REAL TIME STOCK PRICE ANALYSIS

Project Submitted to the
SRM University AP, Andhra Pradesh
for the partial fulfillment of the requirements to award the degree of

Master of Technology
in
Data Science
School of Engineering & Sciences

submitted by

MADDI ESWAR(AP24122060014)

BHANU PRAKASH DAVALURI(AP24122060010)

Under the Guidance of

Dr. Rajiv Senapati

Department of Computer Science & Engineering
SRM University-AP
Neerukonda, Mangalgiri, Guntur
Andhra Pradesh - 522 240
May 2025

DECLARATION

I undersigned hereby declare that the project report **Real Time Stock Price Analysis** submitted for partial fulfillment of the requirements for the award of degree of Master of Technology in the Computer Science & Engineering, SRM University-AP, is a bonafide work done by me under supervision of Dr. Rajiv Senapati. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree of any other University.

Place	: Amaravati	Date	: April 15, 2025
Name of student	: Maddi Eswar	Signature	: ....
Name of student	: D. Bhanu Prakash	Signature	: ....

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SRM University-AP

Neerukonda, Mangalgiri, Guntur

Andhra Pradesh - 522 240

CERTIFICATE

This is to certify that the report entitled **Real Time Stock Price Analysis** submitted by **MADDI ESWAR, BHANU PRAKASH DAVALURI,** to the SRM University-AP in partial fulfillment of the requirements for the award of the Degree of Master of Technology in in is a bonafide record of the project work carried out under my/our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Guide

Name : Dr. Rajiv Senapati

Signature: ....

ACKNOWLEDGMENT

I wish to record my indebtedness and thankfulness to all who helped me prepare this Project Report titled **Real Time Stock Price Analysis** and present it satisfactorily.

I am especially thankful for my guide and supervisor Dr. Rajiv Senapati in the Department of Computer Science & Engineering for giving me valuable suggestions and critical inputs in the preparation of this report. I am also thankful to Murali Krishna Enduri, Head of Department of Computer Science & Engineering for encouragement.

My friends in my class have always been helpful and I am grateful to them for patiently listening to my presentations on my work related to the Project.

MADDI ESWAR, BHANU PRAKASH DAVALURI
(Reg. No. AP24122060014, AP24122060010)
M. TECH - DATA SCIENCE
Department of Computer Science & Engineering
SRM University-AP

ABSTRACT

This project focuses on building a real-time Stock Price Analysis Workflow that integrates data collection, streaming, storage, processing, and machine learning for financial insights. The workflow begins by collecting live stock data from external APIs, which is then streamed using Apache Kafka to ensure efficient and reliable data transmission. The streamed data is stored in MongoDB, a flexible NoSQL database that supports large volumes of time-series data. Key stock market indicators such as Open, High, Low, Close, and Volume are captured and used for analysis. The Open price refers to the first trade at the beginning of a time interval, while the High and Low represent the maximum and minimum prices during that interval, respectively. The Close price is the last trade before the interval ends, and Volume indicates the total number of shares traded in that period. These metrics are crucial for understanding market behavior and trends. After data collection and storage, PySpark is used for distributed data processing and feature engineering. The cleaned and processed data is then fed into machine learning models to uncover hidden patterns, detect anomalies, and forecast future stock prices. This end-to-end pipeline demonstrates how real-time technologies and data science can be combined to support advanced financial analysis and timely decision-making.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Stock markets move fast. Prices go up and down every second, and investors need the most up-to-date information to make smart decisions. In today's world, just looking at stock data at the end of the day isn't enough. We need systems that can collect and analyze stock prices in real time to keep up with the speed of the market.

That's what this project is all about. It's called Stock Price Analysis, and it's built to handle stock market data the moment it becomes available. The process starts by collecting live stock data using APIs. This live data is streamed using Apache Kafka, which helps handle continuous flows of information without any delay or data loss. Once the data is streamed, it gets stored in MongoDB, a type of database that's good at handling large amounts of time-based data.

The main values we look at are simple but powerful: Open, High, Low, Close, and Volume. These tell us when a stock started trading in a time period, how high or low the price went, what it closed at, and how many shares were traded. This information helps us understand what happened in the market during each interval—whether it was a stable period, a sudden spike, or a drop.

After collecting and storing the data, we use PySpark to process it. PySpark allows us to work with big data quickly and efficiently. Once the data is cleaned and prepared, we apply machine learning to find patterns and make predictions about future stock prices.

By combining real-time data collection, fast processing, and smart predictions, this project shows how modern tools can be used to make sense of fast-moving stock markets. It's not just about looking at numbers—it's about turning those numbers into insights that can help people make better financial decisions.

This kind of system isn't just useful for large financial institutions—it can be helpful for individual traders, researchers, and developers who are interested in exploring stock behavior or building their own financial tools. It also shows how different technologies like Kafka, MongoDB, PySpark, and machine learning can work together to solve real-world problems. By building this end-to-end workflow, the project highlights the importance of automation, speed, and accuracy in modern stock analysis, making it a valuable solution for today's data-driven financial world.

Chapter 2

MOTIVATION

The stock market is constantly changing, and prices can move up or down in just a matter of seconds. This makes it really important for investors and traders to have up-to-date information so they can make quick and informed decisions. Relying on outdated or slow data can cause missed opportunities or bad investments. That's what motivated me to create a system that can analyze stock prices in real-time, as soon as the data becomes available.

With so many advanced tools and technologies out there—like Apache Kafka, MongoDB, PySpark, and machine learning—it seemed like the perfect time to explore how they can work together to solve this problem. These tools make it possible to handle and analyze large volumes of live data quickly and efficiently, which is exactly what's needed for stock market analysis.

On top of that, working on this project was a chance to gain practical experience with real-time data processing and predictive modeling. It's not just about building something cool, but also about learning how to use these technologies to help people make better, faster decisions in the financial world.

Chapter 3

LITERATURE SURVEY

[1] Stock market prediction and real-time data analysis have long been areas of interest for researchers and financial analysts. Traditional methods such as ARIMA, Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Decision Trees have been widely applied for forecasting purposes. These models generally work well for smaller datasets or historical data, but struggle to keep up with the increasing volume, velocity, and variety of modern stock market data.

With the rise of big data technologies, a shift has occurred in the way stock data are processed and analyzed. Platforms such as Apache Kafka and Apache Spark have made it possible to handle massive amounts of real-time data with low latency and high scalability. Kafka enables real-time data ingestion from various sources, such ash as stock exchanges and financial news APIs, while Spark Streaming provides the ability to process and analyze the data as it arrives.

Several studies have shown that combining real-time stream processing with machine learning techniques leads to more accurate and timely predictions. For example, integrating Spark with predictive models enables online learning and instant decision making, which is crucial for applications like high-frequency trading. NoSQL databases such as MongoDB further enhance the system by offering flexible and fast storage solutions for both raw and processed data.

Recent research also emphasizes the importance of distributed architectures in stock market analysis. Systems that use Hadoop, MapReduce, or Spark in cluster environments can perform parallel processing, reducing computational time and improving system performance. These tools are used not only for prediction, but also for detecting trends, anomalies, and generating financial alerts in real time.

Overall, the literature suggests a strong move toward using real-time big data frameworks combined with machine learning algorithms to build scalable, accurate, and responsive financial analysis systems. This evolution highlights the limitations of older methods and demonstrates how modern architectures can provide significant advantages for stock market forecasting and decision support.

[2] The paper titled "Stock Market Prediction with Political Data: Analysis SPPDA Model for Handling Big Data" explores how political information can be combined with stock market data to improve the accuracy of predictions. Unlike traditional approaches that mainly rely on past stock values, this research highlights the importance of including political news and events in the forecasting process. The authors proposed a model called SPPDA, which stands for Stock Prediction with Political Data Analysis. This model is designed to work with large datasets using big data technologies, and it uses political sentiment to enhance prediction results.

To achieve this, the authors collected data from online sources that included both stock information and political news articles. They applied natural language processing techniques to understand the sentiment expressed in political news, such as whether the tone was positive or negative. This sentiment was then used as an input alongside historical stock data. To handle the large volume of information, the system was built using big data tools like Hadoop and Spark, which allow the model to process and analyze the data quickly and efficiently.

For prediction, the model uses machine learning techniques such as Decision Tree, Random Forest, and XGBoost. The results showed that Random Forest and XGBoost performed better in terms of accuracy. By combining political sentiment with financial data, the model was able to make more accurate predictions, especially during politically sensitive times like elections or policy announcements.

This paper shows that using big data technologies together with sentiment analysis can improve stock market forecasting. It demonstrates the value of real-time, large-scale data processing and suggests that combining structured and unstructured data can help in making better financial decisions. The ideas from this paper have guided my project, especially the integration of political data and the use of tools like Spark and Hadoop to handle real-time stock market analysis.

[3] The paper titled ”An Improved Prediction Model for Car Prices Using Machine Learning Techniques” mainly focuses on improving the accuracy of car price prediction by using different machine learning models. The authors collected car data from online sources and performed multiple preprocessing steps like removing missing values, converting categorical data into numerical form, and selecting the most relevant features. This helped in building better models. They tested various algorithms including Linear Regression, Decision Tree, Random Forest, and XGBoost.

Among all these, XGBoost gave the best performance. It showed higher accuracy and lower error rates, which means it was able to learn complex patterns in the data effectively. The paper clearly explains how each step in the data cleaning and preparation process affects the final model performance. It also emphasizes the importance of choosing the right features and models to get better prediction results.

This research is helpful for my project, which also deals with car price prediction, but with a different approach. While this paper uses machine learning in a typical setup, I am using Big Data tools like Hive and Hadoop to manage and process large-scale datasets. However, the overall goal of predicting car prices remains the same. The techniques used in the paper give me a good idea of how important data preprocessing and feature selection are, even when working with big data platforms.

Moreover, the evaluation methods used in the paper, like R^2 score and Mean Absolute Error (MAE), help me understand how to measure the accuracy of predictions. I can apply similar evaluation techniques in my project to check how well my system performs. Overall, this paper supports my work and gives a strong foundation to build a more efficient car price prediction model.

[4] The paper titled "Credit Risk Evaluation Using Machine Learning and Deep Learning Algorithms" mainly focuses on how machine learning and deep learning models can be used to predict whether a person will default on a loan. Even though the paper is about credit risk, many of the techniques used—like data preprocessing, model comparison, and performance evaluation—are highly relevant to stock market analysis as well. The study used models such as Logistic Regression, Random Forest, Support Vector Machines, and Multi-layer Perceptrons, comparing their accuracy and efficiency.

In real-time stock analysis, similar models can be used to predict stock trends or price movements. One key takeaway from this paper is the importance of cleaning and preparing the data. The authors used techniques like normalization, handling missing values, and balancing datasets, which are also crucial in stock data due to its dynamic and noisy nature. Another interesting point is how they tackled imbalanced data using SMOTE, which can also be applied in financial applications where certain events (like a crash or spike) are rare but important.

The deep learning approach, especially using MLPs, showed better performance in the paper, which suggests that applying deep learning models to real-time stock analysis can help capture complex patterns in market data. Additionally, the paper discussed SHAP values for model explainability, which is important in stock analysis too, as understanding which features (like volume, technical indicators, or news sentiment) influenced a prediction can make the system more transparent.

Overall, this paper gives a solid foundation for applying machine learning in financial applications. The preprocessing methods, algorithm choices, and evaluation techniques discussed can be adapted for building a real-time stock analysis system that predicts market behavior and supports decision-making.

[5] In recent years, the analysis of stock market data has evolved significantly due to advancements in data science and the availability of real-time data through APIs and online sources. The reference paper titled "Real Time Stock Market Analysis Using Python" serves as an important foundation for understanding the application of Python in live stock market analytics.

