ChatGPT

Using Machine Learning to Filter Stocks for Swing Trading

Swing trading typically involves holding positions for a medium-term period (on the order of weeks). To **use machine learning (ML) for filtering stocks** as swing trade candidates, we need to build a pipeline that gathers relevant data, engineers predictive features (technical indicators, sentiment, fundamentals), trains an ML model to forecast price movement over the desired horizon (e.g. ~3–4 weeks), and then uses the model's output to identify entry and exit points. Below, we outline a comprehensive approach, **with a configurable time frame** (e.g. you can adjust the prediction horizon from say 2 weeks to 4 weeks as needed).

Problem Definition and Time Frame

First, clearly define the prediction goal and swing trading horizon. Here, the goal is to **predict price movement and identify entry/exit points** for trades that last roughly **3–4 weeks** (around 15–20 trading days). This horizon will be a parameter in the model pipeline, so you can easily configure it to different lengths (for example, 10 days or 30 days) as needed. Essentially, the ML model will learn to forecast whether a stock's price is likely to **increase (or by how much)** over the next *N* days, where *N = 15–20* by default (your swing trade period). By predicting this, the model can **filter stocks** – highlighting those with the most promising expected movement – and thereby suggest **entry candidates**. The exit strategy can be tied to the same horizon (e.g. exit after N days) or triggered by the model's signals (more on this later).

Defining the Target (Label): You need to translate the trading objective into a machine learning target. Two common approaches are:

- **Regression formulation:** predict the **percentage price change** over the next N days (3–4 weeks). For example, the model outputs a forecast like +5% or –3% for the coming period.
- **Classification formulation:** predict a **binary outcome** (or multi-class) such as "Will the stock go **up** or **down** in the next N days?" (possibly with a threshold, e.g. up by at least +2%). A binary label could be 1 if the stock's price N days ahead is higher than today (or above a certain outperformance threshold) and 0 if not.

Each approach has merits. A regression gives more nuanced prediction (magnitude of movement), while a classification simplifies to directional signal. Given that the goal is to filter and pick top candidates, a **regression model whose outputs you can rank** (from most positive expected return to most negative) is useful for selecting the best stocks to long or short. Alternatively, a classification model could directly tell you which stocks are likely "winners" vs "losers" in the next few weeks. **Importantly, the horizon N should be configurable** – you can set N=15 trading days (approximately 3 weeks) or N=20 (4 weeks), etc., and regenerate the training labels accordingly. This flexibility lets you adjust the strategy's holding period.

Data Sources and Collection

With the prediction target set, gather historical data for both training the model and feeding it live updates. **Yahoo Finance** (`yfinance` in Python) is a convenient source for historical stock data, and you indicated you have access to it. You can use Yahoo Finance to pull **daily price data** (Open/High/Low/Close prices and Volume) for the stocks of interest. This will be the foundation for computing technical indicators and price-based features. Yahoo Finance also provides some fundamental data (financial ratios, earnings dates, etc.) which we can leverage for fundamental features. Steps for data collection:

- **Historical Prices:** Use `yfinance` to download daily OHLCV data for each stock you want to consider (e.g., all S&P 500 stocks or a predefined watchlist). Ensure you have sufficient history (several years if possible) so the model can learn patterns. For a 3–4 week horizon, you might need data spanning multiple market cycles (the more the better, as long as it's relevant and cleaned).
- **Corporate Fundamentals:** From Yahoo or other sources (like Quandl or your brokerage's API), gather fundamental metrics for each stock. This might include valuations (P/E ratio, P/B, dividend yield), growth metrics (EPS or revenue growth rates), financial health indicators (debt/equity, profit margins), etc. These typically update quarterly, so you can merge the latest known fundamentals into your dataset for each date. Fundamental data can give context about a company's intrinsic strength or value.
- **News and Sentiment Data:** For sentiment analysis, you will need a source of news or social data. Yahoo Finance provides news headlines for stocks, but you might need to use an API or web scraping to collect those systematically. Another approach is using platforms like Alpha Vantage or NewsAPI for financial news, or even social media sentiment from Twitter/StockTwits. The idea is to retrieve news articles or headlines relevant to your stocks on each day. Once collected, these news titles can be fed into a sentiment analysis model to quantify sentiment.

