Stock Market Price Prediction using Time Series Models

IST652: Scripting for Data Analysis

Dylan Phillips and Dhrumil Shreyans Shah

Purpose

The purpose of this project is to create the framework to efficiently predict future stock prices using historical financial data in Python. In the finance sector, the ability to predict price movements of specific stocks is critical for providing insights to investors and analysts. By creating a tool that can accurately predict stock price movements, both individuals and large organizations focusing on wealth management can make informed decisions using data analysis to optimize investment strategies and mitigate risk.

Approach

This project utilizes Python to perform stock price prediction, leveraging historical financial data and various machine learning models. The function involves collecting daily stock data from Yahoo! Finance, followed by data cleaning and preprocessing for modeling. Three time series models are used; ARIMA, LSTM, and Prophet. These models are trained on past stock prices, macroeconomic indicators, and financial indicators to make accurate predictions about the short term outlook of a given stock. The goal is to predict stock prices for future periods and allow the user to make decisions on which model to use, based on error metrics. The function is designed to not need any machine learning experience to make predictions. Six hand-selected stocks from different sectors will be used to show how the different modeling approaches work.

Data Collection

The only source of data is from Yahoo! Finance. Using the yfinance module in Python, the daily stock prices for the requested time period are collected and loaded into a clean dataframe. The stock price includes open, close, high, and low prices for each day for the stock. It also includes trading volume. A second dataset is loaded using the same time period, this time

finding the 10-year treasury rate for bonds. This acts as a macroeconomic indicator, and is used by stock market analysts to measure the relative returns of investing in the stock market.

The function then calculates additional indicators that will be used by the prediction models. The simple moving average, which is the mean value of the closing price of the stock over a period of time, is calculated. Using this calculation, Bollinger Bands are produced, which is the range in which the stock typically lies. Finally, the RSI, or relative strength index, is used to find when a stock is oversold or overbought. Along with the raw Yahoo! Finance data, these indicators are critical in predicting stock movements.

Additionally, before each modeling technique, the data may be slightly manipulated, solely for the purpose of model requirements. This may involve standardizing data or changing how data is stored.

Prediction Models and Methods

To predict future stock price movement, three models were employed, which combined both traditional time series forecasting with machine learning methods, specifically the addition of exogenous variables.

The first model, ARIMA, or AutoRegressive Integrated Moving Average, is a statistical model that captures autocorrelation in time series data. Autocorrelation occurs when there is a correlation between data over time. Because we are introducing exogenous variables, we will run an ARIMAX model, which uses the added variables, along with price, to create a prediction.

The second model is LSTM, or Long Short-Term Memory. LSTM is a type of deep-learning neural network that is designed to handle sequential, time series data. LSTM networks can capture long-term dependencies in data, which is a common disadvantage of other modelling techniques. LSTM is particularly useful when data is non-linear.

The final model used for prediction is Prophet, which is a forecasting tool developed by Facebook (Meta). It is a robust model that was written to achieve many results, but is well-suited for times series data with strong seasonal effects and historical trends. A common pitfall with Prophet is it cannot calculate seasonal effects, it must be fed into the model. In our analysis, we do not assume any seasonality.

Model Features

The reasoning for adding exogenous variables to the models was to help the models create predictions on metrics outside of the previous prices. Left as is, the models will only use the autocorrelation between terms for prediction. Adding outside factors can help explain some of the variation in stock price movements.

The 10-year treasury bond yield, set by the federal government, is a common metric used by investors. This yield is used to calculate the risk-free rate, or the return that would be expected if you invested money in bonds over stocks. When this risk-free rate is high, the federal government may sense a bearish trend in the markets. Measuring this yield is important for investment outlook and predictions.

Two common financial indicators are also included in the models; the relative strength index and the simple moving average. The relative strength index is a measure of how well the stock does in relation to its current price. If the RSI is greater than 70, the stock can be considered overbought, and it is seen as priced too high. At an RSI of 30 or less, the stock is seen as oversold and priced too low. Investment strategy often is focused around the RSI value. The simple moving average is the mean of the prices over a window, and is used to ‘smooth’ the price movements. The window used to calculate this depends on investment strategy. Shorter

moving averages (20 days) are used for short term investments, while long term investors will use up to 200 days to calculate moving averages.

Function Overview

The function is called using stock_analysis. The only variable required for analysis is the New York Stock Exchange Ticker. The other variables can be edited, but the default values are listed in the function definition. The variable start, with the default set at '2020-01-01', is the first day in which data will be fetched from Yahoo! Finance. The variable end is the last day in which data will be fetched from Yahoo! Finance. The default value is the system's current date. The variables ma_window and rsi_window are the length, in days, that the simple moving average and the relative strength index will be calculated, respectively. The stock_preview variable gives the user options for what will be displayed. If this is set to false, the function will jump straight to making predictions using the model type set by model_type. Finally, forecast_periods can be set to a number greater than 1, but the function works best for next-day predictions. Non-important warnings will be filtered out for clean printing.

```
def stock_analysis(ticker, start='2020-01-01', end=date.today(), ma_window=20, rsi_window=14, stock_preview=True, forecast_periods=1, model_type='ALL'):

    warnings.filterwarnings("ignore")
```

The function has been modularized for ease of understanding, and for higher customizability by the user. The first function imbedded in the stock_analysis function runs as follows:

```
def get_stock_data(ticker, start, end):

    # download stock and bond data from yahoo finance, clean data
    stocks = yf.download(ticker, start=start, end=end, auto_adjust=True)
    bonds = yf.download("^TNX", start=start, end=end)

    stocks.columns = ['Close', 'High', 'Low', 'Open', 'Volume']
    bonds = bonds.rename(columns={'Close': '10Y_Yield'})

    # only keep the 10Y yield column to merge
    bonds = bonds[['10Y_Yield']]

    # align frequencies with forward fill to match stock market days
    bonds = bonds.reindex(stocks.index, method='ffill')
    bonds.columns = [col for col in bonds.columns]

    return stocks, bonds
```

Using the ticker, start date, and end date specified in the function call, daily price data will be pulled from Yahoo! Finance using the yfinance module in Python. The stock data includes the opening, closing, high, and low prices for the day, as well as total volume. The bonds data, also from Yahoo! Finance, finds the 10-year bond yield, set by the U.S. Government, and acts as a macroeconomic indicator for price predictions. The indexes for both datasets are matched using forward fill.

The next embedded function is the summary_statistics function. It will print a table that contains summary statistics for the stock data (minimum, maximum, mean, etc.) It will not be printed if stock_preview is set to false. Below is the function definition and the resulting output:

```python
def summary_statistics(stocks, ticker):

    # dataframe information and summary statistics
    print(f"\nNumber of data points collected: {len(stocks)}")
    print(f"\nSummary Statistics of {ticker} data")
    print(stocks.describe(),f"\n")
```

```
Summary Statistics of AMZN data
              Close         High          Low         Open        Volume
count  1336.000000  1336.000000  1336.000000  1336.000000  1.336000e+03
mean    150.323424   152.215247   148.352294   150.344249  6.717423e+07
std      34.646990    34.834483    34.435828    34.702311  3.377853e+07
min      81.820000    83.480003    81.301498    82.075500  1.500750e+07
25%     122.942499   124.400002   120.982376   122.870001  4.460605e+07
50%     155.816002   158.263000   154.244255   156.200500  5.910345e+07
75%     173.213745   174.948627   171.581871   173.547501  7.997700e+07
max     242.059998   242.520004   238.029999   239.020004  3.113460e+08
```

Next, closing_price_chart is the first visualization of the data. It will print a line graph of the closing price over the entire dataset. It will not be printed if stock_preview is set to false.

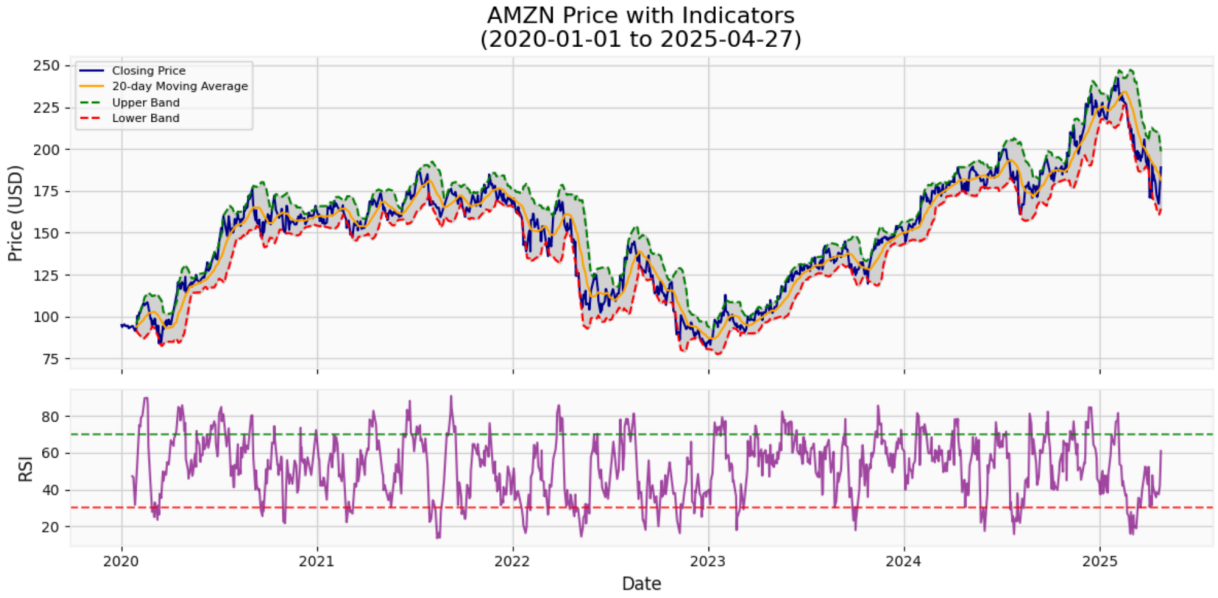
```python
def closing_price_chart(stocks, ticker, start, end):

    plt.figure(figsize=(12, 6))
    plt.plot(stocks.index, stocks['Close'], color='darkblue')
    plt.title(f'{ticker} Closing Price Over Time \n({start} to {end})', fontsize=16)
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Price (USD)', fontsize=12)
    plt.grid(True)
    plt.show()
```

The indicator_calculations function is a necessary function for predicting prices. This function finds the simple moving average, with the window set by ma_window. It then finds the upper and lower Bollinger bands, which is the moving average, plus and minus two times the standard deviation of the closing price of the stock. Finally, it calculated the relative strength index of the stock, which is an indicator of a stock being overbought, and priced too high, when the RSI is over 70, and oversold when it is less than 30. There is no visualization for this function.

```
def indicator_calculations(stocks, ma_window, rsi_window):

    stocks['MA'] = stocks['Close'].rolling(window=ma_window).mean()
    stocks['STD'] = stocks['Close'].rolling(window=ma_window).std()

    stocks['Upper Band'] = stocks['MA'] + (stocks['STD'] * 2)
    stocks['Lower Band'] = stocks['MA'] - (stocks['STD'] * 2)

    stocks['delta'] = stocks['Close'].diff()
    stocks['gain'] = (stocks['delta'].where(stocks['delta'] > 0, 0)).rolling(window=rsi_window).mean()
    stocks['loss'] = (-stocks['delta'].where(stocks['delta'] < 0, 0)).rolling(window=rsi_window).mean()
    stocks['rs'] = stocks['gain'] / stocks['loss']
    stocks['rsi'] = 100 - (100 / (1 + stocks['rs']))

    stocks.drop(columns=['delta', 'gain', 'loss', 'rs'], inplace=True)

    return stocks
```

