Final Capstone in Data Science Report

Project Title: Stock Prediction and Portfolio Optimization App

Student Name: David Heller

Student ID: 6192583

Mentor: Professor Florence George

Course Instructor: Professor Ananda Mondal

Semester: Fall 2024

ABSTRACT

The stock market's volatility and the complexity of portfolio management pose significant challenges for individual investors. This project addresses these issues by developing a user-friendly application that integrates advanced data science techniques to empower investors with actionable insights. The app features stock exploration, LSTM-based price predictions, and Monte Carlo simulations for portfolio optimization.

The project is structured around four components: (1) Data Collection and Feature Engineering, where technical indicators such as Moving Averages and MACD were engineered, although not incorporated into the final app; (2) Machine Learning Models, where LSTM outperformed ARIMA and XGBoost in capturing time-series patterns for stock predictions; (3) Monte Carlo Simulations, which optimize portfolio allocations by maximizing the Sharpe Ratio; and (4) App Development, implemented in Streamlit and deployed to Streamlit Cloud with pre-trained models hosted on GitHub.

The final app bridges the gap between complex financial analytics and accessibility, enabling users to analyze trends, forecast stock prices, and optimize portfolios. While the app currently supports a limited number of stocks for prediction, its modular design provides a foundation for future scalability, including expanded stock coverage and real-time data integration. This project highlights the potential of data science to address real-world financial challenges and enhance decision-making for individual investors.

DELIVERABLES

[Stock Prediction and Portfolio Optimization App Link](https://stock-forecast-and-allocate.streamlit.app/)

[GitHub Repository Link](https://github.com/davidhellerw/DS-Capstone/tree/main)

TABLE OF CONTENTS

1. Introduction

1.1 Challenges in Stock Market Investing

1.2 Project Aim and Proposed Solution

1.3 Goals and Objectives

1. 4 Why This Project?

1.5 Problem Statement

1.6 Research Questions

1.7 Project Description

1.8 Expected Outcomes

2. Data Collection and Feature Engineering

2.1 Overview

2.2 Why These Stocks?

2.3 Data Collection

2.4 Introduction to Feature Engineering

2.5 Features Engineered

2.5.1 Lag features

2.5.2 Simple Moving Averages

2.5.2 Exponential Moving Averages

2.5.4 Bollinger Bands

2.5.5 Moving Average Convergence Divergence (MACD)

2.5.6 Standard Deviation of Previous 5 Days Returns

2.5.7 Average True Range (ATR)

2.5.8 Average Directional Index (ADX)

2.5.9 Commodity Channel Index (CCI)

2.5.10 Rate of Change (ROC)

2.5.11 Relative Strength Index (RSI)

2.5.12 William's %R

2.5.13 Stochastic %K

2.5.14 Force Index (FI)

2.6 Market Conditions: Feature Engineering with Major Indices

2.6.1 Overview and Why They Matter

2.6.2 Index Fund NASDAQ-100 ETF (QQQ) Features

2.6.3 S&P 500 Index (INX) Features

2.6.4 Merging Market Features with Individual Stocks

2.7 Date-Related Features

2.7.1 Overview and Why These Features Matter

2.7.2 Engineered Features

2.8 Removing Unnecessary Features and Handling Missing Values

2.9 Feature Engineering Conclusion

3. Machine Learning Models

3.1 Short Introduction

3.1.1 Feature Selection and Simplification

3.2 ARIMA Model

3.2.1 Introduction to ARIMA

3.2.2 Data Preprocessing

3.2.2.1 Data Loading and Scaling

3.2.2.2 Stationary Check

3.2.3 Model Implementation

3.2.4 Results and Model Evaluation

3.3 XGBoost Model

3.3.1 Introduction to Gradient Boosting and XGBoost

3.3.2 Data Preprocessing

3.3.3 Model Implementation and Cross-Validation

3.3.4 Results and Model Evaluation

3.4 Comparative Analysis of ARIMA and XGBoost

3.5 Long Short-Term Memory (LSTM)

3.5.1 Introduction to LSTM

3.5.2 Methodology

3.5.2.1 Dataset and Scaling

3.5.2.2 Sequence Creation

3.5.2.3 Data Splitting

3.5.2.4 Model Architecture

3.5.2.5 Model Training

3.5.2.6 Predictions

3.5.3 Model Evaluation

3.6 Comparative Analysis of ARIMA, XGBoost, and LSTM Models

3.7 Selecting the Best Model for the App

4. Monte Carlo Simulations in Portfolio Optimization

4.1 Theory of Portfolio Optimization

4.2 Monte Carlo Simulations in Portfolio Optimization

4.3 Implementation of Monte Carlo Simulations

5. App Development

5.1 Introduction to Streamlit

5.2 App Structure and Implementation

5.2.1 About This App Tab

5.2.2 Stock Exploration Tab

5.2.3 Price Predictions Using LSTM Tab

5.2.4 Portfolio Allocation Optimization Tab

5.3 App Deployment

5.4 Final Comments on the App

6. Conclusions

7. References

8. Appendix

8.1 Performance Metrics and Visualizations for All Models and All Stocks

1. INTRODUCTION

**1.1 Challenges in Stock Market Investing**

Investing in the stock market presents a myriad of challenges for both novice and experienced investors. Stock prices are inherently volatile, influenced by a confluence of factors ranging from macroeconomic indicators and geopolitical events to company-specific developments and market sentiment. These variables make the prediction of stock price movements a daunting task, even for seasoned professionals.

Moreover, effective portfolio management—balancing risk and return through strategic allocation of assets—requires advanced financial and mathematical expertise. While some tools offer basic portfolio tracking, many lack the analytical rigor needed to provide meaningful insights. Conversely, more advanced tools often demand extensive knowledge, putting them out of reach for individual investors or those with limited financial expertise.

For individual investors, the stakes are high. Missteps in decision-making can lead to substantial financial losses, while success requires careful forecasting of trends and strategic risk management. These challenges highlight the critical need for reliable, data-driven tools that combine predictive capabilities with actionable investment strategies in a user-friendly manner.

**1.2 Project Aim and Proposed Solution**

This project, titled "Stock Prediction and Portfolio Optimization App," seeks to address these challenges by developing a comprehensive, accessible tool that integrates advanced data science techniques into a practical application. Specifically, this project aims to:

1. Enhance Stock Analysis: Enable users to visualize historical trends, assess key financial indicators, and explore stock-specific information.
2. Provide Accurate Predictions: Use advanced machine learning models to forecast stock price movements, empowering investors to make informed decisions based on data.
3. Optimize Portfolio Allocation: Recommend optimal portfolio allocations using Monte Carlo simulations to maximize returns while managing risk effectively.
4. Simplify Complex Analytics: Deliver these advanced tools in a streamlined, interactive app that is intuitive even for non-expert users.

This solution bridges the gap between the growing demand for data-driven investment tools and the lack of accessible, comprehensive options for individual investors. By combining real-time data, machine learning, and optimization techniques, the app serves as a valuable resource for navigating the complexities of modern financial markets.

**1.3 Goals and Objectives**

The primary objective of this project is to develop a fully integrated, data-driven tool for stock market analysis and portfolio optimization. The specific goals are as follows:

1. Develop Robust Forecasting Models: Build, evaluate, and compare multiple machine learning models—ARIMA, XGBoost, and LSTM—to identify the best approach for time-series forecasting.
2. Implement Portfolio Optimization: Design and deploy a Monte Carlo simulation algorithm to recommend portfolio allocations that balance risk and return by maximizing the Sharpe ratio.
3. Create a User-Friendly App: Develop an interactive Streamlit application that integrates stock analysis, price predictions, and portfolio recommendations in an intuitive format.
4. Advance Personal Expertise: Leverage this project as a learning opportunity to explore new techniques, including deep learning and app development, to strengthen data science skills.

**1.4 Why This Project?**

1. Addressing Real-World Challenges: This project tackles two major hurdles faced by investors: forecasting stock price movements and constructing portfolios that balance risk and return. By integrating state-of-the-art machine learning techniques with financial modeling, it aims to deliver reliable, data-driven solutions.
2. Personal Alignment and Development: With a background in finance and a growing expertise in data science, this project represents an opportunity to combine existing skills with new methodologies. It also introduces challenges—such as deep learning and app development—that align with my goal of becoming proficient in advanced data science techniques.
3. Data-Driven Decision-Making: By leveraging data science methodologies, the app facilitates investment decisions grounded in statistical analysis and predictive modeling rather than intuition. This represents a step toward transforming how investors approach market complexities.
4. Democratizing Financial Analytics: Most tools for stock analysis and portfolio management are either overly simplistic or prohibitively complex. This project aims to make sophisticated financial analytics accessible to a broader audience by delivering actionable insights through a user-friendly interface.

**1.5 Problem Statement**

For many individual investors, the stock market can be intimidating. Investing successfully requires a solid understanding of financial principles, such as evaluating technical indicators, forecasting trends, and managing risk effectively. However, the reality is that many investors lack this expertise, which can lead to uninformed decisions, unnecessary risk exposure, and missed opportunities.

While tools and services exist to aid investors, they often fall into two categories:

1. Expensive Financial Advisors: Professional financial advisors typically charge significant fees. Robo-advisors charge 0.25% to 0.50% of assets under management annually, while traditional advisors charge around 1%. Other options include flat retainers of $2,000 to $7,500 annually or hourly rates of $200 to $400. These costs are prohibitive for many individual investors (AdvisoryHQ, 2024; NerdWallet, 2024).
2. Complicated Analytical Platforms: Platforms like Bloomberg Terminal and MetaTrader offer powerful tools for financial analysis and trading but are either prohibitively expensive or too complex for average users. Bloomberg Terminal costs approximately $24,000 per year per user, targeting institutional investors, while MetaTrader's advanced features require extensive expertise to use effectively (Bloomberg L.P., 2024; MetaQuotes, 2024).

These challenges highlight the lack of accessible, user-friendly solutions for the average investor who seeks actionable insights without requiring advanced financial expertise or the ability to pay high fees.

**1.6 Research Questions**

The development of this app is guided by the following research questions:

1. Prediction Performance: Which machine learning model (ARIMA, XGBoost, or LSTM) performs best for stock price prediction, and what factors contribute to its effectiveness?
2. Portfolio Optimization: How can Monte Carlo simulations be used to recommend portfolio allocations that maximize returns while minimizing risk?
3. Feature Importance: What role do engineered features, such as moving averages, Bollinger Bands, and momentum indicators, play in improving model performance?
4. Accessibility: How can complex financial analytics be presented in an intuitive and actionable format for non-expert users?

**1.7 Project Description**

The project is structured into four major components:

1. Data Collection and Feature Engineering: Stock price data is sourced from Yahoo Finance, and a range of technical indicators—such as moving averages, Bollinger Bands, and momentum indicators—are generated to enrich the dataset. Although these features were not used in the final app, the feature engineering process provided valuable insights into the stock market's underlying patterns and honed the technical skills required for financial data modeling.
2. Machine Learning Models: Three time-series forecasting models (ARIMA, XGBoost, and LSTM) were developed, with LSTM emerging as the best performer due to its ability to capture sequential dependencies in time-series data.
3. Portfolio Optimization: A Monte Carlo simulation algorithm was implemented to recommend portfolio allocations that maximize the Sharpe ratio, providing users with actionable strategies for risk-adjusted returns.
4. App Development: These components were integrated into a Streamlit-based app that allows users to explore stock data, predict prices, and optimize portfolios with minimal technical expertise.

**1.8 Expected Outcomes**

This project aims to deliver significant outcomes that are practical, educational, and impactful for individual investors, data science practitioners, and the academic community. The Stock Prediction and Portfolio Optimization App is expected to provide the following:

1. A Comprehensive and User-Friendly Financial Tool

* Integrated Functionality: A single platform that combines real-time stock data exploration, price prediction, and portfolio optimization.
* Ease of Use: Designed for individuals with minimal financial or technical expertise, the app provides actionable insights with a simple, interactive interface.
* Customizable Features: Users can:
  + Select specific stocks to analyze.
  + Adjust prediction horizons.
  + Tailor portfolio allocations to align with individual financial goals and risk tolerance.

2. Reliable Stock Predictions

* Advanced Forecasting: The app uses Long Short-Term Memory (LSTM) models to predict stock prices with precision by analyzing historical trends and recognizing patterns over time.
* Insights into Future Trends: Predictive outputs, accompanied by visualizations, allow users to gauge potential price movements, enabling informed decision-making.

3. Optimized Portfolio Allocation

* Monte Carlo Simulations: A robust algorithm recommends portfolio allocations by simulating thousands of potential outcomes. Key benefits include:
  + Identifying the best balance between risk and return.
  + Providing metrics such as expected return, volatility, and Sharpe Ratio to inform investment decisions.
* Actionable Recommendations: The app produces a clear, visual representation of optimal portfolio weights, empowering users to build diversified and risk-managed portfolios.

4. Empowerment of Individual Investors

* Accessibility: By removing barriers such as high fees and complex interfaces, the app democratizes advanced financial analytics for small-scale and individual investors.
* Confidence Building: Users gain the confidence to navigate the stock market independently, armed with data-driven insights and tools that guide their decisions.

5. Academic and Professional Growth

* Hands-On Learning: This project exposes me to advanced techniques, such as deep learning, portfolio optimization, and app development, which are integral to modern data science and finance.
* Skill Development: The project allows me to practice:
  + End-to-end data science workflows, from feature engineering to deployment.
  + The integration of machine learning models with real-world applications.
  + The design and implementation of user-centric applications.
* Portfolio Showcase: As a capstone project, it serves as a testament to my ability to tackle real-world problems, positioning me as a strong candidate in the competitive fields of finance and data science.

6. Future Opportunities

* Expanding Stock Universe: Currently, the app focuses on eight stocks to streamline the development process and ensure a robust demonstration of its functionality. In the future, the app could include a much broader range of stocks, providing users with more diverse options for analysis and portfolio optimization.
* Enhanced Metrics: Additional financial metrics and technical indicators could be integrated into the app, offering users even deeper insights.
* Improved Prediction Models: Future iterations could incorporate hybrid techniques that combine multiple machine learning models for enhanced accuracy.
* Scalability: The app’s modular design enables scalability, targeting not just individual investors but also small-scale financial advisors and educators.
* Real-Time Integration: Expanding real-time data sources beyond Yahoo Finance could enhance the app’s utility and reliability.

