```python
import numpy as np
import pandas as pd
import altair as alt
import streamlit as st
import yfinance as yf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Input
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# ------------------
# Helpers
# ------------------

def make_xy(df_scaled, look_back):
    X, y = [], []
    for i in range(len(df_scaled) - look_back):
        X.append(df_scaled[i : i + look_back, 0])
        y.append(df_scaled[i + look_back, 0])
    X = np.array(X).reshape(-1, look_back, 1)
    y = np.array(y).reshape(-1, 1)
    return X, y


def build_model(look_back, units, dense, d1, d2):
    model = Sequential([
        Input(shape=(look_back, 1)),
        LSTM(units, return_sequences=True),
        Dropout(d1),
        LSTM(units, return_sequences=False),
        Dropout(d2),
        Dense(dense, activation="relu"),
        Dense(1),
    ])
    model.compile(optimizer="adam", loss="mse", metrics=["RootMeanSquaredError"])
    return model


def series_metrics(y_true, y_pred):
    y_true = np.asarray(y_true).ravel()
    y_pred = np.asarray(y_pred).ravel()

    mae = float(mean_absolute_error(y_true, y_pred))
    rmse = float(np.sqrt(mean_squared_error(y_true, y_pred)))
    r2 = float(r2_score(y_true, y_pred))

    return {
        "MAE_$": mae,
        "RMSE_$": rmse,
```

```
        "R2": r2,
    }


# ------------------
# UI
# ------------------

st.set_page_config(page_title="Stock Closing Price Forecast (LSTM)", layout="wide")
st.title("📈 Stock Closing Price Forecast – LSTM")
st.caption("Data source: Yahoo Finance (3 years history by default)")

with st.sidebar:
    st.header("1) Data")
    period = st.selectbox("Download period", ["1y", "2y", "3y", "5y", "10y"], index=2)
    ticker = st.selectbox("Ticker", ["AAPL", "AMZN", "MSFT", "GOOGL", "NVDA", "TSLA",
"META"], index=1)

    st.header("2) Window & Horizon")
    look_back = st.number_input("Look-back window (days)", min_value=10, max_value=200,
value=30, step=5)
    horizon = st.number_input("Forecast horizon (days)", min_value=1, max_value=60,
value=14, step=1)

    units = 64
    dense = 32
    d1 = 0.2
    d2 = 0.1
    epochs = 50

    test_frac = 0.2

try:
    df = yf.Ticker(ticker).history(period=period)
except Exception as e:
    st.error(f"yfinance error: {e}")
    st.stop()

if df is None or df.empty:
    st.error("No data returned. Try another period or check ticker symbol.")
    st.stop()

# Ensure Date index and keep Close
df = df.reset_index().set_index("Date").sort_index()
prices = df[["Close"]].dropna().copy()

# normalize timezone -> naive (no tz)
try:
    prices.index = prices.index.tz_localize(None)
except TypeError:
    pass

min_d = prices.index.min().date()
max_d = prices.index.max().date()
```

```
105 c1, c2 = st.columns(2)
106
107 with c1:
108     start_date = st.date_input("Start date", value=min_d, min_value=min_d,
    max_value=max_d)
109 with c2:
110     end_date = st.date_input("End date", value=max_d, min_value=min_d, max_value=max_d)
111
112 mask = (prices.index >= pd.Timestamp(start_date)) & (prices.index <=
    pd.Timestamp(end_date))
113 series = prices.loc[mask].copy()
114
115 # 50 is way to just say buffer to say
116 if len(series) < look_back + 50:
117     st.warning("Selected window looks short for LSTM. Consider more history.")
118
119 st.subheader(f"{ticker} Stock Data preview")
120 st.dataframe(series.tail(10))
121
122 # Chart raw close
123 raw_chart = (
124     alt.Chart(series.reset_index())
125     .mark_line()
126     .encode(
127         x=alt.X("Date:T", axis=alt.Axis(format="%Y-%m-%d", title="Date")),
128         y=alt.Y("Close:Q", title="Close ($)"),
129         tooltip=[
130             alt.Tooltip("Date:T", format="%Y-%m-%d"),
131             alt.Tooltip("Close:Q", format="$.2f"),
132         ],
133     )
134     .properties(height=250)
135 )
136 st.altair_chart(raw_chart, use_container_width=True)
137
138 values = series.values.astype(float)
139 N = len(values)
140
141 if N <= look_back + 1:
142     st.error("Not enough data for chosen look-back.")
143     st.stop()
144
145 # Time-based split
146 train_size = int(N * (1 - float(test_frac)))
147 train_vals = values[:train_size] # 0... train_size
148 test_vals = values[train_size:]  # train_size...end
149
150 # Scaling (fit on train only)
151 scaler = MinMaxScaler()
152 train_scaled = scaler.fit_transform(train_vals)
153 test_scaled = scaler.transform(test_vals)
154
155 # Windows
156 X_train, y_train = make_xy(train_scaled, look_back)
```

```
157 X_test, y_test = make_xy(test_scaled, look_back)
158
159 if len(X_train) == 0 or len(X_test) == 0:
160     st.error("Look-back too large for the selected split; reduce look-back or expand date
    range.")
