BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

DEPARTMENT OF MANAGEMENT

Big Data Analytics Project Report on

"Nifty50 Stock Prediction using LSTM (Long-Short Term Memory model)"

Faculty Incharge: **Prof. Udayan Chanda**

Submitted to: **Prof. Ashutosh Vyas**

Submitted by:

Sagar Jain

Table of Contents

Abstract 3

Executive Summary 3

Introduction 3

Key Features: - 4

Why LSTM (Long Short-Term Memory) 6

Objective 7

Methodology 8

Architecture 11

Result 14

Strengths 16

Areas of Improvement 16

Conclusion 17

Reference 17

Stock Market Prediction Using LSTM

Abstract

Investing in a set of assets has always been a complex and challenging task, and the inherent volatility and unpredictability of financial markets only make it more difficult. The abnormality and ever-changing nature of financial markets prevent the use of simple models from accurately predicting future asset values, often resulting in significant deviations from actual outcomes. With this in mind, one of the most exciting and promising advancements in the field of scientific research is machine learning. Machine learning, which involves teaching computers to perform tasks that would typically require human intelligence, is rapidly becoming the dominant trend in various industries, including finance. This article seeks to develop a predictive model using advanced machine learning techniques, specifically Recurrent Neural Networks (RNN), with a particular focus on Long Short-Term Memory (LSTM) networks, to forecast future stock market values. The primary objective of this study is to evaluate the precision with which machine learning algorithms, particularly LSTM models, can predict stock prices, and to assess how the number of epochs can influence the performance and accuracy of the model. Through this exploration, we aim to determine how machine learning can provide more accurate and reliable predictions compared to traditional methods, thereby potentially transforming the way investments are made in financial markets.

Executive Summary

The provided Python script builds a Stock Price Prediction Application using machine learning and deep learning techniques, accessible via a Streamlit-based user interface. It integrates PySpark for scalable data handling and employs LSTM (Long Short-Term Memory) neural networks for predicting future stock prices. Historical data is fetched using Yahoo Finance API, and users can input parameters like stock tickers, historical data period, LSTM sequence length, and training epochs. Using the given parameters, the output will be visualization of the predicted stock prices.

Introduction

This Python script is a comprehensive application that predicts stock prices using advanced machine learning and deep learning techniques. It integrates a user-friendly interface through Streamlit, allowing users to input stock parameters and visualize predictions accordingly. The application fetches real-time historical stock data from Yahoo Finance, preprocesses it, and employs LSTM (Long Short-Term Memory) models for forecasting future stock prices.

The script is tailored for financial analysts, researchers, and enthusiasts who want to explore stock price movements, assess stock-to-benchmark relationships, and predict future trends based on historical data. It also incorporates scalable big data processing capabilities using PySpark, making it suitable for handling large datasets efficiently.

Key features of this script include:

- Linear regression to analyze stock and benchmark correlations.
- LSTM neural networks for time-series predictions.
- Visualizing historical and predicted stock prices for informed decision-making.

The tool provides an end-to-end solution for stock analysis, from data acquisition to prediction and visualization, enabling better investment insights.

Key Features:-

1. **Interactive User Interface:** The application features an interactive user interface that allows users to easily provide stock tickers and define prediction parameters, such as the historical data period or the forecast horizon. Through the interface, users can input the required information with minimal effort, making the tool accessible even for those without a deep technical background. The app also enables users to visualize the results directly within the interface, displaying historical and predicted stock prices in dynamic plots. These visualizations, powered by Streamlit, provide users with real-time insights into stock trends, enabling them to make more informed decisions based on the model's predictions. This interactive setup enhances user engagement and ensures that the results are presented in a clear and understandable format, supporting both novice and experienced investors alike.

2. **Comprehensive Data Workflow**: The application follows a well-structured, end-to-end data workflow that ensures the accuracy and reliability of stock price predictions

    - **Data Fetching**: Initially, the application fetches historical stock and benchmark prices using the Yahoo Finance API. This reliable data source provides the necessary time series data for both the stock and the benchmark, covering various time periods such as 1 year, 5 years, or 10 years, based on user inputs.