This project bridges the gap between complex financial analysis and accessibility, offering a practical tool to users while serving as a capstone achievement in my data science education. It integrates technical innovation, practical application, and user-centric design, making it a valuable addition to the world of financial analytics and a milestone in my personal and professional growth.

2. DATA COLLECTION AND FEATURE ENGINEERING

**2.1 Overview**

This section details the data collection process, feature engineering techniques, and their theoretical underpinnings for creating a comprehensive dataset to support stock price prediction and portfolio optimization. The dataset spans 15 years (January 2009 - October 2024) and includes 8 major stocks for well-established companies: Netflix (NFLX), Google (GOOGL), IBM (IBM), Johnson & Johnson (JNJ), Coca-Cola (KO), Microsoft (MSFT), Nike (NKE), and Apple (AAPL). Data was retrieved using the Yahoo Finance API (through python’s yfinance library), and feature engineering transformed raw time-series data into meaningful indicators capturing market trends, momentum, and volatility.

**2.2 Why These Stocks?**

The selection of these 8 stocks—Netflix (NFLX), Google (GOOGL), IBM, Johnson & Johnson (JNJ), Coca-Cola (KO), Microsoft (MSFT), Nike (NKE), and Apple (AAPL)—was driven by their representation of diverse industries and their established market presence. These companies were chosen to ensure a well-rounded analysis across different sectors, including technology, healthcare, consumer goods, and entertainment.

Technology giants like Microsoft, Google, and Apple were included due to their substantial influence on global markets, their high growth potential, and their role as leaders in innovation. Netflix was chosen as a representative of the entertainment and digital streaming sector, which has seen dramatic growth and volatility, particularly during the pandemic era. IBM, with its legacy as a tech and business services provider, provides a contrast as a more stable and mature player within the technology sector.

Consumer-focused stocks such as Coca-Cola and Nike were selected to provide insights into defensive and consumer discretionary sectors, respectively. Coca-Cola represents the steady and less volatile performance typical of consumer staples, while Nike demonstrates the potential growth within the lifestyle and apparel industries. Johnson & Johnson was included as a key representative of the healthcare sector, which often exhibits resilience in economic downturns due to consistent demand for healthcare products.

Together, these stocks offer a comprehensive view of the market, capturing the distinct characteristics, risk profiles, and growth potential across different industries. Their inclusion facilitates an analysis of sectoral trends, the impact of macroeconomic factors, and the interplay between high-growth and stable stocks, making them an ideal sample for stock price prediction and portfolio optimization.

**2.3 Data Collection**

The data includes daily stock price information (Open, Close, High, Low, Adjusted Close, and Volume) obtained programmatically using the yfinance Python library. The dataset comprises 3,962 rows for each stock, resulting in 31,696 data points (3,962 x 8 stocks).

Code snippet:

A screenshot of a computer code

Description automatically generated

After collecting the data, the ticker names were deleted, and the ‘Date Time’ was set as the index. As a result, we have 8 dataframes, one for each stock, that look like this:

A screenshot of a computer

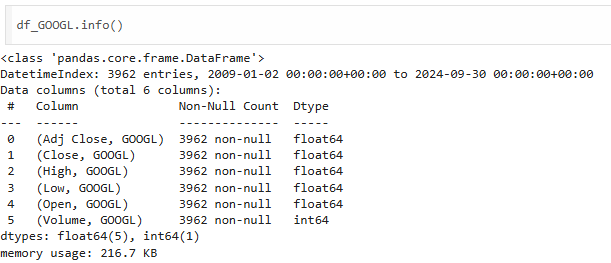
Description automatically generated

As we can see in the Apple stock’s dataframe, there are 6 columns, and ‘Date Time’ is the index.

Column Descriptions:

* **Adj Close** (Adjusted Close): the closing price adjusted for all applicable corporate actions, such as stock splits, dividends, and rights offerings.
  + A stock split (e.g., 2-for-1) would halve the Close price historically, but Adj Close reflects the adjusted price as if the split never occurred.
  + Dividend payments are also factored in, making this value useful for return calculations.
* **Close:** The price of the stock at the end of the trading session.
* **High:** The highest price of the stock reached during the trading session.
* **Low:** The lowest price of the stock reached during the trading session.
* **Open:** The price of the stock at the beginning of the trading session on a given day.
* **Volume:** The total number of shares traded during the session.

We can also check the data types and null values:



As we can see in the info description of Google’s stock dataset, all values are floats (decimals) except for volume which is an integer. There are no null values, we have all the info for the last 15 years of daily trading prices and volume.

We can visualize the close prices for our selected stocks for the specified timeframe:

A graph of stock prices

Description automatically generated

The chart displays the stock price trends for Netflix (NFLX), Google (GOOGL), IBM, Johnson & Johnson (JNJ), Coca-Cola (KO), Microsoft (MSFT), Nike (NKE), and Apple (AAPL) over time. Notably, Netflix (NFLX) and Microsoft (MSFT) exhibit substantial growth, underscoring the strong performance of the tech sector during the observed period. In contrast, stocks like Coca-Cola (KO) and Johnson & Johnson (JNJ) show more stable and less volatile trajectories, typical of consumer staples and healthcare sectors, which are often considered defensive investments.

A key observation is the sharp price volatility of NFLX, particularly after 2019, highlighting its high-risk, high-reward nature. Meanwhile, stocks like IBM and KO demonstrate relatively flat price trends, reflecting limited growth or changes over time. The chart also highlights a significant uptick in stock prices for several companies, especially NFLX, MSFT, and AAPL, post-2020. This growth likely corresponds to market dynamics during the COVID-19 pandemic, which saw an accelerated reliance on technology and digital services. In contrast, the steadiness of defensive stocks like JNJ and KO reinforces their resilience during periods of economic uncertainty.

Overall, this visual analysis reveals the diverse performance trajectories of these companies, emphasizing the impact of sectoral dynamics and external factors on stock price movement. It provides insights into the varying levels of risk and return associated with different industries, making it a valuable reference for portfolio analysis and investment decision-making.

**2.4 Introduction to Feature Engineering**

Feature engineering is a crucial process in transforming raw data into meaningful inputs for predictive modeling and decision-making. In this study, an extensive set of features was engineered, including lag variables, moving averages, Bollinger Bands, momentum indicators, and volatility measures. These features were designed to provide a deeper understanding of stock price behavior, market trends, and potential future movements, with the aim of enhancing predictive accuracy and portfolio optimization strategies.

However, these features were ultimately not used in the final machine learning model or app due to practical considerations and the specific scope of this project. The core idea of the app is to provide real-time predictions, allowing users to access the latest stock prices at any time and generate next-day/week/month forecasts. Many of the engineered features relied on historical data and required extensive pre-processing, which would need to be recalculated dynamically whenever the app was used. This complexity made it infeasible to deliver instantaneous predictions, a key requirement for the app's real-time functionality.

Additionally, for the scope of this project, the focus was on creating a user-friendly application that could serve both novice and experienced investors with minimal delays or technical requirements. Incorporating these advanced features, while potentially improving model accuracy, would have introduced latency and made the app less accessible and practical for its intended audience. As a result, a streamlined approach was adopted, relying on core features that support real-time data analysis.

Although these features were excluded from the final implementation, their development and evaluation are documented in this report to demonstrate the comprehensive and iterative approach undertaken during the feature engineering phase.

**2.5 Features Engineered**

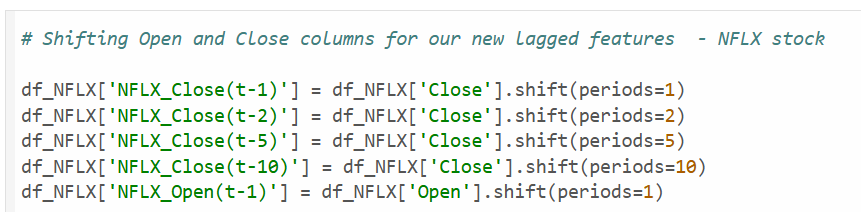
This section explains each engineered feature, its calculation methodology, theoretical basis, and its significance in stock prediction. Note that we engineered these features for the 8 stocks, but for simplicity, we will just show Netflix’s features (we followed the same steps for the other 7 stocks). The calculations were done programmatically using data analytics python libraries like pandas, numpy, matplotlib, and also ta-lib, a library for technical analysis of financial market data.

**2.5.1 Lag features**

Lag features capture historical context by shifting stock prices backward by specified periods. These features allow the model to recognize temporal dependencies, making them vital for identifying short-term patterns and reversals (Hyndman & Athanasopoulos, 2018).

We created lagged values for Open and Close prices by shifting the data. We shifted the Open price 1 day (to have the previous day opening price) and the Close price 1, 2, 5, and 10 days.

Calculation:



**2.5.2 Simple Moving Averages**

The Simple Moving Average smooths out price data by averaging prices over a given number of days, reducing short-term fluctuations.

Formula:

SMA = (P1 + P2 + ... + Pn) / n

where P is the price for each day, and n is the period.

SMAs provide a clear view of trends by reducing short-term fluctuations, aiding in identifying both short- and long-term price momentum (Murphy, 1999).

We used SMA over 5, 10, 20, 50, and 200 days on the closing price to capture various timeframes of trend movement.

Calculation:

A screenshot of a computer code

Description automatically generated

We can visualize this:

A graph with lines and numbers

Description automatically generated

The 5-day moving average (MA5) closely tracks short-term price shifts, capturing recent volatility and quick trend changes, but can be highly reactive. The 10-day (MA10) and 20-day (MA20) MAs offer a more stable look at short-term trends, smoothing out immediate fluctuations. The 50-day MA (MA50) provides a medium-term perspective, often seen as a signal for ongoing trends—if prices stay above it, the stock is generally in an uptrend.

The 200-day moving average (MA200), a key indicator for long-term investors, reflects the overall direction of the stock. When prices stay above the MA200, it signals a long-term uptrend, while dips below can indicate potential downtrends. Together, these MAs reveal Netflix's historical momentum, with crossovers often indicating shifts in trend direction.

**2.5.3 Exponential Moving Averages**

EMA places more weight on recent prices, which makes it more responsive to new data compared to SMA.

Formula:

EMA = (Close - Previous EMA) × (2 / (n + 1)) + Previous EMA

EMA's sensitivity to recent data will help our model respond better to quick shifts in momentum, which is valuable for short-term predictions (Murphy, 1999).

We calculated EMA over 10, 20, 50, 100, and 200 days. Each recent day receives more weight due to a smoothing factor (2 / (n + 1)), allowing for faster adaptation to price changes.

Calculation:

A computer code with text

Description automatically generated with medium confidence

We can visualize this:

A graph showing the number of moving averages

Description automatically generated

As we can see here, the closing price (blue line) generally trends upward, with occasional corrections. Short-term EMAs, like the 10-day (orange) and 20-day (green), react quickly to price changes, capturing immediate trends. The 50-day EMA (red) smooths out some volatility, showing medium-term momentum, while the 100-day (purple) and 200-day (brown) EMAs represent long-term trends, with the 200-day EMA often acting as a support or resistance level.

Key trend shifts are indicated by crossovers, where shorter EMAs move above or below longer ones, signaling potential trend reversals. Toward the end of the chart, all EMAs are trending upward, with the price staying above the 200-day EMA, indicating strong bullish momentum and a likely sustained uptrend. This combination of EMAs provides a multi-timeframe perspective on Netflix’s stock, useful for understanding both short-term fluctuations and long-term trends.

**2.5.4 Bollinger Bands**

Bollinger Bands are a popular technical analysis tool used to assess price volatility in stocks. They consist of three main components: a middle band, which represents the stock's 20-day simple moving average (SMA), providing a baseline for price movements; an upper band, positioned two standard deviations above the SMA, which may indicate overbought conditions; and a lower band, two standard deviations below the SMA, signaling possible oversold conditions (Bollinger, 2001).

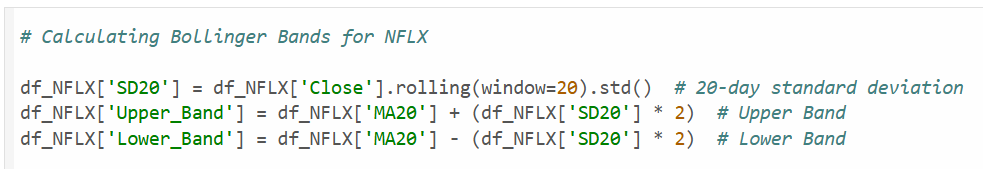
Formula:

Upper Band = SMA(20) + (2 × SD of 20-day Close prices)

Lower Band = SMA(20) - (2 × SD of 20-day Close prices)

By tracking these bands, traders and models alike can better understand shifts in price volatility. When prices move near the upper or lower bands, it often signals extreme price movements, which can indicate potential reversals. In our model, Bollinger Bands will help capture these extreme price shifts, enhancing forecasting accuracy by providing insight into high-volatility periods. This approach allows us to use the upper and lower bands as signals to detect overbought or oversold conditions, offering improved decision-making for potential trading opportunities.

Calculation:



We can visualize this:

A graph of a stock market

Description automatically generated

This chart shows Netflix’s stock price alongside 20-day Bollinger Bands, which help analyze volatility and identify overbought or oversold conditions. The blue line represents the closing price, while the orange line is the 20-day moving average, serving as a central baseline. The green upper band (two standard deviations above the average) and the red lower band (two standard deviations below) expand during volatile periods and contract when the market is calm.

When the price nears the upper band, it may signal overbought conditions; nearing the lower band can indicate oversold conditions. The price generally oscillates within these bands, with sustained moves above or below suggesting momentum shifts. Bollinger Bands thus provide a useful gauge for Netflix's price trends and potential reversal points, helping traders make informed decisions.

**2.5.5 Moving Average Convergence Divergence (MACD)**

The Moving Average Convergence Divergence (MACD) is a momentum indicator used to analyze trend strength and direction. It consists of two components: the MACD Line and the Signal Line (Appel, 2005).

* MACD Line: Calculated as the difference between the 12-day EMA and the 26-day EMA, the MACD Line measures momentum shifts. When the line is above zero, it signals upward momentum; below zero indicates downward momentum.
* Signal Line: This is a 9-day EMA of the MACD Line, smoothing out fluctuations. Crossovers between the MACD Line and Signal Line are key signals: when the MACD Line crosses above the Signal Line, it suggests a potential upward trend (bullish), and when it crosses below, it indicates a potential downward trend (bearish).