Data Frequency: Since swing trades are multi-day holds, daily data is a reasonable frequency for features. You don't necessarily need intraday data (which would explode feature volume and is more relevant to day trading). Daily closing prices and daily news sentiment should suffice. Ensure all data is aligned by date (e.g., features on day *t* will be used to predict the return from *t* to *t*+*N*).

Feature Engineering (Technical, Sentiment, and Fundamental Indicators)

With raw data in hand, engineer a rich set of features that may have predictive power for 3–4 week movements. You have indicated interest in **technical indicators**, **news sentiment**, and **fundamental data** – this comprehensive feature set aligns with what research calls a *"mixed approach"* (combining technical, fundamental, and sentiment analysis) [1] [2]. Such hybrid approaches aim to **leverage the strengths of each type of analysis** and have shown improved predictive accuracy in studies (albeit with increased complexity) [3]. We will break down feature types:

- **Technical Indicators:** These are computed from historical price/volume data and aim to capture trends, momentum, volatility, and other patterns. Common technical features suitable for a few-week outlook include:
- *Trend indicators:* Moving averages (e.g., 10-day, 20-day SMA/EMA), moving average crossover signals, etc. These gauge the stock's recent trend. (For instance, the slope of a 20-day moving average might indicate upward or downward momentum.)

- *Momentum oscillators:* Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Stochastic Oscillator (K%, D%), Williams %R, etc., which signal overbought/oversold conditions or momentum shifts. **Example:** RSI measures recent gains vs losses on a scale 0–100; high RSI (>70) might mean overbought, low (<30) oversold. MACD compares two EMAs to find trend changes.
- *Volatility indicators:* Average True Range (ATR) for recent volatility, Bollinger Bands (which use moving average ± volatility) indicating price extremes. Volatility can help predict if a stock is in a quiet or explosive phase.
- *Volume indicators:* On-Balance Volume (OBV), volume moving averages, volume spikes. Volume often precedes price; OBV in particular accumulates volume based on up vs down days and was found to positively impact predictive strategies [4] [5].
- *Others:* Indicators like **rate of change** (momentum over N days), or pattern-based features (e.g., number of consecutive up days, etc.), could be included.

You'll likely compute many of these for each stock each day. It's often useful to include multiple time window variants (e.g., 5-day RSI, 14-day RSI) and later let the model decide which are useful. *Note:* Research has noted that some single technical signals alone may perform poorly in isolation [6], but as input features to ML they may still be valuable when combined properly. In one study using **MA, MACD, RSI, Stochastic (KD), and OBV** as features, an ML model (XGBoost) outperformed simple indicator-based trading rules [7]. The takeaway is that an ML algorithm can synthesize multiple technical indicators simultaneously to find patterns that individual indicators (used in a naive way) miss.

- **News Sentiment Features:** Incorporating news sentiment can add a **forward-looking signal** based on market psychology and information flow. To include this:
- Perform **Natural Language Processing (NLP)** on news headlines or articles for each stock. For example, you can use a sentiment analysis tool like **VADER** (a lexicon-based sentiment scorer) or fine-tuned models like **FinBERT** (a BERT model trained on financial text) to rate news as positive/negative. One simple approach is shown by Wang (2020) where they *used VADER to get a "compound" sentiment score for each headline and then summed the scores for each day* [8] [9]. This yields a daily sentiment index for the stock (positive if good news dominated, negative if bad news).
- You could also create features like "sentiment over last 3 days" (a rolling average of daily scores) or count of positive vs negative news items in the last week, etc.
- If social media sentiment is available (say, number of bullish vs bearish tweets), that can be additional features.
- **Important:** Align the sentiment feature with the date (news on day *t* could influence price on day *t* and beyond). If no news on a given day, you might fill sentiment as 0 (neutral) [10].

