Time Series Analysis Final Project Report

Forecasting Tata Motors Stock Prices

I. Introduction

This analysis aims to forecast Tata Motors Limited's stock prices using time series and deep learning models. The goal is to identify historical trends, predict future stock prices for the next 90 to 120 days, and assess these models against technical benchmarks for comparison. The findings will provide insights to traders and investors, guiding them in making informed decisions based on current trends and future projections.

II. Dataset

The data for this project was sourced from Kaggle, containing daily historical stock prices of Tata Motors, including:
- Date
- Opening, High, Low, and Closing prices
- Trading volume
- Turnover

III. Data Preprocessing and Exploration

Data Cleaning: Missing or invalid data points were removed, and any remaining gaps were filled appropriately.
Exploratory Data Analysis (EDA): The focus was on the closing price, as it plays a critical role in predicting future prices. A line chart of the closing prices over time revealed overall trends. Moving averages, both Simple (SMA) and Exponential (EMA), were also calculated to observe price changes over time.

The data showed a significant price drop during the COVID-19 pandemic, prompting an in-depth analysis of the stock's performance during that period.

IV. Technical Indicators

To analyze time patterns and enhance predictions, the following indicators were used:
- Moving Averages (SMA & EMA): Capture short-term fluctuations while highlighting long-term trends.
- Relative Strength Index (RSI): Measures the speed and change of price movements, helping to identify overbought or oversold conditions.
- Moving Average Convergence Divergence (MACD): Indicates the strength of price movements, signaling potential buying or selling opportunities.

These indicators help identify market trends and improve the accuracy of future stock price predictions.

V. Modeling Approach

Three models were selected to forecast Tata Motors' future stock prices: ARIMA, SARIMA, and LSTM. Each was chosen for its ability to handle different aspects of time series data, such as trends, seasonality, and long-term dependencies.

1. ARIMA (AutoRegressive Integrated Moving Average)

ARIMA is a widely used model for forecasting time series data, as it considers:
- AR (AutoRegression): Uses past stock price values to predict future ones.
- I (Integrated): Differentiates the data to remove trends, making it stationary.
- MA (Moving Average): Captures the relationship between an observation and the residuals from prior observations.

ARIMA was chosen as the base model for its effectiveness in predicting stationary data. Since stock prices are typically non-stationary, differencing was used to remove the trend. However, ARIMA struggles with seasonal patterns, which can be significant in stock price movements.

2. SARIMA (Seasonal ARIMA)

SARIMA extends ARIMA by incorporating seasonal components, making it more suitable for data with recurring patterns. Tata Motors' stock prices showed seasonal behavior influenced by market cycles and economic trends. SARIMA incorporates seasonal autoregressive (SAR), seasonal differencing (SI), and seasonal moving average (SMA) terms to model this behavior.

SARIMA was chosen for its ability to model seasonality, resulting in more accurate predictions for data with periodic fluctuations.

3. LSTM (Long Short-Term Memory)

LSTM is a deep learning model designed to handle sequential data and capture long-term dependencies in time series. Unlike the linear ARIMA and SARIMA models, LSTM can detect complex, nonlinear relationships. Its memory cells store information over long periods, making it well-suited to capturing the intricate patterns of stock price movements driven by market conditions.

LSTM was chosen for its superior ability to model both short-term fluctuations and long-term trends in stock prices.

Why These Models?

The combination of ARIMA, SARIMA, and LSTM represents a balance between traditional statistical models and advanced deep learning techniques. ARIMA and SARIMA provide interpretable models, helpful for understanding specific components like trends and seasonality.

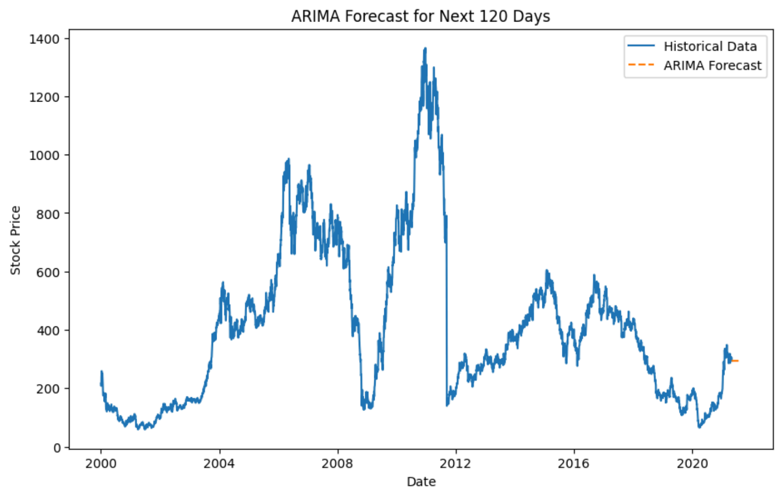
Meanwhile, LSTM leverages the power of neural networks to capture more complex patterns, offering greater accuracy at the cost of interpretability.

VI. Findings and Model Comparisons

After training and testing these models, the following results were obtained:

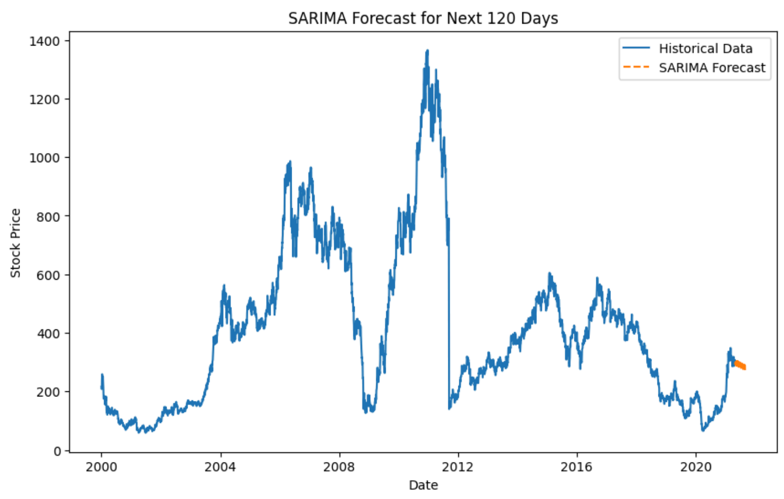
1. ARIMA Model Performance

- Forecast Accuracy: ARIMA performed well for short-term predictions but struggled with sudden market fluctuations and seasonality.
- Strengths: Easy to interpret and effective for stable periods.
- Weaknesses: Lacks the ability to handle seasonal effects, reducing accuracy over longer periods, especially during high volatility.