MACD helps identify momentum changes and trend reversals, assisting in the timing of trade entries and exits.

We will use MACD to detect momentum shifts in the stocks, enabling our models to recognize potential trend reversals and enhance forecasting by adjusting to changes in trend direction.

Calculation:

A computer code with green and black text

Description automatically generated

Lets visualize this:

A graph with orange and blue lines

Description automatically generated

This graph shows significant fluctuations in Netflix’s momentum over time, with notable spikes and dips, especially from 2018 onward. There are periods where the MACD Line (blue) diverges considerably from the Signal Line (orange), indicating strong momentum shifts. Around 2020 and 2022, the chart reveals extreme volatility, with the MACD reaching pronounced highs and lows, likely correlating with market-wide disruptions.

Throughout the timeline, we see frequent crossovers between the MACD and Signal Line, suggesting regular shifts in bullish and bearish momentum. The MACD oscillates around the zero line, with more intense swings in recent years, reflecting increased price volatility and sharper momentum reversals. This chart suggests that Netflix’s stock has experienced intensified trading momentum and volatility in recent years, particularly during market shocks.

**2.5.6 Standard Deviation of Previous 5 Days Returns**

STD5 measures recent price volatility by calculating the standard deviation of daily returns over a 5-day period.

Formula:

STD5 = Standard deviation of daily returns over 5 days

Why We Use It: Capturing recent volatility will help the model adapt to the current market climate, improving accuracy during periods of high or low variability (Montgomery et al., 2015).

The 5-day STD captures short-term volatility, allowing the model to adjust for quick price swings

Calculation:

A close-up of a computer screen

Description automatically generated

**2.5.7 Average True Range (ATR)**

The Average True Range (ATR) is a volatility indicator that measures how much a stock typically moves over a period, usually 14 days. It captures price stability or instability by averaging the True Range, which considers the largest movement each day, including gaps (Wilder, 1978).

Formula:

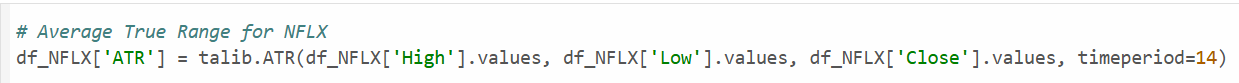
ATR = Average True Range over 14 days, where

True Range = max(Current High-Current Low, |Current High-Previous Close|, |Current Low-Previous Close|)

ATR helps the model adjust for market conditions by accounting for recent volatility. High ATR values indicate significant price swings, while low ATR values suggest steadier movement. This adaptability improves prediction accuracy during varying volatility periods

For this (and the next indicators), we will use TA-Lib (Technical Analysis Library), a library that provides functions for performing technical analysis of financial data, including various indicators and tools commonly used in trading and stock market analysis.

Calculation:



**2.5.8 Average Directional Index (ADX)**

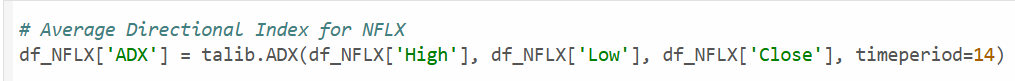
The Average Directional Index (ADX) is a technical indicator used to measure the strength of a trend, whether bullish or bearish, over a period, typically 14 days. ADX doesn't indicate the trend direction but instead quantifies how strong the trend is, helping traders identify whether the market is trending or in a range (Murphy, 1999).

ADX helps the model identify strong trends and distinguish trending markets from range-bound ones. A high ADX value (typically above 25) indicates a strong trend, while a low ADX value suggests a weak trend or sideways market. By identifying these conditions, the model can adapt its predictions to align with either trend-following or mean-reverting strategies, improving overall accuracy.

ADX values range from 0 to 100:

* Below 20: Weak trend or sideways market.
* Above 25: Strong trend.

Calculation:



**2.5.9 Commodity Channel Index (CCI)**

CCI compares the current price to an average over a specified period, identifying cyclical trends (Hyndman & Athanasopoulos, 2018).

Formula:

CCI = (Typical Price - SMA of Typical Price) / (0.015 × Mean Deviation)

where Typical Price = (High + Low + Close) / 3,

SMA is the Simple Moving Average of the Typical Price over a given period, n (typically 20 days),

And Mean Deviation = ∑|Typical Price – SMA|/n

The constant 0.015 normalizes the CCI to ensure most values lie between -100 and +100.

Overbought and Oversold Conditions:

* A CCI value above +100 often signals an overbought condition.
* A CCI value below -100 often indicates an oversold condition.

Why We Use It: CCI helps the model identify potential reversal points, aiding in predicting when prices may be overbought or oversold. We will use a 20-day CCI to capture cyclical patterns in stock prices, especially useful for detecting peaks and troughs.

Calculation:

A screenshot of a computer code

Description automatically generated

**2.5.10 Rate of Change (ROC)**

The Rate of Change (ROC) is a momentum-based technical indicator that calculates the percentage change in a stock's price over a specified period. It quantifies the speed at which the price changes, helping identify momentum shifts and potential reversals (Hyndman & Athanasopoulos, 2018).

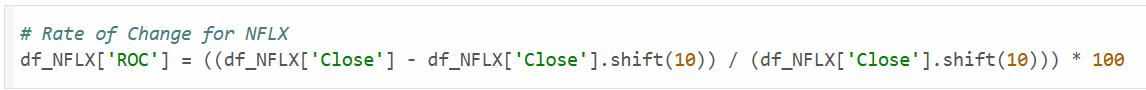
Formula:

ROC = [(Current Price - Price n days ago) / Price n days ago] × 100

Why We Use It: ROC helps the model identify rapid changes in momentum, which can signal upcoming trend reversals.

We will use ROC with a 10-day period to detect short-term momentum shifts, capturing changes in the rate of price movement.

Calculation:



**2.5.11 Relative Strength Index (RSI)**

RSI is a momentum oscillator (an indicator that measures the speed and magnitude of price movements) that identifies overbought or oversold conditions based on recent price gains and losses.

Formula:

RSI = 100 - (100 / (1 + RS)),

where RS = Average Gain / Average Loss over the last 14 days

* Average Gain: The average of all positive price changes over the past n-day period.
* Average Loss: The average of all negative price changes (absolute values) over the same period.

The default lookback period (n) is 14 days, as suggested by J. Welles Wilder, the creator of RSI (Wilder, 1978).

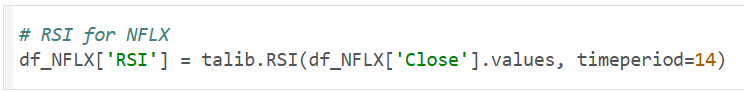
RSI highlights whether a stock's momentum is gaining or losing strength.

* RSI values above 70 often indicate overbought conditions, suggesting a possible downward correction.
* RSI values below 30 indicate oversold conditions, signaling a potential upward rebound.

Why We Use It: RSI can enhance the model's accuracy by highlighting overbought or oversold states, which often precede reversals.

We will use RSI with a 14-day period to identify points where prices might reverse due to extreme market conditions.

Calculation:



**2.5.12 William's %R**

William’s %R (often referred to as %R) is a momentum oscillator that measures the current closing price relative to the high-low range over a specified period, typically 14 days. It is designed to identify overbought or oversold levels in a stock. Unlike RSI, which moves from 0 to 100, William's %R ranges from -100 to 0 (Achelis, 2000).

Formula:

%R = [(Highest High(n) - Close) / (Highest High(n) - Lowest Low(n))] × -100

Where n is the lookback period (typically 14 days).

Highest High: The highest price during the lookback period.

Lowest Low: The lowest price during the same period.

Close: The most recent closing price.

William’s %R is a highly responsive indicator, making it ideal for identifying short-term reversals in stock prices. By incorporating it into our feature set, the model gains an additional perspective on momentum, particularly during volatile periods when extreme price movements are likely.

We will use a 14-day period to calculate William's %R, which helps identify potential reversal points by showing when prices are near recent highs (overbought) or lows (oversold).

Calculation:



**2.5.13 Stochastic %K**

The Stochastic Oscillator %K compares the current closing price to the range of high and low prices over a set period (usually 14 days), indicating the stock's position relative to its recent trading range.

Formula:

%K = [(Close - Lowest Low(n)) / (Highest High(n) - Lowest Low(n))] × 100

Where n is the lookback period (typically 14 days).

Highest High: The highest price during the lookback period.

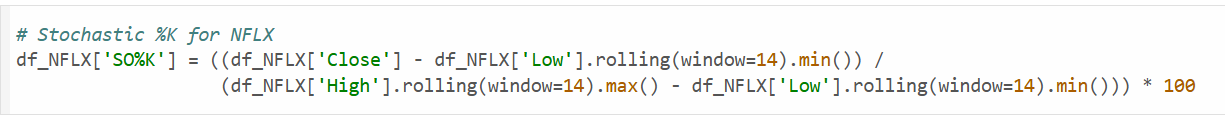
Lowest Low: The lowest price during the same period.

Close: The most recent closing price.

The Stochastic %K oscillator will help our model capture changes in momentum, making it effective for detecting short-term trend reversals, thereby enhancing predictive accuracy.

We will use a 14-day Stochastic %K in our project to capture momentum shifts. This indicator helps identify when prices are nearing the top or bottom of their recent range, signaling potential reversals (Lane, 1984).

Calculation:



**2.5.14 Force Index (FI)**

The Force Index (FI) is a technical indicator that combines price movement and volume to measure the strength of buying or selling pressure. It evaluates the magnitude of price change in combination with volume, making it a useful tool for detecting the intensity behind market moves (Elder, 1993).

Formula:

Force Index = (Close - Previous Close) × Volume

Additionally, a smoothed version of the Force Index is often calculated using an exponential moving average (EMA) over a specific period:

20-day Force Index = EMA(Force Index, 20 days)

The 20-day Force Index is the 20-day exponential moving average (EMA) of the 1-day Force Index values.

Why We Use It: Force Index aids in identifying significant price moves backed by volume, helping the model capture the strength of momentum changes.

We will calculate both a 1-day and 20-day Force Index to assess recent and longer-term buying or selling strength.

Calculation:

A computer code with text

Description automatically generated

**2.6 Market Conditions: Feature Engineering with Major Indices**

**2.6.1 Overview and Why They Matter**

Overview

Market indices such as the NASDAQ-100 and S&P 500 are vital benchmarks for capturing overall market sentiment and trends. Incorporating features derived from these indices allows us to contextualize individual stock price movements within broader market dynamics. This integration ensures that the predictive model considers both stock-specific and market-driven factors, thereby enhancing its robustness and accuracy (Hyndman & Athanasopoulos, 2018).

Why Market Indices Matter

Market indices like the NASDAQ-100 (tracked using the QQQ ETF) and S&P 500 reflect macroeconomic trends, investor sentiment, and sector-specific performance. Stocks often exhibit strong correlations with these indices, particularly during periods of high volatility or economic uncertainty (Achelis, 2001). Including index-related features helps the model:

1. Capture macro-level trends such as sector-wide momentum or global market shifts.
2. Recognize correlations between individual stock behavior and broader market movements.
3. Incorporate short-term and long-term impacts of index fluctuations, enabling accurate forecasts across different time horizons.

**2.6.2 Index Fund NASDAQ-100 ETF (QQQ) Features**

The NASDAQ-100 ETF (QQQ) tracks the performance of the NASDAQ-100, a collection of the largest non-financial companies. This index is particularly relevant for technology and growth-oriented stocks, which frequently correlate with its movements (Elder, 1993).

Process:

1. Data Collection:

* Historical data for the QQQ ETF was retrieved from January 1, 2009, to October 1, 2024 (matching the data from our 8 individual stocks), using the Yahoo Finance API.
* Non-essential columns were removed, and the dataset was sliced to include data from January 2, 2009, onwards.

2. Lagged features:

* Lagged prices for 1, 2, and 5 days (QQQ(t-1), QQQ(t-2), QQQ(t-5)) were added to capture short-term dependencies. These features allow the model to recognize temporal dependencies, which are essential for identifying short-term patterns and reversals (Hyndman & Athanasopoulos, 2018).

A text on a white background

Description automatically generated

3. Simple moving averages:

* Moving averages over 10, 20, and 50 days were calculated to highlight broader market trends. These smoothed averages reduce noise and emphasize long-term patterns, aiding the model in distinguishing persistent trends from short-term volatility (Achelis, 2001).

A close up of text

Description automatically generated

Relevance:

By including QQQ features such as lagged prices and moving averages, the model is equipped to detect correlations between individual stocks and NASDAQ-100 performance. This alignment ensures that stock predictions account for broader market momentum, which is particularly impactful for technology stocks.

QQQ dataset head with extracted features:

A screenshot of a computer

Description automatically generated

**2.6.3 S&P 500 Index (INX) Features**

The S&P 500 Index represents the overall performance of the U.S. stock market, serving as a benchmark for broader economic trends (Investopedia, n.d.).

Process:

1. Data Collection:

* Historical daily closing prices of the S&P 500 were retrieved from January 1, 2009, to October 1, 2024, using the Yahoo Finance API.
* The dataset was preprocessed to include only essential columns and sorted in chronological order.

2. Lagged Features:

* Lagged values such as SnP(t-1) and SnP(t-5) were calculated to capture recent market behavior. These lagged features allow the model to incorporate short-term responses to macroeconomic conditions (Elder, 1993).

A close-up of a computer code

Description automatically generated

Relevance:

Lagged S&P 500 values serve as a proxy for macroeconomic sentiment and investor behavior. By considering these features, the model is better equipped to distinguish between stock-specific and market-driven influences, resulting in more accurate predictions.

SnP dataset head with extracted features:

A screenshot of a computer screen

Description automatically generated

**2.6.4 Merging Market Features with Individual Stocks**

To enrich individual stock datasets, QQQ and S&P 500 features were merged into the corresponding DataFrames.

A screenshot of a computer code

Description automatically generated

Benefits of adding market conditions to stocks dataframes:

* Enhances the model's ability to differentiate stock-specific trends from broader market movements.
* Aligns predictions with prevailing market conditions, improving robustness and accuracy.
* Accounts for macro-level influences that significantly impact individual stock performance.

