Set up

In [ ]:
```
import os

import pandas as pd
from alpaca.data import (
    StockHistoricalDataClient,
)

from alpaca.data.requests import (
    StockBarsRequest,
    TimeFrame,
)
from alpaca.trading import (
    MarketOrderRequest,
    OrderSide,
    TimeInForce,
    TradingClient,
)
from alpaca.trading.models import Position, TradeAccount
```

In [ ]:
```
import dotenv

dotenv.load_dotenv()

API_KEY = os.getenv("ALPACA_KEY")
SECRET_KEY = os.getenv("ALPACA_SECRET")
BASE_URL = "https://paper-api.alpaca.markets/v2"  # paper trading

stock_history_client = StockHistoricalDataClient(API_KEY, SECRET_KEY)
```

In [ ]:
```
trading_client = TradingClient(API_KEY, SECRET_KEY, paper=True)
```

Fetching historical data

In [ ]:
```
# get history datt from alpaca api, daily of last 126 days (half year)

from tracemalloc import start


def get_history_data(symbol="AAPL", days=182) -> pd.DataFrame:
    # end date in US Eastern Timezone
    end_date = pd.Timestamp.now(tz="US/Eastern")
    end_date = end_date - pd.Timedelta(days=1)  # yesterday
    start_date = end_date - pd.Timedelta(days=days)

    stock_request = StockBarsRequest(
        symbol_or_symbols=symbol,
        timeframe=TimeFrame.Day,
        start=start_date,
```

```
        end=end_date,
    )

    bars = stock_history_client.get_stock_bars(stock_request)

    return bars.df
```

In [56]:
```
data = get_history_data()
data
```

Out[56]:

symbol	timestamp	open	high	low	close	volume	trade_co
AAPL	2024-08-12 04:00:00+00:00	216.070	219.5099	215.6000	217.53	38028092.0	60253
	2024-08-13 04:00:00+00:00	219.010	221.8900	219.0100	221.27	44155331.0	55326
	2024-08-14 04:00:00+00:00	220.570	223.0300	219.7000	221.72	41960574.0	5685
	2024-08-15 04:00:00+00:00	224.600	225.3500	222.7600	224.72	46414013.0	59057
	2024-08-16 04:00:00+00:00	223.920	226.8271	223.6501	226.05	44340240.0	56263
	...	...	...	...	...	...	
	2025-02-03 05:00:00+00:00	229.990	231.8300	225.7000	228.01	72998404.0	8508
	2025-02-04 05:00:00+00:00	227.250	233.1300	226.6500	232.80	44902694.0	49470
	2025-02-05 05:00:00+00:00	228.530	232.6700	228.2700	232.47	39664989.0	4456
	2025-02-06 05:00:00+00:00	231.285	233.8000	230.4250	233.22	29925349.0	3754
	2025-02-07 05:00:00+00:00	232.600	234.0000	227.2600	227.63	39487057.0	4842

124 rows × 7 columns

Calculate EMA and RSI

In [ ]:
```
# calculate EMA


def calculate_ema(
    data: pd.DataFrame, period: int = 9, column: str = "vwap"
) -> pd.Series:
    # shift the day to yesterday and calculate the EMA
```

```
    ema = data[column].ewm(span=period, adjust=False).mean()
    return ema
```

In [58]:
```
ema = calculate_ema(data)
ema
```

Out[58]:
```
symbol  timestamp
AAPL    2024-08-12 04:00:00+00:00    217.546317
        2024-08-13 04:00:00+00:00    218.235785
        2024-08-14 04:00:00+00:00    218.888490
        2024-08-15 04:00:00+00:00    220.021524
        2024-08-16 04:00:00+00:00    221.103091
                                        ...
        2025-02-03 05:00:00+00:00    233.097586
        2025-02-04 05:00:00+00:00    232.719967
        2025-02-05 05:00:00+00:00    232.367581
        2025-02-06 05:00:00+00:00    232.370024
        2025-02-07 05:00:00+00:00    231.732667
Name: vwap, Length: 124, dtype: float64
```

