```
# Install common libraries
!pip -q install -U scikit-learn pandas numpy matplotlib tensorflow joblib

# Imports and basic setup
import os, io, zipfile, math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import joblib
import random

# plotting settings
%matplotlib inline
plt.rcParams['figure.figsize'] = (10,5)

# reproducibility
SEED = 42
np.random.seed(SEED)
random.seed(SEED)
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 91.2/91.2 kB 1.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.1/62.1 kB 3.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 38.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 65.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.6/16.6 MB 60.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 70.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 620.7/620.7 MB 2.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 73.7 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.3.2 which is incompatible.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 2.3.2 which is incompatible.
tensorflow-text 2.19.0 requires tensorflow<2.20,>=2.19.0, but you have tensorflow 2.20.0 which is incompatible.
cupy-cuda12x 13.3.0 requires numpy<2.3,>=1.22, but you have numpy 2.3.2 which is incompatible.
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 2.3.2 which is incompatible.
cudf-cu12 25.6.0 requires pandas<2.2.4dev0,>=2.0, but you have pandas 2.3.2 which is incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 2.3.2 which is incompatible.
dask-cudf-cu12 25.6.0 requires pandas<2.2.4dev0,>=2.0, but you have pandas 2.3.2 which is incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.3.2 which is incompatible.
tf-keras 2.19.0 requires tensorflow<2.20,>=2.19, but you have tensorflow 2.20.0 which is incompatible.
tensorflow-decision-forests 1.12.0 requires tensorflow==2.19.0, but you have tensorflow 2.20.0 which is incompatible.
```

```
file_path = None  # if you already have a path, set it here, e.g. "/content/tsla.csv"

if file_path is None:
    from google.colab import files
    print("Please upload your CSV or ZIP file (Kaggle dataset).")
    uploaded = files.upload()
    # take the first uploaded filename
    file_path = next(iter(uploaded.keys()))

print("Using file:", file_path)
```

```
Please upload your CSV or ZIP file (Kaggle dataset).
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving archive (1).zip to archive (1) (1).zip
Using file: archive (1) (1).zip
```

```
# Supports .csv or .zip (with a csv inside)
def load_csv_from_path(path):
    path = str(path)
    if path.lower().endswith('.csv'):
        return pd.read_csv(path)
    if path.lower().endswith('.zip'):
        z = zipfile.ZipFile(path)
        # pick first CSV inside
        csvs = [n for n in z.namelist() if n.lower().endswith('.csv')]
        if not csvs:
            raise ValueError("ZIP contains no CSV files.")
        with z.open(csvs[0]) as f:
            return pd.read_csv(f)
    raise ValueError("Please provide a .csv or .zip file.")

df = load_csv_from_path(file_path)
```

```
print("Raw shape:", df.shape)
display(df.head())
print("\nColumns:", list(df.columns))
```

Raw shape: (2516, 20)

	date	open	high	low	close	volume	rsi_7	rsi_14	cci_7	cci_14	sma_50	ema_50	sma_10
0	2014-01-02	9.986667	10.165333	9.770000	10.006667	92826000	55.344071	54.440118	-37.373644	15.213422	9.682107	9.820167	10.49424
1	2014-01-03	10.000000	10.146000	9.906667	9.970667	70425000	53.742629	53.821521	-81.304471	17.481130	9.652800	9.826069	10.49569
2	2014-01-06	10.000000	10.026667	9.682667	9.800000	80416500	46.328174	50.870410	-123.427544	-37.824708	9.629467	9.825047	10.49674
3	2014-01-07	9.841333	10.026667	9.683333	9.957333	75511500	53.263037	53.406750	-84.784651	-20.779431	9.597747	9.830235	10.50340
4	2014-01-08	9.923333	10.246667	9.917333	10.085333	92448000	58.368660	55.423026	60.799662	43.570559	9.573240	9.840239	10.51114

```
Columns: ['date', 'open', 'high', 'low', 'close', 'volume', 'rsi_7', 'rsi_14', 'cci_7', 'cci_14', 'sma_50', 'ema_50', 'sma_100', 'en
```

```
# Normalize column names
df.columns = [c.strip().lower() for c in df.columns]

# Find a date column and convert it
date_col = None
for c in ['date','datetime','time','timestamp']:
    if c in df.columns:
        date_col = c
        break

if date_col:
    df[date_col] = pd.to_datetime(df[date_col], errors='coerce')
    df = df.dropna(subset=[date_col]).sort_values(date_col).reset_index(drop=True)
    print("Using date column:", date_col)
else:
    print("No date-like column found. We'll use row order as time order.")