**2.7 Date-Related Features**

**2.7.1 Overview and Why These Features Matter**

Date-related features provide critical time-based context to stock price movements. These features allow the predictive model to recognize cyclical patterns, seasonality, and calendar-based behaviors, which often influence financial markets. For instance, stocks may exhibit heightened volatility at quarter-end or unique behaviors during year-end trading sessions (Hyndman & Athanasopoulos, 2018).

Incorporating such features helps capture patterns tied to the calendar, enabling the model to differentiate between seasonality-driven fluctuations and stock-specific trends.

Stocks often demonstrate recurring behaviors tied to specific dates or seasons. For example:

* Day-of-Week Effects: Mondays and Fridays often see higher volatility than midweek trading days.
* Quarter-End and Year-End Effects: Institutional investors frequently rebalance portfolios at quarter-end or year-end, impacting stock prices (Achelis, 2001).
* Holiday-Driven Seasonality: Certain periods, like December, often show unique market dynamics due to holiday trading behavior.

Including these features allows the model to account for temporal patterns and predict more accurately during time-sensitive periods.

**2.7.2 Engineered Features**

The following date-related features were added to each stock dataset:

1. **Day of Month (Day)**: The specific day of the month (e.g., 1–31).
2. **Day of Week (DayofWeek)**: Indicates the trading day (e.g., Monday=0, Friday=4).
3. **Day of Year (DayofYear)**: Captures the trading day in the calendar year (e.g., January 1=1, December 31=365).
4. **Week of Year (Week)**: Represents the ISO week number, accounting for weekly trends.
5. **Month-End/Start (Is\_month\_end, Is\_month\_start)**: Flags for the first and last trading days of each month.
6. **Quarter-End/Start (Is\_quarter\_end, Is\_quarter\_start)**: Identifies key quarterly transitions.
7. **Year-End/Start (Is\_year\_end, Is\_year\_start)**: Captures trading behavior at the turn of the year.
8. **Leap Year Flag (Is\_leap\_year)**: A binary indicator for leap years, which impact calendar length.
9. **Year (Year)**: Tracks annual trends.
10. **Month (Month)**: Provides monthly granularity.

To generate these features, the dataset index (set to Date) was leveraged, and a custom function was applied to extract the specified attributes:

A screenshot of a computer program

Description automatically generated

Benefits of Date Features

* Seasonality Detection: Enables the model to detect cyclical price patterns, such as holiday-driven trends or quarter-end fluctuations (Hyndman & Athanasopoulos, 2018).
* Short-Term Predictive Power: Features like DayofWeek and Week improve short-term forecasting accuracy by capturing weekly trading behavior (Achelis, 2001).
* Macro-Level Insights: Year-end or quarter-end indicators allow the model to account for systematic market rebalancing or tax-driven trading behaviors.

**2.8 Removing Unnecessary Features and Handling Missing Values**

To streamline the dataset and enhance the model's efficiency, we removed several features that either provided redundant or unnecessary information. Specifically:

1. Moving Average (MA20): This feature was redundant as other moving averages (MA5, MA10, MA50, MA200) already captured trend insights across different timeframes.
2. Percentage Change (per\_change): Removed because advanced momentum indicators like Bollinger Bands, RSI, and MACD offered richer information for modeling.
3. Exponential Moving Averages (EMA\_12, EMA\_26): These were used in the calculation of MACD, making them redundant as standalone features.

Code snippet:

A close-up of a computer code

Description automatically generated

Additionally, the generation of lagged features, moving averages, and other technical indicators introduced missing values at the start of the dataset, as these calculations depend on a specified lookback period (e.g., 5, 10, or 20 days).

To understand the extent of missing values, the following analysis was performed for each stock dataset:

A screenshot of a computer code

Description automatically generated

We also added a function to calculate the percentage of missing values for each Dataframe.

A screenshot of a computer code

Description automatically generated

Here are the results:

A screenshot of a computer screen

Description automatically generated

The analysis showed that the missing values constituted 0.30% of the total dataset for each stock.

Instead of imputing missing values, which could introduce bias or distort time-sensitive indicators, we opted to drop rows with missing values:

* Minimal Impact: The missing values accounted for less than 1% of the dataset.
* Preservation of Data Integrity: Removing these rows ensures the dataset remains clean and free of artificial imputation, which might misrepresent actual market behavior.

Code snippet:

A close up of a text

Description automatically generated

**Benefits of the Approach**

1. Minimal Data Loss: Dropping rows with missing values resulted in negligible loss (<1%), preserving the dataset's overall integrity.
2. Improved Model Reliability: By eliminating incomplete rows, the model is trained on a consistent and clean dataset, reducing the risk of biases from imputed values.
3. Accurate Representation: Retaining only complete data ensures the time-sensitive features, such as moving averages and momentum indicators, are based on reliable calculations.

This preprocessing step ensures that our dataset is well-prepared for the next phase of modeling and analysis.

**2.9 Feature Engineering Conclusion**

We have successfully engineered 55 new features tailored to enhance the accuracy of stock price predictions. These features span a variety of financial indicators, including moving averages, momentum oscillators, volatility metrics, and date-based components. Together, they provide a comprehensive understanding of both short-term and long-term market behaviors, equipping our models to identify subtle patterns and trends in stock price movements.

Code snippet of final Netflix stock dataset shape and columns after feature engineering:

A close-up of a computer code

Description automatically generated

The cumulative dataset now includes over 30,000 rows and 61 columns across 8 stocks. With this rich dataset, our machine learning models are well-positioned to deliver precise predictions and support effective portfolio optimization strategies. This feature-rich foundation marks a crucial step toward achieving our project goals.

3. MACHINE LEARNING MODELS

**3.1 Short Introduction**

This section outlines the development, testing, and evaluation of machine learning models for stock price prediction. Three models—ARIMA, XGBoost, and LSTM—were built and compared for each stock to identify the best-performing model.

The goal of this section is to identify the best-performing model to integrate into the app to provide real-time predictions.

**3.1.1 Feature Selection and Simplification**

Initially, extensive feature engineering was performed (see Section 2), including lag features, technical indicators, and market indices. However, these features were ultimately excluded from the final ML model and app for the following reasons:

* Real-Time Predictions: The app's goal is to provide instantaneous predictions based on real-time data. Many engineered features relied on historical data and required extensive pre-processing, which would have introduced delays in real-time usage.
* Accessibility and Practicality: Simplifying the feature set ensured a user-friendly application accessible to both novice and experienced investors.
* Trade-Offs: While additional features might improve accuracy marginally, the trade-off in latency and complexity outweighed the benefits for this project.

For these reasons, the primary features used for these models were the daily closing prices of stocks, along with their corresponding time indices (dates).

**3.2 ARIMA Model**

**3.2.1 Introduction to ARIMA**

**Introduction to ARIMA**

The ARIMA (Auto-Regressive Integrated Moving Average) model is a fundamental statistical technique for univariate time series analysis and forecasting. ARIMA combines three components:

1. **Auto-Regressive (AR):** This component models the dependency between the current value and its lagged (past) values. The parameter p determines the number of lagged observations included. For instance, an AR(2) model uses the two most recent observations to predict the next one (Box & Jenkins, 1970).
2. **Integrated (I):** This component refers to differencing the data to achieve stationarity. Stationarity is crucial for time series modeling as it ensures that the statistical properties of the data, such as mean and variance, do not change over time. The parameter d represents the number of differences applied (Hyndman & Athanasopoulos, 2018).
3. **Moving Average (MA):** This component accounts for the dependency between an observation and residual errors from previous observations. The parameter q specifies the number of lagged residuals to include.

The combined ARIMA model is expressed as ARIMA(p,d,q), where p is the number of autoregressive terms, d is the number of differencing steps to achieve stationarity, and q is the number of moving average terms (Box & Jenkins, 1970).

**Application in Stock Prices:** ARIMA is particularly effective for forecasting time series with consistent patterns. However, stock prices are inherently complex, often exhibiting non-linear trends, high volatility, and seasonality, which may challenge ARIMA's predictive capacity.

**3.2.2 Data Preprocessing**

We are going to show the process done for Apple stock. The same method was used for all other stocks.

**3.2.2.1 Data Loading and Scaling**

We first loaded the Apple stock data frame and selected the columns we will use: ‘Date’ and ‘Adj. Price.’ We then ensured the ‘Date’ column is in the right format and set it as the index.

Code snippets:



A screenshot of a computer code

Description automatically generated

The Adjusted Close prices of Apple stock were normalized using MinMaxScaler to scale the data to a range of [0,1]:

A computer code with text

Description automatically generated with medium confidence

The data frame looks like this:

A screenshot of a computer

Description automatically generated

Normalization ensures that large numerical values do not dominate smaller ones, thereby stabilizing the training process. The graph below shows the scaled Adjusted Close prices over time.

Graph Title: Apple Scaled Adjusted Close Prices Over Time

A blue line graph with numbers

Description automatically generated

**3.2.2.2 Stationary Check**

**Definition of Stationarity:** A stationary time series has a constant mean and variance over time and exhibits no trend or seasonality. Stationarity is essential because ARIMA relies on constant statistical properties for reliable forecasting (Hyndman & Athanasopoulos, 2018).

**Methods to Assess Stationarity:**

1. **Graphical Analysis:** Visual inspection of the time series plot can reveal trends or seasonality.
2. **Statistical Tests:** The Augmented Dickey-Fuller (ADF) test is a widely used hypothesis test for stationarity:
   * **Null Hypothesis (H0​):** The time series is non-stationary.
   * **Alternative Hypothesis (H1​):** The time series is stationary.
   * If the p-value is less than 0.05, we can reject H0​ and say that there is enough evidence to conclude that the series is stationary.

As we can see in the Apple Scaled Adjusted Close Prices Over Time graph, the data follow an upward trend over time and it seems to be seasonal, with constant ups and downs. To make sure these observations are correct, lets use the ADF test.

Implementation:

A screenshot of a computer code

Description automatically generated

Results:



Since the p-value was significantly greater than 0.05, we failed to reject H0, confirming that the data was non-stationary. For this reason, we need to convert the data to stationary so we can apply ARIMA model.

**Converting to Stationary: First-Order Differencing**

Theory of First-Order Differencing: This technique transforms a non-stationary series into a stationary one by subtracting the current observation from the previous observation. Mathematically, for a time series Yt ​, the first-order differenced series is:

Y’t = Yt – Yt-1

Differencing removes trends and stabilizes the mean of the series, making it suitable for ARIMA modeling (Box & Jenkins, 1970).

Implementation:



Lets plot this:

Graph Title: First-Order Differenced Series

A blue and white graph

Description automatically generated

The graph below shows the first-order differenced series, which now appears visually stationary. Mean and variance fluctuations have stabilized.

To confirm this, the ADF test was repeated on the differenced series:

A screenshot of a computer code

Description automatically generated

Results:



The p-value was less than 0.05 (close to 0), allowing us to reject H0 and conclude that the series was stationary.

Now that the data is stationary, we can implement the ARIMA Model.

**3.2.3 Model Implementation**

**Parameter Selection:**

The ARIMA model parameters were selected based on:

* p = 2: Two lagged terms for the autoregressive component.
* d = 1: First-order differencing based on the stationarity analysis.
* q =2: Two lagged terms for the moving average component.

**Model Training:**

The ARIMA model was implemented using the *statsmodel* library:

**A screenshot of a computer code

Description automatically generated**

**Forecasting:**

The model was used to forecast Adjusted Close prices for the next 10 days. Predictions were extended dynamically:

A screenshot of a computer code

Description automatically generated

**3.2.4 Model Evaluation**

After developing and implementing the ARIMA model for Apple's adjusted closing prices, the next critical step involved evaluating the performance of the model based on its ability to forecast future stock prices. The results of the ARIMA model are visually represented in the graph below, followed by statistical metrics used to assess the forecasting accuracy.

The graph below compares the actual Adjusted Close prices with ARIMA predictions, highlighting the model's performance. The red dashed line represents the ARIMA forecast, while the blue line depicts the actual adjusted closing prices.

A graph with a line going up

Description automatically generated

It seems from the graph that the ARIMA model does not do a good job of forecasting prices. It fails to capture the trend and seasonality of the time series.

Observations:

* The ARIMA model captures the overall movement in price direction but demonstrates noticeable deviations from the actual prices during volatile periods.
* The forecast remains relatively flat in areas where actual prices exhibit variability, suggesting potential limitations in the model's ability to capture dynamic price changes over short horizons.

This visual evidence aligns with the nature of ARIMA models, which rely solely on past values and lagged terms without incorporating external predictors or explanatory variables.

**Quantitative Metrics**

The ARIMA model's performance was evaluated using the following metrics:

* Mean Squared Error (MSE): Measures the average squared error between actual and predicted values.
* Mean Absolute Error (MAE): Captures the average magnitude of errors.
* Root Mean Squared Error (RMSE): Indicates the standard deviation of prediction errors.

Code implementation:

A screen shot of a computer code

Description automatically generated

Results:

A group of black text

Description automatically generated

Mean Squared Error (MSE): MSE=35.29

* MSE measures the average squared difference between the predicted and actual values. A higher value indicates larger deviations and potential over- or under-fitting issues.

Mean Absolute Error (MAE): MAE=4.73

* MAE evaluates the average magnitude of errors in the predictions, without considering their direction. It provides a straightforward interpretation of average error per prediction.

Root Mean Squared Error (RMSE): RMSE=5.94

* RMSE is a common metric that provides a measure of prediction error in the same units as the original data. A lower RMSE indicates better predictive performance.

**Discussion on Model Performance**

1. **Strengths:**
   * The ARIMA model effectively leverages historical price data and patterns to provide a foundational forecast.
   * It is simple to implement and interpret, making it a useful baseline model for time series forecasting.
2. **Weaknesses:**
   * The relatively high MSE and RMSE suggest that the model struggles to capture sharp fluctuations in stock prices, which is characteristic of financial time series data.
   * The flat nature of the forecast in periods of significant price changes (see the previous graph) reflects the model's limitation in responding to sudden market shifts, such as news or macroeconomic events.
3. **Limitations of ARIMA for Stock Price Prediction:**
   * ARIMA assumes linearity and stationarity, which might not fully represent the complex, non-linear dynamics of stock market behavior.
   * The model's reliance on past price data alone excludes external factors like market sentiment, economic indicators, or news, which are pivotal in driving stock prices.
4. **Practical Implications:**
   * While the ARIMA model serves as a strong theoretical foundation, its performance highlights the need for more advanced models capable of capturing non-linear dependencies and external influences, such as machine learning-based approaches or hybrid models.

