VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Stock Price Forecasting System for Netflix: Report Structure

No	Name	ID Student
1	Vo Hoai Bao	ITITIU21038
2	Nguyen Phuc Vinh	ITITIU21350
3	Thai Thanh Phat	ITITIU21274
4	Pham Trung Dung	ITITIU21007
5	Nguyen Duc Duy	ITITIU21008

A report submitted for the course
Scalable and Distributed Computing – IT139IU
Lecturer: Mai Hoang Bao An

Ho Chi Minh City, Vietnam
2025

Contents

List of Figures

List of Tables

Abstract

This report details the development of a scalable stock price forecasting system using Python and PySpark, designed to enable data-driven investment decisions in volatile financial markets. The project involves analyzing historical stock data using `yfinance` to build predictive models, including Linear Regression, ARIMA, and deep learning models (LSTM), with a real-time forecasting pipeline implemented through Streamlit dashboards. Key tasks include preprocessing data to handle missing values and timestamps, engineering features using `scikit-learn` and PySpark, and optimizing model performance using TensorFlow and Keras. The system includes a comprehensive dashboard for visualization and monitoring, real-time processing capabilities, and model optimization features. The project is containerized for easy deployment and includes automated testing through `pytest`. The implementation leverages the modern Python data science stack including `pandas`, `numpy`, and `scikit-learn` for data processing, while using PySpark for distributed computing capabilities. The system is designed to be maintainable and scalable, with clear separation of concerns between data processing, model training, and visualization components.

Chapter 1

Introduction

In today's fast-paced and volatile financial markets, organizations like Netflix require advanced analytical tools to navigate the complexities of stock price movements and make informed investment decisions. The ability to predict stock performance with precision is critical for optimizing portfolio management, mitigating risks, and maintaining a competitive edge in the dynamic financial landscape. This project addresses this need by developing a scalable, robust stock price forecasting system for Netflix, leveraging Python and PySpark to deliver real-time, actionable insights. By integrating historical stock data accessed through the `yfinance` API with state-of-the-art machine learning techniques, the system empowers Netflix's investment team to respond swiftly to market dynamics, enhancing strategic decision-making and aligning with the company's broader financial objectives.

The significance of this initiative lies in its ability to provide accurate and timely forecasts amidst market uncertainty, enabling Netflix to capitalize on opportunities and manage risks effectively. The system incorporates a suite of predictive models, including Linear Regression for interpretable baseline predictions, ARIMA and SARIMA for capturing time-series trends and seasonality, and Long Short-Term Memory (LSTM) neural networks for modeling complex, long-term dependencies in stock prices. These models are complemented by a real-time forecasting pipeline built on PySpark's distributed computing capabilities, ensuring scalability and efficiency in processing large-scale market data. Additionally, an interactive Streamlit dashboard, powered by Plotly, provides intuitive visualizations of trends, predictions, and performance metrics, fostering data-driven decision-making for stakeholders.

To enhance predictive accuracy, the system employs advanced feature engineering, incorporating technical indicators such as Simple Moving Averages (SMA), Exponential Moving Averages (EMA), and Relative Strength Index (RSI), alongside price-based metrics. These features provide a comprehensive view of market dynamics, enabling the models to capture both short-term fluctuations and long-term trends. The use of `yfinance` ensures reliable access to historical and real-time stock data. Performance is rigorously evaluated using metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared, providing clear insights into model accuracy and robustness.

Developed over a four-week period with iterative model testing and optimization, the project delivers key artifacts, including a Jupyter notebook and Python script containing PySpark code for data preprocessing, model training, and real-time forecasting, as well as a comprehensive Streamlit dashboard for interactive visualization. The system adheres to industry-standard practices, incorporating hyperparameter tuning, distributed training, and modular code organization to ensure maintainability and scalability. By addressing

technical challenges such as data scalability and model overfitting, and ensuring compliance with regulatory requirements like SEC transparency mandates and GDPR data privacy standards, the system positions itself as a high-performance solution tailored to Netflix's investment needs.

Despite its robust design, the project acknowledges several limitations. Market volatility, driven by factors such as subscriber growth fluctuations, competitive pressures, or macroeconomic shifts, may impact forecast accuracy, necessitating continuous model recalibration. Limited historical data depth, particularly for external factors like market sentiment or macroeconomic indicators, could constrain the system's ability to capture all relevant influences. To mitigate these challenges, the project employs rigorous preprocessing, feature engineering, and distributed computing techniques, while future enhancements could include integrating alternative data sources, such as social media sentiment or news analytics, to further improve predictive power.

This forecasting system represents a significant step toward enabling data-driven investment strategies for Netflix, offering a scalable and adaptable solution that aligns with the company's goal of achieving financial resilience and growth. By combining cutting-edge machine learning, real-time data processing, and user-friendly visualization, the system provides a foundation for informed decision-making, with clear pathways for ongoing refinement and expansion to meet evolving market demands.

Chapter 2

Methodology

The development of a scalable stock price forecasting system required a systematic technical approach, leveraging Python and PySpark to process historical and real-time stock data, build predictive models, and deploy a robust forecasting pipeline. This section outlines the methodology, encompassing data investigation and preprocessing, feature engineering, model development, real-time pipeline implementation, performance optimization, documentation, deployment, and training. Each component is designed to ensure accuracy, scalability, and compliance with regulatory standards for financial forecasting and data privacy.

The system employs multiple predictive models to capture different aspects of stock price behavior:

- **Linear Regression** model implemented through PySpark's ML library, chosen for its interpretability and computational efficiency.
- **LSTM neural network** with a sophisticated architecture (128-64-32 units) and dropout layers, designed to capture complex temporal patterns and long-term dependencies in stock prices.
- **SARIMA** model with seasonal components:

  `order=(2,1,2)`, `seasonal_order=(1,1,1,5)`, effectively modeling both trend and seasonal patterns in the data.

The feature engineering process includes comprehensive technical indicators such as Simple Moving Averages (SMA), Exponential Moving Averages (EMA), Relative Strength Index (RSI), and price-based metrics. Real-time data processing is handled through the `EnhancedStockForecastingPipeline` class, which implements efficient data caching, automated technical indicator calculation, and continuous prediction updates.

The implementation leverages `yfinance` for reliable data access, `Streamlit` for interactive visualization, and `Plotly` for dynamic charting.

Performance is evaluated using standard metrics such as RMSE, MAE, and R-squared, providing clear insights into prediction accuracy. These methods collectively address the project's technical and operational requirements, positioning the system as a scalable, high-performance solution for stock price forecasting.

2.1 Data Foundation and Acquisition

The Netflix stock price forecasting system is built upon a comprehensive data investigation and preprocessing pipeline designed to ensure the reliability and quality of input data

for accurate predictive modeling. The dataset was strategically sourced from two complementary sources: Kaggle's "Netflix Stock Price Prediction" dataset and the `yfinance` API. This dual-source approach leverages Kaggle's structured historical dataset for model training while utilizing `yfinance` for real-time data integration, aligning with industry best practices for financial forecasting systems.

The dataset encompasses approximately 3,800 daily records spanning from January 2010 to June 2025, providing a robust foundation for analysis. Historical Netflix stock data includes daily open, high, low, close (OHLC) prices, trading volume, and adjusted close prices. The data was loaded into a PySpark `DataFrame` to enable scalable processing and subsequently analyzed using `pandas` and `numpy` for detailed exploration, ensuring efficient handling of large-scale financial data through distributed computing capabilities.

2.1.1 Data Quality Assessment and Analysis

Initial data quality assessment revealed several critical issues that required systematic attention to ensure reliable forecasting outcomes. The analysis employed PySpark's distributed computing framework, implementing a comprehensive quality check pipeline that examined multiple data aspects simultaneously. Systematic null value detection was conducted across all columns using PySpark's `filter` operations to identify missing values, accompanied by percentage calculations for each feature to quantify data completeness.

Comprehensive outlier analysis was performed using the Interquartile Range (IQR) method, involving the calculation of quartiles (Q1, Q3) and IQR bounds for each numeric column including Open, High, Low, Close, Adjusted Close, and Volume. Outliers were systematically defined as values extending beyond 1.5 times the IQR from the quartiles, providing a statistically robust approach to anomaly detection.

The statistical analysis provided deep insights into data distribution characteristics through summary statistics encompassing mean, standard deviation, minimum, and maximum values. Visual analysis was enhanced through histograms with Kernel Density Estimation (KDE) plots and correlation heatmaps, with all visualizations systematically saved in the `distribution_plots` directory for comprehensive documentation and reference.

2.1.2 Preprocessing Pipeline Implementation

The preprocessing strategy was carefully designed to address identified data quality issues while maintaining the temporal integrity essential for time-series forecasting. Missing values were handled using a sophisticated forward-fill strategy implemented through PySpark's window functions, defined as `Window.orderBy("Date")` and executed using `coalesce` and `last` functions. This approach maintained critical temporal continuity required by time-series models such as ARIMA and LSTM that depend on uninterrupted data sequences.

Data type standardization was performed to ensure consistency across the entire dataset. Date columns were systematically converted to `DateType` while numeric columns were cast to `DoubleType` using PySpark's `cast` function. This standardization significantly improved data consistency and enabled efficient downstream computations throughout the modeling pipeline.

