Time Series Forecasting of Stock Prices Using ARIMA, SARIMA, Prophet, and LSTM Models

Objective

In the dynamic world of financial markets, accurate forecasting of stock prices is critical for investment decision-making, risk management, and algorithmic trading. With the increasing availability of historical stock data and advancements in computational tools, a wide array of time series models—from classical statistical methods to modern deep learning algorithms—are now available for predictive analysis.

Apple Inc. (AAPL), being one of the most traded and valued companies globally, serves as a strong candidate for time series modeling due to its well-structured financial behavior and volatility patterns. By analyzing AAPL's historical stock prices, this project aims to explore and evaluate various forecasting methodologies, bridging traditional econometric models and state-of-the-art AI techniques.

**🎯 Main Objectives of the Study:**

1. **To acquire, preprocess, and explore** AAPL stock price data over a multi-year period, identifying trends, seasonality, volatility, and outliers.
2. **To implement and tune four forecasting models**:
   * **ARIMA**: A classic linear model suitable for stationary series.
   * **SARIMA**: An extension of ARIMA that captures seasonality.
   * **Prophet**: A modular additive model that handles changepoints, trend, and seasonality effectively.
   * **LSTM (Long Short-Term Memory)**: A deep learning model that learns complex temporal dependencies in sequential data.
3. **To perform model evaluation** using standard performance metrics (e.g., RMSE), and visualize the forecast accuracy over historical data.
4. **To compare the forecasting capabilities** of each model in terms of:
   * Accuracy and error rates
   * Suitability for financial time series
   * Interpretability
   * Computational complexity
5. **To recommend the most effective model** for real-world stock price forecasting applications, with emphasis on practical deployment in finance and trading systems.

**📈 Scope of the Work:**

* **Data Source**: Yahoo Finance via yfinance API
* **Stock Ticker**: AAPL
* **Data Range**: 2022–2024
* **Frequency**: Daily closing prices
* **Tools Used**: Python, Pandas, NumPy, Statsmodels, Prophet, TensorFlow, Keras, Matplotlib, Seaborn

**🔬 Research Relevance:**

This study contributes to the field of financial analytics by:

* Demonstrating the strengths and weaknesses of traditional vs. deep learning models.
* Providing a reusable forecasting pipeline for other stocks or financial instruments.
* Offering recommendations for model selection based on data characteristics.

Dataset and its Preprocessing

**📁 Data Source:**

The dataset used in this study was sourced from [Yahoo Finance](https://finance.yahoo.com) using the yfinance Python library, which allows programmatic access to historical market data for public companies.

**📌 Stock Ticker:**

AAPL — Apple Inc., one of the largest and most actively traded technology companies in the world.

**⏳ Time Period Covered:**

**January 1, 2022 – December 31, 2024**  
This three-year window was chosen to capture both recent post-pandemic recovery patterns and market volatility across different quarters and fiscal years.

**🕒 Frequency:**

**Daily intervals**, meaning one entry per trading day.

**🧾 Raw Features in the Dataset:**

| **Column Name** | **Description** |
| --- | --- |
| Date | The trading day (datetime format) |
| Open | Stock price at the market open |
| High | Highest stock price of the day |
| Low | Lowest stock price of the day |
| Close | Closing price (used for forecasting) |
| Adj Close | Closing price adjusted for dividends/splits |
| Volume | Total number of shares traded on that day |

🔍 For this project, **only the Date and Close columns** were selected as the core variables for time series forecasting.