**ARIMA Performance for Remaining Stocks**

After applying the ARIMA model to the Apple stock dataset, the same methodology was extended to seven additional stocks: Google, IBM, Johnson & Johnson (JNJ), Coca-Cola (KO), Microsoft, Netflix, and Nike. This consistency in approach ensures comparability across the stocks while demonstrating the robustness of the ARIMA model for stock price forecasting. Below, we provide detailed discussions of the results for each stock, including performance metrics.

**Google (GOOGL)**

The ARIMA model for Google stock produced the following performance metrics:

Mean Squared Error (MSE): 51.80

Mean Absolute Error (MAE): 5.36

Root Mean Squared Error (RMSE): 7.20

These metrics indicate a moderate level of error, which aligns with the relatively high volatility of Google's stock prices.

**IBM**

The ARIMA model's performance for IBM stock showed higher error metrics compared to other stocks:

Mean Squared Error (MSE): 453.58

Mean Absolute Error (MAE): 17.99

Root Mean Squared Error (RMSE): 21.30

The larger error values may be attributed to IBM's unique market behavior, characterized by less volatility but more abrupt price shifts. The model struggled to capture these abrupt changes effectively.

**Johnson & Johnson (JNJ)**

Johnson & Johnson's ARIMA model performed well, with relatively low error metrics:

Mean Squared Error (MSE): 12.96

Mean Absolute Error (MAE): 2.90

Root Mean Squared Error (RMSE): 3.60

JNJ's stock prices tend to be more stable and predictable, which likely contributed to the model's strong performance.

**Coca-Cola (KO)**

For Coca-Cola, the ARIMA model yielded the following results:

Mean Squared Error (MSE): 10.36

Mean Absolute Error (MAE): 2.33

Root Mean Squared Error (RMSE): 3.22

Coca-Cola's relatively steady stock behavior enabled the ARIMA model to provide accurate forecasts, as reflected in the low error metrics.

**Microsoft (MSFT)**

The ARIMA model's performance for Microsoft was as follows:

Mean Squared Error (MSE): 90.48

Mean Absolute Error (MAE): 8.16

Root Mean Squared Error (RMSE): 9.51

Microsoft's stock exhibited higher volatility, which may have impacted the accuracy of the model's predictions. Despite this, the model managed to provide reasonable forecasts.

**Netflix (NFLX)**

Netflix displayed the highest error metrics among the stocks analyzed:

Mean Squared Error (MSE): 2328.19

Mean Absolute Error (MAE): 34.63

Root Mean Squared Error (RMSE): 48.25

The high errors can be attributed to Netflix's extreme price fluctuations and volatility, making it challenging for the ARIMA model to predict accurately. This suggests that ARIMA may not be well-suited for highly volatile stocks like Netflix.

**Nike (NKE)**

Nike's ARIMA model demonstrated satisfactory performance with the following metrics:

Mean Squared Error (MSE): 15.77

Mean Absolute Error (MAE): 3.25

Root Mean Squared Error (RMSE): 3.97

Nike's stock exhibited moderate volatility, and the ARIMA model captured its trends effectively.

The ARIMA model's performance varied significantly across the stocks, with Coca-Cola and Johnson & Johnson demonstrating the lowest error metrics and Netflix showing the highest errors. This variability highlights the importance of considering a stock's volatility and price behavior when selecting forecasting models. ARIMA's reliance on stationarity and its assumptions about linear relationships limit its effectiveness for stocks with high volatility or non-linear trends.

Given the challenges observed in ARIMA's forecasting performance for stock price prediction, lets explore more advanced models, XGBoost and LSTM, to address these limitations.

**3.3 XGBoost Model**

**3.3.1 Introduction to Gradient Boosting and XGBoost**

**Traditional Gradient Boosting**

Gradient boosting is an ensemble machine learning technique that builds a strong predictive model by combining the outputs of several weak learners, typically decision trees. Each successive tree is trained to minimize the residual errors of the previous model, resulting in improved predictions at each iteration (Friedman, 2001). This iterative approach involves the following key steps:

1. **Model Initialization:** The algorithm begins with an initial model that predicts a constant value, such as the mean of the target variable in regression tasks.
2. **Residual Calculation:** The residuals, which are the differences between the actual and predicted values, are calculated after each iteration. These residuals represent the errors of the current model.
3. **New Model Fitting:** A new decision tree is trained on the residuals to learn patterns in the errors.
4. **Update Predictions:** The predictions of the new tree are scaled by a learning rate (η) and added to the existing predictions to refine them. This scaling helps control the contribution of each tree and reduces the risk of overfitting.
5. **Iteration:** The process is repeated for a predefined number of boosting rounds or until the error stops improving.

Although effective, traditional gradient boosting can be computationally expensive and lacks mechanisms for efficient parallelization.

**XGBoost: Enhanced Gradient Boosting**

XGBoost, or Extreme Gradient Boosting, extends the traditional gradient boosting framework with advanced optimizations for speed and accuracy (Chen & Guestrin, 2016). The key features of XGBoost include:

1. **Regularization:**
   * XGBoost incorporates both L1 (Lasso) and L2 (Ridge) regularization to reduce overfitting by penalizing complex trees.
   * This is not available in traditional gradient boosting.
2. **Tree Pruning:**
   * During tree construction, XGBoost prunes nodes that fail to yield a minimum loss reduction, ensuring efficient tree structures.
3. **Parallelization:**
   * XGBoost parallelizes the creation of trees, significantly reducing computation time by distributing tasks across multiple cores.
4. **Handling Missing Data:**
   * It natively handles missing values by learning the best imputation direction during tree building, which is particularly useful for financial datasets.
5. **Weighted Quantile Sketch:**
   * A novel algorithm for finding split points, particularly effective for imbalanced and sparse datasets.
6. **Custom Loss Functions:**
   * Users can define custom loss functions, offering flexibility for different types of prediction problems.

In essence, XGBoost refines traditional gradient boosting by enhancing computational efficiency and introducing robust mechanisms to handle real-world data challenges.

**Key Parameters in XGBoost**

* **Objective:** Specifies the type of problem; 'reg:squarederror' is used for regression tasks.
* **Max Depth:** Limits the depth of each decision tree to control overfitting.
* **Learning Rate (η\etaη):** Determines the contribution of each tree to the final prediction.
* **Subsample:** Controls the fraction of training data used for each tree, reducing overfitting.
* **Colsample\_bytree:** Specifies the fraction of features considered for each split, adding randomness for better generalization.
* **Evaluation Metric:** The metric to optimize; RMSE was chosen for regression problems.

**What is a DMatrix and Why We Use It**

A DMatrix is a specialized data structure in XGBoost optimized for training and predicting with gradient boosting models. Unlike standard data formats (e.g., pandas DataFrame), DMatrix precomputes and stores data in a way that reduces memory usage and accelerates computations.

Why Use DMatrix?

* Efficiency: Reduces computational overhead by compressing sparse data and precomputing statistics for faster gradient and split calculations.
* Built-In Support for Missing Data: Handles missing values efficiently, which is especially important in financial datasets.
* Integration with XGBoost Functions: Many XGBoost-specific functions, such as cross-validation (xgb.cv), are designed to work with DMatrix.

In this project, we converted the training and testing data into DMatrix to leverage these benefits, ensuring efficient model training and evaluation.

**3.3.2 Data Preprocessing**

We are going to show the process done for Apple stock. The same method was used for all other stocks.

**Dataset and Feature Engineering**

* Target Variable: The adjusted closing price (Adj Close) of Apple stock.
* Lag Features: Lag features (lag\_1, lag\_2, ..., lag\_5) were created to include the previous days' prices, providing the model with temporal dependencies to predict future prices.

Code snippet:

**A screen shot of a computer code

Description automatically generated**

The rationale for creating lag features is that stock prices exhibit temporal dependencies; past values often influence future trends.

**Data Splitting**

* The dataset was split into 80% training and 20% testing sets in a chronological order to preserve the time-series nature of the data and prevent data leakage.
* This was done using the train\_test\_split function from the sklearn package.

Code snippet:

**A white background with black text

Description automatically generated**

**3.3.3 Model Implementation and Cross-Validation**

A DMatrix was created to optimize the model's computation. Cross-validation with 5 folds was performed to determine the optimal number of boosting rounds. Early stopping was used to avoid overfitting.

Implementation:

**A screenshot of a computer program

Description automatically generated**

The optimal number of boosting rounds for the Apple stock model was determined as 517, balancing accuracy and complexity.

We then used the best number of rounds to train the model:

A screenshot of a computer program

Description automatically generated

**3.3.4 Results and Model Evaluation**

Predictions were made on the test set, and the performance was evaluated using MSE, RMSE, and MAE. These metrics quantify the accuracy of the predictions, with lower values indicating better performance.

Code snippet:

A screenshot of a computer code

Description automatically generated

Performance Metrics for Apple Stock:

* Mean Squared Error (MSE): 968.94  
  This value represents the average squared difference between the actual and predicted prices. While it captures the overall error magnitude, it tends to amplify the impact of larger errors, making it sensitive to outliers.
* Root Mean Squared Error (RMSE): 31.13  
  RMSE provides an interpretable measure of the error, expressed in the same units as the stock price. A lower RMSE indicates better performance. For this dataset, an RMSE of 31.13 shows that the predicted prices deviate from actual prices by an average of $31.13.
* Mean Absolute Error (MAE): 23.55  
  MAE is less sensitive to outliers compared to RMSE and represents the average absolute deviation between the predicted and actual prices. The model's MAE of $23.55 suggests that, on average, the predictions are off by about $23.55.

We can visualize the Apple stock predicted vs actual values:

A graph of a stock price

Description automatically generated

The graph above illustrates the comparison between actual prices (blue line) and predicted prices (red dashed line) for Apple stock over the test period.

1. **Trend Recognition:**
   * The XGBoost model successfully captures the general direction of the stock prices, particularly during periods of steady trends. This indicates that the lag features provided sufficient historical context for the model to learn patterns in price movements.
2. **Prediction Lag During Volatility:**
   * The model struggles to predict sudden upward or downward price movements, as seen during periods of high volatility. For example, when the actual price exhibits sharp increases or decreases, the predictions remain relatively stable or lag behind. This limitation stems from the model's reliance on historical lag features, which may not fully account for external market shocks, news, or investor sentiment.
3. **Underprediction in High-Value Regions:**
   * During periods of high stock prices, the model underpredicts the actual values. This could be attributed to the limited number of lag features or insufficient emphasis on price momentum in the feature set.

**Model Strengths and Limitations**

Strengths:

* Trend Forecasting: The XGBoost model effectively identifies and replicates long-term trends in stock prices, making it useful for stable stocks or for providing a baseline prediction.
* Lag Feature Utilization: By incorporating lagged price values, the model leverages temporal dependencies inherent in stock data.

Limitations:

* Sensitivity to Volatility: The model performs poorly during periods of high price volatility, as it relies solely on historical prices and lacks features like trading volume spikes, market indices, or macroeconomic indicators that could help capture sudden price changes.

While the XGBoost model demonstrates strong performance in identifying general trends and stable periods, its performance declines during volatile market conditions. The results highlight the importance of incorporating diverse features and exploring alternative models to address the complexities of stock price prediction.

The same methodology was applied to seven additional stocks (Google, IBM, Microsoft, Netflix, Nike, Coca-Cola, and Johnson & Johnson). While the model performed well for stable stocks, it struggled with highly volatile ones like Netflix.

**3.4 Comparative Analysis of ARIMA and XGBoost**

We are going to compare the models’ performance on Apple stock, but this comparison is true for all the other stocks (Appendix 8.1 shows metrics and graphs for all models for all stocks).

* Performance Metrics: The ARIMA model outperforms XGBoost in terms of numerical evaluation metrics:
  + ARIMA Metrics:
    - Mean Squared Error (MSE): 35.29
    - Mean Absolute Error (MAE): 4.73
    - Root Mean Squared Error (RMSE): 5.94
  + XGBoost Metrics:
    - Mean Squared Error (MSE): 968.94
    - Mean Absolute Error (MAE): 23.55
    - Root Mean Squared Error (RMSE): 31.13 The significant difference in these metrics suggests that ARIMA is numerically more accurate, especially in periods of stable price movements.
* Graphical Comparison:
  + While ARIMA exhibits better performance metrics, XGBoost demonstrates superior adaptability to changing trends and seasonality in the data.
  + The ARIMA forecasts tend to remain static or follow a linear trend, which causes the model to lag behind or fail to respond to abrupt price changes and seasonal variations. This is particularly evident in volatile periods or when the actual prices exhibit a non-linear pattern.
  + In contrast, the XGBoost predictions, as seen in the graph, align more closely with the general direction of the actual stock prices, even if they lag slightly during sudden price shifts. This behavior can be attributed to XGBoost's ability to learn non-linear relationships and temporal dependencies through its tree-based ensemble structure.

Conclusion

* Although ARIMA provides better evaluation metrics, the XGBoost model excels in adapting to complex trends and seasonal changes in stock prices. This makes XGBoost a more suitable option for predicting stock prices when capturing trend dynamics is a priority, especially in datasets with inherent volatility or seasonality.

We will now proceed to the implementation of LSTM.

**3.5 Long Short-Term Memory (LSTM)**

**3.5.1 Introduction to LSTM**

**Theory of Long Short-Term Memory (LSTM)**

LSTMs, or Long Short-Term Memory networks, are a special kind of recurrent neural network (RNN) designed to learn from and make predictions based on sequential data. They are widely used in applications like time-series forecasting, natural language processing, and speech recognition (Hochreiter & Schmidhuber, 1997).

**Recurrent Neural Networks (RNNs) Overview**

Before delving into LSTMs, it's important to understand RNNs. Unlike traditional feedforward neural networks, RNNs include feedback connections, allowing them to process sequential data by maintaining a "memory" of prior inputs. This makes RNNs suitable for tasks involving time-series data or sequences where the order of data points matters.

Key Characteristics of RNNs:

* Hidden States: RNNs maintain a hidden state​, which serves as the memory of the sequence.
* Recurrent Connection: The hidden state at time T depends not only on the input x at that time but also on the hidden state from the previous step

Despite their sequential learning capabilities, RNNs face challenges with vanishing gradients when training on long sequences. This limitation makes it difficult for RNNs to learn long-term dependencies.

