Moving Average Trading Strategies

1. Simple Moving Average (SMA) Crossover

Overview: Buy when the short-term SMA crosses above the long-term SMA (golden cross); sell when it crosses below (death cross).

Parameters:

- `short_window`: e.g., 50
- `long_window`: e.g., 200

Entry Rule:

```
if SMA(short_window)[t] > SMA(long_window)[t] \
   and SMA(short_window)[t-1] <= SMA(long_window)[t-1]:
    signal = "BUY"
```

Exit Rule:

```
if SMA(short_window)[t] < SMA(long_window)[t] \
   and SMA(short_window)[t-1] >= SMA(long_window)[t-1]:
    signal = "SELL"
```

Pseudocode:

```
import pandas as pd
# params
short_w, long_w = 50, 200
# load data: df with 'close'
df['SMA_short'] = df['close'].rolling(short_w).mean()
df['SMA_long'] = df['close'].rolling(long_w).mean()

def generate_signals(df):
    signals = []
    for i in range(1, len(df)):
        if df.SMA_short[i] > df.SMA_long[i] \
           and df.SMA_short[i-1] <= df.SMA_long[i-1]:
            signals.append('BUY')
        elif df.SMA_short[i] < df.SMA_long[i] \
             and df.SMA_short[i-1] >= df.SMA_long[i-1]:
            signals.append('SELL')
        else:
            signals.append('HOLD')
    return signals
```

---

2. Exponential Moving Average (EMA) Crossover

Overview: Uses EMA for greater sensitivity compared to SMA.

Parameters:

- `short_window`: e.g., 20
- `long_window`: e.g., 50

Rules: Apply the same crossover logic as in the SMA strategy, replacing SMA calculations with EMA.

Pseudocode:

```
# params
short_w, long_w = 20, 50
# compute EMAs
df['EMA_short'] = df['close'].ewm(span=short_w, adjust=False).mean()
df['EMA_long'] = df['close'].ewm(span=long_w, adjust=False).mean()

# apply crossover logic
# df['signal'] = generate_cross_signals(df, 'EMA_short', 'EMA_long')
```

---

3. Triple Moving Average Strategy

Overview: Uses three moving averages—fast, mid, and slow—to confirm strong trends.

Parameters:

- `fast_w`: 10
- `mid_w`: 50
- `slow_w`: 200

Entry Rule:

```
if MA(fast)[t] > MA(mid)[t] > MA(slow)[t]:
    signal = "BUY"
```

Exit Rule:

```
if MA(fast)[t] < MA(mid)[t] < MA(slow)[t]:
    signal = "SELL"
```

Pseudocode:

```
# params
fast_w, mid_w, slow_w = 10, 50, 200
# compute MAs
df['MA_fast'] = df['close'].rolling(fast_w).mean()
df['MA_mid'] = df['close'].rolling(mid_w).mean()
df['MA_slow'] = df['close'].rolling(slow_w).mean()

def triple_ma_signals(df):
    signals = []
    for i in range(len(df)):
        if df.MA_fast[i] > df.MA_mid[i] > df.MA_slow[i]:
            signals.append('BUY')
        elif df.MA_fast[i] < df.MA_mid[i] < df.MA_slow[i]:
            signals.append('SELL')
        else:
            signals.append('HOLD')
    return signals
```

---

4. Moving Average Envelope

Overview: Surrounds an SMA with upper and lower bands based on a percentage offset to signal overbought or oversold conditions.

Parameters:

- `window`: e.g., 20
- `percent`: e.g., 0.02

Envelope Calculation:

```
df['SMA'] = df['close'].rolling(window).mean()
df['Upper'] = df['SMA'] * (1 + percent)
df['Lower'] = df['SMA'] * (1 - percent)
```

Rules:

```
if price[t] > Upper[t]: signal = "SELL"
elif price[t] < Lower[t]: signal = "BUY"
else: signal = "HOLD"
```

Pseudocode:

```
# params
window, pct = 20, 0.02
# compute SMA and bands
```

```
df['SMA'] = df.close.rolling(window).mean()
df['Upper'], df['Lower'] = df.SMA*(1+pct), df.SMA*(1-pct)

def envelope_signals(df):
    signals = []
    for i in range(len(df)):
        if df.close[i] > df.Upper[i]: signals.append('SELL')
        elif df.close[i] < df.Lower[i]: signals.append('BUY')
        else: signals.append('HOLD')
    return signals
```

---

5. Dynamic Moving Average (DMA) Strategy

Overview: Adapts the moving average window based on market volatility measured by the ATR.

Parameters:

- `atr_window`: 14
- `base_window`: 20
- `scaling`: 0.5

Dynamic Window Calculation:

```
import talib as ta
df['ATR'] = ta.ATR(df.high, df.low, df.close, timeperiod=atr_window)
df['dynamic_w'] = (base_window * (1 + scaling * (df.ATR /
df.close))).astype(int)
```

Rules: Compute an ATR-scaled SMA each bar and apply crossover logic to its previous value.

Pseudocode:

```
signals = []
for i in range(max(df.dynamic_w), len(df)):
    win = df.dynamic_w[i]
    sma_now = df.close[i-win+1:i+1].mean()
    prev_win = df.dynamic_w[i-1]
    sma_prev = df.close[i-1-prev_win+1:i].mean()
    if sma_now > sma_prev: signals.append('BUY')
    elif sma_now < sma_prev: signals.append('SELL')
    else: signals.append('HOLD')
```

Integration Tips:

- Backtest each strategy over historical data before live deployment.

• Perform parameter sweeps or use optimization methods to identify optimal settings.
• Incorporate stop-loss and take-profit rules for risk management.
• Implement logging and alerting for executed signals.