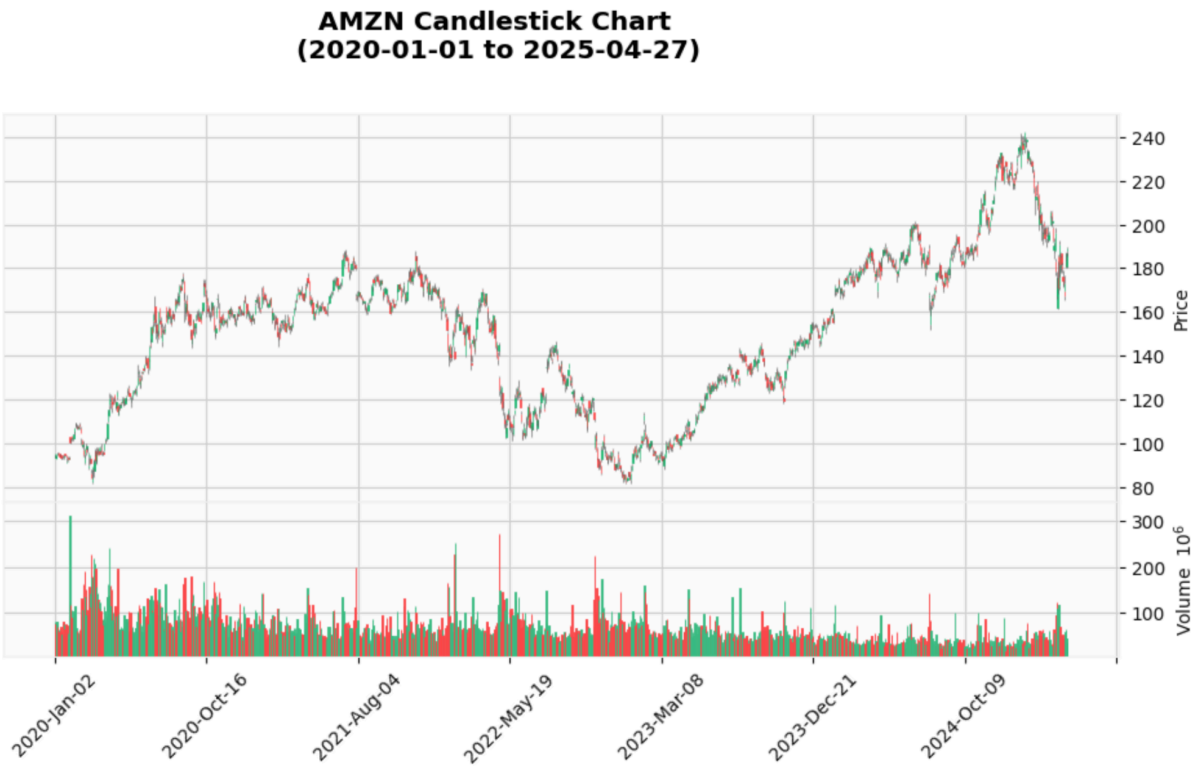
The second visualization is plot_with_indicators, which is a line plot of closing price over time with the indicators we calculated in the previous function. It will not be printed if stock_preview is false.

```
def plot_with_indicators(stocks, ticker, ma_window, start, end):

    # Plot the closing price, moving average, and Bollinger Bands
    fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(12, 6), sharex=True, gridspec_kw={'height_ratios': [2, 1]})
    ax1.plot(stocks.index, stocks['Close'], color='darkblue', label='Closing Price')

    if 'MA' in stocks:
        ax1.plot(stocks.index, stocks['MA'], color='orange', label=f'{ma_window}-day Moving Average')

    if 'Upper Band' in stocks and 'Lower Band' in stocks:
        ax1.plot(stocks.index, stocks['Upper Band'], color='green', linestyle='--', label='Upper Band')
        ax1.plot(stocks.index, stocks['Lower Band'], color='red', linestyle='--', label='Lower Band')
        ax1.fill_between(stocks.index, stocks['Lower Band'], stocks['Upper Band'], color='gray', alpha=0.3)
        ax1.set_title(f'{ticker} Price with Indicators\n({start} to {end})', fontsize=16)
        ax1.set_ylabel('Price (USD)', fontsize=12)
        ax1.grid(True)
        ax1.legend(loc='best', fontsize = 8)

    if 'rsi' in stocks:
        ax2.plot(stocks.index, stocks['rsi'], color='purple', alpha=0.7)
        ax2.axhline(30, color='red', linestyle='--', alpha=0.7)
        ax2.axhline(70, color='green', linestyle='--', alpha=0.7)
        ax2.set_xlabel('Date', fontsize=12)
        ax2.set_ylabel('RSI', fontsize=12)
        ax2.grid(True)
        plt.tight_layout()
        plt.show()
```



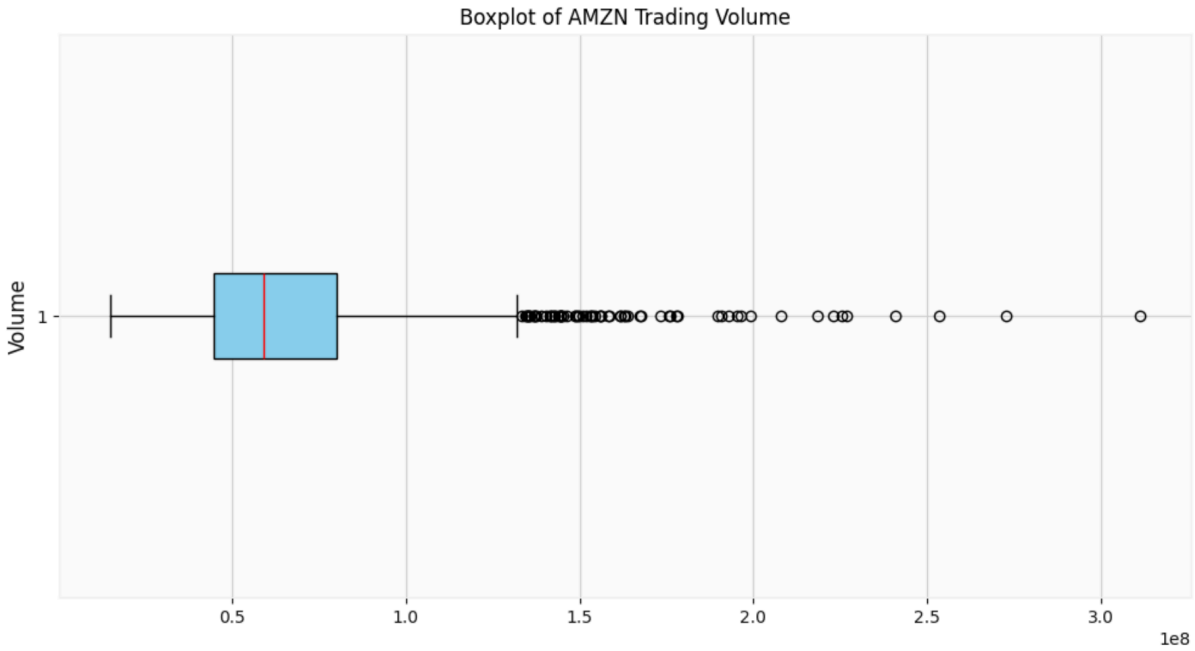
The candlestick_chart function will produce a candlestick chart for the full data. Candlestick charts are helpful for investors, as it shows price movement throughout the day. It also shows days where there were volume spikes. No plot will be shown if stock_preview is set to false.

```
def candlestick_chart(stocks, ticker, start, end):
    # Stock Volatility Analysis (Candlestick Chart)
    mpf.plot(stocks,
             type='candle',
             style='yahoo',
             volume=True,
             warn_too_much_data= len(stocks) + 1,
             title = f"{ticker} Candlestick Chart \n({start} to {end})",
             figsize = (12,6))
```



Outliers of both daily returns and trading volume are found with the return_outlier_detection and the volume_outlier_detection functions. These will produce boxplots of the respective data, as well as show how many and which data points are considered outliers. These visuals will not be printed if stock_preview is set to false. Below is the definition of volume_outlier_detection, with the resulting output:

```python
def volume_outlier_detection(stocks, ticker):

    #Find outliers in trading volume
    plt.figure(figsize=(12,6))
    plt.boxplot(x = stocks['Volume'].dropna(),
                vert=False,
                patch_artist=True,
                boxprops=dict(facecolor='skyblue'),
                medianprops=dict(color='red'))
    plt.title(f'Boxplot of {ticker} Trading Volume')
    plt.ylabel('Volume', fontsize=12)
    plt.grid(True)
    plt.show()

    Q1 = stocks['Volume'].quantile(0.25)
    Q3 = stocks['Volume'].quantile(0.75)
    IQR = Q3 - Q1

    # Define outlier bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    lower_outliers = stocks[stocks['Volume'] < lower_bound]
    upper_outliers = stocks[stocks['Volume'] > upper_bound]

    top_spikes = stocks.sort_values(by='Volume', ascending=False).head(5)

    # Print outlier dates and values
    print(f'\n{ticker} has {len(upper_outliers)} volume spikes. The top 5 spikes are:\n')
    print(top_spikes[['Volume']])
```




```
AMZN has 67 volume spikes. The top 5 spikes are:

              Volume
Date
2020-01-31  311346000
2022-04-29  272662000
2022-02-04  253456000
2020-04-16  240764000
2020-03-12  226924000
```

The next function, titled data_merge, simply merges the stock and bond data together, ensuring the indexes match.

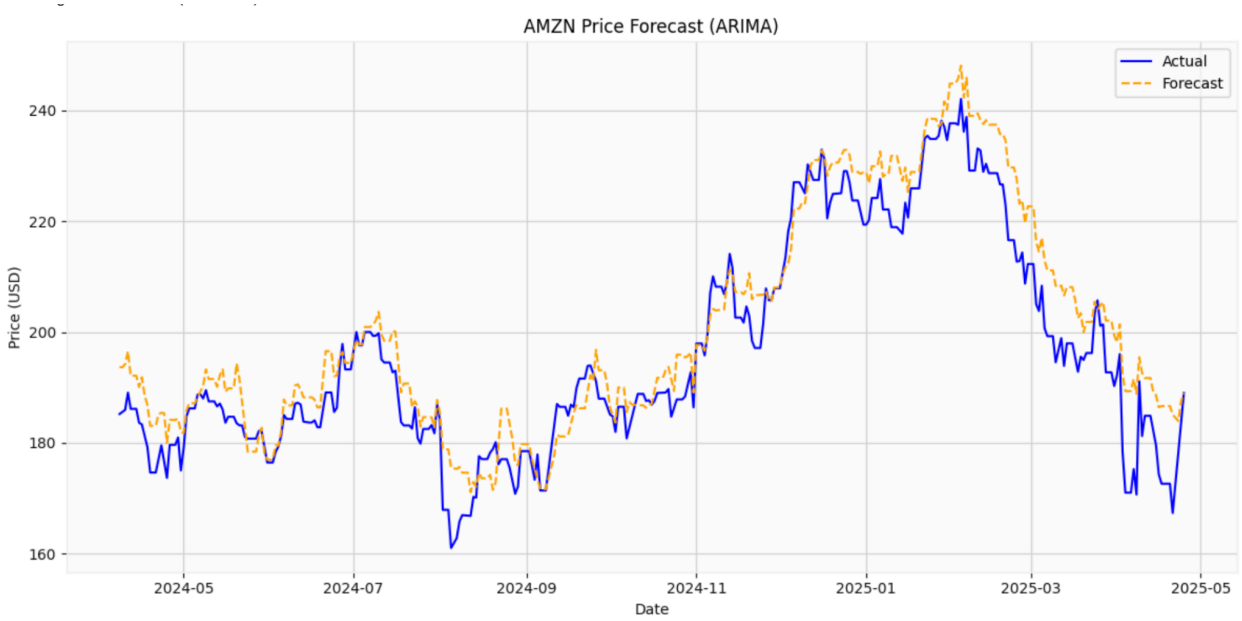
```python
def data_merge(stocks, bonds):

    # Merge stock data and bond yields on date
    merged_data = stocks.copy()
    merged_data = merged_data.merge(bonds, left_index=True, right_index=True)
    merged_data.columns = merged_data.columns[:-1].tolist() + ['bond_yield']
    merged_data.index = pd.to_datetime(merged_data.index)  # Ensure it's a datetime index
    merged_data = merged_data.asfreq('D', method='pad')

    # remove observations with missing MA and RSI values
    model_data = merged_data.dropna(subset=['MA', 'rsi'])

    return merged_data, model_data
```