    - **Data Preprocessing and Feature Engineering**: Once the data is acquired, it undergoes preprocessing to ensure that it is clean and ready for analysis. This step includes handling missing data, merging stock and benchmark datasets on common dates for temporal alignment, and normalizing the data using MinMaxScaler for consistency. Additionally, key financial relationships, such as the Beta coefficient, are computed to assess the stock's volatility in relation to the benchmark. This statistical measure helps investors understand the risk profile of the stock relative to the overall market.

    - **Data Preparation for Modeling**: The processed data is then structured into sequences using a sliding window approach, preparing it for modeling with LSTM (Long Short-Term Memory) networks. This preparation ensures that the model can capture the temporal dependencies and trends necessary for accurate predictions. The data is then split into training and test sets to train the model and assess its performance.

This comprehensive workflow ensures that the data is clean, relevant, and properly formatted for effective model training, enhancing the reliability of the stock price predictions.

3. **Machine Learning and LSTM Prediction:** The application integrates advanced machine learning techniques to model and predict stock prices effectively.

    - **Linear Regression for Stock-Benchmark Correlation**: To understand the relationship between the stock and its benchmark, the script employs linear regression. This step assesses the correlation between the stock's price movements and the benchmark index, using the Beta coefficient as a key output. The Beta coefficient helps determine the stock's volatility relative to the benchmark, which is a critical metric for investors to gauge risk. A higher Beta indicates greater volatility compared to the market, while a lower Beta suggests lower risk.

    - **LSTM Model for Stock Price Prediction**: After establishing the baseline correlation, the script builds and trains an LSTM (Long Short-Term Memory) model, a type of deep learning architecture that excels in processing time-series data. The LSTM model is trained to predict the stock's closing price for the next 365 days based on historical data. By learning temporal dependencies and patterns in the stock price movements, the model forecasts future prices, providing valuable insights into potential price trends. The LSTM model captures long-term dependencies in the data, making it highly suited for time-series forecasting tasks like stock price prediction.

    Together, the combination of linear regression for initial analysis and LSTM for future predictions allows the script to provide comprehensive and robust insights into stock price dynamics.

4. **Visualization:** The application includes a powerful visualization component that displays both historical and predicted stock prices in an intuitive plot, allowing users to easily compare past performance with forecasted trends.

    - **Historical Stock Prices**: The historical prices of the stock, retrieved from Yahoo Finance, are plotted to show past price movements over the chosen time period (e.g., 1, 5, or 10 years). This provides users with a clear view of how the stock has performed in the past, highlighting trends, peaks, and dips.
    - **Predicted Stock Prices**: Alongside the historical data, the predicted stock prices for the next 365 days, generated by the LSTM model, are displayed. This gives users a visual representation of future price trends based on the model's forecasts.
    - **Intuitive Plot**: The plot is designed to be easy to understand, with clear labeling and distinct lines or markers representing the historical and predicted prices. This allows users to quickly identify how the predicted values align with the historical data and how they may evolve over the forecasted period. The plot can also be interactive, enabling users to zoom in, hover over specific data points for more details, or toggle between different views for enhanced analysis.

The script offers a robust and scalable foundation for stock price prediction, which is ideal for educational and practical financial applications. Expanding its features and improving model versatility can enhance its reliability and impact.

Why LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) is one of many types of Recurrent Neural Network RNN, it's also capable of catching data from past stages and use it for future predictions. In general, an Artificial Neural Network (ANN) consists of three layers:

1) Input layer
2) Hidden layers
3) Output layer.

In a NN that only contains one hidden layer the number of nodes in the input layer always depend on the dimension of the data, the nodes of the input layer connect to the hidden layer via links called 'synapses'. The relation between every two nodes from (input to the hidden layer), has a coefficient called weight, which is the decision maker for signals. The process of learning is naturally a continues adjustment of weights, after completing the process of learning, the Artificial NN will have optimal weights for each synapses.