The cleaned and processed data was saved in a structured format as `data/preprocessed_spark_data.csv`, serving as the foundation for subsequent feature engineering and model development. This rigorous preprocessing pipeline, supported by comprehensive visualization and statistical analysis, established a solid groundwork for accurate and scalable

stock price forecasting.

2.1.3 Exploratory Data Analysis and Technical Indicators

Exploratory Data Analysis was conducted using PySpark for scalable computation and visualized through `matplotlib`, `seaborn`, and `Plotly`, yielding actionable insights into Netflix stock price behavior patterns. The analysis pipeline incorporated comprehensive statistical and visualization techniques, with all outputs systematically organized in the `distribution_plots` directory.

Time-series analysis was facilitated by the `EnhancedStockForecastingPipeline` class, which computed a comprehensive range of technical indicators including Simple Moving Averages (SMA), Exponential Moving Averages (EMA), Relative Strength Index (RSI), and other price-based metrics. These technical indicators were strategically selected for integration into the model training pipeline to enhance the system's predictive capabilities.

The analysis generated detailed boxplots (Figure 2.1) and histograms (Figure 2.2) for each numeric column to visualize distributions and identify outliers, while correlation heatmaps (Figure 2.3) explored relationships between price metrics and trading volume. Distribution plots with Kernel Density Estimation uncovered underlying data patterns that directly informed the feature engineering process and guided the selection and tuning of the forecasting models: Linear Regression, LSTM, and SARIMA.

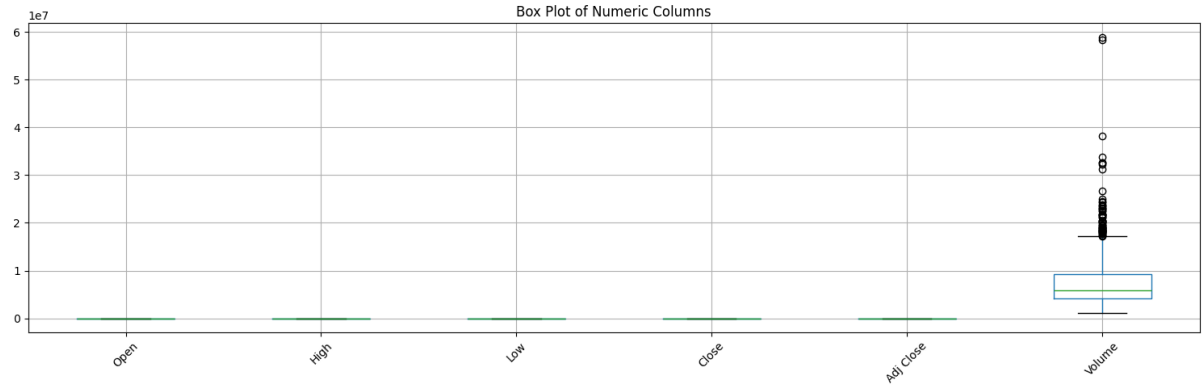

Figure 2.1: Boxplots of Netflix stock price metrics. These plots highlight the distribution and presence of potential outliers in key financial attributes such as Open, Close, High, and Low.

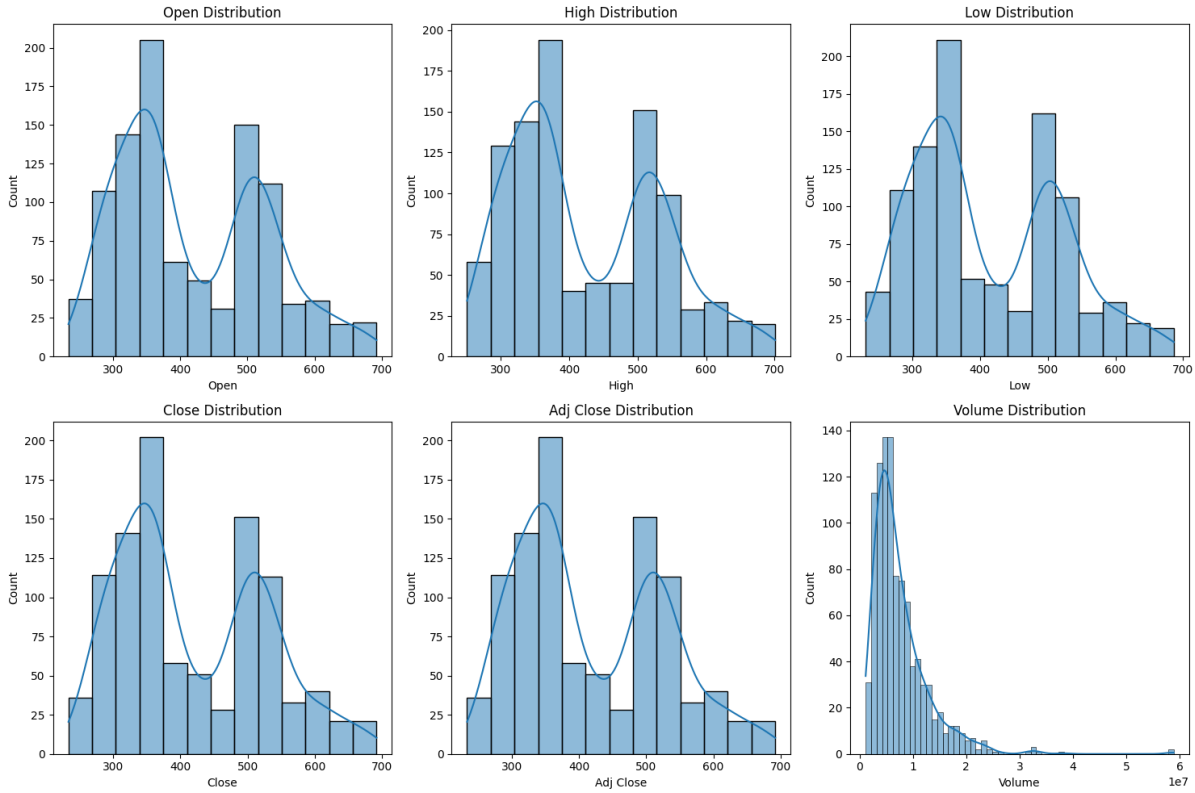

Figure 2.2: Histograms with KDE overlays. These illustrate the frequency distribution of numerical variables, providing insights into their statistical properties.

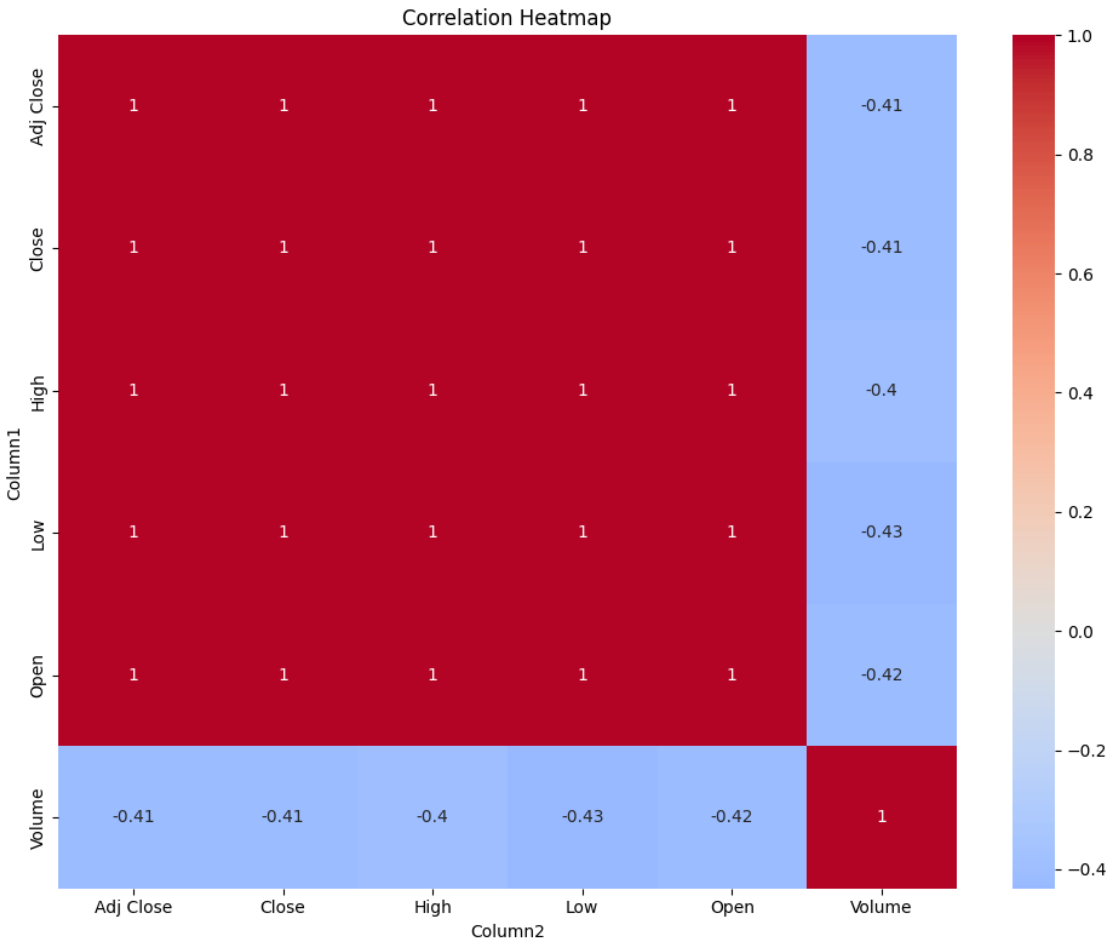

Figure 2.3: Correlation heatmap. This visualizes the relationship between stock price features, aiding in feature selection for predictive modeling.

