Stock Market Prediction using Time Series Analysis ARIMA and LSTM

1. Problem definition & data sources

Objective: Forecast short-term stock prices (e.g., next-day close) or returns using historical time series. Important: this guide is educational and not financial advice.

Common data sources:

- Yahoo Finance (yfinance Python library) for historical OHLCV and minute-level data.
- Alpha Vantage (API keys, free tier limits).
- Interactive Brokers, Polygon, IEX, Quandl for production-grade feeds.

Data frequency: minute, hourly, daily. Choose based on use-case (day trading vs swing trading vs long-term).

Key variables: Close price, volume, adjusted close, technical indicators (MA, RSI), exogenous features (news sentiment, macro variables).

2. Exploratory Data Analysis & Preprocessing

Load data (example with yfinance):

```
python

import yfinance as yf

symbol = 'AAPL'

df = yf.download(symbol, period='2y', interval='1d')
```

Common EDA steps: plot price and log(price), compute returns (pct_change or log returns), inspect missing values, check seasonality and weekly patterns.

Stationarity: many models need stationary series. Stationarity tests: ADF (Augmented Dickey-Fuller), KPSS. If non-stationary, apply differencing or detrending.

3. Stationarity, transforms & feature engineering

Transforms often used: log(price), log returns: r_t = ln(p_t) - ln(p_{t-1}), percentage returns. Differencing: first difference removes trend.

ADF test (statsmodels):

```
python

from statsmodels.tsa.stattools import adfuller

result = adfuller(series.dropna())

print('ADF stat:', result[0], 'p-value:', result[1])
```

Feature engineering: moving averages, rolling std (volatility), lag features, RSI, MACD. For ML, include scaled technical indicators and recent lags.

4. ARIMA: Theory and modelling steps

ARIMA(p,d,q) models autoregression (p), differencing (d), and moving average (q). If seasonality present use SARIMA/SARIMAX.

Order selection: inspect ACF and PACF plots, use information criteria (AIC, BIC) or auto_arima from pmdarima.

Fitting with statsmodels:

```
python

from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(train_series, order=(p,d,q)).fit()

print(model.summary())

pred = model.get_forecast(steps=steps)

fc = pred.summary_frame()
```

Diagnostics: residuals should behave like white noise (plot ACF of residuals, Ljung-Box test). If residuals show structure, revisit orders or add exogenous variables.

5. ARIMA: Walk-forward validation & real-time forecasting

Walk-forward (rolling) validation is essential for time series. Split by time and iteratively train on history and forecast the next window.

Example skeleton for rolling forecast:

```
python

history = list(train)

predictions = []

for t in range(len(test)):

    model = ARIMA(history, order=(p,d,q)).fit()

    yhat = model.forecast()[0]

    predictions.append(yhat)

    history.append(test[t])
```

Real-time usage: poll live data (e.g., via API), add new point to history, call model.forecast(steps=1) or re-fit periodically. Balance between speed and model freshness.

6. LSTM: Why and preprocessing

Why LSTM: Captures nonlinear patterns and longer-term dependencies. Works well when there are complex dynamics not captured by linear models.

Key preprocessing: scale features (MinMaxScaler or StandardScaler), convert series to supervised sequences (sliding windows), create train/validation/test splits that respect time.

Sequence creation example:

```
python

import numpy as np

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaled = scaler.fit_transform(df[['Close']].values)

def create_sequences(data, lookback=20):

    X, y = [], []

    for i in range(lookback, len(data)):

        X.append(data[i-lookback:i, 0])

        y.append(data[i, 0])

    return np.array(X), np.array(y)

X, y = create_sequences(scaled, lookback=60)

X = X.reshape((X.shape[0], X.shape[1], 1))
```

7. LSTM model architecture & training

Typical architecture: one or two LSTM layers, optional Dropout, Dense output. For regression use 'mse' loss and 'adam' optimizer:

```
python

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential()

model.add(LSTM(64, input_shape=(lookback,1), return_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val,y_val))
```

Tips: monitor validation loss, use ReduceLROnPlateau and EarlyStopping callbacks, experiment with lookback window (20-200), normalize using training data only.

8. Real-time inference & deployment

Real-time pattern: an inference service polls or receives new ticks, pre-processes to match model inputs, predicts, and returns forecast.

Example Flask endpoint skeleton (LSTM):

```
python

from flask import Flask, jsonify

import yfinance as yf

import joblib

app = Flask(__name__)

model = keras.models.load_model('lstm.h5')

scaler = joblib.load('scaler.save')

@app.route('/predict')

def predict():

    df = yf.download('AAPL', period='70d')

    scaled = scaler.transform(df[['Close']].values)

    last_seq = scaled[-60:].reshape(1,60,1)

    yhat = model.predict(last_seq)

    price_pred = scaler.inverse_transform(yhat.reshape(-1,1))[0,0]

    return jsonify({'prediction': float(price_pred)})
```

Model updates: retrain offline daily/weekly or perform incremental updates. Use canary deployments and backtesting before replacing production models.

9. Evaluation, ensembles, limitations & appendix

Metrics: RMSE, MAE, MAPE for point forecasts. For directional accuracy use percent correct sign predictions.

Backtesting: simulate the production pipeline over historical data, include transaction costs and slippage if used in trading.

Ensemble idea: combine ARIMA and LSTM forecasts (e.g., simple average or learned stacker) to exploit both linear and nonlinear strengths.

Limitations & risks: Lookahead leakage, overfitting, regime shifts, limited predictive power of prices, data quality issues. Always treat these models as probabilistic tools, not guarantees.

Appendix: Key libraries referenced: pandas, numpy, matplotlib, yfinance, statsmodels, pmdarima (optional), scikit-learn, tensorflow/keras, joblib, flask/fastapi.

Further reading: 'Time Series Analysis' (Box & Jenkins), 'Hands-On Time Series Analysis with Python' and Keras/TensorFlow guides. pmdarima docs for auto_arima, statsmodels docs for ARIMA and diagnostics.