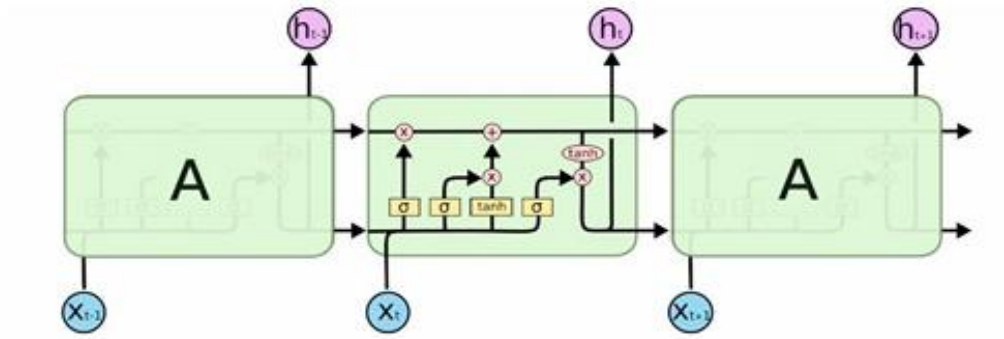
The hidden layer nodes apply a sigmoid or tangent hyperbolic *(tanh)* function on the sum of weights coming from the input layer which is called the activation function, this transformation will generate values, with a minimized error rate between the train and test data using the SoftMax function.

The values obtained after this transformation constitute the output layer of our NN, these value may not be the best output, in this case a back propagation process will be applied to target the optimal value of error, the back propagation process connect the output layer to the hidden layer, sending a signal conforming the best weight with the optimal error for the number of epochs decided. This process will be repeated trying to improve our predictions and minimize the prediction error.

After completing this process, the model will be trained. The classes of NN that predict future value base on passed sequence of observations is called Recurrent Neural Network (RNN) this type of NN make use of earlier stages to learn of data and forecast futures trends.

The earlier stages of data should be remembered to predict and guess future values, in this case the hidden layer act like a stock for the past information from the sequential data. The term recurrent is used to describe the process of using elements of earlier sequences to forecast future data.

RNN can't store long time memory, so the use of the Long Short-Term Memory (LSTM) based on "memory line" proved to be very useful in forecasting cases with long time data. In a LSTM the memorization of earlier stages can be performed trough gates with along memory line incorporated. The following diagram-1 describe the composition of LSTM nodes.



The internal structure of an LSTM

The ability of memorizing sequence of data makes the LSTM a special kind of RNNs. Every LSTM node most be consisting of a set of cells responsible of storing passed data streams, the upper line in each cell links the models as transport line handing over data from the past to the present ones, the independency of cells helps the model dispose filter of add values of a cell to another. In the end the sigmoidal neural network layer composing the gates drive the cell to an optimal value by disposing or letting data pass through. Each sigmoid layer has a binary value (0 or 1) with 0 "let nothing pass through"; and 1 "let everything pass through." The goal here is to control the state of each cell, the gates are controlled as follow:

- Forget Gate outputs a number between 0 and 1, where 1 illustration "completely keep this"; whereas, 0 indicates "completely ignore this."
- Memory Gate chooses which new data will be stored in the cell. First, a sigmoid layer "input door layer" chooses which values will be changed. Next, a *tanh* layer makes a vector of new candidate values that could be added to the state.
- Output Gate decides what will be the output of each cell. The output value will be based on the cell state along with the filtered and freshest added data.

Objective

The primary objective of this script is to create an intuitive and accessible platform that empowers users to predict stock prices using historical data. By harnessing advanced machine learning and deep learning methodologies, the application not only forecasts future price trends but also provides a comprehensive evaluation of stock performance in comparison to market benchmarks. This dual approach aims to assist investors in making informed decisions, combining cutting-edge technology with practical financial analysis.

Methodology

1. Data Acquisition

- **Source:** The historical stock data is sourced directly from the Yahoo Finance API which provides seamless access to extensive market data, ensuring accuracy and reliability in retrieving historical stock and benchmark prices. By utilizing this source, the script ensures the integrity of the data, which is crucial for meaningful analysis and forecasting.
- **Inputs:**
    - **Stock ticker**: This represents the unique identifier for the specific stock to be analyzed, such as "SBIN.NS" for State Bank of India on the National Stock Exchange.
    - **Benchmark ticker**: This corresponds to the identifier for the market index used as a reference for comparison, such as "^NSEI" for the Nifty 50 index.
    - **Historical data period**: This defines the time frame for which the data will be retrieved, offering options such as 1 year, 5 years, or 10 years, depending on the depth of analysis required).
- **Output:** Two datasets containing Date and Close Price for the stock and benchmark.

