instead of hard assignments like k-means, GMM assumes the data is generated from a mixture of Gaussian distributions. It will output both cluster assignments and probabilities (e.g. day X is 70% in cluster1, 30% in cluster2), which can be informative if some days are borderline. GMM can capture clusters of different shapes (not just spherical) by adjusting covariance, and it naturally handles the idea that financial regimes might have different volatility (cluster covariances). If your data has overlapping patterns, GMM's soft assignments might model it better. Many practitioners consider GMM a more flexible version of k-means (k-means is actually a special case of GMM with equal spherical covariances). In fact, advanced sources often suggest trying GMM (with an EM algorithm) for financial clustering [10].

- **DBSCAN and Density-Based Clustering:** DBSCAN is useful if you suspect **anomalous clusters or outliers** are important. It groups points that are tightly packed together and labels sparse points as outliers. In a stock context, this could mean identifying *common market regimes* as clusters and flagging unusual days (e.g. flash crashes, extreme events) as outliers rather than forcing them into a cluster. This is appealing because those outlier days might be exactly the ones you handle specially (maybe they precede reversals or require separate risk management). DBSCAN doesn't need you to set a cluster count, but you do set a distance threshold and minimum points. If your feature space has clear high-density groupings (like most days oscillate in a few typical patterns) and some low-density scattered points (weird days), DBSCAN will capture that. It's one of the methods recommended for deeper exploration of unsupervised clustering in finance [11]. There are related methods like **OPTICS** (which is like an extended DBSCAN for varied densities) too.

- **Hidden Markov Models (HMM):** HMMs deserve consideration for time-series regime detection. An HMM will model the market as switching between a few hidden states, with probabilities of transitioning from one state to another. You might not feed it a high-dimensional feature vector for each day, but rather a univariate or low-dim time series (like returns, or returns and volatility) and it will infer states like "state 1: low-volatility uptrend" vs "state 2: high-volatility downtrend," etc. HMM is *unsupervised* in that you don't label the regimes beforehand – the model learns the most likely hidden state sequence that explains the observed data. This technique has been used to **identify bull vs bear market periods** on indices [1], effectively classifying long historical periods into distinct phases. The benefit of HMM for swing trading is that it accounts for temporal dynamics – e.g. if the market is in a bull state, it might tell you the probability it stays bull vs transitions to bear tomorrow. This can be very powerful for **regime filters**. However, HMMs assume a Markov chain structure and you have to guess the number of states (which you can choose via Akaike/BIC criteria or domain intuition). They also typically use simpler feature inputs (like just returns or returns and some macro indicator) rather than dozens of technical indicators, because modeling high-dim emissions is complex. If your focus is explicitly on **bullish/bearish phase identification** as you initially suggested, an HMM with two or three hidden states is a strong candidate.

- **Self-Organizing Maps (SOM):** SOMs are a neural network approach that projects high-dimensional data onto a 2D grid (a map) while preserving topology (similar patterns end up near each other on the map). They effectively cluster the data but also give a visual layout, which can help in interpretation. SOMs have been applied to financial datasets to **visualize and cluster market conditions** [12]. For example, a SOM might arrange various market days such that one corner of the map represents "bullish trending days" and another represents "bearish crash days," with gradations in between. By looking at where the current day falls on the map, a trader can see which cluster or neighborhood it's in, and possibly identify analogs from history (the map might show that current conditions are closest to, say, late-2018 correction days). This is useful if interpretability and visualization are important to you. SOMs are a bit more involved to set up (and less common recently compared to the other methods above), but they essentially perform an unsupervised clustering (the neurons of the SOM become cluster centroids on the map). They could be a good choice if you want a more exploratory analysis or a visual tool to **find correlations and clusters in multi-dimensional stock data** [12].