The authors have demonstrated the feasibility of collecting, processing, and visualizing stock data in real time using Python libraries. They focused on fetching stock market data from Yahoo Finance using the yfinance module, which provides historical and live stock price data. The study emphasizes the role of Python libraries such as Pandas, NumPy, Matplotlib, and Plotly in performing data manipulation and visualization. This helps users observe market trends, compare stock performance over time, and make informed decisions.

The system architecture discussed in the paper includes data collection, data cleaning, visualization, and real-time updates. Real-time analysis enables traders and analysts to gain insights into stock behavior, track market movements, and anticipate potential shifts in market value.

Moreover, the paper highlights how machine learning algorithms can be integrated for predictive analysis. Though not deeply implemented in the study, the concept paves the way for future work where real-time data can be fed into regression models or classification algorithms to forecast price movement or identify trading signals.

This reference supports the development of our project by underlining the importance of automation in fetching stock data, performing live analysis, and creating user-friendly dashboards. By leveraging tools like Streamlit for building interactive web interfaces and integrating technical indicators such as moving averages, one can build a robust stock monitoring system.

In conclusion, the referenced work provides essential methodologies and tools that form the backbone of any real-time stock analysis project. It offers a practical starting point for implementing live stock monitoring and lays the groundwork for incorporating more advanced features like prediction and alerting systems.

[6] Stock market prediction is a highly dynamic and complex problem that has attracted the attention of researchers for decades. With the advancement of machine learning and deep learning, there has been a significant shift from traditional statistical models to intelligent systems capable of

learning from historical and real-time stock data. The reference paper, titled "Real Time Stock Market Prediction using Machine Learning", presents a comprehensive exploration of predictive modeling using advanced ML techniques.

The authors propose the use of machine learning algorithms such as Support Vector Machines (SVM), Random Forest, and Long Short-Term Memory (LSTM) networks to forecast stock prices with improved accuracy. These models are trained using historical stock data enriched with technical indicators like Moving Average (MA), Relative Strength Index (RSI), and Bollinger Bands. The incorporation of such features allows the models to capture both short-term fluctuations and long-term trends in the market.

A key focus of the study is on real-time data analysis, where models are not only trained on past data but also adapted to respond to live market conditions. The paper emphasizes the importance of preprocessing steps, including data normalization, feature engineering, and handling of missing values, which significantly affect model performance.

The authors demonstrate that LSTM, a variant of Recurrent Neural Networks (RNN), excels in capturing time-dependent patterns in sequential financial data. Its memory structure enables it to learn long-term dependencies better than traditional feedforward networks, making it particularly effective for time-series prediction like stock prices.

Performance evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score are used to assess the accuracy and robustness of the models. The study concludes that LSTM provides the most promising results among all tested models for real-time forecasting.

This research aligns closely with the objectives of the current project, which aims to analyze and visualize real-time stock data using machine learning. The paper serves as a strong foundational reference, highlighting the effectiveness of deep learning models and technical indicators in building predictive stock analysis systems.

[7]The paper titled "Stock Price Analysis and Prediction" by Chintan Vora, Bhavya Shah, Dhairya Sheth, and Prof. Nasim Banu Shah (2021) presents a system for predicting stock prices using Machine Learning techniques. The main goal of the system is to help investors make better decisions by predicting future stock prices based on historical data and technical indicators.

The authors observed how the stock market behaved during the COVID-19 pandemic. While many sectors were shut down, the stock market continued to operate, although it was highly volatile. This inspired them to create a system that uses technology to predict stock trends more accurately.

To build the system, they used different Machine Learning algorithms such as Linear Regression, Decision Tree, Random Forest, K-Nearest Neighbors (KNN), and Long Short-Term Memory (LSTM) networks. Among all the models, LSTM gave the best results because it works well with time-series data, like stock prices, which change over time.

The dataset used in the project included stock data from Nifty-50 companies. It had values like the opening price, closing price, highest and lowest prices of the day, and volume of stocks traded. The authors also used technical indicators like Moving Average (MA) and PE ratio to improve the accuracy of the predictions.

After training and testing the models, they found that the LSTM model gave predictions that were very close to the actual stock prices. They also used visual graphs and confusion matrices to compare the performance of all models.

In conclusion, the paper shows that using Machine Learning models along with technical indicators can help predict stock prices more effectively. While it is impossible to always get accurate results due to the unpredictable nature of the stock market, the proposed system provides a good way to assist investors in making informed decisions. The authors also suggested that future improvements could include using real-time data and combining multiple models for better accuracy.

This paper is useful for my project as it gives a clear idea about how to use machine learning for stock price analysis and which models are most effective for such predictions.

[8] The paper titled "Stock Price Level: A Predictive Analysis" by Pronab Sen, published in Economic and Political Weekly (1974), presents a macroeconomic approach to stock price prediction. Instead of focusing on company-specific or industry-specific analysis, the author examines how broader economic indicators influence stock market trends. The main objective of the study is to develop a predictive model for stock prices using key variables such as money supply and Gross National Product (GNP).

The research uses historical data from the United States for its accuracy and availability. Consider four main independent variables: the absolute level of the money supply (M1), the rate of change in the money supply, the level of GNP and the rate of change in GNP. The stock price is represented using the Standard Poor's 500 Index. Using a multiple regression model, the study analyzes how these variables from the previous quarter influence the current stock price level.

The paper also reviews earlier literature, including the works of economists such as Sprinkel, Keran, Homa, and Jaffee. These earlier studies often emphasized the indirect influence of money supply on stock prices through channels like interest rates and corporate earnings expectations. However, Pronab Sen's research brings attention to the direct statistical relationship between GNP and stock prices, making it a unique contribution to the field.

While the model is not extremely precise, the author stresses that the goal was to create a simple and easy-to-use tool for the average investor rather than a highly complex financial forecasting system. The paper encourages the use of macroeconomic indicators for stock price analysis and opens up further scope for enhancing model accuracy by incorporating other variables such as inflation or price level changes.

Chapter 4

DESIGN AND METHODOLOGY

The methodology of this project revolves around creating an end-to-end pipeline that handles real-time stock market data efficiently from the point of collection to insightful visualizations. This pipeline incorporates several advanced tools and frameworks, namely API, Apache Kafka, MongoDB, PySpark, Machine Learning models, and Power BI. Each of these components plays a vital role in ensuring that the data is processed, analyzed, and visualized with accuracy, speed, and scalability. The methodology is designed to address the dynamic and high-frequency nature of the stock market by ensuring low-latency data handling and timely predictions.

4.1 DATASET DESCRIPTION

4.1.1 Dataset Columns Description

The dataset used in this project consists of six key columns: `date`, `open`, `high`, `low`, `close`, and `volume`. Each of these columns provides essential insights into the daily behavior of stock prices and trading activity. Below is a detailed description of each column:

Date This column captures the specific trading date in the format `DD-MM-YYYY`. It is used to track the temporal progression of stock prices and align the data points in a chronological order. The `date` field serves as a time reference and is fundamental in performing time-series analysis, plotting historical trends, and organizing data for machine learning models.

Open The `open` column represents the stock price at which trading began on a particular day. This price reflects the market sentiment at the beginning of the trading session. It is often compared with the previous day's closing price to determine if the market opened higher or lower, which may indicate overnight sentiment or reactions to external events.

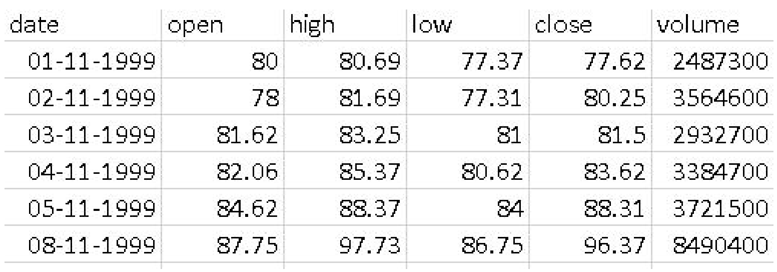
High The `high` column denotes the highest price that the stock reached during the trading day. It provides insight into the upper limit of price movement for that day and can be used to assess the bullish strength in the market. Analysts often use the high price in calculations of volatility and to identify breakout levels.

Low This column shows the lowest price at which the stock was traded during the day. The `low` value is used to determine the range of price fluctuation and is particularly important in assessing daily volatility and risk. The difference between the high and low prices is often used as a measure of the day's trading range.

Close The `close` column indicates the final price at which the stock traded when the market closed for the day. It is one of the most commonly used values in technical and fundamental analysis. The closing price is often used as a target variable in predictive modeling and is critical for calculating daily returns, moving averages, and trend indicators.

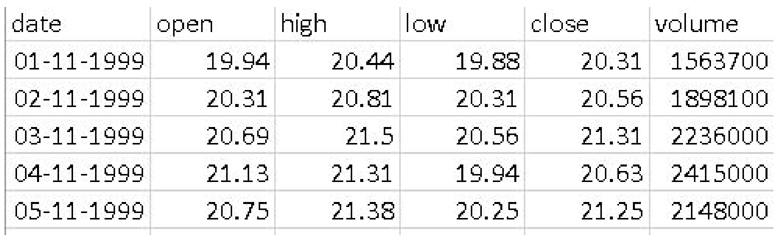
Volume The `volume` column records the total number of shares traded on a specific day. It serves as a measure of market activity and liquidity. Higher trading volume often signifies strong investor interest, and it is frequently used alongside price movements to confirm trends or reversals.

The dataset used in this project consists of historical stock price data for ten major technology companies, covering different time periods depending on the company. Most of the companies have data ranging from November 1, 1999 to April 11, 2025, comprising 6401 records each. These include Apple (AAPL), Advanced Micro Devices Inc. (AMD), Amazon (AMZN), IBM (IBM), Infosys (INFY), Microsoft (MSFT), Oracle (ORCL), and SAP SE (SAP). For Accenture (ACN), the data spans from July 19, 2001 to April 11, 2025, totaling 5969 records. In the case of Google (GOOGL), the records begin from August 19, 2004 to April 11, 2025, with a total of 5196 entries. Each record in the dataset typically includes key stock attributes such as the opening price, closing price, high, low, adjusted close, and trading volume for each trading day. These records enable various statistical and machine learning techniques to be applied for trend analysis, performance comparison, and behavioral study of stocks over time.