161     st.stop()
162
163 # Train & Forecast
164 if st.button("🚀 Train & Forecast",  use_container_width=True):
165     with st.spinner("Training model…"):
166         model = build_model(look_back, units, dense, d1, d2)
167         callbacks = [
168             EarlyStopping(patience = 10, restore_best_weights = True),
169             ReduceLROnPlateau(),
170         ]
171         model.fit(
172             X_train,
173             y_train,
174             batch_size=32,
175             validation_data=(X_test, y_test),
176             epochs=int(epochs),
177             callbacks=callbacks
178         )
179
180     # Predict test
181     y_pred_scaled = model.predict(X_test) ## (n_samples,1)
182     y_pred = scaler.inverse_transform(y_pred_scaled).ravel() ## flaten to 1D array
183     y_true = scaler.inverse_transform(y_test).ravel()
184
185     # Index alignment
186     test_index = series.index[train_size + look_back:]
187     test_df = pd.DataFrame({"Actual_$": y_true, "Pred_$": y_pred}, index=test_index)
188
189     # Metrics
190     met = series_metrics(y_true, y_pred)
191
192     st.subheader("Evaluation Summary")
193     m1, m2, m3 = st.columns(3)
194     m1.metric("MAE ($)", f"{met['MAE_$']:.2f}")
195     m2.metric("RMSE ($)", f"{met['RMSE_$']:.2f}")
196     m3.metric("R²", f"{met['R2']:.3f}")
197
198     # Plot test
199     st.markdown("### Test: Actual vs Predicted")
200
201     test_chart = (
202       alt.Chart(test_df.reset_index())
203        #transform_fold is the key: it turns two columns into a single “series” field so
    Altair can draw two colored lines from one tidy table.
204       .transform_fold(["Actual_$", "Pred_$"], as_=["variable", "value"])
205       .mark_line()
206       .encode(
207           x=alt.X("Date:T", axis=alt.Axis(format="%Y-%m-%d", title="Date")),
208           y=alt.Y("value:Q", title="Close ($)"),
```

```
209             color=alt.Color("variable:N", title="Series"),
210             tooltip=[
211                 alt.Tooltip("Date:T", format="%Y-%m-%d"),
212                 alt.Tooltip("variable:N", title="Series"),
213                 alt.Tooltip("value:Q", title="Close", format="$.2f"),
214             ],
215         )
216         .properties(height=300)
217     )
218 
219     st.altair_chart(test_chart, use_container_width=True)
220 
221     # Multi-step forecast
222     with st.spinner("Forecasting…"):
223         full_scaled = scaler.transform(values)
224         window = full_scaled[-look_back:].reshape(1, look_back, 1) # -look_back: go back
    30 or 60 days until the end. Reshape it into 3 dimen.
225         fut_scaled = []
226         for _ in range(int(horizon)): # Loop through each forcast horizon, example,
    14days.
227             # 1. Initially, it will from last data go back 30 or 60 days.
228             # 3. Recursive function, the window will be predict again
229             nxt = model.predict(window)
230             fut_scaled.append(nxt.item())
231             # 2. window will remove the last element and add the next prediction in the
    window, and do the predict again.
232             window = np.concatenate(
233                 [window[:, 1:, :],   # will remove the oldest value, 1: slice the 1st
    element, from 1..end
234                  nxt.reshape(1, 1, 1)],  # reshape and append the new predicted value
235                 axis=1
236             )
237 
238         fut_scaled = np.array(fut_scaled).reshape(-1, 1) # H×1 scaled predictions
239         fut = scaler.inverse_transform(fut_scaled).ravel() # back to dollars
240 
241     last_date = series.index[-1] # last timestamp in your selected data
242     forecast_idx = pd.date_range(
243         last_date + pd.Timedelta(days=1), # start the day *after* last_date
244         periods=int(horizon), # how many future timestamps
245         freq="B" # 'Business day' frequency (Mon–Fri)
246     )
247     forecast_df = pd.DataFrame({"Forecast_$": fut}, index=forecast_idx)
248 
249     st.markdown("### Forecast")
250     c1, c2 = st.columns([2, 1])
251     with c1:
252         recent = series.iloc[-max(100, look_back + 1):].copy()
253         recent["Type"] = "Recent Close"
254         fc = forecast_df.copy()
255         fc["Type"] = "Forecast"
256         plot_df = pd.concat([
257             recent.rename(columns={"Close": "Close_$"}),
258             fc.rename(columns={"Forecast_$": "Close_$"})
```

```
        ])

        chart = (
            alt.Chart(
                plot_df.reset_index(names="Date")
            )
            .mark_line(point=True)
            .encode(
                x=alt.X("Date:T", axis=alt.Axis(format="%Y-%m-%d", title="Date")),
                y=alt.Y("Close_$:Q", title="Close ($)"),
                color=alt.Color("Type:N"),
                tooltip=[
                    alt.Tooltip("Date:T", format="%Y-%m-%d"),
                    alt.Tooltip("Close_$:Q", title="Close", format="$.2f"),  # <- use Close_$
                    alt.Tooltip("Type:N", title="Series"),
                ],
            )
            .properties(height=300)
        )

        st.altair_chart(chart, use_container_width=True)

    with c2:
        st.dataframe(forecast_df.round(2))
        csv = forecast_df.reset_index().rename(columns={"index": "Date"}).to_csv(index=False).encode("utf-8")
        st.download_button("Download forecast CSV", csv, file_name=f"{ticker}_forecast.csv", mime="text/csv")

    st.info(f"Last known date in range: {last_date.date()} — Forecast starts on {(last_date + pd.Timedelta(days=1)).date()} for {int(horizon)} day(s).")

else:
    st.info("Adjust parameters on the left, then click **Train & Forecast**.")
```