2. Data Preprocessing

- **Common Dates:** To ensure the accuracy and consistency of the analysis, the datasets for the stock and benchmark are merged based on common dates. This temporal alignment guarantees that the closing prices for both the stock and the benchmark correspond to the same trading days. By synchronizing the datasets in this manner, any discrepancies arising from differing trading calendars, missing data, or holidays are eliminated. This step is essential for maintaining the integrity of the comparative analysis, as it ensures that every data point accurately reflects the same time period for both the stock and the benchmark.
- **Spark DataFrames:** Both the stock and benchmark datasets are converted into Spark DataFrames to leverage the scalability and efficiency of distributed computing. This transformation is particularly advantageous for handling large datasets or performing resource-intensive operations, as Spark can process data in parallel across multiple nodes. During this step, the Close column, which contains the closing prices, is explicitly cast to FloatType to ensure uniformity and facilitate numerical computations. This preparation step enhances the dataset's compatibility with analytical tools and machine learning models, laying the groundwork for robust and efficient data processing.
- **Linear Regression:** The stock and benchmark prices are analyzed using linear regression with the help of the scikit-learn library. This method is used to model the

relationship between the stock's price movements and the benchmark's price changes. By fitting a linear regression model to the data, the Beta coefficient is computed, which quantifies the stock's volatility relative to the benchmark. A Beta greater than 1 indicates that the stock is more volatile than the benchmark, while a Beta less than 1 suggests it is less volatile.

3. LSTM Model Preparation

- **Data Scaling:** To ensure better performance during model training, the Close prices for both the stock and the benchmark are normalized using the MinMaxScaler. This data scaling technique transforms the values into a specified range, typically between 0 and 1, by adjusting the data based on the minimum and maximum values of the dataset. Normalizing the data ensures that all features have the same scale, which is crucial for many machine learning algorithms, as it prevents features with larger numerical ranges from dominating the model training process. This step improves the model's convergence rate, stability, and accuracy, especially for algorithms sensitive to the scale of input data, such as neural networks and gradient-based methods.
- **Sequence Creation:** A sliding window approach is employed to create sequences for input into the Long Short-Term Memory (LSTM) model, a type of recurrent neural network (RNN) well-suited for time series forecasting. In this method, a fixed number of time_step values are used to form each sequence, where the goal is to predict the next value in the sequence. The sliding window moves through the historical data, capturing overlapping segments of stock and benchmark prices. Each sequence includes the past time_step values, which the LSTM model will use to learn temporal patterns and dependencies. By training on these sequences, the model can effectively forecast future stock prices based on the historical context provided by the preceding time steps. This approach enables the LSTM model to capture trends, seasonality, and other dynamic behaviors within the time series data.
- **Train-Test Split:** The dataset is divided using an 80-20 train-test split, where 80% of the data is allocated for training the model, and the remaining 20% is reserved for testing and evaluating its performance. This division ensures that the model is trained on a large portion of the data, allowing it to learn the underlying patterns and relationships. The 20% test set acts as a holdout, providing an unbiased evaluation of the model's ability to generalize to unseen data. By reserving a separate portion for testing, the risk of overfitting is minimized, and the model's predictive accuracy and robustness can be properly assessed.

4. LSTM Model Development

- **Architecture:**

- The architecture of the model consists of two LSTM layers, each with 50 units, designed to capture the temporal dependencies and patterns within the time series data. The LSTM layers are equipped with dropout layers to prevent overfitting, ensuring the model generalizes well to new, unseen data by randomly dropping a fraction of the input units during training. This regularization technique helps improve the model's robustness and reduces the risk of memorizing the training data.
- Following the LSTM layers, a dense layer is added to produce the final output, which represents the predicted stock price for the next time step. This layer aggregates the learned features from the LSTM layers and delivers the output in a format suitable for forecasting. The combination of LSTM layers for sequential learning and a dense layer for output generation forms an effective architecture for time series prediction tasks.