The vanishing gradient problem is a common issue faced by recurrent neural networks (RNNs) during training. It occurs when gradients of the loss function with respect to weights become very small as they are backpropagated through time. This leads to negligible updates to weights, making it difficult for the network to learn long-term dependencies. This happens because repeated multiplication of gradients (often less than 1) over many time steps causes them to shrink exponentially, effectively "vanishing" (Hochreiter & Schmidhuber, 1997).

**Introduction to LSTMs**

LSTMs were introduced to address the shortcomings of traditional RNNs, specifically the vanishing gradient problem. By incorporating a more complex memory structure, LSTMs can retain relevant information over longer sequences.

**Core Concepts of LSTM Cells**

The core concepts of an LSTM cell revolve around managing the flow of information through three key "gates"—the forget gate, input gate, and output gate—and a cell state, which acts as a memory.

1. **Forget Gate**: This gate decides which information from the previous memory (cell state) should be discarded. It evaluates past data and selectively retains only the relevant parts, ensuring the model doesn't hold onto unnecessary or outdated information.
2. **Input Gate**: This gate determines what new information to add to the memory. It has two components:
   * The first part decides which values to update in the memory.
   * The second part generates potential new information (called a candidate) to be added. By combining these, the input gate updates the memory with fresh, meaningful information while avoiding irrelevant details.
3. **Cell State Update**: The forget gate and input gate work together to update the memory. The forget gate removes unneeded information, and the input gate adds new, relevant information, creating an updated memory that reflects both past and current inputs.
4. **Output Gate**: This gate controls the output of the LSTM cell. It decides what information to pass on from the updated memory, based on the current inputs and past outputs. The result is the final output of the cell, which is also influenced by the updated memory's contents.

By dynamically regulating the flow of information at each step, LSTM cells effectively learn long-term dependencies in sequential data while avoiding problems like the vanishing gradient.

**Why LSTMs Are Effective:**

* Long-Term Memory: The cell state allows LSTMs to store information over long time periods.
* Selective Memory: Gates enable the network to focus on relevant parts of the sequence and ignore irrelevant data.
* Resilience to Vanishing Gradients: Through gating mechanisms, LSTMs avoid the diminishing gradients that hinder traditional RNNs.

**3.5.2 Methodology**

We are going to show the process done for Apple stock. The same method was used for all other stocks.

**3.5.2.1 Dataset and Scaling**

The dataset consisted of the adjusted closing prices (Adj Close) for Apple stock. To ensure numerical stability and efficient gradient-based optimization, the data was scaled to the range [0, 1] using Min-Max scaling. This is the same thing we did for ARIMA.

Code snippet:

A screenshot of a computer program

Description automatically generated

**3.5.2.2 Sequence Creation**

The LSTM model requires inputs in the form of sequences. A window size of 60 was chosen, meaning the model predicts the next day’s price using the previous 60 days as input.

Implementation:

A computer screen shot of a code

Description automatically generated

**3.5.2.3 Data Splitting**

The dataset was split chronologically into training and testing sets, ensuring no leakage of future information during training.

Code implementation:

A screenshot of a computer code

Description automatically generated

We used a manual splitting, using the first 3000 rows as the training dataset and the remaining 702 rows as the test dataset (roughly 81% data used for training and the remaining 19% for testing).

**3.5.2.4 Model Architecture**

Code implementation:

A screen shot of a computer code

Description automatically generated

Three LSTM layers are included, each with 50 units (neurons) and different configurations:

* First LSTM Layer:
  + Units: 50, meaning the layer will learn 50 features or patterns from the input sequences.
  + Return Sequences: Set to True, indicating that the layer will output the entire sequence for the next layer (required when stacking LSTMs).
  + This layer processes the sequential data and retains temporal dependencies.
* Second LSTM Layer:
  + Similar to the first layer, it has 50 units and return\_sequences=True. It further refines the learned patterns from the first layer and passes the entire sequence to the next layer.
* Third LSTM Layer:
  + Units: 50, like the previous layers.
  + Return Sequences: Set to False, indicating that the layer will only output the final time step's information. This is useful for tasks where only a single output (like regression) is required.

**Dropout Layers**

Dropout layers are added after each LSTM layer to prevent overfitting:

* **Dropout Rate**: 30% of the neurons are randomly "dropped out" (set to zero) during training. This ensures that the model does not overly rely on specific neurons, improving its ability to generalize to unseen data.

**Dense Layer**

The final layer is a fully connected (Dense) layer:

* **Units**: 1, meaning it outputs a single value, in this case, the stock’s price prediction.

**Summary of Workflow**

* The first LSTM layer takes sequential input and extracts temporal features.
* The subsequent LSTM layers build on these features, capturing more abstract and complex patterns in the data.
* Dropout layers reduce the risk of overfitting by randomly deactivating some neurons.
* The Dense layer produces the final output, which is the predicted value for the input sequence.

**3.5.2.5 Model Training**

Implementation:

A screenshot of a computer program

Description automatically generated

The training process of this LSTM model involves several strategies and components that ensure effective learning and prevention of overfitting.

**1. Model Compilation**

* **Optimizer: Adam**
  + Adam (Adaptive Moment Estimation) is used to optimize the model’s weights during training.
  + It adjusts the learning rate dynamically for each parameter based on first-order and second-order moments of gradients (Kingma & Ba, 2014).
  + Adam is computationally efficient and well-suited for time-series tasks.
* **Loss Function: Mean Squared Error (MSE)**
  + MSE is ideal for regression problems, as it calculates the squared difference between predicted and actual values.
  + Larger errors are penalized more heavily, encouraging the model to minimize significant deviations (Chollet, 2018).

**2. Early Stopping**

* **Purpose**: Prevents overfitting and ensures computational efficiency.
* **Monitor**: The validation loss (val\_loss) is tracked after each epoch to evaluate generalization performance.
* **Patience**: Training stops if the validation loss does not improve for 5 consecutive epochs.
* **Restore Best Weights**: Ensures that the model reverts to the weights associated with the best validation loss, avoiding overtraining (Prechelt, 1998).

**3. Model Training (fit method)**

The model is trained using the fit method with the following configurations:

* **Training Data**:
  + X\_train: Input sequences.
  + y\_train: Target values corresponding to each input.
* **Validation Split**:
  + 20% of the training data is set aside as validation data. This evaluates the model’s performance on unseen data after each epoch.
* **Epochs**:
  + The model is allowed to train for up to 100 epochs. However, early stopping typically halts training earlier when validation loss plateaus.
* **Batch Size**:
  + A batch size of 3 means the model processes 3 samples at a time before updating weights. Smaller batches allow finer gradient updates, which can be beneficial for smaller datasets.
* **Callbacks**:
  + Early stopping is applied as a callback, halting training once the validation loss stops improving.

**4. Training Process**

* **Forward Propagation**:
  + The model processes input sequences to generate predictions.
  + Loss is calculated using the MSE function by comparing predicted and actual values.
* **Backward Propagation**:
  + Gradients of the loss with respect to model weights are computed.
  + The Adam optimizer updates the weights based on these gradients, dynamically adjusting learning rates.
* **Validation**:
  + After each epoch, the model evaluates validation loss to ensure it generalizes well to unseen data.
* **History Object**:
  + The history object stores training and validation metrics, such as loss, for each epoch.
  + This can be visualized to assess convergence and detect overfitting.

**Summary**

This training process combines:

1. **Adam optimizer** for efficient weight updates.
2. **MSE loss function** for minimizing prediction errors in regression.
3. **Early stopping** to prevent overfitting and restore the best weights.
4. **Validation split** to monitor generalization performance.

These methods create a robust training pipeline for the LSTM model.

The trained LSTM model was saved in .keras format using TensorFlow's model.save() functionality. This will allow us to integrate it into our web app by calling it when we need to make new predictions.

**3.5.2.6 Predictions**

After training the LSTM model, predictions were made using the test set. The predicted values were scaled back to their original range using the inverse transformation of the scaler to ensure comparability with the actual values. Both the predicted and actual values were prepared for evaluation.

Code implementation:

A screenshot of a computer program

Description automatically generated

**3.5.3 Model Evaluation**

To assess the model's performance, the following evaluation metrics were calculated:

* Mean Absolute Error (MAE): Measures the average magnitude of the errors in the predictions without considering their direction.
* Root Mean Squared Error (RMSE): Provides a measure of the prediction error's magnitude while giving more weight to larger errors.
* Mean Squared Error (MSE): Represents the average of squared differences between predicted and actual values, penalizing larger errors more heavily.

For the "Adj Close" price column:

* MAE: 11.0285
* RMSE: 13.8306
* MSE: 191.2844

These metrics indicate the model's accuracy in predicting the adjusted closing prices of Apple stock. The LSTM model effectively captures general trends but struggles with sudden price changes, especially during volatile periods.

**Visualization:** The graph below compares the actual and predicted Apple stock prices in the test data set. The LSTM model closely follows the overall trend but exhibits some smoothing, particularly during sharp price movements.

A graph showing a blue and red line

Description automatically generated

**Insights from Visualization**

The plots suggest:

* The LSTM model effectively identifies long-term trends in the stock price data.
* Short-term fluctuations are less accurately predicted, possibly due to the inherent noise in stock price movements.
* The model's overall performance highlights its ability to capture sequential patterns in time-series data.

**Strengths:**

The LSTM model effectively captures sequential dependencies in the data, as evidenced by the predictions closely following the general trends of the stock price. Regularization techniques such as Dropout and early stopping helped mitigate overfitting.

**Limitations:**

The model's prediction errors, as indicated by MAE and RMSE, suggest room for improvement in capturing short-term fluctuations.

The same methodology was applied to seven other stocks (Google, IBM, Microsoft, Netflix, Nike, Coca-Cola, and Johnson & Johnson).

**3.6 Comparative Analysis of ARIMA, XGBoost, and LSTM Models**

In this section, we compare the performance of the three models—ARIMA, XGBoost, and LSTM—on Apple stock data. The analysis is based on quantitative metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). While these results are presented for Apple stock, the same methodology and evaluation criteria were applied to the other stocks (Google, IBM, Microsoft, Netflix, Nike, Coca-Cola, and Johnson & Johnson) (Appendix 8.1 shows metrics and graphs for all stocks).

**Apple Metrics Summary**

**ARIMA Model**

* Mean Squared Error (MSE): 35.29
* Mean Absolute Error (MAE): 4.73
* Root Mean Squared Error (RMSE): 5.94

**XGBoost Model**

* Mean Squared Error (MSE): 968.94
* Mean Absolute Error (MAE): 23.55
* Root Mean Squared Error (RMSE): 31.13

**LSTM Model**

* Mean Squared Error (MSE): 191.28
* Mean Absolute Error (MAE): 11.03
* Root Mean Squared Error (RMSE): 13.83

**Graphical Comparison of Models**

**ARIMA Forecast:**

A graph with a line going up

Description automatically generated

The ARIMA model demonstrated a relatively smooth forecast, failing to adapt to the rapid fluctuations in stock prices. It struggled to capture the seasonality and trend reversals effectively.

**XGBoost Forecast:**

A graph of a stock price

Description automatically generated

The XGBoost model exhibited a clear lag in predicting sudden changes in stock prices. While it captured some general trends, the deviations in highly volatile periods were significant, as evident from the error metrics.

**LSTM Forecast:**

A graph showing a blue and red line

Description automatically generated

The LSTM model performed better than XGBoost in adapting to trends and capturing non-linear patterns. However, it still lagged behind ARIMA in terms of quantitative error metrics.

**Performance Analysis**

**1. Error Metrics and Trends**

* **ARIMA** achieved the best performance across all three metrics, indicating its capability to handle short-term dependencies and generate reliable forecasts for financial time-series data. However, its inability to adapt to trend changes is a major limitation.
* **XGBoost** had the highest error metrics. Its reliance on lag features was insufficient for capturing the complexity of stock price patterns, especially during volatile periods.
* **LSTM** balanced between the two, with lower errors compared to XGBoost but higher than ARIMA. The model’s ability to learn non-linear and long-term dependencies provided an advantage, but it still struggled with high volatility.

**2. Graphical Insights**

* While ARIMA excelled in smooth predictions, it failed to capture the amplitude of trends and fluctuations.
* XGBoost predictions largely stayed close to the mean and exhibited significant lag in volatile regions.
* LSTM closely followed the trend, adapting to upward and downward movements better than ARIMA and XGBoost. This demonstrates its capacity to capture sequential dependencies and seasonal trends effectively.

**3.7 Selecting the Best Model for the App**

Based on the analysis, LSTM would be the most suitable model for the web app.

**Why LSTM is the Best Fit:**

**Adaptability to Trends and Seasonality:** LSTM performed better than ARIMA and XGBoost in adapting to both trends and seasonal fluctuations. This is critical for stock prices, which often show non-linear patterns and changing volatility over time.

**Suitability for Long-Term Predictions**: LSTMs are inherently designed to capture long-term dependencies in time-series data, making them ideal for predictions over longer horizons (e.g., up to 30 days). ARIMA, on the other hand, struggles with long-term forecasts, and XGBoost is limited in its sequential understanding of the data.

**Scalability for a Web App:** The LSTM model can be trained and saved as a preloaded model in the backend of the web app. Users' inputs (e.g., stock choice and prediction duration) can then dynamically generate predictions based on a fine-tuned, pre-trained LSTM model.

**User Experience:** Users expect predictions to adapt to market conditions, even if the exact values are not perfect. LSTM's ability to capture non-linear trends provides a better user experience compared to the linear predictions of ARIMA or the lagged trends of XGBoost.

**Addressing ARIMA and XGBoost:**

* **ARIMA:** While ARIMA outperformed in terms of error metrics, its rigidity and inability to handle non-linearities make it less suitable for dynamic, real-time stock forecasting in a web app.
* **XGBoost:** XGBoost struggled with error metrics and trend adaptation. It is better suited for structured datasets with clearly defined features, not complex time-series data with high volatility.

4. MONTE CARLO SIMULATIONS FOR PORTFOLIO OPTIMIZATION

**4.1 Theory of Portfolio Optimization**

What is Portfolio Optimization?

