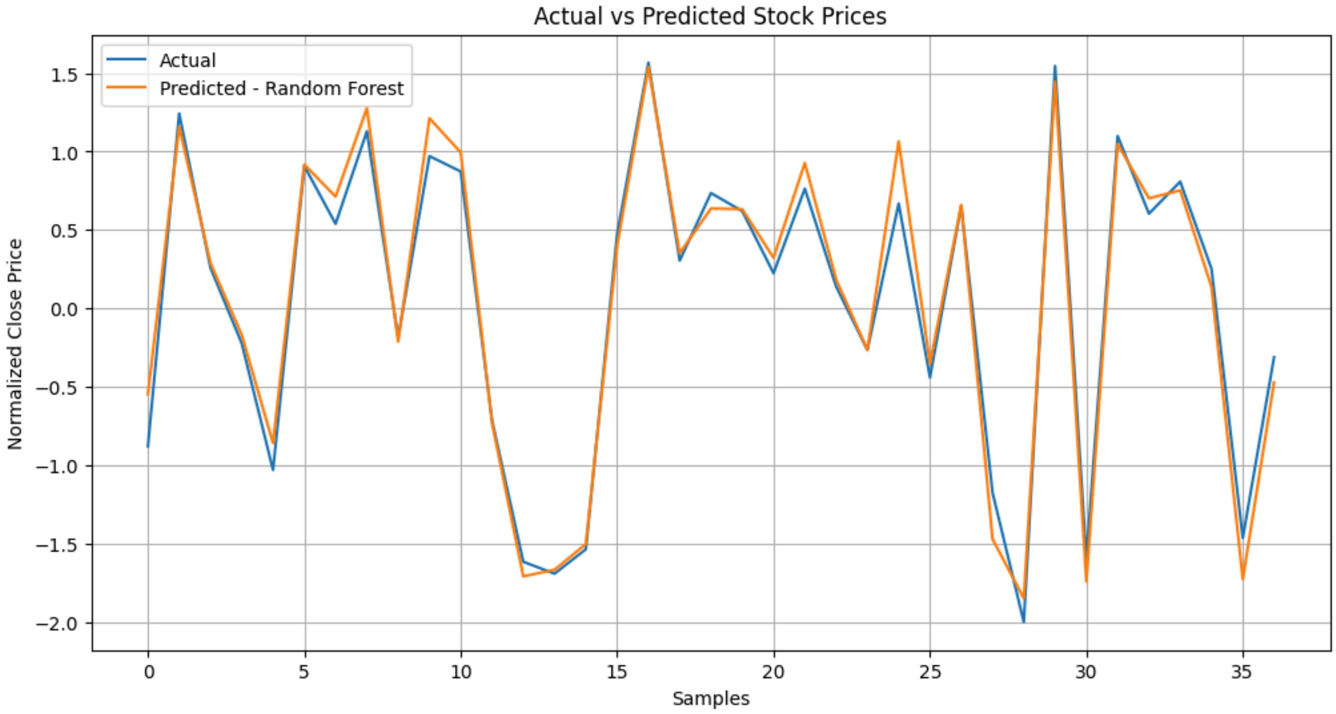
```
# STEP 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.preprocessing import StandardScaler, MinMaxScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# STEP 2: Load Dataset
df = pd.read_csv("/content/ai_stock_data (1).csv")  # Updated path
print("Initial DataFrame shape:", df.shape)
print(df.head())

# STEP 3: Data Preprocessing
# Convert 'Date' column if exists
if 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Drop rows with missing values and duplicates
df = df.dropna()
df = df.drop_duplicates()
print("Cleaned DataFrame shape:", df.shape)

# STEP 4: Feature Normalization (Standard Scaler)
features = ['Open', 'High', 'Low', 'Cls', 'Volume']  # Use 'Cls' for Close price
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

# STEP 5: Feature Engineering
df['MA_5'] = df['Cls'].rolling(window=5).mean()
df['MA_10'] = df['Cls'].rolling(window=10).mean()
df['RSI'] = df['Cls'].diff().apply(lambda x: max(x, 0)).rolling(window=14).mean()

df = df.dropna()

# STEP 6: Train-Test Split
X = df[['Open', 'High', 'Low', 'Volume', 'MA_5', 'MA_10', 'RSI']]
y = df['Cls']  # Use 'Cls' for Close price

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# STEP 7: Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# STEP 8: Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# STEP 9: Model Evaluation
def evaluate(y_true, y_pred, model_name):
    print(f"\n📊 {model_name} Evaluation:")
    print("MAE:", mean_absolute_error(y_true, y_pred))
    print("RMSE:", np.sqrt(mean_squared_error(y_true, y_pred)))
    print("R2 Score:", r2_score(y_true, y_pred))

evaluate(y_test, y_pred_lr, "Linear Regression")
evaluate(y_test, y_pred_rf, "Random Forest")

# STEP 10: Visualization
plt.figure(figsize=(12, 6))
plt.plot(y_test.values, label='Actual')
plt.plot(y_pred_rf, label='Predicted - Random Forest')
plt.legend()
plt.title("Actual vs Predicted Stock Prices")
plt.xlabel("Samples")
plt.ylabel("Normalized Close Price")
plt.grid(True)
plt.show()
```

```
