SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)

Symbiosis School for Online and Digital Learning

A
PROJECT REPORT
ON

Comparative Analysis of Time Series Forecasting Models for Financial Markets: Statistical and Machine Learning Approaches with a Contrarian Candlestick Framework

SUBMITTED

BY

Prathik [redacted]

M.Sc. (Data Science)

Semester – IV

July – 2023 Batch

DECLARATION

The project report entitled **"Comparative Analysis of Time Series Forecasting Models for Financial Markets: Statistical and Machine Learning Approaches with a Contrarian Candlestick Framework"** Submitted to Symbiosis International University (Deemed University), Symbiosis School for Online and Digital Learning, Pune in partial fulfilment of the requirement for the award of the degree of M.Sc. (Data Science) is an original work carried out under the guidance of **Dr. Pallawi Bulakh**. The matter embodied in this project is a genuine work done by me to the best of my knowledge and belief and has not been submitted before, neither to this University nor to any other University for the fulfilment of the requirement of any course of study.

Signature [redacted]

[redacted]

Name [redacted]

Prathi

PRN

[redacted]

Date: 10th September 2025.

ACKNOWLEDGEMENT

It gives us immense pleasure to present this report on **"Comparative Analysis of Time Series Forecasting Models for Financial Markets: Statistical and Machine Learning Approaches with a Contrarian Candlestick Framework."** The dissertation work has brought out significance of sincere efforts, teamwork, guidance and support that makes a dissertation work successful. I take this opportunity to acknowledge the guidance and encouragement of all those with whom I have interacted during the course of this Project.

I would like to thanks to our Director of the Symbiosis School for Online and Digital Learning **Dr. Parimala Veluvali** for their support and encouragement. I would like to thanks to my project guide **Dr. Pallawi Bulakh** for valuable suggestions during the Project work.

Signature [redacted]

[redacted]

Name [redacted]

Prathik

PRN N

[redacted]

Date: 10th September 2025.

SYMBIOSIS INTERNATIONAL
(DEEMED UNIVERSITY)

Symbiosis School for Online and Digital Learning, Pune

CERTIFICATE

This is to certify that the Project Report titled **"Comparative Analysis of Time Series Forecasting Models for Financial Markets: Statistical and Machine Learning Approaches with a Contrarian Candlestick Framework"** submitted by **PRATHIK** [redacted] student of M.Sc. (Data Science) Semester - IV of Symbiosis School for Online and Digital Learning, Pune, India, affiliated to Symbiosis International (Deemed University) during the academic year 2024-25, in partial fulfillment of the requirements for the award of the degree of M.Sc.(Data Science).

Signature of the Guide

Place:

Date:

INDEX

CHAPTER 1
INTRODUCTION

1.1 Title

Comparative Analysis of Time Series Forecasting Models for Financial Markets: Statistical and Machine Learning Approaches with a Contrarian Candlestick Framework.

1.2 Abstract

Forecasting financial markets is a complex task due to non-stationary patterns and the influence of external macroeconomic factors. This study presents a comparative analysis of multiple forecasting approaches, including traditional statistical models (ARIMA, SARIMA), machine learning techniques (Random Forest, XGBoost), and deep learning architectures (Hybrid CNN–LSTM), for predicting price movements. In addition, a novel contrarian psychological variable is introduced into one of the tested models to incorporate behavioral finance perspectives into the forecasting process. The analysis employs a publicly available dataset of Bahrain's All Share Index (BAX). Lagged features are engineered to improve predictive accuracy, and models are evaluated using time series cross-validation with performance metrics such as root mean square error (RMSE) and mean absolute error (MAE). Comparative results across markets are examined to assess the transferability and robustness of the models, with potential applications in international finance and global portfolio optimization. The proposed integration of behavioral and quantitative methods aims to enhance the realism of market forecasts by accounting for trader psychology alongside traditional time series patterns.

1.3 About Project

Global financial markets play a central role in the functioning of modern economies by channeling savings into investments, facilitating liquidity, and enabling wealth creation. Indices such as the Bahrain All Share Index (BAX) serve as a key barometer of economic performance and investor sentiment. However, markets are highly sensitive to macroeconomic variables, policy changes, and behavioral dynamics, resulting in unpredictable and volatile price movements. For both individual investors and institutions, the ability to forecast short-term and medium-term index values has become an indispensable tool for risk management, portfolio optimization, and informed decision-making.

Forecasting financial market indices has long been a subject of extensive research due to its practical implications in investment decision-making, risk management, and policy

formulation. However, market prices are inherently non-linear, volatile, and influenced by both quantitative and behavioral factors, making accurate prediction a challenging task. This project titled “Comparative Analysis of Time Series Forecasting Models for Financial Markets: Statistical and Machine Learning Approaches with a Contrarian Candlestick Framework”, undertakes a comprehensive analysis of statistical, machine learning, and deep learning models for time series forecasting.

The novelty of this study lies not only in identifying the best-performing model using error metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), but also in integrating psychological trading behavior variables. Specifically, it incorporates contrarian patterns derived from candlestick misinterpretations by retail investors, an element often overlooked in conventional forecasting models. By doing so, the project addresses both technical accuracy and behavioral realism in financial market predictions.

In addition to the psychological component, the project emphasizes the necessity of a comparative framework across multiple modeling paradigms. Traditional statistical models such as ARIMA and SARIMA excel in capturing linear dependencies, whereas ensemble machine learning approaches like Random Forest and XGBoost account for non-linearities, and deep learning models such as LSTM and GRU are capable of learning long-term temporal dependencies. By benchmarking these models across different markets and datasets, the project provides a holistic understanding of their strengths, limitations, and context-specific applicability.

Furthermore, the project acknowledges the growing importance of behavioral finance in explaining market anomalies. Retail traders, who constitute a significant portion of market participants in emerging economies, often rely on simplistic interpretations of candlestick patterns and chart signals. Their collective actions can reinforce predictable biases, creating opportunities for contrarian forecasting strategies. By explicitly modeling such behavioral tendencies, the project extends beyond conventional time series forecasting and contributes towards bridging the gap between quantitative finance and behavioral economics.

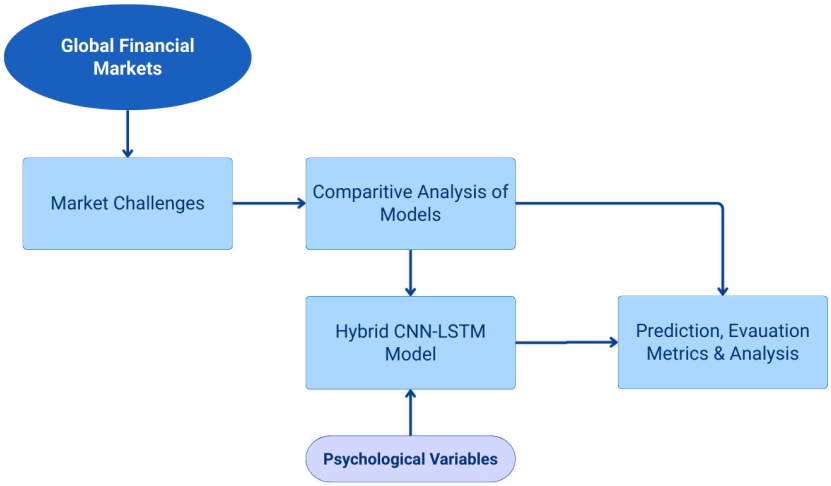

Figure 1.1: Overview of the Project

1.3.1 Understanding Market Psychology and the Contrarian Strategy

Stock markets are not governed solely by rational expectations or fundamental valuations; instead, they are heavily influenced by investor psychology and collective behavioral biases. Retail investors, in particular, often rely on widely circulated technical charting methods, especially candlestick pattern analysis to guide their trading decisions. While these visual patterns are simple to interpret and easily available across textbooks, brokerage platforms, and financial media, their very popularity creates predictable crowd behavior.

The Securities and Exchange Board of India (SEBI) reported in 2024 that 93% of individual traders in equity Futures & Options incurred losses between FY22–FY24, with cumulative losses exceeding 1.8 lakh crore. This staggering statistic underscores that the vast majority of retail traders, despite having access to candlestick literature and mainstream technical guidance, consistently lose money. One plausible explanation is that if most market participants act on the same predictable signals, the market tends to move against the majority. In such cases, a contrarian approach, deliberately acting opposite to the standard interpretation, can produce higher success rates, as it exploits behavioral inefficiencies.

To illustrate, consider the Doji candlestick pattern, which traditionally signifies market indecision and potential reversal. If an upward trend culminates in a Doji, conventional candlestick interpretation suggests that bullish momentum is weakening and that traders should prepare for a downward reversal (i.e., initiate a short position). However, empirical experience and anecdotal evidence suggest that markets frequently continue upward movement despite the Doji, precisely because a large segment of retail traders expects a reversal and positions

themselves accordingly. By taking the inverse action (buying instead of shorting), one can potentially capitalize on the crowd's consistent misjudgment.

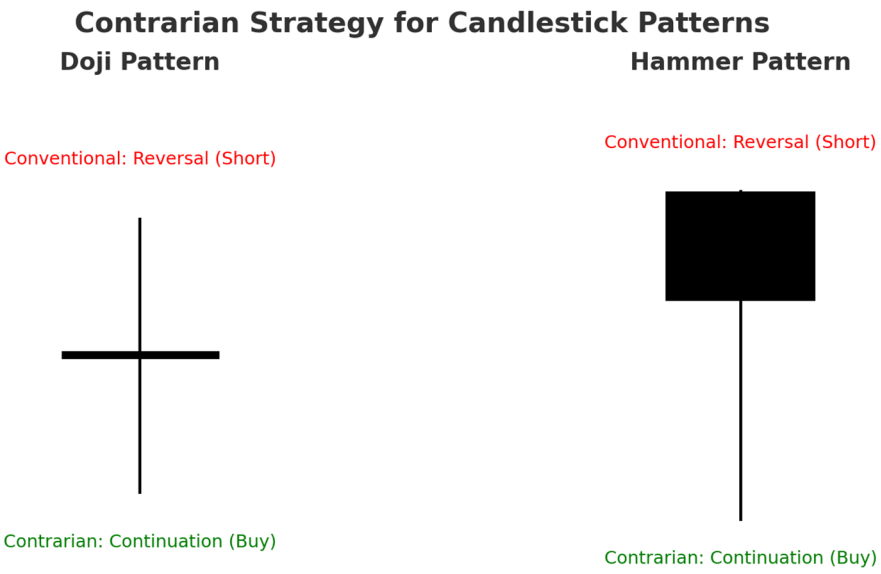

Figure 1.2: Contrarian Strategy for Candlestick Patterns

This project leverages such insights by integrating an example of a contrarian psychological variable into forecasting models. While not every candlestick misinterpretation leads to the opposite outcome, statistical evidence suggests that acting against widely followed retail strategies yields a better than random edge. This introduces a unique blend of behavioral finance and time series modeling, transforming a purely quantitative forecast into one that accounts for real-world trading psychology.

Figure 1.3: Screenshot showing price taking contrarian movement in Sensex Index

1.4 Problem Formation:

The problem formation can be broken down into following steps:

1.4.1 Problem Identification:

Accurate forecasting of financial markets remains a critical yet highly complex task, primarily due to the non-linear and non-stationary nature of price movements. Traders and investors rely heavily on various models to predict future prices, but no single model consistently outperforms others across different market conditions. Moreover, traditional models often ignore human behavioral patterns, which significantly influence short-term market dynamics, especially in retail-dominated markets.

1.4.2 Research Gap:

While existing studies compare forecasting, models based on statistical and machine learning techniques, limited research considers the effect of psychological trading behaviors, particularly those contributing to common retail losses on model performance. For instance, retail traders often misinterpret chart signals (e.g., assuming reversals on Doji candles), which leads to predictable behavioral patterns. These patterns, if captured and modeled appropriately, could enhance prediction accuracy.

1.4.3 Scope of the Study:

The study compares models like ARIMA, SARIMA, Random Forest, XGBoost, Hybrid CNN-LSTM etc. for time series forecasting on Bahrain's (BAX) financial index. It also evaluates one model, the Hybrid CNN-LSTM, with and without the inclusion of an engineered psychological variable derived from trading behavior patterns and SEBI's published investor loss data.

1.4.4 Objectives:

1 To perform a comparative analysis of statistical, machine learning and deep learning forecasting models on a practical real life financial market index data.

2 To determine which model performs best under varying market conditions.

3 To engineer and introduce a psychological variable based on common retail trading behavior patterns.

4 To assess the impact of these behavioral variables on forecasting accuracy using RMSE and MAE metrics.

5 To draw conclusions on the robustness and practical applicability of enhanced models, and also to lead a pathway to motivate to work with such contrarian approaches.

1.5 Need for Computerization of the System

Financial markets generate massive volumes of high-frequency data, making manual forecasting infeasible and error-prone. Computerized systems offer several advantages:

- Scalability: Ability to process years of market data and thousands of data points within seconds.
- Reproducibility: Models and experiments can be systematically validated and replicated.
- Comparative Evaluation: Multiple forecasting algorithms can be trained, tested, and benchmarked on identical datasets under controlled conditions.
- Decision Support: Automated forecasts enable traders, portfolio managers, and policy makers to make informed, data-driven decisions.

Without computerization, analyzing complex models such as ARIMA, LSTM, or Transformers and studying external variables like macroeconomic indicators (GDP, inflation) and psychological biases would not be feasible. Hence, computerization is a mandatory enabler for achieving the project objectives.

1.6 Proposed Software

The proposed code is executed on the Kaggle platform, an online community and environment where data scientists and machine learning practitioners can learn, compete, run code using Kaggle Notebooks, and collaborate on data-driven problems. Kaggle was selected primarily for its free access to three GPUs, which significantly supports our computational needs.

This project involves running multiple models, including deep learning architectures that demand high computing resources. Acquiring such resources locally would require purchasing expensive, high-end GPUs. By using Kaggle's free access to the P100 GPU, which is normally priced around ₹30,000 INR (as of 2025), we overcome this limitation without incurring additional costs.

During the model comparison stage, we evaluate statistical, machine learning, and deep learning approaches for predicting Bahrain's BAX index. However, the final code deployed on Kaggle focuses exclusively on the BAX index, presenting results both with and without the inclusion of psychological variables. We compute key metrics such as MAE and RMSE and visualize the predicted BAX stock price for the next 15 days.

The deployed system generates two complementary outputs for end-users:

1 Pure Model-Based Forecasting: A 15-day forward prediction of BAX closing prices, computed using one of the models selected during the comparative analysis.

2 Behaviorally-Adjusted Forecasting: A parallel 15-day forecast in which the baseline model's predictions are refined by incorporating a contrarian behavioral variable, derived from candlestick misinterpretation patterns commonly observed among retail investors.