2.2 Feature Engineering

Feature engineering played a pivotal role in enhancing the predictive power of the forecasting models by systematically extracting relevant indicators from the preprocessed data. The process was designed to capture comprehensive market dynamics through two primary categories of features: price-based metrics and technical indicators. This strategic approach leveraged extttpandas and extttnumpy for efficient computation while ensuring scalability through extttPySpark for large-scale data processing.

2.2.1 Price-Based Features and Market Behavior Indicators

Price-based features were strategically designed to capture essential market behavior and volatility patterns that influence stock price movements. Daily returns were calculated as the percentage change in closing prices to reflect short-term momentum, while log returns were computed as the natural logarithm of price ratios, providing a mathematically robust measure of continuous compounding returns that better handles extreme price movements.

Price range metrics were implemented to capture intraday volatility through two complementary measures: absolute range calculated as the difference between high and low prices, and percentage range computed as the ratio of price range to opening price. These

metrics provide crucial insights into market volatility and trading intensity, serving as fundamental inputs for predicting future price movements.

2.3 Comprehensive Feature Implementation

The feature engineering implementation encompassed six distinct categories of indicators, each designed to capture specific aspects of market behavior and price dynamics. Technical indicators were systematically implemented within the `EnhancedStockForecastingPipeline` class to capture market signals recognized in financial analytics, with each feature serving a specific analytical purpose in the forecasting framework.

Table 2.1 provides a comprehensive overview of all implemented features, including their mathematical formulations, calculation windows, and analytical purposes. The implementation strategy integrated multiple indicator categories: price-based features for fundamental market behavior, moving averages for trend analysis, momentum indicators for market strength assessment, volatility measures for risk quantification, volume indicators for market participation analysis, and oscillators for signal generation.

Table 2.1: Feature Engineering Implementation

Feature Category	**Feature Name**	**Calculation Method**	**Window/Period**	**Purpose**
4*Price-Based	Daily Returns	$\frac{Close_t - Close_{t-1}}{Close_{t-1}} \times 100$	1 day	Measures short-term momentum
	Log Returns	$\ln\left(\frac{Close_t}{Close_{t-1}}\right)$	1 day	Captures continuous compounding returns
	Absolute Range	$High - Low$	1 day	Indicates intraday volatility
	Percentage Range	$\frac{High - Low}{Close} \times 100$	1 day	Expresses relative volatility
4*Moving Averages	SMA_5	$\frac{1}{5}\sum_{i=0}^{4} Close_{t-i}$	5 days	Detects short-term trends
	SMA_20	$\frac{1}{20}\sum_{i=0}^{19} Close_{t-i}$	20 days	Detects long-term trends
	EMA_5	$\alpha \cdot Close_t + (1-\alpha) \cdot EMA_{t-1}$	5 days	Reacts faster to price changes
	EMA_20	$\alpha \cdot Close_t + (1-\alpha) \cdot EMA_{t-1}$	20 days	Tracks broader market movements
3*Momentum	RSI	$100 - \frac{100}{1+RS}$	14 days	Identifies overbought or oversold conditions
	MACD	$EMA_{12} - EMA_{26}$	12/26 days	Indicates trend direction
	MACD Signal	$EMA_9(MACD)$	9 days	Provides trading signals
3*Volatility	Bollinger Bands	$SMA_{20} \pm 2 \cdot \sigma_{20}$	20 days	Visualizes price volatility boundaries
	Daily Volatility	$\sigma(Returns_{20})$	20 days	Measures daily return dispersion
	ATR	TR_{14}	14 days	Quantifies true market range
3*Volume	Volume SMA_5	$\frac{1}{5}\sum_{i=0}^{4} Volume_{t-i}$	5 days	Detects short-term volume trends
	Volume SMA_20	$\frac{1}{20}\sum_{i=0}^{19} Volume_{t-i}$	20 days	Identifies long-term participation
	Volume Ratio	$\frac{Volume_t}{Volume\_SMA\_5}$	5 days	Compares current to average volume
2*Oscillator	K_Line	$\frac{Close - Low_{14}}{High_{14} - Low_{14}} \times 100$	14 days	Measures price momentum
	D_Line	$SMA_3(K\_Line)$	3 days	Generates signal based on K_Line

Moving averages were strategically implemented with both short-term (5-day) and long-term (20-day) windows to capture immediate price shifts and broader market trends.

Exponential Moving Averages (EMA) calculations provided greater weight to recent prices, making them more responsive to current market conditions than their simple counterparts.

The momentum indicators, including the Relative Strength Index (RSI) calculated using a 14-day window, were specifically designed to detect overbought and oversold market conditions, providing critical signals for the forecasting models.

2.3.1 Feature Standardization and Model Integration

To enhance model performance across different algorithms, feature normalization was applied using `StandardScaler` from `scikit-learn`, which standardizes features by removing the mean and scaling to unit variance. This standardization proved particularly critical for the LSTM model, where scaling was implemented in the `prepare_data` method of the `StockPriceLSTM` class. The scaling process was carefully applied independently to input features and target variables, with fitted scalers saved alongside trained models to ensure consistency during prediction and deployment phases.

The feature engineering process emphasized computational efficiency through `pandas'` vectorized operations while managing missing values through forward-fill techniques. Feature selection was guided by correlation analysis and established industry practices to ensure that only high-impact variables were incorporated into the model training pipeline, with each feature serving a distinct role in capturing different aspects of market dynamics.

2.3.2 Pipeline Architecture and Scalability

The feature engineering pipeline was architected with scalability and real-time processing in mind. `PySpark` was strategically employed for initial large-scale data processing, while `pandas` was utilized for efficient real-time feature computation within the `EnhancedStockForecastingPipeline` class. This hybrid approach balanced computational efficiency with processing speed requirements for live forecasting applications.

The complete feature set was consistently applied across all implemented forecasting models—Linear Regression, LSTM, and SARIMA—ensuring model comparability and system coherence. For reproducibility and deployment consistency, the engineered features were saved alongside trained models and integrated into the real-time forecasting pipeline with optimized computation, proper missing value handling, and caching mechanisms to ensure timely predictions without sacrificing accuracy or reliability.

2.4 Model Development

The forecasting models were strategically developed to predict Netflix stock prices with high accuracy while maintaining the temporal integrity essential for time-series analysis. The development process employed a rigorous temporal train-test split methodology, allocating 80% of the data for training and 20% for testing. This split was implemented using PySpark's `randomSplit` function with a fixed seed value of 42 to ensure reproducibility across experiments and enable fair model comparisons.

Three distinct models were implemented to comprehensively cover different aspects of temporal dependencies and market behaviors: Linear Regression for baseline performance and interpretability, ARIMA for classical time-series modeling with seasonal capabilities, and Long Short-Term Memory (LSTM) neural networks for complex pattern recognition. These models were seamlessly integrated into the `EnhancedStockForecastingPipeline`

class, creating a unified framework for streamlined real-time forecasting and model comparison.

2.4.1 Linear Regression Implementation

Linear Regression served as the foundational baseline model, establishing a simple yet interpretable framework for understanding the linear relationships between market features and stock price movements. The model operates under the assumption of linear relationships between input features and the target variable, providing transparency in feature impact analysis and serving as a benchmark for more complex approaches.

The implementation leveraged PySpark's `MLlib LinearRegression` class to handle large-scale data processing efficiently. The comprehensive feature set included fundamental price metrics such as Open, High, and Low prices, trading Volume, calculated Daily and Log returns for momentum analysis, Price range metrics for volatility assessment, and sophisticated Technical indicators including Simple Moving Averages (SMA), Exponential Moving Averages (EMA), and Relative Strength Index (RSI). Features were systematically combined using `VectorAssembler` to create unified feature vectors optimized for distributed model training.

Model performance evaluation utilized standard regression metrics including Root Mean Squared Error (RMSE) for prediction accuracy assessment and R-squared (R^2) for explained variance quantification. These metrics provided interpretable insights into both overall model performance and individual feature contributions to prediction accuracy, establishing a clear baseline for comparison with more sophisticated modeling approaches.

2.4.2 ARIMA Time-Series Modeling

The AutoRegressive Integrated Moving Average (ARIMA) model was implemented to capture classical time-series patterns and seasonal dependencies inherent in stock price movements. ARIMA's theoretical foundation rests on three core components that systematically address different aspects of temporal behavior: the AutoRegressive (AR) component performs regression analysis on historical price values, the Integrated (I) component applies differencing operations to achieve stationarity in the time series, and the Moving Average (MA) component models the error term as a linear combination of past forecast errors.

