USD/ZAR Exchange Rate Forecasting with LSTM Neural Networks

This Python code implements a deep learning model for predicting foreign exchange rates, specifically the USD/ZAR (US Dollar to South African Rand) currency pair. The implementation uses a Long Short-Term Memory (LSTM) neural network architecture to capture temporal patterns in historical exchange rate data.

Core Components and Functionality

Data Acquisition and Preparation

- Retrieves historical USD/ZAR exchange rate data using the Yahoo Finance API (`yfinance`)
- Processes daily OHLC (Open, High, Low, Close) price data spanning from 2000 to 2025
- Includes manual adjustments for specific future dates (2025-01-15, 2025-01-17, 2024-11-15)
- Creates derived features including logarithmic returns, price ranges, and close-open differences

Feature Engineering

- Normalizes all input features using MinMaxScaler to ensure consistent scale for the neural network
- Constructs sequential data with a 20-day lookback window for time series forecasting
- Splits the dataset into training (80%) and validation (20%) sets

Model Architecture

- Implements a stacked LSTM neural network with:
- First LSTM layer with 64 units and sequence return
- Dropout layer (20%) for regularization to prevent overfitting
- Second LSTM layer with 32 units
- Dense output layer with 4 units (for predicting Open, High, Low, Close values)
- Compiles with Mean Absolute Error (MAE) loss function and Adam optimizer

Training Process

- Trains for 25 epochs with early stopping based on validation loss
- Uses batch size of 16 for gradient updates
- Monitors and visualizes training and validation loss curves

Evaluation and Visualization

- Generates predictions on the validation set and inverse-transforms to original scale
- Creates comprehensive visualizations comparing actual vs. predicted values:
- Time series plots of actual vs. predicted prices for each OHLC component
- Absolute error spread analysis for each price component
- Calculates performance metrics (MAE and RMSE) for each predicted variable

Future Prediction

- Implements a recursive prediction mechanism for forecasting 30 days into the future
- Creates prediction bounds using the model's Mean Absolute Error
- Visualizes future predictions with uncertainty bands
- Generates interactive candlestick charts with Plotly showing predicted price movements

The model demonstrates the application of deep learning techniques to financial time series forecasting, with particular attention to prediction accuracy, error analysis, and visualization of future price movements with confidence intervals.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import yfinance as yf
import random
import tensorflow as tf
```

```
# Download USD/ZAR exchange rate data
ticker = "USDZAR=X"
df = yf.download(ticker, interval="1d", start="2000-01-01", end="2025-05-04")

# Update 'Open' and 'Close' prices for a specific date
df.loc['2025-01-15', ['Close']] = [18.75480]
df.loc['2025-01-17', ['Close']] = [18.72077]
df.loc['2024-11-15', ['Close']] = [18.14312]

df
```

```
[*********************100%***********************]  1 of 1 completed
```

Price	Close	High	Low	Open	Volume
Ticker	USDZAR=X	USDZAR=X	USDZAR=X	USDZAR=X	USDZAR=X
Date					
2003-12-01	6.342300	6.410000	6.333800	6.390000	0
2003-12-02	6.295100	6.405000	6.270000	6.384000	0
2003-12-03	6.192400	6.330000	6.075100	6.315000	0
2003-12-04	6.335100	6.360100	6.127500	6.192400	0
2003-12-05	6.220100	6.424700	6.220100	6.357000	0
...	...	...	...	...	...
2025-04-28	18.682489	18.746401	18.517349	18.682489	0
2025-04-29	18.494499	18.585711	18.462601	18.494499	0
2025-04-30	18.513201	18.654400	18.501089	18.513201	0
2025-05-01	18.538980	18.704000	18.571899	18.538980	0
2025-05-02	18.540701	18.552151	18.342899	18.540701	0

5572 rows × 5 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Basic features
df['Return'] = np.log(df['Close'] / df['Close'].shift(1))
df['Range'] = df['High'] - df['Low']
df['Close-Open'] = df['Close'] - df['Open']
df.dropna(inplace=True)


features = ['Open', 'High', 'Low', 'Close', 'Volume', 'Return', 'Range', 'Close-Open']
target = ['Open', 'High', 'Low', 'Close']


# Scale data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()


scaled_features = scaler_x.fit_transform(df[features])
scaled_targets = scaler_y.fit_transform(df[target])


# Create sequences
# Prepare the dataset for LSTM

def create_sequences(X, y, window_size=20):
    Xs, ys = [], []
    for i in range(len(X) - window_size):
        Xs.append(X[i:i+window_size])
        ys.append(y[i+window_size])
    return np.array(Xs), np.array(ys)

window_size = 20
X, y = create_sequences(scaled_features, scaled_targets, window_size)

# Split into training and validation sets
split = int(0.8 * len(X))
X_train, X_val = X[:split], X[split:]
y_train, y_val = y[:split], y[split:]


model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.2),
    LSTM(32),
```

```
    Dense(4)
])

model.compile(loss='mae', optimizer=Adam(0.001))
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argur
  super().__init__(**kwargs)
```