In order to calculate accuracy in modeling, and to prevent overfitting, the data must be split into training and testing data. The train_test_split function does this, assigning the first 80% of the data, in terms of time, to training, and the last 20% to testing.

```
def train_test_split(model_data):

    y = model_data['Close']  # Dependent variable: Stock Price (Close)
    X = model_data[['MA', 'bond_yield', 'rsi']]  # Exogenous variables: Moving Average, Yield, RSI

    # create training (first 80%) and testing (last 20%)
    split_index = int(len(model_data) * 0.8)

    X_train = X.iloc[:split_index]
    X_test = X.iloc[split_index:]

    y_train = y.iloc[:split_index]
    y_test = y.iloc[split_index:]

    return y, X, split_index, X_train, X_test, y_train, y_test
```

The last three embedded functions are written to train time series models using the calculated indicators and closing price. The three models used are ARIMA, LSTM, and Prophet. They then plot the testing data and the models predicted values for that data. It will also forecast the next period price and display the model's error.

The function definitions are too long to display in a report, but they can all be viewed in the attached Python file. The following is an example visualization that is produced using ARIMA to predict stock prices for Amazon.




```
ARIMA predicted price for next 1 day(s): 188.99 (+0.00%)
    MAE: 5.4492
    RMSE: 6.6807
    MAPE: 2.8123
```

Results

For analysis, six large companies in different sectors were chosen. Apple, with the ticker AAPL, is a technology company known for the creation of the iPhone and other smart devices. It also is a major company in software. Nvidia, or NVDA, is also in the tech sector, but is more focused on computer parts, specifically GPUs. It is a major driver in computational power and artificial intelligence. Tesla (TSLA) is the popular electric automobile manufacturer, run by Elon Musk. Amazon (AMZN) is the online shopping website, but also created Amazon Web Services and is one of the largest companies in the world. Procter and Gamble (PG) represents the consumer goods sector, which produces many well known products, such as Tide and Crest. Finally, representing the banking sector is JPMorgan Chase (JPM). It is the largest bank in the United States by assets and is a major player in global banking.

To reduce redundancy in the results, a full demonstration will be shared for Apple, but only predictions will be displayed for the remaining selected stocks.

Apple (AAPL)

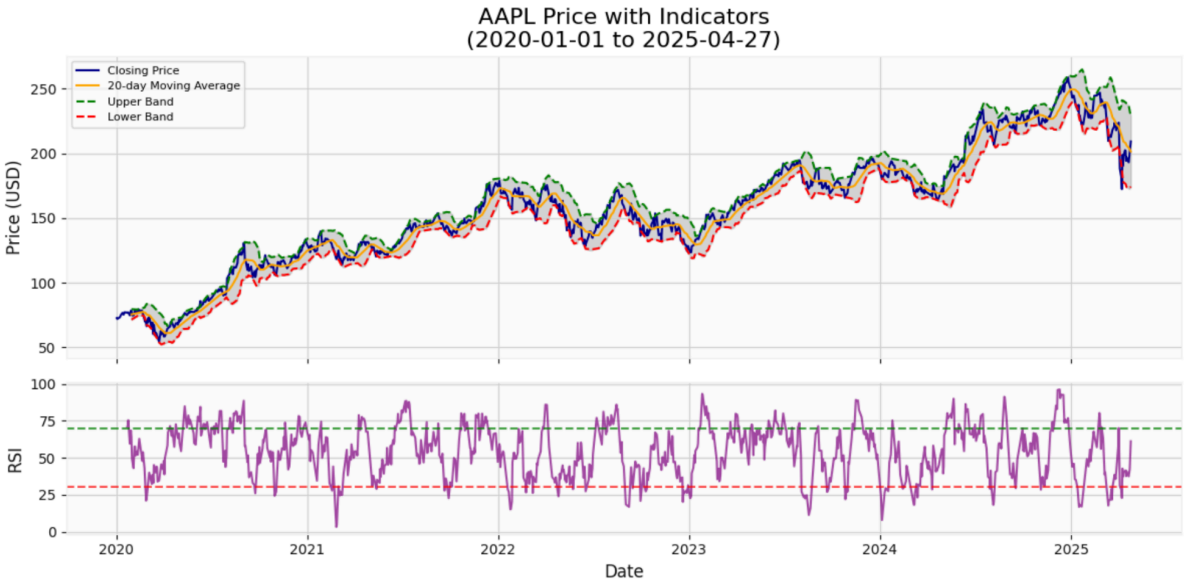
Function call: stock_analysis('AAPL')

Summary Statistics of AAPL data

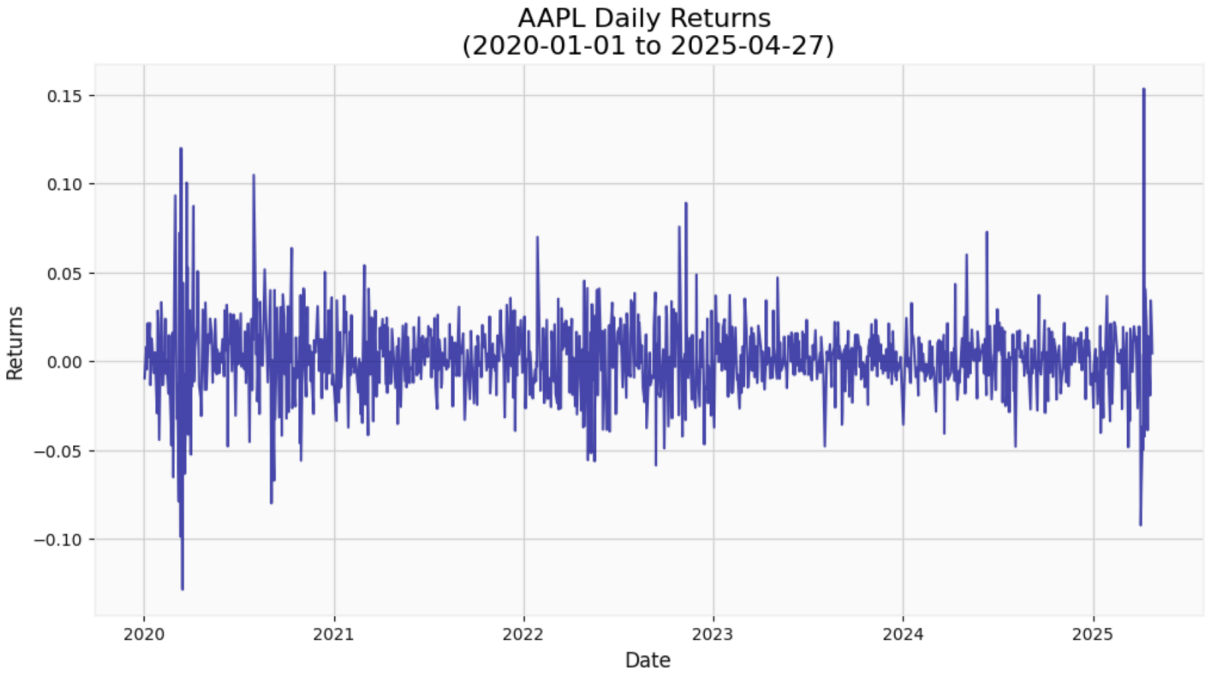
	Close	High	Low	Open	Volume
count	1336.000000	1336.000000	1336.000000	1336.000000	1.336000e+03
mean	156.380471	158.007931	154.572297	156.213239	8.877118e+07
std	44.376440	44.628581	44.045529	44.321073	5.263087e+07
min	54.449883	55.452148	51.595979	55.350223	2.323470e+07
25%	128.583504	129.896254	127.002689	128.710689	5.383952e+07
50%	154.408218	156.497783	152.016808	154.468582	7.479440e+07
75%	183.306355	184.914845	181.535597	182.877021	1.054614e+08
max	258.735504	259.814335	257.347047	257.906429	4.265100e+08

Focusing on Apple's closing price, represented by close in the above summary statistics table, we can see there are 1,336 samples in our data, and the average closing price of the period was $156.38, with a standard deviation of 44.38.

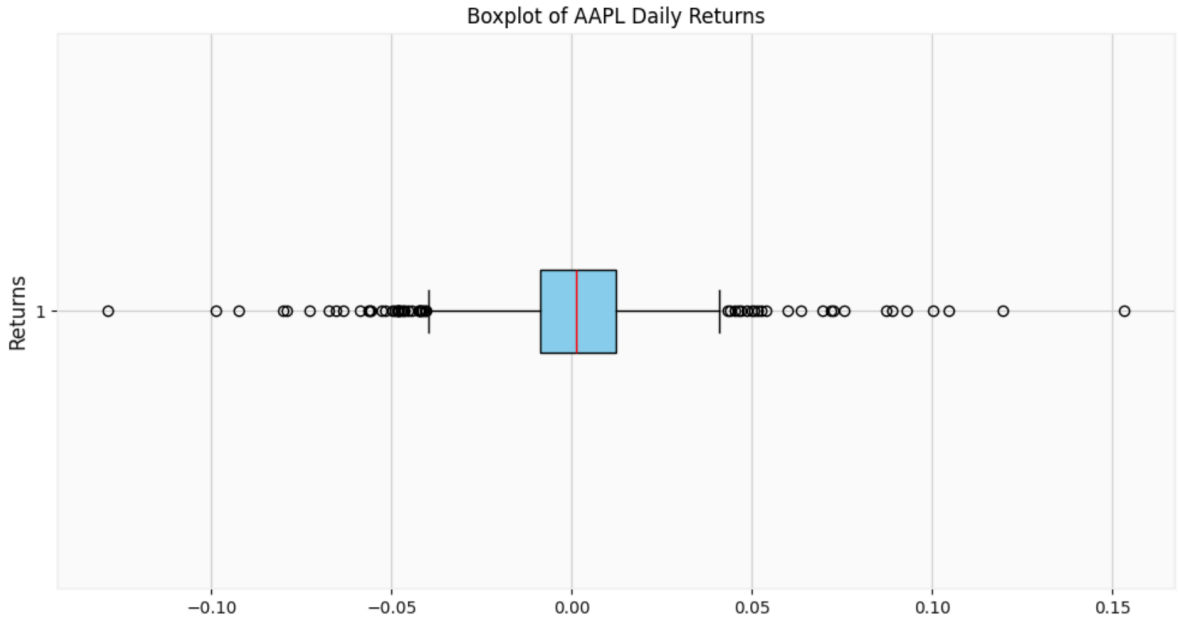
Over this period, Apple had a great run on the market. At the beginning of 2020, AAPL was selling at around $75. By the end of 2024, AAPL price rose to over $250, at its peak. Since then, it has fallen, mainly due to tariffs.