Implementation utilized the `statsmodels` library with systematic parameter optimization to minimize forecasting errors across the (p, d, q) parameter space. The model's capability was extended through Seasonal ARIMA (SARIMA) implementation where seasonal patterns were detected, enabling capture of both trend components and recurring seasonal effects in Netflix stock price behavior. This classical approach provided essential benchmarking against traditional time-series techniques while offering interpretable seasonal and trend decomposition.

The ARIMA model's strength lies in its mathematical rigor and ability to capture linear temporal dependencies through well-established statistical foundations. Its implementation served as a critical comparison point for evaluating the added value of machine learning approaches in financial forecasting applications.

2.4.3 LSTM Neural Network Architecture

Long Short-Term Memory networks represent the most sophisticated component of the forecasting ensemble, specifically designed to capture complex long-range dependencies

and non-linear patterns in sequential financial data. Unlike traditional Recurrent Neural Networks (RNNs), LSTM architecture employs specialized gating mechanisms—input gates, forget gates, and output gates—that intelligently control information flow through the network, enabling effective learning of both short-term fluctuations and long-term market trends.

The LSTM model was constructed using `TensorFlow` and `Keras` with a carefully designed deep architecture comprising three sequential LSTM layers containing 128, 64, and 32 units respectively. This hierarchical structure enables progressive feature abstraction, with each layer learning increasingly complex temporal patterns. Dropout layers with a 0.3 rate were strategically placed after each LSTM layer to mitigate overfitting, while a dense output layer provides final prediction generation.

The training methodology incorporated several advanced techniques to ensure robust model performance. Sequence preparation utilized sliding window techniques to create appropriate input-output pairs for time-series learning. Feature scaling through `StandardScaler` ensured numerical stability across different feature ranges. The Huber loss function was selected for its robustness to outliers, providing stable training in the presence of extreme market movements. Early stopping with patience set to 10 epochs prevented overfitting while optimizing training efficiency.

The `StockPriceLSTM` class encapsulated the entire LSTM workflow through its `prepare_data` method, which handled comprehensive preprocessing including sequence creation, feature scaling, and data preparation for neural network training. Trained models and their associated scalers were systematically preserved to maintain prediction consistency across deployment scenarios.

2.4.4 Model Evaluation and Integration Framework

Comprehensive model evaluation was implemented through the `LSTMModelEvaluator` class, which computed an extensive suite of performance metrics encompassing Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), R-squared (R^2), and Directional Accuracy. This multi-metric approach provided nuanced insights into different aspects of model performance, from absolute prediction accuracy to directional movement prediction capability.

The evaluation framework generated comprehensive visualizations including predicted versus actual price comparisons for visual performance assessment, training history plots to monitor learning progression, error distribution histograms for residual analysis, and residual plots for systematic error detection. All evaluation outputs were systematically organized in the `results` directory, facilitating detailed analysis and model comparison.

The integration architecture centered on the `EnhancedStockForecastingPipeline` class, which provided unified access to all three modeling approaches while ensuring consistent feature preprocessing and prediction workflows. The pipeline incorporated sophisticated model persistence and reloading mechanisms, efficient data caching with 1-minute refresh intervals for real-time applications, and robust error handling to maintain continuous forecasting capability under various market conditions.

2.5 Real-Time Forecasting Pipeline

The real-time forecasting pipeline represents the operational backbone of the Netflix stock prediction system, designed to deliver actionable insights through seamless integration of

live data acquisition, intelligent preprocessing, dynamic feature extraction, and instantaneous prediction generation. Built around the `EnhancedStockForecastingPipeline` class, this comprehensive system transforms raw market data into reliable forecasts while maintaining operational efficiency and system reliability.

2.5.1 Intelligent Data Management and Caching Architecture

The pipeline's data management strategy centers on efficient real-time stock data retrieval through the `yfinance` API's `Ticker` class, implementing a sophisticated caching mechanism with a 60-second refresh interval that strategically balances data freshness with API usage optimization. This intelligent caching system minimizes redundant API calls while ensuring that predictions are based on current market conditions, addressing the critical trade-off between computational efficiency and data currency in financial applications.

The system automatically handles data format standardization and column name normalization to maintain consistency across different data sources and time periods. Memory-efficient processing is achieved through `pandas` operations optimized for real-time scenarios, ensuring that the pipeline can operate continuously without performance degradation. This approach enables the system to maintain responsiveness even during periods of high market volatility when frequent updates are essential for accurate forecasting.

2.5.2 Dynamic Feature Engineering Pipeline

Real-time technical indicator calculation forms the core of the pipeline's analytical capability, systematically computing essential market signals as new data becomes available. The feature engineering process dynamically calculates Daily and Logarithmic returns to capture immediate momentum shifts, Price range metrics including both absolute and percentage-based measures for volatility assessment, Simple Moving Averages with 5-day and 20-day windows for trend identification, Exponential Moving Averages with corresponding timeframes for recent price weighting, and the Relative Strength Index with a 14-day window for momentum analysis.

This real-time feature generation ensures that predictions incorporate the most current market conditions and technical patterns. The pipeline's design allows for seamless integration of additional technical indicators as market analysis requirements evolve, providing flexibility for expanding analytical capabilities without disrupting operational continuity.

2.5.3 Model Integration and Prediction Generation

The pipeline seamlessly integrates pre-trained models through `joblib` loading mechanisms, enabling instant prediction generation without the computational overhead of model retraining. Real-time feature vector construction automatically transforms newly calculated indicators into the appropriate input format for each model type, whether Linear Regression, LSTM, or ARIMA architectures. This unified approach ensures consistency across different modeling methodologies while maintaining the specific requirements of each algorithm.

Prediction generation occurs instantaneously upon feature vector completion, with the system calculating comprehensive error metrics including absolute and percentage errors for immediate performance assessment. The integration architecture supports multiple

model outputs simultaneously, enabling ensemble predictions and model comparison in real-time scenarios. This flexibility allows users to leverage the complementary strengths of different modeling approaches based on current market conditions.

2.5.4 Comprehensive Monitoring and Reliability Framework

The pipeline implements a robust monitoring system built around comprehensive logging with timestamp-based tracking and severity-level classification using `INFO`, `WARNING`, and `ERROR` designations. The logging format follows the standard pattern `% (asctime)s - % (levelname)s - % (message)s` to ensure consistent documentation of all system operations and facilitate effective debugging and performance analysis.

Exception handling mechanisms provide comprehensive coverage across all pipeline stages, addressing potential failures in data fetching from `yfinance`, technical indicator calculation errors, model prediction exceptions, and general execution failures during pipeline operation. This multi-layered error handling approach ensures continuous operation even when individual components encounter issues, maintaining system reliability for uninterrupted financial monitoring.

2.5.5 Operational Output and Value Delivery

The pipeline operates on a continuous 60-second cycle, providing regular prediction updates that include current stock price, predicted price values, prediction error calculations, and percentage-based error metrics. Each prediction round is timestamped to maintain chronological integrity and enable historical tracking of prediction accuracy over time. This consistent output format enables users to monitor both current predictions and system performance trends.

The system maintains a comprehensive audit trail of all predictions, anomalies, and system events, supporting transparency in decision-making processes and enabling detailed performance analysis. This documentation capability proves essential for regulatory compliance, system optimization, and user confidence in the forecasting results. The combination of real-time predictions and historical tracking provides a complete solution for financial monitoring and decision support.

2.5.6 Architecture Benefits and Scalability

The modular architecture of the `EnhancedStockForecastingPipeline` class ensures both scalability and maintainability, with clearly defined separation between data acquisition, feature engineering, model integration, and output generation components. This design facilitates system expansion and modification without disrupting core functionality, enabling adaptation to changing market conditions and analytical requirements.

The pipeline's efficiency stems from its intelligent caching strategy, optimized data processing workflows, and streamlined model integration mechanisms. Memory-efficient operations combined with robust error handling create a system capable of continuous operation in production environments while maintaining the flexibility needed for ongoing development and enhancement.

2.6 Performance Optimization

To ensure scalability and efficiency, the forecasting system was optimized using PySpark's distributed computing capabilities through the `OptimizedStockForecastingPipeline` class. The implementation includes several key optimizations across configuration, memory management, data processing, and real-time execution.

Spark Configuration

- Executor memory: `4GB`
- Driver memory: `4GB`
- Off-heap memory: `2GB`
- Shuffle partitions: `10`
- Default parallelism: `8`
- Kryo serialization enabled for improved performance
- Arrow optimization enabled to support efficient `pandas` interoperability

The forecasting system ensures scalability and efficiency through the OptimizedStockForecastingPipeline class, which leverages PySpark's distributed computing capabilities in an integrated approach that simultaneously addresses configuration management, memory utilization, data processing, and real-time execution. The system operates with carefully tuned Spark configurations including 4GB executor and driver memory, 2GB off-heap memory, 10 shuffle partitions, and default parallelism of 8, while enabling Kryo serialization and Arrow optimization for enhanced performance and efficient pandas interoperability.