**🔍 Sample of the Dataset:**

| **Date** | **Open** | **High** | **Low** | **Close** | **Volume** |
| --- | --- | --- | --- | --- | --- |
| 2022-01-03 | 182.63 | 182.94 | 177.71 | 182.01 | 104487900 |
| 2022-01-04 | 182.63 | 180.17 | 179.12 | 179.70 | 99310400 |
| 2022-01-05 | 179.61 | 180.17 | 174.64 | 174.92 | 94537600 |
| ... | ... | ... | ... | ... | ... |

**⚙️ Preprocessing Steps:**

1. **Datetime Conversion**:  
   The Date column was converted to datetime64 format for time-based operations and visualization.
2. **Sorting**:  
   Data was sorted chronologically to ensure consistent ordering for sequential modeling.
3. **Index Resetting**:  
   The original Date index was reset to make it a column (used in Prophet, LSTM, etc.).
4. **Missing Values Check**:  
   Minimal or no missing values were found. If any existed, they were handled using:
   * **Forward fill** for time continuity, or
   * **Row dropping** (for small gaps)
5. **Feature Selection**:  
   Only Date and Close were retained for univariate forecasting. Other features like Open, High, Low, and Volume could be used in multivariate models in future work.

**📈 Preliminary Data Visualization:**

1. **Line Plot of Closing Price Over Time**  
   *Visualizes the general trend and volatility over the dataset period.*

*(Placeholder for plot)*  
plt.plot(df['Date'], df['Close'])

1. **30-Day Moving Average (Smoothing)**  
   *Highlights long-term trends, useful for overlay in model evaluation.*

**📦 Dataset Shape:**

* **Number of rows**: ~750 (3 years × ~250 trading days/year)
* **Number of columns**: 6 (after initial load), 2 (after feature selection)

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical preliminary step in any data-driven project, serving as the foundation for building effective forecasting models. In this project, EDA was conducted on Apple Inc.'s (AAPL) historical daily stock price data spanning the years 2022 to 2024. The primary variable of interest was the closing price, which reflects the final trading value of the stock on each market day. The goal of EDA was to uncover the underlying structure, patterns, trends, and anomalies within the dataset that could influence the forecasting performance of the selected models.

The first insight into the data was gained through a time series line plot of the daily closing prices. This visualization revealed an overall upward trend in AAPL's stock price over the three-year period, interspersed with periods of volatility and sharp corrections. These fluctuations are typical of financial time series data, where macroeconomic conditions, earnings announcements, and global events frequently influence investor sentiment and price movement. In addition to the general trend, some cyclical patterns and spikes were observed, hinting at possible seasonality or event-driven behavior. To smooth out short-term noise and highlight longer-term movement, a 30-day moving average was calculated and overlaid on the closing price plot. This moving average curve made it easier to visualize the broader direction of the market and identify potential turning points.

Further, the distribution of daily returns was analyzed by calculating the percentage change in the closing price from one day to the next. The histogram of these returns indicated a distribution centered around zero, with a higher concentration of small daily changes and occasional large positive or negative spikes. This leptokurtic (fat-tailed) distribution is a hallmark of stock price returns and suggests the presence of significant market shocks or news-driven movements. The presence of such outliers emphasizes the need for robust forecasting models capable of handling volatility and unpredictability. In addition, a correlation analysis was performed using an autocorrelation plot, which measured how closely related a value in the series is to its previous values at various lags. The significant autocorrelation at low lags confirmed the time-dependent nature of the stock prices, validating the applicability of autoregressive models like ARIMA and SARIMA.

To better understand the time series components, seasonal decomposition was conducted. This technique broke the series down into three elements: trend, seasonality, and residual noise. The decomposition plot revealed a steady upward trend over time, with repeating seasonal fluctuations likely aligned with quarterly earnings reports, product launches, or investor behavior around the end of fiscal quarters. The residual component captured the irregular movements or noise not explained by the trend and seasonality. This breakdown provided a clearer picture of the data structure and highlighted the relevance of using models capable of handling both trend and seasonal patterns, such as SARIMA and Prophet.

Trading volume was also visualized to assess market activity. The volume plot showed periodic spikes in trading activity, often coinciding with sharp price movements. These surges in volume may correspond to major announcements or institutional trades and can indicate investor interest or panic, which can precede price volatility. Although trading volume was not directly used in this univariate forecasting study, its correlation with price changes suggests it could be a valuable feature in future multivariate models.