The lower portion of the candlestick chart shows volume. AAPL was being traded much more frequently in the early stages of our data than it is now, but there have been some volume spikes in recent months.

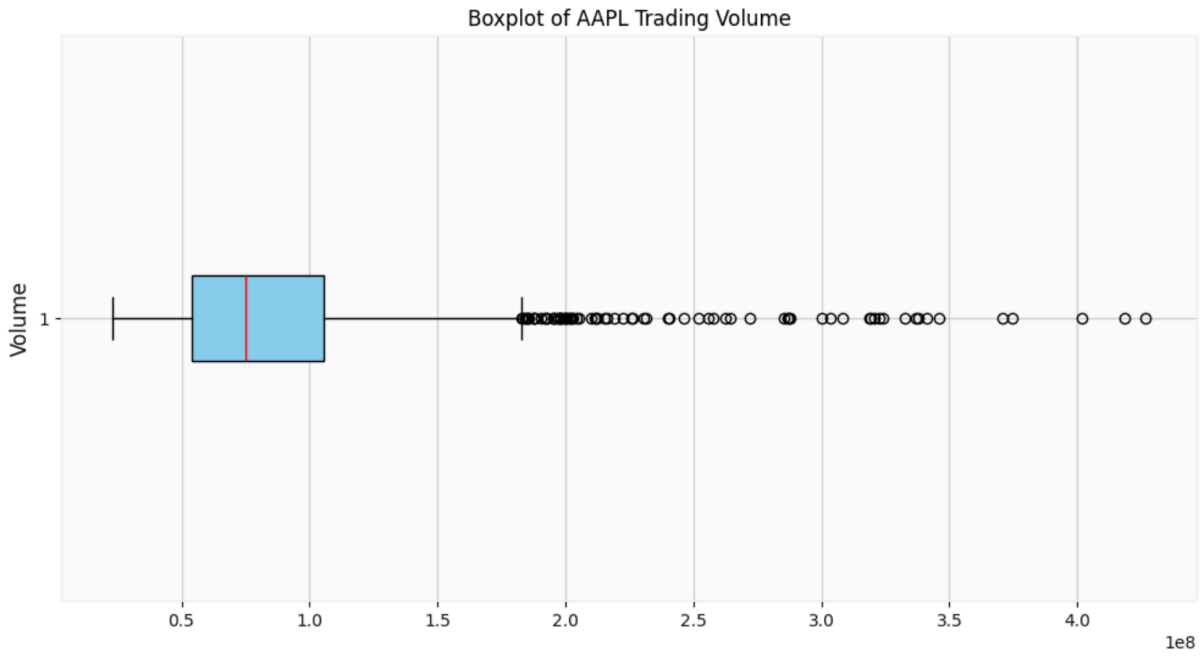


The daily returns chart shows relatively steady returns, but AAPL has some volatility at the beginning and end of the dataset, specifically around the COVID-19 pandemic and the tariffs put in place by President Trump.




```
AAPL has 24 high outliers in daily returns. The top 5 are

             Returns
Date
2025-04-09  0.153289
2020-03-13  0.119808
2020-07-31  0.104689
2020-03-24  0.100326
2020-03-02  0.093101

AAPL has 33 low outliers in daily returns. The top 5 are

             Returns
Date
2020-03-16 -0.128647
2020-03-12 -0.098755
2025-04-03 -0.092456
2020-09-03 -0.080061
2020-03-09 -0.079092
```

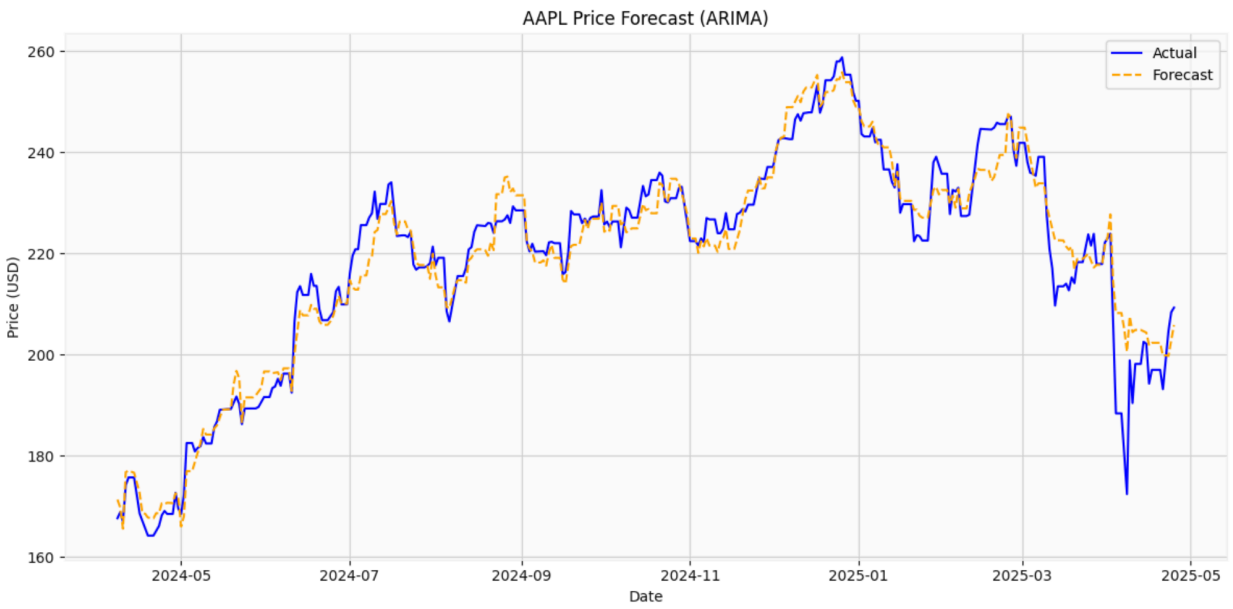
The volatility in AAPL returns is further displayed here, with 24 outliers on the high end, and 33 outliers on the low end. The day with the highest return was April 9, 2025. The day with the lowest return was March 16, 2020.



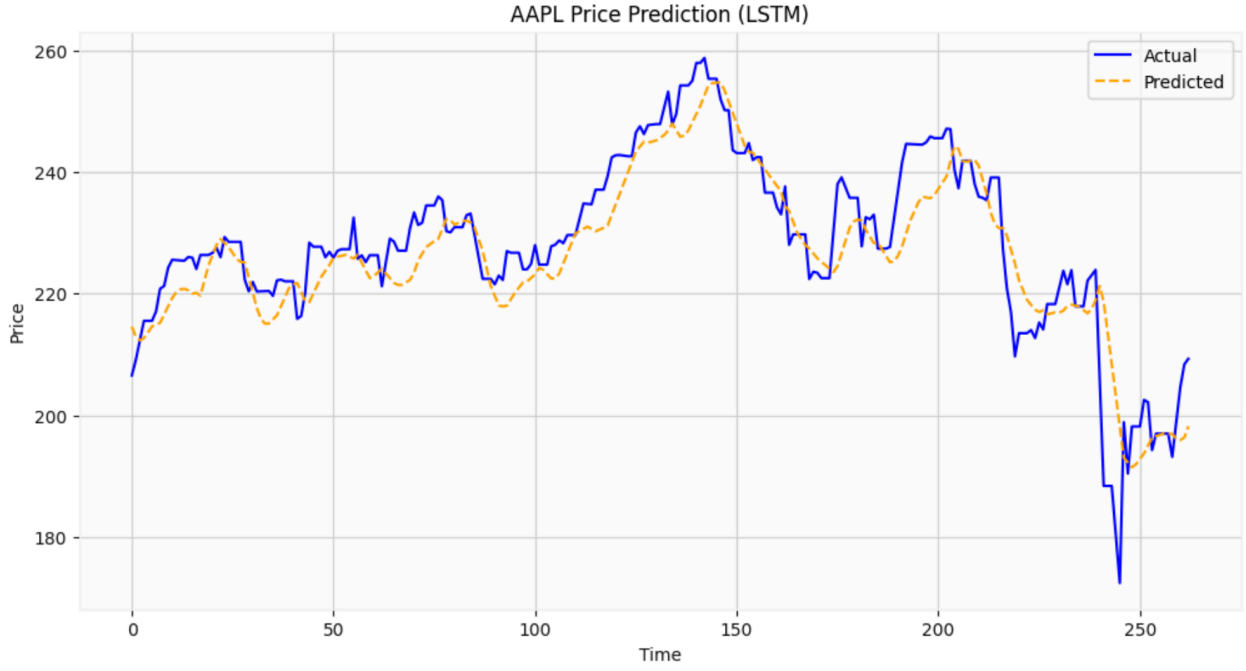
AAPL has 74 volume spikes. The top 5 spikes are:

Date	Volume
2020-02-28	426510000
2020-03-12	418474000
2020-03-20	401693200
2020-07-31	374336800
2020-03-13	370732000

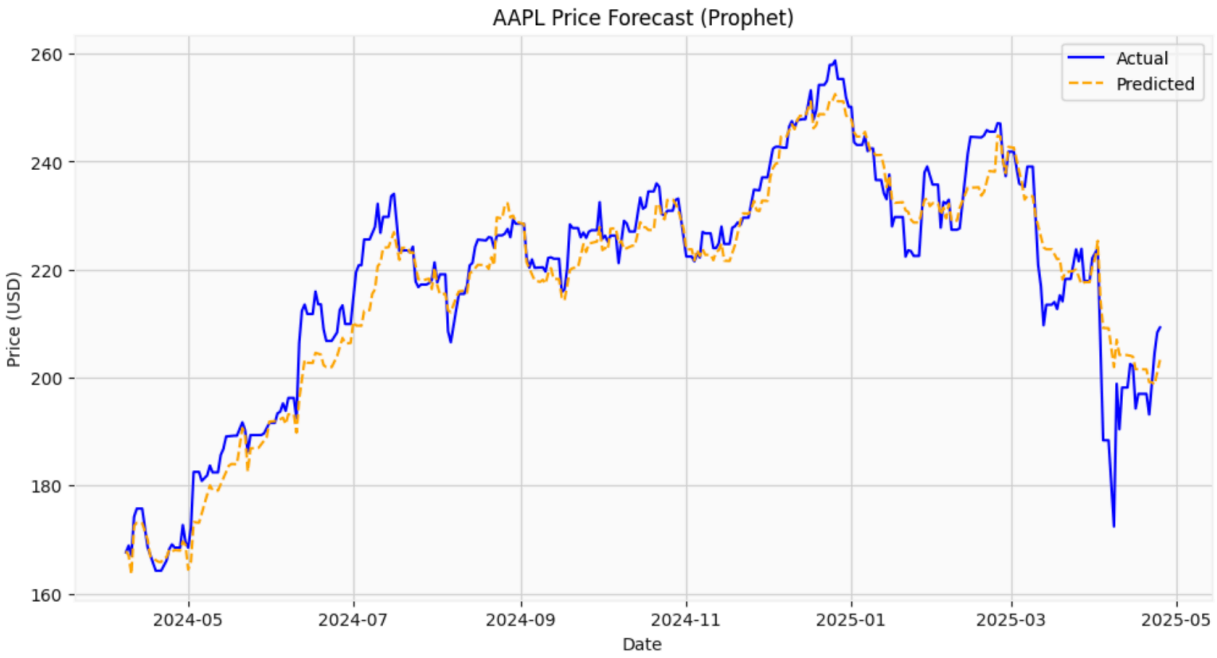
The outlier analysis identified 74 instances where trading volume spiked. The top five spikes all occurred at the beginning of our data.




```
ARIMA predicted price for next 1 day(s): 208.83 (-0.22%)
    MAE: 3.4217
    RMSE: 4.7555
    MAPE: 1.5932
```




```
LSTM Price Forecast:
Day 1: 201.25

LSTM predicted price for next 1 day(s): [201.24711421] ([-3.83834318]%)
LSTM MAE: 5.466580580065865
LSTM MAPE: 2.4481719886302638
LSTM RMSE: 6.988960606472151
```




```
Prophet predicted price for next day: 203.27 (-2.87%)
    MAE: 3.9958
    RMSE: 5.4869
```

Our first model, the ARIMA model, predicts a price of $208.83 for the next trading day, which is a drop of 0.22%. The mean absolute percentage error is 1.5932, which means the model is, on average, 1.59% away from the actual price.