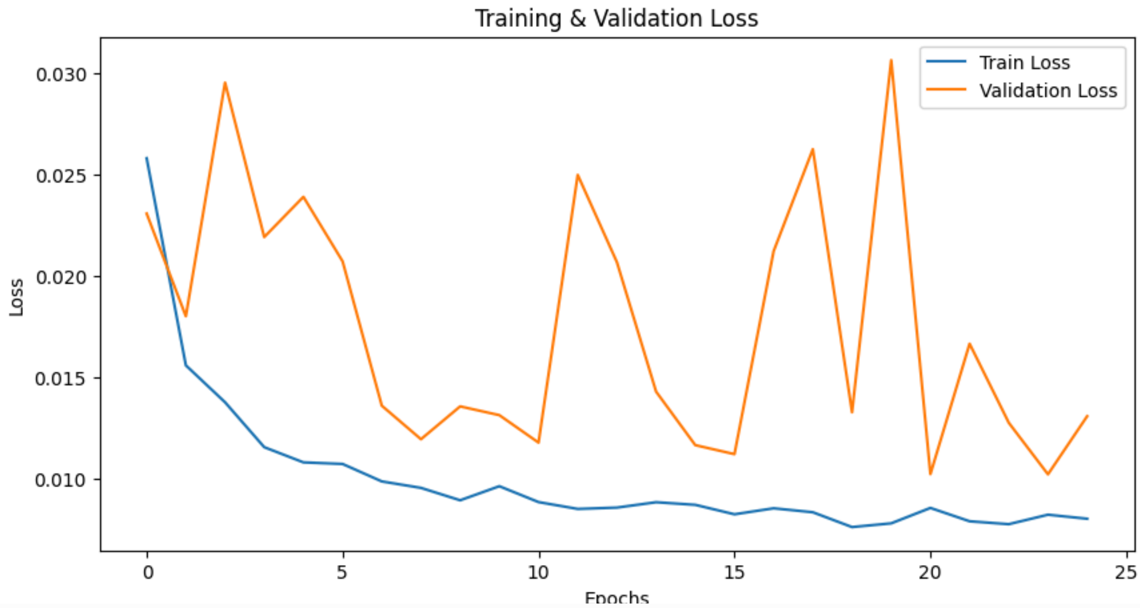
Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 20, 64)	18,688
dropout_1 (Dropout)	(None, 20, 64)	0
lstm_3 (LSTM)	(None, 32)	12,416
dense_1 (Dense)	(None, 4)	132

Total params: 31,236 (122.02 KB)
Trainable params: 31,236 (122.02 KB)
Non-trainable params: 0 (0.00 B)

```
# Train model
es = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=25, batch_size=16, validation_data=(X_val, y_val), callbacks=[es])

# Plot training & validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training & Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

```
Epoch 1/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 0.0478 - val_loss: 0.0231
Epoch 2/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 0.0165 - val_loss: 0.0180
Epoch 3/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 11s 21ms/step - loss: 0.0143 - val_loss: 0.0295
Epoch 4/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 0.0121 - val_loss: 0.0219
Epoch 5/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step - loss: 0.0108 - val_loss: 0.0239
Epoch 6/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - loss: 0.0108 - val_loss: 0.0207
Epoch 7/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 23ms/step - loss: 0.0098 - val_loss: 0.0136
Epoch 8/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 13s 32ms/step - loss: 0.0099 - val_loss: 0.0120
Epoch 9/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 0.0089 - val_loss: 0.0136
Epoch 10/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 0.0096 - val_loss: 0.0131
Epoch 11/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 9s 16ms/step - loss: 0.0087 - val_loss: 0.0118
Epoch 12/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 21ms/step - loss: 0.0088 - val_loss: 0.0250
Epoch 13/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - loss: 0.0086 - val_loss: 0.0207
Epoch 14/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - loss: 0.0086 - val_loss: 0.0143
Epoch 15/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 0.0092 - val_loss: 0.0117
Epoch 16/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 0.0080 - val_loss: 0.0112
Epoch 17/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 0.0084 - val_loss: 0.0212
Epoch 18/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 9s 17ms/step - loss: 0.0093 - val_loss: 0.0263
Epoch 19/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.0078 - val_loss: 0.0133
Epoch 20/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - loss: 0.0077 - val_loss: 0.0307
Epoch 21/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 0.0087 - val_loss: 0.0102
Epoch 22/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - loss: 0.0079 - val_loss: 0.0167
Epoch 23/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - loss: 0.0076 - val_loss: 0.0128
Epoch 24/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 0.0084 - val_loss: 0.0102
Epoch 25/25
278/278 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 0.0080 - val_loss: 0.0131
```

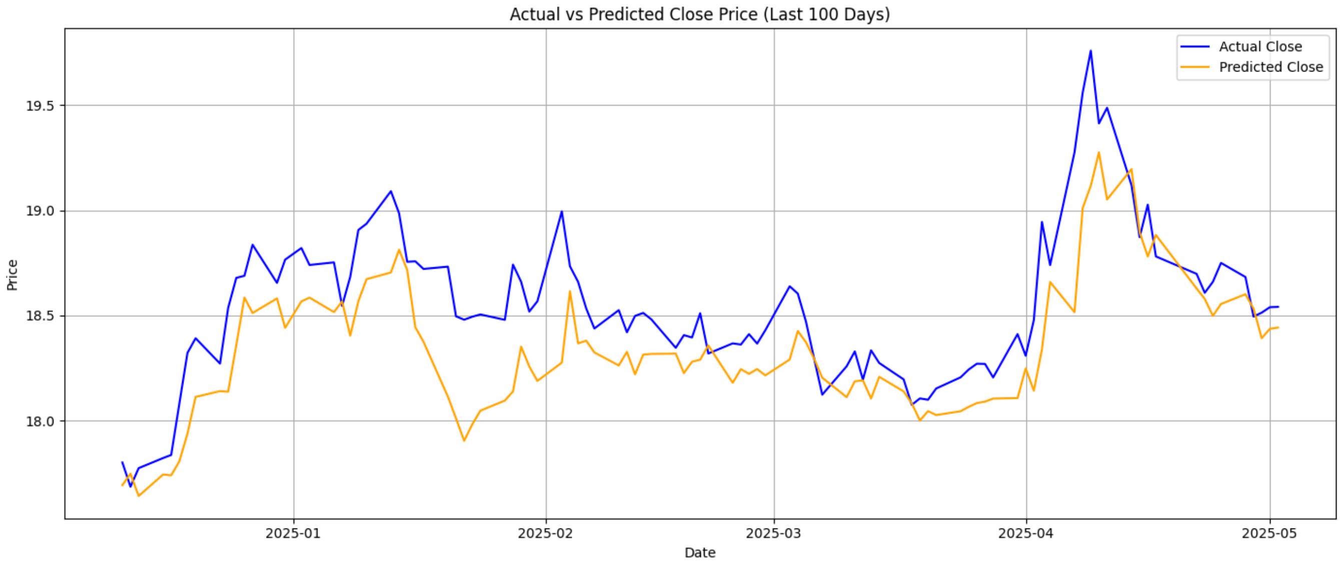