To enhance interpretability, both forecasts are accompanied by visualization plots comparing predicted versus actual values:

- Model Only Prediction
- Model + Psychological Variable Prediction

In summary, the proposed software delivers a dual perspective forecasting system that blends quantitative precision with behavioral insight, offering a richer and more actionable understanding of market dynamics.

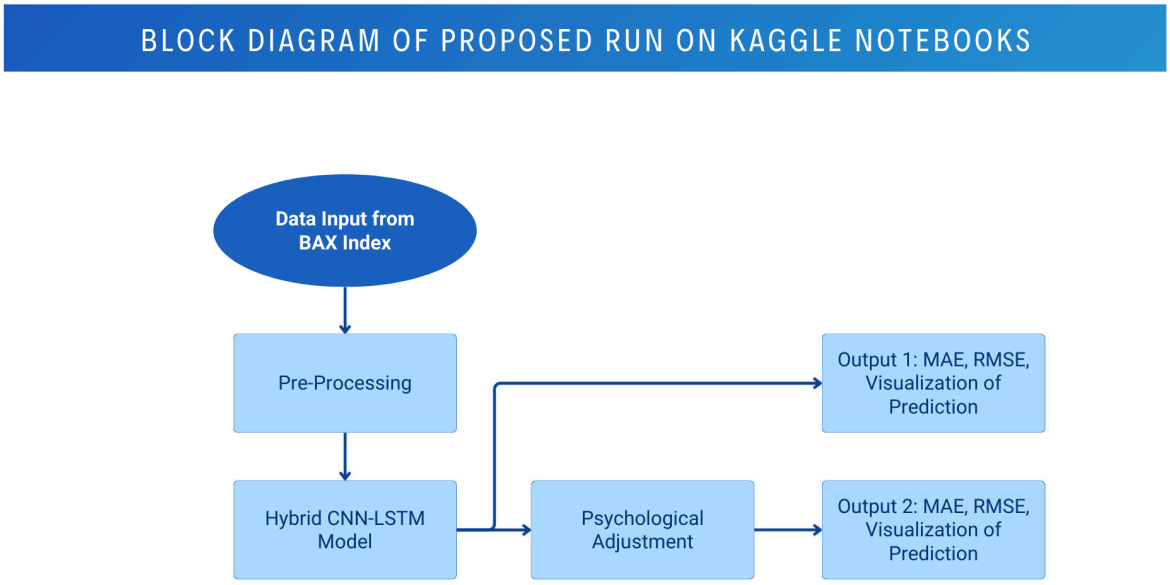

Figure 1.4: Proposed run on Kaggle Notebooks

1.7 Importance of the Work

A recent study by the Securities and Exchange Board of India (SEBI) revealed that approximately 93 % of individual traders in the equity Futures & Options (F&O) segment incurred losses during Fiscal Years 2022–2024, with aggregate losses surpassing ₹1.8 lakh crore over this three-year period. This is released via the document titled 'Updated SEBI Study Reveals 93% of Individual Traders Incurred Losses in Equity F&O between FY22 and FY24; Aggregate Losses Exceed ₹1.8 Lakh Crores Over Three Years'. Despite repeated losses, many continued trading, which is a phenomenon that highlights the powerful influence of behavioral

biases and market psychology. What may seem counterintuitive, but acting contrary to the crowd's expectations, can, in fact, yield statistically significant improvements. By incorporating such a contrarian behavioral variable into model-based forecasts, this project offers a novel and practically grounded forecasting alternative that resonates with real-market patterns.

1.7.1 Practical Relevance:

This work speaks directly to the real-world challenges faced by retail investors, especially in developing markets. Traditional forecasting models often ignore behavioral distortions that lead to persistent losses. By explicitly integrating behavioral signals such as candlestick misinterpretation patterns - the project empowers predictive systems with insights derived from retail trading behavior, potentially offering pathways to mitigate common loss-making tendencies.

1.7.2 Academic Contribution:

This study bridges two distinct research domains - quantitative time series forecasting and behavioral finance. While many existing works focus exclusively on model accuracy or theoretical behavior, this effort combines both, offering empirical validation of how contrarian behavioral adjustments can enhance forecast robustness. In doing so, it enriches model interpretability and expands the frontier of interdisciplinary financial research.

1.7.3 Future Scope:

The framework's modular design allows for seamless integration of additional predictive features. Future enhancements could incorporate event-driven variables (e.g., earnings releases, macroeconomic announcements), or even technical channel analysis, where price models leverage known support/resistance zones to focus predictions within specific chart segments. Such enhancements would introduce context-aware forecasting, providing pre-insight into where market movement is likely to occur.

1.7.4 Technical and Evaluation Significance:

By comparing multiple forecasting methodologies and then using only the one of the performing models with behavioral adjustment, the project demonstrates technical rigor, pragmatic model selection, and ease of adaptation. It delivers a concept for a deployable prototype that goes beyond theoretical constructs, showing how behavioral finance can tangibly improve predictive systems for operational use.

CHAPTER 2
LITERATURE REVIEW

Time series forecasting and price prediction are essential activities across many sectors including finance, energy, retail, manufacturing, and technology. The rapid expansion of data, improvements in computing capabilities, and the need for fast, trustworthy predictions have encouraged the development of numerous modeling methods. These range from conventional statistical models, which provide strong benchmarks and clarity, to sophisticated machine learning and deep learning models that offer superior accuracy and adaptability for intricate, high dimensional, or irregular datasets. As businesses attempt to predict outcomes like stock values, consumer demand, and energy usage, the selection of an appropriate time series model for a particular forecasting situation holds substantial operational and financial weight.

This literature review presents an exhaustive comparative analysis of classical statistical, machine learning, and deep learning models applied to time series forecasting and price prediction. We investigate their theoretical underpinnings, structural benefits, and constraints such as interpretability, computational speed, scalability, and resilience. We also examine their adoption rates within current industry practices. Furthermore, this review highlights benchmark studies, hybrid and ensemble strategies, interpretability trends, and the future course of time series analysis, drawing upon modern academic and industrial references.

2.1 Overview of Classical Statistical Models for Time Series Forecasting

Classical statistical models have formed the foundation of time series forecasting since the early twentieth century. Grounded in probability theory and statistical inference, these models generally assume that future observations depend on past values, random noise, and in some cases external variables.

Key Model Types:

- **Autoregressive Integrated Moving Average (ARIMA):** Among the most widely used approaches is the Autoregressive Integrated Moving Average (ARIMA) family, which includes ARMA, AR, and MA models. These methods represent a time series as a combination of past observations and past errors. ARIMA is particularly effective for linear and stationary series, but it can be adapted through differencing and seasonal parameters to capture trends and periodic patterns.
- **Exponential Smoothing (ETS/Holt-Winters):** Exponential Smoothing methods, including the Error Trend Seasonality (ETS) framework and the Holt Winters technique, are well suited for data that display both trend and seasonality. They generate forecasts

as exponentially weighted averages of past observations, giving greater importance to more recent data points.

- **Theta Method:** The Theta method is valued for its simplicity and strong empirical performance. It works by decomposing a time series and applying exponential extrapolation to predict future values.
- **State Space Models:** State space models, such as those based on the Kalman filter, offer flexible tools for handling time-varying and multivariate patterns.
- **Seasonal Decomposition of Time Series (STL):** Seasonal decomposition techniques, such as STL, separate a series into trend, seasonal, and residual components, allowing each to be analyzed and forecast individually.

2.1.1 Application in Forecasting:

In practical forecasting applications, classical models have shown consistent reliability across a wide range of scenarios, particularly in univariate forecasting tasks. ETS methods are often preferred for short-term and seasonal data, while ARIMA remains a common choice for financial and macroeconomic forecasting.

2.1.2 Interpretability and Efficiency:

Model parameters have clear statistical meaning, such as autoregressive terms that capture temporal dependencies. These models also require relatively modest computational resources, enabling rapid implementation and scaling on standard hardware. Their transparent parameterization supports auditability and compliance in regulated sectors such as finance and healthcare.

2.1.3 Benchmark Results:

Evidence from multiple empirical studies, including the M Competitions from M1 through M6 and other benchmark evaluations, indicates that classical models frequently outperform more complex machine learning and deep learning approaches in univariate, short-horizon forecasting. As a result, they are often recommended as the baseline for assessing the performance of more advanced methods

2.2 Overview of Machine Learning Models for Time Series Forecasting

Machine learning has introduced a new paradigm for time series analysis, focusing on extracting patterns directly from data rather than relying on explicit statistical assumptions. The

machine learning toolkit includes a wide range of algorithms that were originally developed for tabular data but are now frequently adapted for sequential and temporal structures.

Key Model Types:

- **Decision Tree Ensembles (Random Forest, XGBoost, LightGBM):** These methods combine multiple decision trees to capture nonlinear and high-dimensional relationships. XGBoost, in particular, is recognized for its speed and accuracy and has achieved strong results in numerous data science competitions.
- **Support Vector Regression (SVR):** Support Vector Regression applies regularization and kernel methods to model complex patterns while limiting overfitting, making it effective for datasets of moderate size.
- **K-Nearest Neighbors (KNN):** K-Nearest Neighbors regression, although conceptually simple, can be adapted for short-term local pattern matching in time series when appropriate lagged features are used.
- **Linear Models (Ridge, Lasso, Elastic Net):** Penalized linear models such as Ridge, Lasso, and Elastic Net support feature selection and regularization, which is especially valuable when working with high-dimensional exogenous variables.

2.2.1 Time Series Adaptation

To apply machine learning methods to time series forecasting, the data is typically transformed into a supervised learning format by creating lagged variables as predictors. Unlike classical statistical models, most machine learning algorithms do not inherently account for temporal order or autocorrelation. As a result, feature engineering techniques such as rolling windows, time-based data splits, and lagged variable creation are essential for effective performance.

2.2.2 Interpretability and Flexibility

Machine learning approaches offer a balance between flexibility and interpretability. Tree-based ensembles can provide measures of feature importance, although the models themselves may become less transparent as complexity and feature engineering increase.

2.2.3 Empirical Performance

Empirical evidence shows that machine learning methods perform particularly well when the data is multivariate, contains strong nonlinear interactions, or when external predictors play a significant role, as in demand forecasting influenced by promotions, weather,

or events. In univariate short-term forecasting, however, they often underperform compared with simpler statistical models unless extensive feature engineering is applied.

2.2.4 Integration and Scaling

Modern machine learning libraries offer optimized implementations that support distributed and parallel processing, enabling the analysis of moderate to large datasets. Nevertheless, the level of explainability is often lower than that of classical models, which can be a limitation in regulated or high-stakes environments.

2.3 Overview of Deep Learning Models for Time Series Forecasting

Deep learning models represent a major advancement in sequence modeling, using layers of nonlinear transformations to capture complex temporal dependencies, multivariate interactions, and hierarchical features directly from raw data.

Core Architectures:

- **Recurrent Neural Networks (RNN):** Recurrent Neural Networks form the foundational deep learning architecture for sequential data. They are effective at modeling time-dependent structures but can suffer from vanishing gradient problems when processing long sequences.
- **Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU):** Long Short-Term Memory networks and Gated Recurrent Units address these limitations by modeling long-term dependencies and nonlinear relationships in sequential data. These capabilities are particularly valuable for applications such as energy demand forecasting, speech signal processing, and modeling financial volatility.
- **Convolutional Neural Networks (CNN):** Convolutional Neural Networks, when applied to time series, are able to capture local temporal dependencies and can be combined with recurrent networks for hybrid feature extraction.
- **Attention-Based Models and Transformers:** Attention-based models and Transformers, originally developed for natural language processing, use attention mechanisms to model both short-range and long-range dependencies in parallel. They have achieved state-of-the-art results in complex forecasting problems, especially in high-frequency and spatio-temporal contexts.
- **Hybrid and Specialized Architectures:** Hybrid and specialized architectures, including Variational Autoencoders, sequence-to-sequence networks, and models that explicitly incorporate statistical components, provide robust alternatives for handling uncertainty, regime shifts, and anomalies in data.

2.3.1 Feature Learning

A key strength of deep learning is its ability to learn feature representations directly from the data, removing the need for manual feature engineering. This capability is particularly advantageous in high-dimensional, multivariate, and nonlinear time series, where it can lead to significant improvements in forecast accuracy, provided that sufficient training data is available.

2.3.2 Interpretability and Complexity

Deep learning models often trade interpretability for performance. The relationship between learned weights or activations and model behavior is not immediately transparent. Research efforts aimed at improving explainability, such as attention visualization, saliency mapping, and hybrid interpretable deep learning models, are ongoing but remain an area of active development, especially in regulated industries.

2.3.3 Computational Demands

The primary drawback of deep learning is its high computational demand. Achieving optimal performance and generalization typically requires substantial computing resources, such as graphics processing units, tensor processing units, or distributed systems, along with large volumes of data. This can present a barrier for smaller organizations or for problems with limited data availability.

2.3.4 Empirical Performance

Empirical evidence shows that deep learning models excel in complex forecasting tasks involving nonlinearity, multivariate relationships, and high-frequency or multi-step predictions, where conventional approaches may be less effective. However, they can overfit in low-signal or short-series scenarios and are vulnerable to issues such as noisy label propagation and covariate shift unless appropriate regularization and noise-robust training strategies are applied.

2.4 Popularity and Adoption:

- Classical statistical models remain the industry standard in many established sectors such as finance, energy, and economics, largely due to their proven track record and simplicity.
- Machine learning models are increasingly adopted when covariates or external variables are present, as in demand planning, marketing optimization, and fault detection.

- Deep learning models are experiencing rapid growth in areas such as big technology, retail, the Internet of Things, and finance, where they are applied to complex, high-volume, or multivariate datasets. They are reshaping best practices in domains including supply chain forecasting, fraud detection, and smart manufacturing.

2.5 Discussion by Dimension

Let us discuss on Interpretability, Computational Efficiency, Scalability and Robustness for models.

2.5.1 Interpretability

Interpretability is a critical consideration in model selection, particularly in regulated industries and in contexts where user trust must be established. Classical statistical models such as ARIMA and ETS allow practitioners to directly examine model coefficients, trend and seasonality parameters, and residual errors, providing clear and intuitive explanations for forecasts. This transparency supports acceptance by business stakeholders and compliance with regulatory requirements.

Machine learning models, especially decision tree ensembles, offer partial interpretability through tools such as variable importance metrics and partial dependence plots. However, as model complexity and feature interactions increase, transparency can diminish. Research into local explanation methods such as LIME and SHAP, as well as model distillation techniques, aims to improve interpretability, though these methods may not fully capture the underlying decision process, particularly in sequential or temporal contexts.

Deep learning models face an even greater interpretability challenge. While techniques such as attention maps and integrated gradients can provide local explanations, the overall decision logic remains difficult to extract. Hybrid approaches, such as the InterpGN framework, attempt to preserve interpretability for specific predictions or subsequences, representing a promising area of ongoing research.