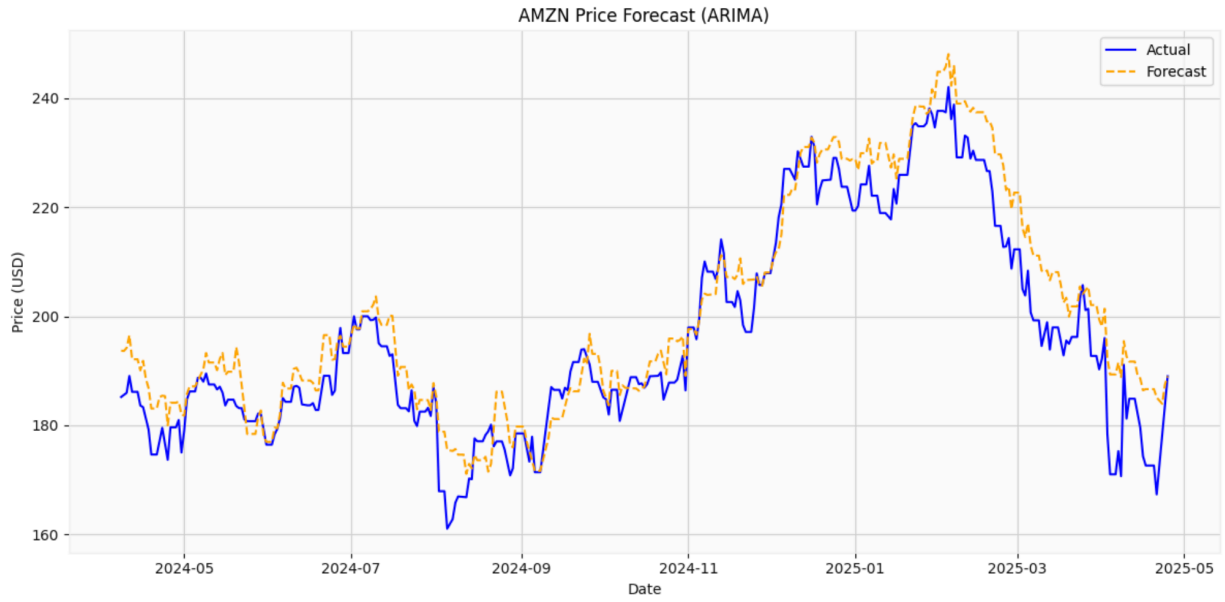
The LSTM model predicts a sharp decrease of -3.84% on the next trading day. The MAPE of this model is 2.45, which means, on average, it is 2.45% away from the actual price.

Prophet predicted AAPL to drop 2.87% on the next trading day, but its error rates are higher than that of the ARIMA model.

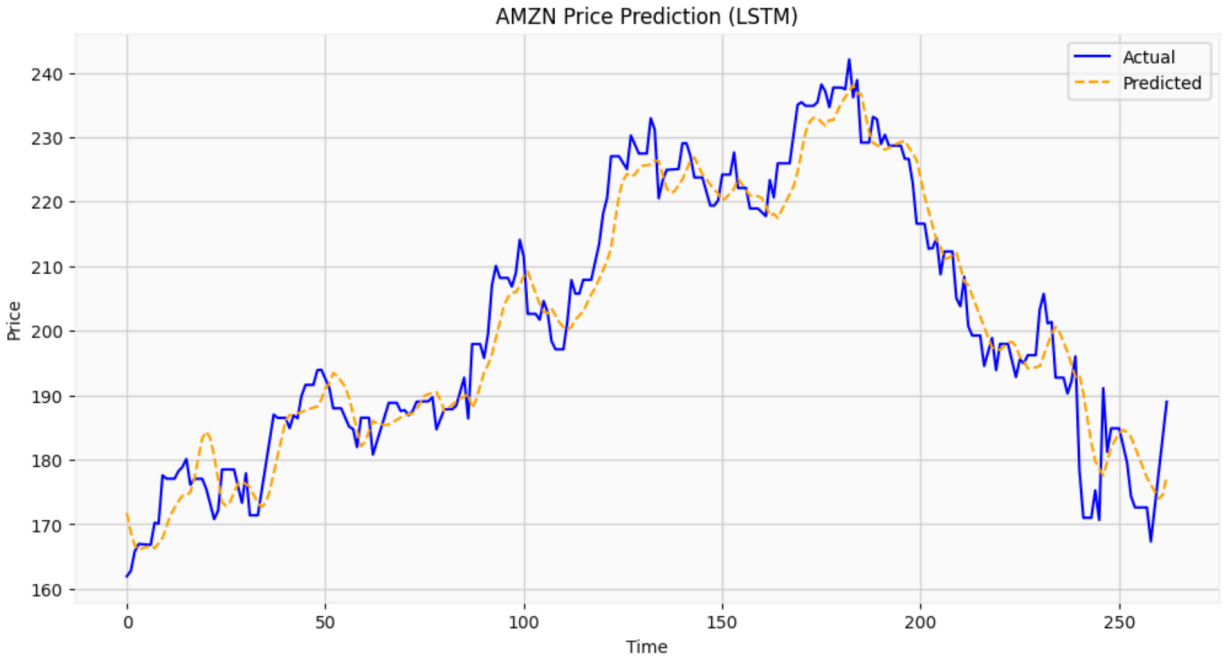
For Apple, the most accurate model was the ARIMA, with a MAE of 3.42, RMSE of 4.76, and a MAPE of 1.59.

Amazon (AMZN)

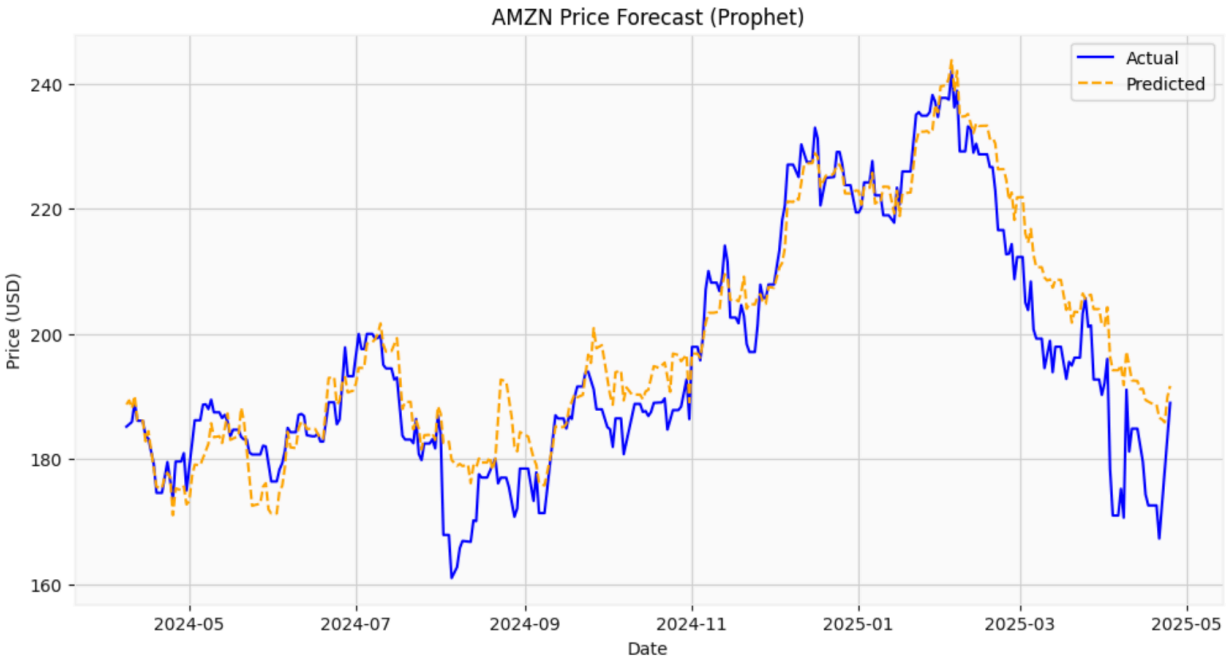
Function call: stock_analysis(‘AMZN’, stock_preview=False)




```
ARIMA predicted price for next 1 day(s): 188.99 (+0.00%)
    MAE: 5.4492
    RMSE: 6.6807
    MAPE: 2.8123
```




```
LSTM Price Forecast:
Day 1: 186.10

LSTM predicted price for next 1 day(s): [186.0977608] ([-1.53036913]%)
LSTM MAE: 3.6877165024676493
LSTM MAPE: 1.9203186525262996
LSTM RMSE: 4.869828937487975
```

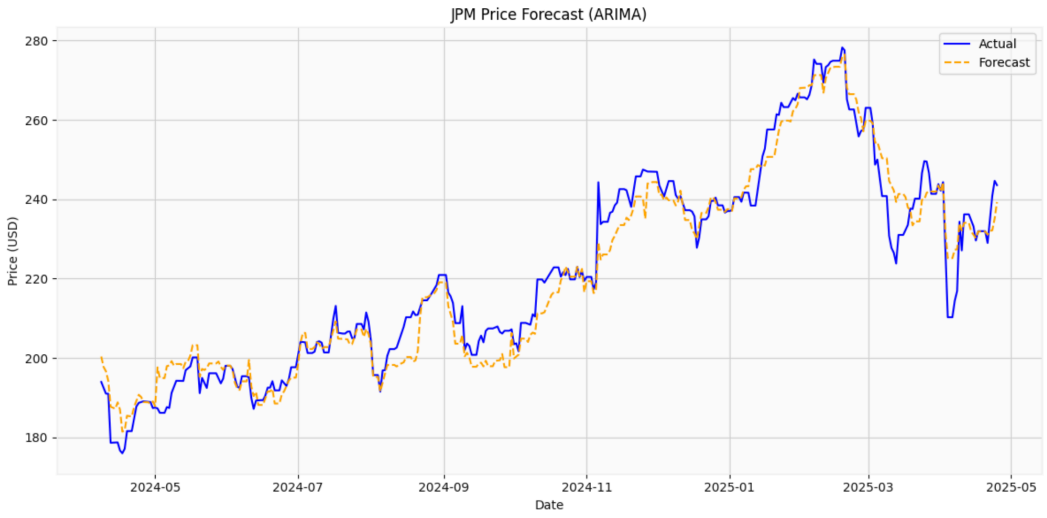



```
Prophet predicted price for next day: 191.69 (+1.43%)
    MAE: 5.1824
    RMSE: 6.8930
```

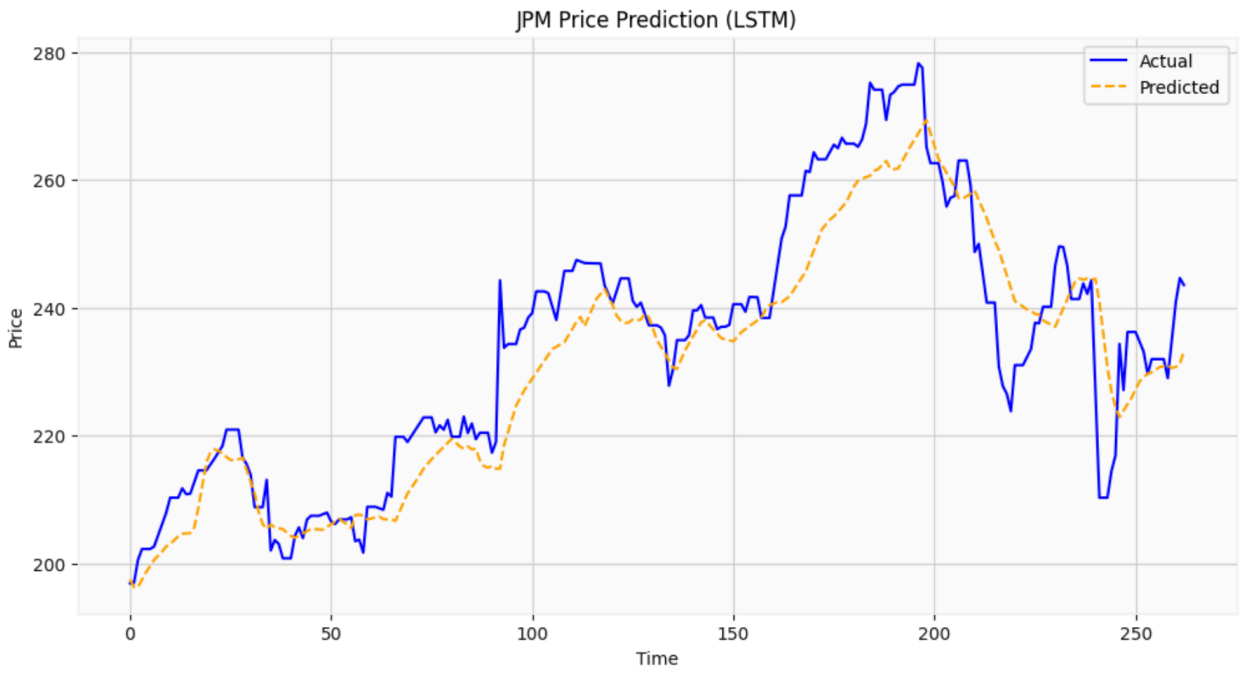
For Amazon, the model that had the lowest error metrics was the LSTM model, with a MAPE value of 1.92. The LSTM predicted a value drop in price of 1.53%.

JPMorgan Chase (JPM)

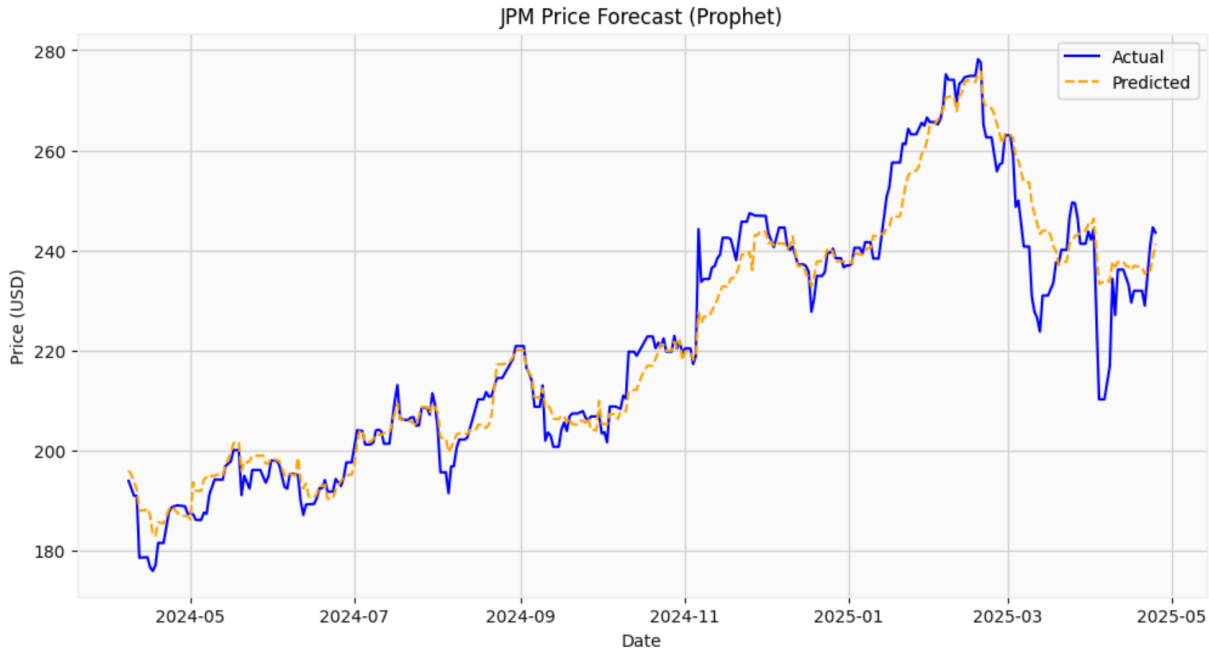
Function call: stock_analysis('JPM', stock_preview=False)




```
ARIMA predicted price for next 1 day(s): 243.10 (-0.19%)
    MAE: 3.9324
    RMSE: 5.2071
    MAPE: 1.7948
```




```
LSTM Price Forecast:
Day 1: 243.74

LSTM predicted price for next 1 day(s): [243.73635267] (+[0.07651391]%)
LSTM MAE: 4.8456590213089115
LSTM MAPE: 2.1299831295896574
LSTM RMSE: 6.864218366329708
```

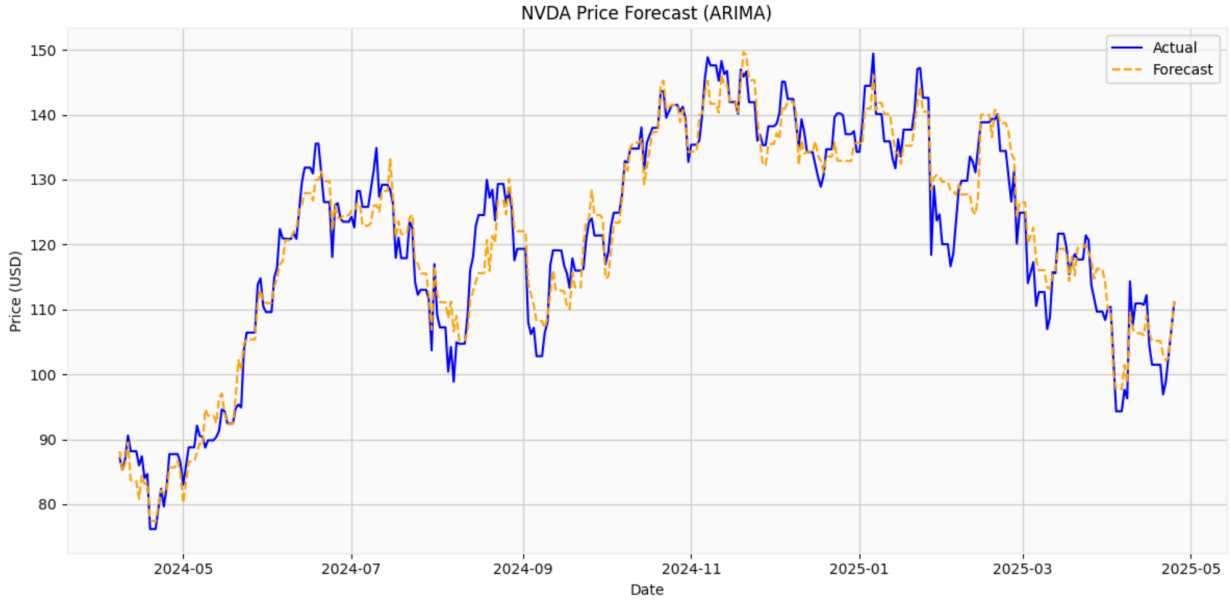



```
Prophet predicted price for next day: 241.36 (-0.90%)
    MAE: 3.7955
    RMSE: 5.5484
```

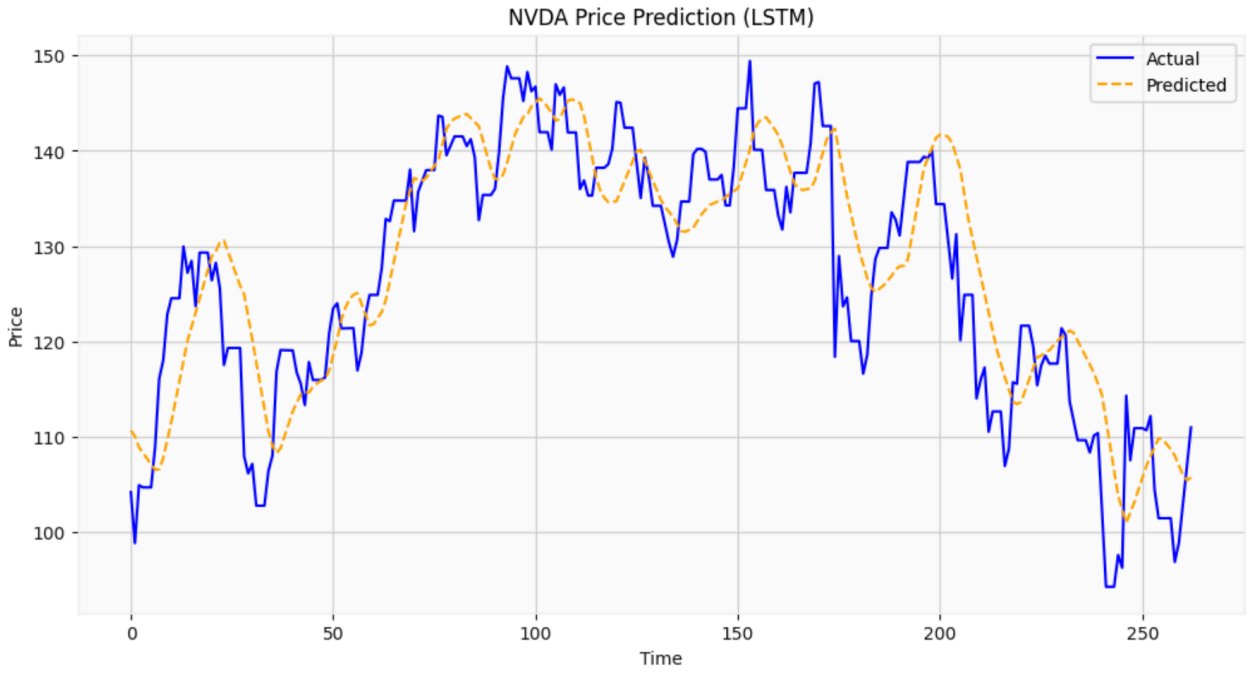
The models for JPMorgan Chase are difficult to compare. None of the models predict a steep change in price in either direction, with all changes being less than one percent. The error metric values are also similar. Based on this, no model does a significantly better job than others at predicting the JPMorgan Chase price.

Nvidia (NVDA)

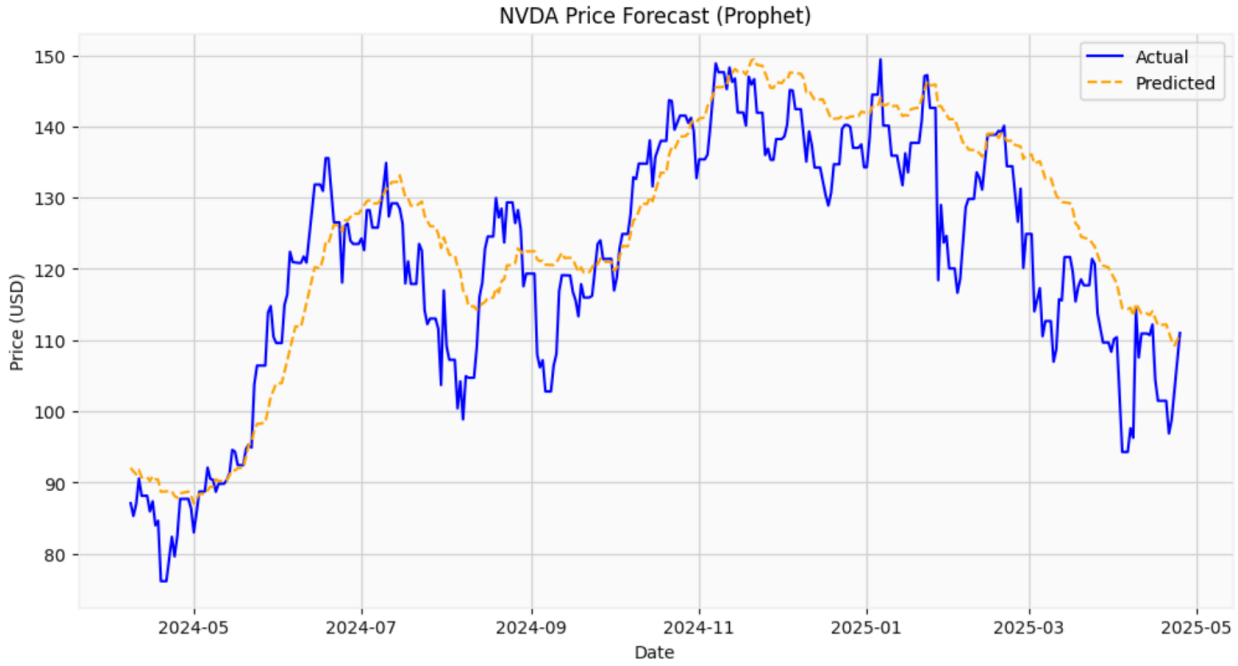
Function call: stock_analysis('NVDA', stock_preview=False)




```
ARIMA predicted price for next 1 day(s): 111.60 (+0.53%)
    MAE: 2.9379
    RMSE: 3.6941
    MAPE: 2.4833
```




```
LSTM Price Forecast:
Day 1: 102.62

LSTM predicted price for next 1 day(s): [102.61899841] ([-7.5587817]%)
LSTM MAE: 5.510862975185341
LSTM MAPE: 4.426752136420233
LSTM RMSE: 6.781910959943842
```

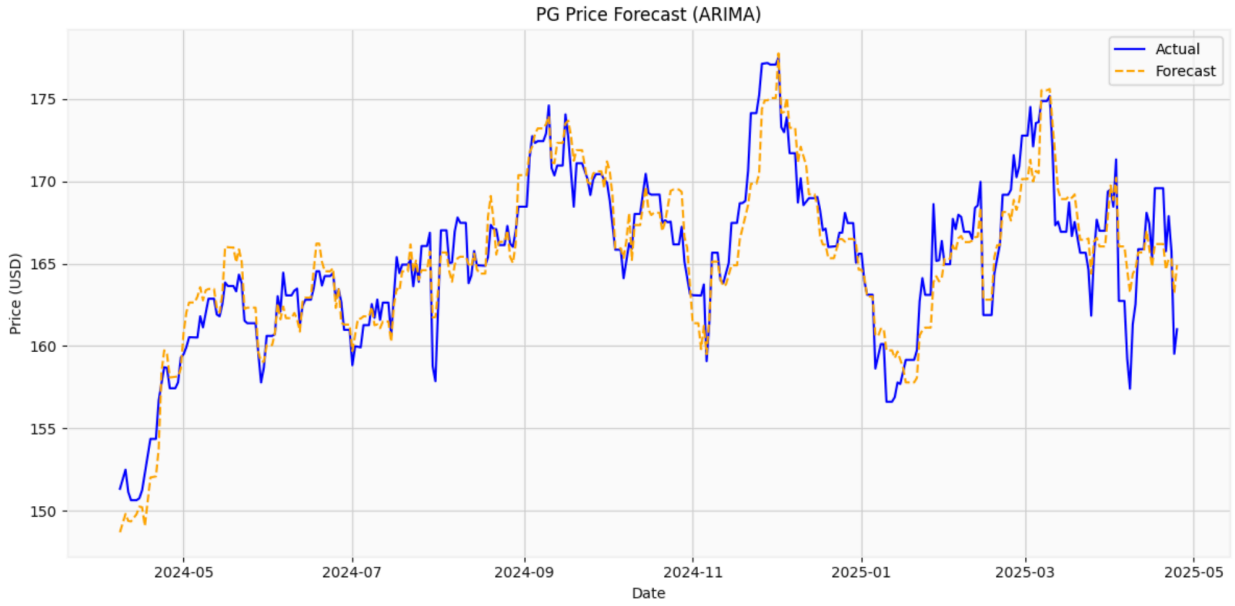



```
Prophet predicted price for next day: 110.77 (-0.22%)
    MAE: 6.8606
    RMSE: 8.7148
```

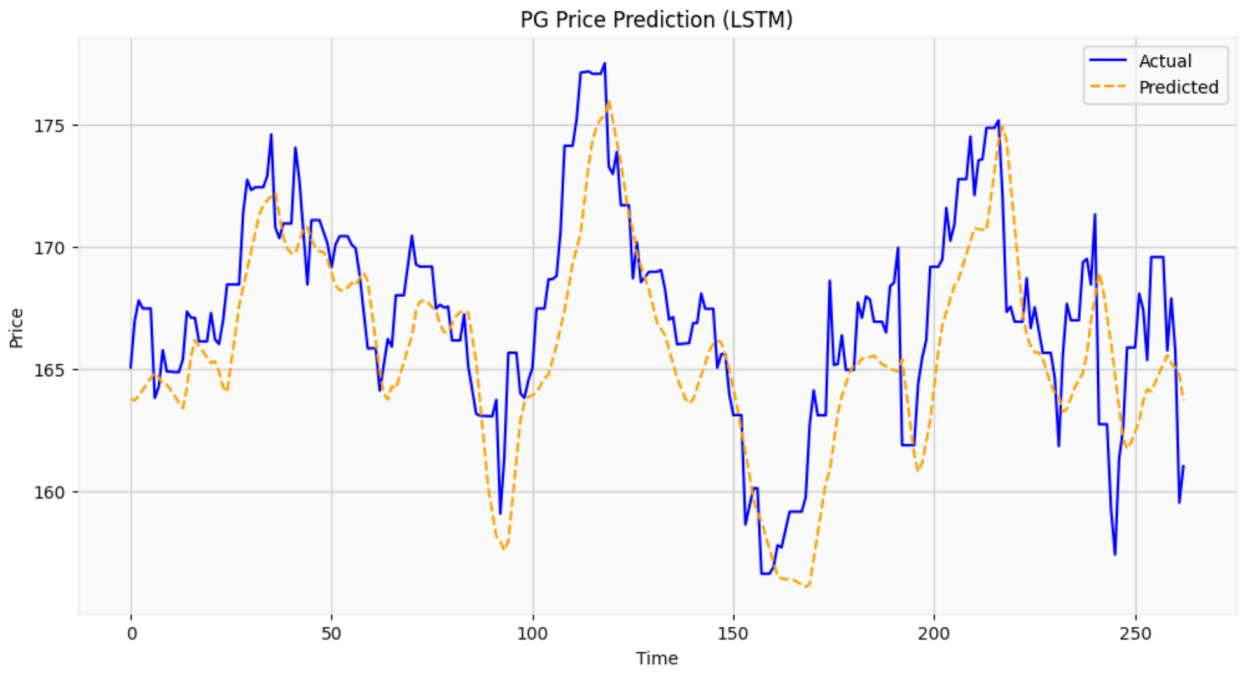
The LSTM model for Nvidia predicted a steep drop in price of 7.56%. Based on the predicted values for the other two models, which are slight changes, we can assume this result is unlikely. The ARIMA model has the lowest error metrics for this stock.

Procter & Gamble (PG)

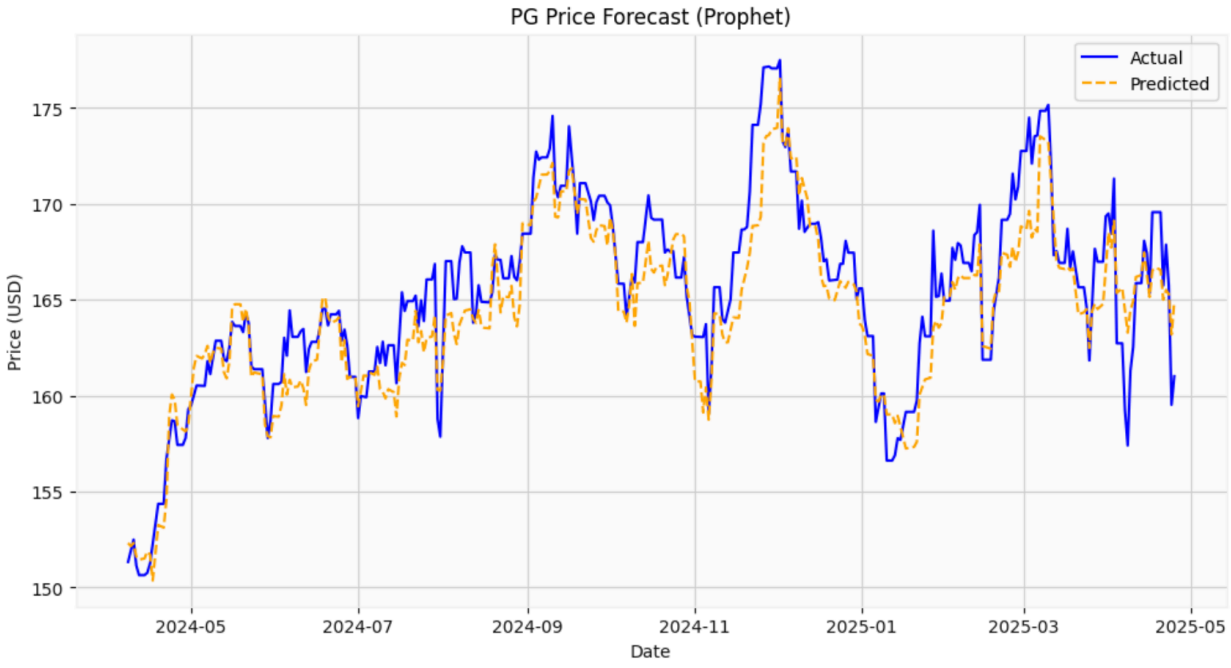
Function call: stock_analysis('PG', stock_preview=False)




```
ARIMA predicted price for next 1 day(s): 161.85 (+0.51%)
    MAE: 1.3374
    RMSE: 1.6777
    MAPE: 0.8099
```




```
LSTM Price Forecast:
Day 1: 164.03

LSTM predicted price for next 1 day(s): [164.03432347] (+[1.87201535]%)
LSTM MAE: 1.8026852841663894
LSTM MAPE: 1.0813375366935223
LSTM RMSE: 2.366113416471412
```

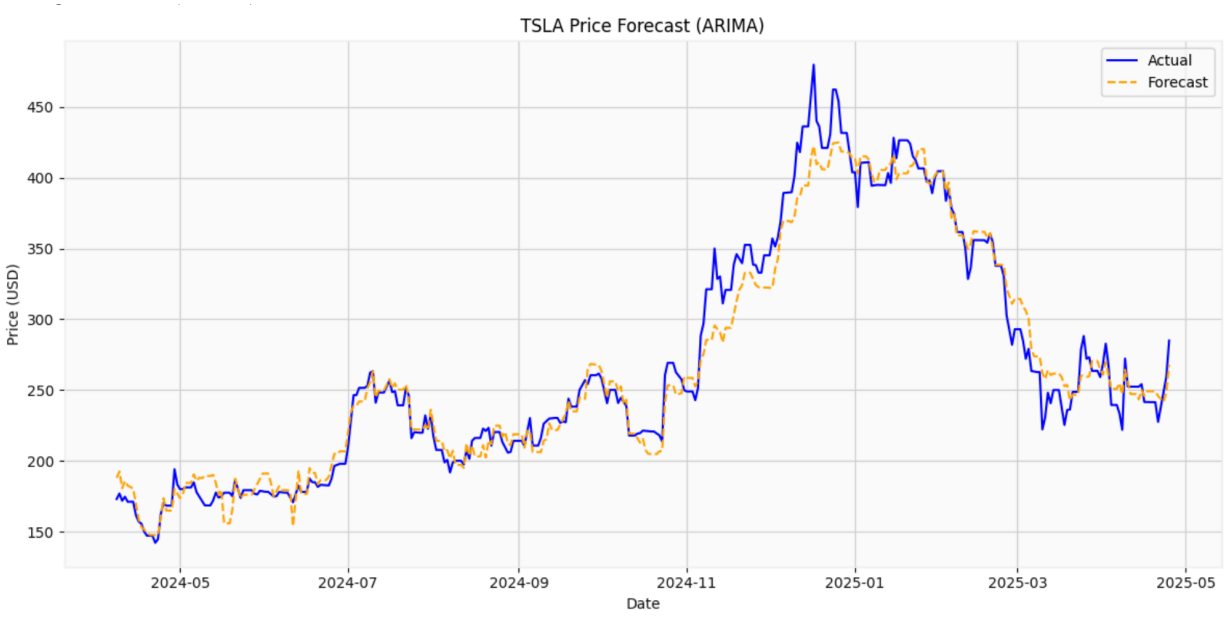



```
Prophet predicted price for next day: 164.90 (+2.41%)
    MAE: 1.5707
    RMSE: 1.9516
```

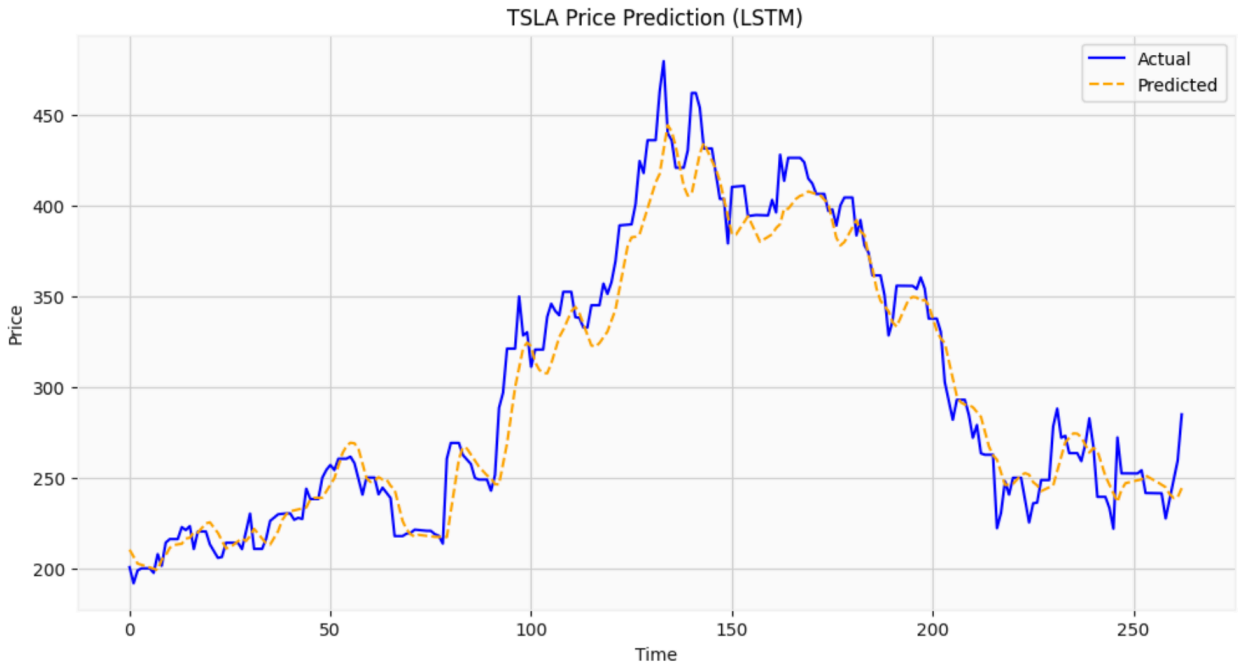
The error metrics for Procter & Gamble are all low when compared with the values for other stocks. All three have a MAE less than 2, and the ARIMA model has a MAPE of 0.81. This means, on average, the ARIMA model predicts the stock price within 0.81% of the actual price. This is an excellent model for predictions.

Tesla (TSLA)

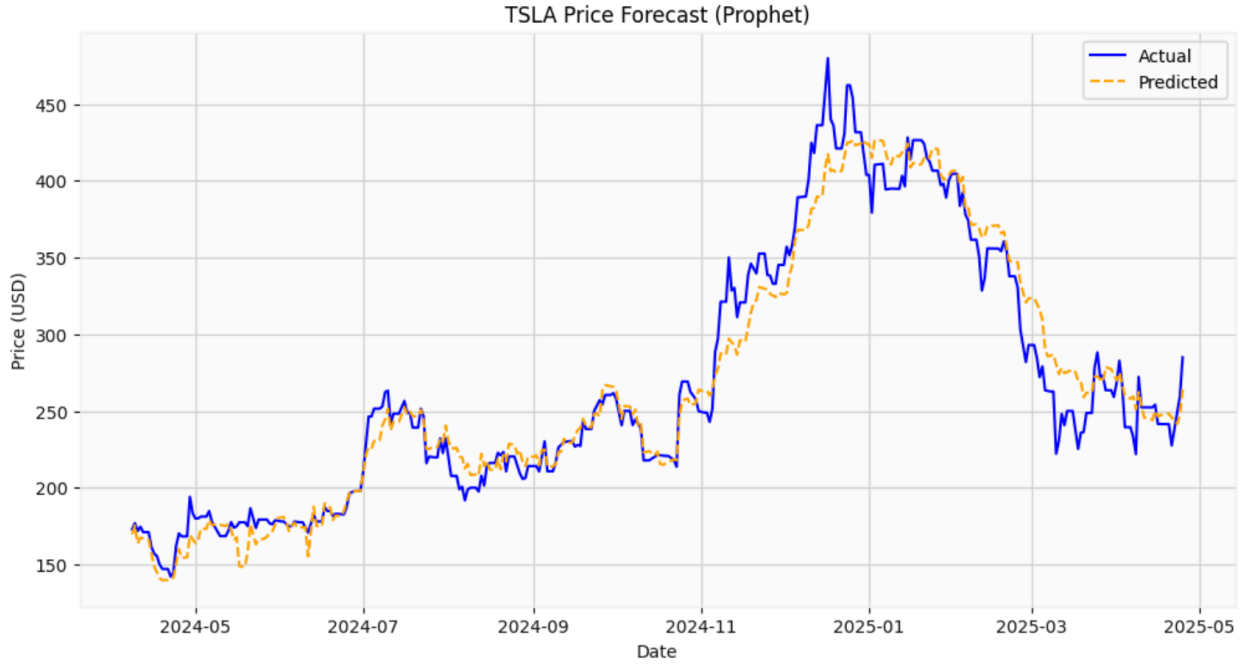
Function call: stock_analysis('TSLA', stock_preview=False)




```
ARIMA predicted price for next 1 day(s): 282.00 (-1.03%)
    MAE: 10.7815
    RMSE: 14.5507
    MAPE: 3.9641
```




```
LSTM Price Forecast:
Day 1: 256.12

LSTM predicted price for next 1 day(s): [256.11809108] ([-10.11823825]%)
LSTM MAE: 10.650136533313386
LSTM MAPE: 3.7834291949411334
LSTM RMSE: 14.777017294416957
```




```
Prophet predicted price for next day: 264.39 (-7.22%)
    MAE: 12.6957
    RMSE: 16.9402
```

Lastly, the models have difficulty closely predicting the price of Tesla. The Prophet model has an RMSE of 16.94, the LSTM has an RMSE of 14.78, and the ARIMA has an RMSE of 14.55. This is likely due to the volatility of the stock, as Elon Musk is polarizing himself as a prominent figure in the United States Government.

Limitations

Stock market movements are difficult to accurately predict due to the noise and unpredictable nature of the financial sector. There are countless factors that go into the price of a stock, such as political events, company decisions, and public sentiment. Many of these factors cannot be predicted using simple historic price data. Because these factors are sudden and impossible to forecast, short term price movements are essentially random. This randomness causes low confidence in predictions in the future. Additionally, a model that fits well with past data cannot predict the volatility of the stock market when major events occur.

In the past decade, two major events occurred that had a serious impact on the stock market. First, the sell-off that was caused by the spread of COVID-19 in early 2020 caused high volatility in the market. The government pushed to revitalize the economy, and prices returned, but many argue that the market never stabilized. In recent news, the tariffs imposed by President Donald Trump had a similar effect on the market. Almost every stock saw a major downturn as the uncertainty of these tariffs increased. The market is currently highly volatile, and the accuracy of predictions may hurt.

Although the models used in this function have promise, there are statistical limitations that must be taken into account. The analysis of the models was done purely on large-cap stocks, meaning the factors included may not result in accurate predictions for smaller companies. The limited forecast horizon for the models is also a concern. Many investors do not wish to invest for a day at a time, and require longer forecasts. Finally, the models are hard to interpret, and are

considered ‘black-box models.’ Simply put, this means that a model may produce an accurate prediction, but how it got to that prediction is unknown.

Future Work

The immediate future of this function is focused on fixing or decreasing the limitations described above. World events cannot be predicted, but sentiment analysis from investors and the public can be. Getting text from new articles surrounding a stock, using VADER or another sentiment analysis tool, and feeding the data into our models can help improve accuracy. Giving the user more options is also needed, specifically with which indicators they want included, and the time period for predictions. Giving the option to predict the stock price in a month may reduce the randomness in the data, leading to better predictions. More technical indicators can help further explain the variation in the movements of the stock market.

Conclusion

In the analysis of our six chosen stocks, no model seemed to outperform any of the others on a consistent basis. Each model — whether ARIMA, LSTM, or Prophet — demonstrated the ability to closely predict stock prices, but also revealed weaknesses, particularly in highly volatile environments, such as Tesla. The Prophet models were never the most accurate for any stock, which is concerning for analysis, but this is likely due to the need to feed seasonality and trends into the Prophet models, a necessary improvement for the future. Overall, the inability of any model to consistently outperform the others is a testament to the difficult nature of stock market predictions.