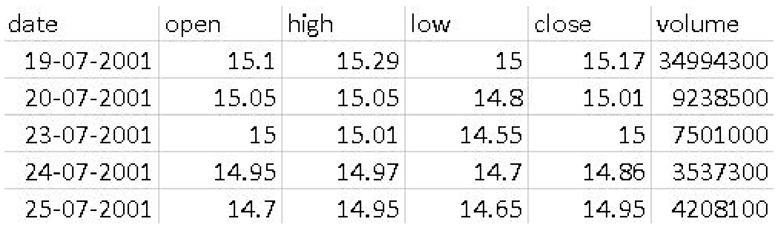
date	open	high	low	close	volume
01-11-1999	80	80.69	77.37	77.62	2487300
02-11-1999	78	81.69	77.31	80.25	3564600
03-11-1999	81.62	83.25	81	81.5	2932700
04-11-1999	82.06	85.37	80.62	83.62	3384700
05-11-1999	84.62	88.37	84	88.31	3721500
08-11-1999	87.75	97.73	86.75	96.37	8490400

Figure 4.1: Apple Inc



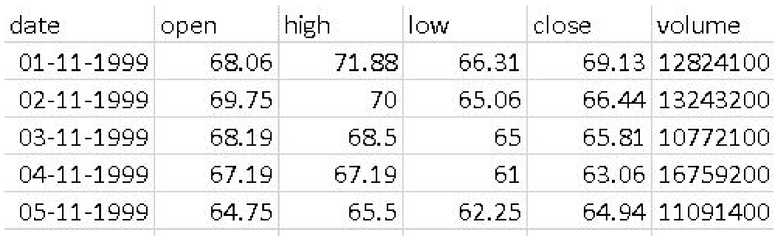
date	open	high	low	close	volume
01-11-1999	19.94	20.44	19.88	20.31	1563700
02-11-1999	20.31	20.81	20.31	20.56	1898100
03-11-1999	20.69	21.5	20.56	21.31	2236000
04-11-1999	21.13	21.31	19.94	20.63	2415000
05-11-1999	20.75	21.38	20.25	21.25	2148000

Figure 4.2: Advanced Micro Devices Inc



date	open	high	low	close	volume
19-07-2001	15.1	15.29	15	15.17	34994300
20-07-2001	15.05	15.05	14.8	15.01	9238500
23-07-2001	15	15.01	14.55	15	7501000
24-07-2001	14.95	14.97	14.7	14.86	3537300
25-07-2001	14.7	14.95	14.65	14.95	4208100

Figure 4.3: Accenture Plc



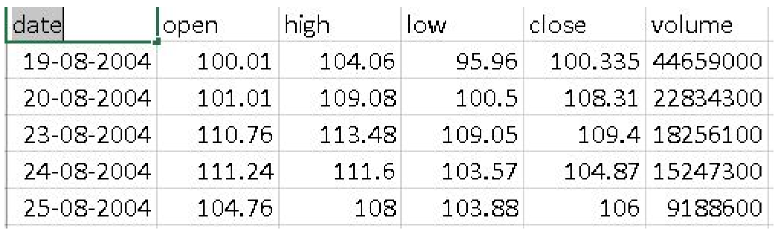
date	open	high	low	close	volume
01-11-1999	68.06	71.88	66.31	69.13	12824100
02-11-1999	69.75	70	65.06	66.44	13243200
03-11-1999	68.19	68.5	65	65.81	10772100
04-11-1999	67.19	67.19	61	63.06	16759200
05-11-1999	64.75	65.5	62.25	64.94	11091400

Figure 4.4: Amazon.com Inc

4.2 REAL-TIME DATA ACQUISITION THROUGH APIS

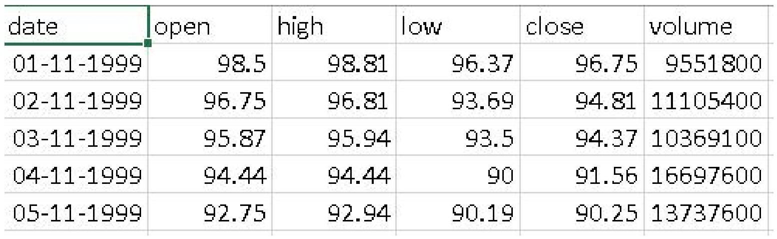
The first step in the workflow is to acquire real-time stock data from external sources through APIs. These APIs serve as bridges to live financial data and provide information such as stock prices, trading volumes, and market timestamps in structured formats like JSON or CSV. In this project, an API (such as Alpha Vantage, Yahoo Finance API, or Polygon.io) is accessed using scheduled Python scripts that continuously pull live data at fixed intervals. These scripts authenticate with the API using a unique API key, then send HTTP requests to retrieve stock metrics such as Open, High, Low, Close, and Volume.

To ensure the reliability of this data-fetching mechanism, error handling and retry logic are implemented. The retrieved data is then converted into a consistent format suitable for downstream processing. Since APIs are subject to rate limits and potential downtime, the system is designed to handle retries, alternate API sources, and basic caching mechanisms.



date	open	high	low	close	volume
19-08-2004	100.01	104.06	95.96	100.335	44659000
20-08-2004	101.01	109.08	100.5	108.31	22834300
23-08-2004	110.76	113.48	109.05	109.4	18256100
24-08-2004	111.24	111.6	103.57	104.87	15247300
25-08-2004	104.76	108	103.88	106	9188600

Figure 4.5: Alphabet Inc Class A



date	open	high	low	close	volume
01-11-1999	98.5	98.81	96.37	96.75	9551800
02-11-1999	96.75	96.81	93.69	94.81	11105400
03-11-1999	95.87	95.94	93.5	94.37	10369100
04-11-1999	94.44	94.44	90	91.56	16697600
05-11-1999	92.75	92.94	90.19	90.25	13737600

Figure 4.6: IBM Common Stock

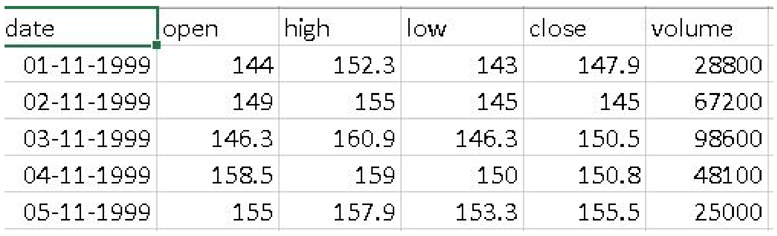
Key features:

- Fetches OHLCV data every few seconds or minutes.
- Handles API limits and errors gracefully.
- Produces clean, formatted JSON data for streaming.

4.3 DATA STREAMING USING APACHE KAFKA

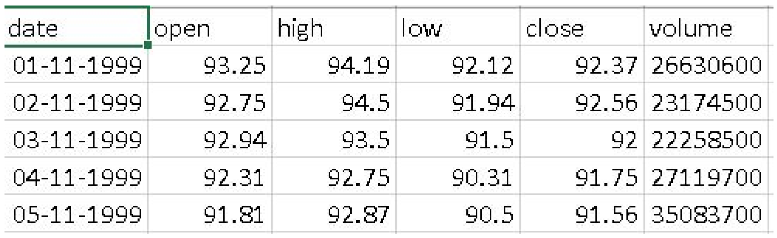
Once the real-time stock data is collected via the API, it is sent directly to Apache Kafka for streaming. Kafka is chosen for its ability to handle large streams of data with minimal latency and high fault tolerance. Kafka acts as a distributed messaging system that decouples the data producer (API script) from the downstream consumers (storage and processing layers).

In this setup, the Python API script acts as a Kafka producer, publishing stock price data to specific Kafka topics. Topics may be structured by company, sector, or timestamp. Kafka brokers manage and distribute the messages across partitions, ensuring durability and allowing multiple



date	open	high	low	close	volume
01-11-1999	144	152.3	143	147.9	28800
02-11-1999	149	155	145	145	67200
03-11-1999	146.3	160.9	146.3	150.5	98600
04-11-1999	158.5	159	150	150.8	48100
05-11-1999	155	157.9	153.3	155.5	25000

Figure 4.7: Infosys Ltd



date	open	high	low	close	volume
01-11-1999	93.25	94.19	92.12	92.37	26630600
02-11-1999	92.75	94.5	91.94	92.56	23174500
03-11-1999	92.94	93.5	91.5	92	22258500
04-11-1999	92.31	92.75	90.31	91.75	27119700
05-11-1999	91.81	92.87	90.5	91.56	35083700

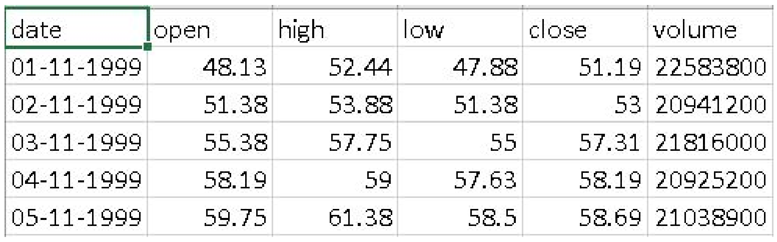
Figure 4.8: Microsoft Corp

consumers to access the data independently. This decoupled architecture enables horizontal scalability and fault tolerance, critical for real-time systems.

Consumers, such as the MongoDB ingestion pipeline or Spark stream processors, subscribe to these topics and read data asynchronously. This ensures that no data is lost, even if the consumer is temporarily offline or under heavy load. Kafka also enables data reprocessing, making it easier to replay past events if needed.

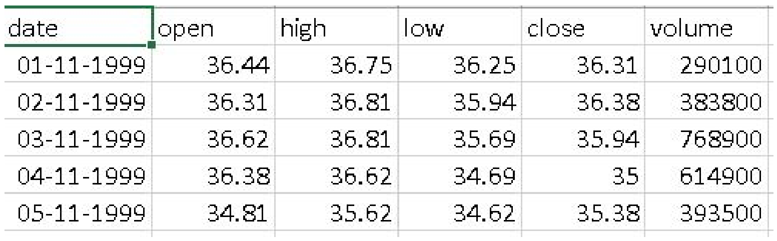
Core capabilities of Kafka:

- Low-latency streaming.
- High fault tolerance via replication.
- Scalable message queues for distributed systems.



date	open	high	low	close	volume
01-11-1999	48.13	52.44	47.88	51.19	22583800
02-11-1999	51.38	53.88	51.38	53	20941200
03-11-1999	55.38	57.75	55	57.31	21816000
04-11-1999	58.19	59	57.63	58.19	20925200
05-11-1999	59.75	61.38	58.5	58.69	21038900

Figure 4.9: Oracle Corp



date	open	high	low	close	volume
01-11-1999	36.44	36.75	36.25	36.31	290100
02-11-1999	36.31	36.81	35.94	36.38	383800
03-11-1999	36.62	36.81	35.69	35.94	768900
04-11-1999	36.38	36.62	34.69	35	614900
05-11-1999	34.81	35.62	34.62	35.38	393500

Figure 4.10: SAP SE

4.4 STORING REAL-TIME DATA WITH MONGODB

The next stage in the pipeline involves storing the streaming data efficiently in a database designed for handling high-velocity, time-series data. For this purpose, MongoDB is employed. As a NoSQL document-oriented database, MongoDB offers excellent support for dynamic schemas, hierarchical data structures, and efficient indexing of time-based entries.

Kafka consumers write the streamed data into MongoDB collections in real-time. Each stock entry is saved as a document containing the stock symbol, timestamp, and OHLCV values. MongoDB's capability to manage large volumes of unstructured and semi-structured data makes it a perfect fit for this application. Additionally, its built-in support for geospatial queries, time-based indexing, and aggregation pipelines facilitates fast querying and analysis of stored data.

The database schema is designed to ensure fast reads and writes. Time-based indexing on the timestamp field enables efficient lookups for time-series visualizations and model inputs. Collections are also periodi-

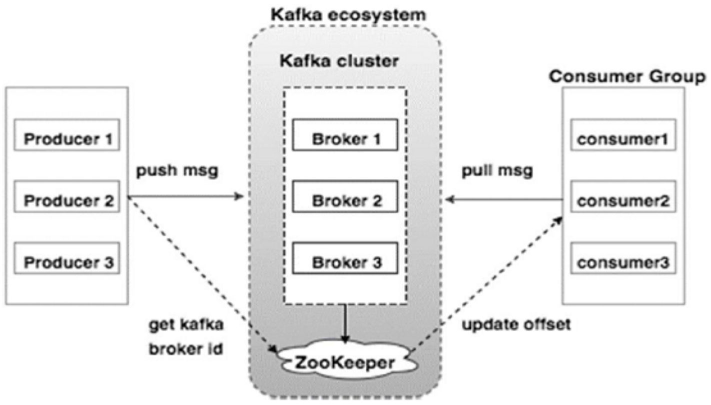


Figure 4.11: Kafka

cally archived or compressed to optimize storage.

Advantages of using MongoDB:

- Schemaless document storage.
- High write throughput.
- Native time-series handling.