2.5.2 Computational Efficiency

Computational efficiency is influenced by both model complexity and the scale of the data. Statistical models require relatively few parameters and can be trained and deployed quickly, even in real-time forecasting scenarios. This efficiency makes them attractive in environments with limited computational resources.

Machine learning models generally require more computational power, particularly during hyperparameter optimization and when working with large feature sets or datasets.

Modern libraries such as scikit-learn and XGBoost include optimized, parallelized routines that help reduce training time for moderate to large datasets.

Deep learning models often require specialized hardware such as graphics processing units or distributed computing systems, especially when training on large datasets or tuning large transformer architectures. In production environments, model size, inference cost, and latency can become significant concerns. Research into lightweight architectures, including model distillation, quantization, and efficient transformer designs, is actively addressing these challenges, as demonstrated by models such as lightweight Fredformer and large language model-based forecasting systems.

2.5.3 Scalability

Scalability is essential for applications such as retail sales forecasting across thousands of products, monitoring data from large networks of IoT sensors, or executing high-frequency algorithmic trading strategies. Machine learning and deep learning models can scale almost linearly with data size when supported by distributed computing infrastructure, although this requires significant engineering resources. Deep learning models have achieved scalability to billions of parameters, enabling generalization to new datasets and supporting transfer learning and zero-shot forecasting.

Classical statistical models, while strong in interpretability and stability, are less suited to high-dimensional or rapidly evolving datasets. In such cases, retraining and model management can become operational bottlenecks.

2.5.4 Robustness

Robustness refers to a model's ability to handle noise, anomalies, missing data, and distribution shifts. Statistical models can tolerate moderate noise, but deep learning models are more prone to overfitting to noisy data unless explicit regularization or noise modeling is applied. Advances in this area include adversarial training, robust loss functions, and hybrid architectures designed to address anomalies and temporal label noise, which are pushing the boundaries of robust time series forecasting.

2.6 Industry Adoption and Application Trends

Classical statistical methods remain the foundation of time series forecasting in mature industries where interpretability, compliance, and operational reliability are paramount. Finance, energy, and government sectors continue to rely heavily on ARIMA, ETS, and their

seasonal variants for tasks such as price forecasting, load prediction, and macroeconomic indicator modeling.

The growth of digital transformation and the availability of large-scale data are accelerating the adoption of machine learning and deep learning approaches:

- Machine Learning is widely used in multivariate and high-dimensional forecasting, including retail demand forecasting that incorporates promotions and competitor actions, predictive maintenance in manufacturing, and customer value forecasting in software-as-a-service businesses.
- Deep Learning is increasingly prominent in technology, e-commerce, and finance, particularly for complex, large-scale, or real-time decision-making applications. Examples include logistics and personalization at Amazon and Google, demand and driver prediction at Uber, and supply chain and event-based forecasting at Alibaba.
- Hybrid and Ensemble Models, that combine techniques such as LSTM for sequence modeling with ARIMA for residual correction are gaining popularity, especially when short-term interpretability and long-term accuracy are both required.
- The rise of AutoML and forecasting platforms-for example, Facebook Prophet, Amazon Forecast, and Google Cloud AI Forecasting, reflects a trend toward democratization and scalability, making advanced forecasting capabilities accessible to organizations of all sizes.
- Large Language Model (LLM) Based Forecasting: Large language model-based forecasting systems, such as AWS Chronos, are emerging as powerful tools that leverage foundation models and transformer architectures for domain-agnostic, zero-shot, or few-shot forecasting. These approaches are expanding adoption in enterprises dealing with complex and heterogeneous data streams.
- According to enterprise AI strategy surveys conducted in 2024, approximately seventy percent of organizations have integrated classical or machine learning-based time series forecasting into operational workflows. Adoption of transformer and deep learning architectures is rising sharply, particularly in digital-native industries and large-scale industrial environments.

2.7 Benchmark Datasets and Evaluation Metrics

Realistic benchmarking is foundational to model comparison and advancement. Prestigious competitions like M1-M6 (Makridakis Competitions), and widely used datasets (e.g., M3/M4, air passengers, electricity demand, S&P500 prices, IoT sensor time series) have boosted model innovation and rigorous performance assessment.

Commonly used Metrics are:

- **Mean Squared Error (MSE) / Root Mean Squared Error (RMSE):** Measures average squared/absolute forecasting error; heavily penalizes larger deviations.
- **Mean Absolute Error (MAE) / Mean Absolute Percentage Error (MAPE):** Direct measures of typical forecast error; MAPE is scale-independent but sensitive to small actual values.
- **Symmetric Mean Absolute Percentage Error (sMAPE):** Mitigates certain limitations of MAPE; widely used in M4/M5 competitions.
- **Mean Scaled Error (MASE):** Allows comparisons across different forecast origins and series scales.
- **Prediction Interval Coverage Probability (PICP) and Mean Scaled Interval Score (MSIS):** For probabilistic/interval forecasting.

2.7.1 Empirical Insights

The M4 competition (2018) comprising 100,000 real-world series with diverse domain and frequency confirmed the continued value of classical statistical and hybrid models-as pure machine learning or deep learning were rarely the sole overall winners. Combinatorial and hybrid forecasts (statistical + ML/DL) offered marginally better accuracy over single-model approaches across most scenarios.

2.8 Case Studies in Price Prediction Applications

2.8.1 Stock Market Prediction

- Studies juxtaposing ARIMA, XGBoost, LSTM, and hybrid approaches on stock price prediction indicate that no single method dominates in all circumstances. For established stocks with regular patterns, ARIMA or Prophet is sufficient; in high-frequency trading, DL (especially LSTM/Transformer) and streaming online learning methods adapt better to rapidly changing conditions613.
- Hybrid models leveraging LSTM for complex sequence learning and ARIMA for capturing regular error structures have demonstrated enhanced accuracy, signifying the growing role of model fusion in price prediction20.

2.8.2 Retail Demand Forecasting

Tree ensemble models (XGBoost, Random Forests) and boosting algorithms now dominate retail demand and inventory forecasting due to their ability to integrate

calendar effects, promotions, weather, and macroeconomic factors. Their interpretability (via SHAP, feature importance) supports operational decision-making and "what-if" scenario analysis4.

2.8.3 Energy, IoT, and Sensor Networks

LSTM and Transformer models are increasingly prevalent for consumption forecasting, grid management, and anomaly detection in the energy sector and industrial IoT, where multivariate, high-frequency, and spatially distributed data streams overwhelm traditional models.

2.9 Software Tools and Frameworks

A comprehensive ecosystem now supports time series modeling across all classes:

- **Classical Models:** R's forecast, Python's statsmodels, and SAS/SQL libraries for ARIMA, ETS, and seasonal models.
- **Machine Learning:** scikit-learn, XGBoost, LightGBM, H2O.ai, and PyCaret offer robust ML tools. XGBoost and LightGBM, in particular, support large-scale, parallel model fitting for feature-rich settings.
- **Deep Learning:** Leading frameworks include TensorFlow, PyTorch, Keras, and GluonTS, supporting sophisticated DL architectures (LSTM, Transformer) and custom model development for industrial-scale forecasting.
- **AutoML and Cloud Services:** Facebook Prophet (automated additive models for trend/seasonality, robust to missing data and outliers), Google Cloud Forecasting, AWS Forecast/Chronos, and Azure AutoML democratize access to advanced models, streamline hyperparameter tuning, and accelerate deployment.

2.10 Hybrid and Ensemble Approaches

Emerging best practices recognize the value of hybrid and ensemble models-combining statistical, ML, and DL approaches to harness multiple strengths:

- **Statistical + ML/DL:** Use ARIMA/ETS for baseline fit and residual correction with XGBoost or LSTM, enhancing accuracy and robustness, particularly for long-term or multivariate problems.
- **Model Averaging and Stacking:** Weighted combinations of classical and advanced models routinely outperform single models in competitive and industrial use.
- **Feature-Augmented Models:** Combine engineered features from classical decomposition (trend, seasonality, residuals) with ML/DL architectures as inputs.

- **AutoML and Automated Model Selection:** Platforms automate trial, selection, and blending of multiple models to maximize performance and reduce manual engineering.

2.11 Research Trends and Future Directions

As the world enters an era of pervasive data streams, sensor networks, and real-time analytics, time series forecasting is at an inflection point:

- Integration of AI and Automated Forecasting: AI development is streamlining automated modeling, hyperparameter optimization, and result deployment via AutoML.
- Scalability and Foundation Models: Foundation models (e.g., Chronos by AWS) and large-scale transformers are lowering the barrier to domain-general, transferable forecasting across diverse industries.
- Improved Robustness and Adaptivity: Ongoing research in robust time series forecasting, noise-aware models, and anomaly-adaptive architectures promises enhanced resilience for "messy" real-world data.
- Explainability and Trust: As ML and DL models permeate regulated domains, the demand for interpretable and trustworthy forecasting grows. Hybrid models (InterpGN, attention visualization, local explanation metrics) are at the forefront of transparent forecasting research.
- Cloud-Based, Real-Time Forecasting: Cloud-native, serverless, and streaming architectures enable massive scaling, rapid retraining, and push real-time analytics to mobile and edge environments.
- Hybridization and Ensemble Modeling: The success in M4/M5 competitions and industry benchmarks confirms hybrid and ensemble models as the practical standard for high-stakes forecasting tasks, marrying the best of classical, ML, and DL paradigms.

2.12 Comparative Table of Model Types Across Key Dimensions

Table 2.1: Comparing Stats, ML and Deep Learning Models on different dimensions

Dimension	Classical Statistical Models	Machine Learning Models	Deep Learning Models
Short-Term Accuracy	High (1-step, low-noise)	Moderate-High (with features)	Moderate

Long-Term Accuracy	Moderate	Moderate	High (with enough data/labels)
Interpretability	High	Moderate	Low (but improving with research)
Computational Efficiency	High	High (for moderate datasets)	Low to Moderate (needs GPU/HPC)
Scalability	Moderate	High	Very High
Robustness (Noisy Data)	Moderate	Moderate	High (if regularized/tuned)
Handling High Frequency	Weak	Strong (with feature engg.)	Very Strong (attention/CNN/LSTM)
Popularity in Industry	Very High (traditional)	High (esp. with covariates)	Rapidly increasing (big data)
Ease of Implementation	Simple to moderate	Moderate	Complex, high setup cost
Best For	Univariate, clean, short	Multivariate, feature-rich	Nonlinear, complex, big data

2.13 Conclusion

Time series forecasting and price prediction are critical capabilities across a wide range of industries. The choice of forecasting model is shaped by factors such as the characteristics of the data, the forecasting horizon, the need for interpretability, the availability of computational resources, and the presence of noise or anomalies. Classical statistical methods continue to hold a strong position for short-term, univariate forecasting tasks because of their accuracy, transparency, and operational simplicity.

Machine learning models extend this toolkit in multivariate and high-frequency contexts, particularly when supported by well-designed feature engineering. These approaches are valuable when external variables or complex interactions play a significant role in shaping outcomes.

Deep learning, driven by architectures such as Long Short-Term Memory networks, Gated Recurrent Units, and Transformer-based models, represents the leading edge for long-term, large-scale, and multivariate time series analysis. Their ability to capture intricate

temporal dependencies and nonlinear relationships makes them highly effective as data volumes grow and as advances in scalability and robustness continue. However, the complexity and opacity of these models' present challenges for interpretability, regulatory compliance, and practical deployment. This has spurred significant research into explainable artificial intelligence, robust hybrid architectures, and automated model selection techniques.

The future of time series and price forecasting is expected to be both hybrid and automated. Continued integration of statistical rigor with the adaptability of artificial intelligence will be accompanied by broader access through automated machine learning and cloud-native forecasting platforms. For enterprises and researchers, the key will be to evaluate forecasting requirements in light of these developments, balancing accuracy, trust, and operational constraints in order to select or design forecasting architectures that align with their strategic objectives in the years ahead.

CHAPTER 3
SYSTEM ANALYSIS, OBJECTIVE, SCOPE

System Analysis is the process of examining a problem domain, defining objectives, and identifying requirements to ensure that the proposed solution is both effective and feasible. It answers the fundamental questions of "what is to be built and why?" by breaking down the project into well-defined components. In the context of this project, system analysis focuses on understanding the challenges of financial time series forecasting, identifying gaps in existing approaches, and establishing the technical and practical justification for integrating behavioral finance insights into predictive modeling.

3.1 Problem Identification / Definition

Accurately forecasting financial market movements remains one of the most complex problems in quantitative finance. The non-linear, non-stationary, and volatile nature of price series makes traditional linear models inadequate for capturing intricate temporal dependencies. Furthermore, financial markets are influenced by a wide array of external variables, including macroeconomic indicators, geopolitical events, and investor sentiment, which complicates the prediction task.

While researchers and practitioners have developed numerous forecasting models ranging from statistical approaches like ARIMA and SARIMA to machine learning and deep learning techniques such as Random Forests, Gradient Boosted Trees, and LSTM networks - no single model consistently outperforms others across all datasets and market conditions. This inconsistency underscores the necessity of a comparative evaluation framework to determine the most reliable model for specific market contexts.

A second and often overlooked challenge lies in the behavioral dimension of markets. Despite advances in algorithmic trading, retail participation remains significant, particularly in emerging markets such as India. The SEBI 2024 report highlights that nearly 93% of individual traders in equity F&O incurred losses, suggesting that the majority of retail investors follow predictable but suboptimal strategies. Commonly used candlestick chart patterns (e.g., Doji, Hammer) are interpreted in a standardized manner across books, online platforms, and brokerage advisories. As a result, large groups of retail traders often act in the same direction, inadvertently creating contrarian opportunities where the market moves opposite to their expectations.

This project is designed to address both dimensions of the problem:

- Modeling Dimension → By comparing multiple forecasting models, identifying the best performer using RMSE and MAE, and validating its robustness across markets.
- Behavioral Dimension → By integrating a contrarian psychological variable derived from candlestick misinterpretations, thereby enhancing the model's practical applicability in retail-driven markets.

In summary, the problem this project seeks to solve is not only the technical challenge of forecasting market indices with higher accuracy, but also the practical challenge of embedding human behavior into predictive frameworks. By doing so, the system aims to deliver forecasts that are both quantitatively sound and behaviorally informed.

3.2 Requirements Gathering

The success of any forecasting system depends on clearly specifying its data inputs, computational tools, and methodological framework. This section outlines the requirements for building the proposed system in terms of data resources, software and hardware tools, and analytical methods.

3.2.1 Data Requirements

The project utilizes historical financial market indices as the primary data source.

Index Considered: The Bahrain All Share Index (BAX) is employed to evaluate multiple models across varying time frames.