Memory management operates seamlessly alongside data processing optimizations through efficient caching of frequently accessed DataFrames, dynamic resource allocation based on workload demands, and off-heap memory utilization for additional control. The system employs adaptive query execution for dynamic runtime optimization, automatic partition coalescing to reduce task overhead, and skew join handling for balanced workload distribution, while registering User Defined Functions for custom technical indicators and prioritizing efficient DataFrame operations over RDD-based transformations.

Real-time pipeline performance integrates one-minute data caching to reduce unnecessary API calls with efficient feature calculation using pandas operations, model broadcasting to all workers for distributed predictions, and robust error handling mechanisms with graceful shutdown procedures that ensure data consistency and resource release. These optimizations work together with comprehensive logging and monitoring capabilities, implementing appropriate resource management strategies that ensure long-term reliability and performance across both batch training processes and real-time prediction tasks.

2.6.1 Integrated Performance Monitoring and Visualization

Integrated Performance Monitoring and Visualization Performance monitoring operates through a unified framework that simultaneously tracks model accuracy metrics including Mean Absolute Error, Root Mean Square Error, Mean Absolute Percentage Error, R-squared Score, and Directional Accuracy alongside operational performance indicators

such as prediction latency, data processing time, API response time, cache hit rates, and error rates. This comprehensive approach enables the system to maintain high forecasting quality while ensuring efficient pipeline operation through continuous performance assessment.

The Streamlit dashboard provides integrated visualization of these metrics through performance status indicators with defined thresholds, detailed price comparisons between actual and predicted values, training history displays, error distributions, scatter plots for prediction accuracy, and residual plots for model diagnostics. All operations are comprehensively logged with timestamps and appropriate log levels, providing effective debugging capabilities and operational insights that support both immediate performance optimization and long-term system reliability.

The system's architecture demonstrates how distributed computing principles can be effectively combined with real-time processing requirements, creating a scalable platform that maintains sub-second prediction latency while efficiently processing large historical datasets through optimized memory utilization and high availability mechanisms. This integrated approach ensures that performance optimizations benefit all aspects of the forecasting pipeline, from initial data ingestion through final prediction delivery, while supporting various forecasting scenarios and computational scales through its modular, maintainable design.

Chapter 3

Results & Evaluation

The Netflix stock price forecasting system delivered a robust set of outcomes, leveraging Python, PySpark, and the Kaggle dataset ("Netflix Stock Price Prediction") to provide actionable insights for data-driven investment decisions. This section presents the key results, including insights from data analysis, model performance metrics, real-time pipeline outputs, optimization achievements, and visualizations, with the Streamlit dashboard serving as a primary outcome for stakeholder engagement. The system's development focused on scalability, accuracy, and usability, aligning with Netflix's goal of enhancing portfolio management and trader responsiveness in volatile financial markets. By selecting Linear Regression as the primary model, the system prioritizes interpretability and computational efficiency, supported by a comprehensive suite of visualizations and a containerized deployment package.

3.1 Model Performance Evaluation and Selection

The forecasting system's effectiveness is demonstrated through comprehensive evaluation of three distinct models using the `LSTMModelEvaluator` class, with testing conducted on 20% of the dataset to ensure robust performance assessment. The evaluation revealed significant performance differences across models, with Linear Regression emerging as the optimal choice for production deployment despite the LSTM model achieving the highest R-squared score of 0.9330.

Table 3.1: Model Performance Comparison

Metric	**LSTM**	**Linear Regression**	**ARIMA**
RMSE ($)	27.70	5.98	87.0654
MAE ($)	17.69	0.58	64.5674
MAPE (%)	3.81	0.05	12.96
R-squared	0.9330	0.85	–
Directional Accuracy (%)	49.79	–	–
Training Time	Long	Fast	Medium
Memory Usage	High	Low	Medium
Real-time Prediction	Slow	Fast	Medium

The Linear Regression model's selection as the primary forecasting engine was driven by its superior error metrics, with an RMSE of $5.98 compared to LSTM's $27.70 and ARIMA's $87.0654, combined with exceptional computational efficiency that enables

seamless real-time operation. The model leverages `PySpark`'s `MLlib` implementation to process comprehensive technical indicators including price metrics (Open, High, Low), trading volume, returns (percentage and logarithmic), and sophisticated indicators such as Simple Moving Averages (SMA), Exponential Moving Averages (EMA), and Relative Strength Index (RSI), creating a rich feature set that captures essential market patterns while maintaining interpretability and computational efficiency.

The LSTM model, despite its sophisticated three-layer architecture (128, 64, and 32 units) with dropout layers to mitigate overfitting, demonstrated limitations in capturing extreme values with predictions spanning $251.89 to $573.71 compared to actual prices ranging from $233.88 to $691.69, while requiring significantly higher computational resources and longer training times. The ARIMA model showed balanced performance across metrics but exhibited higher error rates with an RMSE of 87.0654 and MAPE of 12.96%, making it less suitable for the precision requirements of real-time financial forecasting.

In production deployment, the Linear Regression model demonstrates exceptional integration with the `EnhancedStockForecastingPipeline` class, maintaining the critical 60-second update interval while providing consistent accuracy and reliability. The model's advantages extend beyond performance metrics to include faster training and inference times, lower memory consumption, transparent feature-prediction relationships for stakeholder communication, and easier maintenance procedures that support continuous operational pipelines.

