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
- 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="2025-08-04")

# 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
```

```
/tmp/ipython-input-3383502823.py:3: FutureWarning: YF.download() has changed argument auto_adjust default to True
  df = yf.download(ticker, interval="1d", start="2000-01-01", end="2025-08-04")
[*********************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
...	...	...	...	...	...
2025-07-28	17.738100	17.892401	17.711710	17.738100	0
2025-07-29	17.889900	17.987000	17.869730	17.889900	0
2025-07-30	17.867701	17.965799	17.830400	17.867701	0
2025-07-31	17.981779	18.175699	17.938101	17.981779	0
2025-08-01	18.210581	18.357309	18.016100	18.210581	0

5637 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', '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 = 7
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` argum
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 7, 64)	18,688
dropout (Dropout)	(None, 7, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dense (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()
```

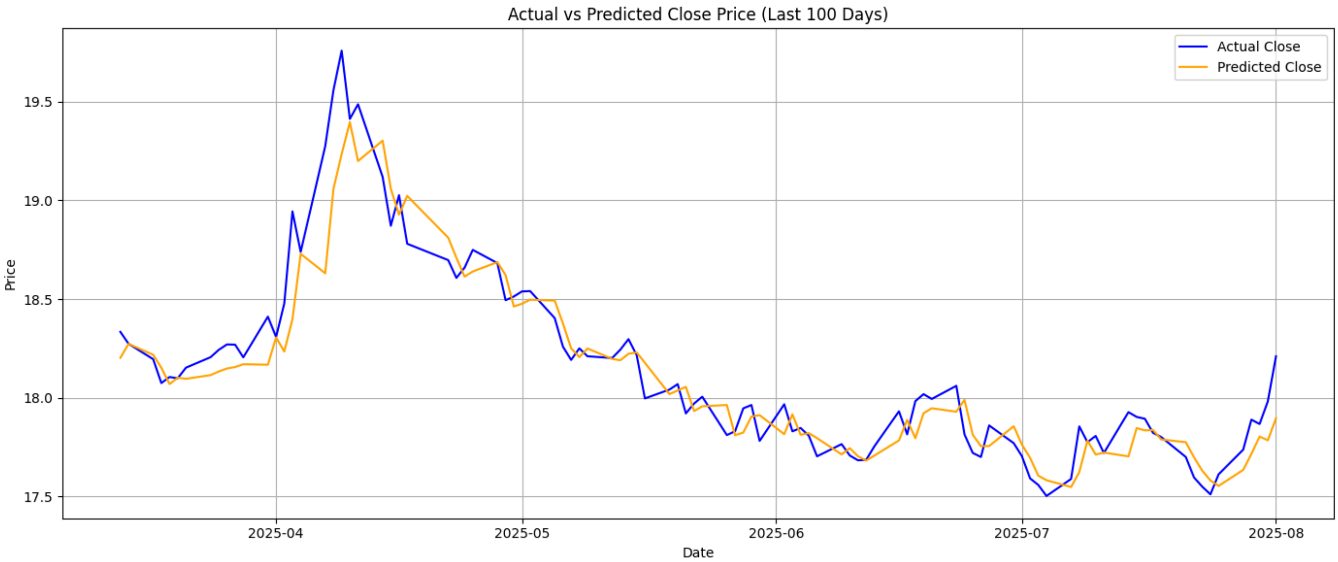
```
Epoch 1/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - loss: 0.0688 - val_loss: 0.0281
Epoch 2/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.0154 - val_loss: 0.0297
Epoch 3/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.0145 - val_loss: 0.0409
Epoch 4/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - loss: 0.0128 - val_loss: 0.0162
Epoch 5/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 0.0120 - val_loss: 0.0322
Epoch 6/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0109 - val_loss: 0.0315
Epoch 7/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0105 - val_loss: 0.0270
Epoch 8/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 7s 24ms/step - loss: 0.0101 - val_loss: 0.0182
Epoch 9/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0092 - val_loss: 0.0176
Epoch 10/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - loss: 0.0094 - val_loss: 0.0499
Epoch 11/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0103 - val_loss: 0.0276
Epoch 12/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0090 - val_loss: 0.0324
Epoch 13/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.0094 - val_loss: 0.0170
Epoch 14/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 0.0086 - val_loss: 0.0151
Epoch 15/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.0090 - val_loss: 0.0178
Epoch 16/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 0.0081 - val_loss: 0.0247
Epoch 17/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - loss: 0.0083 - val_loss: 0.0111
Epoch 18/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0084 - val_loss: 0.0125
Epoch 19/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.0077 - val_loss: 0.0157
Epoch 20/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - loss: 0.0085 - val_loss: 0.0144
Epoch 21/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - loss: 0.0078 - val_loss: 0.0095
Epoch 22/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - loss: 0.0079 - val_loss: 0.0174
Epoch 23/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.0077 - val_loss: 0.0169
Epoch 24/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - loss: 0.0074 - val_loss: 0.0100
Epoch 25/25
282/282 ━━━━━━━━━━━━━━━━━━━━ 6s 22ms/step - loss: 0.0084 - val_loss: 0.0174
```


```
# 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)
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step
```

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

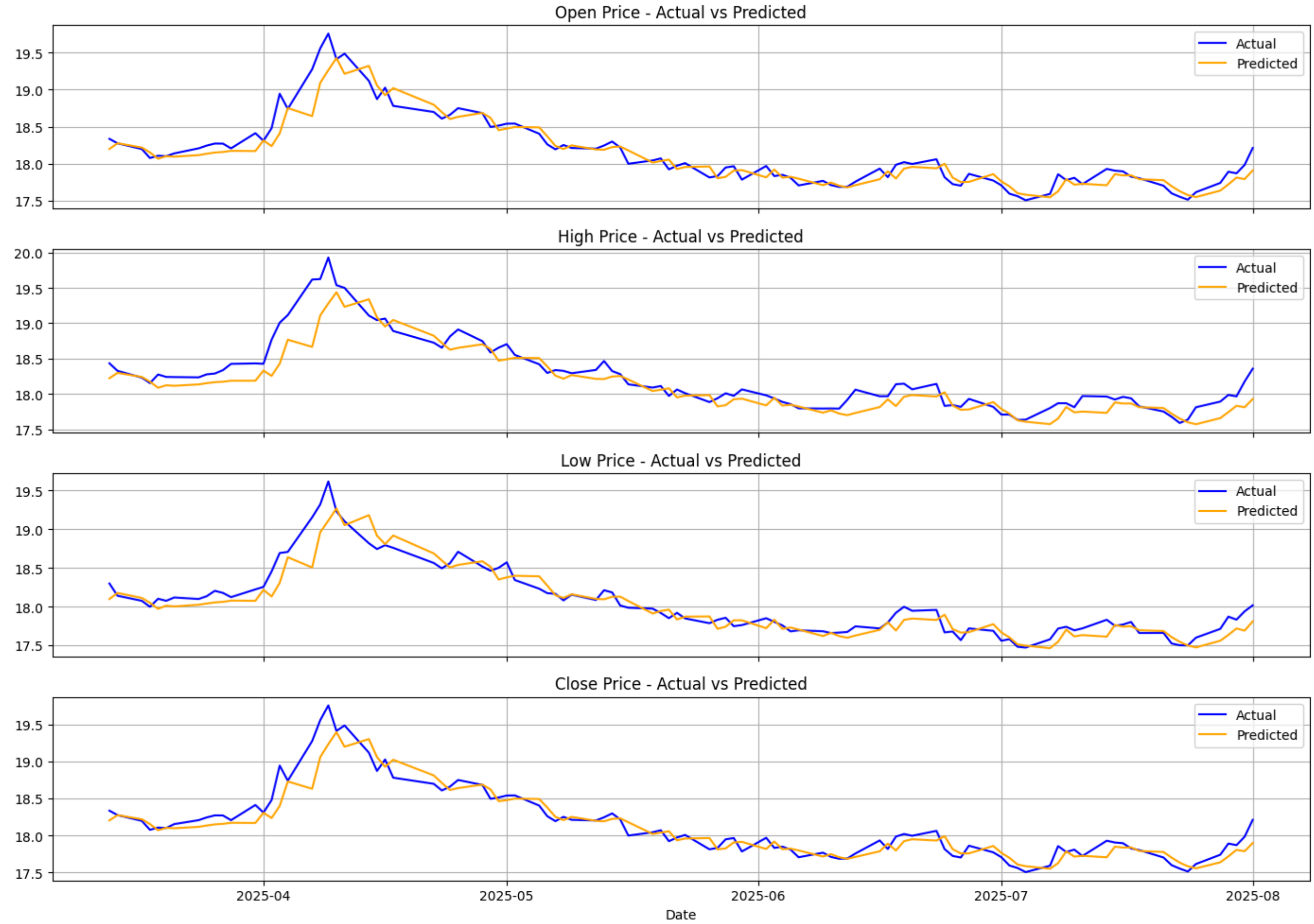
```
true_close = y_true[:, 3]


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)
```

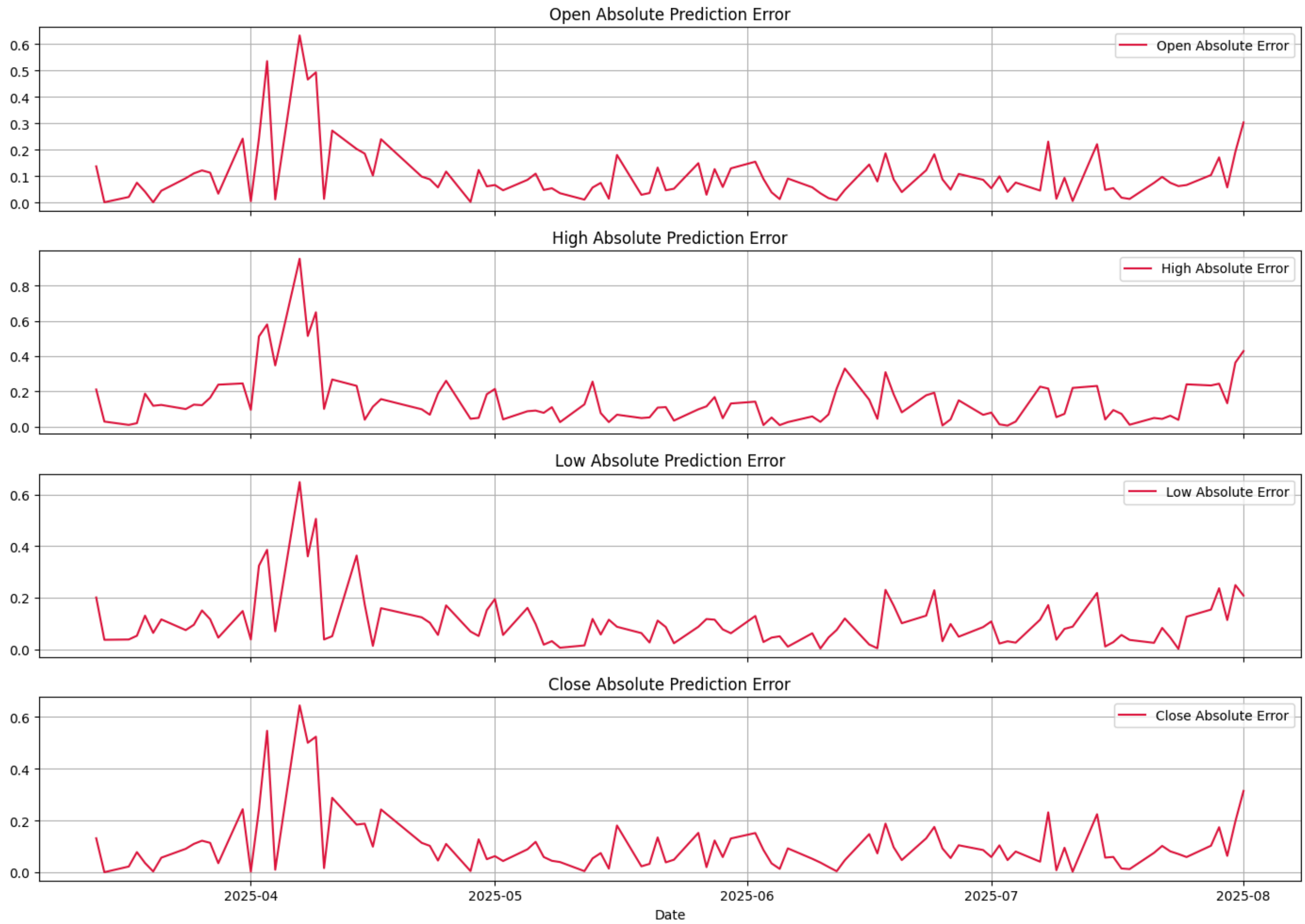
Date	Actual_Open	Actual_High	Actual_Low	Actual_Close	Pred_Open	Pred_High	Pred_Low	Pred_Close
2025-07-21	17.700800	17.751989	17.659300	17.700800	17.775143	17.801203	17.683809	17.775757
2025-07-22	17.597200	17.677000	17.522600	17.597200	17.694185	17.720970	17.604940	17.698940
2025-07-23	17.550900	17.589800	17.499500	17.550900	17.625557	17.651812	17.544600	17.631651
2025-07-24	17.511700	17.638399	17.497700	17.511700	17.573753	17.599857	17.496645	17.582094
2025-07-25	17.613100	17.812500	17.597900	17.613100	17.546879	17.572390	17.472015	17.554379
2025-07-28	17.738100	17.892401	17.711710	17.738100	17.634132	17.658731	17.558222	17.635593
2025-07-29	17.889900	17.987000	17.869730	17.889900	17.718901	17.743546	17.633396	17.716042
2025-07-30	17.867701	17.965799	17.830400	17.867701	17.810516	17.833035	17.717318	17.804438
2025-07-31	17.981779	18.175699	17.938101	17.981779	17.789673	17.812420	17.689421	17.785294
2025-08-01	18.210581	18.357309	18.016100	18.210581	17.907072	17.928711	17.808308	17.896656

```
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()
```



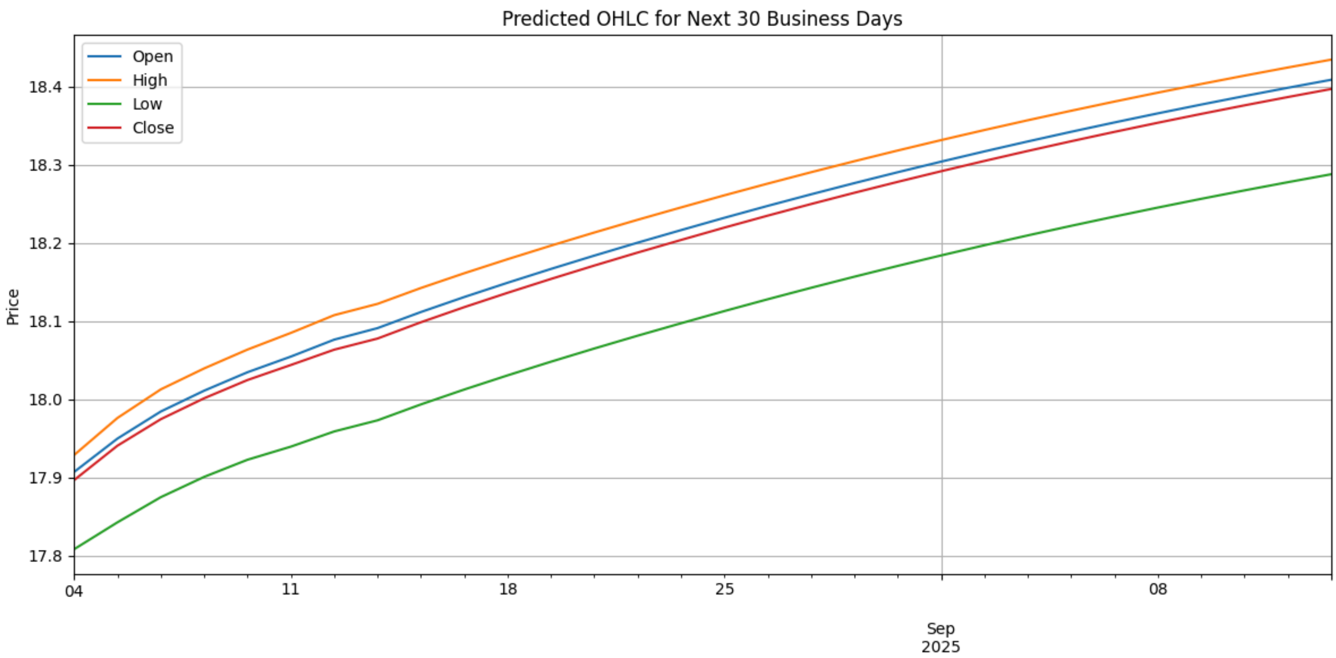

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# 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   0.132915  0.217476
High   0.164007  0.209444
Low    0.149458  0.229461
Close  0.128081  0.160842
```

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
2025-08-04	17.907072	17.928711	17.808306	17.896656
2025-08-05	17.949755	17.976114	17.842699	17.940664
2025-08-06	17.984350	18.012640	17.874866	17.974537
2025-08-07	18.010979	18.039417	17.900785	18.001272
2025-08-08	18.034548	18.063526	17.922701	18.024591

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
2025-08-04  17.928711   18.102013   17.755409
2025-08-05  17.976114   18.149416   17.802813
2025-08-06  18.012640   18.185942   17.839338
2025-08-07  18.039417   18.212719   17.866116
2025-08-08  18.063526   18.236828   17.890224
```

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

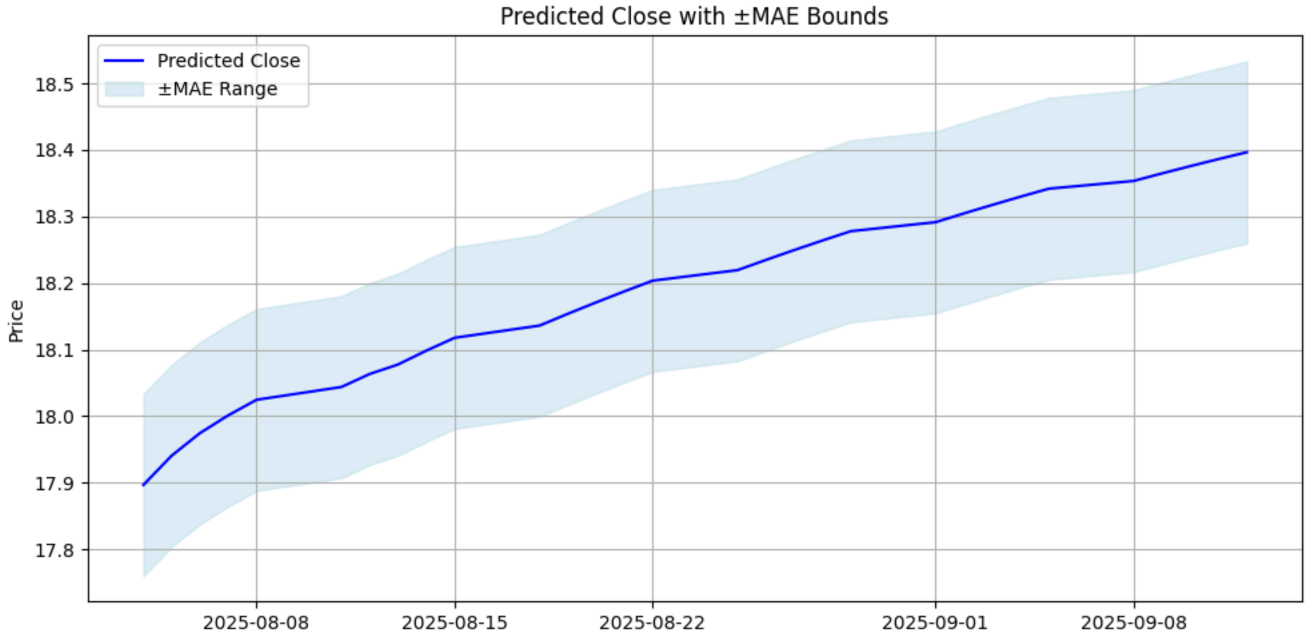
```
                Close  Close_upper  Close_lower
2025-08-04  17.896656    18.033192    17.760120
2025-08-05  17.940664    18.077200    17.804129
2025-08-06  17.974537    18.111073    17.838001
2025-08-07  18.001272    18.137808    17.864737
2025-08-08  18.024591    18.161127    17.888056
```

```
# 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
2025-08-04  17.808306  17.968832  17.647779
2025-08-05  17.842699  18.003225  17.682173
2025-08-06  17.874866  18.035393  17.714340
2025-08-07  17.900785  18.061312  17.740259
2025-08-08  17.922701  18.083227  17.762175
```

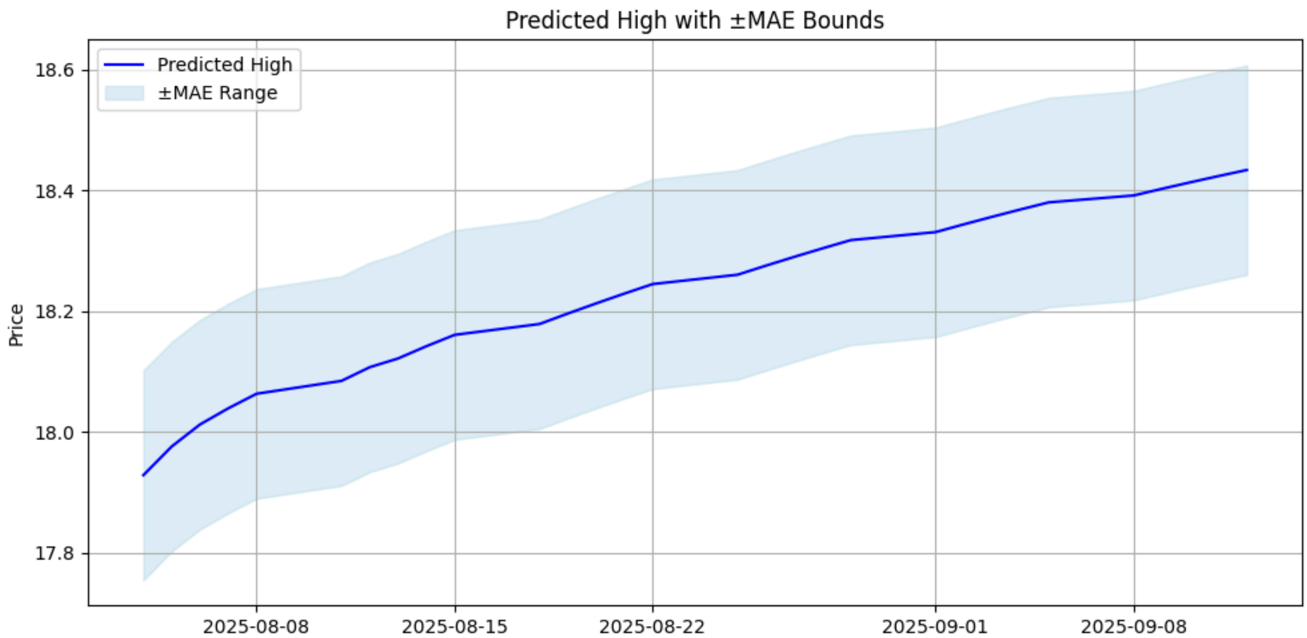
Pred Clos with Bounds

```
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

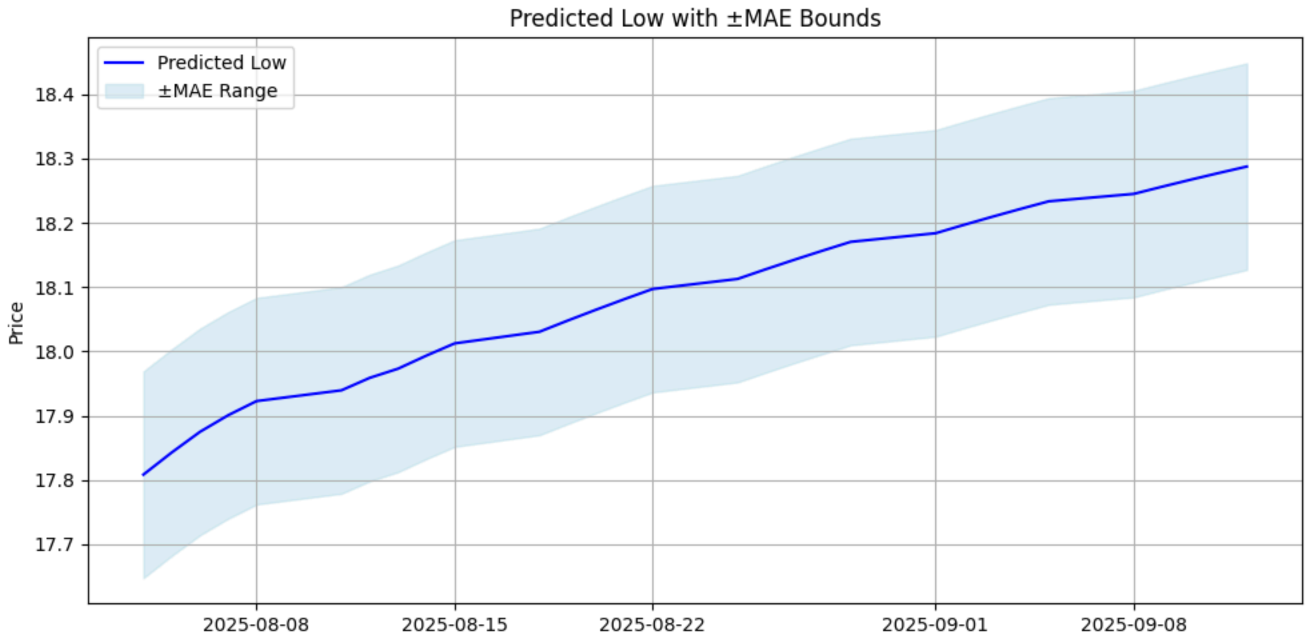
```
import matplotlib.pyplot as plt

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')
```

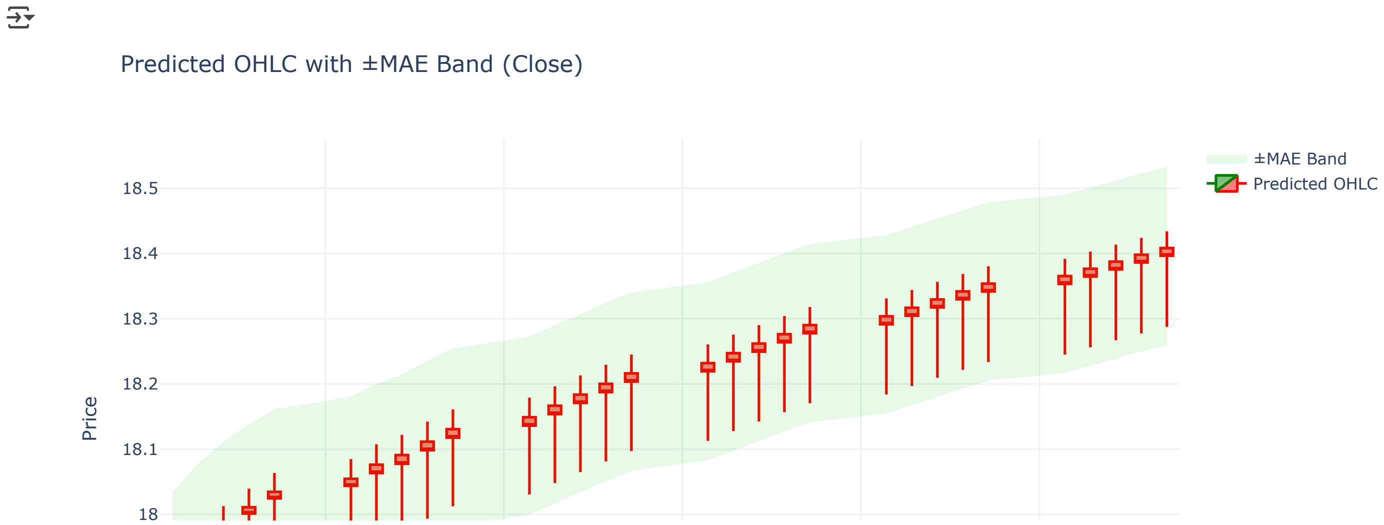
```
plt.title('Predicted Low with ±MAE Bounds')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



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
18.5
18.4
18.3
18.2
18.1
18
±MAE Band
Predicted OHLC