In [ ]:
```
# calculate RSI


def calculate_rsi(
    data: pd.DataFrame, period: int = 14, column: str = "vwap"
) -> pd.Series:
    delta = data[column].diff()
    gain = (delta.where(delta > 0, 0)).ewm(span=period).mean()
    loss = (-delta.where(delta < 0, 0)).ewm(span=period).mean()
    rs = gain / loss
    return 100 - (100 / (1 + rs))
```

In [33]:
```
rsi = calculate_rsi(data)
rsi
```

Out[33]:
```
symbol  timestamp
AAPL    2024-08-12 04:00:00+00:00           NaN
        2024-08-13 04:00:00+00:00    100.000000
        2024-08-14 04:00:00+00:00    100.000000
        2024-08-15 04:00:00+00:00    100.000000
        2024-08-16 04:00:00+00:00    100.000000
                                        ...
        2025-02-03 05:00:00+00:00     38.715583
        2025-02-04 05:00:00+00:00     45.493158
        2025-02-05 05:00:00+00:00     45.019707
        2025-02-06 05:00:00+00:00     48.515308
        2025-02-07 05:00:00+00:00     41.646280
Name: vwap, Length: 124, dtype: float64
```

In [34]:
```
# pick last rsi value
rsi.iloc[-1]
```

Out[34]:
```
41.64627991927544
```

Build Trading Strategy

In [ ]:
```python
def get_signals(symbol="AAPL"):
    data = get_history_data(symbol)
    rsi = calculate_rsi(data).shift(1)
    short_ema = calculate_ema(data, 9).shift(1)
    long_ema = calculate_ema(data, 21).shift(1)

    # a series of same length as data, but add 1 to buy/long, -1 to sell/sho
    signals = pd.Series(0, index=data.index)

    for i in range(1, len(data)):
        if rsi.iloc[i] < 30 and long_ema.iloc[i] > short_ema.iloc[i]:
            signals.iloc[i] = 1
        elif rsi.iloc[i] > 70 and long_ema.iloc[i] < short_ema.iloc[i]:
            signals.iloc[i] = -1

    return signals  # this signals could be used for backtesting, but backte
```

In [60]:
```python
df = get_signals()
```

In [61]:
```python
df.head(100)
```

Out[61]:
```
symbol  timestamp
AAPL    2024-08-12 04:00:00+00:00    0
        2024-08-13 04:00:00+00:00    0
        2024-08-14 04:00:00+00:00   -1
        2024-08-15 04:00:00+00:00   -1
        2024-08-16 04:00:00+00:00   -1
                                    ..
        2024-12-26 05:00:00+00:00   -1
        2024-12-27 05:00:00+00:00   -1
        2024-12-30 05:00:00+00:00    0
        2024-12-31 05:00:00+00:00    0
        2025-01-02 05:00:00+00:00    0
Length: 100, dtype: int64
```

In [ ]:
```python
def trade(symbol="AAPL", quantity=1000):
    signals = get_signals(symbol)
    # get latest signal
    latest_signal = signals.iloc[-1]
    if latest_signal == 1 or True:
        # buy/long
        order_request = MarketOrderRequest(
            symbol=symbol,
            qty=quantity,
            side=OrderSide.BUY,
            time_in_force=TimeInForce.DAY,
        )
        trading_client.submit_order(order_request)
    elif latest_signal == -1:
        # sell/short
        order_request = MarketOrderRequest(
            symbol=symbol,
```

```
            qty=quantity,
            side=OrderSide.SELL,
            time_in_force=TimeInForce.DAY,
        )
        trading_client.submit_order(order_request)
    else:
        # keep the position
        pass

    return latest_signal
```

In [ ]:
```
# test trade function
trade('AAPL', 1)
```

Out[ ]:
```
0
```

Futher Work

Now we have a simple trading strategy, that is based on long term EMA and short term EMA and RSI.

Next, we will focus on infrastructure part to build and deploy this strategy on AWS Lambda.