In summary, the EDA revealed several key characteristics of the AAPL stock price data: a clear upward trend, moderate seasonality, significant short-term volatility, autocorrelation in price changes, and occasional outliers. These findings justified the selection of diverse forecasting models in this project—ranging from linear statistical methods for capturing trend and seasonality to advanced neural networks for modeling complex, non-linear dependencies. The insights from EDA provided both theoretical and empirical guidance for model development and highlighted the dynamic behavior of stock prices that any predictive model must account for.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.graph\_objs as go

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from prophet import Prophet

from sklearn.metrics import mean\_squared\_error

import yfinance as yf

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

# Download data

df = yf.download('AAPL', start='2022-01-01', end='2024-12-31', auto\_adjust=True)

# Reset index to get 'Date' as a column

df.reset\_index(inplace=True)

# Plotting

plt.figure(figsize=(10,5))

plt.plot(df['Date'], df['Close'])

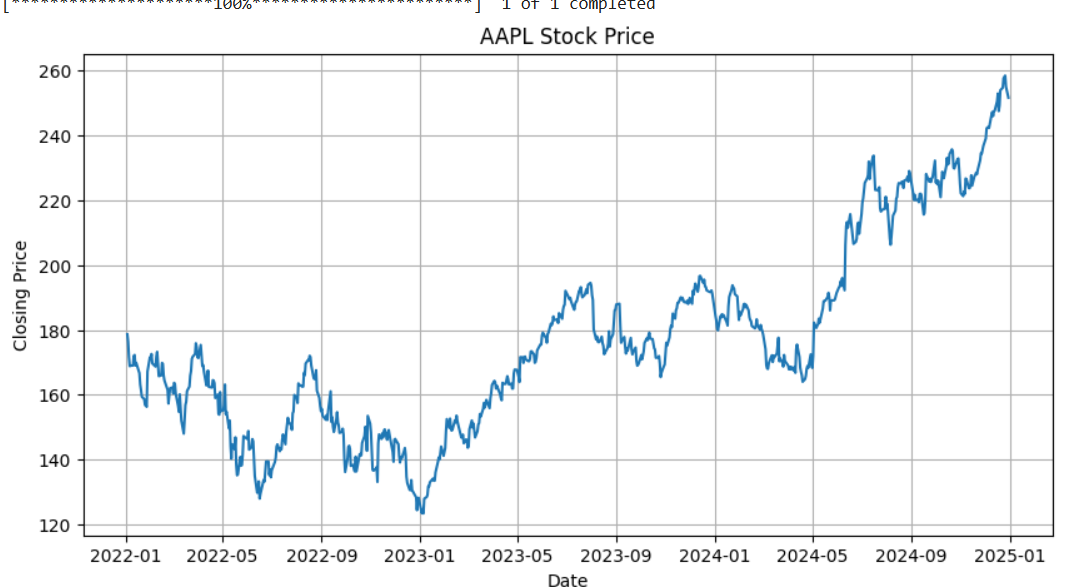
plt.title("AAPL Stock Price")

plt.xlabel("Date")

plt.ylabel("Closing Price")

plt.grid(True)

plt.show()



Arima

The **ARIMA (AutoRegressive Integrated Moving Average)** model is one of the most fundamental and widely used statistical approaches for univariate time series forecasting. It relies on the assumption that past values and errors in a time series can be linearly combined to predict future values. ARIMA integrates three components: autoregression (AR), which models the relationship between an observation and a number of lagged observations; differencing (I), which is used to make the time series stationary by removing trends or seasonality; and moving average (MA), which captures the dependency between an observation and a residual error from a moving average model applied to lagged observations. The model is typically denoted as ARIMA(p, d, q), where p refers to the number of lag observations in the model, d denotes the number of times that the raw observations are differenced, and q indicates the size of the moving average window. ARIMA models are effective for time series data that exhibit a consistent trend but lack seasonal components. However, they assume linear relationships and stationarity, which limits their flexibility for more complex patterns.