- **Autoencoders and Anomaly Detection:** Autoencoders are neural networks that attempt to compress and reconstruct data; by training them on historical "normal" market data, you can detect anomalies when reconstruction error is high. In a swing trading context, an **autoencoder** could learn the typical patterns of your stock's technical indicators. If one day the pattern is so unusual that the autoencoder can't reconstruct it well, that day might be an outlier – possibly indicating a regime change or some catalyst that breaks the usual pattern. Anomaly = potential trading signal (e.g. a very sharp increase in error might precede a volatility spike – a possible trading opportunity or risk-off warning). While autoencoders don't directly cluster, you can also use the lower-dimensional code layer of an autoencoder as feature input to a clustering algorithm (this way you capture non-linear combinations of features in the code). Autoencoders (and related deep models like variational autoencoders or restricted Boltzmann machines) are among the **sophisticated unsupervised techniques suggested for financial pattern extraction** [11]. These can capture complex feature interactions that simpler models might miss. The trade-off is they are less interpretable – you might not know exactly what pattern the autoencoder is keying off, just that it's "unusual." Thus, they're often used in conjunction with more interpretable methods, or as a supplement (e.g. use clustering for regimes, and an autoencoder-based anomaly score to flag particularly extreme conditions within each regime).

In practice, you don't have to pick only one method – you could combine them. For example, you might use PCA to reduce dimensions, **k-means to cluster into broad regimes**, and then a **DBSCAN** on the same data to see if any days fall outside those regime clusters (potential anomalies). Or use an HMM on top of k-means: cluster the data into states and also impose a Markov chain to model transitions (this is like a HMM where emission distributions are defined by those clusters). Many researchers experiment with multiple algorithms to see which yields the most **stable and economically meaningful clusters**. For instance, one recent algorithmic strategy started with k-means, but plans to try DBSCAN, hierarchical, and GMM to possibly improve performance [13]. The "best" method truly depends on what patterns exist in the data: if regimes are well-separated, k-means might be sufficient; if there are subtle gradations, GMM or SOM might reveal more; if temporal switching is key, HMM is designed for that. It's often advisable to begin with a simple method (like k-means) to get a baseline, then refine or replace it if needed.

Balancing Interpretability vs. Performance

When using unsupervised learning for trading, there is a natural tension between **interpretable clusters** and pure predictive performance. On one hand, highly interpretable models (like a simple 3-cluster k-means) let you clearly *see* and *label* what each cluster means – you can confidently say "Cluster A is a bullish consolidation pattern, so I'll buy," which is intuitively satisfying. This transparency helps ensure the strategy makes sense and lets you trust the signals. On the other hand, more complex or less transparent methods (like an autoencoder or a high-dimensional cluster with 10+ regimes) might capture finer patterns that yield better trade signals, albeit at the cost of being hard to label or reason about.

In practice, a **balance** is often best. You might start with interpretable clusters to establish a baseline strategy and understanding, and then gradually incorporate complexity to improve performance. For example, you could use a **coarse regime cluster (bullish/neutral/bearish)** to decide basic positioning (long/hold/short), which is easy to interpret. Within each regime, you could then use a more complex unsupervised signal – say an anomaly detector to fine-tune entry/exit (less interpretable, but if it adds alpha, you use it). This way, you maintain some high-level interpretability while squeezing extra performance where possible.

It's also important to validate that any additional complexity is truly improving outcomes and not just overfitting noise. One advantage of unsupervised learning is that it doesn't directly optimize on the target (no explicit "maximize return" in clustering), so it can avoid some overfitting pitfalls [14]. As one analyst noted, using an unsupervised learner **reduces the chances of over-fitting** since you're not tuning it to predict a specific output, but the **trade-off is that the patterns found may have no real predictive utility** [14]. In other words, you might discover a cluster that looks interesting but doesn't actually lead to better trades. Always check the real-world performance of clusters (via backtesting) to ensure they're not just artifacts. If a more complex method yields only a marginal improvement that doesn't hold up out-of-sample, prefer the simpler, interpretable solution.

Ultimately, **better trade signals** are the goal, but in trading, a slightly less optimal strategy that you understand can often be superior to an opaque "optimal" strategy that might break in unseen ways. You mentioned to use best judgment – here's a reasonable approach: **prioritize interpretability to the extent that it does not severely sacrifice performance**. If an unsupervised model that's easy to interpret (like a 3-state HMM or 4-cluster k-means) gives you a significant edge, that's ideal. If you find that you can substantially boost returns by a more complex clustering (say 10 clusters including some quirky micro-patterns), then by all means consider it – but remain cautious and monitor it in live trading. You can also try to interpret even complex models in creative ways (for example, use SHAP values or feature importance for clusters to understand what drives an autoencoder anomaly).