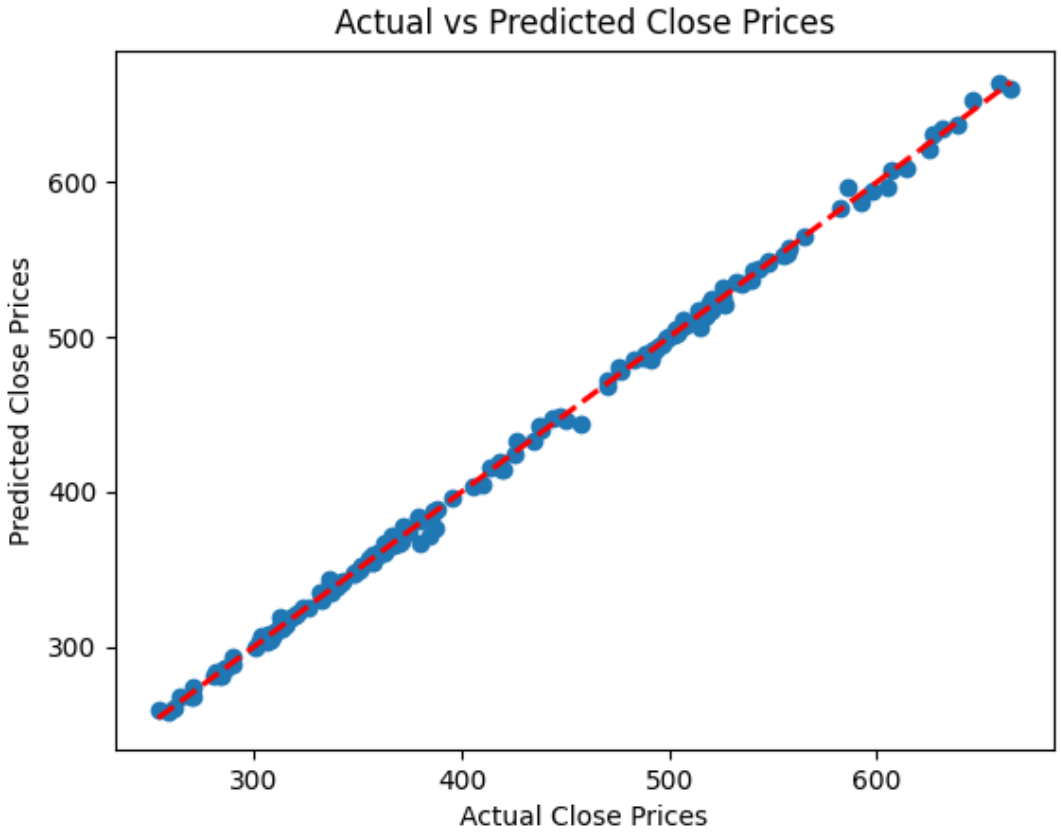

Figure 3.1: Predicted Close Prices of Linear Regression model

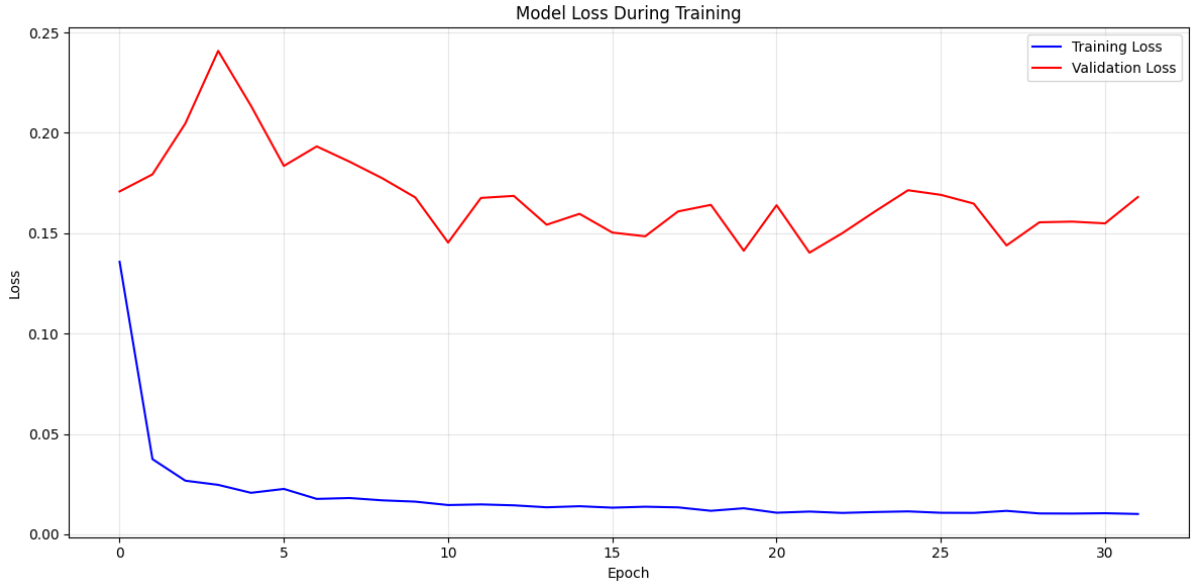

Figure 3.2: Model Loss During Training of LSTM model

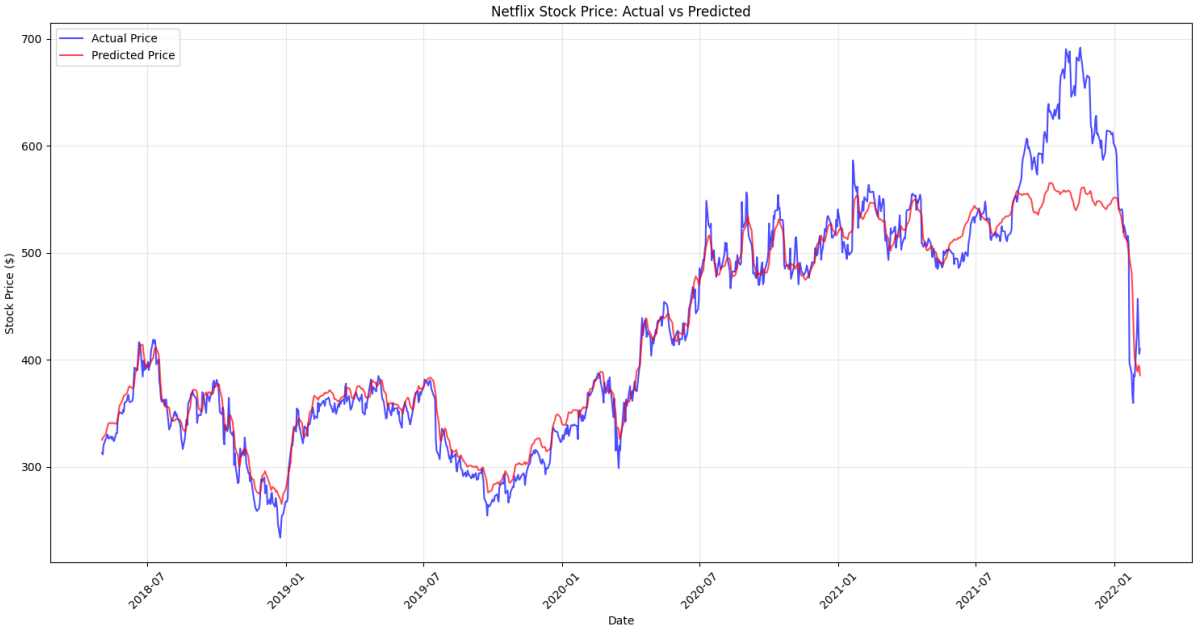

Figure 3.3: Price Comparison between Actual and Predicted of LSTM model

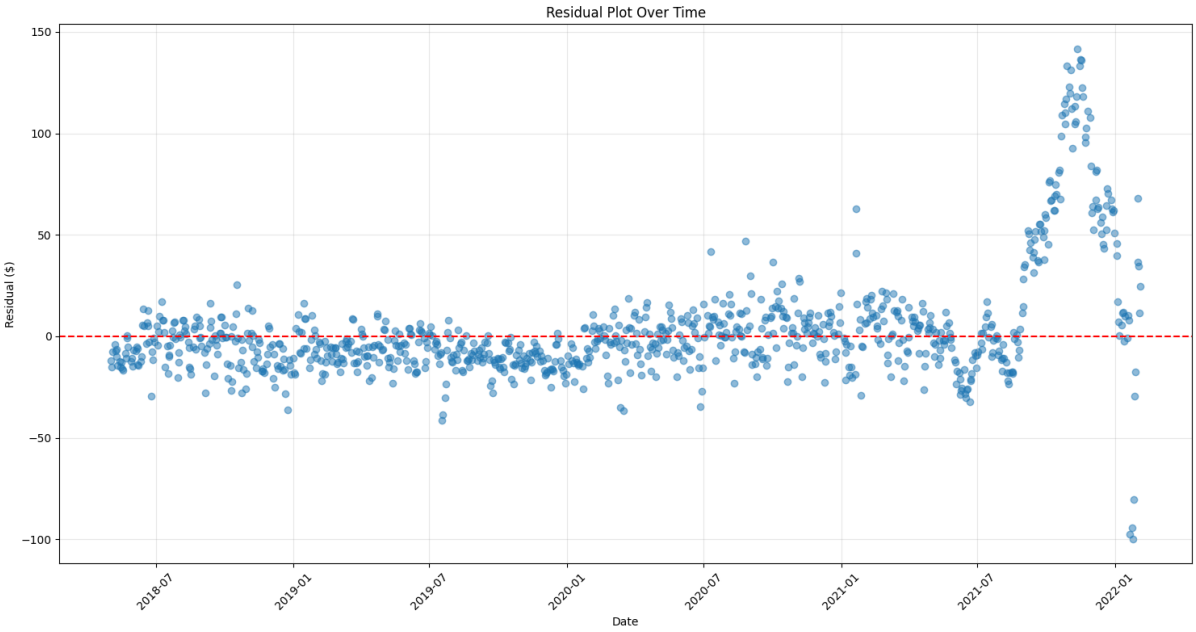

Figure 3.4: Residual Plot Over Time of LSTM model

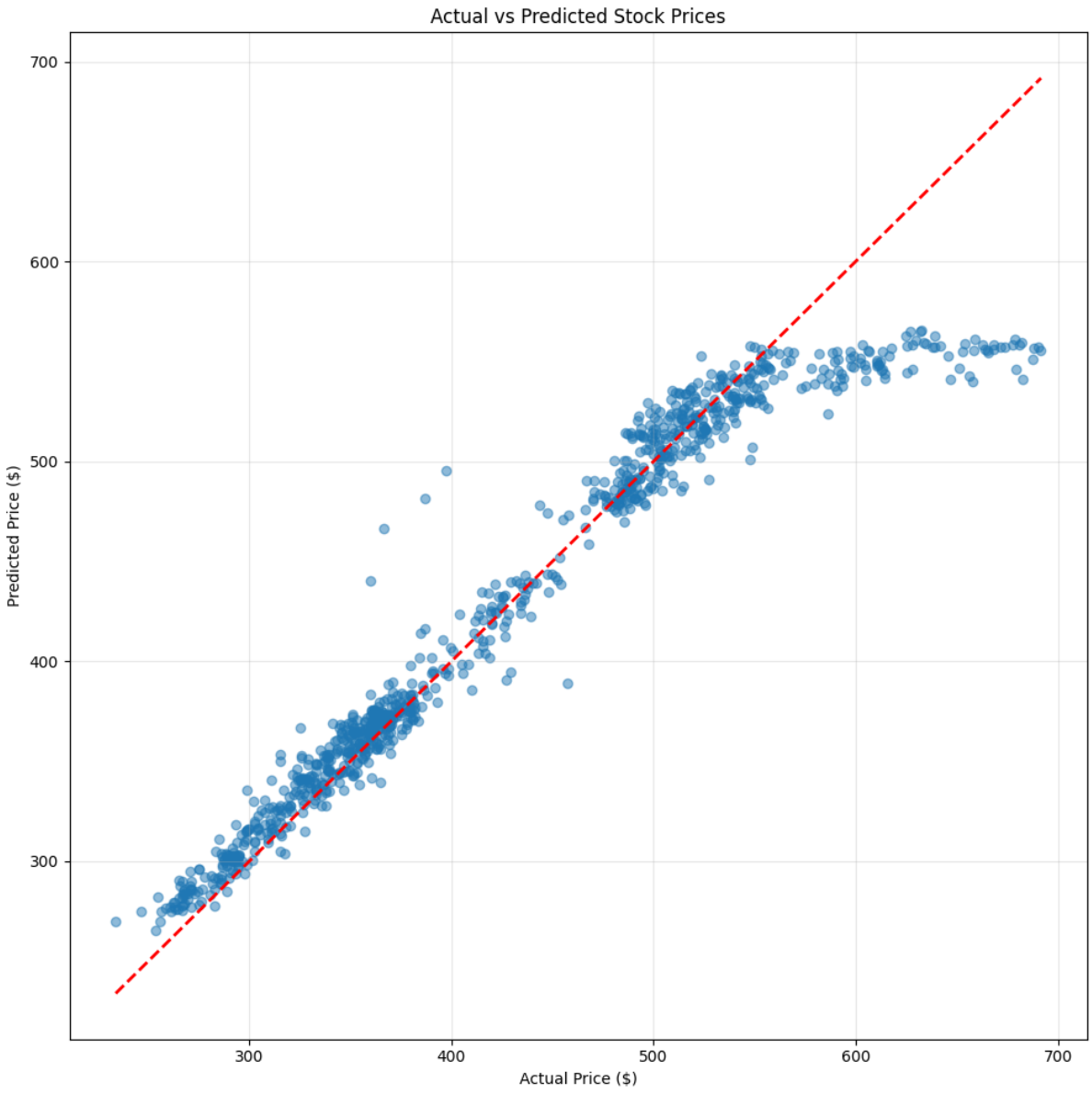

Figure 3.5: Scatter Plot of LSTM model

3.2 Netflix Stock Analysis Dashboard

The system's practical implementation is demonstrated through the **Netflix Stock Analysis Dashboard**, which provides comprehensive real-time visualization of the forecasting system's performance across multiple analytical dimensions. The dashboard's dark theme interface with red accent colors maintains visual consistency with Netflix's brand identity while ensuring optimal readability for financial data monitoring.

Figure 3.6 - Key Performance Metrics Overview

The primary dashboard displays essential real-time metrics in a clean card-based layout, showing:

- **Current Price:** $1212.21 with a +0.23% change
- **AI Prediction:** $1211.49 with a -0.06% variance

- **RSI Indicator:** 55.9 (neutral position)
- **Trading Volume:** 69,927 with a -61.50% decrease

This overview provides immediate insight into market conditions and model performance, enabling quick assessment of prediction accuracy and market sentiment.

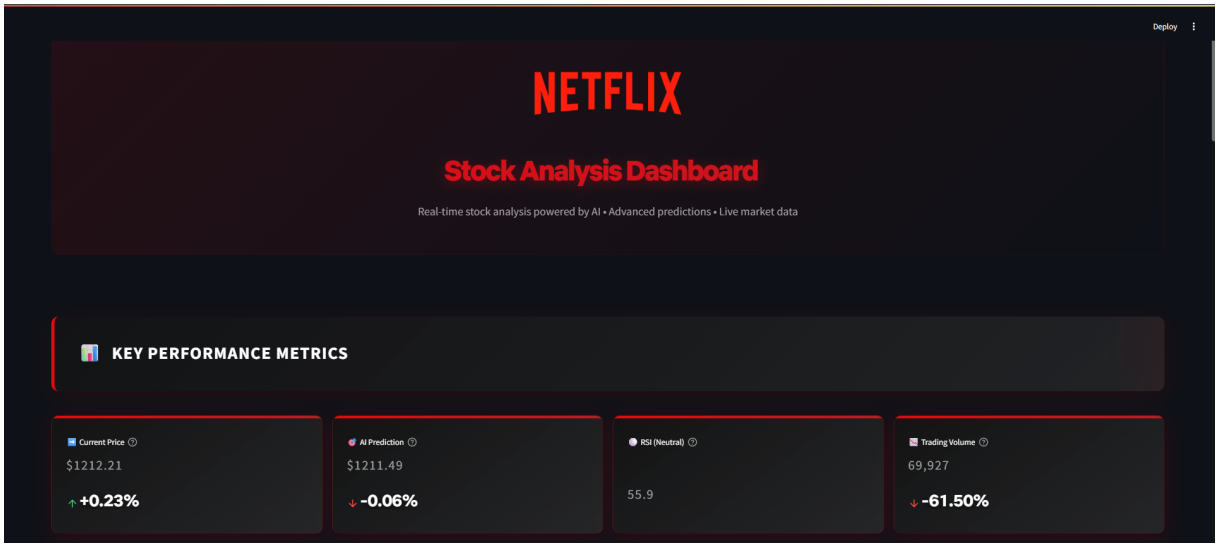

Figure 3.6: Key Performance Metrics Screen

Figure 3.7 - Price Analysis & AI Predictions