# OPTIONAL STEP 11: LSTM Model (Univariate for 'Close' Price)

#  Prepare data for LSTM
df_lstm = df[['Cls']]  # Use 'Cls' for Close price
scaler_lstm = MinMaxScaler()
df_lstm_scaled = scaler_lstm.fit_transform(df_lstm)

X_seq, y_seq = [], []
window = 60

for i in range(window, len(df_lstm_scaled)):
    X_seq.append(df_lstm_scaled[i-window:i, 0])
    y_seq.append(df_lstm_scaled[i, 0])

X_seq, y_seq = np.array(X_seq), np.array(y_seq)
X_seq = X_seq.reshape((X_seq.shape[0], X_seq.shape[1], 1))

# Split into train/test
split_index = int(len(X_seq) * 0.8)
X_train_lstm, X_test_lstm = X_seq[:split_index], X_seq[split_index:]
y_train_lstm, y_test_lstm = y_seq[:split_index], y_seq[split_index:]

# Build LSTM Model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_seq.shape[1], 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train Model
model.fit(X_train_lstm, y_train_lstm, epochs=10, batch_size=32, verbose=1)

# Predict and Evaluate
y_pred_lstm = model.predict(X_test_lstm)
y_pred_lstm = scaler_lstm.inverse_transform(y_pred_lstm.reshape(-1, 1))
y_test_lstm_orig = scaler_lstm.inverse_transform(y_test_lstm.reshape(-1, 1))

# Plot LSTM Results
plt.figure(figsize=(12, 6))
plt.plot(y_test_lstm_orig, label='Actual')
plt.plot(y_pred_lstm, label='Predicted - LSTM')
plt.title("LSTM Model - Actual vs Predicted Stock Prices")
plt.xlabel("Samples")
plt.ylabel("Original Close Price")
plt.legend()
plt.grid(True)
plt.show()

from IPython import get_ipython
from IPython.display import display
# %%
# STEP 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.preprocessing import StandardScaler, MinMaxScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

import gradio as gr # Import Gradio

# STEP 2: Load Dataset
df = pd.read_csv("/content/ai_stock_data (1).csv")  # Updated path
print("Initial DataFrame shape:", df.shape)
print(df.head())

# STEP 3: Data Preprocessing
# Convert 'Date' column if exists
if 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# Drop rows with missing values and duplicates
df = df.dropna()
df = df.drop_duplicates()
```