Academic research is mixed on how much sentiment data improves predictions – some have found **news sentiment indices can enhance predictive power for stock moves** [11] [12], while others found a particular sentiment metric made little difference perhaps because news was already priced in intraday [13]. The efficacy may depend on your implementation and the market; regardless, it's a valuable feature to test given that **investor sentiment often drives short-term price swings**.

- **Fundamental Features:** These capture the underlying value and financial condition of companies. Although fundamentals evolve slowly (quarterly reports, etc.), they can help an ML model distinguish, for example, a fundamentally strong company whose dip might be a buying opportunity from a fundamentally weak one. Features to consider:
- Valuation ratios like **P/E (price-to-earnings)**, **PEG ratio**, **P/B (price-to-book)**, dividend yield – could influence how much mean-reversion or momentum a stock has (a very undervalued stock might have a higher chance to bounce back).

- Growth metrics: revenue or earnings growth rates, profit margins, **ROE**, etc., indicating business momentum.
- **Earnings or events calendar:** a binary feature if the company is about to report earnings in the next N weeks. Earnings announcements can cause swing-sized moves. Similarly, features for other upcoming events (analyst days, product launches, etc., if available).
- Sector or industry indicators: One-hot encode the sector, or include sector average performance as a feature (to account for sector momentum).
- Macro context: Since 3–4 weeks is not too long, broad market indicators (S&P 500 trend, volatility index VIX, interest rates) could be included to inform if overall conditions are bullish or bearish.

Fundamentally, a swing trade might benefit from *fundamental confirmation* (e.g., bullish technical signal on a stock that's undervalued could be higher conviction). Including these as features allows the model to learn such interactions if they matter. Keep in mind fundamental data changes infrequently, so you'll have forward-fill or keep the last known value until a new report comes out.

Feature Processing: Once you compute these features, it's good practice to **scale or normalize** them for the model (especially for algorithms like neural networks or distance-based methods). For tree-based models (Random Forest, XGBoost), scaling is less critical but still can't hurt. Given your expertise, you can also explore **feature selection** or dimensionality reduction if the feature list grows very large. For example, a Random Forest's feature importance or using PCA on highly correlated technical indicators could simplify the input space. However, be careful not to leak future data in this process – perform any selection based on the training set only.

Model Selection and Training

Now to the machine learning model itself. Since you have a background in ML and quant, you can consider a range of models from simple to complex:

- **Baseline models:** Start with something like a **Logistic Regression** (for classification) or **Linear Regression** (for regression target) as a sanity check. This sets a baseline accuracy and ensures the pipeline is working. These linear models might not capture nonlinear market patterns well, but they are quick to run and provide interpretability (you can see which features have what coefficients, giving insight into relationships).
- **Tree-based ensembles: Random Forests** and **Gradient Boosted Trees (XGBoost or LightGBM)** are powerful for tabular financial data. They can capture nonlinear interactions and typically handle mixed feature types well. Notably, the study from 2023 found that **XGBoost-based models outperformed traditional indicator-only strategies** under certain conditions [7]. Tree models also give feature importance scores, which can be informative (e.g., you might discover that **volume or OBV is a top predictor** of 3-week returns, as found in that study [5]).
- **Neural networks:** If you have a lot of data, you could use deep learning. Two approaches: (1) Treat it like any other regression/classification on tabular data – e.g., a Feed-Forward Neural Network that takes the feature vector for day *t* and outputs prediction for t+N. This might capture some nonlinearities but doesn't inherently use sequence info. (2) Use sequence models – for example, an **LSTM or GRU** that takes in a window of past days' features as a time series and predicts the future. For swing trading, a sequence of the last, say, 30 days could be input to predict the next 20 days trend. RNNs or CNNs can potentially learn temporal patterns (like chart patterns) beyond fixed indicators. In the literature, hybrids like CNN+LSTM or attention mechanisms have been applied to capture price sequences [14] [15]. These are more complex and data-hungry, so only pursue if you have a large dataset and the problem seems to warrant it.