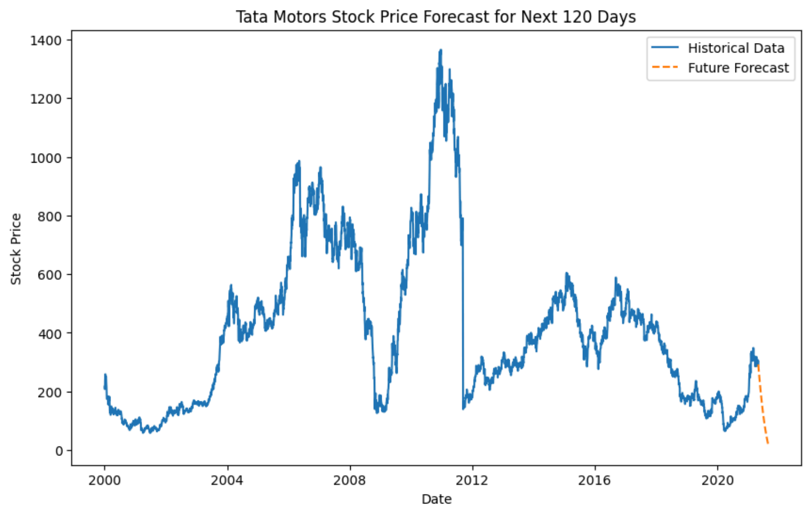
2. SARIMA Model Performance

- Forecast Accuracy: SARIMA outperformed ARIMA by accounting for seasonality, providing more accurate long-term forecasts.
- Strengths: By including seasonal components, SARIMA offered more precise forecasting during recurring market cycles.
- Weaknesses: While effective with seasonality, it still struggled to predict abrupt, non-periodic price changes.



3. LSTM Model Performance

- Forecast Accuracy: LSTM significantly outperformed ARIMA and SARIMA by capturing complex, nonlinear relationships in the data. It excelled in predicting stock prices during volatile periods, such as the COVID-19 pandemic.
- Strengths: LSTM's ability to model long-term dependencies made it the most accurate model. It adapted well to market shocks and fluctuations.
- Weaknesses: LSTM's main drawback is its lack of interpretability compared to ARIMA and SARIMA.



Model Comparison Summary:
- ARIMA: Best for short-term forecasting in stable market conditions.
- SARIMA: Improved long-term accuracy by modeling seasonality but still struggled with unexpected changes.
- LSTM: The most accurate model, especially during volatility, though less interpretable than the other models.

VII. Key Findings and Insights

1. Stock Price Behavior and Trends
- The stock price data was non-stationary, showing both long-term upward trends and short-term fluctuations. Differencing was necessary to make the data stationary.
- External events, particularly the COVID-19 pandemic, caused significant price drops and volatility spikes, influencing stock behavior.
- Seasonal patterns were present but not dominant, making SARIMA effective but not the primary driver of stock price movements.

2. Model Performance
- LSTM's predictions aligned well with technical indicators, validating the model's results.
- Market volatility, especially during the pandemic, was better captured by LSTM compared to the other models.

VIII. Next Steps and Recommendations

- Incorporating More Data: Including macroeconomic factors, sentiment analysis, and industry-specific data could improve model performance.
- Exploring Advanced Models: Experimenting with models like GRU or Transformer-based architectures could provide even better results.