4.5 STORING REAL-TIME DATA WITH MONGODB

After the data is streamed through Kafka, it is stored in MongoDB for persistent and structured storage. MongoDB, being a NoSQL, document-oriented database, is well-suited for storing large volumes of semi-structured, time-series data like stock prices. Kafka consumers are implemented to subscribe to the stock data stream and write each record into MongoDB collections in real-time.

Each document in MongoDB typically includes fields such as stock symbol, timestamp, Open, High, Low, Close, and Volume. MongoDB collections are structured in a way that optimizes time-series operations, with indexes on timestamps and symbols to enable fast querying. The schema-less nature of MongoDB allows the storage of dynamic data, and it supports fast inserts and updates which are essential for real-time analytics.

One of the strengths of MongoDB in this project is its ability to handle data ingestion at a high velocity without compromising read performance. MongoDB's aggregation pipelines are also useful for pre-processing and summarizing data before passing it to PySpark for deeper analysis. If required, MongoDB collections can be sharded for scalability or replicated for high availability.

Furthermore, MongoDB serves as a central data repository from which both the machine learning models and visualization tools can fetch information. This centralization simplifies data governance and helps maintain consistency across the pipeline.

4.6 BIG DATA PROCESSING AND FEATURE ENGINEERING WITH PYSPARK

After storing the raw data, the next phase involves processing and transforming it using PySpark, the Python API for Apache Spark. Spark is a distributed data processing framework well-suited for big data analytics. PySpark allows us to clean, transform, and extract features from stock data across distributed systems, ensuring high processing speed and efficiency.

The processing pipeline starts by reading data from MongoDB using the MongoSpark connector. The data is then cleaned to handle missing values, remove duplicates, and normalize formats. After cleaning, the pipeline computes technical indicators and statistical features required for machine learning. These features include daily returns, volatility, moving averages, volume changes, and lag features across different time windows.

PySpark's DataFrame and SQL capabilities make it easy to perform complex operations in a readable manner. More importantly, PySpark supports real-time structured streaming, which allows us to process data in near-real-time, thus ensuring that the machine learning predictions are based on the most recent data available.

Key transformations in PySpark:

- Time-window aggregations (e.g., 15-mins moving averages)
- Feature engineering (e.g., returns, volatility).
- Real-time data streaming and ETL.

4.7 MACHINE LEARNING MODEL

With the processed data ready, the next step is to apply machine learning models to extract insights and predict future stock price movements. The project uses historical and recent features generated by PySpark to train supervised learning models. These features include lagged price values, rolling statistics (such as moving averages and volatility), trading volume trends, and time-based indicators like day of the week and month. These enriched features aim to capture temporal dependencies and market patterns more effectively.

The core objective is to predict future values of the 'Close' price or the directional trend (up/down) based on past behavior. Depending on the requirement, the problem is approached as either a regression task (predicting numeric values) or a classification task (predicting price movement direction).

Several algorithms are experimented with, including Linear Regression, Random Forest Regressor, and advanced time-series models like ARIMA and LSTM (if deep learning is incorporated). LSTM networks are particularly useful for capturing long-term dependencies in sequential data and are implemented using Keras and TensorFlow when computational resources permit. ARIMA, on the other hand, is leveraged for stationary time series modeling, with hyperparameters optimized using grid search techniques.

The data is split into training and testing sets using time-aware validation, such as expanding window or walk-forward validation, to maintain chronological integrity and prevent data leakage. Random shuffling is strictly avoided to ensure realistic simulation of future predictions.

Key machine learning steps:

- Model training on historical data.
- Prediction of stock price movements.
- Evaluation using RMSE, MAE.

4.8 VISUALIZING PREDICTIONS AND MARKET TRENDS USING POWER BI

The final layer of the methodology is data visualization, which is accomplished using Microsoft Power BI. Power BI connects to MongoDB (through data exports or intermediate services) and visualizes both the raw market data and model-generated predictions. Dashboards in Power BI display time-series charts, candlestick patterns, trend lines, volume bar charts, and other interactive visuals that allow users to interpret market conditions effectively.

Real-time updates are enabled in Power BI dashboards through scheduled data refreshes, allowing them to incorporate the latest predictions and metrics. Users can apply filters by stock symbol, time range, or trend category to drill into specific segments of interest. A comparative overlay of predicted vs. actual stock prices helps evaluate the performance of the machine learning models over time.

Power BI also supports embedding analytics into web dashboards or mobile devices, which enhances accessibility and user engagement. In this project, Power BI acts as the bridge between technical analytics and business decision-making, translating complex data patterns into actionable insights.

4.9 SYSTEM ARCHITECTURE OVERVIEW

The system follows a modular, stream-based architecture where each component is independently deployed but collectively integrated to function as a real-time analytics pipeline. The workflow follows this order: API→ Kafka→ MongoDB→ PySpark→ Machine Learning→ Power BI

Each component communicates asynchronously, enabling fault-tolerant and scalable operation. Kafka handles streaming and messaging. MongoDB provides persistent storage with fast read/write capabilities. PySpark performs real-time analytics and feeds the machine learning models. Power BI consumes the outputs and offers intuitive visual dashboards for users.

This design allows for high performance, modular extensibility, and real-time feedback loops. It also enables the integration of additional components in the future such as alerting systems, portfolio tracking modules, or advanced anomaly detection engines.

4.10 CONCLUSION OF METHODOLOGY

To conclude, this methodology provides a comprehensive and real-time solution for stock price analysis. It demonstrates how multiple cutting-edge tools—each specializing in a part of the data lifecycle—can be integrated into a powerful analytics system. APIs ensure real-time data availability, Apache Kafka ensures robust and low-latency data streaming, MongoDB enables fast and flexible storage, PySpark handles large-scale data transformation, machine learning models generate predictive insights, and Power BI provides accessible visual interpretation.

Chapter 5

IMPLEMENTATION

The implementation of the Stock Price Analysis project involves a real-time data pipeline that leverages big data tools and machine learning models to predict stock price trends. The entire system is designed to handle large volumes of streaming data from stock APIs, preprocess and store it, apply machine learning models, and provide live predictions. The architecture integrates components like Apache Kafka, MongoDB, PySpark, and MLlib to build a scalable and efficient solution. The system is composed of several interconnected modules as described below:

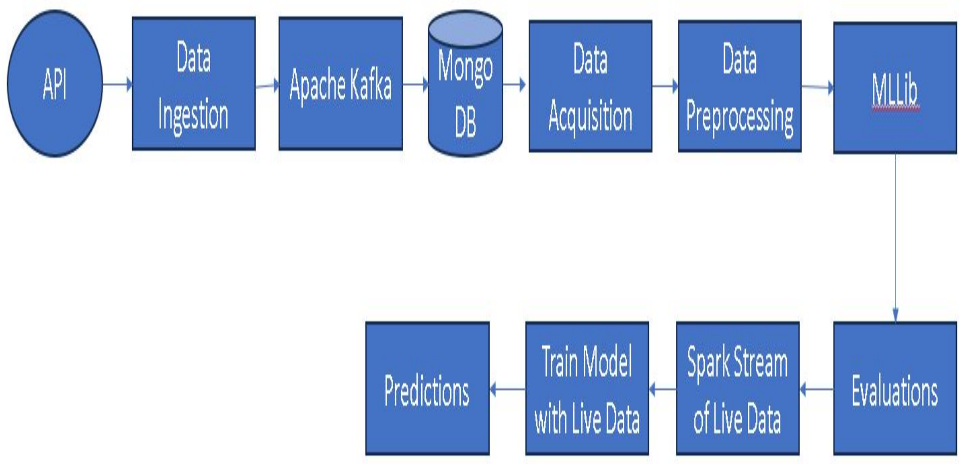


Figure 5.1: Flow Diagram

5.1 API

In the stock price analysis project, the API serves as the primary source of real-time market data, acting as the entry point for the entire data pipeline. It connects the system to external financial data providers, such as Alpha Vantage, Yahoo Finance, or other stock exchange APIs, which offer up-to-date information about stock prices, trading volumes, opening and closing values, and other market indicators. The API is designed to fetch data at regular intervals or in real-time, depending on the configuration, ensuring a continuous stream of fresh and accurate stock data. This data is typically returned in structured formats like JSON or CSV, which makes it easy to parse and integrate into downstream processing stages. The reliability and speed of the API are critical, as they directly affect the timeliness and accuracy of predictions. By automating the retrieval of stock data, the API eliminates the need for manual intervention and ensures that the system always has access to the most recent market information, forming the foundation for all subsequent data processing, analysis, and machine learning tasks in the project.

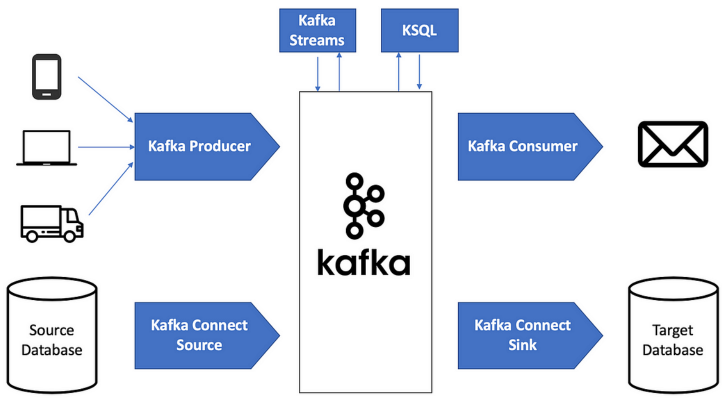


Figure 5.2: Collection of Data

5.2 DATA INGESTION

In a real-time stock price analysis system, the Data Ingestion layer acts as the critical gateway that enables the continuous flow of data from the external world into the analytics ecosystem. After the API fetches the latest stock information, the ingestion system picks up this data and ensures that it is properly transmitted to the internal processing tools without delay. This component is designed to handle both structured and semi-structured data formats and ensures that all incoming records are parsed correctly, checked for integrity, and forwarded in the correct schema. For example, it may convert raw JSON payloads into more readable and consistent formats that align with internal schema standards used across tools like MongoDB and Apache Spark.

One of the most important aspects of data ingestion is its scalability and resilience. Stock market data can arrive in bursts — especially during market opening or high volatility — so the ingestion module must be able to scale dynamically to handle peak loads without failure. Additionally, it must be fault-tolerant, ensuring that no data is lost in case of a system crash or API timeout. In most implementations, this is handled by buffering the data using tools like Kafka or writing intermediate files in a distributed file system temporarily. The ingestion layer also plays a key role in maintaining timestamp synchronization, ensuring that each data point is tagged with the correct date and time, which is crucial when performing time-series analysis or training predictive models.

5.3 APACHE KAFKA

Apache Kafka is a highly scalable, distributed event streaming platform that plays a central role in this project by enabling real-time data streaming and communication between various components of the pipeline.

In the context of stock price analysis, Apache Kafka acts as a reliable backbone for handling large volumes of stock market data that are being streamed continuously from the API. Stock markets generate data at very high speeds, and handling such rapid streams efficiently is crucial. Kafka solves this problem by acting as a publish-subscribe messaging system, which decouples data producers (API/Data Ingestion) from consumers (MongoDB, Spark Streaming, etc.).

- **Real-Time Data Flow:** Kafka ensures real-time and low-latency delivery of stock market data, which is essential for accurate and timely predictions.
- **Scalability:** As the number of stock tickers or users grows, Kafka can scale horizontally across multiple brokers and partitions, supporting more throughput without redesigning the architecture.
- **Fault Tolerance:** Kafka ensures durability and reliability by replicating data across multiple brokers. This means that even in the case of a server failure, the data stream is not lost.
- **Buffering Capability:** In case downstream systems like MongoDB or Spark are busy, Kafka can buffer incoming messages and replay them later without data loss, thus ensuring asynchronous and decoupled communication between components.

- **Multiple Consumers:**Kafka allows multiple independent consumers to subscribe to the same stream of stock data.
- **Stream Replay:** Kafka retains messages for a configurable retention period, allowing reprocessing of historical data by replaying streams. This feature is useful for retraining models or debugging issues in the pipeline.
- **Integration with Spark:** Kafka integrates seamlessly with Apache Spark Streaming, which allows real-time processing of the data stream for machine learning and forecasting tasks.

5.4 MONGODB

MongoDB is used as the primary NoSQL database in this project to store and manage the large volume of stock price data collected in real-time from various sources. It serves as a robust data repository that supports the rest of the pipeline, especially during data acquisition and preprocessing stages.

MongoDB is a document-oriented database, which stores data in flexible, JSON-like documents. This makes it a suitable choice for handling semi-structured and evolving datasets, such as stock market data that may vary slightly in structure over time due to different APIs or additional financial indicators.

- **RSchema Flexibility:** Since stock price data can include various fields (e.g., open, high, low, close, volume, ticker name, date, etc.), MongoDB's dynamic schema allows for seamless storage and easy modification without needing rigid table structures like in relational databases.

- **High Write Throughput:** MongoDB supports fast insert operations, which is crucial in this project as stock data is ingested at a rapid pace and stored continuously.

- **Efficient Querying:** MongoDB provides powerful indexing and querying capabilities, enabling quick retrieval of historical stock data for model training, evaluation, and visualization purposes.

- **Integration with Apache Kafka:** MongoDB integrates smoothly with Kafka consumers, allowing real-time ingestion of streamed data into the database. This creates a consistent and persistent data store for all the records processed by the system.

- **Support for Time-Series Data:** MongoDB supports time-series collections which are specifically optimized for storing and analyzing time-stamped data like stock prices. This makes MongoDB highly efficient for storing data across many years and multiple companies.

- **Horizontal Scalability:** MongoDB can scale horizontally across servers using sharding, which is essential when storing years of high-frequency trading data for multiple companies (such as Apple, Amazon, IBM, Infosys, etc.).

  In the project architecture, MongoDB sits immediately after Apache Kafka, acting as a persistent sink for all incoming stock price data. It stores the raw and preprocessed data, which is then fetched during data acquisition and model training stages. MongoDB's high performance and flexibility enable real-time updates, historical data access, and smooth integration with machine learning pipelines and visualization tools.

5.5 DATA ACQUISITION

Data Acquisition refers to the process of extracting relevant data from the storage system (MongoDB in this case) and preparing it for further processing and analysis. In a stock price analysis project, accurate and timely data acquisition is essential to ensure that the machine learning models are trained on up-to-date and relevant information.

In this project, the acquisition process involves querying MongoDB for historical and live data using filters such as stock ticker symbols (e.g., AAPL, AMZN, MSFT), specific time periods, and essential attributes like open, high, low, close, and volume. By setting proper conditions and indexes in MongoDB, the data can be fetched efficiently even when dealing with tens of thousands of records.

Additionally, the system supports incremental acquisition, which means it can periodically pull only the newly updated or appended data, reducing processing time and resource consumption. This is particularly useful in real-time systems where stock data is updated continuously and needs to be analyzed promptly.

Data acquisition also helps in organizing data from multiple sources into a consistent format. For example, when integrating datasets from different stock exchanges or APIs, this phase standardizes field names, date formats, and measurement units. The acquired data is then temporarily held in memory or staging areas before moving to the preprocessing phase.

5.6 DATA PREPROCESSING

The data preprocessing phase in this project is designed to prepare raw stock market data for machine learning and time-series analysis. The dataset includes daily trading information for multiple companies such as Apple, Amazon, Google, Infosys, and others. Each record contains important fields such as Date, Open, High, Low, Close, Adjusted Close, and Volume.

1. Date Conversion and Indexing The first step involves converting the Date column to a datetime format, which is essential for time-series operations. The dataset is then indexed by the Date field to ensure proper chronological order during analysis, modeling, and visualization.

2. Handling Missing Values The data is checked for any missing or null values. Missing entries, if found, are handled using forward-fill methods or are dropped to maintain consistency in the time-series sequence. Clean data ensures the integrity of indicators like MACD and moving averages.

3. Feature Selection Only relevant columns such as Open, High, Low, Close, and Volume are selected for analysis. These are the core financial indicators used to derive technical metrics and feed into predictive models.

4. Technical Indicator: MACD A major part of preprocessing includes the calculation of the MACD (Moving Average Convergence Divergence). This indicator is calculated by subtracting the 26-period EMA (Exponential Moving Average) from the 12-period EMA. A 9-period EMA of the MACD itself, called the signal line, is also computed.

- MACD = EMA(12) - EMA(26)
- Signal Line = EMA(9) of MACD

These indicators are used to capture momentum and trend direction. They are appended to the dataset as new features and are used later in training the machine learning models.

5. Normalization and Scaling (if applicable) Depending on the machine learning model used (e.g., MLPClassifier), scaling may be applied to numerical features such as volume and price to bring them into a consistent range. This prevents dominance of high-magnitude features and improves convergence speed.

6. Final Dataset Preparation After feature engineering, the final preprocessed dataset contains:

- Time-indexed daily trading data
- Derived MACD and Signal Line features
- Cleaned and filtered numerical columns
- Normalized values (where required)

5.7 MLLIB

The machine learning phase of this project is implemented using Apache Spark's MLlib, a powerful library for scalable machine learning. After preprocessing the stock data and calculating relevant technical indicators like MACD, the next step involves building and training classification models to predict future stock trends.

The main goal is to classify whether the stock price will increase, decrease, or remain stable based on historical price behavior and derived features. Spark MLlib is used for its ability to handle large datasets across distributed systems efficiently.

5.7.1 MLlib Model Training and Implementation

In this project, Apache Spark's `MLlib` library was utilized to train and evaluate machine learning models for stock price movement prediction. The models included Random Forest, Gradient-Boosted Trees, and Multilayer

Perceptron Classifier. All models were developed using a structured MLlib pipeline consisting of feature assembly, scaling, training, and evaluation phases.

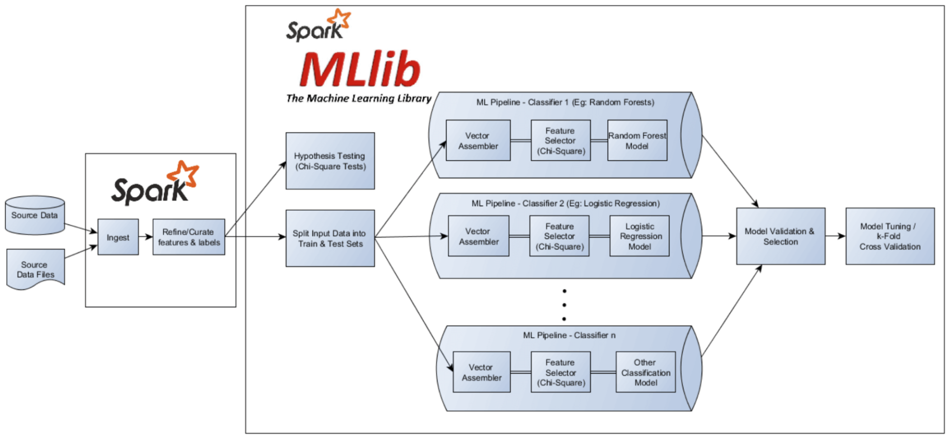

Source Data
Source Data Files
Spark
Ingest
Refine/Curate features & labels
Spark
MLlib
The Machine Learning Library
Hypothesis Testing (Chi-Square Tests)
Split Input Data into Train & Test Sets
ML Pipeline - Classifier 1 (Eg: Random Forests)
Vector Assembler
Feature Selector (Chi-Square)
Random Forest Model
ML Pipeline - Classifier 2 (Eg: Logistic Regression)
Vector Assembler
Feature Selector (Chi-Square)
Logistic Regression Model
ML Pipeline - Classifier n
Vector Assembler
Feature Selector (Chi-Square)
Other Classification Model
Model Validation & Selection
Model Tuning / k-Fold Cross Validation

Figure 5.3: SPARK Architecture

5.7.2 Feature Engineering and Vectorization

Prior to model training, relevant features were engineered from the historical stock data. These included:

- Price-based indicators (e.g., daily return, moving averages)
- Percentage changes
- Lagged closing prices

These features were combined into a single vector using `VectorAssembler`, making them suitable input for MLlib models. Additionally, `StandardScaler` was applied to normalize the features where required, especially for neural networks.

5.7.3 Long Short-Term Memory

In this project, we used an LSTM (Long Short-Term Memory) model, which is a special type of neural network that works really well with time-series or sequential data. Since stock prices change over time, LSTM was a good choice to try and capture the patterns in that data. Unlike models like Random Forest or Gradient Boosting, which treat each row of data separately, LSTM looks at the sequence of data and understands how previous time steps affect the current one.

LSTM has something called memory cells, and they help the model remember important information for a long time. It uses three gates—input, forget, and output gates—to decide what information to keep, what to ignore, and what to pass on. This makes LSTM better at handling long-term dependencies in the data, which is something regular RNNs struggle with.

As a implementation, I trained the LSTM on past stock values to see if it could learn the patterns and make good predictions. It processes the data one step at a time and updates its internal memory, which helps it learn things like trends, spikes, or drops in stock prices over time. That's why LSTM can be more effective when working with data where the order and time are important.

5.7.4 Random Forest Classifier

The `RandomForestClassifier` was implemented using PySpark's MLlib. It is an ensemble model that creates multiple decision trees on random subsets of the data and aggregates their outputs to determine the final classification.

- **Training:** The model was trained using the prepared feature vector on the training set split using `randomSplit()`.

- **Parameters:** Key parameters included `numTrees` for the number of trees and `maxDepth` to control overfitting.
- **Use Case:** This model helped predict binary price direction labels (up/down) with robustness to noisy data.

5.7.5 Gradient-Boosted Trees Classifier (GBTClassifier)

The `GBTClassifier` is a boosting ensemble method that sequentially builds decision trees, each one correcting errors from the previous trees.

- **Training:** The GBT model was trained on the same feature vector used for Random Forest.
- **Parameters:** The `maxIter` parameter controlled the number of iterations, and `maxDepth` was used to manage model complexity.
- **Use Case:** The model offered high prediction accuracy and was used for its ability to capture subtle trends in stock movement.

5.7.6 Multilayer Perceptron Classifier (MLP)

The `MultilayerPerceptronClassifier` represents a neural network with multiple layers that captures complex non-linear patterns in the data.

- **Training:** Input features were scaled using `StandardScaler` before being fed into the MLP to ensure proper convergence.
- **Architecture:** The layer configuration (e.g., `[inputSize, hidden1, hidden2, outputSize]`) was defined based on the number of input features and binary output labels.
- **Use Case:** MLP was applied to model deeper hidden relationships in financial time series data that tree-based models may overlook.

5.7.7 Model Evaluation

All models were evaluated using PySpark's `MulticlassClassificationEvaluator`, based on metrics such as accuracy, precision, recall, and F1-score. Confusion matrices were also generated for performance comparison.

5.7.8 Conclusion

The MLlib framework provided a scalable and efficient environment to train and evaluate multiple machine learning models. By integrating Random Forest, Gradient-Boosted Trees, and Multilayer Perceptron within the MLlib pipeline, the project demonstrated robust predictive modeling for stock price direction classification.

5.8 EVALUATION PHASE

The evaluation phase is crucial to understanding how well the trained models perform on unseen data. In this project, evaluation was conducted using the testing dataset that was separated from the original data using an 80:20 split. This phase involved generating predictions, comparing them with actual labels, and calculating various performance metrics.

5.8.1 Prediction and Testing

Each trained model—Random Forest, Gradient-Boosted Trees, and Multilayer Perceptron—was applied to the test dataset to generate predictions. These predictions were then compared to the actual labels to assess model accuracy and reliability.

5.8.2 Evaluation Metrics

To quantify the model performance, the following metrics were computed using `MulticlassClassificationEvaluator` in MLlib:

- **Accuracy:** The ratio of correctly predicted instances to the total number of predictions. It provides an overall measure of the model's performance.
- **Precision:** Measures the proportion of correctly predicted positive observations to the total predicted positives. High precision indicates low false positive rate.
- **Recall:** Also known as sensitivity, it is the ratio of correctly predicted positive observations to all actual positives. High recall indicates low false negative rate.
- **F1-Score:** The harmonic mean of precision and recall. It balances both metrics and is especially useful when class distribution is imbalanced.

5.8.3 Model Comparison

The evaluation results were compared across the three models:

- **Random Forest:** Provided balanced performance with good accuracy and low variance.
- **GBTClassifier:** Delivered the highest accuracy among the models, capturing complex patterns well due to its boosting nature.
- **MLP Classifier:** Showed strong learning on non-linear relationships, though its performance depended heavily on proper scaling and network architecture.

5.8.4 Summary

The evaluation phase highlighted the strengths and limitations of each machine learning model. Gradient-Boosted Trees emerged as the most accurate classifier for predicting stock price direction, while Random Forest and MLP also performed reasonably well. These insights are critical for selecting the most suitable model for real-time stock prediction applications.

5.9 SPARK STREAMING OF LIVE DATA

To extend the predictive analytics capabilities to real-time applications, this project integrates **Spark Structured Streaming** for handling live stock market data. Spark Streaming allows scalable and fault-tolerant processing of continuous data streams in near real-time.

5.9.1 Streaming Pipeline

The streaming pipeline consists of the following components:

- **Data Source:** A simulated or real-time data stream, such as a Kafka topic or socket, is used to ingest stock price data continuously.
- **Spark Structured Streaming:** Processes the input stream using the same feature engineering and model prediction pipeline defined in MLlib.
- **Model Application:** The pre-trained machine learning model (e.g., GBTClassifier or Random Forest) is loaded and applied to the streaming data in mini-batches.
- **Output Sink:** The predictions are written to a console, dashboard, or external sink like Kafka or a database.

5.9.2 Implementation Details

The implementation uses PySpark's `readStream()` function to receive incoming data and `writeStream()` to output the results. Key steps include:

- Reading JSON/CSV format live data using:
- Preprocessing and feature transformation are applied similarly to the static data pipeline using `VectorAssembler` and `StandardScalerModel` (fitted on training data).
- The trained classification model is loaded and used to generate predictions on each micro-batch.
- Writing predictions to console or file sink:

```
predictions.writeStream.format("console").start()
```

5.9.3 Use in Stock Prediction

By integrating Spark Streaming, the project enables real-time decision-making based on live stock data. This is particularly useful for:

- Monitoring live trends and market signals.
- Triggering alerts or trading actions based on model predictions.
- Continuously adapting to changing market conditions.

5.9.4 Scalability and Fault Tolerance

Spark Structured Streaming provides scalability across distributed clusters and ensures high availability through checkpointing and write-ahead logs. This makes the system robust for processing large-scale financial data in real time.

5.9.5 Summary

The use of Spark Streaming transforms the stock prediction model from a static analytical tool into a real-time intelligence system. By applying MLlib models on streaming data, the project demonstrates how big data technologies can empower live financial forecasting and automated trading systems.

5.10 TRAINING THE MODEL WITH LIVE DATA

While traditional machine learning models are trained on static datasets, this project explores the potential of training models using live streaming data through Apache Spark Structured Streaming. This approach is essential for building adaptive systems that learn from continuously evolving stock market data.

5.10.1 Streaming-Based Model Training Approach

To enable training on live data, the streaming source is connected to the MLlib training pipeline. The data is ingested as micro-batches, and each batch is treated as a mini training set.

- **Data Ingestion:** Streaming stock data is read from a source such as Kafka, socket, or file stream using Spark's `readStream()`.
- **Feature Transformation:** For each incoming batch, preprocessing is applied including missing value handling, feature generation, `VectorAssembler`, and `StandardScaler`.
- **Incremental Training:** Each mini-batch can be used to retrain the model, either by:

    - Appending the new batch to a rolling window of historical data.
    - Using algorithms that support incremental learning (e.g., online gradient descent).

- **Model Management:** The updated model is saved after each retraining cycle using `model.write().overwrite().save()` for future use in predictions.

5.10.2 Limitations in Spark MLlib

While Spark MLlib provides powerful tools for batch training and streaming inference, it currently does not support true **incremental model training** natively (i.e., continuing from an existing model without full retraining). To handle this:

- A retraining strategy was used where each micro-batch was appended to the training dataset and the model was retrained periodically.
- This retraining was scheduled in intervals (e.g., every N batches or every T minutes) depending on data volume and compute resources.

5.10.3 Use Case for Live Model Training

Training on live data allows the model to adapt to new market conditions and patterns as they emerge. This is particularly useful in:

- Detecting regime shifts or anomalies in stock behavior.
- Fine-tuning model parameters based on the latest trends.
- Enhancing robustness and responsiveness in algorithmic trading systems.

5.10.4 Summary

Training the model on live streaming data is an advanced technique that adds adaptability to machine learning systems. Although Spark MLlib has some limitations in incremental learning, strategies like frequent batch retraining and window-based data accumulation were implemented to simulate real-time learning. This ensures that the model remains up to date with the latest stock market dynamics.

5.11 PREDICTIONS

In the final stage of the project, predictions were generated for future stock prices using the historical data that had been processed and analyzed. After training the model on past data, it was applied to forecast stock prices during the test period. These predicted values were then compared to the actual closing prices to evaluate the model's performance.

To make the predictions directly comparable to real stock prices, the values were inverse-transformed back to their original scale. A plot was created to visualize both the actual and predicted prices over time. The predicted curve closely followed the general trend of the actual stock prices, showing that the model was able to capture overall market movement effectively.

This prediction phase demonstrated how historical stock data can be used to estimate future prices, offering valuable insights into potential trends and supporting data-driven decision-making.

Chapter 6

SOFTWARE TOOLS USED

6.1 JUPYTER NOTEBOOK

Jupyter Notebook is an interactive, open-source web application that supports multiple programming languages, most notably Python. It allows users to create and share documents that contain live code, equations, visualizations, and narrative text. This combination of code and documentation in one environment makes it a powerful tool for both learning and real-world application development.

In the context of our project, Jupyter Notebook served as the primary development environment. It provided a user-friendly interface for writing and testing Python code, which was essential for data preprocessing, model training, and integrating various components such as Apache Kafka, MongoDB, and PySpark. Through its rich text features and visual output capabilities, we could visualize data streams, monitor model predictions, and debug code more effectively.

Moreover, Jupyter Notebook facilitated seamless collaboration and documentation, enabling team members to understand and reproduce each other's work with ease. Its integration with real-time data pipelines helped us analyze incoming data dynamically, test transformations on-the-fly, and display analytics in an understandable format for stakeholders.

Overall, Jupyter Notebook played a crucial role in making the development process more interactive, efficient, and transparent.

6.2 APACHE KAFKA

Apache Kafka is a powerful open-source distributed event streaming platform developed by the Apache Software Foundation. It is designed to publish, subscribe to, store, and process streams of records in real time. Kafka is widely used for building real-time data pipelines and streaming applications that need to reliably process large volumes of data with minimal latency.

At its core, Kafka works based on the producer-consumer model. Data is sent by producers to Kafka topics, which act like channels. These messages are then consumed by consumers, which can process the data, store it, or forward it to other systems. Kafka stores data in a fault-tolerant and durable way, making it reliable for enterprise-level data operations.

Kafka's architecture is highly scalable and can handle millions of messages per second. It achieves high availability and fault tolerance by replicating data across multiple brokers in a Kafka cluster.

6.3 MONGODB

MongoDB's flexibility and scalability make it a robust choice for modern data-driven applications, particularly in environments where data is dynamic, unstructured, or semi-structured. Unlike traditional relational databases that rely on a rigid schema of tables and columns, MongoDB stores data in BSON (Binary JSON) format. This document-oriented structure is not only easier to scale but also allows for more flexibility in how data is stored and queried. The key advantage of this approach lies in its ability to handle complex and varied data types, making MongoDB a preferred solution for big data applications, particularly those that involve fast-growing, high-volume, or real-time datasets.

The core of MongoDB's design revolves around collections, which are analogous to tables in relational databases but do not require a fixed schema. Each document within a collection can have its own unique structure, allowing different records to have different fields. This schema-less design is particularly advantageous in big data scenarios where the data schema may evolve over time or where the data sources are inconsistent. For example, in a real-time data pipeline, the incoming data from different sources (such as Kafka streams or IoT devices) may not adhere to a uniform structure, and MongoDB's flexibility allows the system to efficiently ingest, store, and manage this data without needing upfront schema definitions or costly database schema migrations.

In the context of this project, MongoDB serves as a central repository for both raw and processed data. When data streams in from Kafka, MongoDB acts as the persistent layer, allowing the system to capture real-time data and store it in a highly scalable manner. This enables the pipeline to continuously ingest and store incoming events, whether they are stock prices, sensor readings, or social media updates, without delays. The ability to handle high-throughput data, combined with MongoDB's low-latency read and write operations, ensures that the system can keep up with the demands of real-time data processing.

Once the data is ingested, PySpark performs transformations such as data cleaning, filtering, and feature extraction, preparing the data for further analysis or machine learning. After these operations, the transformed data is stored back in MongoDB for future retrieval. MongoDB's ability to handle complex data structures, such as arrays, nested objects, or time-series data, allows it to store the output of these transformations efficiently, even as the dataset grows over time.

MongoDB's fast read/write capabilities are another key feature that makes it ideal for this project. In scenarios where the data pipeline is expected to process large datasets in real time, the database must support quick insertion and retrieval of data without causing performance bottlenecks. MongoDB's indexing mechanisms further enhance query performance, ensuring that data can be accessed quickly even as the volume increases. This is particularly important when the application needs to generate insights in real time, such as predicting stock prices or analyzing trends.

Furthermore, MongoDB's ability to integrate seamlessly with data analytics and visualization tools, such as Power BI, plays a crucial role in this project. Power BI, a powerful business intelligence tool, can connect to MongoDB to retrieve and visualize the data stored in the database. This enables end-users to interactively explore the data, perform ad-hoc queries, and generate reports or dashboards that reflect the latest insights. The integration between MongoDB and Power BI also supports the creation of real-time data visualizations, which is essential for applications like monitoring market trends or assessing real-time risk in financial systems.

Additionally, MongoDB's horizontal scalability enables it to handle vast amounts of data as the application grows. As data volumes increase, MongoDB can scale out by distributing data across multiple servers or clusters, ensuring that performance remains optimal even as the dataset expands. This makes MongoDB a highly suitable choice for projects that anticipate significant growth in data volume over time.

6.4 PYSPARK

PySpark, as the Python API for Apache Spark, enables developers to access the full potential of Spark's distributed computing capabilities while benefiting from Python's user-friendly syntax and extensive libraries. Apache Spark itself is a unified analytics engine designed to process large-scale data sets in a parallel and distributed manner, allowing it to handle complex data processing tasks efficiently. By leveraging Spark's in-memory computing and fault-tolerant processing, PySpark can rapidly process and analyze massive datasets across multiple nodes in a cluster, which significantly improves the performance of big data analytics.

In this project, PySpark plays a pivotal role as the central engine for a series of key operations, particularly in the areas of data transformation, aggregation, and machine learning tasks. As the data streams from Kafka, PySpark continuously consumes this real-time data, ensuring that no time is wasted in processing new information. This real-time data flow allows the system to handle dynamic and high-throughput data, such as stock prices, sensor readings, or market trends, which can be fed into the pipeline without delays.

The initial step in PySpark involves the filtering and cleaning of incoming data. Given that raw data often contains noise or irrelevant information, PySpark ensures that only the most accurate and pertinent data is retained. This is done through operations such as data cleaning, missing value imputation, and normalization, all of which are crucial to ensure that downstream analytics and machine learning models receive clean, structured data.

Feature extraction, another important aspect of the project, involves transforming raw data into meaningful insights. PySpark leverages its powerful DataFrame API to process large datasets and derive features that can be used by machine learning algorithms. For instance, if the dataset consists of time-series data, PySpark can aggregate the data over different time windows, calculate moving averages, or generate statistical features such as volatility or trends.

Once the data has been transformed, the results are either stored in a persistent database such as MongoDB or sent to external systems like Power BI for visualization. MongoDB, being a NoSQL database, efficiently stores large volumes of unstructured or semi-structured data, which fits the nature of the data processed in this project. The integration between PySpark and MongoDB allows seamless storage and retrieval of data, enabling a robust, scalable data pipeline for continuous updates.

The combination of Spark's distributed computing architecture and PySpark's Pythonic interface ensures that the data pipeline is highly scalable, fault-tolerant, and capable of processing large volumes of real-time data. PySpark's built-in libraries, such as MLlib for machine learning, facilitate the implementation of advanced analytics and predictive models, making it an ideal choice for developing scalable, real-time data pipelines. Furthermore, by integrating PySpark with Kafka for real-time data ingestion and visualization platforms like Power BI for reporting, the system can continuously update and provide actionable insights, which is critical for decision-making in applications like stock price prediction or market trend analysis.

Overall, PySpark's flexibility, scalability, and performance make it an indispensable tool in the development of big data solutions, particularly in scenarios where real-time data processing, aggregation, and machine learning are required.

Chapter 7

RESULTS & DISCUSSION

7.1 RANDOM FOREST CLASSIFIER PERFORMANCE

Random Forest performed quite well in terms of classification metrics like accuracy, precision, and recall. The high precision (0.83) indicates that when the model predicts a positive class, it is correct most of the time, implying a low false positive rate. This is especially important in applications where false alarms are costly.

The recall (0.76) suggests the model is also good at identifying actual positive cases, meaning it misses relatively few true positives. The F1-score (0.72) reflects a good balance between precision and recall.

However, the AUC (0.56) is low. AUC (Area Under the ROC Curve) measures the model's ability to differentiate between classes across all thresholds. A score closer to 1 is ideal, and 0.5 is as good as random guessing. The low AUC implies the model might be overfitting on certain features or the class distribution might be skewed, which can cause the ROC performance to degrade.

7.2 GBT PERFORMANCE

GBT performs identically to Random Forest in all metrics except AUC, which is slightly worse at 0.53. Like Random Forest, GBT gives high scores in accuracy, precision, recall, and F1-score, which suggests that it's effectively learning patterns from the data and classifying correctly based on a selected threshold.

However, the very low AUC (0.53) reveals that it may not generalize well or may be biased toward the majority class. Despite the good performance at a fixed threshold, the GBT struggles to differentiate between positive and negative classes when varying the classification threshold, limiting its utility in scenarios that demand fine-grained ranking or probability outputs.

7.3 MULTI LAYER PERCEPTRON PERFORMANCE

The MLP classifier has the highest AUC (0.8030) among the three models. This means it's better at ranking positive examples higher than negative ones. In other words, it captures the underlying structure of the data more effectively in terms of class separability.

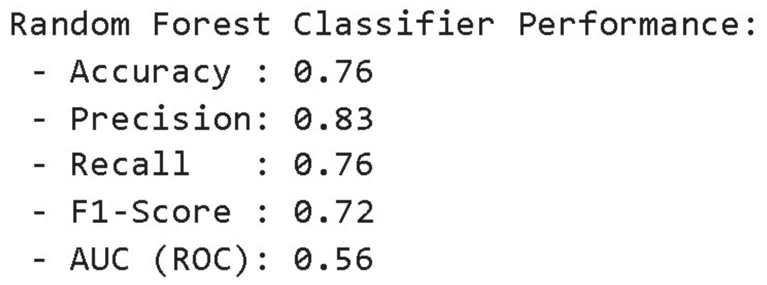
However, its precision (0.4187) and F1-score (0.5084) are significantly lower. This implies a high number of false positives, which could be problematic in certain domains. The accuracy (0.6471) and recall (0.6471) are lower than the other two models, indicating that while MLP has potential, it may require further tuning, more data, or better feature engineering to perform competitively in this context.

MLPs are powerful but sensitive to hyperparameters, input scaling, and data imbalance. The poor precision may result from such issues.

7.4 LONG SHORT-TERM MEMORY

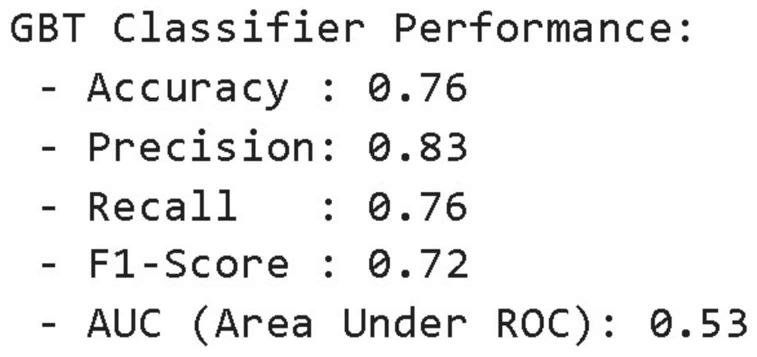
The RMSE (Root Mean Squared Error) values for all stock tickers give us a clear picture of how accurate the model's predictions were. A lower

RMSE means the model's predicted stock prices were close to the actual values, while a higher RMSE shows more error. From the results, it's clear that the model performed best on stocks like INFY (4.11), SAP (4.24), and IBM (4.27), where the predictions were quite accurate. It also gave decent results for AMD (6.29), ORCL (8.31), and MSFT (8.78). However, for stocks like AAPL (13.84) and ACN (9.03), the RMSE was a bit higher, meaning the model's predictions were less accurate but still manageable. The biggest errors were seen in GOOGL (100.58) and AMZN (127.52), showing that the model struggled the most with these stocks. This could be because they are more volatile or have large price swings, making them harder to predict. Overall, the model worked well for most stocks, but there's still room for improvement—especially with high-priced or highly volatile ones.



```
Random Forest Classifier Performance:
 - Accuracy : 0.76
 - Precision: 0.83
 - Recall   : 0.76
 - F1-Score : 0.72
 - AUC (ROC): 0.56
```

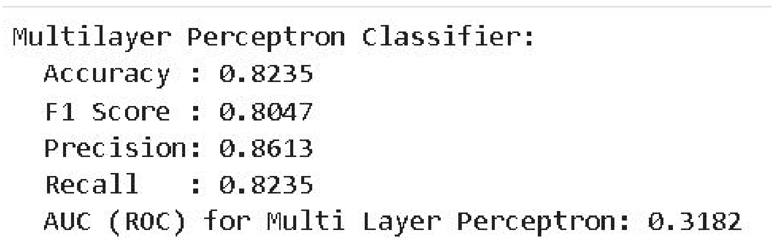
Figure 7.1: Random Forest Classifier



```
GBT Classifier Performance:
 - Accuracy : 0.76
 - Precision: 0.83
 - Recall   : 0.76
 - F1-Score : 0.72
 - AUC (Area Under ROC): 0.53
```

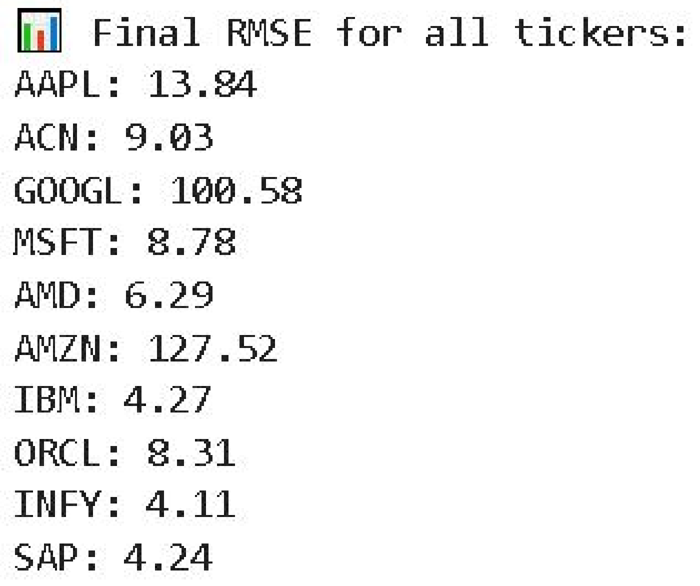
Figure 7.2: GBT Classifier

The final predictions were visualized by comparing actual and pre-



```
Multilayer Perceptron Classifier:
  Accuracy : 0.8235
  F1 Score : 0.8047
  Precision: 0.8613
  Recall   : 0.8235
  AUC (ROC) for Multi Layer Perceptron: 0.3182
```

Figure 7.3: Multi Layer Perceptron



```
📊 Final RMSE for all tickers:
AAPL: 13.84
ACN: 9.03
GOOGL: 100.58
MSFT: 8.78
AMD: 6.29
AMZN: 127.52
IBM: 4.27
ORCL: 8.31
INFY: 4.11
SAP: 4.24
```

Figure 7.4: LSTM

dicted stock prices for multiple companies. These results are presented through line plots, with performance measured using the Root Mean Squared Error (RMSE) for each company.

The LSTM model demonstrated strong predictive capability for most stocks. In particular, companies like IBM, SAP, AMD, and INFY showed very close alignment between actual and predicted prices, reflected in their low RMSE values (e.g., IBM: 4.27, SAP: 4.20, INFY: 4.11). These indicate high model accuracy and stability in forecasting these stock trends.

On the other hand, stocks such as AMZN and GOOGL yielded relatively higher RMSE values (127.52 and 100.58 respectively), suggesting greater volatility or possible data-related challenges, such as sharp price drops that were harder for the model to predict accurately.

Across all the tested stocks, the predicted curves followed the general direction of the actual prices, demonstrating the model's effectiveness in capturing trends. These results highlight the LSTM model's capability in forecasting time series data, especially when trends are stable and consistent.

The visual plots provided for each company further support the performance of the model, clearly showing where predictions succeeded and where deviations occurred.

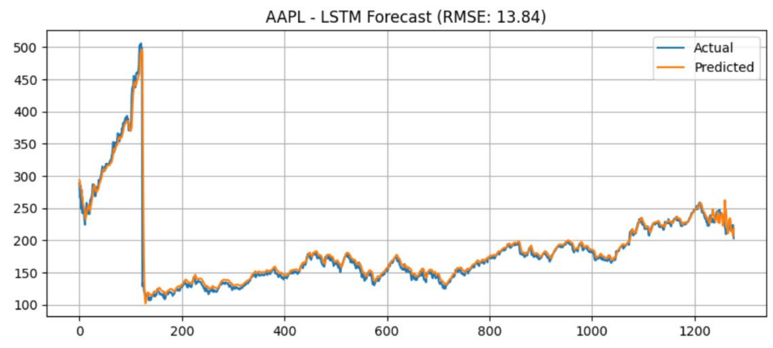


Figure 7.5: AAPL

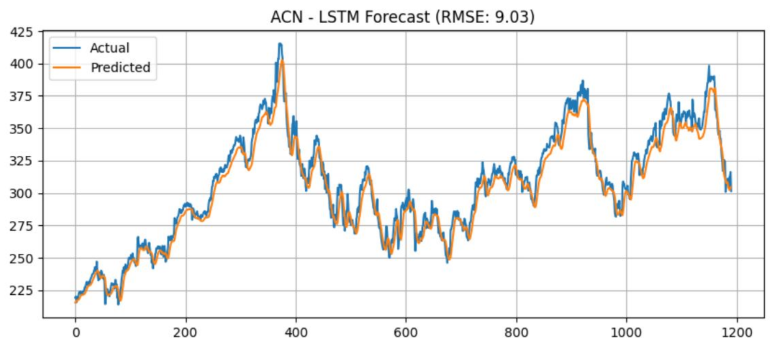


Figure 7.6: ACN

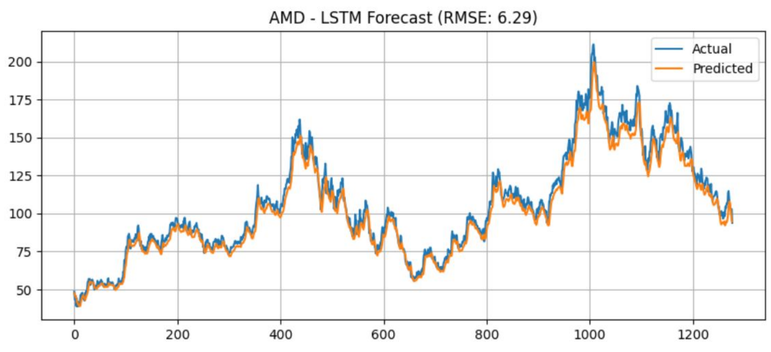


Figure 7.7: AMD

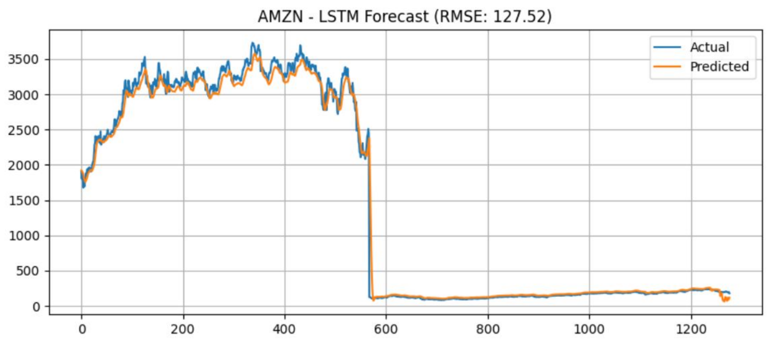


Figure 7.8: AMZN

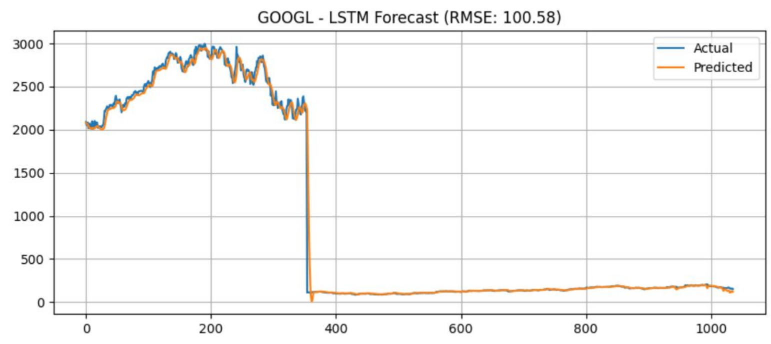


Figure 7.9: GOOGL

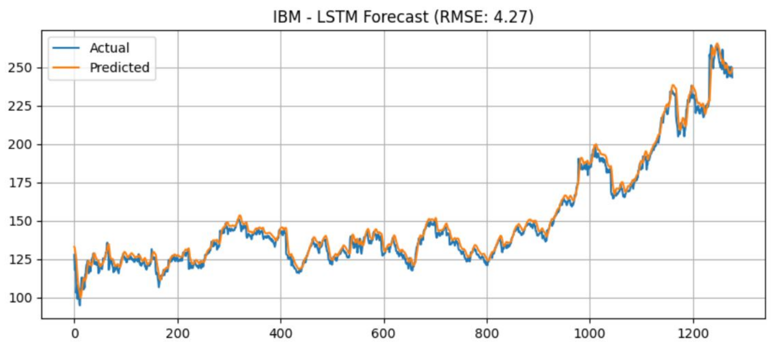


Figure 7.10: IBM

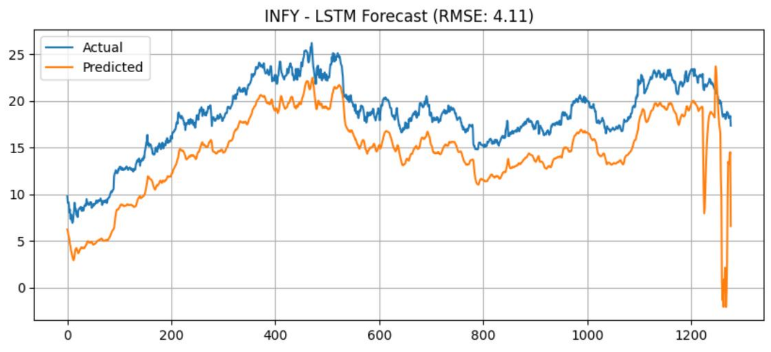


Figure 7.11: INFY

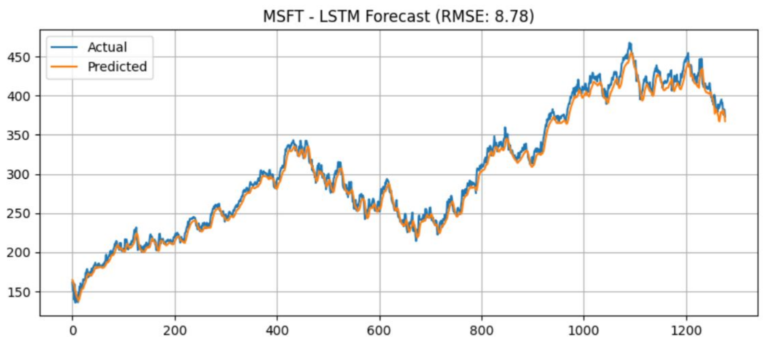


Figure 7.12: MSFT

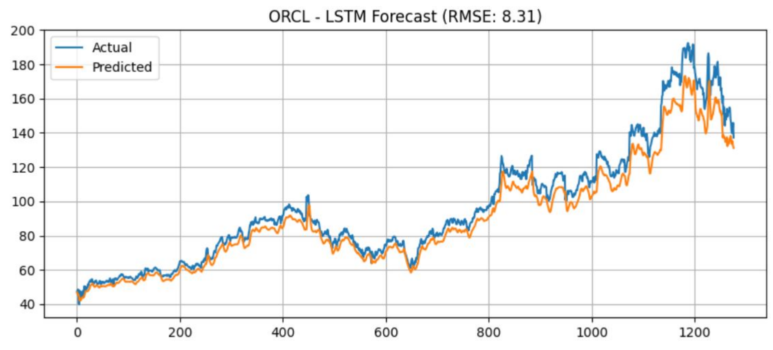


Figure 7.13: ORCL

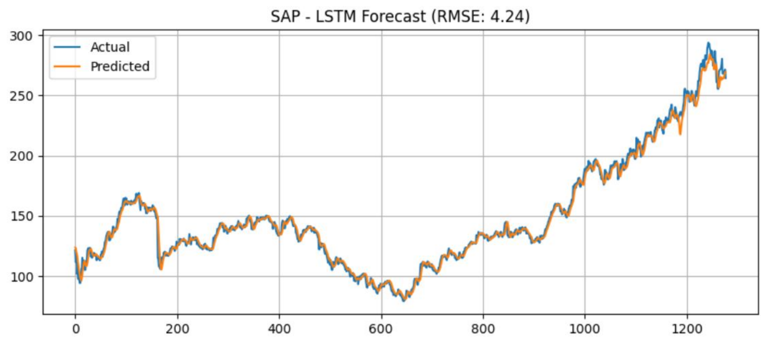


Figure 7.14: SAP

Chapter 8

CONCLUSION

In this project, We worked on stock price analysis using different machine learning models like Random Forest, Gradient Boosted Trees (GBT), Multilayer Perceptron (MLP), and LSTM. The goal was to predict or classify stock movement based on historical data and see which model performs the best.

From the results, Random Forest and GBT gave the highest accuracy, precision, and recall values, which means they performed well in making correct predictions. However, their AUC scores were low, so they weren't great at distinguishing between classes across all thresholds. On the other hand, MLP had a lower accuracy and F1 score, but its AUC was much better, showing it was good at separating the classes in general, even if the exact predictions weren't always correct. LSTM was used to handle the time-series nature of the data, and it was helpful in learning patterns from previous stock values.

Overall, this project helped me understand how different models behave with stock data, and how important it is to choose the right model depending on the problem. For future improvements, I could try more data preprocessing, tuning the models, or even combining them for better results.

8.1 SCOPE OF FUTURE WORK

8.1.1 What is future direction in a project?

There’s still a lot of potential to improve and build on this project. One key area is feature engineering—I can add more technical indicators, include news sentiment analysis, or even bring in economic data to make the predictions more accurate. Also, tuning the hyperparameters for models like MLP and LSTM could boost their performance, since the current setup uses mostly default settings. Another direction is trying out more advanced models, like Bidirectional LSTM or Transformers, which are known to work well with time-series data. I could also experiment with ensemble methods by combining different models to get more stable predictions. Finally, connecting the model to a real-time stock data feed and building a dashboard for live forecasting would be an exciting upgrade for this project.

REFERENCES

[1] **S.-T. Nam, C.-Y. Jin, and S.-Y. Shin**, "A forecasting of stock trading price using time series information based on big data", in *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 3, pp. 2548–2554, Jun. 2021.

[2] **Y. Ayyappa and A. P. S. Kumar**, "Stock market prediction with political data Analysis (SP-PDA) model for handling big data," Multimedia Tools and Applications, vol. 83, pp. 80583–80611, 2024, doi: 10.1007/s11042-024-18610-4.

[3] **Z. Peng**, "Stocks analysis and prediction using big data analytics", *in Proc. 2019 Int. Conf. on Intelligent Transportation, Big Data Smart City (ICITBS), Dalian, China,*, 2019, pp. 309–312. doi: 10.1109/ICITBS.2019.00081.

[4] **G. Sismanoglu, F. Kocer, M. A. Onde, and O. K. Sahingoz**, "Deep learning based forecasting in stock market with big data analytics", in Proc. 2019 Int. Conf. on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), Ankara, Turkey, 2019, pp. 113–116. doi: 10.1109/IBIGDELFT.2018.8625278

[5] **U. Kekevî and A. A. Aydin**, "Real-Time Big Data Processing and Analytics: Concepts, Technologies, and Domains," Journal of Computer Science, vol. 7, no. 2, pp. 111–123, 2022. doi: 10.53070/bbd.1204112.

[6] **AS. T. Nam, C. Y. Jin, and S. Y. Shin** (1972) “A forecasting of stock trading price using time series information based on big data,” International Journal of Electrical and Computer Engineering (IJECE), vol. 11, no. 3, pp. 2548–2554, Jun. 2021. doi: 10.11591/ijece.v11i3.pp2548-2554.

[7] **Chintan Vora, Bhavya Shah, Dhairya Sheth, and Nasim Banu Shah** (2021) “Stock price analysis and prediction,” IEEE International Conference on Communication Information and Computing Technology (ICCICT).

[8] **Pronab Sen** (1974) “Stock price level: A predictive analysis,” *Economic and Political Weekly*, vol. 9, no. 48, pp. Nov. 1974.