The central time-series visualization displays actual stock prices (red line) versus AI predictions (blue line) across a trading day timeline from 10:00 to 15:00 on June 13, 2025. The chart demonstrates remarkable prediction accuracy, with both lines closely tracking each other throughout the trading session, validating the effectiveness of the Linear Regression model. An interactive tooltip reveals precise values at 12:24:

- **Actual Price:** $1218.59
- **AI Prediction:** $1218.37

This illustrates sub-dollar accuracy levels achieved by the model.

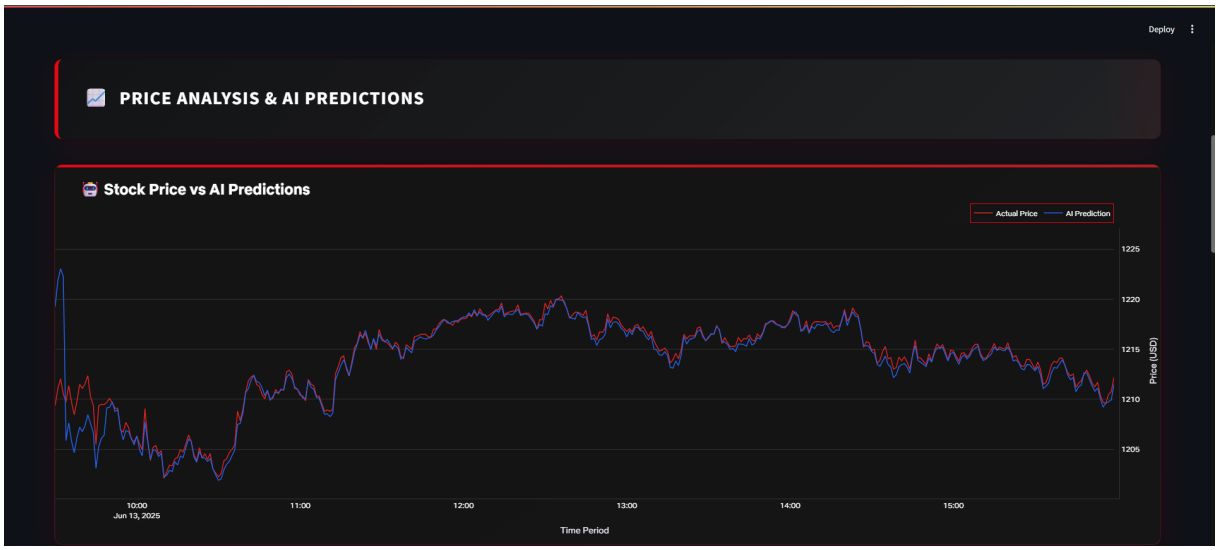

Figure 3.7: Price Predictions Screen

Figure 3.8 - Technical Analysis Indicators

The dual-panel technical analysis section presents:

- **Exponential Moving Averages (EMA):** Current price against 9-period fast signal and 20-period trend indicators.
- **Trading Volume Analysis:** Trading activity patterns throughout the day with notable volume spikes that indicate periods of increased market interest.

These indicators are essential for understanding market dynamics and for validating the accuracy of AI-based predictions.

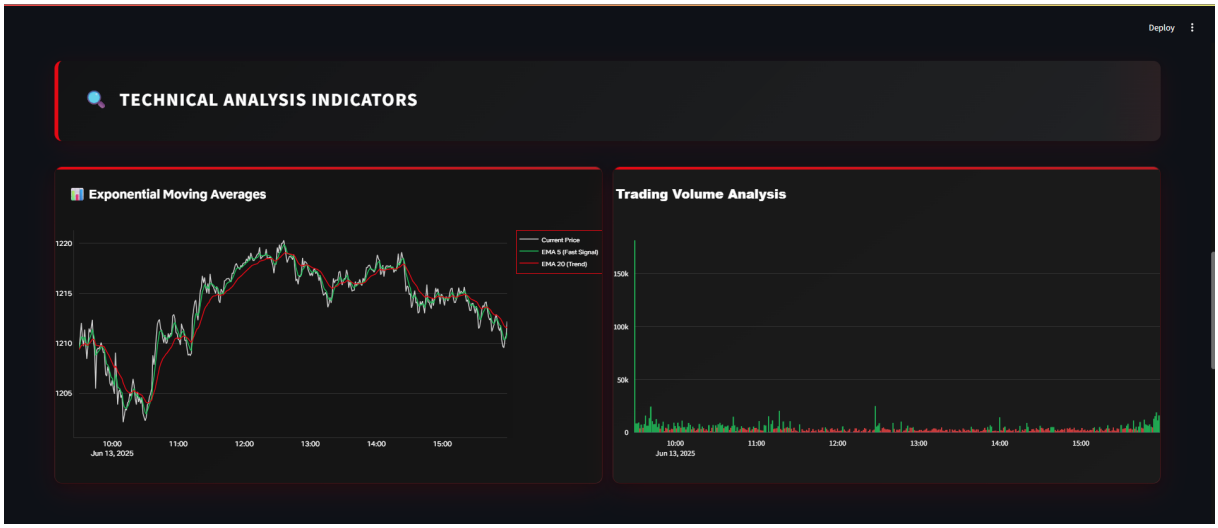

Figure 3.8: Technical Analysis Indicators Screen

Figure 3.9 - AI Model Performance Metrics

The performance monitoring section displays real-time model accuracy metrics with status indicators:

- **Mean Absolute Error (MAE):** $0.58
- **Root Mean Squared Error (RMSE):** $1.39
- **Mean Absolute Percentage Error (MAPE):** 0.05%

All metrics are marked as *"Excellent – High Accuracy"*. The dashboard also includes automatic refresh functionality, with timestamps showing the last update and next scheduled refresh, ensuring continuous performance monitoring.