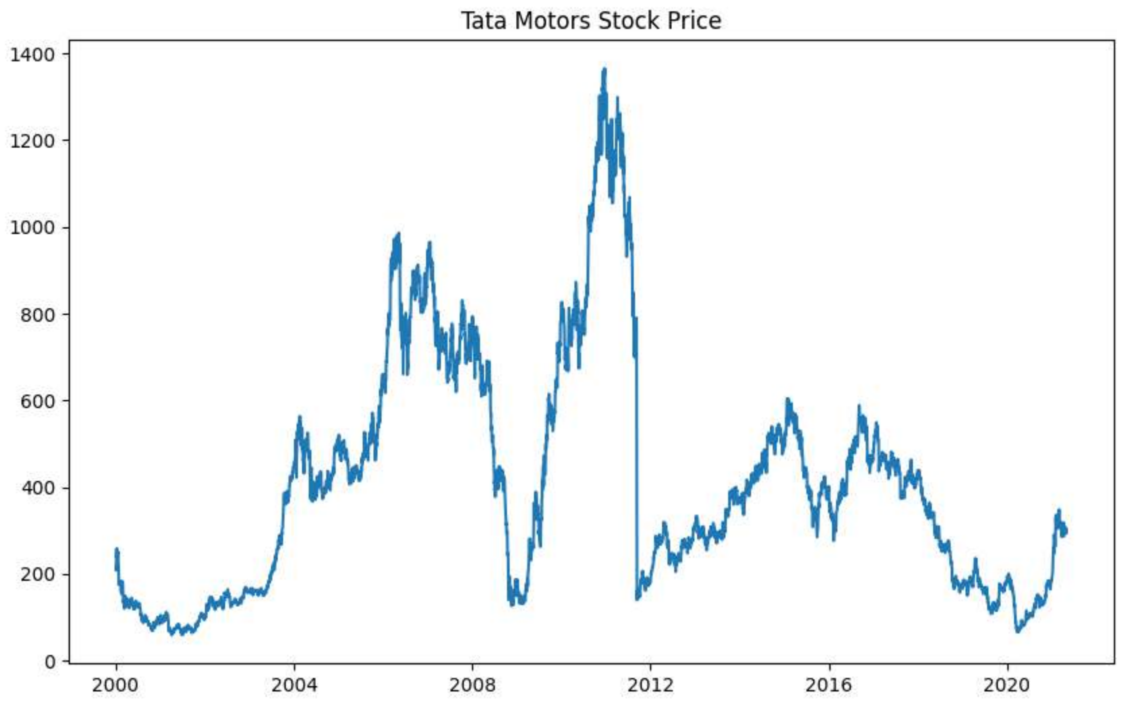
This analysis highlights the benefits of combining traditional time series models with deep learning techniques. Future improvements should focus on incorporating external data and exploring advanced neural networks to enhance forecasting accuracy.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
```

In [2]:
```
# Load Data
data = pd.read_csv('TATAMOTORS.csv', parse_dates=['Date'], index_col='Date
```

In [3]:
```
# Data Preprocessing
data.fillna(method='ffill', inplace=True)  # Forward Fill Missing Values
```

In [4]:
```
# Exploratory Data Analysis (EDA)
plt.figure(figsize=(10,6))
plt.plot(data['Close'])
plt.title('Tata Motors Stock Price')
plt.show()
```



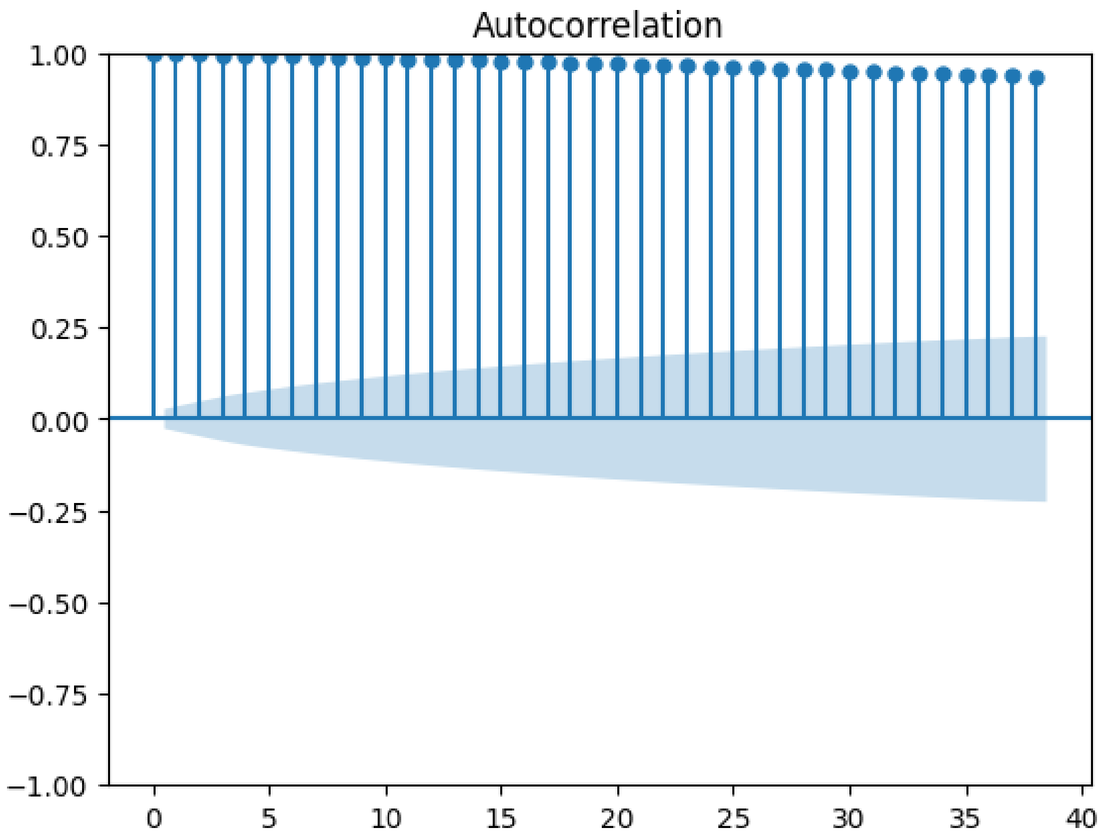
In [5]:
```
# Stationarity Test (Dickey-Fuller)
from statsmodels.tsa.stattools import adfuller
result = adfuller(data['Close'])
print('ADF Statistic:', result[0])
```

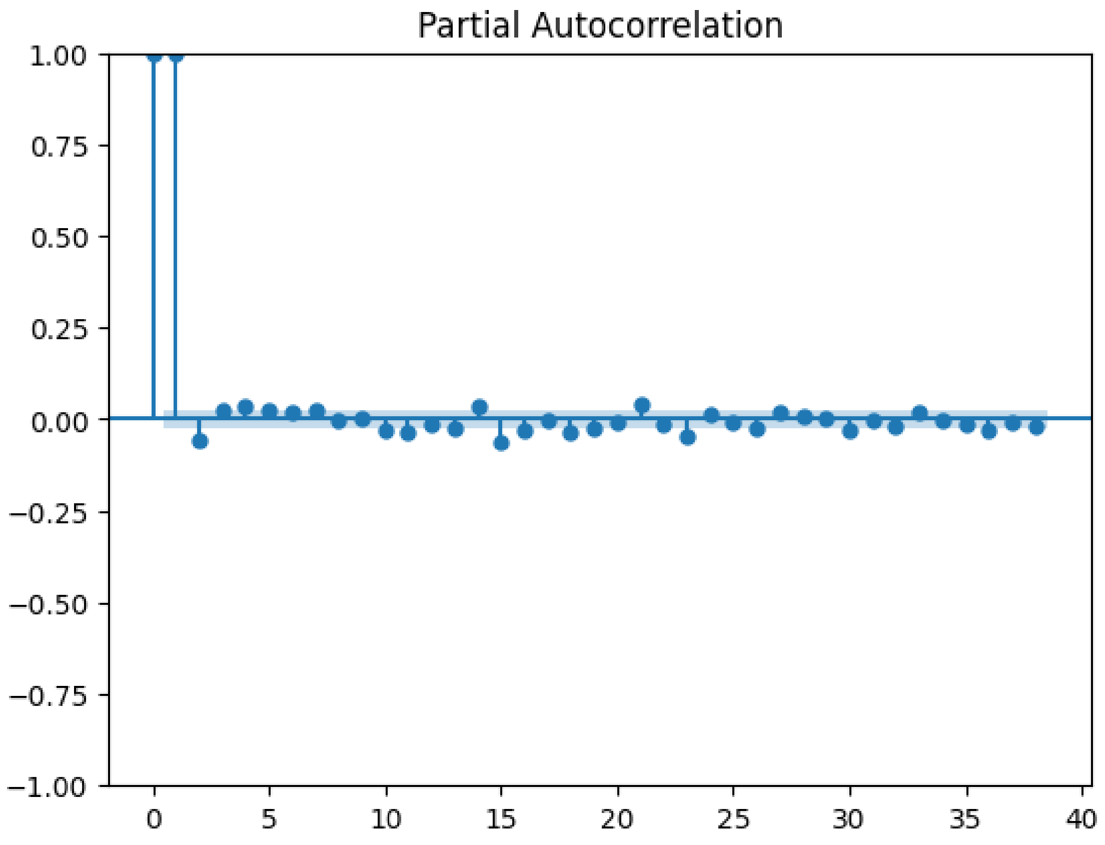
```
ADF Statistic: -2.3076091005997177
p-value: 0.16952562810863342
```

In [6]:

```
# ACF and PACF Plots
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(data['Close'])
plot_pacf(data['Close'])
```

```
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\graphics\tsaplo
ts.py:348: FutureWarning: The default method 'yw' can produce PACF values
outside of the [-1,1] interval. After 0.13, the default will change touna
djusted Yule-Walker ('ywm'). You can use this method now by setting metho
d='ywm'.
  warnings.warn(
```





In [8]:
```
from statsmodels.tsa.arima.model import ARIMA

# ARIMA Model (order can be adjusted based on ACF/PACF plots)
arima_model = ARIMA(data['Close'], order=(5, 1, 0))  # Example: order=(5,1
arima_fit = arima_model.fit()
```

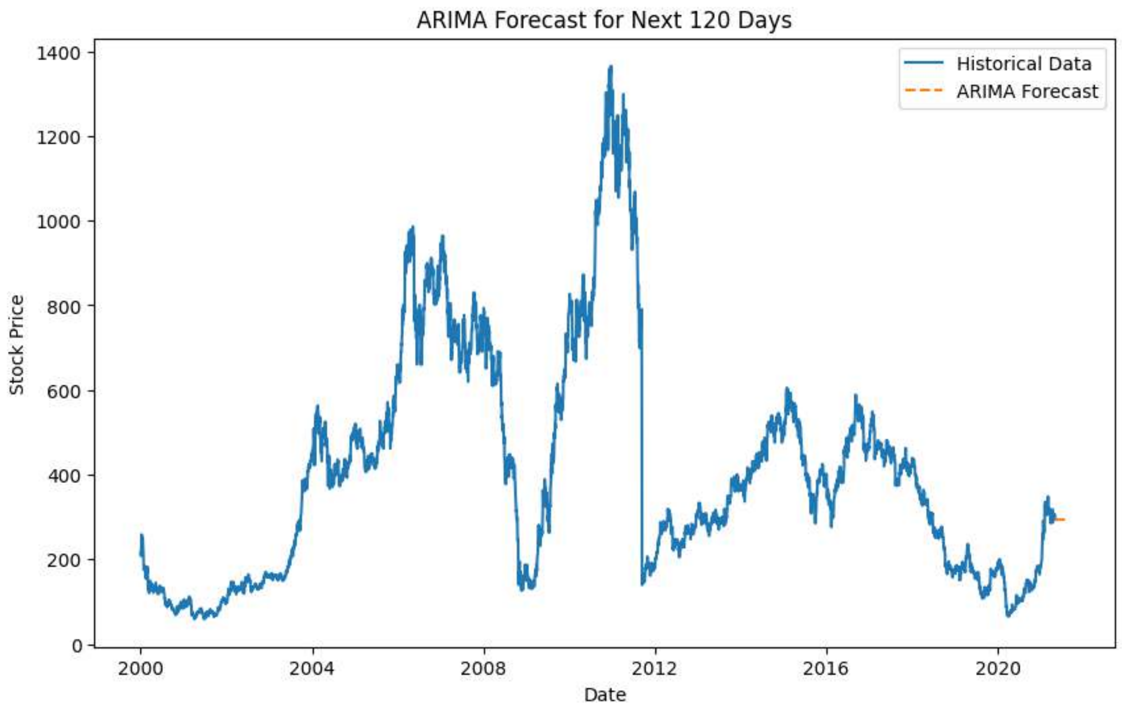
```
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:471: ValueWarning: A date index has been provided, but it has no a
ssociated frequency information and so will be ignored when e.g. forecast
ing.
  self._init_dates(dates, freq)
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:471: ValueWarning: A date index has been provided, but it has no a
ssociated frequency information and so will be ignored when e.g. forecast
ing.
  self._init_dates(dates, freq)
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:471: ValueWarning: A date index has been provided, but it has no a
ssociated frequency information and so will be ignored when e.g. forecast
ing.
  self._init_dates(dates, freq)
```

In [9]:
```
# Forecast for the next 120 days
```