```
# Predict and invert scale
y_pred_scaled = model.predict(X_val)
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_true = scaler_y.inverse_transform(y_val)
```

```
35/35 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```

```
# Extract predicted and actual Close prices
pred_close = y_pred[:, 3]  # 3rd index = Close
```

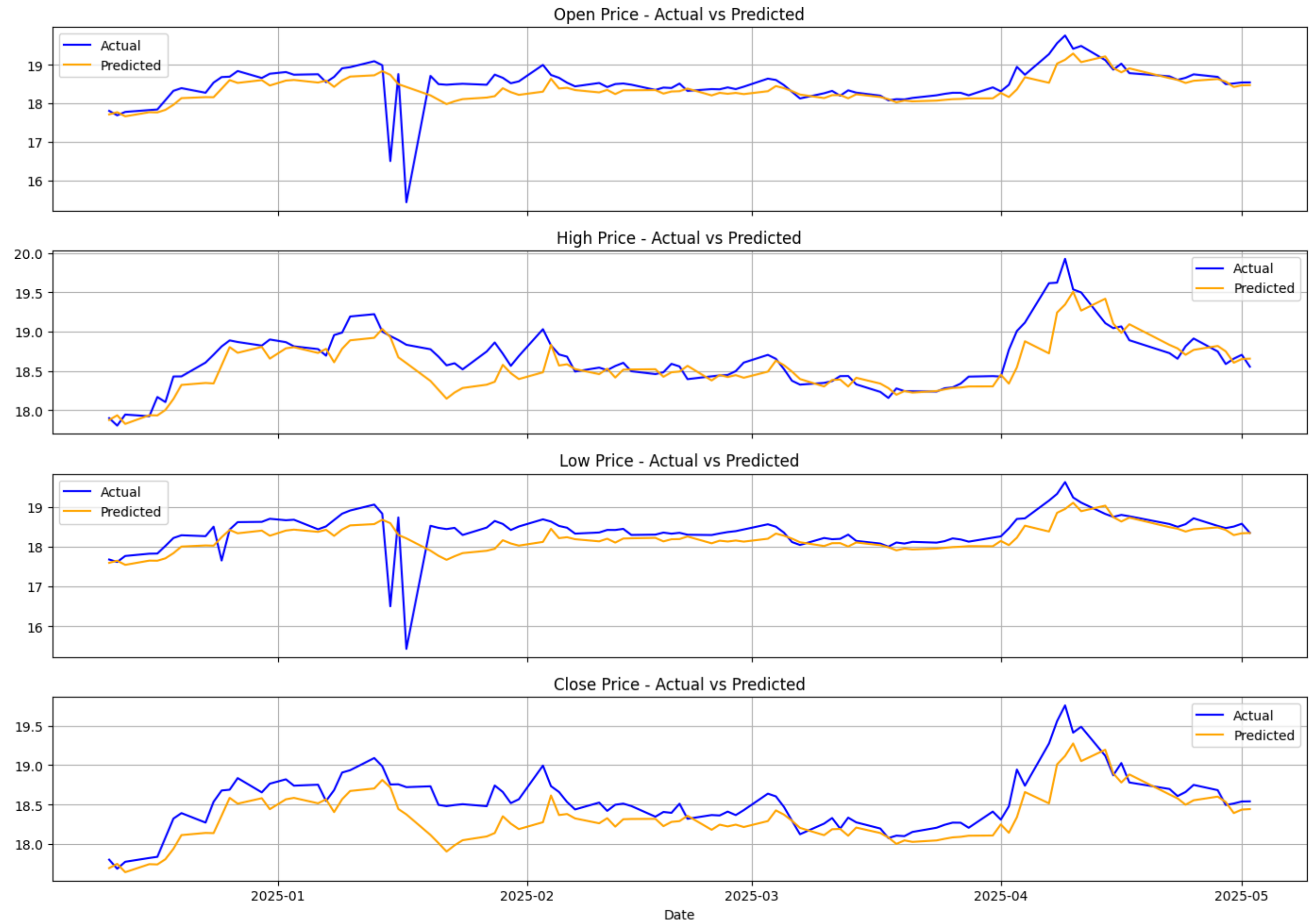
```
true_close = y_true[:, 3]


import matplotlib.pyplot as plt
import pandas as pd

#
# create DateTimeIndex
dates = df.index[-len(true_close):]  # Get the corresponding dates for the predictions

# Convert true_close and pred_close to pandas Series with the correct DateTimeIndex
true_close_series = pd.Series(true_close, index=dates, name='Actual Close')
pred_close_series = pd.Series(pred_close, index=dates, name='Predicted Close')

# Extract the last 100 days of data for both actual and predicted values
last_100_days = true_close_series.index[-100:]

# Get the corresponding actual and predicted close prices
actual_close_100 = true_close_series.loc[last_100_days]
predicted_close_100 = pred_close_series.loc[last_100_days]

# Plot the last 100 days of Actual vs Predicted Close prices
plt.figure(figsize=(14, 6))
plt.plot(actual_close_100, label='Actual Close', color='blue')
plt.plot(predicted_close_100, label='Predicted Close', color='orange')
plt.title("Actual vs Predicted Close Price (Last 100 Days)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
# Columns
ohlc_cols = ['Open', 'High', 'Low', 'Close']

# Create DataFrames
df_pred = pd.DataFrame(y_pred, columns=[f'Pred_{col}' for col in ohlc_cols])
df_true = pd.DataFrame(y_true, columns=[f'Actual_{col}' for col in ohlc_cols])

#  Create date index aligned to validation set
val_index = df.index[-len(y_true):]

# Combine all
df_ohlc_predictions = pd.concat([df_true, df_pred], axis=1)
df_ohlc_predictions.index = val_index


df_ohlc_predictions.tail(10)
```

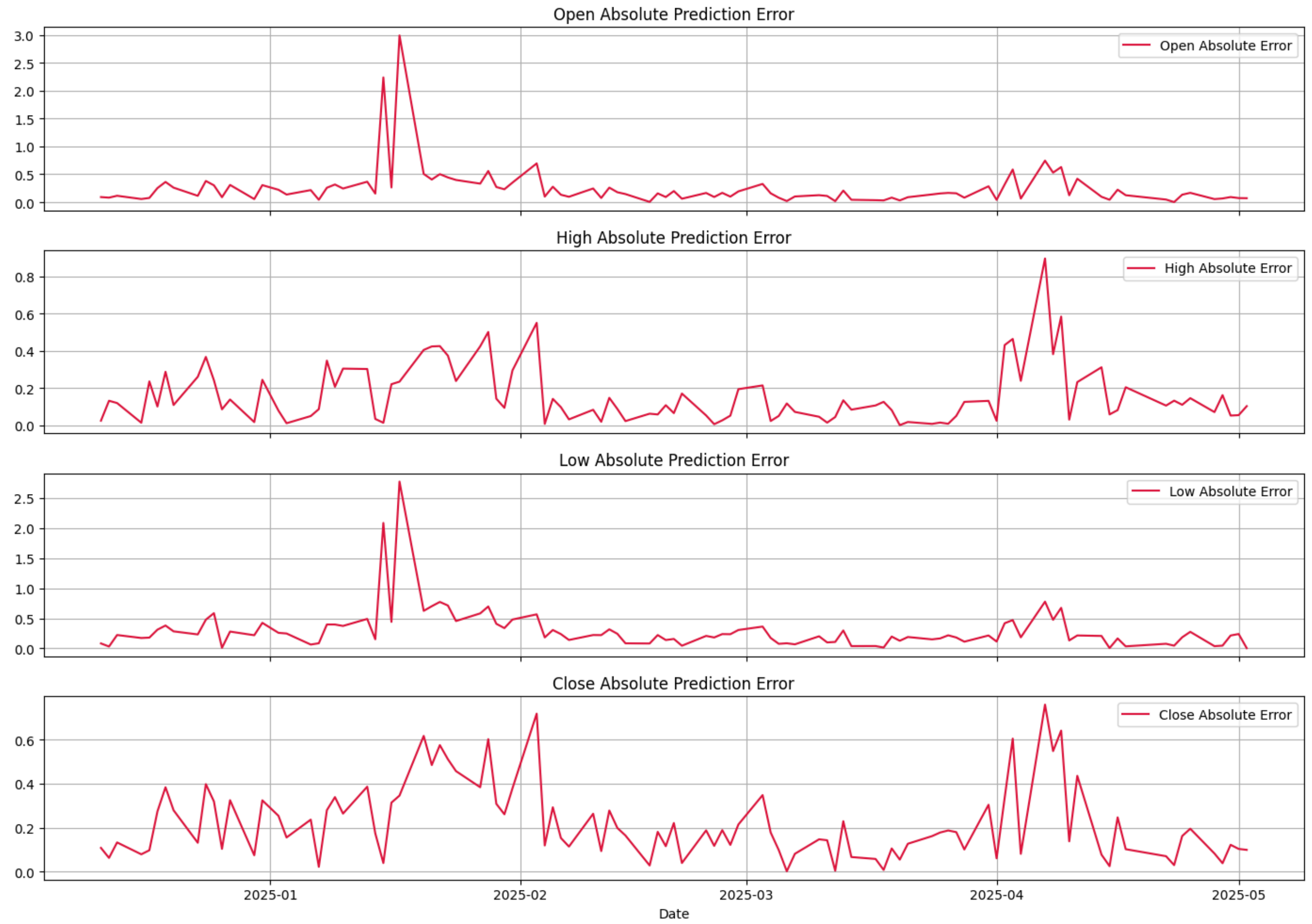
Date	Actual_Open	Actual_High	Actual_Low	Actual_Close	Pred_Open	Pred_High	Pred_Low	Pred_Close
2025-04-17	18.780251	18.889700	18.759300	18.780251	18.903996	19.093643	18.724722	18.881861
2025-04-22	18.697460	18.724819	18.563700	18.697460	18.651060	18.830212	18.486233	18.627819
2025-04-23	18.607519	18.653601	18.493700	18.607519	18.606558	18.785505	18.447681	18.578527
2025-04-24	18.659401	18.813400	18.560539	18.659401	18.525738	18.703709	18.375648	18.497892
2025-04-25	18.749310	18.911720	18.708549	18.749310	18.582714	18.766451	18.432859	18.554424
2025-04-28	18.682489	18.746401	18.517349	18.682489	18.627899	18.816961	18.479654	18.600630
2025-04-29	18.494499	18.585711	18.462601	18.494499	18.560278	18.747339	18.416092	18.532150
2025-04-30	18.513201	18.654400	18.501089	18.513201	18.421875	18.602554	18.286484	18.391684
2025-05-01	18.538980	18.704000	18.571899	18.538980	18.466227	18.649977	18.332373	18.436205
2025-05-02	18.540701	18.552151	18.342899	18.540701	18.469992	18.654615	18.339367	18.442217

```
import matplotlib.pyplot as plt

# Column names
ohlc_cols = ['Open', 'High', 'Low', 'Close']
n_days_to_plot = 100  # number of days to visualize

# Plot each OHLC value in a separate subplot
fig, axs = plt.subplots(4, 1, figsize=(14, 10), sharex=True)

for i, col in enumerate(ohlc_cols):
    axs[i].plot(df_ohlc_predictions[f'Actual_{col}'][-n_days_to_plot:], label='Actual', color='blue')
    axs[i].plot(df_ohlc_predictions[f'Pred_{col}'][-n_days_to_plot:], label='Predicted', color='orange')
    axs[i].set_title(f'{col} Price - Actual vs Predicted')
    axs[i].legend()
    axs[i].grid(True)

plt.xlabel("Date")
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
import matplotlib.pyplot as plt

# Compute absolute spread if not already done
for col in ['Open', 'High', 'Low', 'Close']:
    df_ohlc_predictions[f'{col}_Abs_Spread'] = (
        df_ohlc_predictions[f'Actual_{col}'] - df_ohlc_predictions[f'Pred_{col}']
    ).abs()

# Plot the last 100 days of absolute spread
fig, axs = plt.subplots(4, 1, figsize=(14, 10), sharex=True)

for i, col in enumerate(['Open', 'High', 'Low', 'Close']):
    axs[i].plot(df_ohlc_predictions[f'{col}_Abs_Spread'][-100:], label=f'{col} Absolute Error', color='crimson')
    axs[i].set_title(f'{col} Absolute Prediction Error')
    axs[i].legend()
    axs[i].grid(True)

plt.xlabel("Date")
plt.tight_layout()
plt.show()
```



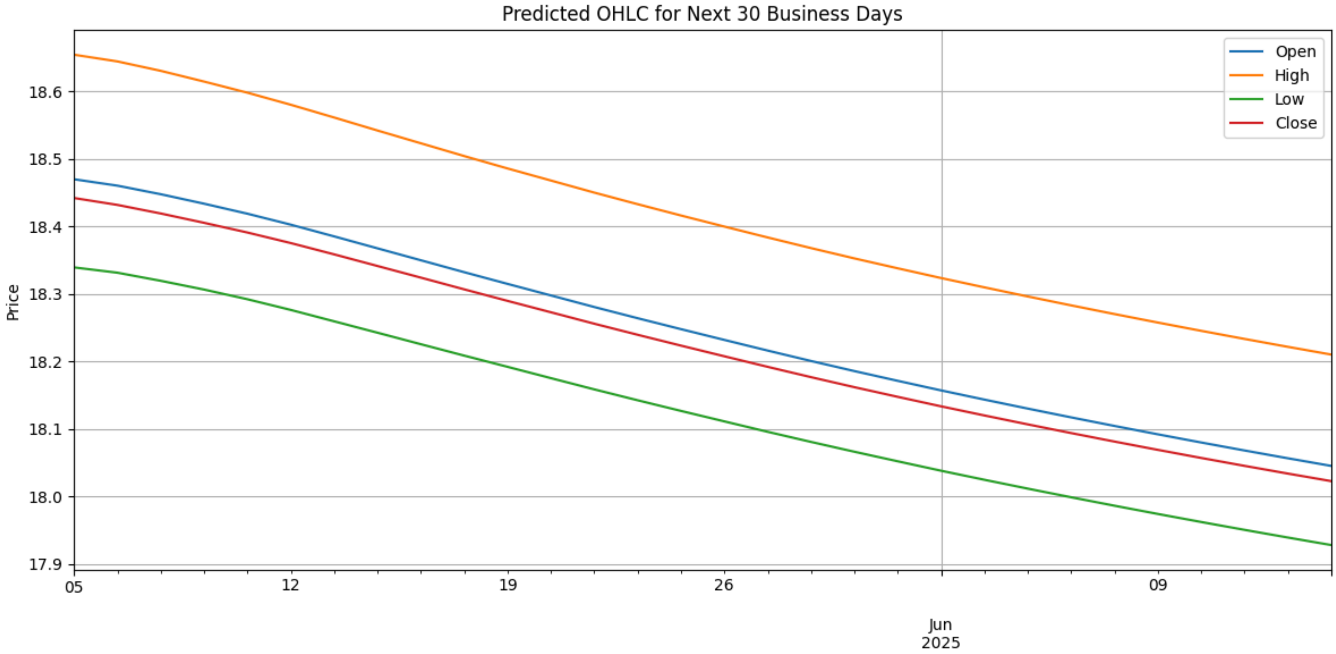

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Store metrics
metrics = {}

for col in ['Open', 'High', 'Low', 'Close']:
    actual = df_ohlc_predictions[f'Actual_{col}']
    predicted = df_ohlc_predictions[f'Pred_{col}']

    mae = mean_absolute_error(actual, predicted)
    rmse = np.sqrt(mean_squared_error(actual, predicted))

    metrics[col] = {'MAE': mae, 'RMSE': rmse}

# Convert to DataFrame for display
df_metrics = pd.DataFrame(metrics).T
print(df_metrics)
```

```
            MAE      RMSE
Open   0.149624  0.231277
High   0.146403  0.187208
Low    0.174054  0.252379
Close  0.148426  0.186816
```

Start coding or generate with AI.

```
def predict_future_days(model, X_last, scaler_x, scaler_y, n_days=30):
    future_preds = []
    current_sequence = X_last.copy()

    for _ in range(n_days):
        pred_scaled = model.predict(current_sequence.reshape(1, *current_sequence.shape), verbose=0)
        pred_actual = scaler_y.inverse_transform(pred_scaled)[0]
        future_preds.append(pred_actual)
```

```
        next_input = np.hstack([
            pred_actual,
            scaler_x.inverse_transform(current_sequence[-1].reshape(1, -1))[0][4:]
        ]).reshape(1, -1)

        next_input_scaled = scaler_x.transform(next_input)
        current_sequence = np.vstack([current_sequence[1:], next_input_scaled])

    return np.array(future_preds)


# === Predict ===
n_days = 30
last_window = X[-1]
future_ohlc = predict_future_days(model, last_window, scaler_x, scaler_y, n_days)

# === Create DataFrame ===
future_dates = pd.bdate_range(start=df.index[-1] + pd.Timedelta(days=1), periods=n_days)
future_df = pd.DataFrame(future_ohlc, columns=['Open', 'High', 'Low', 'Close'], index=future_dates)


# === Show and plot ===
future_df.plot(title=f"Predicted OHLC for Next {n_days} Business Days", figsize=(12, 6))
plt.ylabel("Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
future_df.head()
```

	Open	High	Low	Close
2025-05-05	18.469992	18.654617	18.339365	18.442215
2025-05-06	18.460518	18.644640	18.331486	18.431984
2025-05-07	18.447811	18.630686	18.319483	18.419247
2025-05-08	18.433765	18.614775	18.306494	18.405426
2025-05-09	18.418915	18.598261	18.292137	18.391047

Next steps: Generate code with future_df | View recommended plots | New interactive sheet

Bounded Predictions

```
# Define MAE values (you can replace these with computed ones)
mae_values = {
    'Open': 0.135752,
```

```
    'High': 0.173301,
    'Low': 0.160527,
    'Close': 0.136535
}

# Add columns for upper and lower bounds using ±MAE
for col in ['Open', 'High', 'Low', 'Close']:
    future_df[f'{col}_upper'] = future_df[col] + mae_values[col]
    future_df[f'{col}_lower'] = future_df[col] - mae_values[col]

# Optional: view a few rows
print(future_df[[col for col in future_df.columns if 'High' in col]].head())
```

```
                 High  High_upper  High_lower
2025-05-05  18.654617   18.827919   18.481316
2025-05-06  18.644640   18.817942   18.471338
2025-05-07  18.630686   18.803988   18.457384
2025-05-08  18.614775   18.788076   18.441473
2025-05-09  18.598261   18.771563   18.424959
```

```
# Optional: view a few rows
print(future_df[[col for col in future_df.columns if 'Close' in col]].head())
```

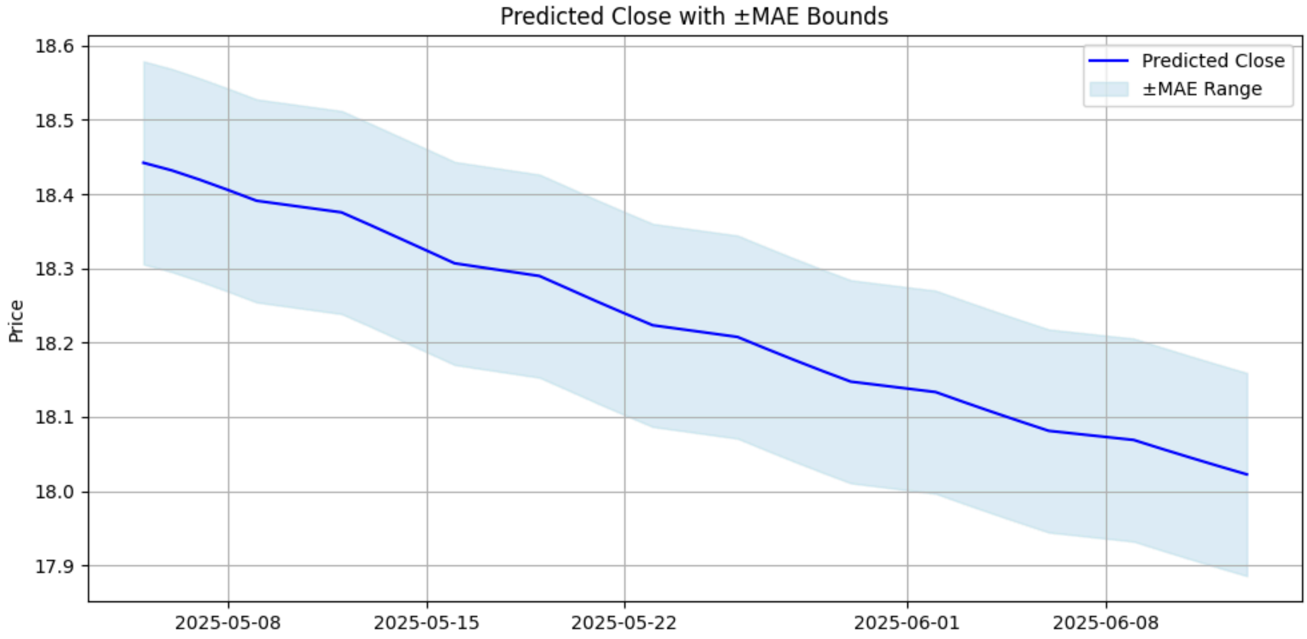
```
                Close  Close_upper  Close_lower
2025-05-05  18.442215    18.578751    18.305679
2025-05-06  18.431984    18.568520    18.295448
2025-05-07  18.419247    18.555782    18.282711
2025-05-08  18.405426    18.541962    18.268890
2025-05-09  18.391047    18.527582    18.254511
```

```
# Optional: view a few rows
print(future_df[[col for col in future_df.columns if 'Low' in col]].head())
```

```
                  Low  Low_upper  Low_lower
2025-05-05  18.339365  18.499891  18.178839
2025-05-06  18.331486  18.492012  18.170959
2025-05-07  18.319483  18.480009  18.158957
2025-05-08  18.306494  18.467020  18.145967
2025-05-09  18.292137  18.452663  18.131611
```

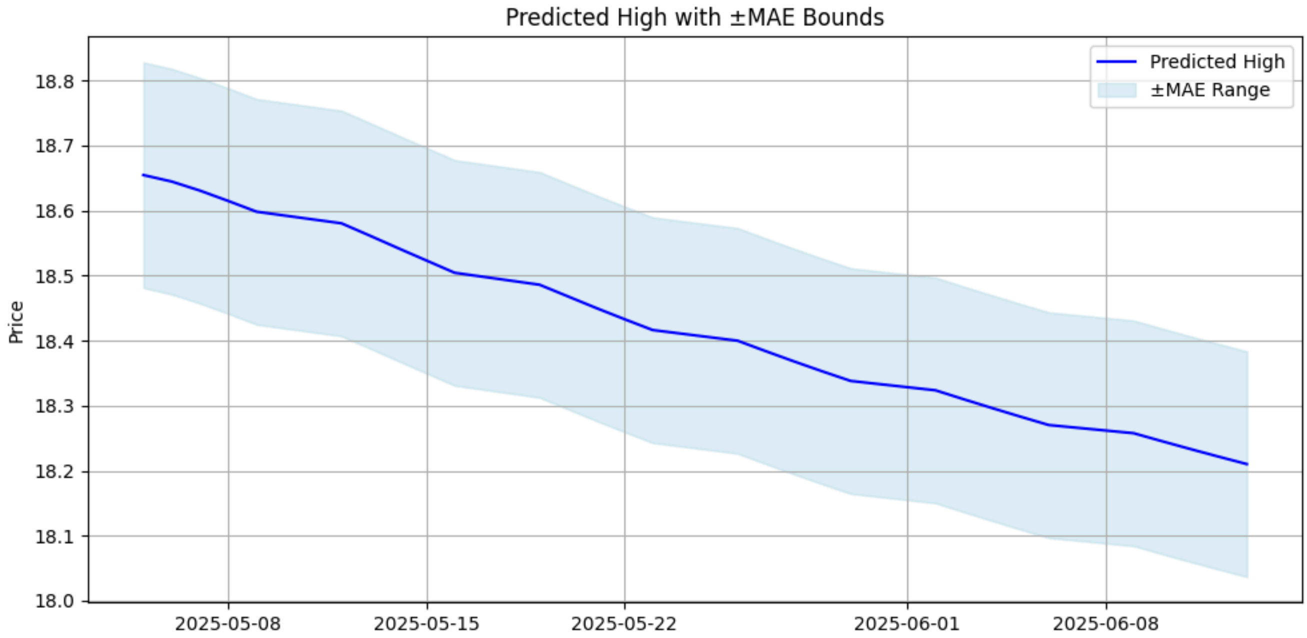
Pred Clos with Bounds

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(future_df.index, future_df['Close'], label='Predicted Close', color='blue')
plt.fill_between(future_df.index, future_df['Close_lower'], future_df['Close_upper'],
                 color='lightblue', alpha=0.4, label='±MAE Range')
plt.title('Predicted Close with ±MAE Bounds')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Pred High with Bounds

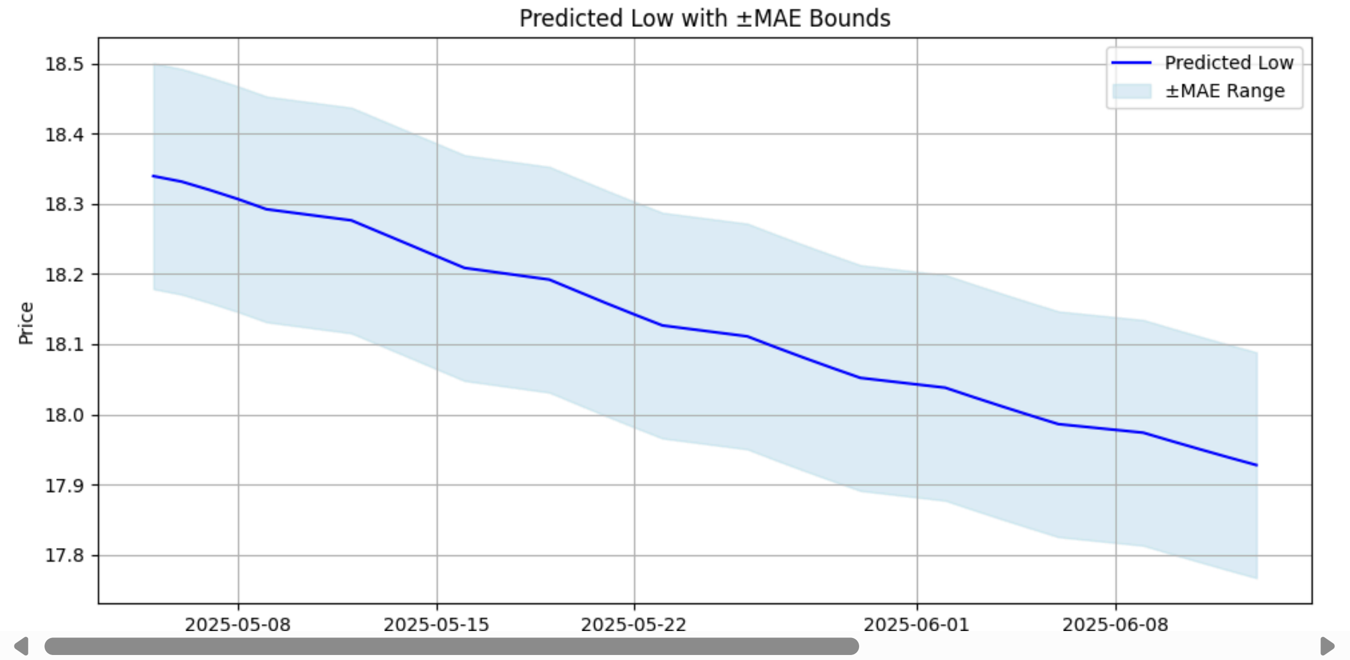
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(future_df.index, future_df['High'], label='Predicted High', color='blue')
plt.fill_between(future_df.index, future_df['High_lower'], future_df['High_upper'],
                 color='lightblue', alpha=0.4, label='±MAE Range')
plt.title('Predicted High with ±MAE Bounds')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Pred Low with Bounds

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(future_df.index, future_df['Low'], label='Predicted Low', color='blue')
plt.fill_between(future_df.index, future_df['Low_lower'], future_df['Low_upper'],
                 color='lightblue', alpha=0.4, label='±MAE Range')
```

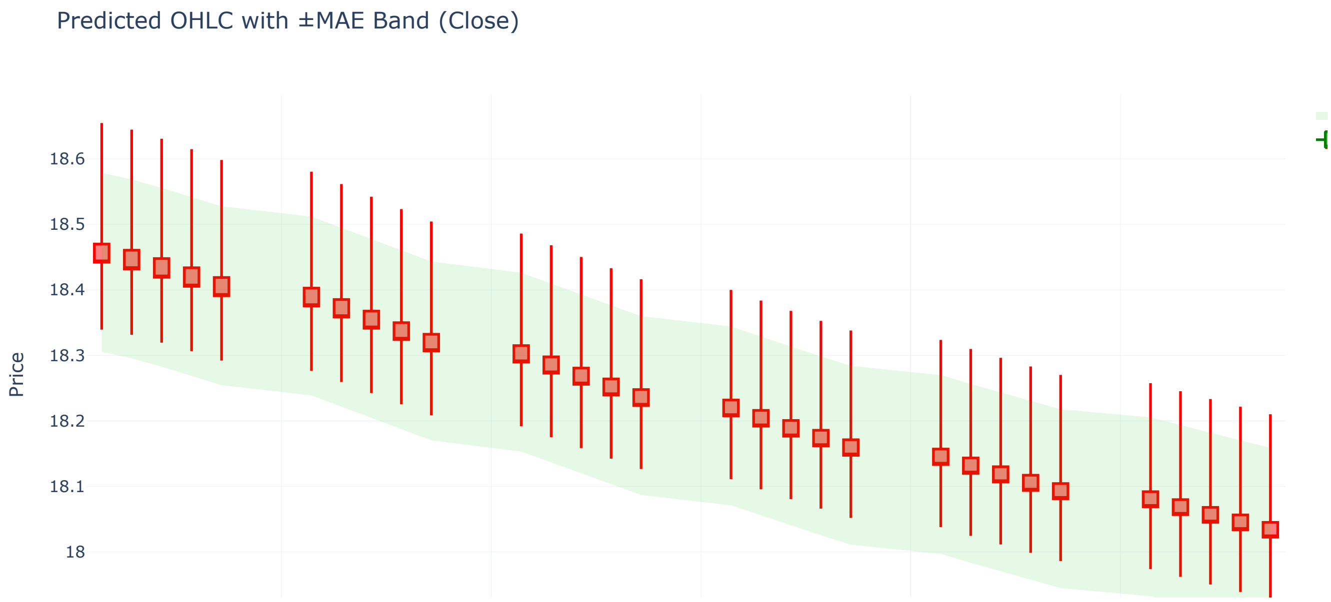
```
plt.title('Predicted Low with ±MAE Bounds')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
import plotly.graph_objects as go

# Prepare core candlestick
fig = go.Figure(data=[
    go.Candlestick(
        x=future_df.index,
        open=future_df['Open'],
        high=future_df['High'],
        low=future_df['Low'],
        close=future_df['Close'],
        name="Predicted OHLC",
        increasing_line_color='green',
        decreasing_line_color='red'
    )
])

# Add Close MAE bands as a shaded area
fig.add_trace(go.Scatter(
    x=future_df.index,
    y=future_df['Close_upper'],
    mode='lines',
    line=dict(width=0),
    name='Close Upper MAE',
    showlegend=False
))
fig.add_trace(go.Scatter(
    x=future_df.index,
    y=future_df['Close_lower'],
    fill='tonexty',
    fillcolor='rgba(0, 200, 0, 0.1)',  # Light green fill
    mode='lines',
    line=dict(width=0),
    name='±MAE Band',
))

# Final layout
fig.update_layout(
    title='Predicted OHLC with ±MAE Band (Close)',
    xaxis_title='Date',
    yaxis_title='Price',
    xaxis_rangeslider_visible=False,
    template='plotly_white',
    height=600
)
```

```
fig.show()
```


Predicted OHLC with ±MAE Band (Close)
Price
18.6
18.5
18.4
18.3
18.2
18.1
18