Portfolio optimization aims to allocate assets in a way that achieves specific financial objectives, such as maximizing returns, minimizing risk, or balancing the two. Introduced by Markowitz (1952), Modern Portfolio Theory (MPT) provides a quantitative framework for diversification, emphasizing that combining assets with low correlations can reduce overall portfolio risk without sacrificing returns.

**Key Metrics in Portfolio Optimization**

**1. Portfolio Return:** Portfolio return represents the weighted average return of all assets in the portfolio. The formula is:

Portfolio Return = Sum (Weight of asset i times mean return of asset i)

A higher Portfolio Return indicates a greater expected annualized return.

**2. Portfolio Volatility:** Portfolio volatility measures the overall risk of the portfolio by accounting for individual asset variances and covariances:

σp​=(wTΣw)1/2

Where:

* w: Asset weight vector.
* Σ: Covariance matrix of asset returns.

Lower volatility suggests lower risk, making the portfolio more stable.

**3. Covariance Matrix:** The covariance matrix quantifies the relationships between asset returns:

* A positive covariance means two assets move in the same direction.
* A negative covariance means two assets move in opposite directions.

Covariance is central to diversification; combining assets with low or negative covariance reduces overall risk (Markowitz, 1952).

**4. Sharpe Ratio:** The Sharpe Ratio, developed by Sharpe (1966), is the most widely used metric for evaluating risk-adjusted returns. It measures how much excess return a portfolio generates for every unit of risk:

Sharpe Ratio = (Expected return of portfolio - risk-free rate of return) / portfolio volatility

The risk-free rate of return is the interest rate an investor would earn on an investment that has zero risk. It is assumed to be 1% as this is the 1-year U.S. Treasury bill return.

Sharpe Ratio interpretation:

**Positive Sharpe Ratio**:

* A higher Sharpe Ratio indicates better risk-adjusted performance. It means the portfolio is generating higher excess returns (returns above the risk-free rate) for the level of risk taken.
* For example, a Sharpe Ratio of 1 means that the portfolio earns 1 unit of return for every unit of risk.

**Negative Sharpe Ratio**:

* A negative Sharpe Ratio occurs when the portfolio's return is lower than the risk-free rate. This indicates that the investment is underperforming relative to a risk-free asset, even when accounting for risk.

**Ranges for Interpretation**:

* **< 0**: Poor performance; the investment is not compensating for risk.
* **0–1**: Acceptable performance; better than risk-free assets but not exceptional.
* **1–2**: Good performance; the portfolio generates solid risk-adjusted returns.
* **> 2**: Excellent performance; the investment provides very high returns relative to risk.

**Benchmark Comparison**:

* The Sharpe Ratio is most meaningful when compared to other investments, portfolios, or benchmarks. For instance:
  + If Portfolio A has a Sharpe Ratio of 1.5 and Portfolio B has 1.0, Portfolio A offers better risk-adjusted returns.

**4.2 Monte Carlo Simulations in Portfolio Optimization**

**What is a Monte Carlo Simulation?**

Monte Carlo simulation is a computational technique that uses random sampling to explore various scenarios in complex systems. In finance, it models uncertain outcomes by generating thousands of possible portfolio configurations and evaluating their performance (Glasserman, 2003).

**How Does Monte Carlo Simulation Work in Portfolio Optimization?**

1. **Random Weight Generation**
   * Each simulation begins by assigning random weights to portfolio assets, ensuring they sum to 1.
   * For example: w=[0.3, 0.5, 0.2] means 30% is allocated to Asset 1, 50% to Asset 2, and 20% to Asset 3.
2. **Portfolio Performance Metrics**
   * **Expected Return (Portfolio Return​)**: Calculated using historical mean returns of assets and their weights.
   * **Volatility (Portfolio Volatility):** Computed using the covariance matrix to account for diversification effects.
   * **Sharpe Ratio**: Evaluates risk-adjusted performance by comparing excess return to portfolio volatility.
3. **Iteration**
   * The process is repeated thousands of times (e.g., 10,000 simulations), each with a unique set of weights.
4. **Optimization**
   * The portfolio with the highest Sharpe Ratio is identified as the optimal portfolio, providing the best balance of risk and return.

**Why Use Monte Carlo Simulations?**

* **Exploration of Possibilities**: By generating thousands of portfolios, Monte Carlo simulations ensure that many potential allocations are assessed.
* **Quantifying Uncertainty**: They model the variability in outcomes, helping investors understand the trade-offs between risk and return.
* **Data-driven decision-making**: Monte Carlo simulations leverage historical data to provide actionable insights, reducing reliance on intuition or arbitrary allocations.

**4.3 Implementation of Monte Carlo Simulations**

**1. Data Preparation**

To perform Monte Carlo simulations, historical stock price data is processed to calculate daily returns:

1. **Adjusted Closing Prices**: Historical adjusted closing prices for stocks are used to account for stock splits and dividends, ensuring consistency over time.
2. **Daily Returns**: Calculated as the percentage change between consecutive days, excluding missing values caused by the initial shift.

Code snippet:

A close-up of a text

Description automatically generated

Daily returns serve as the foundation for calculating portfolio-level metrics, including return and volatility.

**2. Monte Carlo Simulation Parameters**

The simulation runs multiple iterations, each with a random portfolio allocation. Key parameters include:

1. **Number of Simulations (num\_simulations)**: Determines the number of portfolio configurations tested. In this case, 10,000 simulations were run for comprehensive exploration.
2. **Risk-Free Rate (risk\_free\_rate)**: Set to 1% (0.01) annually, representing the return of a virtually risk-free investment like Treasury bills.
3. **Storage Arrays**: Results for each portfolio configuration—return, volatility, and Sharpe Ratio—are stored in a matrix for analysis.

Code snippet:

A number and numbers on a white background

Description automatically generated

**3. Random Portfolio Generation**

For each simulation, random portfolio weights are generated, normalized to sum to 1:

A close-up of words

Description automatically generated

These weights represent the proportion of the portfolio allocated to each stock in a given simulation.

**4. Portfolio Metrics Calculation**

The following metrics are calculated for each portfolio configuration:

**a. Expected Portfolio Return**

Rp=∑(wi⋅μi) × 252R

* wi: Weight of stock iii.
* μi ​: Mean daily return of stock i.
* 252: Number of trading days in a year, used to annualize the return.

Code snippet:



**b. Portfolio Volatility (Risk)**

σp​=(wTΣw)1/2

Where:

* w: Asset weight vector.
* Σ: Covariance matrix of asset returns.

Portfolio volatility accounts for diversification effects, where the risk of a portfolio can be lower than the sum of individual risks due to negative correlations.

Code snippet:



**c. Sharpe Ratio**

The Sharpe Ratio measures risk-adjusted return:

Sharpe Ratio = (Portfolio Return - Risk-Free Rate of Return) / Portfolio Volatility

Code snippet:



**5. Optimization: Identifying the Best Portfolio**

The simulation identifies the portfolio with the maximum Sharpe Ratio. This is done by comparing the Sharpe Ratio of each simulated portfolio and updating the maximum Sharpe Ratio and the corresponding weights if a higher value is found.

Code snippet:

A white background with black text

Description automatically generated

**6. Storing Results**

The metrics for each portfolio configuration are stored in a matrix for analysis:

A group of black text

Description automatically generated

After all simulations, the results are converted into a DataFrame for easier analysis and visualization:



**7. Optimal Portfolio Metrics**

After the simulation completes, the portfolio with the highest Sharpe Ratio is identified, and its metrics are calculated:

1. **Optimal Portfolio Return**: Annualized return of the best portfolio.
2. **Optimal Portfolio Volatility**: Annualized risk of the best portfolio.
3. **Optimal Sharpe Ratio**: The highest Sharpe Ratio across all simulations.

Code snippet:

A close up of a text

Description automatically generated

**8. Key Features of Implementation**

* **Scalability**: The simulation can handle any number of stocks, limited only by computational resources.
* **Robustness**: Randomized weights ensure that the simulation explores a wide range of portfolio configurations.
* **Optimization**: The approach identifies the portfolio that maximizes the risk-adjusted return (Sharpe Ratio).

This Monte Carlo implementation leverages historical data to make informed, data-driven investment decisions, providing a robust framework for portfolio optimization. While historical performance is no guarantee of future results, the Monte Carlo simulation offers a powerful tool for exploring potential outcomes and identifying optimal asset allocations.

This framework will be integrated into the web-based app so users can choose in what stocks they would like to invest, and the app will suggest how to allocate their investments, the potential return, volatility, and Sharpe ratio.

5. WEB APP DEVELOPMENT

The Stock Prediction and Portfolio Optimization App was developed using Streamlit, a Python-based framework tailored for creating interactive, user-friendly web applications. This section details the tools and technologies used, the app's structure, and the implementation of its various functionalities.

The complete Python script for the streamlit web app can be found [here](https://github.com/davidhellerw/DS-Capstone/blob/main/app.py).

**5.1 Introduction to Streamlit**

**What is Streamlit?**

Streamlit is an open-source Python library designed specifically for building and sharing data applications. Unlike traditional web frameworks, Streamlit simplifies the app development process by allowing users to write Python scripts that dynamically render interactive web applications. Its primary benefits include:

* **Ease of Development**: Developers can focus on writing Python code without needing expertise in HTML, CSS, or JavaScript.
* **Real-Time Feedback**: The app updates live as developers modify the script, offering an efficient iterative development process.
* **Rich Visualization Support**: It integrates seamlessly with Python visualization libraries such as Matplotlib, Plotly, and Seaborn, enabling the creation of interactive charts and plots.
* **Wide Widget Support**: Streamlit provides intuitive widgets (e.g., sliders, dropdowns, checkboxes) that allow users to interact with the application dynamically (Rao et al., 2020).

Streamlit was chosen for this project because it aligns perfectly with the goal of creating a user-friendly, data-driven application for investors with minimal technical expertise.

**5.2 App Structure and Implementation**

The app was designed with a modular structure, divided into four distinct tabs, each focusing on a specific functionality. Below is a detailed, tab-by-tab explanation of the app's development.

There are 4 tabs:

1. About this App
2. Stock Exploration
3. Price Predictions Using ML Models
4. Portfolio Allocation Optimization

**5.2.1 About This App Tab**

This tab serves as the introduction to the app, explaining its purpose, features, and the author's background.

**Implementation Details**:

1. **Title and Description**:
   * The app title and an overview of its features were added using st.title() and st.markdown().
2. **Features Section**:
   * A detailed breakdown of the app’s functionalities, including stock exploration, price prediction using LSTM, and portfolio optimization, was provided.
3. **Author Information**:
   * A brief introduction to the author, along with clickable links to my LinkedIn, GitHub, and personal website.

Code snippet:

A screenshot of a computer

Description automatically generated

Front-end snippet:

A screenshot of a application

Description automatically generated

A screenshot of a computer

Description automatically generated

**5.2.2 Stock Exploration Tab**

This tab allows users to explore historical stock data, visualize trends, and access key metrics.

**Implementation Details**:

1. **Stock Selection**:
   * A dropdown menu (st.selectbox) enables users to select a stock ticker from a pre-defined list of 100 stocks.
   * Historical stock data is fetched using the Yahoo Finance API via the yfinance library (Huang et al., 2022), so users can choose the timeframe of data they want to fetch.
2. **Data Display**:
   * A summary table provides key statistics like maximum close price, minimum close price, and descriptive statistics.
   * Company-specific metrics such as market capitalization and 52-week highs/lows are displayed using the yfinance.Ticker.info attribute.
3. **Moving Averages Visualization**:
   * Calculations for 20-day and 50-day moving averages were performed using Pandas' rolling() method.
   * A line chart visualizing the closing price alongside these moving averages was created using Matplotlib.
   * Interpretation: The moving averages help users identify trends and potential buy/sell signals. For example, when the 20-day moving average crosses above the 50-day moving average, it indicates a bullish trend (Chen, 2018).

Code snippets:

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

Front-end snippets:

A screenshot of a stock data

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A graph with lines and numbers

Description automatically generated

**5.2.3 Price Predictions Using LSTM Tab**

This tab provides stock price predictions using pre-trained Long Short-Term Memory (LSTM) models for our 8 stocks.

**Implementation Details**:

1. **Stock Selection and Prediction Horizon**:
   * Users select a stock and specify the prediction horizon (1–30 days) using st.selectbox and st.slider.
2. **Pre-Trained Models**:
   * LSTM models trained on historical stock data were loaded using TensorFlow's load\_model() (Hochreiter & Schmidhuber, 1997).
3. **Iterative Predictions**:
   * The model predicts one day at a time using the last 60 days of stock data as input. Each prediction is appended to the input for the next iteration.
4. **Visualization**:
   * Predictions are displayed alongside historical data using a candlestick chart created with *mplfinance.*

Code snippets:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Front-end snippets:

A screenshot of a computer

Description automatically generated

A white text on a white background

Description automatically generated

**5.2.4 Portfolio Allocation Optimization Tab**

This tab uses Monte Carlo simulations to recommend portfolio allocations.

**Implementation Details**:

1. **Data Preparation**:
   * Daily returns for the selected stocks are calculated using percentage changes.
   * A covariance matrix is generated to account for stock correlations.
2. **Monte Carlo Simulations**:
   * Random weights are assigned to the selected stocks, ensuring they sum to 1.
   * For each portfolio configuration, expected return, volatility, and Sharpe Ratio are calculated (Sharpe, 1966).
   * The portfolio with the highest Sharpe Ratio is identified as the optimal allocation.
3. **Visualization**:
   * A scatter plot displays the risk-return tradeoff for all simulated portfolios, with a red star highlighting the optimal portfolio.
   * A pie chart shows the allocation percentages for the optimal portfolio.

Code snippets:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

Front-end snippets:

A screenshot of a computer

Description automatically generated

A graph of a graph showing a rising rate

Description automatically generated with medium confidence

A blue circle with a red line

Description automatically generated

**5.3 App Deployment**

The application was deployed on Streamlit Cloud, allowing it to be accessible via a web browser. The app is connected to a GitHub repository, which stores the main application script, as well as all pre-trained LSTM models and dependencies.

This deployment process enables:

1. **Version Control**: Updates to the codebase in GitHub automatically reflect in the deployed app.
2. **Accessibility**: Users can interact with the app without needing to install Python or its dependencies locally.
3. **Scalability**: Streamlit Cloud provides the infrastructure to handle multiple users simultaneously.