- **Training:**
  - The model is trained for a user-defined number of epochs (with a default value of 10), which dictates how many times the entire dataset will pass through the network during training. During each epoch, the model learns from the data by adjusting its internal parameters. The Adam optimizer, a popular gradient-based optimization algorithm, is used to minimize the loss function. Adam adapts the learning rate during training, improving the model's efficiency and convergence speed.
  - The mean squared error (MSE) loss function is employed to quantify the difference between the predicted and actual values, penalizing large errors more than small ones. This loss function is commonly used for regression tasks, such as stock price prediction, as it ensures the model learns to minimize the variance between its predictions and the actual outcomes. Together, the combination of the Adam optimizer and MSE loss function enables the model to effectively learn and make accurate predictions over the course of the training process.

5. Future Price Prediction

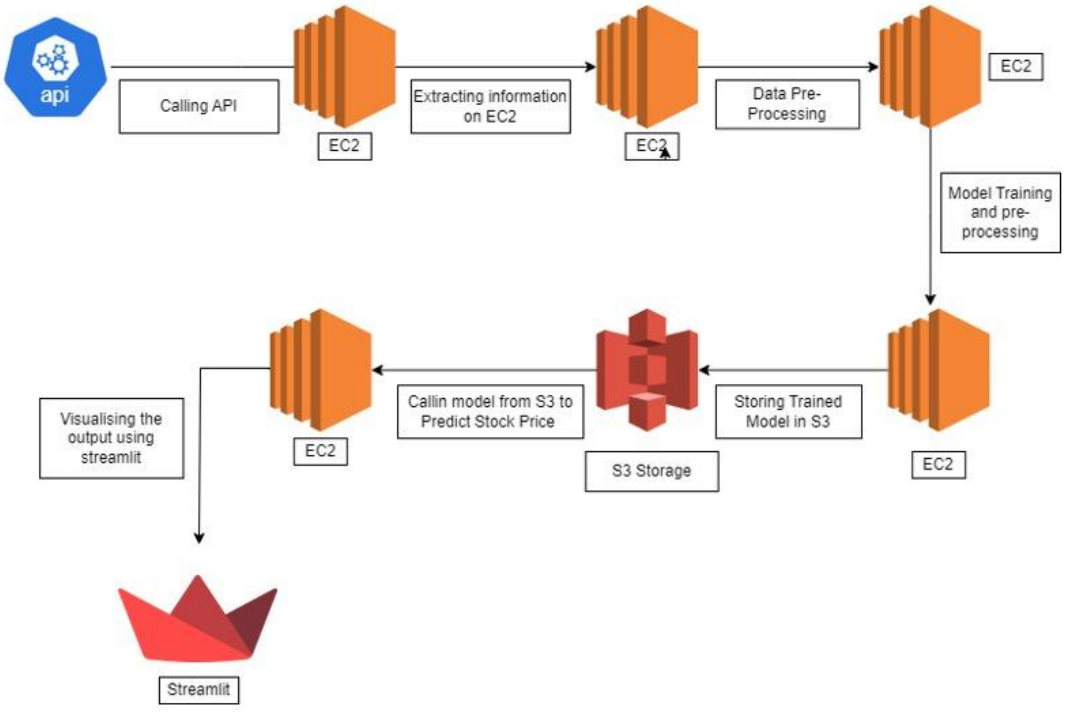
- **Prediction Horizon:** The prediction horizon is set to forecast stock prices for the next 365 days, starting from the last available sequence of historical data. Using the most recent data points, the model predicts the stock's closing prices over the next year, providing a one-year forecast.
- **Inverse Scaling:** After the model generates predictions for future stock prices, the results are in the scaled range (typically between 0 and 1) due to the earlier application of the MinMaxScaler. To convert the predictions back to their original scale, the inverse scaling process is applied. This is done using the inverse of the **MinMaxScaler**, which reverts the normalized values back to their original price

range. By performing inverse scaling, the predicted stock prices are returned to their real-world values, allowing them to be interpreted and used in practical financial analysis, such as portfolio management and investment decision-making.