```
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:834: ValueWarning: No supported index is available. Prediction res
ults will be given with an integer index beginning at `start`.
  return get_prediction_index(
```

In [10]:
```
 #Plot the ARIMA forecast
plt.figure(figsize=(10,6))
plt.plot(data.index, data['Close'], label='Historical Data')
future_dates = pd.date_range(data.index[-1], periods=121, closed='right')
plt.plot(future_dates, arima_forecast, label='ARIMA Forecast', linestyle='
plt.title('ARIMA Forecast for Next 120 Days')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
```

```
C:\Users\Sangeeta\AppData\Local\Temp\ipykernel_4520\4229007275.py:4: Futu
reWarning: Argument `closed` is deprecated in favor of `inclusive`.
  future_dates = pd.date_range(data.index[-1], periods=121, closed='righ
t')  # For future forecast plot
```

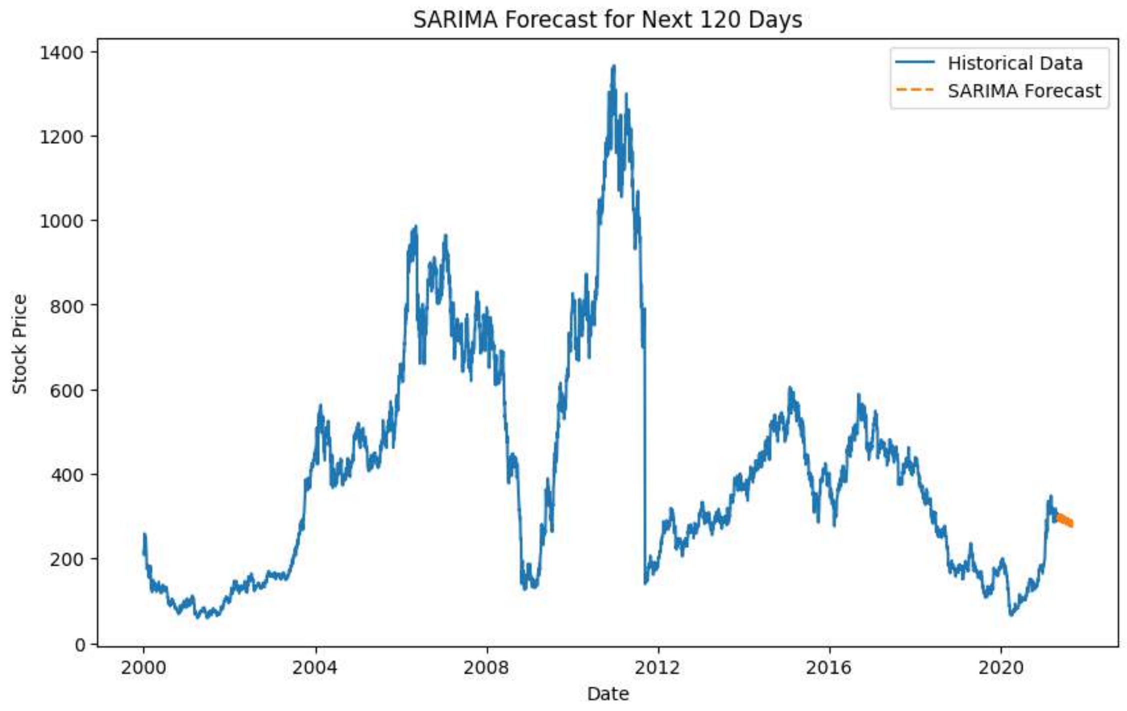


In [12]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Updated SARIMA Model: Based on ACF/PACF plots
sarima_model = SARIMAX(data['Close'],
                       order=(1, 1, 0),              # (p,d,q) based on
                       seasonal_order=(1, 1, 0, 12))   # (P,D,Q,S) with as
```

```
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:471: ValueWarning: A date index has been provided, but it has no a
ssociated frequency information and so will be ignored when e.g. forecast
ing.
  self._init_dates(dates, freq)
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:471: ValueWarning: A date index has been provided, but it has no a
ssociated frequency information and so will be ignored when e.g. forecast
ing.
  self._init_dates(dates, freq)
```

In [13]:
```
# Forecast for the next 120 days
sarima_forecast = sarima_fit.forecast(steps=120)

# Plotting SARIMA forecast
plt.figure(figsize=(10,6))
plt.plot(data.index, data['Close'], label='Historical Data')
future_dates = pd.date_range(data.index[-1], periods=121, closed='right')
plt.plot(future_dates, sarima_forecast, label='SARIMA Forecast', linestyle
plt.title('SARIMA Forecast for Next 120 Days')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
```

```
C:\Users\Sangeeta\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mo
del.py:834: ValueWarning: No supported index is available. Prediction res
ults will be given with an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\Sangeeta\AppData\Local\Temp\ipykernel_4520\4198697991.py:7: Futu
reWarning: Argument `closed` is deprecated in favor of `inclusive`.
  future_dates = pd.date_range(data.index[-1], periods=121, closed='righ
t')  # Dates for the future forecast
```



In [14]:
```
# LSTM Model
# Data Preparation
data_values = data['Close'].values.reshape(-1,1)
train_size = int(len(data_values) * 0.8)
```

In [15]:
```
# Normalize Data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
train_scaled = scaler.fit_transform(train)
test_scaled = scaler.transform(test)
```

In [16]:
```
# Prepare Data for LSTM
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        X.append(dataset[i:(i+look_back), 0])
        Y.append(dataset[i + look_back, 0])
```

In [17]:
```
look_back = 60
X_train, y_train = create_dataset(train_scaled, look_back)
```

In [18]:
```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

In [19]:
```
# LSTM Model
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1],
model.add(Dropout(0.2))
model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(25))
```