In summary, **start simple**: identify bullish/bearish/neutral regimes or a few key clusters and get comfortable with how they predict swings. This addresses the core of swing trading – knowing when to go long or short. Then, if needed, iterate towards complexity: maybe your swing trading could benefit from a secondary unsupervised signal that is less interpretable (like a cluster of unusual volume spikes that precedes breakouts). Always keep the feedback loop of testing and validation. By doing so, you ensure that your unsupervised learning approach remains grounded in reality and effective in practice. With this balanced strategy, unsupervised learning can indeed be a powerful ally in swing trading – it can **uncover hidden structures in the market data** that give you an edge, while you, as the trader, impose the necessary judgment and risk management on how to exploit those insights [15] [14].

Sources:

- QuantStart (Halls-Moore), *"K-Means Clustering of Daily OHLC Data."* – discusses using clustering to identify market regimes and aid entry/exit rules [2] [16], and notes advanced clustering methods for noisy financial data [10].
- RobotWealth Blog, *"Unsupervised Candlestick Classification (Part 2)"* – case study clustering daily candlesticks; showed certain clusters of days had significantly higher next-day returns [8] and discusses benefits/trade-offs of unsupervised learning in trading (less overfit, but patterns not guaranteed useful) [14]. Also suggests including technical indicators as features and using hierarchical clustering to determine cluster count [3].
- IKnowFirst Article, *"HMM-Based Classification of Stock Market Stages."* – demonstrates using Hidden Markov Models to identify bull vs bear market regimes unsupervised [1], highlighting the value of regime identification for adjusting trading strategies.
- MDPI Journal (Barradas et al.), *"Identification of Patterns in the Stock Market through Unsupervised Algorithms."* – combined **news sentiment (Google News)** with **market data (Yahoo Finance)**, using PCA and k-means to cluster market conditions [4]. Found that most periods are steady, but one cluster captured high-volatility conditions, illustrating how news+technical data clustering can spot unusual regimes. Also notes that k-means has been used to generate trading signals from fundamentals [17] and that unsupervised patterns can aid trend forecasting by reducing risk [18].
- Parker Carrus (Medium, 2025), *"Unsupervised ML in Algorithmic Trading (Cluster-Relative Strategy)."* – outlines a pipeline mixing unsupervised clustering of stocks by fundamentals with supervised learning. Emphasizes proper scaling of features (log/robust scale) [5] and how unsupervised clustering enhances prediction by isolating relative patterns, improving robustness across regime shifts [19]. Suggests experimenting with various clustering algorithms (DBSCAN, hierarchical, GMM) to improve results [13].
- R-Bloggers (Data-Based Investor), *"Mapping the stock market using Self-Organizing Maps."* – shows how SOMs can visualize and cluster multi-dimensional market data, revealing correlations (e.g. valuation levels, interest rates) and identifying which cluster (on a 2D map) the current market state falls into [12]. This illustrates SOM's usefulness in clustering financial data for insight.

[1] AI-Powered Stock Forecasting Algorithm | I Know First |HMM-Based Classification of Stock Market Stages
https://iknowfirst.com/hmm-based-classification-of-stock-market-stages

[2] [9] [10] [11] [16] K-Means Clustering of Daily OHLC Bar Data | QuantStart
https://www.quantstart.com/articles/k-means-clustering-of-daily-ohlc-bar-data/

[3] [7] [8] [14] Candlestick Classification – Part 2
https://robotwealth.com/unsupervised-candlestick-classification-for-fun-and-profit-part-2/

[4] [6] [17] [18] Identification of Patterns in the Stock Market through Unsupervised Algorithms
https://www.mdpi.com/2813-2203/2/3/33

[5] [13] [15] [19] Unsupervised ML in Algorithmic Trading | by Parker Carrus | Jun, 2025 | Medium
https://medium.com/@carrusparker/self-organization-in-algorithmic-trading-3086dfa26916

[12] Mapping the stock market using self-organizing maps | R-bloggers
https://www.r-bloggers.com/2018/08/mapping-the-stock-market-using-self-organizing-maps/