6. Visualization

- For clear and intuitive comparison, both the historical and predicted stock prices are visualized using matplotlib, a powerful plotting library in Python. This visualization allows users to easily observe how the predicted prices align with the actual historical data, providing insights into the model's performance and accuracy.
- Additionally, the application integrates Streamlit's plotting functionality to present these results interactively. Streamlit allows users to interact with the visualizations in real-time, offering features like zooming, panning, and dynamic updates, which enhance the user experience. This interactive approach helps users explore the predictions more effectively, enabling them to make better-informed decisions based on the displayed trends and forecasts.

Architecture



1. Calling API

- Purpose: Data is retrieved from an external API (likely providing financial or stock market data).
- Implementation: An EC2 instance (Amazon Elastic Compute Cloud) is used to call the API and fetch raw data.

2. Extracting Information on EC2

- Purpose: The raw data obtained from the API is extracted and processed for initial use.
- Implementation: Data extraction processes are performed on the EC2 instance to ensure the data is in a usable format.

3. Data Pre-Processing

- Purpose: Prepares the extracted data for training a machine learning model.
- Tasks:
    - Cleaning data (e.g., handling missing values, removing duplicates).
    - Transforming data into a structured format suitable for modeling.
- Implementation: Done on another EC2 instance.

4. Model Training and Pre-Processing

- Purpose: Train a predictive machine learning model using the pre-processed data.
- Tasks:
    - Feature engineering.
    - Training the model to predict stock prices.
- Implementation: A dedicated EC2 instance is used for resource-intensive tasks like model training.

5. Storing Trained Model in S3

- Purpose: The trained machine learning model is stored in Amazon S3 (Simple Storage Service) for later retrieval.
- Reason:
    - Scalability and durability of S3 storage.
    - Enables centralized access to the model for deployment.

6. Calling Model from S3 to Predict Stock Price

- Purpose: The stored model is retrieved from S3 and used for making predictions.

- Implementation: An EC2 instance loads the model to process live or batch data and generate stock price predictions.

