'''

Project Title: Stock Market Trading Automation (Paper Trading)

Name: Adrian Chavez Loya

Date: November, 2024

Project Summary:

This project automates stock trading simulations using Python and the Alpaca API in a simulated \*\*paper trading environment\*\*.

It implements multiple trading strategies, evaluates profitability, and automates daily execution using a cron job.

1. Objective:

- Fetch historical stock data for 10 selected stocks.

- Apply three trading strategies: SMA Crossover, Mean Reversion, and Volatility Breakout.

- Simulate trades, including short selling, to calculate profit/loss.

- Identify the most profitable stock and strategy and save results to `results.json`.

- Automate execution at 9 AM ET (14:00 UTC) on weekdays via cron.

2. Key Features:

- \*\*Data Fetching\*\*: Retrieves and updates stock data using the Alpaca API, saving it as CSV files.

- \*\*Trading Strategies\*\*:

1. SMA Crossover: Uses short- and long-term averages to generate buy/sell signals.

2. Mean Reversion: Identifies overbought/oversold conditions.

3. Volatility Breakout: Trades based on significant price movements.

- \*\*Simulation\*\*: Simulated trades with tracked cash, positions, and profit in a risk-free paper trading environment.

- \*\*Automation\*\*: A cron job ensures daily execution and fresh analysis.

3. Workflow:

- Stock data is updated daily at 9 AM ET.

- Trading strategies are applied, and results are saved in `results.json`:

{

"Best Strategy": "SMA Crossover",

"Stock": "NFLX",

"Profit": 130.02

}

'''

import pandas as pd

from alpaca\_trade\_api.rest import REST, TimeFrame

import json

# Credentials and Base URL

API\_KEY = "\*\*\*\*\*\*"

API\_SECRET = "\*\*\*\*\*\*\*"

BASE\_URL = "https://paper-api.alpaca.markets"

# Initialize Alpaca API client

alpaca = REST(API\_KEY, API\_SECRET, BASE\_URL)

# Relative path for data folder

data\_folder = "Final Project- Stock Market Trading/data"

# Fetch stock data

def fetch\_stock\_data(symbol, start\_date="2023-01-01", end\_date="2023-12-31"):

try:

bars = alpaca.get\_bars(symbol, TimeFrame.Day, start=start\_date, end=end\_date, feed="iex").df

bars = bars[["open", "high", "low", "close", "volume"]]

file\_path = f"{data\_folder}/{symbol}.csv"

bars.to\_csv(file\_path, mode="w", index=True) # Files will overwrite old ones when updating stock data

print(f"Data for {symbol} saved successfully in {file\_path}.")

except Exception as e:

print(f"Error fetching data for {symbol}: {e}")

# Fetch data for 10 stocks

stocks = ["AAPL", "GOOGL", "AMZN", "MSFT", "TSLA", "META", "NFLX", "NVDA", "INTC", "IBM"]

for stock in stocks:

print(f"Fetching data for {stock}...")

fetch\_stock\_data(stock)

def sma\_crossover(data, short\_window=10, long\_window=50):

data["SMA\_Short"] = data["close"].rolling(window=short\_window).mean()

data["SMA\_Long"] = data["close"].rolling(window=long\_window).mean()

data["Signal"] = 0

data.loc[data["SMA\_Short"] > data["SMA\_Long"], "Signal"] = 1 # Buy signal

data.loc[data["SMA\_Short"] <= data["SMA\_Long"], "Signal"] = -1 # Sell signal

return data

def mean\_reversion(data, threshold=2):

data["Mean"] = data["close"].rolling(window=20).mean()

data["StdDev"] = data["close"].rolling(window=20).std()

data["Upper"] = data["Mean"] + (threshold \* data["StdDev"])

data["Lower"] = data["Mean"] - (threshold \* data["StdDev"])

data["Signal"] = 0

data.loc[data["close"] > data["Upper"], "Signal"] = -1 # Sell signal

data.loc[data["close"] < data["Lower"], "Signal"] = 1 # Buy signal

return data

def volatility\_breakout(data, breakout\_multiplier=1.5):

data["Range"] = data["high"] - data["low"]

data["Breakout"] = data["close"].shift(1) + (data["Range"].shift(1) \* breakout\_multiplier)

data["Signal"] = 0

data.loc[data["close"] > data["Breakout"], "Signal"] = 1 # Buy signal

data.loc[data["close"] <= data["Breakout"], "Signal"] = -1 # Sell signal

return data

def simulate\_trades(data):

cash = 100000 # Initial cash balance (started with 10,000)

positions = 0 # Number of shares owned

profit = 0 # Total profit

for i in range(1, len(data)):

signal = data.iloc[i]["Signal"]

price = data.iloc[i]["close"]

if signal == 1: # Buy

positions += 1

cash -= price

print(f"Bought at {price}")

elif signal == -1 and positions > 0: # Sell

positions -= 1

cash += price

profit += price - data.iloc[i-1]["close"]

print(f"Sold at {price}, Profit: {profit}")

print(f"Final Cash: {cash}, Total Profit: {profit}")

return profit

def save\_results(strategy\_name, stock, profit):

results = {

"Best Strategy": strategy\_name,

"Stock": stock,

"Profit": profit

}

with open("Final Project- Stock Market Trading/results.json", "w") as f:

json.dump(results, f)

print("Results saved to results.json")

def display\_last\_signal(data, stock):

last\_signal = data.iloc[-1]["Signal"]

if last\_signal == 1:

print(f"Last signal for {stock}: BUY")

elif last\_signal == -1:

print(f"Last signal for {stock}: SELL")

else:

print(f"Last signal for {stock}: HOLD")

# Process all stocks

strategy\_functions = {

"SMA Crossover": sma\_crossover,

"Mean Reversion": mean\_reversion,

"Volatility Breakout": volatility\_breakout

}

best\_profit = float("-inf")

best\_strategy = None

best\_stock = None

for stock in stocks:

print(f"Processing {stock}...")

# Load stock data

file\_path = f"{data\_folder}/{stock}.csv"

data = pd.read\_csv(file\_path, index\_col=0, parse\_dates=True)

# Apply strategies and simulate trades

for strategy\_name, strategy\_function in strategy\_functions.items():

print(f"Applying {strategy\_name} to {stock}...")

strategy\_data = strategy\_function(data)

profit = simulate\_trades(strategy\_data)

# Track the most profitable strategy and stock

if profit > best\_profit:

best\_profit = profit

best\_strategy = strategy\_name

best\_stock = stock

# Display last signal

display\_last\_signal(strategy\_data, stock)

# Save the best results

save\_results(best\_strategy, best\_stock, best\_profit)