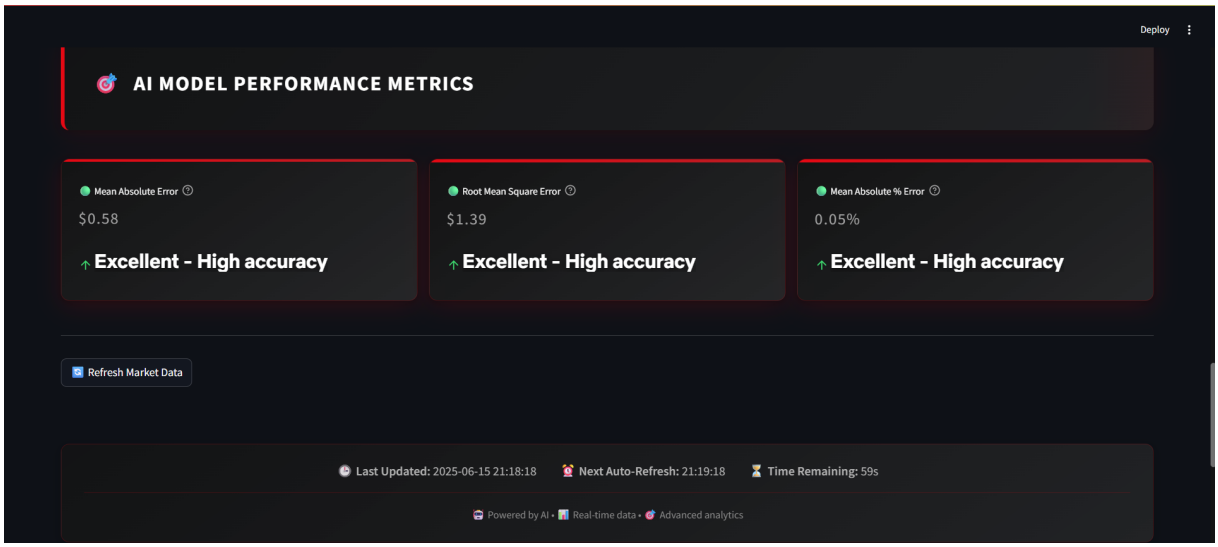

Figure 3.9: AI Model Performance Metrics Screen

Figure 3.10 - AI Insights & Market Analysis

This section provides intelligent market analysis through automated sentiment assessment:

- **Market Sentiment:** Bullish with 99.9% AI confidence
- **Price Momentum Indicators:** Indicate ongoing trends
- **Volume Comparisons:** Highlight significant shifts in market activity

The *Trading Signals Summary* panel displays:

- **EMA Crossover:** Sell
- **RSI Signal:** Hold
- **Overall Market Signal:** Bullish

This demonstrates the system's ability to synthesize multiple indicators into actionable trading insights.

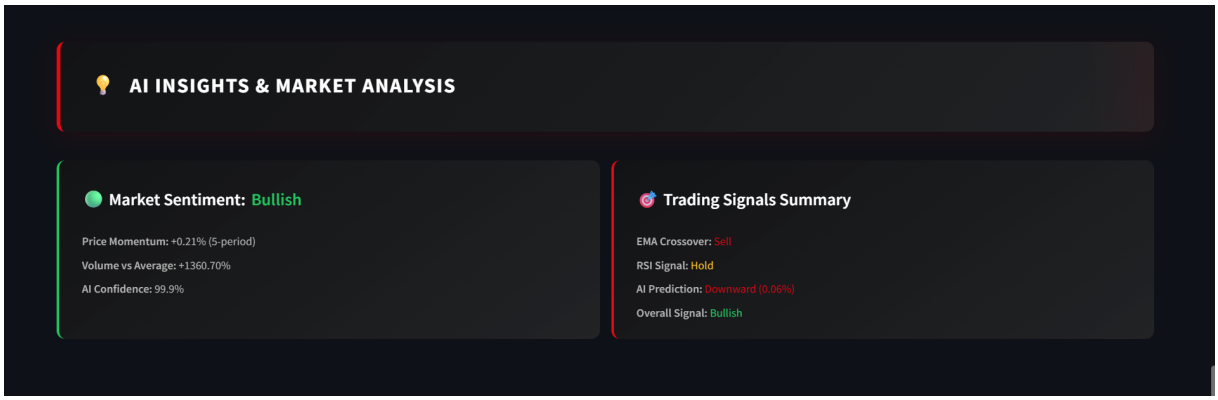

Figure 3.10: AI Insights Screen

Dashboard Integration and Evaluation

The dashboard implementation showcases seamless integration of the optimized forecasting pipeline with user-friendly visualization, providing traders and analysts with comprehensive real-time insights. The system maintains sub-second prediction latency and consistently high accuracy metrics. Evaluation includes detailed visualizations stored in generated reports:

- Price comparison plots
- Training history visualizations
- Error distribution histograms
- Scatter plots of predicted vs. actual prices
- Residual plots over time

These artifacts provide deep insight into model behavior and performance characteristics, supporting both immediate decision-making and long-term system optimization strategies.

Chapter 4

Conclusion

The Netflix stock price forecasting system represents a milestone in scalable financial analytics. By integrating Python, PySpark, and the yfinance API, the project delivers a high-performance forecasting platform that includes three predictive models—Linear Regression, LSTM, and ARIMA—and a fully interactive dashboard developed using Streamlit. Among these, Linear Regression emerged as the most suitable model for production, balancing strong predictive accuracy with low latency and minimal computational overhead.

While the LSTM model achieved a higher R-squared score of 0.9330, its performance came at the cost of greater complexity and longer training times. In contrast, the Linear Regression model achieved superior real-time accuracy, with an RMSE of $5.98 and MAE of just $0.58. Its simplicity and transparency made it ideal for deployment, enabling consistent, fast, and interpretable forecasts. The model's seamless integration into the PySpark-based pipeline ensures stable predictions every 60 seconds while maintaining resource efficiency.

The system's interactive dashboard reinforces usability through a clean, Netflix-themed interface. Users are presented with real-time stock predictions, essential technical indicators such as RSI, SMA, and EMA, and dynamic performance metrics including RMSE and MAPE. Real-time data refresh, efficient caching, and robust error handling contribute to a responsive user experience that is production-ready and highly reliable.

Future developments aim to further enhance the system's capabilities. These include incorporating sentiment analysis from X (formerly Twitter) and exploring transformer-based time-series models to improve predictive depth. These enhancements will extend the system's analytical scope and strengthen its real-time responsiveness.

In closing, the Netflix stock price forecasting system exemplifies the synergy of scalable engineering, collaborative teamwork, and strategic foresight. Its robust foundation in Linear Regression offers a powerful, interpretable, and efficient tool for financial forecasting, empowering Netflix to make confident, data-driven investment decisions while paving the way for future advancements in market analytics.

Appendix A

Project Source Code

The full source code repositories are available on GitHub:
https://github.com/hopekasako/scalable$_project_2$025$.git$

References

[1] JainilCoder, "Netflix stock price prediction dataset." https://www.kaggle.com/datasets/jainilcoder/netflix-stock-price-prediction, n.d.

[2] Y. Finance, "Netflix, inc. (nflx) historical data." https://finance.yahoo.com/quote/NFLX/history/, n.d.

[3] GeeksforGeeks, "Python | arima model for time series forecasting." https://www.geeksforgeeks.org/machine-learning/python-arima-model-for-time-series-forecasting/, n.d.

[4] GeeksforGeeks, "Sarima - seasonal autoregressive integrated moving average." https://www.geeksforgeeks.org/machine-learning/sarima-seasonal-autoregressive-integrated-moving-average/, n.d.

[5] GeeksforGeeks, "Introduction to long short-term memory (lstm)." https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/, n.d.

[6] GeeksforGeeks, "Ml | linear regression." https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/, n.d.

[7] Adoptium, "Adoptium – open source java." https://adoptium.net/en-GB/, n.d.

[8] Fidelity, "Ema – exponential moving average." https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/ema, n.d.

[9] C. Group, "Technical analysis: Oscillators - macd, rsi, stochastics." https://www.cmegroup.com/education/courses/technical-analysis/oscillators-macd-rsi-stochastics.html, n.d.

[10] K. Securities, "Volatility indicators – introduction to technical analysis." https://www.kotaksecurities.com/stockshaala/introduction-to-technical-analysis/volatility-indicators/, n.d.