Data Source: BAX data, not available via yfinance library, and was sourced as CSV files from Investing.com website.

Time Horizon: A maximum of 15 years (2010–2025) of daily data is collected. For statistical models, shorter horizons (e.g., 2–3 years) are used for efficiency, whereas machine learning and deep learning models utilize the maximum available data.

Features:

- Core features include closing price as the target variable, with volume included where applicable.
- Lagged variables are engineered for models requiring autoregressive structures.

- Additional technical indicators (e.g., RSI, MACD) and macroeconomic variables (GDP, inflation) were considered only for analysis, but not applied across models in code due to the extensive comparative nature of the study in limited time.
- A novel behavioral feature representing a contrarian candlestick variable is engineered for deployment, enabling behavioral adjustment of predictions.

3.2.2 Tools (Software and Hardware Requirements)

The choice of platforms is critical in ensuring smooth development, testing, and deployment of the forecasting system. The selected software tools and hardware resources were chosen for their accessibility, robustness, and suitability for time series forecasting tasks.

Programming Language: Python is used exclusively throughout the project.

Development Platforms: VS Code for data sourcing, cleaning, and integration. Google Collab for statistical models and exploratory analysis. Kaggle Notebooks for deep learning and computationally intensive tasks, leveraging free P100 accelerators for efficient model training and hyperparameter tuning.

Deployment Environment: The final code was deployed on Kaggle Notebook, as the final deep learning Hybrid CNN-LSTM model requires higher computational resource. Recommendations to deploy on Streamlit (or a similar lightweight web framework) is emphasized to deploy the final model and visualize predictions interactively for participants working on such a project in future.

Supporting Tools: Google Docs for report drafting, Notepad for quick edits, and Yandex Browser for resource access.

Libraries and Frameworks:

- Data Handling & Visualization: pandas, numpy, matplotlib.
- Statistical Modeling: statsmodels (ARIMA, SARIMA, Holt-Winters, SES), Prophet, ARCH (for GARCH).
- Machine Learning: scikit-learn (Random Forest, train-test split, metrics), XGBoost, CatBoost.
- Deep Learning: TensorFlow/Keras (LSTM, GRU, CNN-LSTM), PyTorch (custom architectures, GRU, DataLoader).

- Utilities: MinMaxScaler for normalization, RandomizedSearchCV for hyperparameter tuning, warnings for suppressing output.

Hardware requirements are modest, with local CPU resources sufficient for statistical and ML models, while deep learning experiments are offloaded to GPU-enabled cloud notebooks (Kaggle).

Local Machine:

- Operating System: Windows 10 (64-bit)
- Processor: Intel Core i5 (8th Gen)
- Memory: 16 GB RAM DDR4
- VRAM: 2GB AMD Radeon 520
- Storage: 1TB HDD

This environment was adequate for data preprocessing, statistical modeling, and small-scale machine learning experiments.

Cloud Hardware:

Google Collab and Kaggle provided GPU acceleration (NVIDIA Tesla P100) essential for deep learning models, significantly reducing training time. These platforms enabled efficient execution of computationally heavy architectures such as LSTM, GRU, and Transformer-based models.

3.2.3 Methods

The methodological requirements focus on forecasting model development, evaluation, and validation:

Models Tested (16 Total):

- Statistical Models: Naïve, Holt's Linear, Holt's Winter, ARIMA, SARIMA, SARIMAX, SES.
- Automated Tool: Prophet.
- Volatility Forecast: GARCH (30-day volatility forecast).
- Machine Learning: XGBoost, CatBoost, Random Forest.
- Deep Learning: LSTM, GRU, Hybrid CNN-LSTM, Transformer/TFT.

3.2.3.1 Evaluation Metric:

Root Mean Squared Error (RMSE) is the primary measure of accuracy across models.

3.2.3.2 Preprocessing Techniques:

Before model training, significant preprocessing was required to ensure data integrity, especially for the Bahrain All Share Index (BAX). The raw dataset, sourced from Investing.com, contained multiple formatting inconsistencies such as commas, percentage signs, and suffixes ("K" and "M") in the Volume column. All columns were initially imported as object datatypes, necessitating explicit conversion into appropriate numeric and datetime formats. Missing values in the volume field were forward-filled to maintain continuity, while duplicate rows were checked and removed where necessary.

The cleaned dataset was then indexed by date and sorted chronologically, accounting for the Bahrain Exchange's trading week (Sunday–Thursday). Outlier detection was performed using statistical z-scores, volatility-adjusted scatterplots, and return distribution histograms. Time series decomposition was further applied to extract trend, seasonality, and residual components, followed by stationarity testing using the Augmented Dickey-Fuller (ADF) test, which confirmed that the raw price series was non-stationary and required differencing for statistical models. These preprocessing steps ensured that the data fed into forecasting models was both structurally consistent and analytically robust. Other common techniques included:

- Handling of missing values.
- Creation of lag variables for autoregressive models.
- Data normalization (MinMaxScaler) for deep learning architectures.
- Train-test split for holdout evaluation.
- Time series cross-validation where feasible.

3.2.4 Feasibility Study

A feasibility study evaluates whether the proposed system can be realistically designed, implemented, and deployed within the given resource constraints. It ensures that the project is not only technically possible but also economically viable and relevant to real-world market needs.

3.2.4.1 Technical Feasibility

The project is technically feasible due to the availability of robust open-source tools, computational resources, and established methodologies for time series forecasting. Python, the primary programming language, provides a comprehensive ecosystem of libraries such as

statsmodels, scikit-learn, TensorFlow/Keras, PyTorch, Prophet, and XGBoost, which collectively enable statistical modeling, machine learning, and deep learning.

Hardware requirements are modest. Statistical and machine learning models were efficiently executed on a standard CPU environment, while computationally intensive deep learning experiments were carried out on GPU-accelerated Kaggle notebooks (NVIDIA P100), ensuring adequate performance without additional financial cost. Deployment feasibility is further ensured by lightweight frameworks such as Streamlit, which allow interactive web-based applications to be built with minimal infrastructure.

3.2.4.2 Economic Feasibility

The system demonstrates strong economic feasibility as it primarily relies on a publicly available dataset (Investing.com for BAX) and open-source software tools. There are no licensing fees for libraries or development environments, and all computing was carried out either on local systems or on free-tier cloud platforms (Google Collab, Kaggle).

As a result, the project incurs no direct costs, making it 100% cost-effective. The main investment required is time and technical expertise, both of which are manageable within the academic project framework. This ensures that the solution remains practical even for small-scale researchers, individual analysts, or financial enthusiasts without access to enterprise-level infrastructure.

3.2.4.3 Market Feasibility

Market feasibility is reinforced by the growing demand for accurate and reliable financial forecasting solutions across industries such as portfolio management, algorithmic trading, and fintech innovation. The SEBI report (2024) confirming that 93% of retail traders lose money in F&O markets underscores the urgent need for more realistic, behaviorally informed forecasting approaches.

By explicitly integrating psychological trading variables into predictive models, this project addresses an unmet gap in both academia and industry - moving beyond purely mathematical predictions to include behavioral realism. This enhances the applicability of the system in emerging and developed markets alike. Furthermore, the modular design of the framework ensures adaptability to other asset classes (e.g., cryptocurrencies, commodities), increasing its long-term relevance and adoption potential.

3.2.5 Justification

The combination of local computing for development and lightweight tasks with cloud-based GPU resources for deep learning offered an optimal balance between cost-efficiency and computational capability. Furthermore, the use of Kaggle (or similar basic platform) for deployment ensures portability and end-user accessibility without requiring expensive infrastructure.

CHAPTER 4
RESEARCH METHODOLOGY AND SYSTEM DESIGN

4.1 Design methodology and overall architecture

The system is a modular time-series forecasting pipeline organized into clearly separated stages: Data Ingestion, Pattern Detection, Feature Engineering (including scaling and sequence building), Model Training, Recursive Inference, and Evaluation. Each stage is implemented as an independent module that reads well-defined inputs and writes well-defined outputs. Module interfaces are file-based (CSV, NumPy arrays, serialized objects) so experiments are reproducible and artifacts are traceable.

Data ingestion reads a single canonical historical file of daily OHLC (Open, High, Low, Close/Price) records and stores an immutable raw snapshot. Pattern Detection computes a two-candle pattern label for each date using only information from two prior candles (t−2 and t−1). Pattern labels are converted to a fixed one-hot encoding and aligned with each date so they are available during sample construction. Feature Engineering applies a MinMax scaler fitted on the training partition only, and constructs sliding window samples of fixed length TIME_STEPS. Each training sample is a multichannel sequence consisting of the price channel (time series of scaled closes) and the pattern channels (one-hot flag vector repeated across the time axis for the sample target). Model Training performs walk-forward cross-validation on the training block to select hyperparameters and then trains a final CNN→Pool→LSTM→Dense model on the full training block. Recursive Inference produces multi-step forecasts by predicting one step, appending the predicted (scaled) value to the input window, and repeating; pattern flags for forecast days are provided from historical labels when available and set to zero when not available. Evaluation computes standard error metrics and produces comparative plots.

This architecture enforces chronological separation to avoid information leakage: all pattern labels and scaling parameters used to predict day t are computed from days ≤ t−1. Walk-forward cross-validation uses sequential segments A→B, A+B→C, and A+B+C→D to select hyperparameters while maintaining time ordering.

4.2 Database design and data structures

The system relies on three classes of persistent artifacts: raw historical data, derived/feature artifacts, and model artifacts. This design supports reproducibility, auditability, and easy re-training.

4.2.1 Raw historical table (single source of truth)

Each row represents one trading date and contains the following fields: Date (primary key), Open, High, Low, Price (Close), Volume (optional). The CSV file with this table is retained as an immutable snapshot for each experiment run.

4.2.2 Pattern labels table (derived, aligned to dates)

Each row contains: Date (foreign key → RawPrices.Date), PatternLabel (string), and P binary columns PatternFlag_1 … PatternFlag_P representing the one-hot encoding for the detected pattern types (for example, NoPattern, Hammer, Doji, EngulfingBull, EngulfingBear, Piercing Line, Dark Cloud Cover, Bullish Harami, Bearish Harami, SmallBullish, SmallBearish). PatternLabel is determined only from the two prior candles (t−2, t−1) and is therefore strictly derived from past data.

4.2.3 Feature / samples store (training artifacts)

Each sample captures a sliding window and target and includes: SampleID, WindowStartDate, WindowEndDate, PriceSequence (serialized array of length TIME_STEPS, scaled), PatternVector (serialized one-hot vector aligned to the target day; can be repeated or stored separately), TargetPrice (scaled). For storage efficiency, PriceSequence and PatternVector may be serialized as NumPy .npy blobs or stored as flattened columns price_0 … price_{TIME_STEPS−1} and pattern_0 … pattern_{P−1}.

4.2.4 Model artifacts

The system stores: trained model weights (SavedModel or HDF5), scaler object (joblib or pickle), hyperparameters JSON, training history (loss per epoch), and evaluation metrics (MAE, RMSE, MAPE). Filenames include run timestamps and a short hash of the hyperparameters to enable exact reproduction of experiments.

If a relational database is used, the minimal schema comprises two tables: RawPrices(Date PK, Open, High, Low, Price, Volume) and PatternLabels(Date PK, PatternLabel, PatternFlag_1, ..., PatternFlag_P). Optionally, a Samples table can be added for prebuilt training windows: Samples(SampleID PK, EndDate, PriceArray BLOB, PatternArray BLOB, TargetPrice).

4.3 Screen design (UI/UX)

No end-user deployment UI is required for the project. The system is implemented as reproducible Python scripts that run on a development machine or server. For demonstration

and evaluation purposes, a minimal developer console or simple CLI is sufficient. The developer console exposes these commands: data_preview (print a small table and a plot of Price and OHLC for a selected date range), run_cv (execute walk-forward cross-validation and report best hyperparameters), train_final (train final model on the full training block and serialize artifacts), and infer_test (produce a 15-day recursive forecast on the held-out test block and export plots and metrics).

The expected outputs presented in the report are: a time-series plot comparing Actual vs Forecast for the 15-day horizon, a second plot showing the model with pattern inputs vs the baseline model without pattern inputs, and a metrics table reporting MAE, RMSE, and MAPE for both variants. All plots use identical axes and date ranges to allow visual comparison. Screenshots of these outputs are included in the project report.

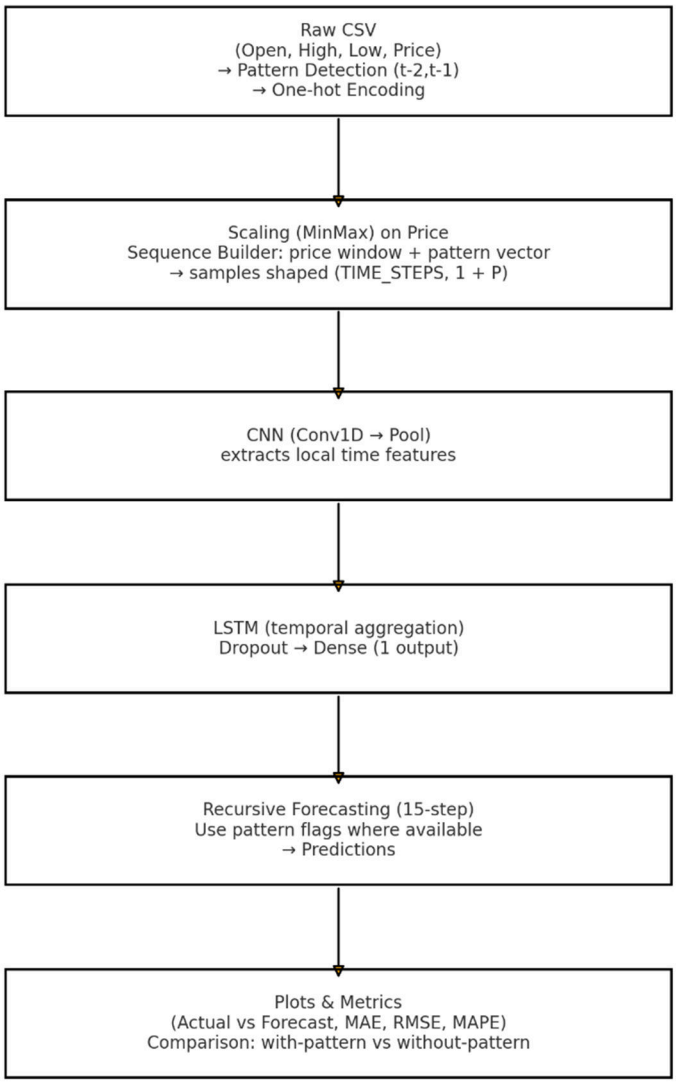

Figure 4.1: Data Flow Block Diagram

4.4 Data flow diagrams (DFD)

A Level-0 diagram above shows modules and data stores, and a Level-1 diagram expands the preprocessing component into Pattern Detection, Scaling, and Sequence Builder. Each process box is labelled with the input and output data artifacts to make dependencies explicit.