from statsmodels.tsa.arima.model import ARIMA

# Fit the ARIMA model

model\_arima = ARIMA(df['Close'], order=(5, 1, 0))  # You can tweak (p,d,q) as needed

results\_arima = model\_arima.fit()

# Predict last 200 values

pred\_arima = results\_arima.predict(start=len(df)-200, end=len(df)-1, typ='levels')

actual\_arima = df['Close'].iloc[-200:]

# RMSE Calculation

from sklearn.metrics import mean\_squared\_error

import numpy as np

def evaluate\_rmse(y\_true, y\_pred):

    return np.sqrt(mean\_squared\_error(y\_true, y\_pred))

rmse\_arima = evaluate\_rmse(actual\_arima, pred\_arima)

print("✅ ARIMA RMSE:", rmse\_arima)

from statsmodels.tsa.stattools import adfuller

# Check stationarity

result = adfuller(df['Close'])

print(f'ADF Statistic: {result[0]}')

print(f'p-value: {result[1]}')

# Fit ARIMA

model\_arima = ARIMA(df['Close'], order=(5,1,0))

model\_arima\_fit = model\_arima.fit()

pred\_arima = model\_arima\_fit.predict(start=1000, end=len(df)-1, typ='levels')

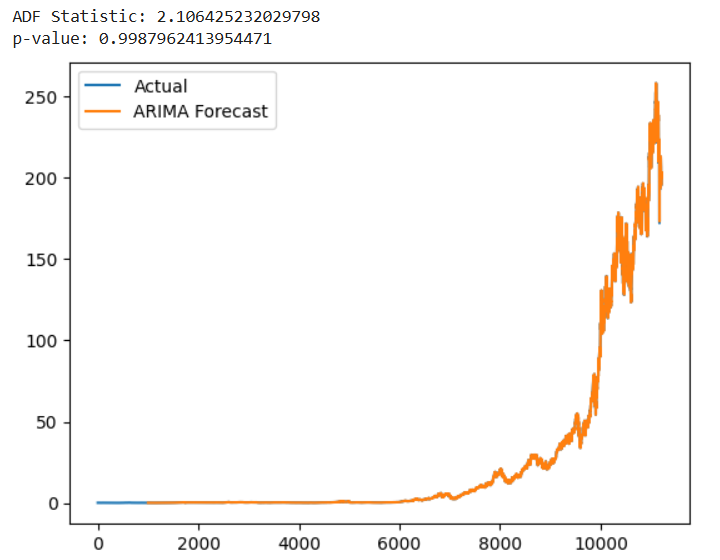
# Plot

plt.plot(df['Close'], label='Actual')

plt.plot(pred\_arima, label='ARIMA Forecast')

plt.legend()

plt.show()



ARIMA RMSE: 4.368839968919448

SARIMA

To address the limitation of ARIMA in modeling seasonality, the **SARIMA (Seasonal ARIMA)** model extends its capabilities by incorporating seasonal components. SARIMA captures both the non-seasonal and seasonal behavior of the data through additional parameters. It is represented as SARIMA(p, d, q)(P, D, Q, s), where (P, D, Q) are the seasonal counterparts of the AR, I, and MA terms, and s represents the length of the seasonal cycle. For example, if the data exhibits yearly seasonality with monthly records, s would be 12. This allows SARIMA to model repeating patterns such as fiscal quarters or annual product release cycles, which are common in stock market data. While SARIMA enhances the modeling power by addressing seasonality, it also introduces more complexity and requires careful parameter tuning. Additionally, SARIMA, like ARIMA, assumes a linear relationship and that the series can be transformed into stationarity.