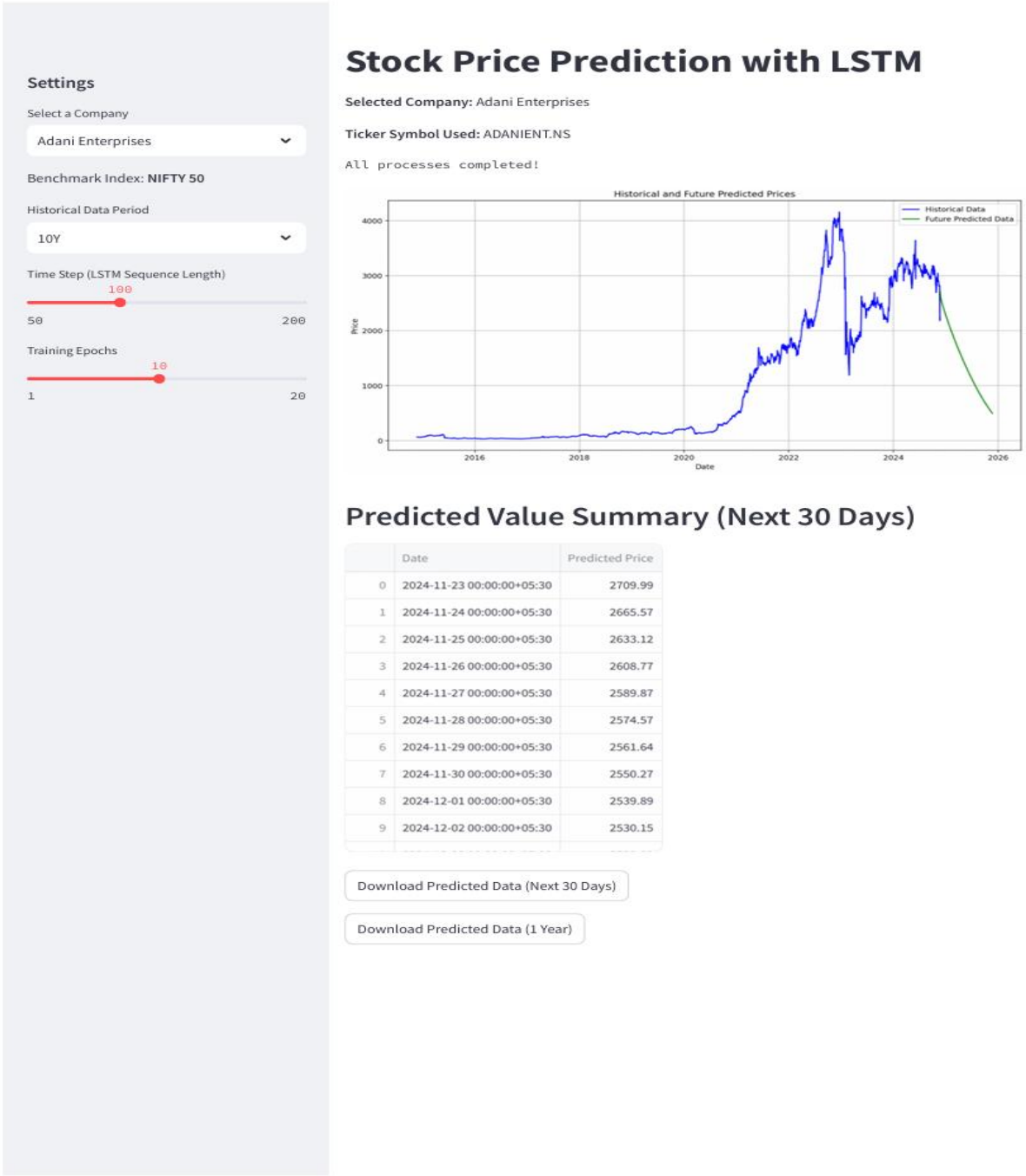
7. Visualizing Output Using Streamlit

- Purpose: The predictions are visualized on a user-friendly web interface created using Streamlit.
- Tasks:
    - Displaying graphs or dashboards.
    - Showing stock price trends, predictions, or other insights.
- Implementation: The output from the EC2 instance is passed to Streamlit for visualization.

End-to-End Flow:

1. Data Ingestion: Data is fetched via API calls and processed on EC2 instances.
2. Data Preparation: Pre-processing steps refine the data for training.
3. Model Development: The model is trained, stored in S3, and used for predictions.
4. Output Presentation: Streamlit provides an interactive platform to view the predicted results.

Result



Streamlit Result Snapshot

Next 30 Days Predicted Values

Date	Predicted Price
2024-11-23 00:00:00+05:30	2709.995
2024-11-24 00:00:00+05:30	2665.568
2024-11-25 00:00:00+05:30	2633.12
2024-11-26 00:00:00+05:30	2608.774
2024-11-27 00:00:00+05:30	2589.865
2024-11-28 00:00:00+05:30	2574.567
2024-11-29 00:00:00+05:30	2561.644
2024-11-30 00:00:00+05:30	2550.265
2024-12-01 00:00:00+05:30	2539.885
2024-12-02 00:00:00+05:30	2530.147
2024-12-03 00:00:00+05:30	2520.821
2024-12-04 00:00:00+05:30	2511.763
2024-12-05 00:00:00+05:30	2502.882
2024-12-06 00:00:00+05:30	2494.122
2024-12-07 00:00:00+05:30	2485.447
2024-12-08 00:00:00+05:30	2476.838
2024-12-09 00:00:00+05:30	2468.282
2024-12-10 00:00:00+05:30	2459.77
2024-12-11 00:00:00+05:30	2451.298
2024-12-12 00:00:00+05:30	2442.862
2024-12-13 00:00:00+05:30	2434.462
2024-12-14 00:00:00+05:30	2426.094
2024-12-15 00:00:00+05:30	2417.759
2024-12-16 00:00:00+05:30	2409.455
2024-12-17 00:00:00+05:30	2401.181
2024-12-18 00:00:00+05:30	2392.937
2024-12-19 00:00:00+05:30	2384.722
2024-12-20 00:00:00+05:30	2376.535
2024-12-21 00:00:00+05:30	2368.378
2024-12-22 00:00:00+05:30	2360.249