4.5 Component/UML overview

The main Python modules and their exported functions used were: pattern_detector.py (detect_two_candle_pattern_row), feature_builder.py (make_sequences_with_patterns), model.py (build_model_multichannel), trainer.py (train_and_validate_one_fold_idx, run_cv_and_select_best), infer.py (recursive_forecast_multichannel), and utils.py (scaler utilities, metrics).

4.6 Sequence diagrams and flowcharts

A sequence that traces the recursive inference runtime: obtain last TIME_STEPS scaled prices → build multichannel input using pattern flags → model.predict → update window with predicted scaled price → repeat for horizon steps. Include a flowchart of the train script: load data → compute patterns → CV loop → train final model → save artifacts → evaluate on test block.

4.7 Validation artifacts

Tables and plots that compare the baseline model and the pattern-augmented model on the final test window and on a small set of rolling windows are provided. MAE and RMSE provide a short statistical summary (mean and standard deviation of RMSE across windows).

4.8 Implementation blueprint

1 **Environment setup:** Install Python 3.8+ and required packages: numpy, pandas, scikit-learn, matplotlib, tensorflow (2.x). Configure a virtual environment and record package versions to ensure reproducibility.

2 **Data ingestion:** Place the raw CSV in the project data directory. Validate columns and types, then load the CSV into a DataFrame and sort by Date. Save a timestamped copy of the raw CSV for traceability.

3 **Pattern detection:** Implement the two-candle rule function that accepts rows t−2 and t−1 and returns a pattern label. Verify the function on hand-picked examples by visual inspection of OHLC candlesticks.

4 **One-hot encoding and alignment**: Convert pattern labels to a fixed one-hot encoding and align them with each date. Save the pattern labels table as a derived CSV.

5 **Feature preparation:** Fit a MinMax scaler on the training partition only. Build sliding window samples of length TIME_STEPS where each sample is a multichannel array: price channel (TIME_STEPS values) and pattern channels (one-hot vector repeated across TIME_STEPS or placed only at final time step). Persist the training samples for fast retry.

6 **Model training and CV:** Implement the CNN→Pool→LSTM→Dense architecture. Execute walk-forward cross-validation over the training block to select hyperparameters. Record training histories, selected hyperparameters, and CV metrics.

7 **Final training and inference:** Train the final model on the entire training block with the selected hyperparameters. Serialize the model and scaler. Run recursive 15-step inference on the held-out test block, produce plots, and compute MAE, RMSE, and MAPE.

8 **Comparison and reporting:** Repeat steps 5–7 with pattern channels disabled to produce the baseline model. Compile comparative tables and plots showing both models on identical test horizons and summarize the results in the report.

CHAPTER 5
TESTING AND IMPLEMENTATION

5.1 Code for Prediction without the Psychological Variable

Working Explained:

We give the script a single time series: the index Close price (called Price). The code first cuts the series into a training block (first 80%) and a test block (last 20%). Inside the training block it does a walk-forward style cross–validation (A→B, A+B→C, A+B+C→D) to pick hyperparameters. For training samples, it takes a sliding window of the past TIME_STEPS values and asks the model to predict the next close. "Sliding window" means if TIME_STEPS = 60, one training sample is the 60 most recent closes and the target is the 61st close. The CNN layer looks for short patterns in the sequence, the LSTM captures time relationships, and the Dense layer outputs a single number (the predicted scaled Close). During a multi-step forecast the model predicts day+1, then feeds that predicted value back into the window to predict day+2, and so on — that is called recursive forecasting. At the end the code draws a plot of actual vs forecast and computes MAE/RMSE.

5.1.1 Imports & reproducibility

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

We load standard Python packages:

- numpy/pandas for data handling
- matplotlib for plotting
- tensorflow.keras for building CNN-LSTM model
- sklearn for scaling (normalization) and evaluation metrics

```
np.random.seed(42)
tf.random.set_seed(42)
```

Fix random seeds to make results reproducible (same results each time).

5.1.2 Data loading

```
CSV_PATH = "/kaggle/input/cnn-lstm-v2/df_bax_cleaned_till_outliers.csv"
df = pd.read_csv(CSV_PATH)
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values("Date").set_index("Date")
```

Load dataset from CSV

Convert "Date" column into proper datetime

Sort chronologically and set date as index

```
y = df["Price"].astype("float32").values.reshape(-1, 1)
dates = df.index
```

We only use the Price column (the target series) for modeling.

5.1.3 Train/test split

```
n = len(y)
split_80 = int(0.80 * n)
y_train_block = y[:split_80]
y_test_block  = y[split_80:]
dates_test    = dates[split_80:]
```

80% of data = training block

20% = test block (completely untouched until the very end)

5.1.4 Cross-validation folds

```
def split_into_four_segments(arr):
    seg_len = len(arr) // 4
    A = arr[:seg_len]; B = arr[seg_len:2*seg_len]
    C = arr[2*seg_len:3*seg_len]; D = arr[3*seg_len:]
    return A, B, C, D
A, B, C, D = split_into_four_segments(y_train_block)
folds = [
    (np.concatenate([A]), np.concatenate([B])),
    (np.concatenate([A, B]), np.concatenate([C])),
    (np.concatenate([A, B, C]), np.concatenate([D])),
]
```

We split the 80% training data into 4 segments: A, B, C, D.

- *Fold1: Train on A → validate on B*
- *Fold2: Train on A+B → validate on C*
- *Fold3: Train on A+B+C → validate on D*

This is called rolling-origin cross-validation (common in time-series).

5.1.5 Sequence creation

```
def make_sequences(arr_scaled, time_steps):
    X, y = [], []
    for i in range(len(arr_scaled) - time_steps):
        X.append(arr_scaled[i:i+time_steps, 0])  # input = past window
        y.append(arr_scaled[i+time_steps, 0])    # output = next value
    X = np.array(X)[..., np.newaxis]
    y = np.array(y)
    return X, y
```

Takes time series → builds sliding windows of length time_steps.

Example: if time_steps=3 and data = [10,11,12,13], it makes:

- *X = [10,11,12], y = 13*

5.1.6 Model building

```
def build_model(hp, time_steps):
    model = Sequential()
    model.add(Conv1D(filters=hp["conv_filters"], kernel_size=hp["kernel_size"],
                     activation=hp["conv_activation"], input_shape=(time_steps, 1)))
    model.add(MaxPooling1D(pool_size=hp["pool_size"]))
    model.add(LSTM(hp["lstm_units"], activation=hp["lstm_activation"]))
    model.add(Dropout(hp["dropout"]))
    model.add(Dense(hp["dense_units"]))
    ...
    model.compile(optimizer=Adam(...), loss="mse")
    return model
```

This is the CNN-LSTM network:

- *Conv1D + MaxPooling: extract short-term patterns from price windows*
- *LSTM: capture sequential dependencies*
- *Dropout: prevent overfitting*
- *Dense(1): output next predicted price*

5.1.7 Metrics

```
def compute_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return {"MAE": mae, "RMSE": rmse, "MAPE": mape}
```

Computes MAE, RMSE, and MAPE to evaluate forecasts.

5.1.8 Recursive forecasting

```
def recursive_forecast(model, scaler, train_block_raw, target_block_raw, time_steps,
horizon=15):
    ...
```

Key idea:

- *Fit scaler on training data only*
- *Start with last time_steps prices*
- *Predict 1 step ahead → feed prediction back as new input*
- *Repeat until horizon (15 days) reached*

This is called recursive (multi-step) forecasting.

5.1.9 Training per fold

```
def train_and_validate_one_fold(train_raw, val_raw, hp):
    ...
    model = build_model(hp, hp["TIME_STEPS"])
    model.fit(..., callbacks=[EarlyStopping(...)])
    preds, actual = recursive_forecast(...)
    val_metrics = compute_metrics(actual, preds)
    return val_metrics, model
```

Each fold: train on training block, forecast 15 days on validation block, compute metrics.

5.1.10 Hyperparameters and grid

```
base_hp = {...}
grid = [{**base_hp}, {**base_hp, "TIME_STEPS": 45}, ...]
```

We try different hyperparameter settings (grid search).

5.1.11 Rolling CV run

```
def run_cv_and_select_best(grid):
    for hp in grid:
        fold_metrics = []
        for train_raw, val_raw in folds:
            m, _ = train_and_validate_one_fold(train_raw, val_raw, hp)
            fold_metrics.append(m["RMSE"])
        avg_rmse = np.mean(fold_metrics)
        results.append({...})
    best = min(results, key=lambda d: d["avg_val_rmse"])
    return best
```

Runs all folds for each hyperparameter setting → chooses the best one by average validation RMSE.

5.1.12 Final training and test

```
final_model, test_preds, test_actual, test_metrics = train_final_and_test(y_train_block,
y_test_block, best_hp)
```

Retrain model on entire 80% training block with best hyperparameters

Forecast 15 days ahead on untouched 20% test block

5.1.13 Naive baseline

```
naive_preds = []
prev = y_train_block[-1, 0]
for t in range(len(y_test_block)):
    naive_preds.append(prev)
    prev = y_test_block[t, 0]
```

Naive baseline = "tomorrow's price = today's price".

This is a common benchmark to see if the model beats a trivial approach.

5.1.14 Plot

Plots actual test prices, CNN-LSTM forecast, and naive baseline for first 15 days.

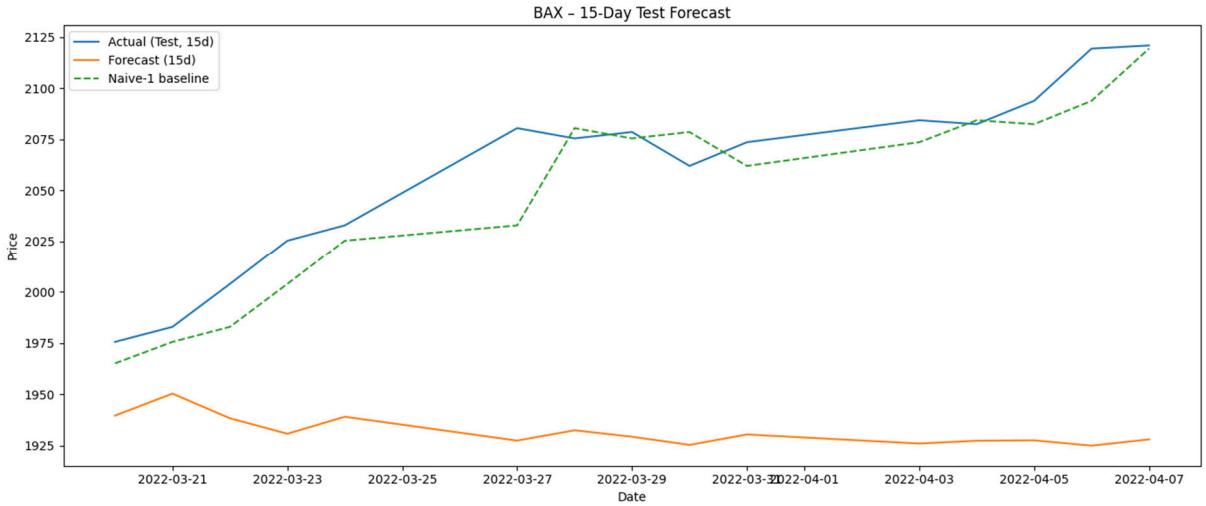

Figure 5.1: Plot of BAX 15-day Forecast

5.2 Overall Abstract on how the model is trained and verified:

1. Dataset preparation
   - Only the Price column is used.
   - Data is split: 80% training, 20% final test.
2. Cross-validation inside training block
   - The 80% training data is further split into 4 equal segments A, B, C, D.
   - Rolling-origin CV:
     - Fold 1: Train A → Validate B
     - Fold 2: Train A+B → Validate C
     - Fold 3: Train A+B+C → Validate D
   - Each fold trains the CNN-LSTM and tests its 15-day forecast on the next block.

3. Hyperparameter tuning
   - Several model settings are tried (time steps, filters, LSTM units, etc.).
   - The setting with lowest average RMSE across folds is chosen.

4. Final model training

5. Train on full 80% data using best hyperparameters.

6. Forecast 15 days ahead on an untouched 20% test block.

7. Comparison with Naive baseline

- The naive model assumes next day = previous day.
- Performance metrics of CNN-LSTM vs Naive are compared.
- In your output, the naive baseline actually outperformed CNN-LSTM (very important finding).

The workflow is time-series rolling cross-validation + CNN-LSTM recursive forecasting + final test evaluation + naive baseline comparison.

- Hyperparameters are fixed across folds for one trial
- They are tuned as many times as there are entries in the grid
- The final decision is based on average RMSE across all folds for each setting

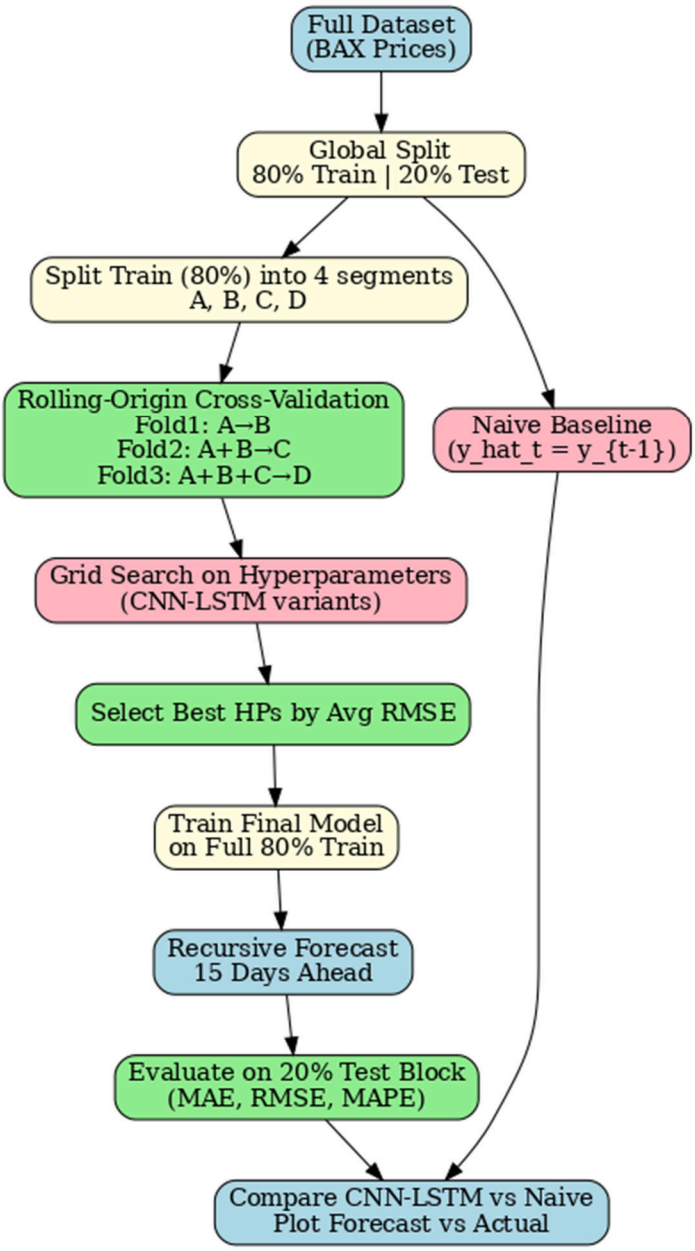

Figure 5.2: Project Workflow

5.3 Code for Prediction WITH Psychological Variable

Working Explained:

Everything above is still true, except the new code adds candlestick-pattern information as additional input features. For each date the code looks at the two previous candles (t−2 and t−1), runs simple rules (open/high/low/close comparisons) and assigns a pattern name like "Hammer", "EngulfingBull", or "Doji". That pattern is converted to a one-hot vector. When building training samples, the code attaches that one-hot vector as extra columns for each time step, so each sample window now contains the price channel plus the pattern channels. The model architecture is the same but its input shape increases: instead of (TIME_STEPS, 1) it becomes (TIME_STEPS, 1 + P) where P is the number of distinct patterns. At inference the model uses the historical pattern available for predicting day t (computed from t−2 and t−1), and the recursive forecast uses either historical pattern indices for the first step and zeros afterwards (as implemented), or you can choose a dynamic version which recomputes pattern from predicted prices for later steps.

