PRACTICAL GUIDE

How to Predict Stock Prices Using Machine Learning

Ive Botunac

Swipe →

Introduction

Forecasting stock prices with machine learning can provide a significant edge in today's financial world.

This post explores the intersection of finance and advanced algorithms, focusing on LSTM neural networks for prediction.

We'll cover:

- Financial market basics
- Data preparation
- LSTM model implementation
- Training and prediction
- Result visualization

Whether you're a finance professional, a data scientist, or a curious investor, this guide offers insights into combining financial knowledge with these machine learning techniques.

Types of Financial Markets

Financial markets play a crucial role in the global economy, facilitating the exchange of capital and assets. These markets provide essential mechanisms for companies to raise funds, investors to grow wealth, and governments to finance operations.

There are several distinct categories of financial markets, each serving specific purposes and catering to different financial instruments:

- **Money Markets:** Short-term financing instruments like treasury bills.
- **Capital Markets:** Long-term securities such as stocks and bonds.
- **Primary Markets:** New securities are issued.
- **Secondary Markets:** Existing securities are traded.

AI in Financial Markets

AI enhances trading strategies by predicting market trends. It processes vast datasets at high speeds, providing valuable insights and enabling data-driven decisions.

- **Predictive analytics:** Uses machine learning to predict market trends, stock prices, and customer behaviors.
- **Algorithmic trading:** AI analyzes real-time market data, executes trades, and adjusts strategies autonomously.
- **Fraud detection:** Identifies irregular patterns and anomalies, flagging suspicious activities in real-time.
- **Risk management:** Assesses credit, market, and operational risks by analyzing historical data and predicting potential risks.

Popular Python Libraries

- **Pandas & Numpy:** For numerical operations, data manipulation, and analysis.
- **Matplotlib:** For data visualization, creating static, animated, and interactive visualizations in Python.
- **Scikit-learn:** For simple and fast machine learning algorithms implementation.
- **TensorFlow & Keras:** For building deep learning models.
- **PyPortfolioOpt:** For portfolio optimization, providing tools for constructing optimal portfolios.
- **TA-Lib:** For technical analysis, offering a variety of technical indicators for analyzing financial market data.
- **Backtrader:** For backtesting trading strategies, allowing simulation of trading strategies on historical data.
- **yfinance:** For fetching financial data, enabling easy access to historical market data from Yahoo Finance.

Prepare Your Data

The foundation of any successful machine learning model lies in its data. To predict stock prices, we need to gather historical price data and enrich it with relevant technical indicators.

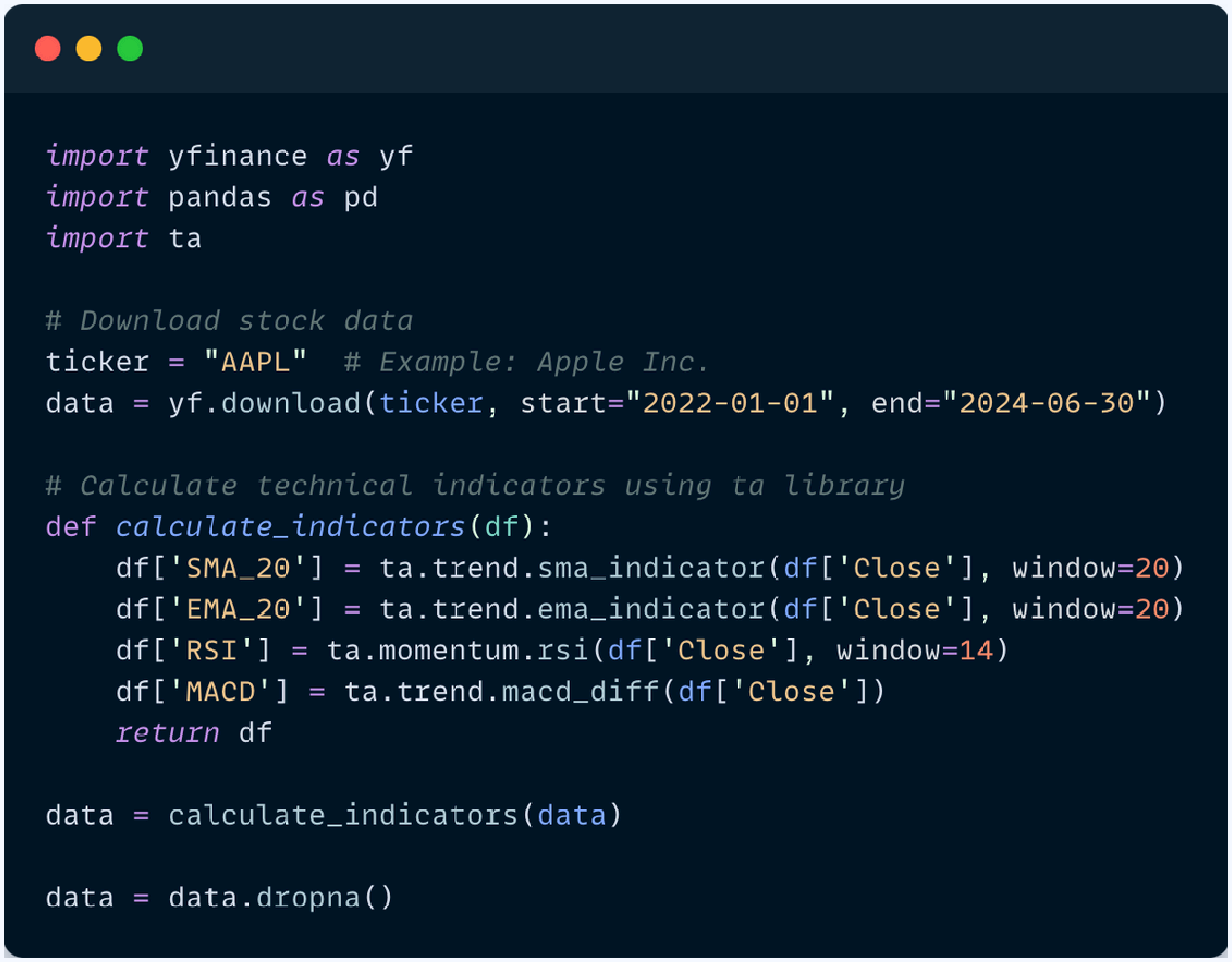
This process involves downloading stock information and calculating various metrics traders use to make informed decisions.

Essential steps:

- Download historical stock data using yfinance
- Calculate technical indicators (SMA, EMA, RSI, MACD)
- Scale the data for neural network input
- Split data into training and testing sets

Data Preparation

Proper data preparation sets the stage for accurate predictions.



```python
import yfinance as yf
import pandas as pd
import ta

# Download stock data
ticker = "AAPL"  # Example: Apple Inc.
data = yf.download(ticker, start="2022-01-01", end="2024-06-30")

# Calculate technical indicators using ta library
def calculate_indicators(df):
    df['SMA_20'] = ta.trend.sma_indicator(df['Close'], window=20)
    df['EMA_20'] = ta.trend.ema_indicator(df['Close'], window=20)
    df['RSI'] = ta.momentum.rsi(df['Close'], window=14)
    df['MACD'] = ta.trend.macd_diff(df['Close'])
    return df

data = calculate_indicators(data)

data = data.dropna()
```

Sequence Creation for LSTM Input

After scaling our data, we must create sequences suitable for LSTM input. This step involves structuring our time series data and splits into train and test sets.

```python
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Prepare data for LSTM
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data[['Close', 'SMA_20', 'EMA_20', 'RSI', 'MACD']])

# Function to create input sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length), :])
        y.append(data[i + seq_length, 0])
    return np.array(X), np.array(y)

# Set sequence length and create train/test sets
seq_length = 1

X, y = create_sequences(scaled_data, seq_length)
train_size = int(len(X) * 0.8)

X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Building the LSTM Model

LSTM (Long Short-Term Memory) neural networks are well-suited for time series prediction tasks like stock price forecasting.

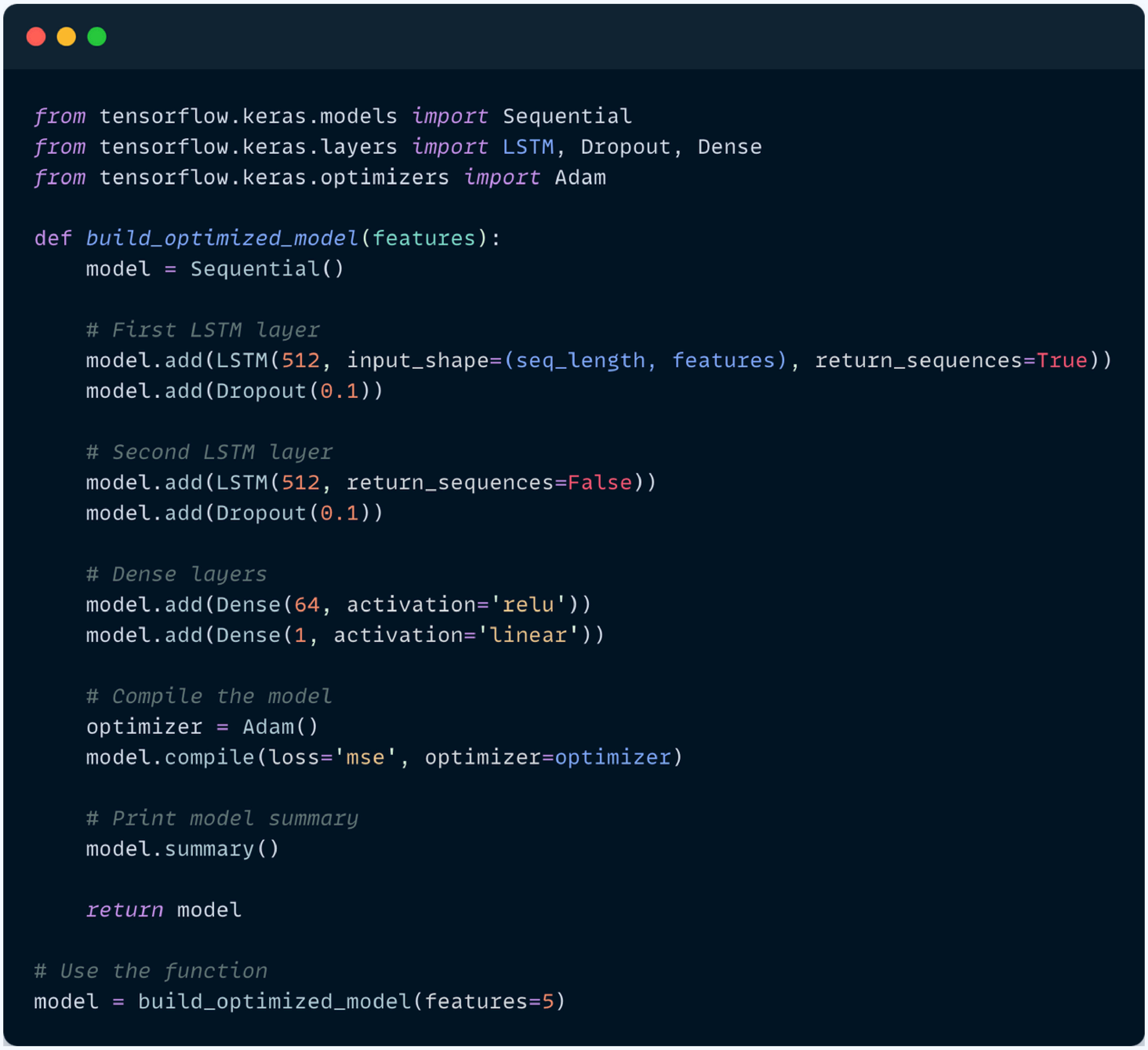
These networks can capture both short-term and long-term patterns in the data, making them ideal for analyzing complex market behaviors.

Model architecture:

- Multiple LSTM layers for deep learning
- Dropout layers to prevent overfitting
- Dense layers for final prediction
- Adam optimizer for efficient training

LSTM Model

A well-structured LSTM model can capture complex patterns in stock price movements.



```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam

def build_optimized_model(features):
    model = Sequential()

    # First LSTM layer
    model.add(LSTM(512, input_shape=(seq_length, features), return_sequences=True))
    model.add(Dropout(0.1))

    # Second LSTM layer
    model.add(LSTM(512, return_sequences=False))
    model.add(Dropout(0.1))

    # Dense layers
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='linear'))

    # Compile the model
    optimizer = Adam()
    model.compile(loss='mse', optimizer=optimizer)

    # Print model summary
    model.summary()

    return model

# Use the function
model = build_optimized_model(features=5)
```

Training and Making Predictions

With our data prepared and model built, we can train the neural network on historical data and use it to make predictions.

This process involves fitting the model to the training data and generating predictions for the test period. We then transform these predictions back to their original scale for interpretation.

Key steps:

- Train the model on historical data
- Generate predictions for the test period
- Inverse transform predictions for interpretability

Training and Predictions

Here's how we train our LSTM model on historical data and use it to generate stock price predictions:

```python
# Train the model
history = model.fit(X_train, y_train, epochs=200, batch_size=256, verbose=0)

# Make predictions
test_predictions = model.predict(X_test)

# Inverse transform predictions
test_predictions = scaler.inverse_transform(np.concatenate((test_predictions,
                                                           np.zeros((len(test_predictions), 4))),
                                                           axis=1))[:, 0]

y_test_inv = scaler.inverse_transform(np.concatenate((y_test.reshape(-1, 1),
                                                     np.zeros((len(y_test), 4))),
                                                     axis=1))[:, 0]

# Get dates for the test set
test_dates = data.index[-len(y_test):]
```

Visualizing and Evaluating Results

The final step in our stock price prediction journey is visualizing our results and evaluating the model's performance. By plotting predicted prices against actual prices, we can assess how well our model performs visually.

Additionally, calculating error metrics provides a quantitative measure of prediction accuracy.

Evaluation methods:

- Plot actual vs. predicted stock prices
- Calculate Mean Squared Error (MSE)
- Calculate Mean Absolute Error (MAE)
- Analyze trends and patterns in predictions

Visualization and Evaluation

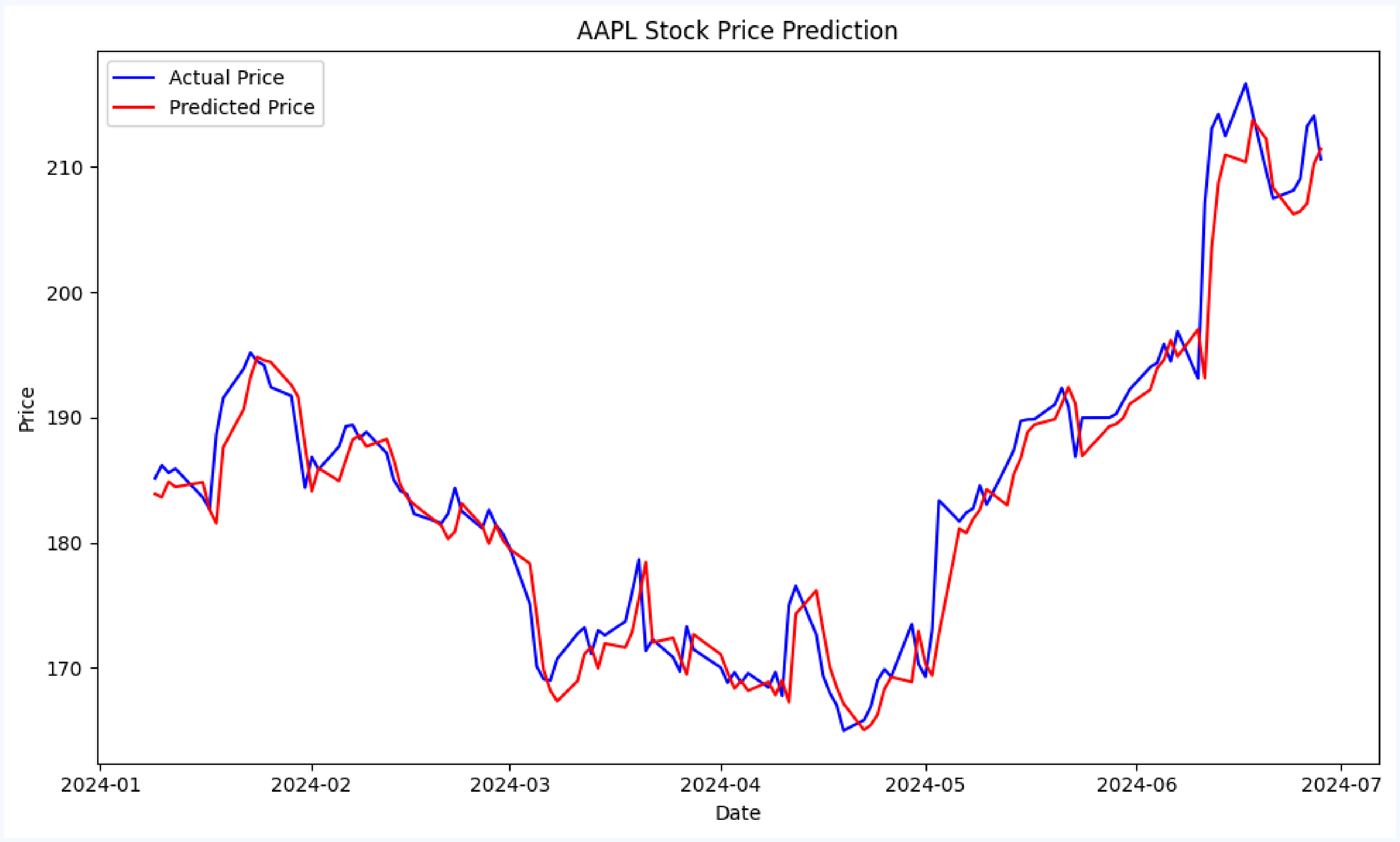
Let's visualize our predictions alongside actual prices and calculate error metrics to assess our model's performance:

```python
import matplotlib.pyplot as plt

# Plot
plt.figure(figsize=(10, 6))

# Plot actual prices
plt.plot(test_dates, y_test_inv, label='Actual Price', color='blue')

# Plot predictions
plt.plot(test_dates, test_predictions, label='Predicted Price', color='red')

plt.title(f'{ticker} Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.tight_layout()
plt.show()
```

Visualizing and Evaluating Graph

The graph below compares our model's predictions (in red) with the actual stock prices (in blue) over the test period.

This visual representation helps us quickly assess how well our LSTM model captures the trends and patterns in the stock price movement.

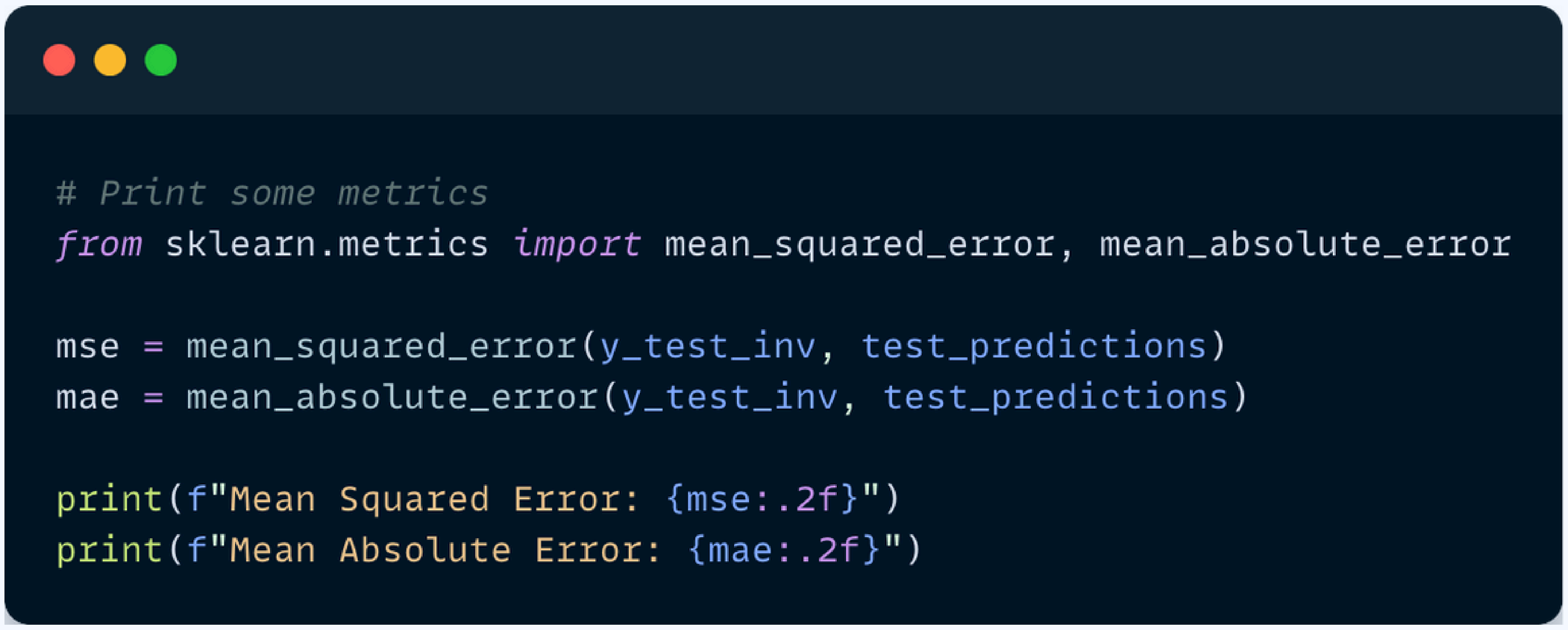


Performance Metrics

To quantify our model's accuracy, we calculate two common error metrics:

- Mean Squared Error: 9.64
- Mean Absolute Error: 2.21

These metrics provide a numerical measure of how close our predictions are to the actual stock prices.



```python
# Print some metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error

mse = mean_squared_error(y_test_inv, test_predictions)
mae = mean_absolute_error(y_test_inv, test_predictions)

print(f"Mean Squared Error: {mse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
```

Benefits of AI in Trading

- **Accuracy:** AI models improve prediction accuracy by analyzing vast amounts of data and identifying hidden patterns.
- **Risk management:** Enhanced risk mitigation through continuous monitoring and quick adaptation to market changes.
- **Speed:** Faster decision-making by automating trade analysis and execution in real-time.
- **Consistency:** Removes emotional biases from trading decisions, ensuring a disciplined approach.

Conclusion

Using the right AI/ML libraries inside Python unlocks new potential in financial markets. This powerful combination facilitates advanced data analysis, enhances trading strategies, and streamlines risk management, providing significant advantages in the competitive financial landscape.

- **Advanced data analysis:** Quickly uncover insights for better decision-making.
- **Improved trading strategies:** Optimize trade execution for higher returns.
- **Efficient risk management:** Continuously monitor and adjust strategies to mitigate risks.
- **Competitive advantage:** Stay ahead of market trends and outperform the market index.