- **Other advanced models:** As a quant, you might consider **Support Vector Machines**, which have been used in stock prediction research as well [16]. Or even specialized approaches like **reinforcement learning** for timing entry/exit. However, given the focus on filtering stocks and a fixed horizon, supervised learning is the straightforward path. Some cutting-edge research even uses **GANs and autoencoders** in hybrid models [17] [18], but those may be beyond our scope here.

For our pipeline, a sensible choice is to **start with a gradient boosted tree (XGBoost)** or a Random Forest, as these often perform well out-of-the-box on structured financial data. You can then later test a neural network if desired.

Training Process:

1. **Prepare training examples:** Typically, you will form examples as follows – for each stock and each date *t* in the historical data (except the last N days which can't form a full target horizon), compute all features using data up to *t*. Then compute the target using data from *t* to *t+N* (e.g., percent change from close_t to close_t+N for regression, or class label based on that change). This way each example (*features*, *label*) corresponds to "what we knew on day *t*" -> "what happened by day *t+N*". Do this for all stocks and all days across your history. This will yield a large training set. **Be careful to avoid any lookahead bias** – e.g., when you compute a moving average on day *t*, ensure it's truly using past data, not creeping into day *t+1*, etc. Also, if you include fundamental data, only use info that was available at time *t* (not a revised value that came later).
2. **Training/Validation Split:** Since this is time series data, do not shuffle randomly for train/test split. Instead, use an approach where you train on older data and test on newer data (to mimic future predictions). For example, train on 2015–2019 data, validate on 2020, test on 2021–2022, etc. This respects the chronological order. If you have a lot of data, you can also do **rolling cross-validation**. Scikit-learn's `TimeSeriesSplit` is useful – it creates folds that respect order (each fold might train on an earlier period and test on the next period). This technique was used, for instance, in an example combining technical and sentiment features to avoid leaking future information [19] [20].
3. **Model Training:** Train the chosen model on the training set. If using scikit-learn or XGBoost, you can perform hyperparameter tuning (e.g., grid search or random search) within each training fold of a TimeSeriesSplit. Since you are experienced, you might also consider using techniques like **walk-forward optimization** (retraining model periodically on expanding window and testing on the next period) to mimic how the model would be updated in real trading.
4. **Performance Metrics:** For classification, look at metrics like accuracy, precision/recall for the "up" class (since perhaps only a subset of stocks will be labeled as positive at any time), and the confusion matrix. For regression, look at mean absolute error or directional accuracy (what fraction of time you correctly predicted up vs down). However, **the ultimate metric is trading performance**: you can simulate how a strategy based on the model would do (more in next section). If the model's pure predictive metrics are only slightly better than random, it could still be useful when combined with a sensible strategy (and likewise, a high predictive R^2 might still fail after transaction costs if the signals are small).

In practice, expect that predictive performance will be modest – many studies find that stock direction accuracy might hover not far from 50–60% even with complex models [21]. The key is that even a slight edge, when exploited consistently with good risk management, can yield profits. Always evaluate out-of-sample results; if a model performs no better than chance on fresh data, you'll need to revisit features or model complexity (e.g., simpler models might generalize better in some cases [22]).

Backtesting the Strategy and Evaluation

Once the ML model is trained and you believe it has some predictive power, integrate it into a **backtesting framework** to see how it would have performed in selecting stocks for swing trades historically. This involves using the model's predictions to make trading decisions in a simulated environment:

- **Simulation Procedure:** Step through historical data (ideally date by date). On each day (or each week, depending on how frequently you want to redeploy the strategy), do the following:
- Compute features for all candidate stocks as of that day.
- Have the model predict the next N-day return or probability of rise for each stock.
- **Select stocks to trade:** e.g., pick the top *K* stocks with highest predicted returns or highest probability of positive move (the filtering step). *K* could be configurable (maybe you buy the top 5 or top 10 predictions). You might also set a cutoff (e.g., only trade if model predicts > +5% move or probability > 0.7 to avoid low-conviction bets).
- Enter simulated positions on those stocks at next day's open (or that day's close, depending on how you design the timing).
- Hold the positions for N days (3–4 weeks) or until an exit condition triggers (see **Entry/Exit** section below for options).
- Track the returns of these trades, including transaction costs if applicable.
- Repeat for the next trade date. There will be overlaps if you are holding for N days – this means at any given time you might have a portfolio of several trades in different stages. This is fine; just ensure you're accounting for that in the simulation (and not accidentally trading the same capital twice if you simulate portfolio constraints).
- **Evaluation Metrics:** After the simulation, measure:
- **Cumulative return** of the strategy vs. a benchmark (e.g., compare to S&P 500 return over the same period).
- **Annualized return** and **volatility**, **Sharpe ratio** (risk-adjusted return).
- **Max drawdown** to see worst-case losses.
- **Win rate** (percentage of trades that were profitable) and **average win vs average loss**.
- **Exposure** (how many positions held on average, to gauge capital usage).

If the model is good at filtering winners, you should see a positive performance and ideally outperformance over just holding an index or random stock picks. If not, analyze where it fails: are the predictions systematically off during certain market regimes? Perhaps add features or use an ensemble of models to cover different conditions.

- **Feature/Signal Importance:** Using the trained model, check which features are most influential. For example, XGBoost can give you an importance ranking. You might discover, say, that **news sentiment 3 days ago** and **a spike in volume** are strong predictors in your model. This can validate or inspire tweaks to the strategy (e.g., if momentum indicators aren't showing up as important, maybe the model is focusing more on mean-reversion cues, etc.). Keep an eye on this especially when you adjust the horizon – different features might matter for 5-day prediction vs 20-day prediction (longer horizon might rely more on fundamentals/trend, shorter on technical noise).

- **Robustness Checks:** It's wise to test the model on different periods (e.g., did it work in both bull and bear markets?), and possibly do a **paper trading** or forward-test on live data without real money to see if it behaves as expected. Avoid overfitting to backtest results – if you fine-tune parameters to maximize historical Sharpe, you might end up with a model that doesn't generalize. Aim for a balance between performance and simplicity to ensure robustness.

Using the Model’s Signals for Entry and Exit

Identifying **entry and exit points** is crucial for a complete swing trading strategy. The ML model’s output helps with **entries** by telling you *which stocks to enter and when*. For **exits**, you can use a few approaches:

- **Fixed Holding Period Exit:** Since the model is predicting over N days, a straightforward approach is to **exit after N days**. For example, if today the model signals stock XYZ is a top pick for the next 20 days, you buy it and plan to sell 20 trading days later (approximately 4 weeks) at the close. This aligns with the model’s horizon. Many backtests in research use this method (enter and exit strictly based on the period) to evaluate performance cleanly [23] .
- **Target/Stop Exit:** You can set a profit target and stop-loss based on the model’s predicted move. If the model predicted +10%, maybe set a take-profit at +8% and a stop-loss at –4% (for a 2:1 reward/risk). If either hits before 3–4 weeks, you exit the trade. This way, if a stock rallies quickly as predicted, you lock in profit early; if it goes against you, you cut loss before the full period. The specific levels can be optimized or at least informed by the model’s confidence.
- **Dynamic Model Re-rating:** Because you can update the model’s predictions as new data comes in, you might choose to **re-evaluate positions periodically**. For example, every week, run the model again. If a stock you’re holding no longer appears as a top candidate (or the model flips to predicting a drop), that could be a signal to exit early. Essentially, the model provides an *ongoing filter*. However, beware of excessive trading – frequent churn might erode gains with transaction costs. Since your horizon is a few weeks, updating the signals weekly or bi-weekly is probably sufficient.
- **Technical Indicator Confirmation:** Some traders combine ML signals with traditional charting for timing. For instance, you use ML to shortlist promising stocks, but then only enter when a specific technical trigger occurs (say, the stock’s price breaks above a resistance or a moving average crossover confirms an uptrend). Similarly, for exits, you might watch a trailing stop (like price crossing below a moving average) to decide an exit if the model isn’t explicitly giving an exit. This hybrid approach can sometimes improve timing and aligns with human intuition about entry/exit points (e.g., avoid entering right before a known resistance level, etc.).

The key is to design the entry/exit rules and **stick to them** in the backtest to fairly judge performance. Given that you aim to predict both *movement* and *entry/exit*, your ML model primarily aids the *selection and entry* decision (when a positive prediction occurs), while the *exit* can be a predetermined policy (time-based or rule-based).

For example, a concrete strategy could be: “Each Monday, use the model to predict 3-week returns for all stocks. Buy the 5 stocks with highest predicted returns at Monday’s close. Hold each for 15 trading days (3 weeks), then sell. Manage risk with a 5% stop-loss on each. Repeat weekly.” You can adjust this framework (maybe daily selection instead of weekly, different number of stocks or stop criteria) based on what yields the best trade-off between return and risk in backtests.

Implementation Considerations

To implement this in practice (especially since you have an ML engineering background):

- **Pipeline Automation:** Set up a pipeline (possibly using Python scripts or Jupyter notebooks initially, then airflow or similar for automation) that can:
- Fetch the latest data (daily) from Yahoo Finance for your stock universe.
- Compute the technical indicators and update sentiment/fundamental features as needed.

- Load your trained model (or retrain it periodically with rolling windows).
- Generate predictions for each stock.
- Produce a “score” or ranking that can be used to filter stocks for trading.
- **Configurability:** Parameterize the key settings – e.g., `HORIZON_DAYS = 20` for a 4-week horizon (you can easily change this to 10 or 60 to experiment with shorter/longer swings). Similarly, have a parameter for how many top picks to select, etc. This makes your pipeline flexible in testing different swing trade styles.
- **Model Update Frequency:** Decide whether you will **retrain the model** regularly with new data (to adapt to recent market conditions). A common approach is rolling retraining – e.g., every month or quarter, retrain using the last few years of data. This helps the model not become stale. With a highly configurable pipeline, you can experiment with retraining frequency as well.
- **Frameworks and Tools:** Since you’re comfortable with frameworks like scikit-learn, TensorFlow, PyTorch – use what fits best:
- For many tabular features, scikit-learn with XGBoost (via `xgboost` or `sklearn.ensemble.GradientBoostingRegressor`) or LightGBM can be quick to implement. You can use `GridSearchCV` or `RandomizedSearchCV` with `TimeSeriesSplit` for hyperparameter tuning [24] .
- For deep learning, PyTorch or TensorFlow give you flexibility to build custom LSTMs or hybrid models if needed. PyTorch Lightning or TensorFlow’s high-level APIs could help manage training loops for time series.
- Libraries like **ta-lib** (Technical Analysis library) or **pandas-ta** can simplify computing technical indicators. NLP libraries like **NLTK (for VADER)** or HuggingFace Transformers (for BERT models) can be used for sentiment. Ensure to cache or precompute what you can for efficiency (e.g., computing all technical features in one pass for the whole dataset).
- **Performance & Overfitting:** Always be mindful of overfitting – with many features and relatively limited stock data (even a decade of daily data is only ~2500 samples per stock, and if you pool stocks, you have more but also potentially high variance between stocks), a complex model can overfit noise. Techniques like cross-validation, regularization, and testing on truly out-of-sample periods (e.g., year 2023–2024 if not used in training) will indicate if your approach generalizes. It’s often useful to start simple (few features, simple model) and add complexity gradually, watching if metrics actually improve.

Conclusion and Further Ideas

In summary, using ML for swing trade stock filtering involves **marrying traditional trading analysis with data science**: gather historical price data (Yahoo Finance), augment it with technical indicators, news sentiment analysis, and fundamental metrics, then train a predictive model (e.g. XGBoost, neural network) to forecast medium-term price movements. Research and practical examples show that incorporating diverse feature types (price trends, volume, news, fundamentals) can improve prediction accuracy by leveraging different information dimensions [1] [3] . The model’s predictions then guide which stocks to **enter** (those with the most bullish outlook, for instance), and you implement a clear **exit plan** (time-based or signal-based) to complete the strategy.

A few additional tips:

- **Start with a smaller universe** of stocks to develop the strategy (maybe 50–100 stocks with ample liquidity). This will make data management and model training easier. You can expand later.
- **Watch out for market regime changes** – a model trained on a mostly bullish period might not handle a bear market well. Consider training over various conditions or include regime indicators (volatility regime, etc.) as features.

- **Validate on individual stocks vs pooled:** Sometimes a model that works on average might still pick a few stocks that behave poorly. Check if certain sectors or stocks are giving the model trouble (you might then decide to exclude those or handle them differently).
- **Continuous Learning:** As new data comes in each day, update your features and possibly retrain periodically so the model adapts. Markets evolve, and models need to be refreshed.

By following this plan, you will effectively create an **ML-powered stock screener** tailored to swing trading. It will sift through lots of candidates and surface those with favorable 3–4 week prospects, grounded in data patterns it has learned. From there, your trading rules (which can be coded into the pipeline as well) will execute the entries and exits. This integration of ML predictions with a trading strategy is at the cutting edge of quant trading techniques, and while it requires careful validation, it holds the promise of uncovering opportunities that a manual approach might miss.

Sources:

- Lin et al. (2023) – *Application of Machine Learning with News Sentiment in Stock Trading Strategies*: Demonstrated using technical indicators (MA, MACD, RSI, Stochastic %KD, OBV) and a text-mined **news sentiment ratio** as features for ML models (SVM, XGBoost, RNN, LSTM) [25]. Notably, their XGBoost models outperformed some indicator-only strategies [7], though their particular news sentiment feature didn't significantly improve accuracy [13].
- Wang (2020) – Experiment on **predicting stock trends with news sentiment and technical indicators**: Implemented daily feature extraction (RSI, SMA, stochastic oscillator) and used **VADER sentiment** scores summed by day [8] [9]. Highlighted the importance of using TimeSeriesSplit for cross-validation to avoid lookahead bias [19] [20].
- Saberironaghi et al. (2025) – *Stock Market Prediction Using ML and DL: A Review*: Provides an overview of combining **technical, fundamental, and sentiment analysis** in hybrid models. It notes that a mixed approach leveraging all three can improve prediction accuracy by harnessing multiple data sources [1] [2]. For example, integrating technical indicators (MACD, TEMA) with ML models (SVR, Random Forest) to generate trading signals showed improved accuracy in one study [3].
- Research findings on **sentiment**: Various studies indicate that news sentiment can have predictive power for stock moves [11], especially when using machine-learning-based NLP to accurately gauge sentiment [12]. However, the impact can depend on market conditions and how quickly news is absorbed [26] [27].
- Practical swing trading knowledge: Swing traders often use technical patterns and indicators for entries/exits, which aligns with our feature set (e.g., RSI, moving averages for entry signals) [28] [29]. Our ML approach essentially generalizes and tests these signals quantitatively, potentially giving an edge in consistency and the ability to process more information (like news and fundamentals) than a human could manually.

---

[1] [2] [3] [14] [15] [17] [18] Stock Market Prediction Using Machine Learning and Deep Learning Techniques: A Review
https://www.mdpi.com/2673-9909/5/3/76

[4] [5] [6] [7] [11] [12] [13] [16] [23] [25] [26] [27] Contents
https://www.sciedupress.com/journal/index.php/ijfr/article/download/23538/14833

[8] [9] [10] [19] [20] [21] [22] [24] Predicting the Stock Trend with News Sentiment Analysis and Technical Indicators: A Research Review and A Reproduction Example | by Kaichong Wang | Medium
https://medium.com/@kaichong.wang/predicting-the-stock-trend-with-news-sentiment-analysis-and-technical-indicators-f5f0ca2e76b8

28 Swing Trading Strategies Explained - Bookmap
https://bookmap.com/blog/swing-trading-explained-strategies-for-capturing-short-term-price-swings

29 Introduction to Swing Trading - Investopedia
https://www.investopedia.com/trading/introduction-to-swing-trading/