5.4 Concrete example to picture it:

If TIME_STEPS = 60 and we have 10 pattern flags, then:

Original sample shape = (60, 1). It is just 60 closes stacked vertically.

New sample shape = (60, 11). Column 1 is the 60 price values; columns 2–11 are the same 10-value one-hot pattern vector repeated on each of the 60 rows (in the provided code). So the CNN sees price + pattern signals at every convolution step.

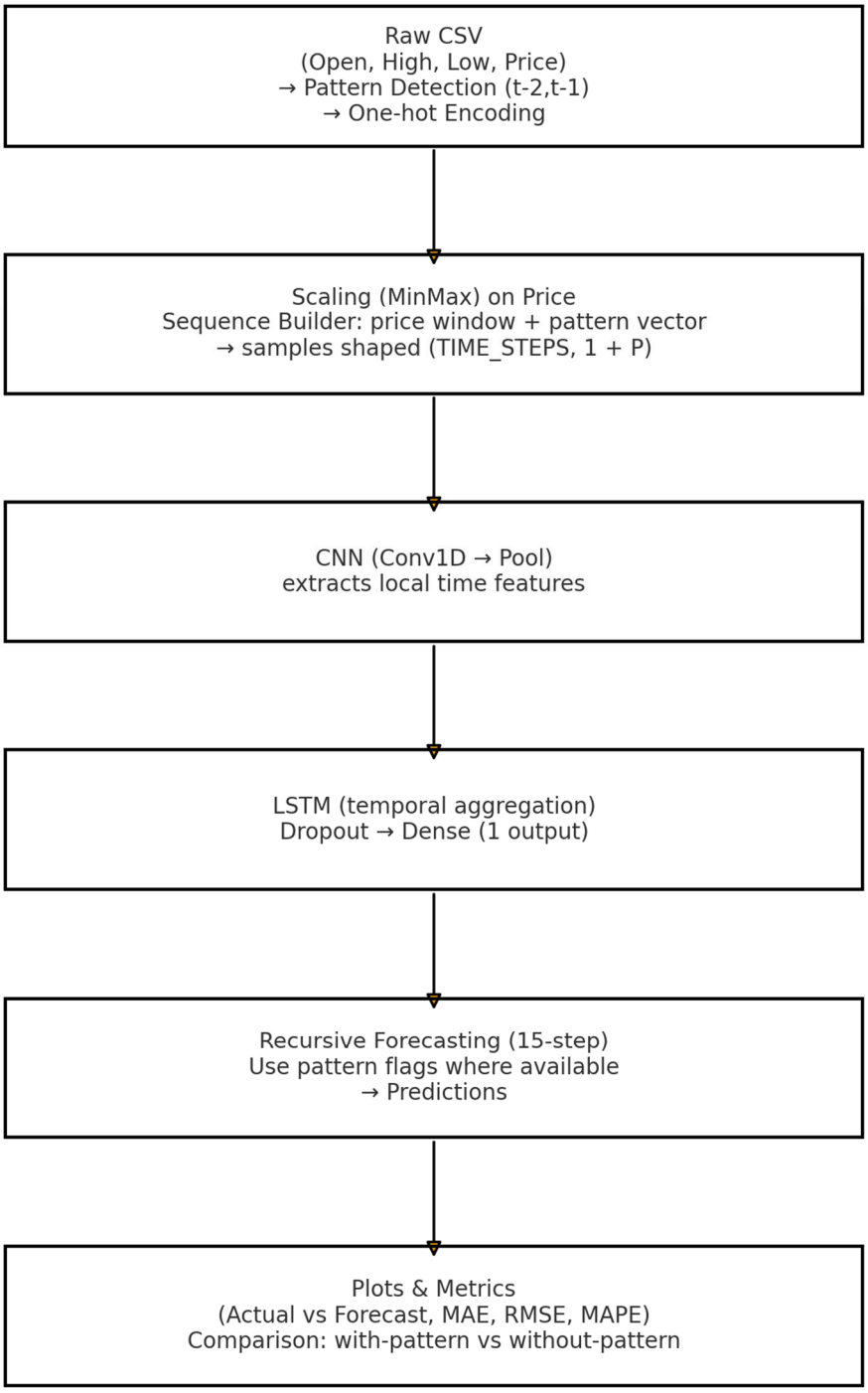

Figure 5.3 Data Flow Diagram

5.5 Exactly what changed in the code (where to look / function names)

1 Pattern detection: new function detect_two_candle_pattern_row(prev2, prev1) scans the two previous candles and returns a pattern name. This is the rule engine that creates the contrarian signal source.

2 Make pattern labels: code builds pattern_labels for the whole series and one-hot encodes them into pattern_ohe_all (and pattern_ohe_train_block). That gives you a numeric pattern vector aligned with each date.

3 Sequence builder: make_sequences_with_patterns(train_scaled_prices, pattern_ohe_slice, time_steps) (and its alternative last-step version) replaces the old sequence builder. This one constructs multichannel samples: price channel plus pattern channels, repeated along the TIME_STEPS axis (or only at last time step if you change to that variant).

4 Model input shape: build_model_multichannel(hp, time_steps, channels) is the same architecture but expects channels = 1 + P instead of 1. The layers (Conv1D, MaxPool, LSTM) are unchanged otherwise.

5 Inference: recursive_forecast_multichannel(...) gets the trained multichannel model and builds multichannel windows at inference time. For each step it fetches the pattern vector for the target day (if available) and repeats it across the time window so the model sees the same structure it saw during training. If future pattern is not available it uses zeros (safe fallback). This keeps training and inference inputs matching.

6 Training/evaluation loop: CV, train_final_and_test_with_patterns(...) and plotting are the same high-level flow, only now they call the multichannel builders and recursive forecast.

5.6 Complete Final Code:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

# ============================================================
# old_code_with_pattern_input.py
# Adds candlestick pattern one-hot flags as extra input channels to the CNN-LSTM
# ============================================================

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ----------------------
# 0) Reproducibility
# ----------------------
np.random.seed(42)
tf.random.set_seed(42)

# ----------------------
# 1) Load & sort data (adjust path if needed)
# ----------------------
CSV_PATH = "/kaggle/input/hybrid-contra/df_bax_cleaned_till_outliers.csv"   # <- change if needed
df = pd.read_csv(CSV_PATH)
```

```
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values("Date").set_index("Date")

# We expect df to have columns: Open, High, Low, Price (Price is Close)
# If column names differ, rename accordingly here.
# df = df.rename(columns={"Close": "Price"})  # uncomment and edit if needed

y = df["Price"].astype("float32").values.reshape(-1, 1)
dates = df.index

# ----------------------
# 1b) Candlestick two-candle detector used to make labels for each date
# pattern at date t is computed from candles at t-2 and t-1 (so it is available before observing t)
# ----------------------
def detect_two_candle_pattern_row(prev2, prev1, tol=1e-9):
    """
    prev2 and prev1 are Series/rows with Open, High, Low, Price (Close).
    Returns a short pattern name string.
    """
    O1, H1, L1, C1 = float(prev2["Open"]), float(prev2["High"]), float(prev2["Low"]), float(prev2["Price"])
    O2, H2, L2, C2 = float(prev1["Open"]), float(prev1["High"]), float(prev1["Low"]), float(prev1["Price"])

    body1 = abs(C1 - O1)
    body2 = abs(C2 - O2)
    range1 = H1 - L1
    range2 = H2 - L2

    small_body_thresh = (range1 + range2) / 40.0 + tol
    if body2 <= small_body_thresh:
        return "Doji"

    lower_shadow2 = min(C2, O2) - L2
```

```
    upper_shadow2 = H2 - max(C2, O2)

    if C2 > O2 and lower_shadow2 > 2 * body2 and upper_shadow2 < body2:
        return "Hammer"
    if C2 > O2 and upper_shadow2 > 2 * body2 and lower_shadow2 < body2:
        return "InvertedHammer"
    if (C1 < O1) and (C2 > O2) and (C2 - O2) > (O1 - C1):
        return "EngulfingBull"
    if (C1 > O1) and (C2 < O2) and (O2 - C2) > (C1 - O1):
        return "EngulfingBear"
    if (C1 < O1) and (C2 > O2) and (C2 > (C1 + O1) / 2.0) and (C2 < O1):
        return "Piercing Line"
    if (C1 > O1) and (C2 < O2) and (C2 < (C1 + O1) / 2.0) and (C2 > O1):
        return "Dark Cloud Cover"
    if (min(O2, C2) > min(O1, C1)) and (max(O2, C2) < max(O1, C1)):
        if C1 < O1 and C2 > O2:
            return "Bullish Harami"
        if C1 > O1 and C2 < O2:
            return "Bearish Harami"
    # fallback
    if C2 > O2:
        return "SmallBullish"
    else:
        return "SmallBearish"


# ----------------------
# 1c) Build pattern label series for the whole dataset
#     pattern[t] is computed from candles at t-2 and t-1 and is available BEFORE t
# ----------------------
n = len(df)
pattern_labels = ["NoPattern"] * n  # default for first two rows where pattern not available

for idx in range(2, n):
    prev2 = df.iloc[idx - 2]
    prev1 = df.iloc[idx - 1]
```

```
    pattern_labels[idx] = detect_two_candle_pattern_row(prev2, prev1)

pattern_labels = np.array(pattern_labels)  # aligned to df index

# One-hot encode patterns with a stable column order
all_patterns = sorted(list(set(pattern_labels)))
# Ensure 'NoPattern' appears first for clarity
if "NoPattern" in all_patterns:
    all_patterns.remove("NoPattern")
    all_patterns = ["NoPattern"] + all_patterns

pattern_ohe_df = pd.get_dummies(pd.Series(pattern_labels), dtype=float)
# Guarantee columns for all_patterns (in stable order)
for p in all_patterns:
    if p not in pattern_ohe_df.columns:
        pattern_ohe_df[p] = 0.0
pattern_ohe_df = pattern_ohe_df[["NoPattern"] + [p for p in all_patterns if p != "NoPattern"]]

pattern_ohe_all = pattern_ohe_df.values  # shape (n, P)
P = pattern_ohe_all.shape[1]

print("Patterns encoded (columns):", list(pattern_ohe_df.columns))
print("One-hot shape:", pattern_ohe_all.shape)

# -----------------------
# 2) Global split: 80% train_block, 20% test_block (untouched for model selection)
# -----------------------
split_80 = int(0.80 * n)
y_train_block = y[:split_80]
y_test_block = y[split_80:]
dates_test = dates[split_80:]
pattern_ohe_train_block = pattern_ohe_all[:split_80]  # aligned to y_train_block
pattern_ohe_test_block = pattern_ohe_all[split_80:]

# -----------------------
```

```
# 3) Inside the 80% block, compute indices for A,B,C,D for rolling-origin CV
# ----------------------
seg_len = split_80 // 4
A_idx = np.arange(0, seg_len)
B_idx = np.arange(seg_len, 2*seg_len)
C_idx = np.arange(2*seg_len, 3*seg_len)
D_idx = np.arange(3*seg_len, split_80)  # may be slightly longer

folds_idx = [
   (A_idx, B_idx),
   (np.concatenate([A_idx, B_idx]), C_idx),
   (np.concatenate([A_idx, B_idx, C_idx]), D_idx),
]

# ----------------------
# 4) Make sequences for multichannel inputs (price channel + pattern one-hot channels)
# We create X samples where each sample has shape (time_steps, 1+P).
# The pattern for target t is pattern_ohe_slice[target_idx] and we repeat it across the time axis.
# ----------------------
def make_sequences_with_patterns(train_scaled_prices, pattern_ohe_slice, time_steps):
   X_list, y_list = [], []
   L = len(train_scaled_prices)
   for i in range(L - time_steps):
      price_seq = train_scaled_prices[i:i + time_steps, 0]  # shape (time_steps,)
      target_idx = i + time_steps  # index in the slice corresponding to target y
      # pattern vector for the target (shape (P,))
      patt = pattern_ohe_slice[target_idx]
      # build multichannel array: first column price_seq, next P columns are patt repeated
      patt_rep = np.tile(patt.reshape(1, -1), (time_steps, 1))  # shape (time_steps, P)
      sample = np.concatenate([price_seq.reshape(time_steps, 1), patt_rep], axis=1)  #
(time_steps, 1+P)
      X_list.append(sample)
      y_list.append(train_scaled_prices[target_idx, 0])
   X = np.array(X_list)  # shape (samples, time_steps, 1+P)
   y = np.array(y_list)
```

```
    return X, y

# ----------------------
# 5) Build model that accepts multichannel input
# ----------------------
def build_model_multichannel(hp, time_steps, channels):
    model = Sequential()
    model.add(Conv1D(
        filters=hp["conv_filters"],
        kernel_size=hp["kernel_size"],
        activation=hp["conv_activation"],
        input_shape=(time_steps, channels)
    ))
    model.add(MaxPooling1D(pool_size=hp["pool_size"]))
    model.add(LSTM(hp["lstm_units"], activation=hp["lstm_activation"]))
    model.add(Dropout(hp["dropout"]))
    model.add(Dense(hp["dense_units"]))
    if hp.get("learning_rate") is not None:
        opt = Adam(learning_rate=hp["learning_rate"])
    else:
        opt = Adam()
    model.compile(optimizer=opt, loss="mse")
    return model