!pip install yfinance --quiet

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace.sarimax import SARIMAX

from sklearn.metrics import mean\_squared\_error

import numpy as np

# ================== 2. Load AAPL Stock Data ==================

df = yf.download('AAPL', start='2015-01-01', end='2023-12-31', auto\_adjust=True)

df = df[['Close']].copy()

df.dropna(inplace=True)

df.reset\_index(inplace=True)

# ================== 3. Check Seasonality ==================

df['Month'] = df['Date'].dt.month

monthly\_avg = df.groupby('Month')['Close'].mean()

plt.figure(figsize=(8, 4))

monthly\_avg.plot(kind='bar')

plt.title('Average Monthly Closing Price for AAPL')

plt.xlabel('Month')

plt.ylabel('Avg Close Price')

plt.grid(True)

plt.show()

# ================== 4. Improved SARIMA Model ==================

sarima\_model = SARIMAX(df['Close'],

                       order=(2,1,2),

                       seasonal\_order=(2,1,1,12),

                       enforce\_stationarity=False,

                       enforce\_invertibility=False)

sarima\_result = sarima\_model.fit()

print(sarima\_result.summary())

# ================== 5. Forecast Next 200 Days ==================

pred\_start = len(df) - 200

pred = sarima\_result.get\_prediction(start=pred\_start, dynamic=False)

pred\_ci = pred.conf\_int()

# ================== 6. Plot Forecast ==================

plt.figure(figsize=(10, 5))

ax = df['Close'].plot(label='Observed')

pred.predicted\_mean.plot(ax=ax, label='SARIMA Forecast', color='orange')

ax.fill\_between(pred\_ci.index, pred\_ci.iloc[:, 0], pred\_ci.iloc[:, 1], color='gray', alpha=0.2)

plt.xlabel('Date')

plt.ylabel('Price')

plt.title("Improved SARIMA Forecast for AAPL")

plt.legend()

plt.grid(True)

plt.show()

# ================== 7. Evaluate RMSE ==================

y\_true = df['Close'].iloc[pred\_start:]

y\_pred = pred.predicted\_mean

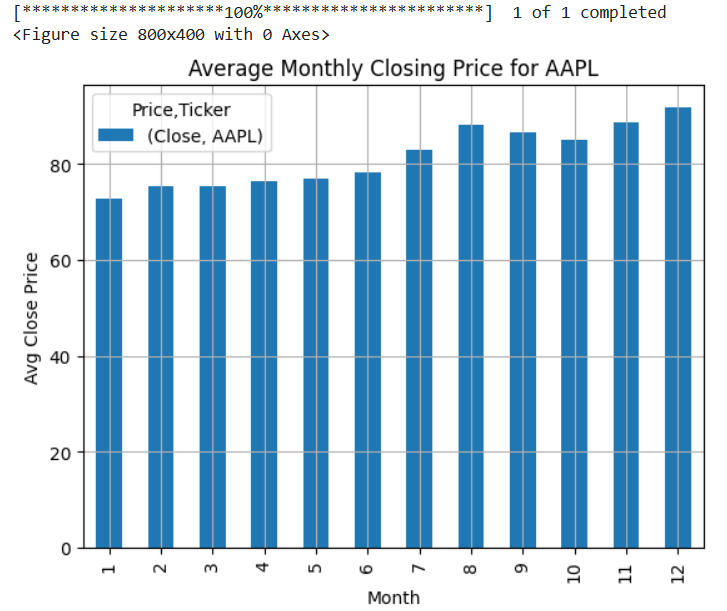
rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