# Required OHLCV columns
required = ['open','high','low','close','volume']
missing = [c for c in required if c not in df.columns]
if missing:
    raise ValueError(f"Missing required columns: {missing}. Please provide a dataset with open/high/low/close/volume.")
else:
    print("Found OHLCV columns.")

# Quick summary
print("After cleaning shape:", df.shape)
display(df[required + [date_col] if date_col else required].head())
```

```
Using date column: date
Found OHLCV columns.
After cleaning shape: (2516, 20)
```

	open	high	low	close	volume	date
0	9.986667	10.165333	9.770000	10.006667	92826000	2014-01-02
1	10.000000	10.146000	9.906667	9.970667	70425000	2014-01-03
2	10.000000	10.026667	9.682667	9.800000	80416500	2014-01-06
3	9.841333	10.026667	9.683333	9.957333	75511500	2014-01-07
4	9.923333	10.246667	9.917333	10.085333	92448000	2014-01-08

```
if 'next_day_close' in df.columns:
    df['y'] = df['next_day_close']
    print("Using provided next_day_close column as target 'y'.")
else:
    df['y'] = df['close'].shift(-1)
    print("Created target 'y' as next row's close (close shifted -1).")

# Drop the final row if y is NaN (because we shifted)
df = df.dropna(subset=['y']).reset_index(drop=True)
print("Rows after creating y:", len(df))
```

```
Using provided next_day_close column as target 'y'.
Rows after creating y: 2516
```

```
# We'll use lagged OHLCV and any extra indicators that exist in your file.
base_feats = ['open','high','low','close','volume']
# find extras present in your file
possible_extras = ['rsi_7','rsi_14','cci_7','cci_14','sma_50','ema_50','sma_100','ema_100','macd','bollinger','atr_7','atr_14','truerang
extras = [c for c in possible_extras if c in df.columns]

print("Extras found (will include as lagged features if present):", extras)

lags = [1,2,3,5]   # you can edit this
for lag in lags:
    for c in base_feats + extras:
        df[f'{c}_lag{lag}'] = df[c].shift(lag)

# Drop rows that became NaN due to lagging
df = df.dropna().reset_index(drop=True)
print("Rows after lagging and dropping NaNs:", len(df))

# Build features list (only lag columns)
feature_cols = [c for c in df.columns if any(c.endswith(f'_lag{l}') for l in lags)]
print("Number of features used for classic ML:", len(feature_cols))
print(feature_cols[:30])
```

```
Extras found (will include as lagged features if present): ['rsi_7', 'rsi_14', 'cci_7', 'cci_14', 'sma_50', 'ema_50', 'sma_100', 'em
Rows after lagging and dropping NaNs: 2511
Number of features used for classic ML: 72
['open_lag1', 'high_lag1', 'low_lag1', 'close_lag1', 'volume_lag1', 'rsi_7_lag1', 'rsi_14_lag1', 'cci_7_lag1', 'cci_14_lag1', 'sma_5
```

```
n = len(df)
train_end = int(n * 0.70)
val_end = int(n * 0.85)

X = df[feature_cols].values
y = df['y'].values

X_train, y_train = X[:train_end], y[:train_end]
X_val, y_val     = X[train_end:val_end], y[train_end:val_end]
X_test, y_test   = X[val_end:], y[val_end:]

print("Sizes -> train:", len(X_train), "val:", len(X_val), "test:", len(X_test))
```

```
Sizes -> train: 1757 val: 377 test: 377
```

```
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_val_s   = scaler.transform(X_val)
X_test_s  = scaler.transform(X_test)
```

```
# We compare to a naive rule: predict next-day close = today's close.
close_series = df['close'].values  # note: df already had lagged rows removed, so indices align with X/y

# Today's close corresponding to each test row: starting at index val_end in df
y_pred_naive = close_series[val_end: val_end + len(y_test)]

mae_naive = mean_absolute_error(y_test, y_pred_naive)
mse_naive = mean_squared_error(y_test, y_pred_naive)
rmse_naive = math.sqrt(mse_naive)

print("Naive baseline -> MAE: {:.4f}, RMSE: {:.4f}".format(mae_naive, rmse_naive))
```

```
Naive baseline -> MAE: 5.7034, RMSE: 7.5539
```

```
# Ridge regression (linear)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train_s, y_train)
pred_val_ridge = ridge.predict(X_val_s)

# Random Forest (tree-based)
rf = RandomForestRegressor(n_estimators=300, random_state=SEED, n_jobs=-1)
rf.fit(X_train, y_train)   # trees don't require scaling
pred_val_rf = rf.predict(X_val)

# Metric helper (returns MAE, RMSE, R2)
```

```
def metrics(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = math.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    return mae, rmse, r2

print("Ridge val -> MAE, RMSE, R2:", [round(x,4) for x in metrics(y_val, pred_val_ridge)])
print("RF    val -> MAE, RMSE, R2:", [round(x,4) for x in metrics(y_val, pred_val_rf)])

# pick best by RMSE on validation
best_model_name = 'rf' if metrics(y_val, pred_val_rf)[1] < metrics(y_val, pred_val_ridge)[1] else 'ridge'
print("Best on validation is:", best_model_name)
```

```
Ridge val -> MAE, RMSE, R2: [18.9906, 23.2967, 0.8094]
RF    val -> MAE, RMSE, R2: [53.2133, 71.1206, -0.7768]
Best on validation is: ridge
```

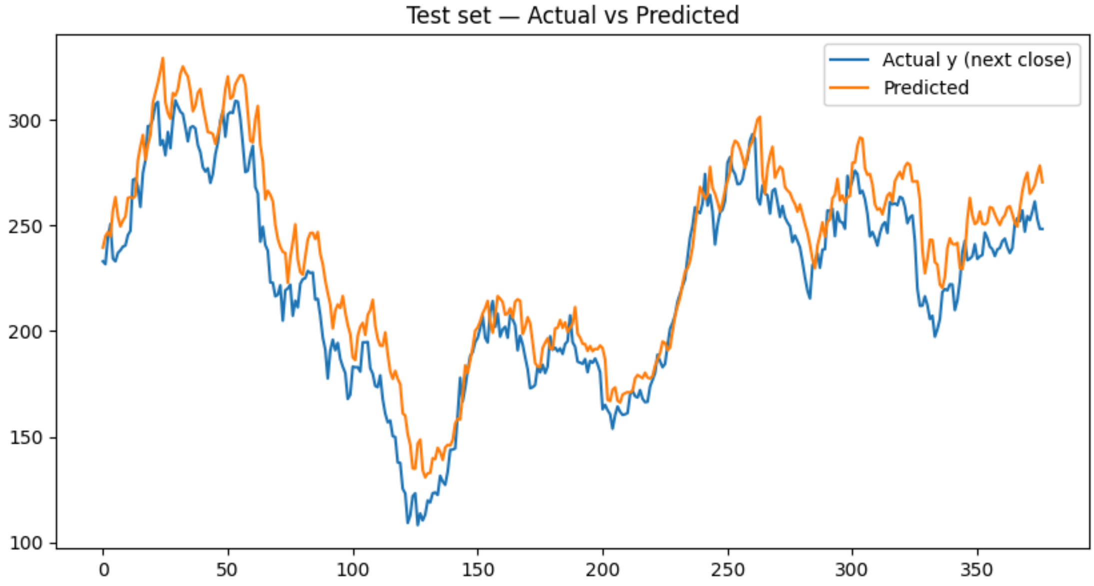
```
# choose model and compute test predictions
if best_model_name == 'rf':
    test_pred = rf.predict(X_test)
else:
    test_pred = ridge.predict(X_test_s)

mae_test, rmse_test, r2_test = metrics(y_test, test_pred)
print("Naive baseline (test) -> MAE: {:.4f}, RMSE: {:.4f}".format(mae_naive, rmse_naive))
print("Best model (test)    -> MAE: {:.4f}, RMSE: {:.4f}, R2: {:.4f}".format(mae_test, rmse_test, r2_test))

# Directional accuracy: did the model get up/down right vs previous day's close?
prev_close_for_test = close_series[val_end - 1 : val_end - 1 + len(y_test)]  # today's close corresponding to each prediction
true_dir = np.sign(y_test - prev_close_for_test)
pred_dir = np.sign(test_pred - prev_close_for_test)
directional_accuracy = (true_dir == pred_dir).mean()
print("Directional accuracy (up/down): {:.2%}".format(directional_accuracy))

# Plot actual vs predicted (test)
plt.figure()
plt.plot(y_test, label='Actual y (next close)')
plt.plot(test_pred, label='Predicted')
plt.title('Test set — Actual vs Predicted')
plt.legend()
plt.show()
```

```
Naive baseline (test) -> MAE: 5.7034, RMSE: 7.5539
Best model (test)    -> MAE: 15.3724, RMSE: 18.8205, R2: 0.8392
Directional accuracy (up/down): 48.54%
```




```
os.makedirs('models', exist_ok=True)
if best_model_name == 'rf':
    joblib.dump(rf, 'models/price_model_rf.joblib')
else:
    joblib.dump(ridge, 'models/price_model_ridge.joblib')

joblib.dump(scaler, 'models/scaler_standard.joblib')
print("Saved best model and scaler to ./models/")
```

```
Saved best model and scaler to ./models/
```

```
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks

# Choose sequence features (raw OHLCV)
seq_feats = ['open','high','low','close','volume']
seq_feats = [c for c in seq_feats if c in df.columns]  # ensure present

# Build X_all and y_all from the df after lagging (so indexing aligns)
X_all_raw = df[seq_feats].values
y_all = df['y'].values

# Scale features using MinMaxScaler fit on train portion to avoid leakage
train_rows_for_scaler = int(len(X_all_raw) * 0.70)
mm = MinMaxScaler()
mm.fit(X_all_raw[:train_rows_for_scaler])
X_all_s = mm.transform(X_all_raw)

# Create sequences
SEQ_LEN = 60   # last 60 days to predict next day
Xs, ys = [], []
for i in range(len(X_all_s) - SEQ_LEN):
    Xs.append(X_all_s[i : i + SEQ_LEN])
    ys.append(y_all[i + SEQ_LEN])
Xs = np.array(Xs)
ys = np.array(ys)
print("Total sequences:", len(Xs))

# train/val/test split for sequences (70/15/15 of sequences)
nseq = len(Xs)
t_end = int(nseq * 0.70)
v_end = int(nseq * 0.85)
X_train_nn, y_train_nn = Xs[:t_end], ys[:t_end]
X_val_nn, y_val_nn = Xs[t_end:v_end], ys[t_end:v_end]
X_test_nn, y_test_nn = Xs[v_end:], ys[v_end:]

# Build a small LSTM
tf.random.set_seed(SEED)
model = models.Sequential([
    layers.Input(shape=(SEQ_LEN, X_train_nn.shape[-1])),
    layers.LSTM(64, return_sequences=True),
    layers.Dropout(0.2),
    layers.LSTM(32),
    layers.Dropout(0.2),
    layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
es = callbacks.EarlyStopping(patience=8, restore_best_weights=True)

history = model.fit(
    X_train_nn, y_train_nn,
    validation_data=(X_val_nn, y_val_nn),
    epochs=100, batch_size=64,
    callbacks=[es], verbose=1
)

# Evaluate on sequences test set
pred_test_nn = model.predict(X_test_nn).ravel()
mae_l = mean_absolute_error(y_test_nn, pred_test_nn)
rmse_l = math.sqrt(mean_squared_error(y_test_nn, pred_test_nn))
r2_l = r2_score(y_test_nn, pred_test_nn)
print("LSTM test -> MAE: {:.4f}, RMSE: {:.4f}, R2: {:.4f}".format(mae_l, rmse_l, r2_l))

plt.figure()
plt.plot(y_test_nn, label='Actual')
plt.plot(pred_test_nn, label='LSTM Pred')
plt.legend(); plt.title('LSTM — Actual vs Predicted'); plt.show()
```

```
Total sequences: 2451
Epoch 1/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 13s 199ms/step - loss: 3126.3525 - mae: 32.2424 - val_loss: 72822.3750 - val_mae: 264.3744
Epoch 2/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - loss: 2857.4644 - mae: 27.4635 - val_loss: 71634.5078 - val_mae: 262.1181
Epoch 3/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 80ms/step - loss: 2757.4285 - mae: 25.6469 - val_loss: 71090.2812 - val_mae: 261.0778
Epoch 4/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 104ms/step - loss: 2709.5369 - mae: 24.6387 - val_loss: 70599.4844 - val_mae: 260.1361
Epoch 5/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 73ms/step - loss: 2663.3179 - mae: 23.7590 - val_loss: 70104.5859 - val_mae: 259.1831
Epoch 6/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2620.9861 - mae: 22.7725 - val_loss: 69614.2109 - val_mae: 258.2354
Epoch 7/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - loss: 2580.1692 - mae: 21.8849 - val_loss: 69159.2500 - val_mae: 257.3530
Epoch 8/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 113ms/step - loss: 2544.6587 - mae: 21.1229 - val_loss: 68725.9062 - val_mae: 256.5096
Epoch 9/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step - loss: 2508.9099 - mae: 20.2540 - val_loss: 68311.3906 - val_mae: 255.7004
Epoch 10/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2469.2856 - mae: 19.7001 - val_loss: 67911.4453 - val_mae: 254.9171
Epoch 11/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2445.3064 - mae: 19.2665 - val_loss: 67527.0234 - val_mae: 254.1620
Epoch 12/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2420.7000 - mae: 18.9608 - val_loss: 67156.2734 - val_mae: 253.4315
Epoch 13/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 108ms/step - loss: 2393.9177 - mae: 18.6938 - val_loss: 66798.0391 - val_mae: 252.7238
Epoch 14/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - loss: 2365.5381 - mae: 18.4091 - val_loss: 66452.1250 - val_mae: 252.0385
Epoch 15/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2350.5649 - mae: 18.4740 - val_loss: 66117.3047 - val_mae: 251.3734
Epoch 16/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2326.8669 - mae: 18.3390 - val_loss: 65795.9688 - val_mae: 250.7334
Epoch 17/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 83ms/step - loss: 2307.8650 - mae: 18.3770 - val_loss: 65482.3359 - val_mae: 250.1072
Epoch 18/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - loss: 2283.5342 - mae: 18.4240 - val_loss: 65179.2070 - val_mae: 249.5005
Epoch 19/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step - loss: 2279.2205 - mae: 18.6034 - val_loss: 64891.2539 - val_mae: 248.9227
Epoch 20/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2262.6226 - mae: 18.6129 - val_loss: 64611.6289 - val_mae: 248.3604
Epoch 21/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2263.5850 - mae: 18.9320 - val_loss: 64347.2891 - val_mae: 247.8277
Epoch 22/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - loss: 2240.1714 - mae: 19.0276 - val_loss: 64088.1016 - val_mae: 247.3042
Epoch 23/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 81ms/step - loss: 2231.2676 - mae: 19.2704 - val_loss: 63839.7734 - val_mae: 246.8017
Epoch 24/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2208.4075 - mae: 19.3696 - val_loss: 63597.8633 - val_mae: 246.3111
Epoch 25/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - loss: 2209.3381 - mae: 19.6643 - val_loss: 63368.5938 - val_mae: 245.8452
Epoch 26/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 115ms/step - loss: 2202.0999 - mae: 19.9233 - val_loss: 63148.8242 - val_mae: 245.3978
Epoch 27/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - loss: 2179.5776 - mae: 20.0809 - val_loss: 62934.5938 - val_mae: 244.9609
Epoch 28/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step - loss: 2168.3672 - mae: 20.2545 - val_loss: 62727.4453 - val_mae: 244.5378
Epoch 29/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2176.3267 - mae: 20.5382 - val_loss: 62525.7891 - val_mae: 244.1251
Epoch 30/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 74ms/step - loss: 2163.5801 - mae: 20.8442 - val_loss: 62338.5664 - val_mae: 243.7413
Epoch 31/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 109ms/step - loss: 2173.3140 - mae: 20.9431 - val_loss: 62156.2109 - val_mae: 243.3670
Epoch 32/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 2178.2478 - mae: 21.2820 - val_loss: 61989.9453 - val_mae: 243.0252
Epoch 33/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 76ms/step - loss: 2150.8298 - mae: 21.5132 - val_loss: 61828.2656 - val_mae: 242.6923
Epoch 34/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2143.8394 - mae: 21.6711 - val_loss: 61672.4609 - val_mae: 242.3711
Epoch 35/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - loss: 2140.9797 - mae: 21.9065 - val_loss: 61523.5508 - val_mae: 242.0637
Epoch 36/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 83ms/step - loss: 2138.6846 - mae: 22.0612 - val_loss: 61381.0547 - val_mae: 241.7691
Epoch 37/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2137.0989 - mae: 22.4156 - val_loss: 61250.9766 - val_mae: 241.5000
Epoch 38/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2118.4473 - mae: 22.3574 - val_loss: 61121.6562 - val_mae: 241.2321
Epoch 39/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - loss: 2133.4553 - mae: 22.6293 - val_loss: 61005.1523 - val_mae: 240.9905
Epoch 40/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2134.3049 - mae: 22.9218 - val_loss: 60894.3711 - val_mae: 240.7605
Epoch 41/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 111ms/step - loss: 2125.7888 - mae: 22.9496 - val_loss: 60789.9883 - val_mae: 240.5437
Epoch 42/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2127.0779 - mae: 23.0469 - val_loss: 60691.3164 - val_mae: 240.3385
Epoch 43/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2122.5417 - mae: 23.1595 - val_loss: 60597.7734 - val_mae: 240.1438
Epoch 44/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2113.7798 - mae: 23.3907 - val_loss: 60508.5156 - val_mae: 239.9579
Epoch 45/100
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2121.0601 - mae: 23.4877 - val_loss: 60423.9609 - val_mae: 239.7816
Epoch 46/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2126.1050 - mae: 23.6153 - val_loss: 60344.6797 - val_mae: 239.6163
Epoch 47/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 106ms/step - loss: 2110.5925 - mae: 23.7674 - val_loss: 60269.9180 - val_mae: 239.4602
Epoch 48/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - loss: 2105.8188 - mae: 23.5810 - val_loss: 60195.3242 - val_mae: 239.3044
Epoch 49/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2116.9258 - mae: 24.0092 - val_loss: 60135.7617 - val_mae: 239.1799
Epoch 50/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2122.7722 - mae: 24.1180 - val_loss: 60080.2617 - val_mae: 239.0639
Epoch 51/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 81ms/step - loss: 2102.9551 - mae: 24.0811 - val_loss: 60028.4297 - val_mae: 238.9554
Epoch 52/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 93ms/step - loss: 2121.1887 - mae: 24.1039 - val_loss: 59974.9570 - val_mae: 238.8435
Epoch 53/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2105.0117 - mae: 24.2737 - val_loss: 59926.8047 - val_mae: 238.7427
Epoch 54/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2106.4597 - mae: 24.3091 - val_loss: 59885.5547 - val_mae: 238.6563
Epoch 55/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2116.1565 - mae: 24.3078 - val_loss: 59845.3516 - val_mae: 238.5721
Epoch 56/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - loss: 2115.3247 - mae: 24.3943 - val_loss: 59809.0586 - val_mae: 238.4960
Epoch 57/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 116ms/step - loss: 2126.0349 - mae: 24.4777 - val_loss: 59776.7656 - val_mae: 238.4283
Epoch 58/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - loss: 2105.0586 - mae: 24.2984 - val_loss: 59741.4414 - val_mae: 238.3542
Epoch 59/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - loss: 2111.3110 - mae: 24.3914 - val_loss: 59711.2891 - val_mae: 238.2910
Epoch 60/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - loss: 2102.9421 - mae: 24.6123 - val_loss: 59683.4297 - val_mae: 238.2325
Epoch 61/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - loss: 2101.7192 - mae: 24.4836 - val_loss: 59656.6094 - val_mae: 238.1762
Epoch 62/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 2117.0054 - mae: 24.7000 - val_loss: 59634.3242 - val_mae: 238.1294
Epoch 63/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2109.0774 - mae: 24.6398 - val_loss: 59613.5039 - val_mae: 238.0857
Epoch 64/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2107.6777 - mae: 24.5713 - val_loss: 59588.9258 - val_mae: 238.0341
Epoch 65/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2121.8958 - mae: 24.8502 - val_loss: 59574.0508 - val_mae: 238.0028
Epoch 66/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - loss: 2113.0183 - mae: 24.6644 - val_loss: 59549.6914 - val_mae: 237.9516
Epoch 67/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2110.9082 - mae: 24.8014 - val_loss: 59527.3438 - val_mae: 237.9046
Epoch 68/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - loss: 2105.9421 - mae: 24.7840 - val_loss: 59516.7070 - val_mae: 237.8823
Epoch 69/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - loss: 2111.0798 - mae: 24.7521 - val_loss: 59504.1836 - val_mae: 237.8560
Epoch 70/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step - loss: 2109.3071 - mae: 24.8995 - val_loss: 59497.0000 - val_mae: 237.8409