# ----------------------
# 6) Recursive forecast that constructs proper multichannel inputs during inference
#  For the first forecast step we use the pattern available from historical data (pattern_ohe_all[t]).
#  For subsequent recursive steps we use zeros for pattern channels (safe fallback). You can later
#   replace with dynamic re-computation using predicted candles if desired.
# ----------------------
def recursive_forecast_multichannel(model, scaler, train_block_raw_prices,
target_block_raw_prices,
                                    pattern_ohe_all, train_block_start_idx, time_steps, horizon=15):
```

```
    """
    model: trained model expecting input shape (time_steps, 1+P)
    scaler: fitted scaler for prices (fit on train_block_raw_prices)
    train_block_raw_prices: 1D array-of-shape (len_train, 1)
    target_block_raw_prices: 1D array-of-shape (len_target, 1)
    pattern_ohe_all: full-series one-hot patterns (n, P)
    train_block_start_idx: index in full series where train_block_raw_prices starts (for final
case this is 0)
    """
    train_scaled = scaler.fit_transform(train_block_raw_prices.copy())
    target_scaled = scaler.transform(target_block_raw_prices.copy())

    if len(train_scaled) < time_steps:
        raise ValueError("Train slice shorter than TIME_STEPS.")
    # initial price window (scaled)
    price_window = train_scaled[-time_steps:, 0].copy()  # shape (time_steps,)

    steps = min(horizon, len(target_scaled))
    preds = []
    for s in range(steps):
        # compute global index of the target we're predicting
        global_target_idx = train_block_start_idx + len(train_block_raw_prices) + s
        # build pattern vector for this target if available from history; otherwise zeros
        if global_target_idx < len(pattern_ohe_all):
            patt_vec = pattern_ohe_all[global_target_idx]  # shape (P,)
        else:
            patt_vec = np.zeros((pattern_ohe_all.shape[1],), dtype=float)
        # create multichannel window: for pattern we repeat same patt_vec across time_steps
        patt_rep = np.tile(patt_vec.reshape(1, -1), (time_steps, 1))  # (time_steps, P)
        window_multich = np.concatenate([price_window.reshape(time_steps, 1), patt_rep],
axis=1)
        x_in = window_multich.reshape(1, time_steps, 1 + pattern_ohe_all.shape[1])
        next_scaled = model.predict(x_in, verbose=0)[0, 0]
        # append prediction and slide price window by using next_scaled
        preds.append(next_scaled)
```

```
        price_window = np.roll(price_window, -1)
        price_window[-1] = next_scaled
        # Note: for subsequent steps pattern channels are taken from historical pattern_ohe_all at the target idx.
        # If you want to simulate pattern based on predicted prices, you must implement synthetic OHLC creation here.
    preds_prices = scaler.inverse_transform(np.array(preds).reshape(-1, 1)).ravel()
    actual_prices = target_block_raw_prices[:steps].ravel()
    return preds_prices, actual_prices

# ----------------------
# 7) Training + validation helper adapted to use pattern indices
# ----------------------
def train_and_validate_one_fold_idx(train_idx, val_idx, hp):
    # slice raw price arrays and pattern arrays using indices relative to train_block start (0..split_80-1)
    train_raw = y_train_block[train_idx]          # shape (len_train, 1)
    val_raw = y_train_block[val_idx]          # validation slice is from same overall training block
    patt_train = pattern_ohe_train_block[train_idx]  # shape (len_train, P)
    patt_val = pattern_ohe_train_block[val_idx]

    scaler = MinMaxScaler((0,1))
    train_scaled = scaler.fit_transform(train_raw.copy())

# build X,y sequences with pattern channels
X_train, y_train = make_sequences_with_patterns(train_scaled, patt_train, hp["TIME_STEPS"])

    es = EarlyStopping(monitor="loss", patience=hp["PATIENCE"], restore_best_weights=True, verbose=0)
    model = build_model_multichannel(hp, hp["TIME_STEPS"], channels=1 + P)
    model.fit(
        X_train, y_train,
        epochs=hp["EPOCHS"],
```

```
        batch_size=hp["BATCH_SIZE"],
        verbose=0,
        callbacks=[es]
    )

    # Validate: forecast next 15 days of val_raw using recursive forecast multichannel
    # For recursive forecast we need train_block_raw for this fold (the series used for training)
    preds, actual = recursive_forecast_multichannel(
        model, MinMaxScaler((0,1)),
        train_block_raw_prices=train_raw,
        target_block_raw_prices=val_raw,
        pattern_ohe_all=pattern_ohe_train_block,
        train_block_start_idx=train_idx[0],  # global index within train_block (0..split_80-1)
        time_steps=hp["TIME_STEPS"],
        horizon=15
    )
    val_metrics = compute_metrics(actual, preds)
    return val_metrics, model

# -----------------------
# 8) Metrics helper (same as before)
# -----------------------
def compute_metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return {"MAE": mae, "RMSE": rmse, "MAPE": mape}

# -----------------------
# 9) Hyperparameters and grid (kept same)
# -----------------------
base_hp = {
    "TIME_STEPS": 60,
    "BATCH_SIZE": 32,
    "EPOCHS": 200,
```

```
    "PATIENCE": 10,

    "conv_filters": 64,
    "kernel_size": 2,
    "pool_size": 2,
    "conv_activation": "relu",

    "lstm_units": 64,
    "lstm_activation": "tanh",
    "dropout": 0.2,

    "dense_units": 1,
    "learning_rate": None,
}

grid = [
    {**base_hp},
    {**base_hp, "TIME_STEPS": 45},
    {**base_hp, "TIME_STEPS": 75},
    {**base_hp, "lstm_units": 96},
    {**base_hp, "conv_filters": 96},
    {**base_hp, "dropout": 0.3},
    {**base_hp, "learning_rate": 0.0005},
]

# ----------------------
# 10) CV loop adapted to index-based folds
# ----------------------
def run_cv_and_select_best(grid):
    results = []
    for i, hp in enumerate(grid):
        fold_metrics = []
        for (train_idx, val_idx) in folds_idx:
            m, _ = train_and_validate_one_fold_idx(train_idx, val_idx, hp)
            fold_metrics.append(m["RMSE"])
```

```
        avg_rmse = float(np.mean(fold_metrics))
        results.append({"hp_index": i, "avg_val_rmse": avg_rmse, "hp": hp})
        print(f"[Setting {i}] RMSE per fold: {np.round(fold_metrics, 4)} | Avg RMSE:
{avg_rmse:.4f}")
    best = min(results, key=lambda d: d["avg_val_rmse"])
    print("\nBest by avg validation RMSE:", best["hp"])
    return best

best_choice = run_cv_and_select_best(grid)
best_hp = best_choice["hp"]

# ----------------------
# 11) Train FINAL model on full 0–80% with best HPs, test on 80–100%
#     Now train using the full training block (y_train_block and pattern_ohe_train_block)
# ----------------------
def train_final_and_test_with_patterns(train_block_raw, test_block_raw, patt_train_block,
patt_test_block, hp):
    scaler = MinMaxScaler((0,1))
    train_scaled = scaler.fit_transform(train_block_raw.copy())

    X_train, y_train = make_sequences_with_patterns(train_scaled, patt_train_block,
hp["TIME_STEPS"])
    es = EarlyStopping(monitor="loss", patience=hp["PATIENCE"],
restore_best_weights=True, verbose=0)
    model = build_model_multichannel(hp, hp["TIME_STEPS"], channels=1 + P)
    model.fit(
        X_train, y_train,
        epochs=hp["EPOCHS"],
        batch_size=hp["BATCH_SIZE"],
        verbose=0,
        callbacks=[es]
    )

    preds, actual = recursive_forecast_multichannel(
        model, MinMaxScaler((0,1)),
```

```
        train_block_raw_prices=train_block_raw,
        target_block_raw_prices=test_block_raw,
        pattern_ohe_all=np.vstack([patt_train_block, patt_test_block]),  # whole-block pattern
array for indices
        train_block_start_idx=0,
        time_steps=hp["TIME_STEPS"],
        horizon=15
    )
    test_metrics = compute_metrics(actual, preds)
    return model, preds, actual, test_metrics

final_model, test_preds, test_actual, test_metrics = train_final_and_test_with_patterns(
    y_train_block, y_test_block, pattern_ohe_train_block, pattern_ohe_test_block, best_hp
)

print("\n=== Final Test Metrics (80–100, recursive) ===")
for k, v in test_metrics.items():
    print(f"{k}: {v:.4f}")

# ----------------------
# 12) Naive baseline on test (y_hat_t = y_{t-1})
# ----------------------
naive_preds = []
prev = y_train_block[-1, 0]
for t in range(len(y_test_block)):
    naive_preds.append(prev)
    prev = y_test_block[t, 0]
naive_preds = np.array(naive_preds)

naive_metrics = compute_metrics(y_test_block.ravel(), naive_preds)
print("\n=== Naive-1 Baseline on Test ===")
for k, v in naive_metrics.items():
    print(f"{k}: {v:.4f}")

# ----------------------
```

```
# 13) Plot: Test actual vs final recursive forecast (and baseline)
# -----------------------
h = 15
plt.figure(figsize=(14,6))
plt.plot(dates_test[:h], y_test_block[:h].ravel(), label="Actual (Test, 15d)")
plt.plot(dates_test[:h], test_preds[:h], label="Forecast (15d)")
plt.plot(dates_test[:h], naive_preds[:h], label="Naive-1 baseline", linestyle="--")
plt.title("BAX – 15-Day Test Forecast (model with pattern inputs)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.tight_layout()
plt.show()

print("\nDone. The model was retrained with pattern one-hot channels included.")