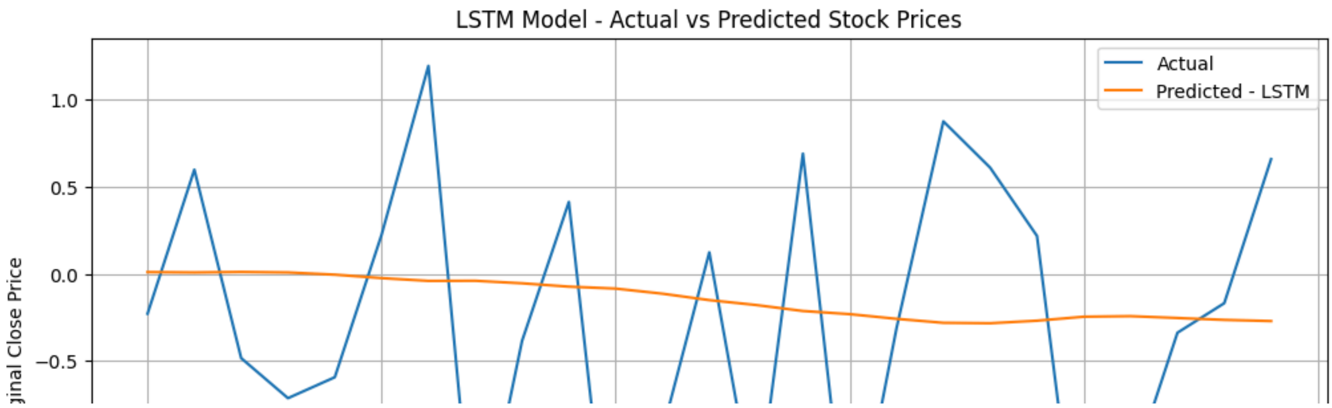
```
print("Cleaned DataFrame shape:", df.shape)

# STEP 4: Feature Normalization (Standard Scaler)
features = ['Open', 'High', 'Low', 'Cls', 'Volume']  # Use 'Cls' for Close price
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

# STEP 5: Feature Engineering
df['MA_5'] = df['Cls'].rolling(window=5).mean()
df['MA_10'] = df['Cls'].rolling(window=10).mean()
df['RSI'] = df['Cls'].diff().apply(lambda x: max(x, 0)).rolling(window=14).mean()

df = df.dropna()

# STEP 6: Train-Test Split
X = df[['Open', 'High', 'Low', 'Volume', 'MA_5', 'MA_10', 'RSI']]
y = df['Cls']  # Use 'Cls' for Close price

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# STEP 7: Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# STEP 8: Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# STEP 9: Model Evaluation
def evaluate(y_true, y_pred, model_name):
    print(f"\n📊 {model_name} Evaluation:")
    print("MAE:", mean_absolute_error(y_true, y_pred))
    print("RMSE:", np.sqrt(mean_squared_error(y_true, y_pred)))
    print("R2 Score:", r2_score(y_true, y_pred))

evaluate(y_test, y_pred_lr, "Linear Regression")
evaluate(y_test, y_pred_rf, "Random Forest")

# STEP 10: Visualization
plt.figure(figsize=(12, 6))
plt.plot(y_test.values, label='Actual')
plt.plot(y_pred_rf, label='Predicted - Random Forest')
plt.legend()
plt.title("Actual vs Predicted Stock Prices")
plt.xlabel("Samples")
plt.ylabel("Normalized Close Price")
plt.grid(True)
plt.show()

# OPTIONAL STEP 11: LSTM Model (Univariate for 'Close' Price)

#  Prepare data for LSTM
df_lstm = df[['Cls']]  # Use 'Cls' for Close price
scaler_lstm = MinMaxScaler()
df_lstm_scaled = scaler_lstm.fit_transform(df_lstm)

X_seq, y_seq = [], []
window = 60

for i in range(window, len(df_lstm_scaled)):
    X_seq.append(df_lstm_scaled[i-window:i, 0])
    y_seq.append(df_lstm_scaled[i, 0])

X_seq, y_seq = np.array(X_seq), np.array(y_seq)
X_seq = X_seq.reshape((X_seq.shape[0], X_seq.shape[1], 1))

# Split into train/test
split_index = int(len(X_seq) * 0.8)
X_train_lstm, X_test_lstm = X_seq[:split_index], X_seq[split_index:]
y_train_lstm, y_test_lstm = y_seq[:split_index], y_seq[split_index:]

# Build LSTM Model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_seq.shape[1], 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train Model
model.fit(X_train_lstm, y_train_lstm, epochs=10, batch_size=32, verbose=1)
```