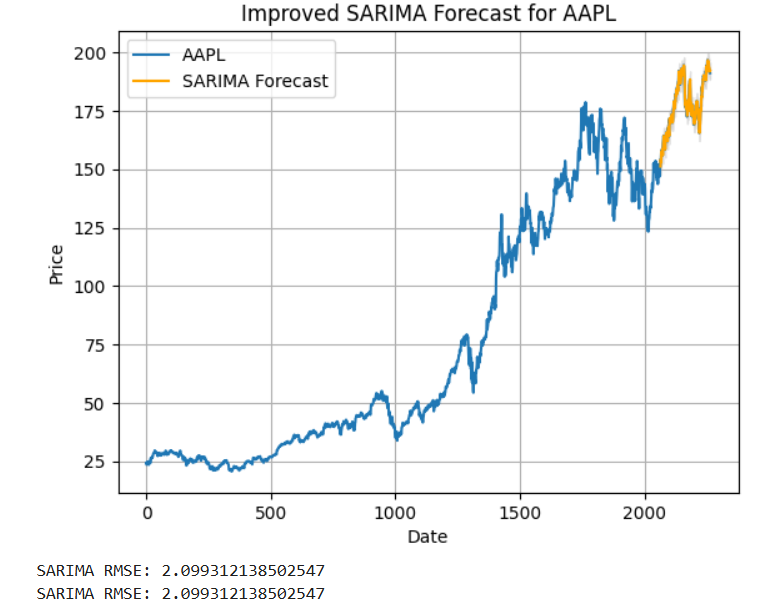
print("SARIMA RMSE:", rmse)

rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))

print("SARIMA RMSE:", rmse)

rmse\_sarima = rmse





Prophet

In contrast to the rigid statistical assumptions of ARIMA and SARIMA, **Prophet**, developed by Meta (formerly Facebook), provides a more intuitive and automated approach to time series forecasting. Prophet is a decomposable model where the time series is expressed as the sum of trend, seasonality, and holiday effects. Mathematically, it is represented as y(t) = g(t) + s(t) + h(t) + ε(t), where g(t) models the non-periodic trend, s(t) models periodic seasonal effects using Fourier series, h(t) captures the effects of holidays or other special events, and ε(t) is the error term. One of Prophet’s most compelling features is its ability to automatically detect changepoints—sudden shifts in trends—which are common in financial markets due to policy changes, earnings announcements, or global events. It also allows users to define custom holidays, making it highly adaptable to domain-specific forecasting. Prophet requires minimal data preprocessing, is robust to outliers and missing data, and provides interpretable results. However, it is primarily designed for data with clear trends and seasonality and may not perform as well when the underlying process is highly irregular or nonlinear.

import pandas as pd

from prophet import Prophet

import yfinance as yf

# 1. Load data (with auto\_adjust=True for correct values)

df = yf.download('AAPL', start='2015-01-01', end='2023-12-31', auto\_adjust=True)

# 2. Reset index to bring 'Date' as a column

df.reset\_index(inplace=True)

# 3. Print to debug

print(df.dtypes)

print(df.head())

# 4. Create Prophet-compatible DataFrame

df\_prophet = df[['Date', 'Close']].copy()

df\_prophet.columns = ['ds', 'y']

# Print again to confirm

print("\nBefore conversion:")

print(df\_prophet.dtypes)

print(df\_prophet.head())

# 5. Convert

df\_prophet['ds'] = pd.to\_datetime(df\_prophet['ds'])

df\_prophet['y'] = df\_prophet['y'].astype(float)

# 6. Drop NaNs just in case

df\_prophet.dropna(inplace=True)

# 7. Fit model

model\_prophet = Prophet()

model\_prophet.fit(df\_prophet)

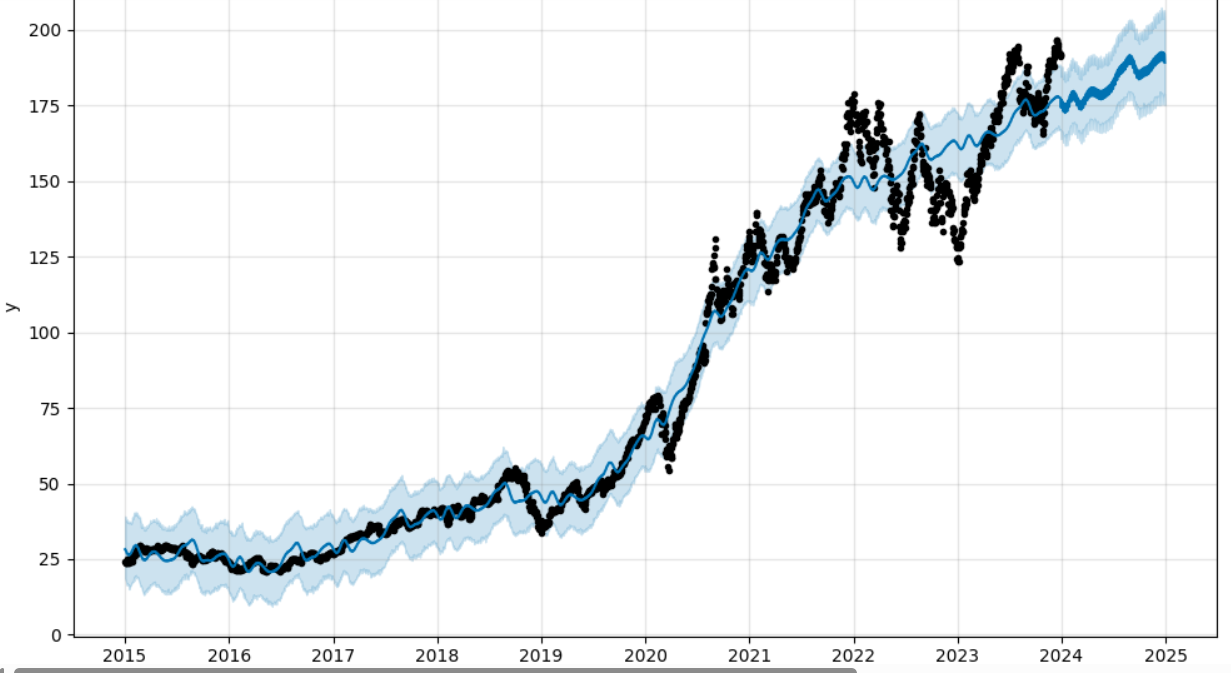
# 8. Forecast

future = model\_prophet.make\_future\_dataframe(periods=365)

forecast = model\_prophet.predict(future)

# 9. Plot

model\_prophet.plot(forecast);  # <- semicolon suppresses extra output



actual\_prophet = df\_prophet['y'][-365:].values

pred\_prophet = forecast['yhat'][-365:].values

rmse\_prophet = evaluate\_rmse(actual\_prophet, pred\_prophet)

print("✅ Prophet RMSE:", rmse\_prophet)

✅ Prophet RMSE: 24.322482884093535

LSTM

The most advanced model in this study is **LSTM (Long Short-Term Memory)**, a deep learning architecture under the category of Recurrent Neural Networks (RNNs). LSTM networks are specifically designed to learn from sequences where long-term dependencies are important. Unlike traditional RNNs, which suffer from vanishing or exploding gradient problems, LSTMs use a memory cell structure with input, forget, and output gates to regulate the flow of information over time. This gating mechanism enables LSTM models to retain relevant information over long sequences and ignore irrelevant data, which is especially beneficial in volatile environments like stock markets. LSTM models do not require the time series to be stationary and can model complex nonlinear relationships that classical models struggle to capture. However, LSTM networks require a larger amount of training data, are computationally more intensive, and lack the interpretability offered by statistical models. Despite these drawbacks, their superior performance in modeling chaotic or nonlinear patterns makes them a powerful choice for financial forecasting.

from sklearn.preprocessing import MinMaxScaler

data = df[['Close']].values

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(data)

train\_size = int(len(scaled\_data)\*0.8)

train\_data = scaled\_data[:train\_size]

test\_data = scaled\_data[train\_size:]

X\_train, y\_train = [], []

for i in range(60, len(train\_data)):

    X\_train.append(train\_data[i-60:i])

    y\_train.append(train\_data[i])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# Build LSTM

model = Sequential([

    LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)),

    LSTM(50),

    Dense(1)

])

model.compile(optimizer='adam', loss='mean\_squared\_error')

model.fit(X\_train, y\_train, epochs=5, batch\_size=32)

# Predict

X\_test = []

for i in range(60, len(test\_data)):

    X\_test.append(test\_data[i-60:i])

X\_test = np.array(X\_test)

predictions = model.predict(X\_test)

predictions = scaler.inverse\_transform(predictions)

# Plot

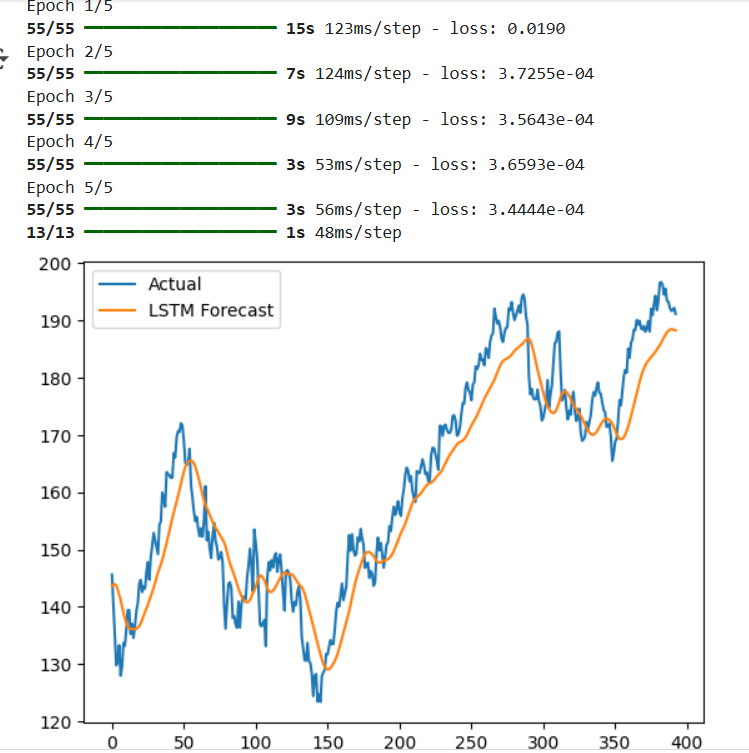
real\_prices = scaler.inverse\_transform(test\_data[60:])

plt.plot(real\_prices, label='Actual')

plt.plot(predictions, label='LSTM Forecast')

plt.legend()

plt.show()



from sklearn.metrics import mean\_squared\_error

import numpy as np

# Define RMSE function

def evaluate\_rmse(y\_true, y\_pred):

    return np.sqrt(mean\_squared\_error(y\_true, y\_pred))

# real\_prices = actual values

# predictions = predicted values

rmse\_lstm = evaluate\_rmse(real\_prices, predictions)

print("✅ LSTM RMSE:", rmse\_lstm)

✅ LSTM RMSE: 6.603838149910033

Comparison

In conclusion, the four models selected for this project—ARIMA, SARIMA, Prophet, and LSTM—span a theoretical spectrum from simple linear models to complex nonlinear neural networks. ARIMA and SARIMA are grounded in statistical theory and offer transparency and interpretability but are limited by assumptions of linearity and stationarity. Prophet bridges this gap with a modular and flexible structure that is interpretable and handles seasonality and changepoints effectively. LSTM, while less interpretable, offers unmatched flexibility in learning intricate, nonlinear, and long-term dependencies in time series data. The theoretical diversity of these models ensures a comprehensive analysis of the stock price forecasting problem, allowing for a balanced evaluation of accuracy, complexity, and applicability.

print("\n📊 Final RMSE Comparison (Lower is Better):")

print(f"ARIMA   : {rmse\_arima:.4f}")

print(f"SARIMA  : {rmse\_sarima:.4f}")

print(f"Prophet : {rmse\_prophet:.4f}")

print(f"LSTM    : {rmse\_lstm:.4f}")