Strengths

- **End-to-End Pipeline:** The application offers a comprehensive end-to-end pipeline that integrates all essential steps for stock price prediction, from data acquisition and preprocessing to modeling and visualization. This streamlined workflow ensures that users can easily go from raw financial data to actionable insights without the need for manual intervention at each stage, making the process efficient and seamless.
- **Scalable:** Built on PySpark, the application is highly scalable and can efficiently handle large datasets. PySpark enables distributed computing, allowing the script to process vast amounts of financial data quickly, even for extensive historical periods or multiple stocks. This scalability is crucial for real-time analysis and for users working with large volumes of market data, ensuring the application remains performant as data sizes grow.
- **User-Friendly:** The interactive and user-friendly interface ensures that even non-technical users can engage with the application effectively. Through a simple interface, users can input stock tickers, adjust prediction parameters, and visualize results directly within the app. The intuitive design, coupled with interactive visualizations, makes it accessible to investors, analysts, and financial professionals who may not have a technical background, while still offering the depth and sophistication required for advanced analysis.

Areas of Improvement

1. **Feature Limitation:** The predictions in this model are based exclusively on the closing prices of the stock and benchmark, which limits the amount of information considered in the forecasting process. By focusing only on closing prices, the model ignores other potentially influential market indicators, such as trading volume, market sentiment, economic indicators, or technical factors like moving averages and volatility measures. These additional features could provide a more comprehensive view of market conditions, potentially improving the model's ability to capture broader trends and making the predictions more accurate. The exclusion of these variables represents a limitation, as it overlooks other aspects of the market that could affect stock price movements.
2. **Heavy LSTM Dependence:** The current script heavily relies on LSTM (Long Short-Term Memory) models for stock price prediction, which may not always be the best choice for every dataset or user preference. LSTM models are well-suited for sequential data and capturing long-term dependencies, but they can be computationally expensive and require careful tuning to avoid overfitting. Depending on the nature of the data—such as if it contains strong seasonality or is highly volatile—alternative models like ARIMA, Random Forests, or Gradient Boosting Machines could yield better results. To enhance flexibility and improve performance, the script could offer users the ability to choose from various model architectures based on their specific data characteristics or preferences, allowing for a more tailored approach to stock price prediction.

The script provides a robust foundation for stock price prediction, focusing on user interactivity, big data scalability, and advanced time-series modelling. Addressing its

limitations, expanding model options, and enriching feature engineering would significantly enhance its practical applications in financial analytics.

Conclusion

The provided script effectively implements an end-to-end stock price prediction solution, utilizing machine learning, deep learning, and big data techniques to offer a robust platform for forecasting stock prices. By integrating LSTM for advanced time-series forecasting and PySpark for scalability, it demonstrates the potential to handle large and complex financial datasets efficiently. The user-friendly interface, powered by Streamlit, allows investors and analysts to easily input parameters, visualize results, and make data-driven decisions.

However, the application can be further improved by addressing the following enhancements:

- **Robust Error Handling**: Implementing comprehensive error handling to manage invalid inputs, missing values, and data inconsistencies would make the application more resilient and reliable for users.
- **Expanding Model Options**: Introducing additional forecasting algorithms, such as ARIMA (Auto-Regressive Integrated Moving Average) or GRU (Gated Recurrent Unit), would offer users more flexibility and help tailor the predictions to different types of time series data.
- **Enriching the Feature Set**: Including additional market indicators like volume, moving averages, and other technical or macroeconomic factors would improve the model's ability to capture various market dynamics, resulting in more accurate predictions.
- **Full Utilization of PySpark for Preprocessing**: Leveraging PySpark for all preprocessing tasks, including feature extraction and scaling, would enhance scalability for processing large datasets and improve the overall performance of the application.

Overall, this script serves as a well-rounded prototype that strikes a balance between accessibility and technical sophistication. It offers a strong foundation for predictive financial modeling and can be further refined to cater to the evolving needs of investors, analysts, and financial data scientists.

Reference

- **https://www.atlantis-press.com/article/125971905.pdf**
- **https://www.researchgate.net/publication/352510074_STOCK_PRICE_PREDICTION_USING_LSTM**