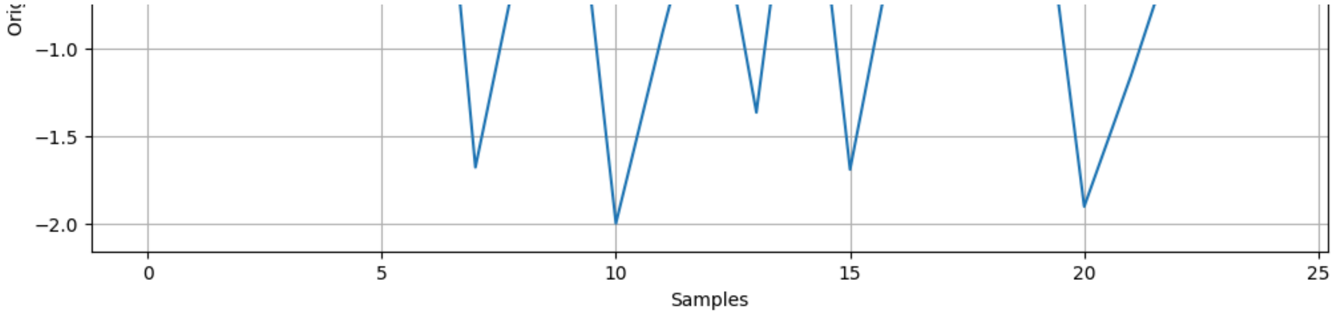
```
# Predict and Evaluate
y_pred_lstm = model.predict(X_test_lstm)
y_pred_lstm = scaler_lstm.inverse_transform(y_pred_lstm.reshape(-1, 1))
y_test_lstm_orig = scaler_lstm.inverse_transform(y_test_lstm.reshape(-1, 1))

# Plot LSTM Results
plt.figure(figsize=(12, 6))
plt.plot(y_test_lstm_orig, label='Actual')
plt.plot(y_pred_lstm, label='Predicted - LSTM')
plt.title("LSTM Model - Actual vs Predicted Stock Prices")
plt.xlabel("Samples")
plt.ylabel("Original Close Price")
plt.legend()
plt.grid(True)
plt.show()

# Gradio Deployment
# Install Gradio (if not already installed)
try:
    import gradio as gr
except ImportError:
    print("Installing gradio...")
    !pip install gradio==3.48.0
    import gradio as gr

# Define the prediction function for the Gradio interface
def predict_stock_price(Open, High, Low, Volume, MA_5, MA_10, RSI):
    # Scale the input features using the same scaler used during training
    features_input = np.array([[Open, High, Low, Volume, MA_5, MA_10, RSI]])
    # Ensure the scaler is fitted to the training data before this step
    # In this notebook, the scaler is fitted in STEP 4
    scaled_features = scaler.transform(features_input)

    # Make the prediction using the trained Random Forest model
    prediction = rf.predict(scaled_features)

    # Return the prediction
    return prediction[0]

# Create the Gradio interface
interface = gr.Interface(
    fn=predict_stock_price,
    inputs=[
        gr.Number(label="Open Price"),
        gr.Number(label="High Price"),
        gr.Number(label="Low Price"),
        gr.Number(label="Volume"),
        gr.Number(label="MA_5"),
        gr.Number(label="MA_10"),
        gr.Number(label="RSI")
    ],
    outputs="text",
    title="Stock Price Prediction (Random Forest)",
    description="Enter the stock features to predict the closing price using the Random Forest model."
)

# Launch the Gradio interface
interface.launch(share=True)
```

```
Initial DataFrame shape: (201, 6)
         Date    Open    High     Low     Cls  Volume
0  2024-01-01  129.66  133.97  125.17  130.58    4015
1  2024-01-02  161.87  167.03  155.56  157.36    9241
2  2024-01-03  148.60  156.97  142.22  154.84   17884
3  2024-01-04  138.68  142.55  132.78  135.50   18814
4  2024-01-05  153.94  157.17  148.90  149.65    8166
Cleaned DataFrame shape: (198, 6)

📊 Linear Regression Evaluation:
MAE: 7.447514619262843
RMSE: 44.743083847954296
R2 Score: -2040.63449461884

📊 Random Forest Evaluation:
MAE: 0.1096505270107092
RMSE: 0.1447553447480265
R2 Score: 0.9786304496988535
```

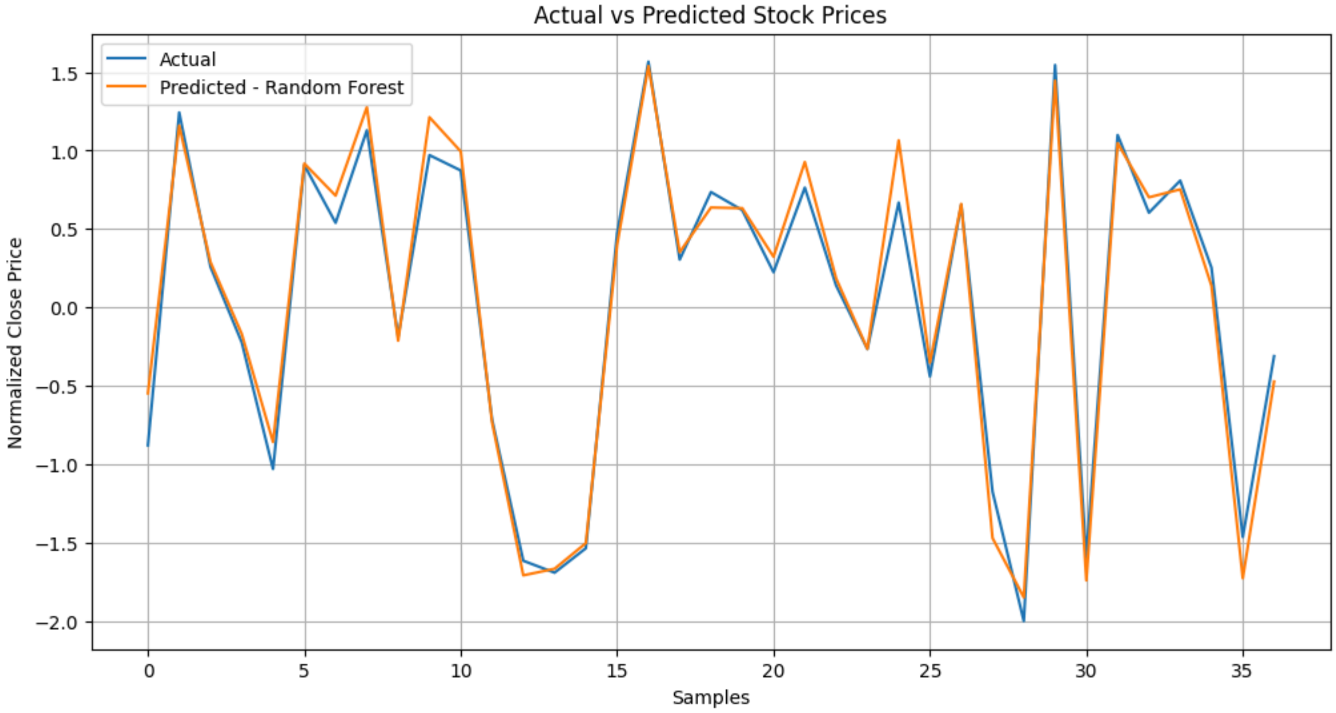



```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 52ms/step - loss: 0.4223
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.1734
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.0841
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.1274
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.0816
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.0832
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.0870
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.0734
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0750
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0760
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 342ms/step
```

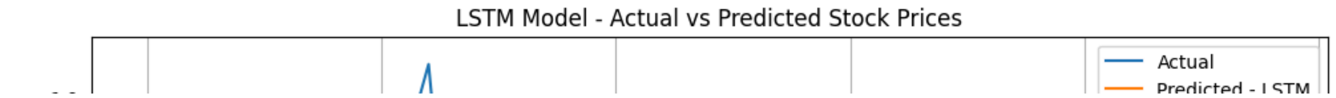


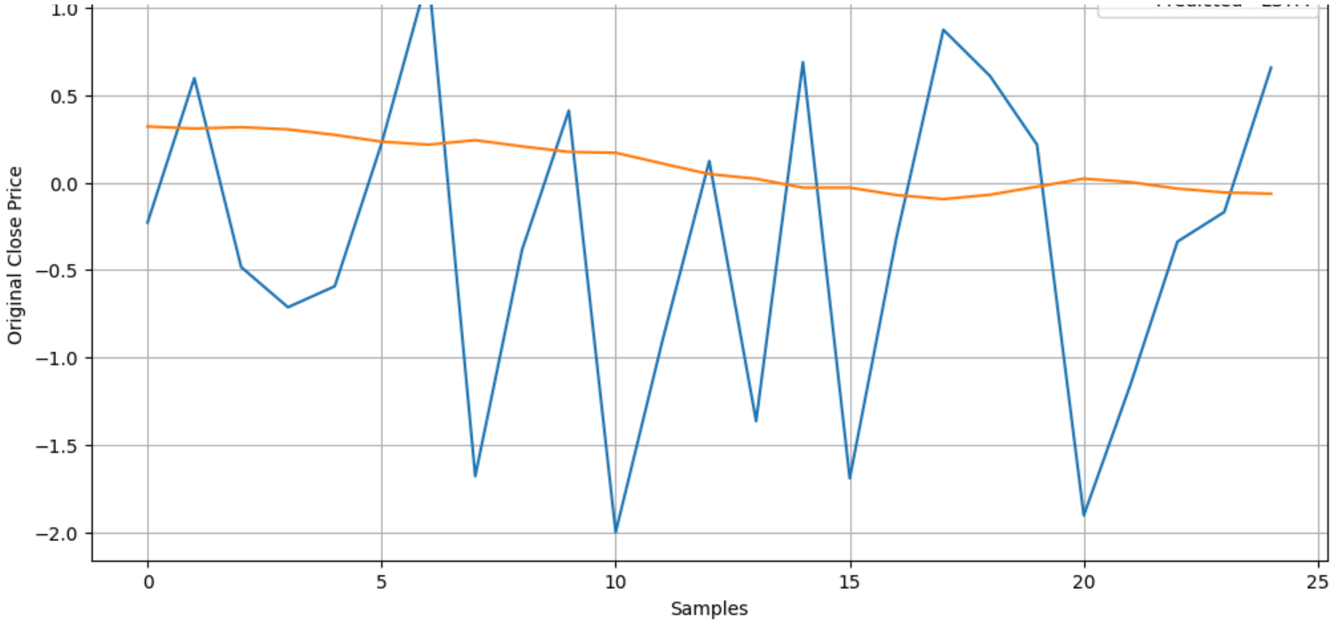



```
Initial DataFrame shape: (201, 6)
         Date    Open    High     Low     Cls  Volume
0  2024-01-01  129.66  133.97  125.17  130.58    4015
1  2024-01-02  161.87  167.03  155.56  157.36    9241
2  2024-01-03  148.60  156.97  142.22  154.84   17884
3  2024-01-04  138.68  142.55  132.78  135.50   18814
4  2024-01-05  153.94  157.17  148.90  149.65    8166
Cleaned DataFrame shape: (198, 6)

📊 Linear Regression Evaluation:
MAE: 7.447514619262843
RMSE: 44.743083847954296
R2 Score: -2040.63449461884

📊 Random Forest Evaluation:
MAE: 0.1096505270107092
RMSE: 0.1447553447480265
R2 Score: 0.9786304496988535
```




```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - loss: 0.3353
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.1049
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.1314
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.1006
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.0781
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.0885
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 0.0807
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.0736
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.0772
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0872
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
```






```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
IMPORTANT: You are using gradio version 3.48.0, however version 4.44.1 is available, please upgrade.
--------
Running on public URL: https://ea50979c79b9c4e8e8.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spa
```

Stock Price Prediction (Random Forest)

Enter the stock features to predict the closing price using the Random Forest model.

Open Price

0

High Price

0

Low Price

0

Volume

0

Start coding or generate with AI.