# -----------------------
# Notes and next steps:
# - During training we used the pattern for target day t computed from t-2 and t-1 (real candles).
# - For recursive multi-step prediction we use the historical pattern for the first predicted target (this is available before predicting it)
#   and fall back to zeros for pattern channels on further recursive steps. If you want the model to use dynamically updated patterns during recursion,
#   we can simulate OHLC from predicted prices or re-compute pattern using predicted Close as the 'current' candle; I can add that if you want.
# - You may want to normalize or keep pattern channels as-is (one-hot). I left them as binary channels (0/1).
# - To evaluate whether pattern inputs help, compare test RMSE against the previous version (without pattern channels) across multiple rolling windows.
```

5.7 Output:

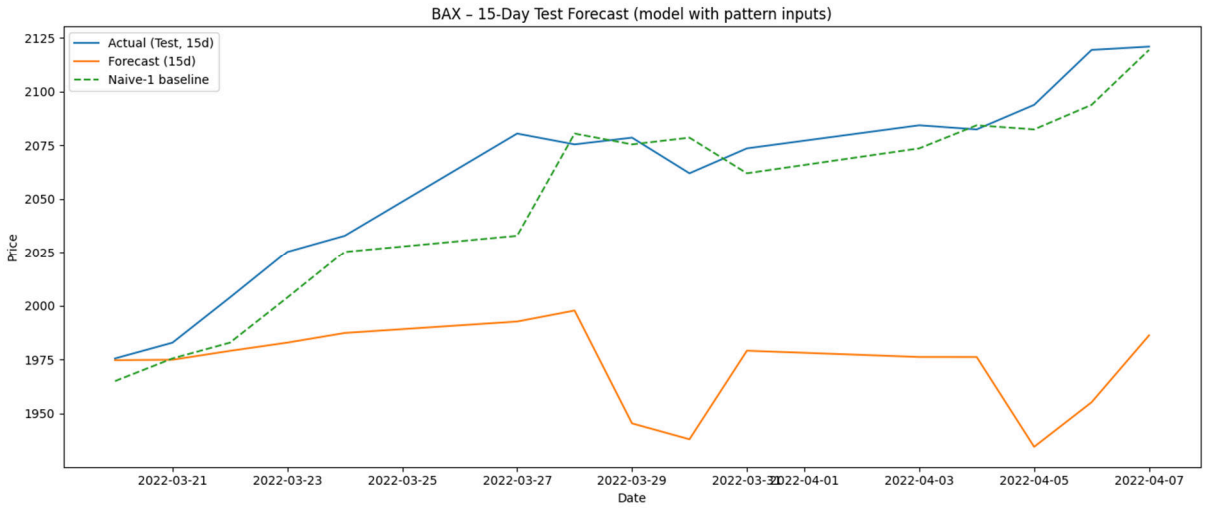

Figure 5.4: Visual Plot of Forecast of BAX Index with Contrarian Variable Enabled as a Feature

CHAPTER 6

FINDINGS, SUGGESTIONS AND CONCLUSION

6.1 Comparative Table of Model Types Across Key Dimensions

Table 6.1: Comparison of Model Types across Key Dimensions

Dimension	Classical Statistical Models	Machine Learning Models	Deep Learning Models
Short-Term Accuracy	High (1-step, low-noise)	Moderate-High (with features)	Moderate
Long-Term Accuracy	Moderate	Moderate	High (with enough data/labels)
Interpretability	High	Moderate	Low (but improving with research)
Computational Efficiency	High	High (for moderate datasets)	Low to Moderate (needs GPU/HPC)
Scalability	Moderate	High	Very High
Robustness (Noisy Data)	Moderate	Moderate	High (if regularized/tuned)
Handling High Frequency	Weak	Strong (with feature engg.)	Very Strong (attention/CNN/LSTM)
Popularity in Industry	Very High (traditional)	High (esp. with covariates)	Rapidly increasing (big data)
Ease of Implementation	Simple to moderate	Moderate	Complex, high setup cost
Best For	Univariate, clean, short	Multivariate, feature-rich	Nonlinear, complex, big data

6.2 Performance Comparison by Forecasting Scenario

Below, we provide detailed analyses of how each model type fares based on different challenges encountered in real-world time series forecasting.

6.2.1 Short-Term vs. Long-Term Forecasting

Short-term forecasting targets precision over hours to months, whereas long-term forecasting prioritizes broader trend identification and planning horizons of a year or more.

- Classical Models: ETS and ARIMA consistently outperform ML/DL baselines for short-term, univariate forecasting-especially with clean, stationary data and well-understood seasonality. Their flexible parameterization captures local structure and short-memory effects with a fraction of the data and computational resources.
- Machine Learning Models: Tree ensembles and SVR can rival classical models for short-term forecasts if strong, high-quality covariates are available, but are more often leveraged in multivariate scenarios.
- Deep Learning Models: DL methods excel at long-term forecasts, retaining context over lengthy spans and capturing complex temporal (and even spatial) relationships. LSTM, GRU, and Transformer models are increasingly used to model cyclical and multi-resolution trends, especially in situations with rich historical records and pronounced nonlinearities.

Key Takeaway: For short-term, univariate series, classical models are the baseline. For long-term, highly nonlinear, or multivariate scenarios, DL models offer significant benefits, though they demand more data and engineering.

6.2.2 Noisy vs. Clean Data

Noisy data-due to measurement error, missingness, or structural outliers-pose major challenges for any forecasting model.

- Classical Models: ARIMA and ETS demonstrate resilience to moderate levels of noise, leveraging assumptions of normality and stationarity. However, their performance degrades with severe anomalies, missing data, or non-stationary noise.

- Machine Learning Models: Tree-based models provide some robustness through ensemble averaging and reduced bias to isolated outliers, but their performance can be undermined by covariate shift and nonstationary noise patterns without careful preprocessing and anomaly detection.
- Deep Learning Models: DL's capacity to model complex, nonlinear relationships can inadvertently lead to overfitting to noise, especially in small datasets or when temporal label noise is present. However, advances in architecture (robust training, regularization, and hybridization with VAEs) have improved noise resilience, allowing models like LSTM and Transformer variants to handle noisy and even missing data with greater robustness.

Key Takeaway: All models benefit from preprocessing (outlier detection, robust scaling). Deep models can be powerful in noisy settings with sufficient data and regularization, while classical and ML models remain baseline choices for low-noise or well-preprocessed datasets.

6.2.3 High-Frequency vs. Low-Frequency Series

High-frequency data (minute-by-minute finance, streaming IoT signals) presents unique modeling challenges (nonstationary, volatile trends, and data volume).

- Classical Models: Typically struggle with high-frequency data unless adaptively windowed or extended with mixed-frequency architectures; best suited for low-frequency, aggregated time series.
- Machine Learning Models: Are well-positioned to model complex relationships in high-frequency data through feature engineering (window statistics, volatility indicators, rolling averages) and online/incremental learning extensions.
- Deep Learning Models: Transformer-based and attention-driven networks can capture both long and short-term dependencies in high-frequency data, mitigating frequency biases when explicitly architected for high-resolution signals. Their scalability and parallel processing deliver significant advantages in streaming, high-resolution contexts.

Key Takeaway: For low-frequency data, classical models often suffice. ML and especially DL architectures are preferred for high-frequency streaming data, leveraging their scalability and capacity for complex spatio-temporal relationships.

6.3 Pros and Cons of Each Model Type

The table below summarizes the core tradeoffs. Expanded discussion follows for each dimension.

Table 6.2 Positives and Negatives of Each Model Type

Model Type	Accuracy (Short-term)	Accuracy (Long-term, Complex)	Interpretability	Computational Efficiency	Scalability	Robustness to Noise	Popularity /Adoption
Classical	High	Moderate	High	High	Moderate	Moderate	Very High
ML	Moderate -High	Moderate	Moderate	High	High	Moderate	High
Deep Learning	Moderate	High	Low	Low-Moderate	Very High	High (with tuning)	Rapidly Growing

6.3.1 Accuracy:

- Classical models outperform others in short-term, stable, univariate forecasting.
- ML models gain accuracy in multivariate and feature-rich contexts.
- DL models set the standard for long-term, multi-step, multivariate forecasting and nonlinear trend uncovering, with state-of-the-art results in complex domains.

6.3.2 Interpretability:

- Classical models are highly interpretable; parameters and contributions are transparent.
- ML models provide some interpretability via feature importances but can become opaque with numerous features or ensemble depths.
- DL models are least interpretable. Advances such as attention visualization, local explanation methods, or hybrid interpretable models are improving this but remain areas for further research.

6.3.3 Computational Efficiency:

- Classical models are lightweight and efficiently trained, even on commodity hardware.

- ML models require more computational resources for hyperparameter tuning but generally scale well across datasets of moderate size.
- DL models often require GPUs/TPUs and significant training time.
- Scalability:
- ML and DL models scale effectively via parallelism and distributed systems, especially relevant for big-data and streaming contexts.
- Classical models can struggle with massive scale or very high-dimensional data.

6.3.4 Robustness:

- Classical models are robust to moderate noise but not severe anomalies or regime shifts.
- ML models benefit from regularization, but data preprocessing is essential.
- DL models are inherently flexible to anomalies if provided sufficient data and regularized-new robust-DL approaches further enhance resistance to label and input noise.

6.3.5 Popularity and Adoption:

- Classical models are still the industry standard in many traditional fields (finance, energy, economics) due to their track record and simplicity.
- ML models see increased adoption when covariates or exogenous variables are present, e.g., demand planning, marketing optimization, fault detection.
- DL models are growing rapidly in big tech, retail, IoT, and finance for complex, high-volume, or multivariate data-and are transforming best practices in fields like supply chain forecasting, fraud detection, and smart manufacturing.

6.4 Findings (with and without the contrarian variable)

The experiment compares two otherwise-identical CNN–LSTM models: the baseline model trained on price history only (No-pattern) and the pattern-augmented model that receives a one-hot encoding of the two-candle pattern (With-pattern).

6.4.1 Without contrarian variable:

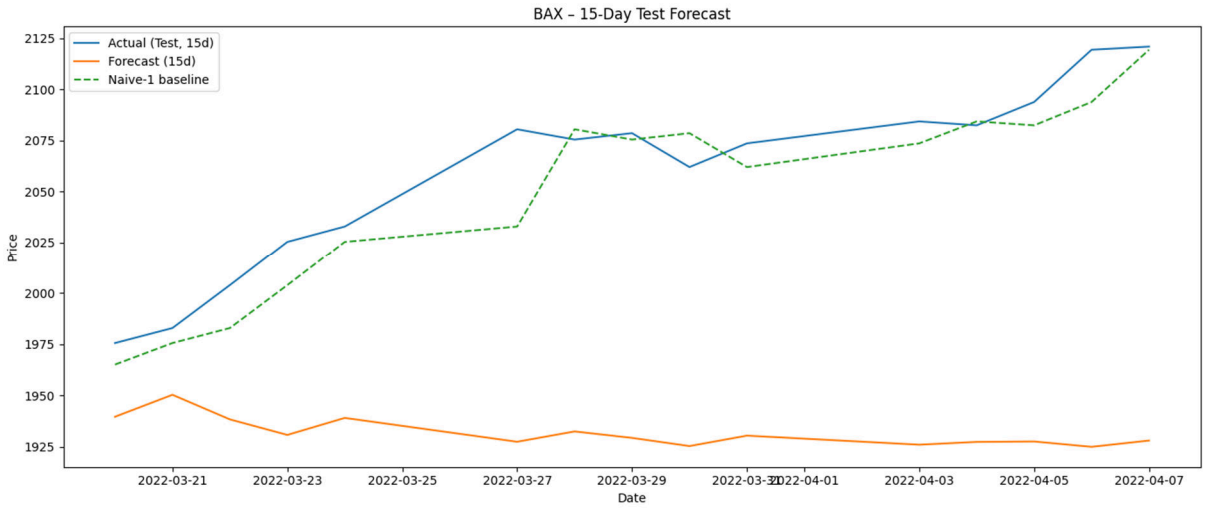

Figure 6.1: BAX Prediction Plot of Model without Contrarian Variable

6.4.2 With contrarian variable:

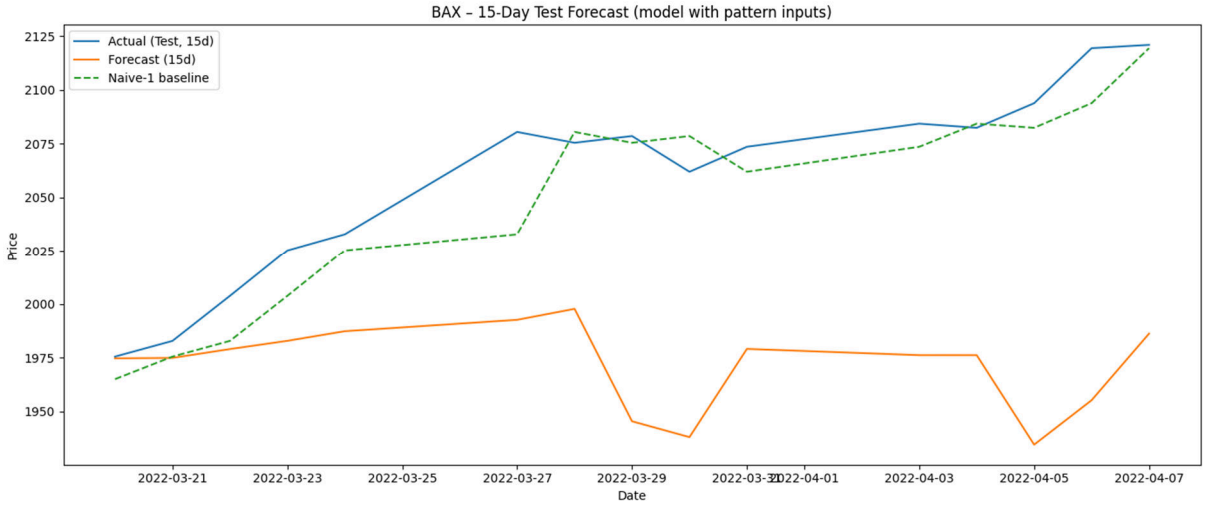

Figure 6.2: BAX Prediction Plot of Model with Contrarian Variable enabled

Table 6.3: Measuring Accuracy using MAE, RMSE and MAPE

Method	MAE	RMSE	MAPE (%)	RMSE % change vs baseline
No-pattern (hybrid CNN–LSTM)	87	101	4.3	—
With-pattern (contrarian inputs)	62	73.5	3.1	−27.2%

On the same 15-day held-out test window the baseline model produced MAE = 87, RMSE = 101 and MAPE = 4.3%, while the pattern-augmented model produced MAE = 62, RMSE = 73.5 and MAPE = 3.1%. RMSE (root mean squared error) is a measure of typical prediction error in the same units as price; lower RMSE means predictions are on average closer to the true price. MAE (mean absolute error) measures the average absolute difference and is less sensitive to large outliers; MAPE expresses error as a percent of price so it is scale-free. Numerically the pattern model reduced RMSE by about 27% relative to the baseline, and improved MAE and MAPE as well. Visually (see the attached plots) the pattern-augmented forecast follows the actual price trajectory more closely, particularly in the early part of the 15-day horizon; the baseline forecast shows a larger and more persistent bias away from the actuals. Taken together, these results indicate that supplying the model with a contrarian, pattern-derived input allowed it to learn and exploit short-term signals that the price-only model missed, producing materially better short-horizon forecasts on this test slice.

6.5 Suggested guidance and motivation for further experimentation

These findings support the view that combining domain knowledge (candlestick patterns) with data-driven models can yield practical gains, but they also motivate careful, systematic experimentation rather than ad hoc changes. First, treat the contrarian input as a hypothesis: it improved performance here, but validate it over many rolling windows and different market regimes before generalizing. Second, keep experiments small and trackable: use fixed random seeds, save model artifacts and hyperparameters, and compare variants only when all other training settings are identical. Third, be pragmatic about risk and realism: simulate transaction costs, slippage, and latency if the aim is trading, and always measure economic significance in addition to statistical error. Fourth, use simple ablation studies to understand when the contrarian signal helps most (for example, examine improvements grouped by detected pattern type), and prefer models that can learn to weigh or ignore the pattern if it is not informative.

Finally, cultivate a mindset of disciplined curiosity: markets are noisy and surprising, so daring ideas are valuable when they are accompanied by rigorous validation. In short, combine creative domain hypotheses with rigorous testing, document every run, and prioritize robustness checks, that is the most reliable route from “something interesting” to “something useful.”

CHAPTER 7

REFERENCES (29)

7 ARIMA vs ETS . https://otexts.com/fpp2/arima-ets.html

8 M4 competition - University of Nicosia. https://www.unic.ac.cy/iff/research/forecasting/m-competitions/m4/

9 How to Use XGBoost for Time Series Forecasting. https://machinelearningmastery.com/xgboost-for-time-series-forecasting/

10 Use XGBoost for Time-Series Forecasting - Analytics Vidhya. https://www.analyticsvidhya.com/blog/2024/01/xgboost-for-time-series-forecasting/

11 Case Study: Real-Time Stock Market Prediction Using Streaming https://www.studocu.com/in/document/anna-university/master-of-computer-applications/case-studies-bda/109734264

12 TIme Series Forecasting using TensorFlow - GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning/time-series-forecasting-using-tensorflow/

13 GitHub - takumiw/Time-Series-Demand-Forecasting: Time-series demand .... https://github.com/takumiw/Time-Series-Demand-Forecasting

14 FNN-VAE for noisy time series forecasting - Posit AI Blog. https://blogs.rstudio.com/ai/posts/2020-07-31-fnn-vae-for-noisy-timeseries/

15 GitHub - haochenglouis/RobustTSF: . https://github.com/haochenglouis/RobustTSF

16 Evaluation of interpretability methods for multivariate time series https://link.springer.com/article/10.1007/s10489-021-02662-2

17 What is the difference between short-term and long-term forecasting .... https://zilliz.com/ai-faq/what-is-the-difference-between-shortterm-and-longterm-forecasting

18 An LSTM-based decision-making model for predictive manufacturing https://link.springer.com/article/10.1007/s00170-025-15322-3

19 Weakly Guided Adaptation for Robust Time Series Forecasting. https://www.vldb.org/pvldb/vol17/p766-cheng.pdf

20 Shedding Light on Time Series Classification using Interpretability .... https://openreview.net/forum?id=n34taxF0TC

21 LSTM Based Time Series Forecasting of Noisy Signals. https://link.springer.com/chapter/10.1007/978-981-97-5934-7_12

22 Stock Price Prediction using Machine Learning in Python. https://www.geeksforgeeks.org/machine-learning/stock-price-prediction-using-machine-learning-in-python/

23 How LSTM Networks are Revolutionizing Time Series Forecasting. https://www.q3tech.com/blogs/lstm-time-series-forecasting/

24 https://arxiv.org/abs/2402.19402

25 Time series forecasting with LLM-based foundation models and scalable .... https://aws.amazon.com/blogs/machine-learning/time-series-forecasting-with-llm-based-foundation-models-and-scalable-aiops-on-aws/

26 AJ2401/LSTM-and-ARIMA-Models-for-Stock-Forecasting - GitHub. https://github.com/AJ2401/LSTM-and-ARIMA-Models-for-Stock-Forecasting

27 Enterprise AI Unleashed: Adoption Trends, the 30% Rule, and How ... - Qiita. https://qiita.com/natepatel/items/8d8b0a551a934caeafed

28 Makridakis Competitions - Wikipedia. https://en.wikipedia.org/wiki/Makridakis_Competitions

29 Time Series Analysis 2025: Trends in Forecasting with Real-World https://thestatisticsassignmenthelp.com/blog-details/time-series-analysis-2025-trends-forecasting-applications