```
C:\Users\Sangeeta\anaconda3\lib\site-packages\keras\src\layers\rnn\rnn.p
y:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)` obj
ect as the first layer in the model instead.
  super().__init__(**kwargs)
```

In [20]:
```
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
Epoch 1/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 18s 149ms/step - loss: 0.0204
Epoch 2/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 149ms/step - loss: 0.0016
Epoch 3/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 135ms/step - loss: 0.0014
Epoch 4/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - loss: 9.8055e-04
Epoch 5/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 146ms/step - loss: 0.0011
Epoch 6/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 141ms/step - loss: 8.9625e-04
Epoch 7/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - loss: 0.0011
Epoch 8/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 146ms/step - loss: 9.0270e-04
Epoch 9/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 130ms/step - loss: 9.3051e-04
Epoch 10/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 131ms/step - loss: 9.3646e-04
Epoch 11/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 137ms/step - loss: 8.1741e-04
Epoch 12/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 129ms/step - loss: 7.0965e-04
Epoch 13/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 125ms/step - loss: 6.5295e-04
Epoch 14/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 127ms/step - loss: 6.5228e-04
Epoch 15/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 5.9404e-04
Epoch 16/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 148ms/step - loss: 6.6267e-04
Epoch 17/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 5.4846e-04
Epoch 18/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 12s 174ms/step - loss: 7.3951e-04
Epoch 19/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 11s 158ms/step - loss: 5.4193e-04
Epoch 20/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 5.9520e-04
Epoch 21/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 13s 193ms/step - loss: 5.9683e-04
Epoch 22/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 128ms/step - loss: 4.8866e-04
Epoch 23/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 125ms/step - loss: 5.1001e-04
Epoch 24/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 121ms/step - loss: 4.6519e-04
Epoch 25/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 127ms/step - loss: 4.9052e-04
Epoch 26/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 148ms/step - loss: 4.6488e-04
Epoch 27/50
```

```
66/66 ━━━━━━━━━━━━━━━━━━━━ 13s 189ms/step - loss: 4.5195e-04
Epoch 28/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 13s 192ms/step - loss: 4.2816e-04
Epoch 29/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 11s 164ms/step - loss: 6.0922e-04
Epoch 30/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 123ms/step - loss: 4.4596e-04
Epoch 31/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 150ms/step - loss: 5.1473e-04
Epoch 32/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 124ms/step - loss: 4.4140e-04
Epoch 33/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 11s 163ms/step - loss: 4.3330e-04
Epoch 34/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 132ms/step - loss: 4.4552e-04
Epoch 35/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 13s 172ms/step - loss: 5.0070e-04
Epoch 36/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 128ms/step - loss: 3.7252e-04
Epoch 37/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 128ms/step - loss: 4.6072e-04
Epoch 38/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 4.4688e-04
Epoch 39/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 4.4656e-04
Epoch 40/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 13s 196ms/step - loss: 3.7011e-04
Epoch 41/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 10s 150ms/step - loss: 5.1348e-04
Epoch 42/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 125ms/step - loss: 4.0520e-04
Epoch 43/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 121ms/step - loss: 4.1531e-04
Epoch 44/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 133ms/step - loss: 3.7528e-04
Epoch 45/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 12s 184ms/step - loss: 3.8502e-04
Epoch 46/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 134ms/step - loss: 3.7837e-04
Epoch 47/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 13s 191ms/step - loss: 3.7919e-04
Epoch 48/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 5.0205e-04
Epoch 49/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 9s 131ms/step - loss: 4.4678e-04
Epoch 50/50
66/66 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 4.0386e-04
```

Out[20]:
```
<keras.src.callbacks.history.History at 0x2479b552050>
```

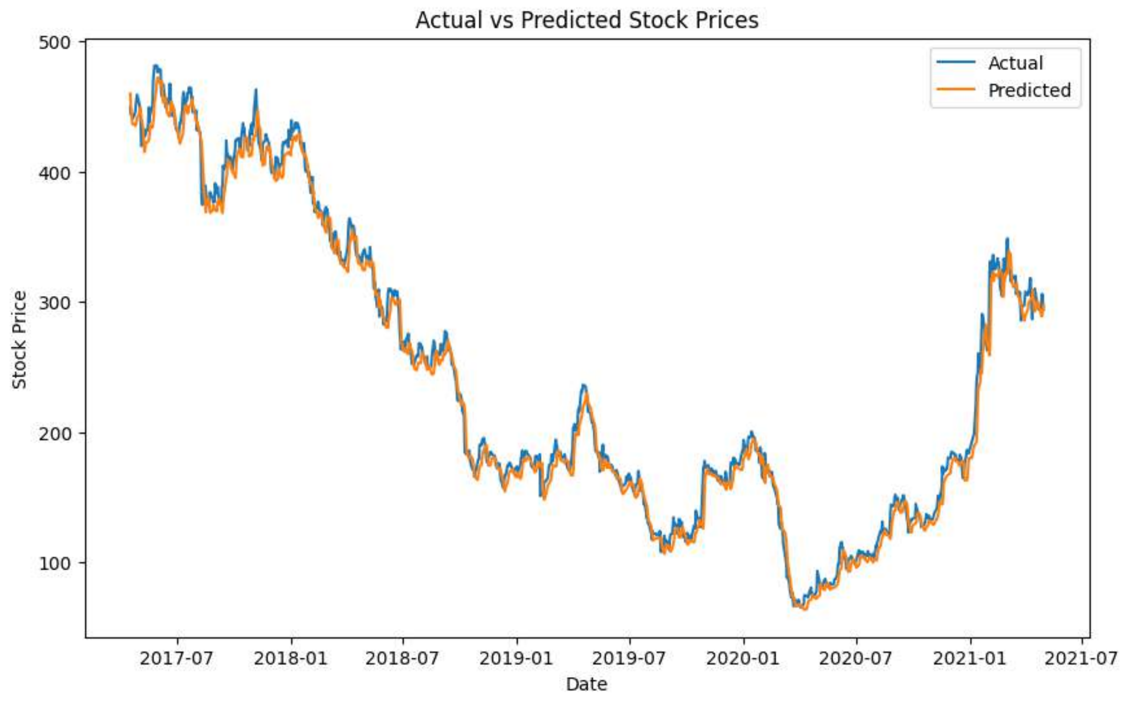
In [21]:
```python
# Make Predictions
predictions = model.predict(X_test)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step
```

In [23]:

```
# Check the shapes of the original data and predictions
print(f"Shape of data: {data['Close'].shape}")
print(f"Shape of predictions: {predictions.shape}")

# Adjust indexing for plotting if necessary
plt.figure(figsize=(10,6))
plt.plot(data.index[-len(predictions):], data['Close'][-len(predictions):]
plt.plot(data.index[-len(predictions):], predictions, label='Predicted')
plt.legend()
plt.title('Actual vs Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.show()
```

```
Shape of data: (5306,)
Shape of predictions: (1001, 1)
```



In [24]:

```
# Model Evaluation (e.g., RMSE)
rmse_arima = np.sqrt(mean_squared_error(data['Close'][-120:], arima_foreca
rmse_sarima = np.sqrt(mean_squared_error(data['Close'][-120:], sarima_fore
print(f'RMSE - ARIMA: {rmse_arima}')
```

```
RMSE - ARIMA: 78.82148102373868
RMSE - SARIMA: 81.08509781225528
```

In [25]:
```
# Forecasting Future Prices for the Next 120 Days
def forecast_future_trends(model, data_scaled, look_back, forecast_days=12
    predictions = []

    # Start with the last available data from the training set
    last_data = data_scaled[-look_back:]  # Take the last 'look_back' data

    for _ in range(forecast_days):
        # Reshape the input to fit the LSTM model
        input_data = np.reshape(last_data, (1, look_back, 1))

        # Predict the next data point
        next_pred = model.predict(input_data)

        # Append the predicted value to the predictions list
        predictions.append(next_pred[0, 0])

        # Update the last_data by adding the predicted value and removing
        last_data = np.append(last_data[1:], next_pred, axis=0)

    # Inverse transform the predicted values to get back the original scal
    predictions = np.array(predictions).reshape(-1, 1)
    forecasted_values = scaler.inverse_transform(predictions)
```

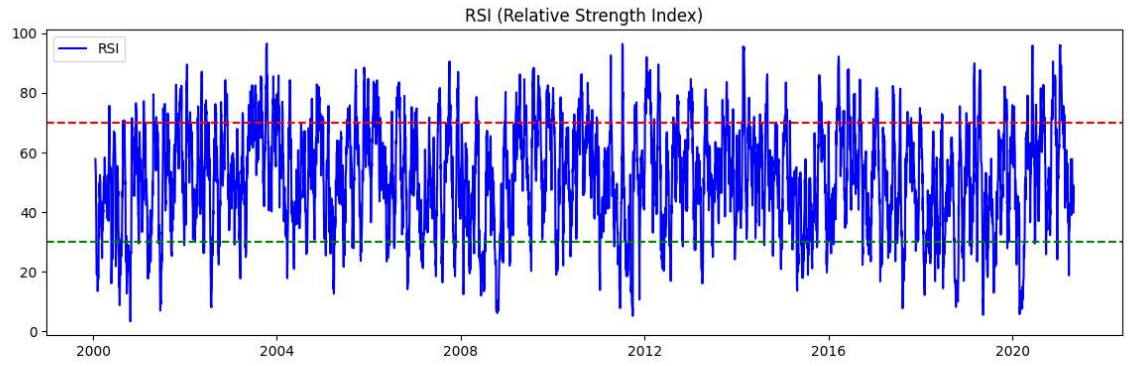
In [26]:
```
# Forecast for the next 120 days
forecast_days = 120
future_forecast = forecast_future_trends(model, test_scaled, look_back, fo

# Plot the forecasted future trend along with historical data
plt.figure(figsize=(10,6))
plt.plot(data.index, data['Close'], label='Historical Data')
future_dates = pd.date_range(data.index[-1], periods=forecast_days+1, clos
plt.plot(future_dates, future_forecast, label='Future Forecast', linestyle
plt.title(f'Tata Motors Stock Price Forecast for Next {forecast_days} Days
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
```

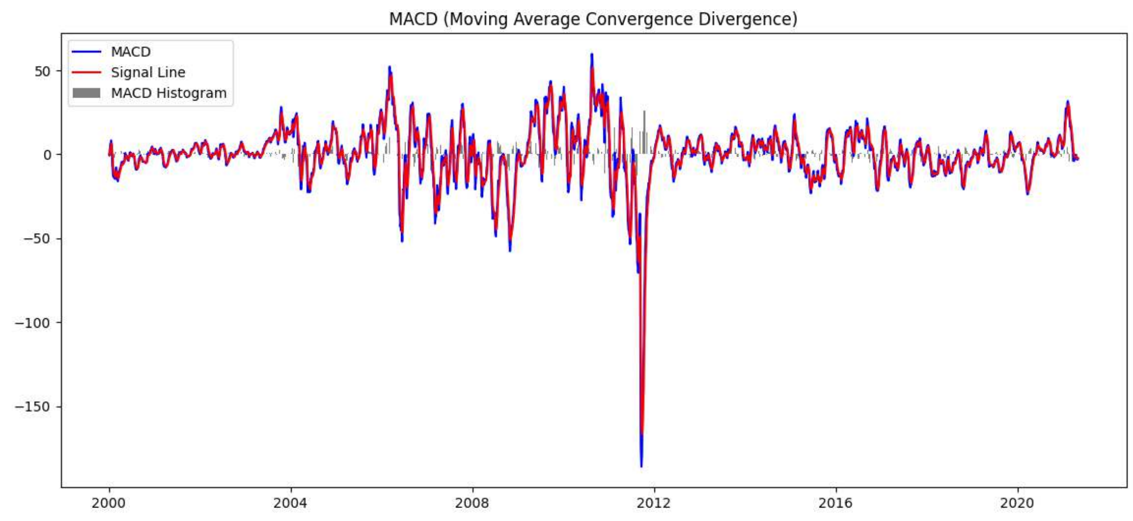
In [27]:
```
# Relative Strength Index (RSI)
def calculate_rsi(data, window=14):
    delta = data['Close'].diff(1)
    gain = (delta.where(delta > 0, 0)).fillna(0)
    loss = (-delta.where(delta < 0, 0)).fillna(0)