Epoch 71/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - loss: 2111.3115 - mae: 24.7996 - val_loss: 59481.7656 - val_mae: 237.8089
Epoch 72/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 2116.6021 - mae: 24.8825 - val_loss: 59473.7539 - val_mae: 237.7920
Epoch 73/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2122.7771 - mae: 25.1077 - val_loss: 59477.5312 - val_mae: 237.7999
Epoch 74/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 75ms/step - loss: 2112.3152 - mae: 24.9003 - val_loss: 59471.3789 - val_mae: 237.7870
Epoch 75/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - loss: 2109.8032 - mae: 24.9065 - val_loss: 59470.3164 - val_mae: 237.7848
Epoch 76/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 73ms/step - loss: 2112.4058 - mae: 24.8929 - val_loss: 59463.4180 - val_mae: 237.7703
Epoch 77/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2109.9250 - mae: 24.9961 - val_loss: 59461.2383 - val_mae: 237.7657
Epoch 78/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 74ms/step - loss: 2102.2856 - mae: 24.8825 - val_loss: 59458.6953 - val_mae: 237.7603
Epoch 79/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 93ms/step - loss: 2097.2256 - mae: 24.9071 - val_loss: 59456.3633 - val_mae: 237.7554
Epoch 80/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - loss: 2110.1396 - mae: 24.9542 - val_loss: 59453.6016 - val_mae: 237.7496
Epoch 81/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - loss: 2112.1399 - mae: 24.8563 - val_loss: 59449.9883 - val_mae: 237.7420
Epoch 82/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 2103.7795 - mae: 24.9387 - val_loss: 59446.7734 - val_mae: 237.7353
Epoch 83/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 2099.2961 - mae: 24.7382 - val_loss: 59593.1641 - val_mae: 238.0433
Epoch 84/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - loss: 2095.8828 - mae: 24.4156 - val_loss: 59394.3711 - val_mae: 237.6250
Epoch 85/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 2117.4319 - mae: 24.7678 - val_loss: 60500.3359 - val_mae: 239.9413
Epoch 86/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 75ms/step - loss: 2124.8831 - mae: 23.4447 - val_loss: 60391.8750 - val_mae: 239.7220
Epoch 87/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - loss: 2086.6172 - mae: 23.0157 - val_loss: 59966.9453 - val_mae: 238.8271
Epoch 88/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 109ms/step - loss: 2085.2852 - mae: 22.4586 - val_loss: 60003.1914 - val_mae: 238.9125
Epoch 89/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 76ms/step - loss: 1970.9771 - mae: 17.0206 - val_loss: 59237.8906 - val_mae: 237.2963
Epoch 90/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 1914.1005 - mae: 15.6450 - val_loss: 58287.3359 - val_mae: 235.2847
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 1914.1003 - mae: 15.0490 - val_loss: 58287.5359 - val_mae: 235.2847
Epoch 91/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - loss: 1875.8096 - mae: 15.0428 - val_loss: 57632.3203 - val_mae: 233.8883
Epoch 92/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - loss: 1806.9805 - mae: 14.7875 - val_loss: 57065.9453 - val_mae: 232.6743
Epoch 93/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 1800.7455 - mae: 14.5696 - val_loss: 56523.5273 - val_mae: 231.5057
Epoch 94/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - loss: 1768.5190 - mae: 14.3590 - val_loss: 56002.0859 - val_mae: 230.3768
Epoch 95/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 1767.4841 - mae: 14.1795 - val_loss: 55502.3047 - val_mae: 229.2895
Epoch 96/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - loss: 1718.4257 - mae: 14.0758 - val_loss: 55012.1562 - val_mae: 228.2181
Epoch 97/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 88ms/step - loss: 1700.0615 - mae: 14.0128 - val_loss: 54540.1953 - val_mae: 227.1817
Epoch 98/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - loss: 1671.2545 - mae: 13.7754 - val_loss: 54081.0547 - val_mae: 226.1690
Epoch 99/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - loss: 1657.9788 - mae: 13.5133 - val_loss: 53635.5508 - val_mae: 225.1819
Epoch 100/100
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - loss: 1627.7742 - mae: 13.3201 - val_loss: 53199.0000 - val_mae: 224.2105
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step
LSTM test -> MAE: 177.7687, RMSE: 183.9896, R2: -14.0417
```