**5.4 Final Comments on the App**

The Stock Prediction and Portfolio Optimization App is designed to provide a user-friendly platform for analyzing stocks, predicting prices, and optimizing portfolios. Each tab serves a specific purpose tailored to both novice and experienced investors, but there are certain limitations and design choices worth highlighting:

1. Stock Exploration and Portfolio Allocation Tabs:
   * In these tabs, users can select stocks from a comprehensive list of 100 prominent stocks spanning various industries and market sectors. This extensive list ensures that users have the flexibility to explore a wide range of investment options and create diversified portfolios.
   * The inclusion of a large number of stocks in these tabs leverages the breadth of data available through the Yahoo Finance API, making the app versatile for different investment strategies.
2. LSTM Prediction Tab:
   * In the Price Predictions Using LSTM tab, users are restricted to selecting from only 8 stocks (AAPL, GOOGL, NKE, IBM, JNJ, KO, MSFT, and NFLX). This limitation exists because the LSTM models were pre-trained on these specific stocks as part of the project scope.
   * Training LSTM models is computationally intensive and requires significant resources, particularly for fine-tuning hyperparameters and handling large datasets. Thus, focusing on these 8 stocks allowed for a manageable and effective demonstration of the app’s predictive capabilities within the constraints of the project.
3. Scalability and Future Work:
   * Expanded Stock Coverage: The current architecture provides a strong foundation for scaling the LSTM prediction feature. Future work can involve training LSTM models for additional stocks from the 100-stock list, ensuring broader coverage for users.
   * Dynamic Model Training: Incorporating a pipeline to dynamically train models on user-selected stocks can make the app more versatile and reduce the need for pre-trained models.
   * Hybrid Models: Enhancements like integrating hybrid models (e.g., combining LSTM with ARIMA or XGBoost) could improve prediction accuracy and expand functionality.
   * Real-Time Data Integration: While predictions are based on historical data, integrating real-time data sources could further enhance the app's utility for active traders and investors.

This project demonstrates how advanced data science techniques can be applied to create an accessible tool for investors. By addressing current limitations and building upon the existing framework, the app has significant potential for future improvements and scalability. These enhancements will enable the app to cater to a wider audience and provide even greater value in the rapidly evolving financial markets.

6. CONCLUSIONS

This project, *Stock Prediction and Portfolio Optimization App*, aimed to address the challenges individual investors face in navigating the complexities of stock market investing by providing a comprehensive, data-driven tool for stock exploration, price prediction, and portfolio optimization. Through the integration of advanced machine learning techniques, financial modeling, and interactive app development, the project successfully delivered a robust solution.

**Summary of the Project**

The project followed a systematic approach to achieve its objectives:

1. Data Collection and Feature Engineering:
   * Historical stock data was collected from Yahoo Finance.
   * Various technical indicators, such as Moving Averages, MACD, Bollinger Bands, and RSI, were engineered to enhance model performance.
   * Additional features, such as market index data and date-related variables, provided context for broader market conditions.
2. Machine Learning Models for Stock Prediction:
   * Three models—ARIMA, XGBoost, and LSTM—were implemented and evaluated for their predictive capabilities.
   * LSTM emerged as the best-performing model, demonstrating its ability to capture sequential dependencies in time-series data.
   * The models were evaluated and compared using metrics such as RMSE, MAE, and MSE, and visualizations to understand how well they capture the time series data, providing a foundation for selecting LSTM for the app's prediction feature.
3. Monte Carlo Simulations for Portfolio Optimization:
   * A Monte Carlo simulation algorithm was implemented to identify optimal portfolio allocations by maximizing the Sharpe Ratio.
   * The simulation evaluated thousands of random portfolio configurations, providing actionable recommendations for risk-adjusted investment strategies.
4. App Development and Deployment:
   * A Streamlit-based app was developed with four tabs: About This App, Stock Exploration, Price Predictions Using LSTM, and Portfolio Allocation Optimization.
   * The app integrates all components of the project, providing an interactive, user-friendly platform.
   * The app was deployed to Streamlit Cloud, with pre-trained LSTM models hosted in a connected GitHub repository.

**Tying the App to Project Objectives**

The final app meets the project’s objectives:

1. Enhancing Stock Analysis:
   * Users can explore historical trends, technical indicators, and detailed company information for 100 prominent stocks.
   * The Stock Exploration tab provides visualizations of moving averages and metrics for informed analysis.
2. Providing Accurate Predictions:
   * The LSTM-based prediction feature enables users to forecast stock prices for the next 1 to 30 days, supporting data-driven decision-making.
3. Optimizing Portfolio Allocation:
   * Monte Carlo simulations recommend optimal portfolio allocations by balancing risk and return, aligning with individual financial goals.
4. Simplifying Complex Analytics:
   * The app’s intuitive interface bridges the gap between advanced analytics and user accessibility, making complex financial concepts easy to understand.

**Strengths**

* Comprehensive Functionality: The app combines three critical components—exploration, prediction, and optimization—into one seamless tool.
* Advanced Methodologies: The use of LSTM for predictions and Monte Carlo simulations for portfolio optimization demonstrates the integration of state-of-the-art techniques.
* User-Centric Design: The app’s interface is intuitive, ensuring that even non-expert users can interact with advanced analytics effortlessly.
* Scalability: The modular design allows for future expansion, such as training models for additional stocks or incorporating new financial metrics.

**Weaknesses**

* Limited Prediction Scope: The LSTM model is limited to 8 pre-trained stocks, restricting user flexibility for price prediction.
* Reliance on Historical Data: Predictions and portfolio recommendations are based solely on historical data, which may not account for unforeseen market events.
* Static Features: While robust, the current app lacks real-time data integration and dynamic model training for user-selected stocks.

**Future Work**

* Expanding Prediction Coverage: Training LSTM models for additional stocks will provide broader prediction capabilities.
* Real-Time Data Integration: Incorporating real-time market data will enhance the app’s utility for active traders.
* Enhanced Prediction Models: Future iterations can explore hybrid models or ensemble techniques for improved forecasting accuracy.
* Dynamic Model Training: Enabling users to train models for custom-selected stocks dynamically will increase the app’s versatility.
* New Financial Metrics: Adding advanced indicators and metrics can provide deeper insights for stock analysis and portfolio optimization.

The Stock Prediction and Portfolio Optimization App marks a significant step toward democratizing financial analytics by providing a comprehensive yet accessible tool for individual investors. While it successfully addresses the project’s objectives, its modular structure lays the groundwork for future scalability and improvement, ensuring continued relevance in an ever-evolving financial landscape. This project not only bridges the gap between advanced analytics and accessibility but also stands as a testament to the power of data-driven solutions in tackling real-world challenges.

Deliverables:

[Stock Prediction and Portfolio Optimization App Link](https://stock-forecast-and-allocate.streamlit.app/)

[GitHub Repository Link](https://github.com/davidhellerw/DS-Capstone/tree/main)

7. REFERENCES

AdvisoryHQ. (2024). How much do financial advisors charge? Retrieved from <https://www.advisoryhq.com>

Bloomberg L.P. (2024). Bloomberg Terminal pricing and features. Retrieved from <https://www.bloomberg.com>

MetaQuotes. (2024). MetaTrader 4/5 platform overview and pricing. Retrieved from <https://www.metaquotes.net>

NerdWallet. (2024). What are financial advisor fees? Retrieved from <https://www.nerdwallet.com>

Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice (2nd ed.). OTexts. Retrieved from <https://otexts.com/fpp3/index.html>

Murphy, J. J. (1999). Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications (2nd ed.). New York: New York Institute of Finance.

Achelis, S. B. (2000). Technical Analysis from A to Z (2nd ed.). New York: McGraw-Hill

Bollinger, J. (2001). Bollinger on Bollinger Bands. New York: McGraw-Hill.

Appel, G. (2005). Technical Analysis: Power Tools for Active Investors. New York: Financial Times Press.

Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2015). Introduction to Time Series Analysis and Forecasting. Hoboken, NJ: Wiley.

Wilder, J. W. (1978). New Concepts in Technical Trading Systems. Greensboro: Trend Research.

Lambert, D. (1980). Commodity Channel Index (CCI): Tools and Tactics. Commodities Magazine.

Lane, G. C. (1984). Lane’s Stochastic Oscillator Methodology. Stocks & Commodities Magazine.

Investopedia. (n.d.). Moving Averages. Retrieved from <https://www.investopedia.com/>

Box, G. E. P., & Jenkins, G. M. (1970). Time Series Analysis: Forecasting and Control.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5), 1189–1232.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780. <https://doi.org/10.1162/neco.1997.9.8.1735>

Chollet, F. (2018). Deep Learning with Python. Manning Publications.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. <https://arxiv.org/abs/1412.6980>

Prechelt, L. (1998). Early stopping — but when? In Neural Networks: Tricks of the Trade (pp. 55–69). Springer. <https://link.springer.com/chapter/10.1007/3-540-49430-8_3>

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Graves, A. (2012). Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence.

Glasserman, P. (2003). Monte Carlo Methods in Financial Engineering. Springer.

Markowitz, H. (1952). Portfolio Selection. The Journal of Finance, 7(1), 77–91.

Sharpe, W. F. (1966). Mutual Fund Performance. The Journal of Business, 39(1), 119–138.

Rao, S., et al. (2020). Streamlit Documentation.   
Retrieved from <https://streamlit.io>.

Huang, J., et al. (2022). "yfinance: A Python library for financial data." Journal of Finance, 3(2), 112-121.

Chen, H. (2018). "Moving Averages in Financial Markets: A Comprehensive Review." Financial Analysis Quarterly, 7(3), 89-102.

8. APPENDIX

**8.1 Performance Metrics and Visualizations for All Models and All Stocks**

This section provides the performance metrics for the three models—ARIMA, XGBoost, and LSTM—across all eight stocks. Each stock's results are presented with their corresponding metrics table and visualizations to facilitate an understanding of how each model performed.

Table 1: Performance Metrics for All Models for All Stocks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stock** | **Model** | **MSE** | **RMSE** | **MAE** |
| **Apple (AAPL)** | ARIMA | 35.29 | 5.94 | 4.73 |
|  | XGBoost | 968.94 | 31.13 | 23.55 |
|  | LSTM | 191.28 | 13.83 | 11.03 |
| **Google (GOOGL)** | ARIMA | 51.80 | 7.20 | 5.36 |
|  | XGBoost | 150.96 | 12.29 | 6.88 |
|  | LSTM | 59.05 | 7.68 | 5.82 |
| **IBM** | ARIMA | 453.58 | 21.30 | 17.99 |
|  | XGBoost | 946.31 | 30.76 | 18.58 |
|  | LSTM | 141.02 | 11.88 | 7.54 |
| **Microsoft (MSFT)** | ARIMA | 90.48 | 9.51 | 8.16 |
|  | XGBoost | 4829.39 | 69.49 | 47.58 |
|  | LSTM | 834.69 | 28.89 | 22.16 |
| **Netflix (NFLX)** | ARIMA | 2328.19 | 48.25 | 34.63 |
|  | XGBoost | 904.39 | 30.07 | 15.58 |
|  | LSTM | 824.01 | 28.71 | 21.27 |
| **Nike (NKE)** | ARIMA | 15.77 | 3.97 | 3.25 |
|  | XGBoost | 3.14 | 1.77 | 1.36 |
|  | LSTM | 17.29 | 4.16 | 3.14 |
| **Johnson & Johnson (JNJ)** | ARIMA | 12.96 | 3.60 | 2.90 |
|  | XGBoost | 64.15 | 8.01 | 7.33 |
|  | LSTM | 26.78 | 5.17 | 4.45 |
| **Coca Cola (KO)** | ARIMA | 10.40 | 3.22 | 2.33 |
|  | XGBoost | 65.61 | 8.10 | 7.04 |
|  | LSTM | 1.45 | 1.21 | 0.97 |

It is important to note here that while these metrics are very useful, they are average values. In time series analysis, we are more interested in how the model captures the short- and long-term trends and seasonality of the time series. Therefore, although some models have better metrics than LSTM, if we see the graphs, LSTM outperformed the other models in capturing the time series movements and patterns (across all stocks), which is why it is the best-performing model for the task of predicting stock prices.

**Graphs for All Models by Stock:**

**AAPL**

ARIMA:

A graph with a line going up

Description automatically generated

XGBoost:

A graph of a stock price

Description automatically generated

LSTM:

A graph showing a price

Description automatically generated with medium confidence

**GOOGL**

ARIMA:

A graph of a line graph

Description automatically generated with medium confidence

XGBoost:

A graph showing a price

Description automatically generated with medium confidence

LSTM:

A graph showing a price

Description automatically generated with medium confidence

**IBM**

ARIMA:

A graph with a line going up

Description automatically generated

XGBoost:

A graph of a stock price

Description automatically generated

LSTM:

A graph showing a price

Description automatically generated with medium confidence

**MSFT**

ARIMA:

A graph with a line going up

Description automatically generated

XGBoost:

A graph showing the price of a stock price

Description automatically generated

LSTM:

A graph showing a line graph

Description automatically generated with medium confidence

**NFLX**

ARIMA:

A graph of a line graph

Description automatically generated with medium confidence

XGBoost:

A graph showing a line graph

Description automatically generated with medium confidence

LSTM:

A graph showing a line graph

Description automatically generated with medium confidence

**NKE**

ARIMA:

A graph of a line

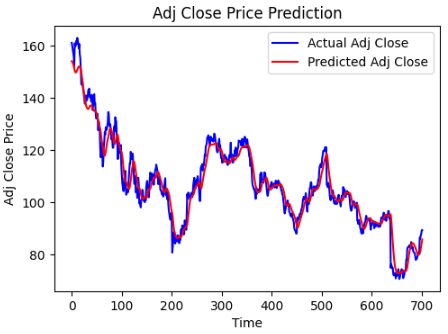
Description automatically generated with medium confidence

XGBoost:

A graph showing a price

Description automatically generated with medium confidence

LSTM:



**JNJ**

ARIMA:

A graph with lines and numbers

Description automatically generated

XGBoost:

A graph of a stock price

Description automatically generated

LSTM:

A graph showing the price of a stock market

Description automatically generated

**KO**

ARIMA:

A graph with lines and numbers

Description automatically generated

XGBoost:

A graph of stock prices

Description automatically generated

LSTM:

A graph showing the price of a stock market

Description automatically generated