    avg_gain = gain.rolling(window=window).mean()
    avg_loss = loss.rolling(window=window).mean()

    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))

    return rsi

data['RSI'] = calculate_rsi(data)
```

In [28]:
```
# Moving Average Convergence Divergence (MACD)
def calculate_macd(data, fast_period=12, slow_period=26, signal_period=9):
    fast_ema = data['Close'].ewm(span=fast_period, adjust=False).mean()
    slow_ema = data['Close'].ewm(span=slow_period, adjust=False).mean()

    macd = fast_ema - slow_ema
    signal = macd.ewm(span=signal_period, adjust=False).mean()
    histogram = macd - signal

    return macd, signal, histogram

data['MACD'], data['Signal'], data['Histogram'] = calculate_macd(data)
```

In [34]:
```
import matplotlib.pyplot as plt

# RSI Plot
plt.figure(figsize=(14,4))
plt.plot(data['RSI'], label='RSI', color='blue')
plt.axhline(70, linestyle='--', color='red')  # Overbought line
plt.axhline(30, linestyle='--', color='green')  # Oversold line
plt.title('RSI (Relative Strength Index)')
plt.legend()
```



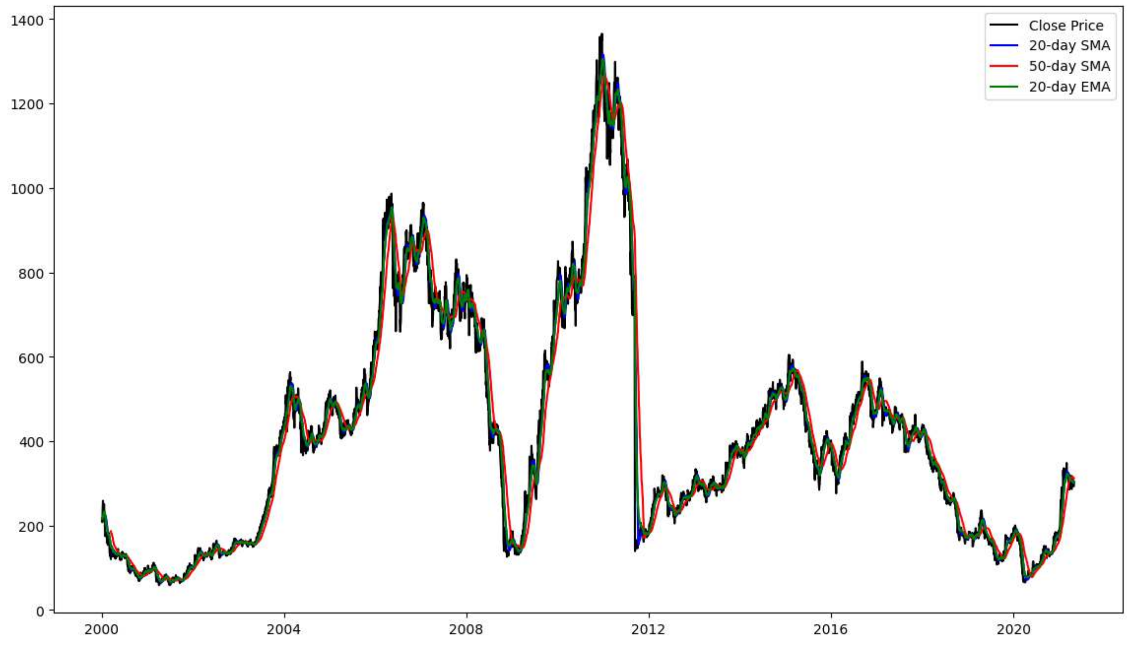
In [30]:
```
# MACD Plot
plt.figure(figsize=(14,6))
plt.plot(data['MACD'], label='MACD', color='blue')
plt.plot(data['Signal'], label='Signal Line', color='red')
plt.bar(data.index, data['Histogram'], label='MACD Histogram', color='gray
plt.title('MACD (Moving Average Convergence Divergence)')
plt.legend(loc='best')
```



In [31]:
```
# Simple Moving Average (SMA)
def calculate_sma(data, window):
    return data['Close'].rolling(window=window).mean()

data['SMA_20'] = calculate_sma(data, 20)
data['SMA_50'] = calculate_sma(data, 50)

# Exponential Moving Average (EMA)
def calculate_ema(data, window):
    return data['Close'].ewm(span=window, adjust=False).mean()

data['EMA_20'] = calculate_ema(data, 20)
data['EMA_50'] = calculate_ema(data, 50)
```

In [33]:
```
plt.figure(figsize=(14,8))

# Plot the stock prices and moving averages
plt.plot(data['Close'], label='Close Price', color='black')
plt.plot(data['SMA_20'], label='20-day SMA', color='blue')
plt.plot(data['SMA_50'], label='50-day SMA', color='red')
plt.plot(data['EMA_20'], label='20-day EMA', color='green')
```

Out[33]:
```
<matplotlib.legend.Legend at 0x247abcce2c0>
```



In [ ]: