CONESTOGA
Connect Life and Learning

Reinforcement Learning and the Stock Market

Exploring the use of reinforcement learning in stock trading.

TESSA NEJLA AYVAZOGLU

Reinforcement Learning Programming CSCN8020

DATE: 05/08/2024

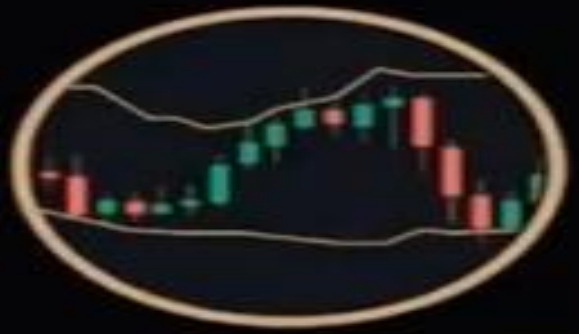
Introduction to the stock market

The stock market is where shares of companies and other financial instruments are bought and sold.

Seven widely used indicators

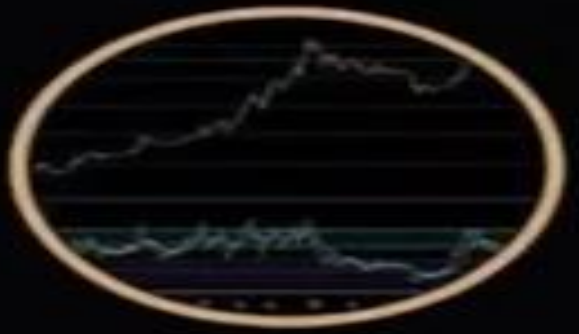
- On-balance volume (OBV)
- Accumulation/distribution (A/D) line
- Average directional index
- Aroon oscillator
- Moving average convergence divergence (MACD)
- Relative strength index (RSI)
- Stochastic oscillator

SIX INDICATORS
ALL TRADERS SHOULD KNOW



BOLLINGER BANDS

Plots **price and volatility** over time with upper, lower & middle band

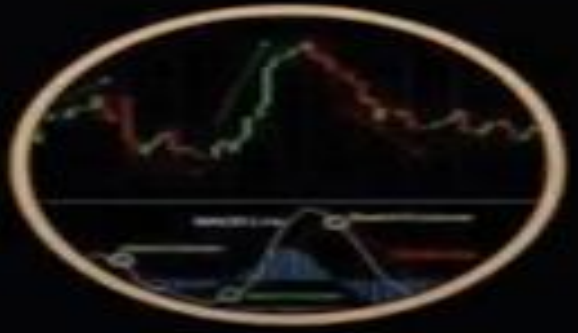


EXPONENTIAL MA

Moving average with more weight given to recent data

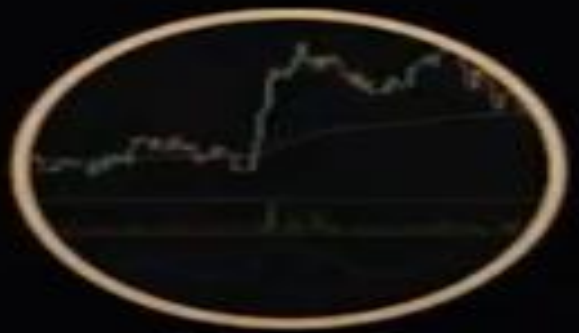
SIMPLE MA

Arithmetic average of price over the given data period



MACD

Moving average convergence/divergence momentum oscillator



RSI

Relative strength index with overbought and oversold conditions

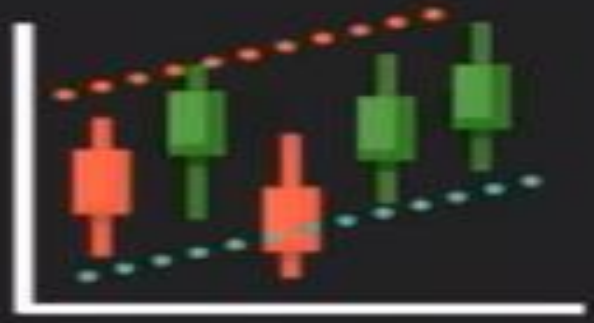
VWAP

Volume weighted average price over data period

TOP SIX INDICATORS EVERY TRADER SHOULD KNOW

EXPONENTIAL MOVING AVERAGE

SIMPLE MOVING AVERAGE



FIBONNACI RETRACEMENTS

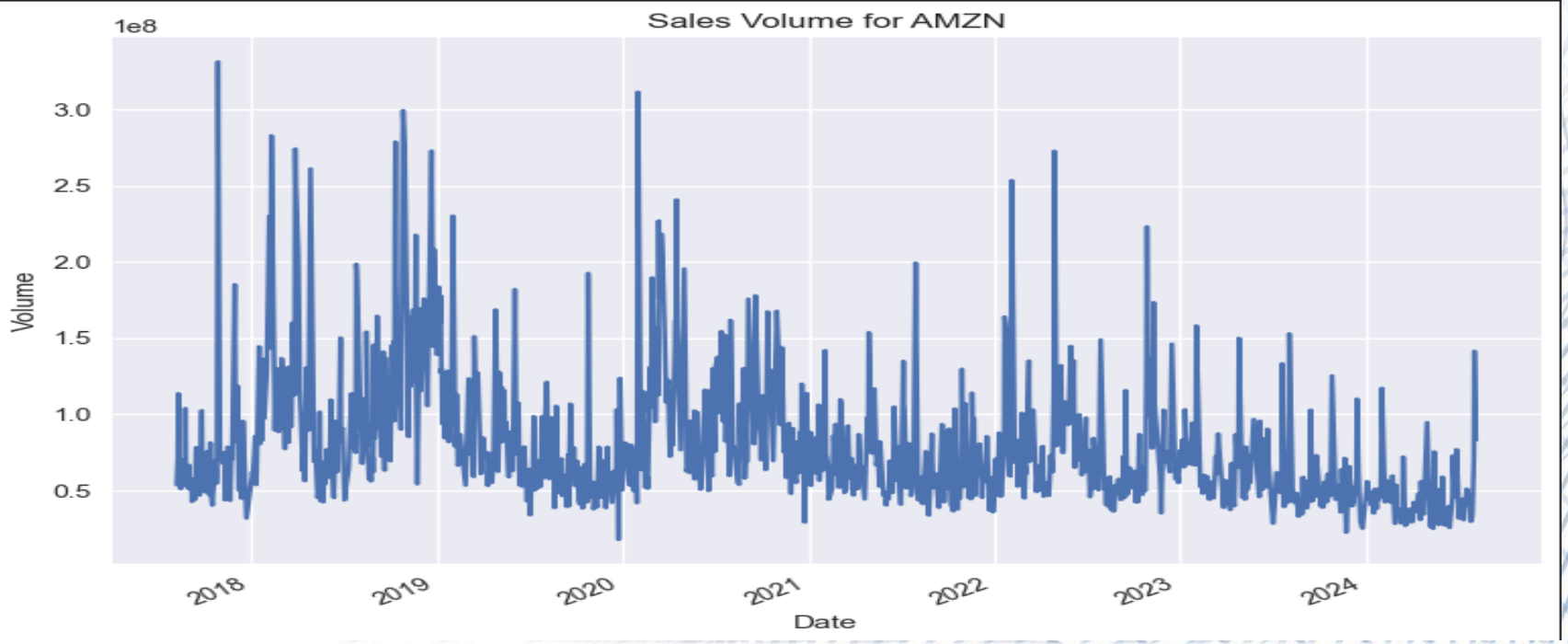
ON BALANCE VOLUME (OBV)

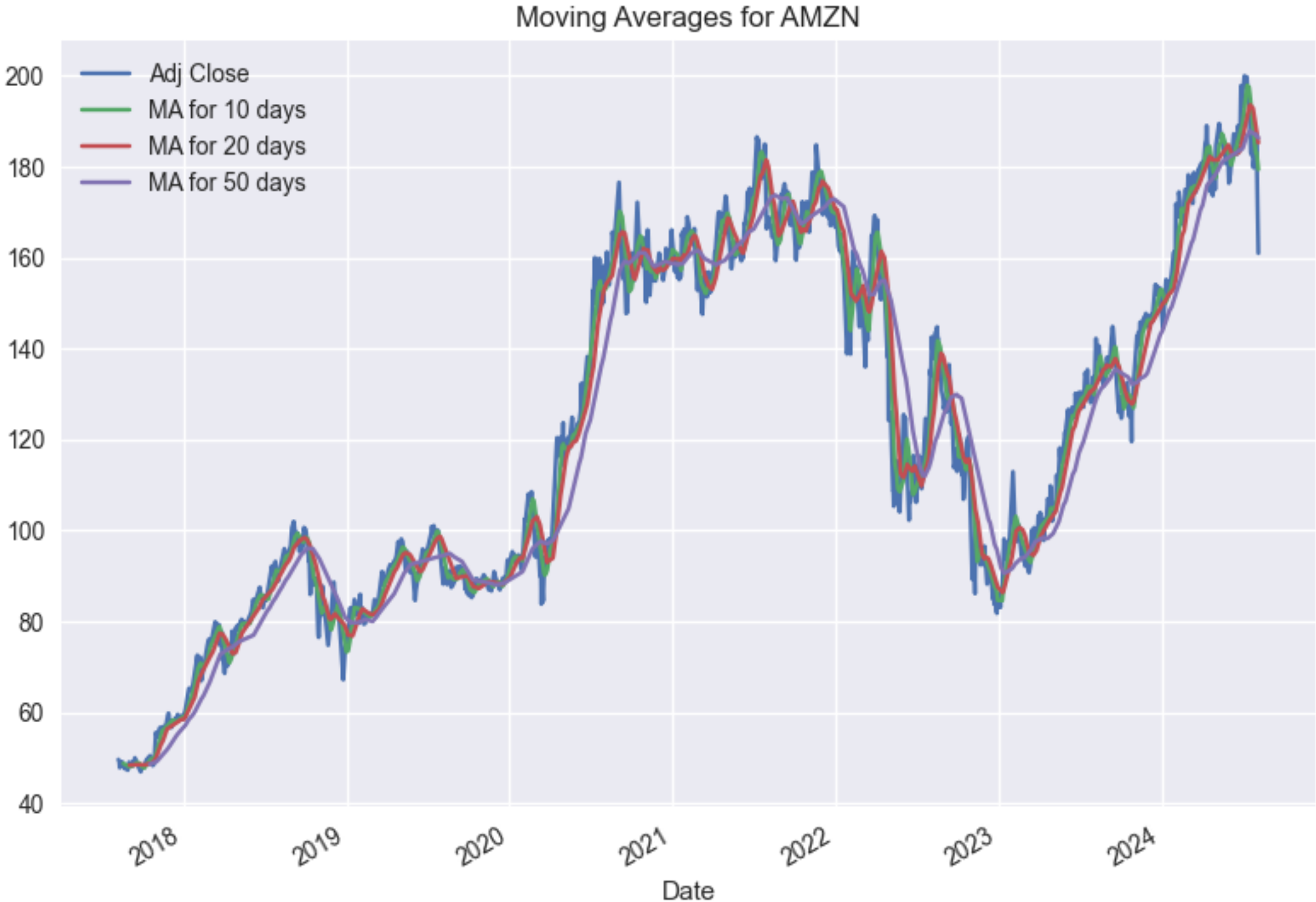
RELATIVE STRENGTH INDICATOR (RSI)

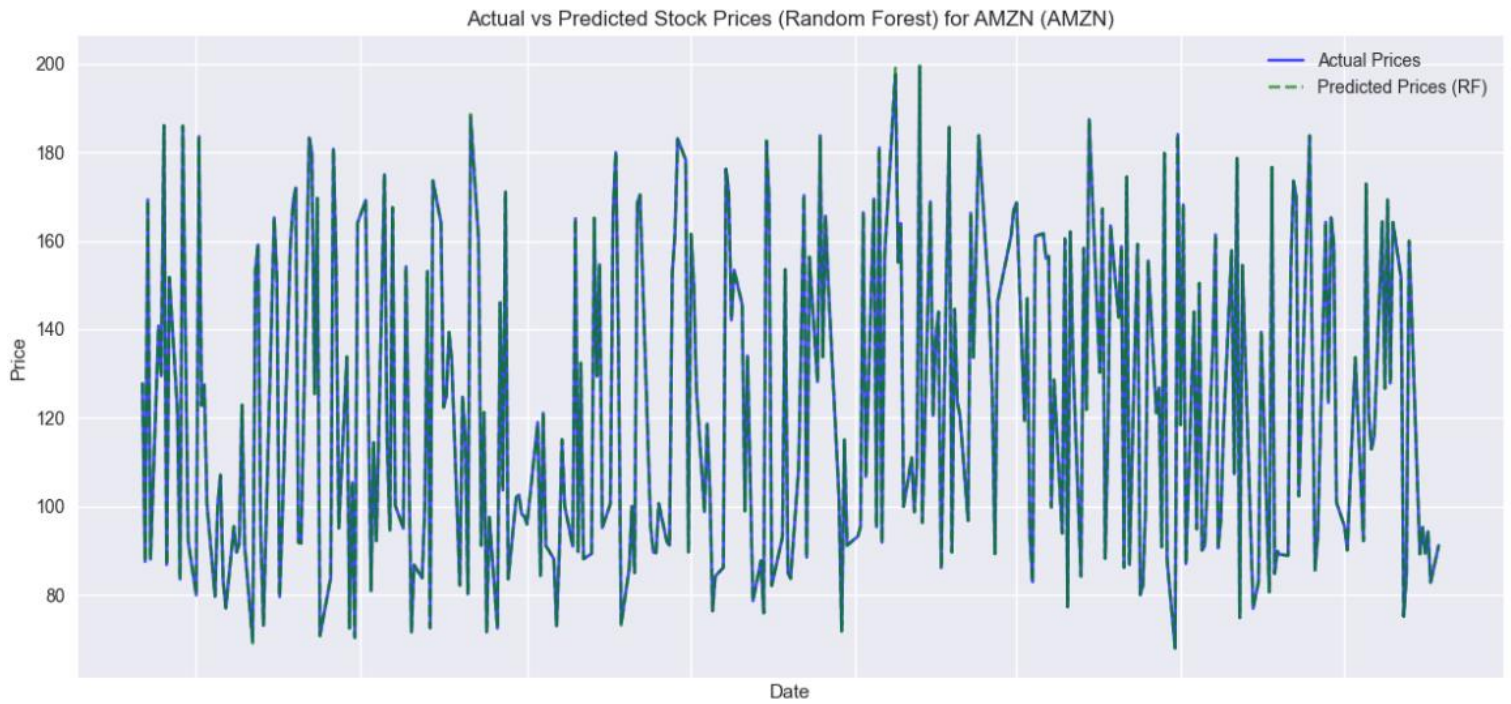
AVERAGE TRUE RANGE (ATR)

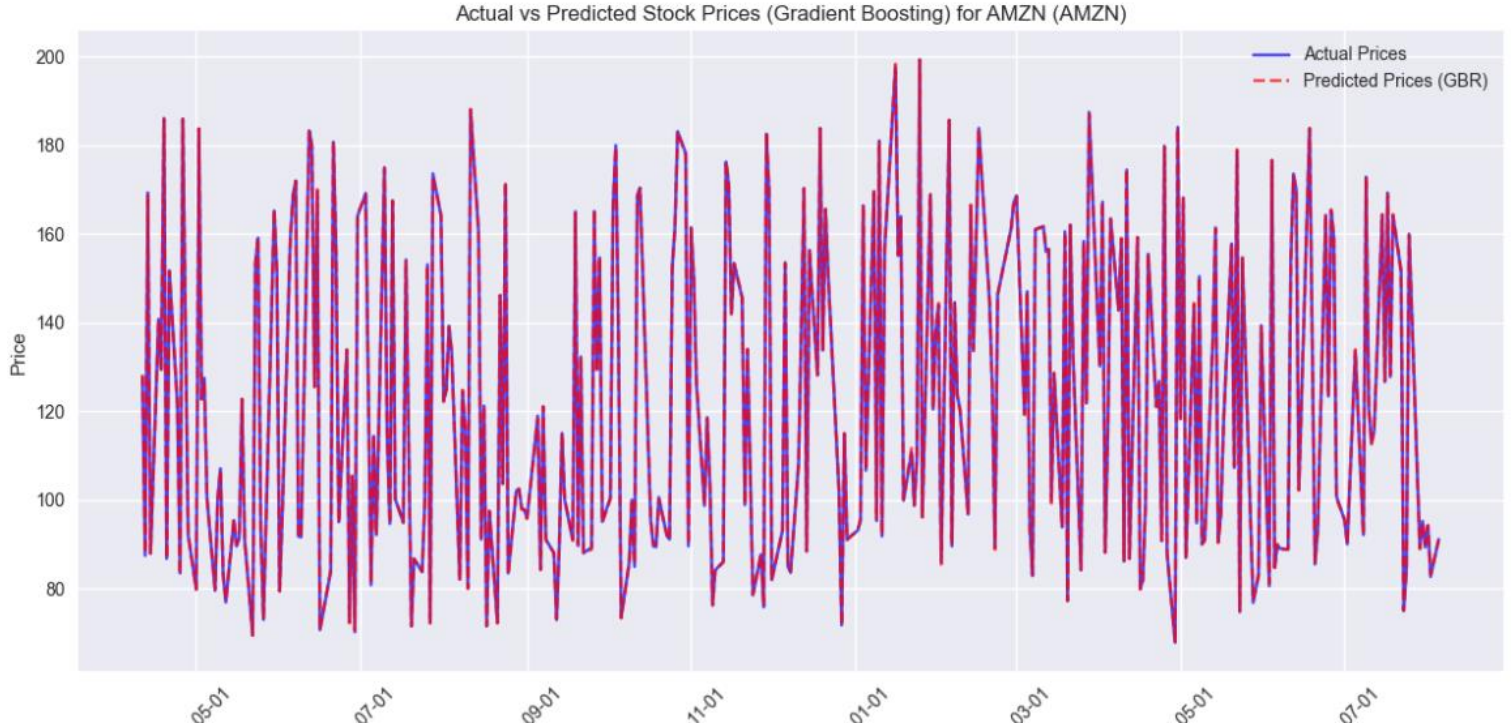
OptionsSwing

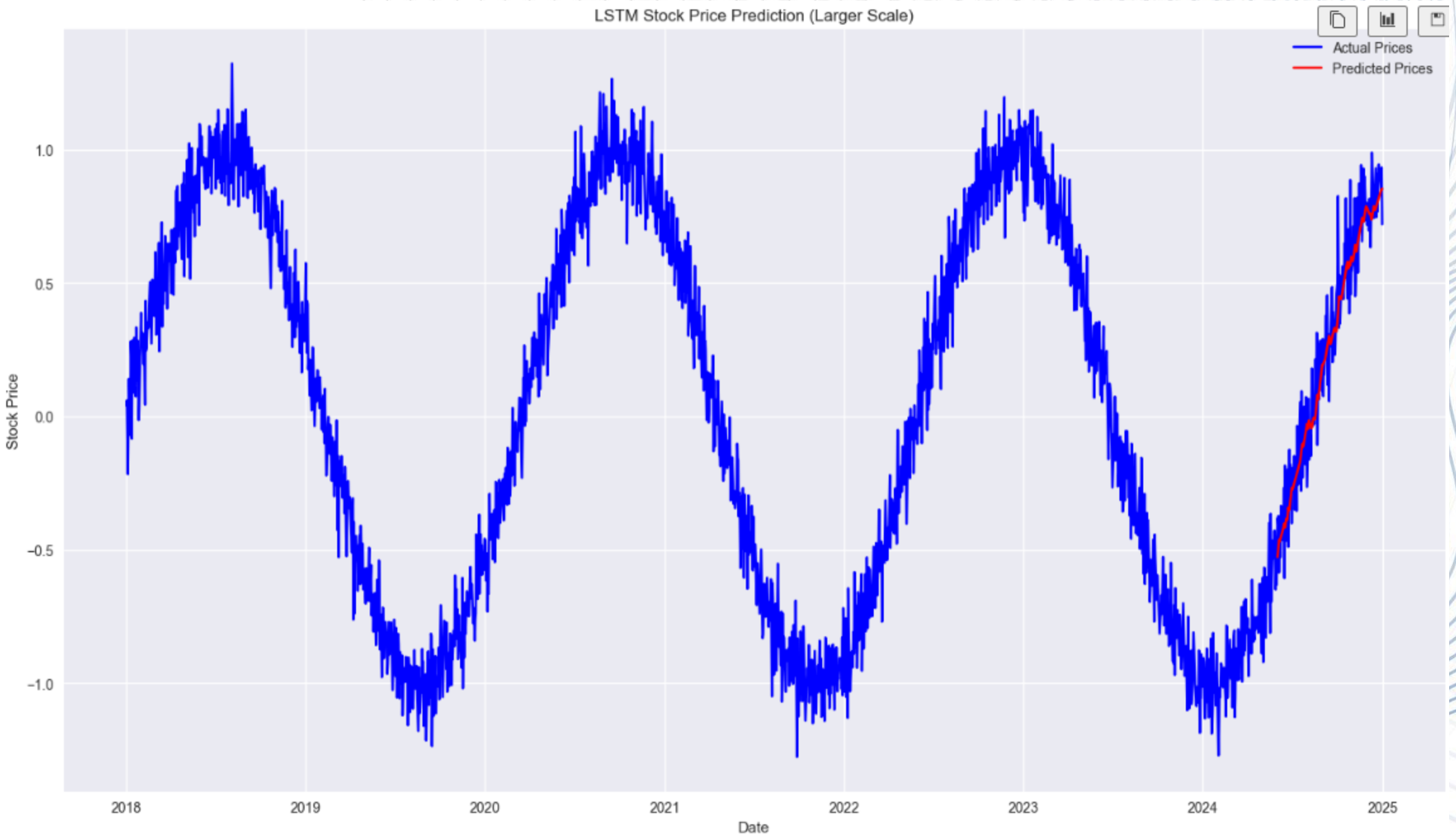

Closing Price of AMZN
Adj Close
200
180
160
140
120
100
80
60
40
2018
2019
2020
2021
2022
2023
2024
Date

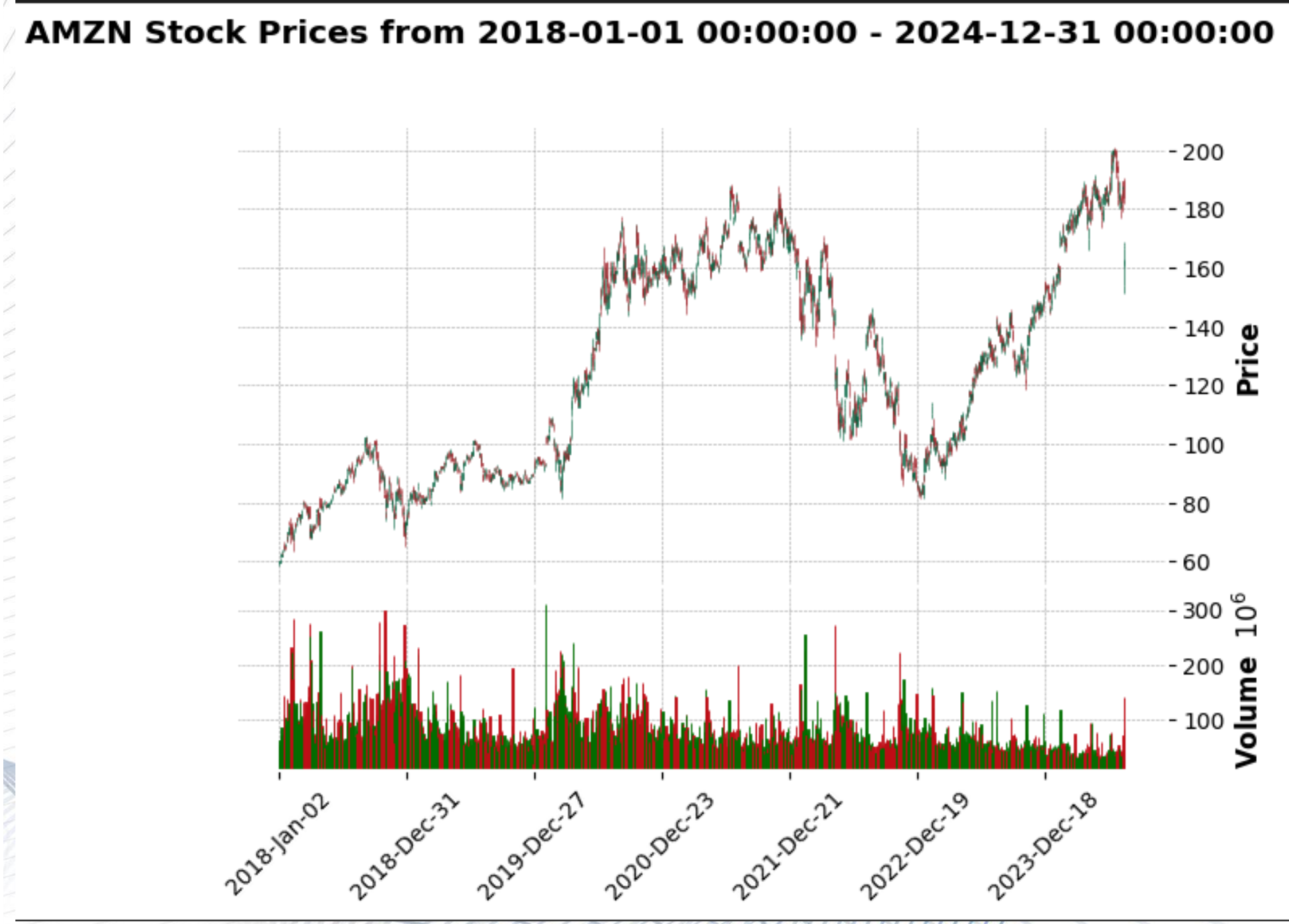

Sales Volume for AMZN
1e8
Volume
3.0
2.5
2.0
1.5
1.0
0.5
2018
2019
2020
2021
2022
2023
2024
Date


Moving Averages for AMZN
Adj Close
MA for 10 days
MA for 20 days
MA for 50 days
200
180
160
140
120
100
80
60
40
2018
2019
2020
2021
2022
2023
2024
Date


Actual vs Predicted Stock Prices (Random Forest) for AMZN (AMZN)
Actual Prices
Predicted Prices (RF)
Price
Date
200
180
160
140
120
100
80


Actual vs Predicted Stock Prices (Gradient Boosting) for AMZN (AMZN)
Actual Prices
Predicted Prices (GBR)
Price
200
180
160
140
120
100
80


LSTM Stock Price Prediction (Larger Scale)
Actual Prices
Predicted Prices
Stock Price
1.0
0.5
0.0
−0.5
−1.0
2018
2019
2020
2021
2022
2023
2024
2025
Date


AMZN Stock Prices from 2018-01-01 00:00:00 - 2024-12-31 00:00:00
200
180
160
140
120
100
80
60
Price
300
200
100
Volume 10^6
2018-Jan-02
2018-Dec-31
2019-Dec-27
2020-Dec-23
2021-Dec-21
2022-Dec-19
2023-Dec-18

5. Technical Indicators and Strategies

A technical indicator is a series of data points that are derived by applying a formula to the price data of a security. Basically, they are price-derived indicators that use formulas to translate the momentum or price levels into quantifiable time series.

There are two categories of indicator: leading and lagging, and four types: trend, momentum, volatility and volume, which serve three broad functions: to alert, to confirm and to predict

5.1 Trend-following strategies

Trend-following is about profiting from the prevailing trend through buying an asset when its price trend goes up, and selling when its trend goes down, expecting price movements to continue.

5.1.1 Moving averages

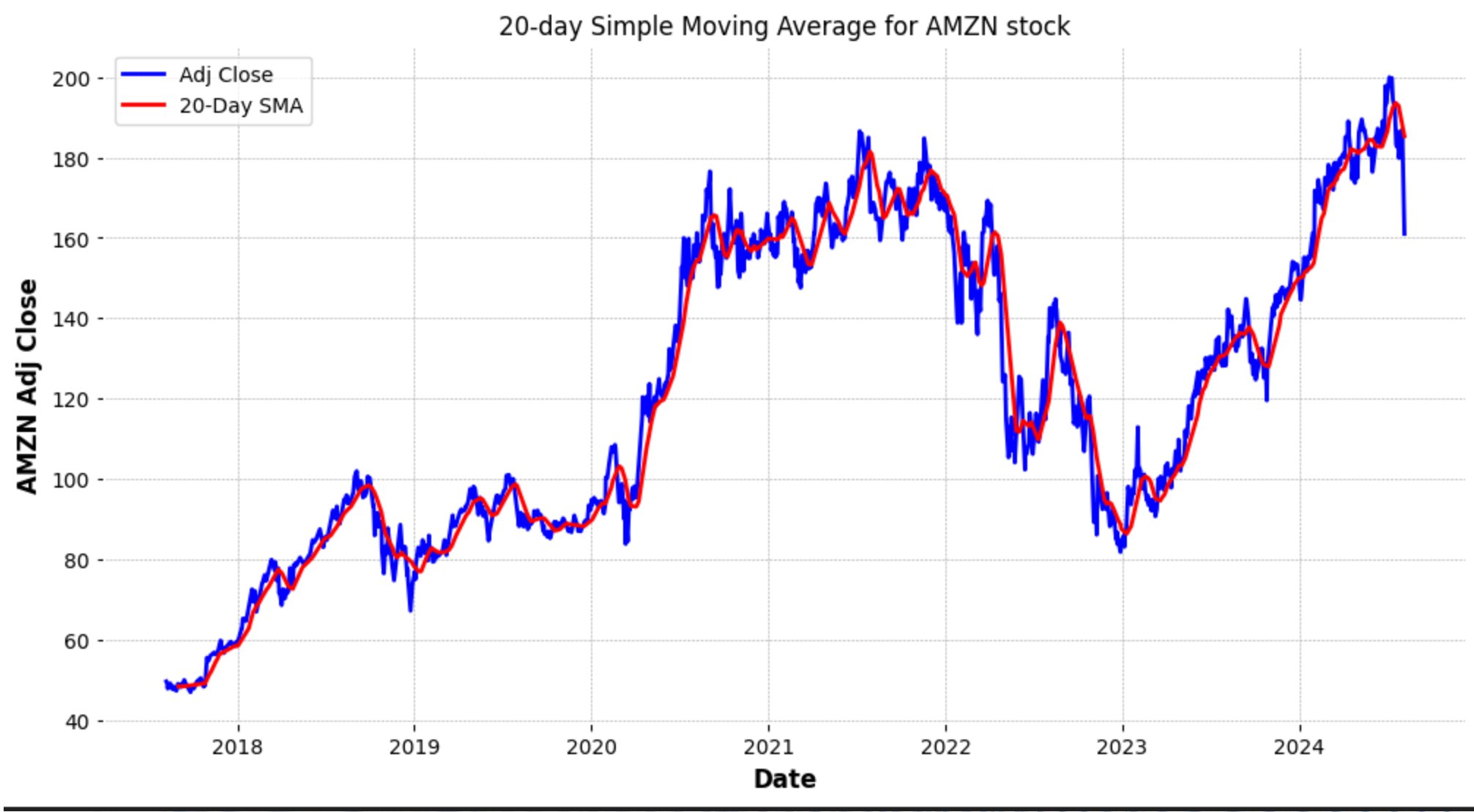
Moving averages smooth a series filtering out noise to help identify trends, one of the fundamental principles of technical analysis being that prices move in trends. Types of moving averages include simple, exponential, smoothed, linear-weighted, MACD, and as lagging indicators they follow the price action and are commonly referred to as trend-following indicators.

5.1.2 Simple Moving Average (SMA)

The simplest form of a moving average, known as a Simple Moving Average (SMA), is calculated by taking the arithmetic mean of a given set of values over a set time period. This model is probably the most naive approach to time series modelling and simply states that the next observation is the mean of all past observations and each value in the time period carries equal weight.

Modelling this an as average calculation problem we would try to predict the future stock market prices (for example, xt+1) as an average of the previously observed stock market prices within a fixed size window (for example, xt-n, ..., xt). This helps smooth out the price data by creating a constantly updated average price so that the impacts of random, short-term fluctuations on the price of a stock over a specified time-frame are mitigated.

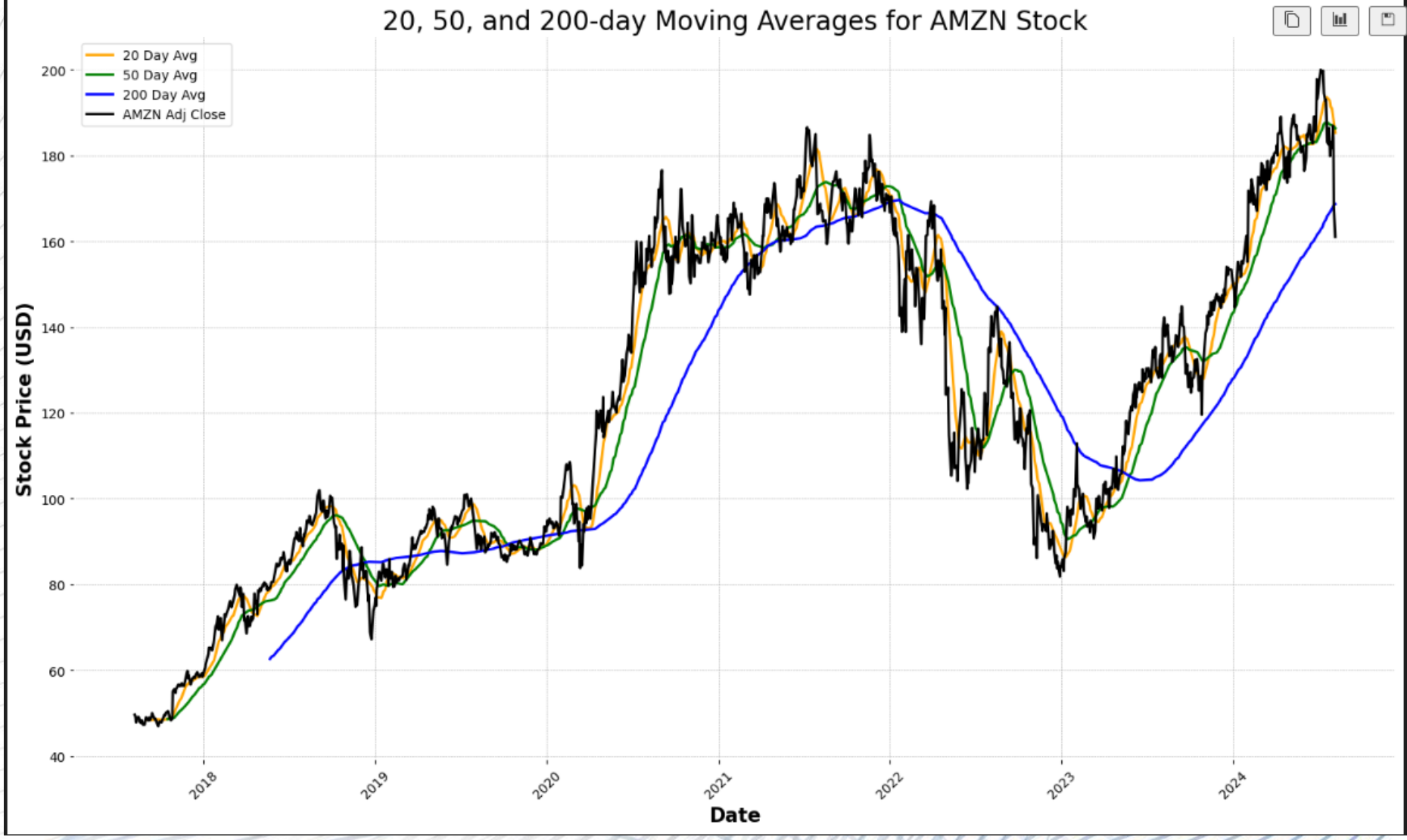
The SMA follows the time series removing noise from the signal and keeping the relevant information about the trend. If the stock price is above its moving average it is assumed that it will likely continue rising in an uptrend.

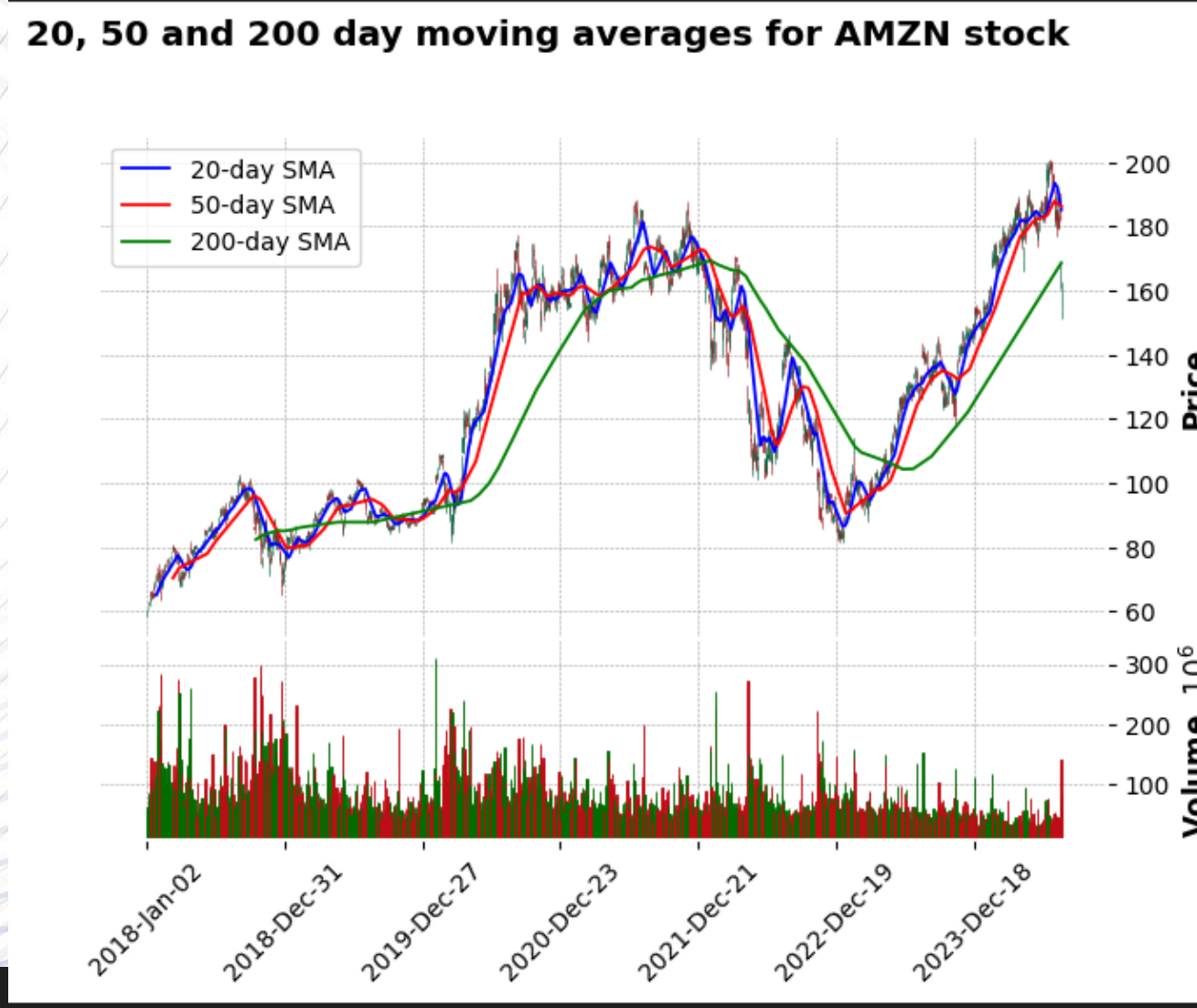


The chart shows that the 20-day moving average is the most sensitive to local changes, and the 200-day moving average the least. Here, the 200-day moving average indicates an overall bullish trend - the stock is trending upward over time. The 20- and 50-day moving averages are at times bearish and at other times bullish.

The major drawback of moving averages, however, is that because they are lagging, and smooth out prices, they tend to recognise reversals too late and are therefore not very helpful when used alone.

Trading Strategy
The moving average crossover trading strategy will be to take two moving averages - 20-day (fast) and 200-day (slow) - and to go long (buy) when the fast MA goes above the slow MA and to go short (sell) when the fast MA goes below the slow MA.





Date	Open	High	Low	Close	Adj Close	Volume	Ticker	20d	50d	200d
2024-07-30	184.720001	185.860001	179.380005	181.710007	181.710007	39508600	AMZN	190.16	187.04	167.81
2024-07-31	185.050003	187.940002	184.460007	186.979996	186.979996	41667300	AMZN	189.51	187.09	168.09
2024-08-01	189.289993	190.600006	181.869995	184.070007	184.070007	70435600	AMZN	188.83	187.10	168.36
2024-08-02	166.750000	168.770004	160.550003	167.899994	167.899994	141448400	AMZN	187.23	186.80	168.53
2024-08-05	154.210007	162.960007	151.610001	161.020004	161.020004	82961700	AMZN	185.31	186.35	168.68

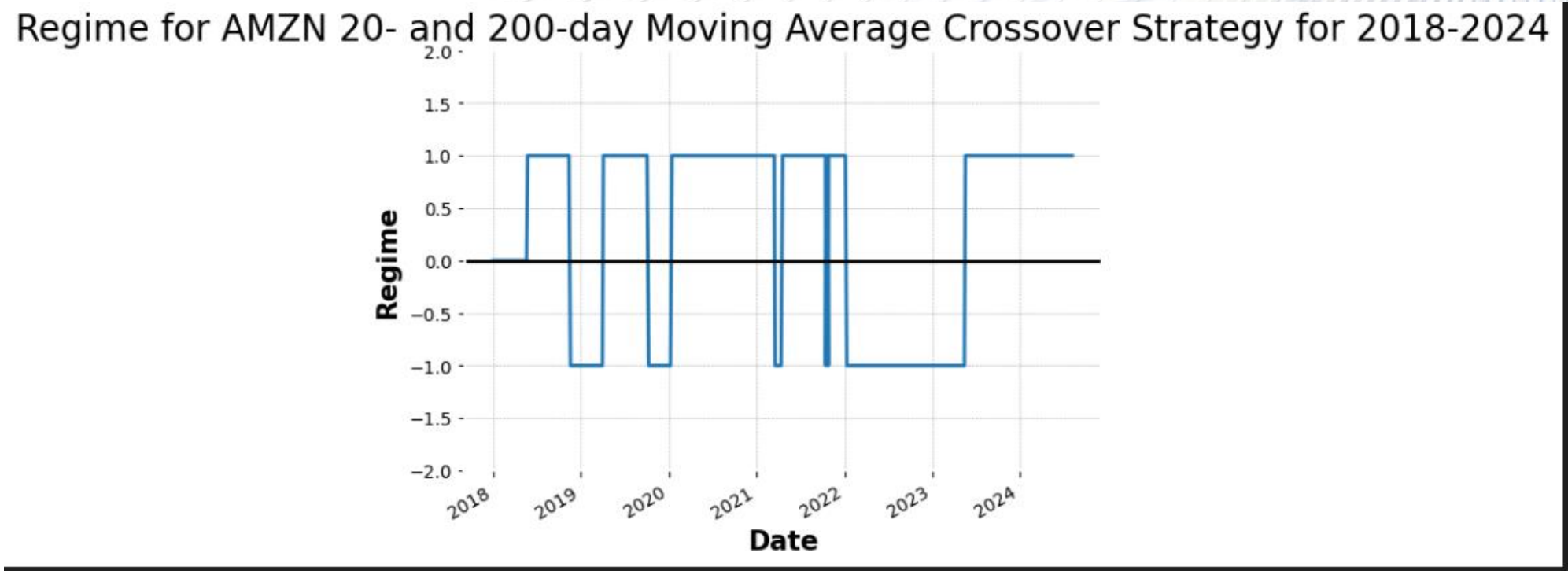
Backtesting

Before using the strategy we will evaluate the quality of it first by backtesting, or looking at how profitable it is on historical data.

```
Regime
 1    1039
-1     522
 0     199
Name: count, dtype: int64
```

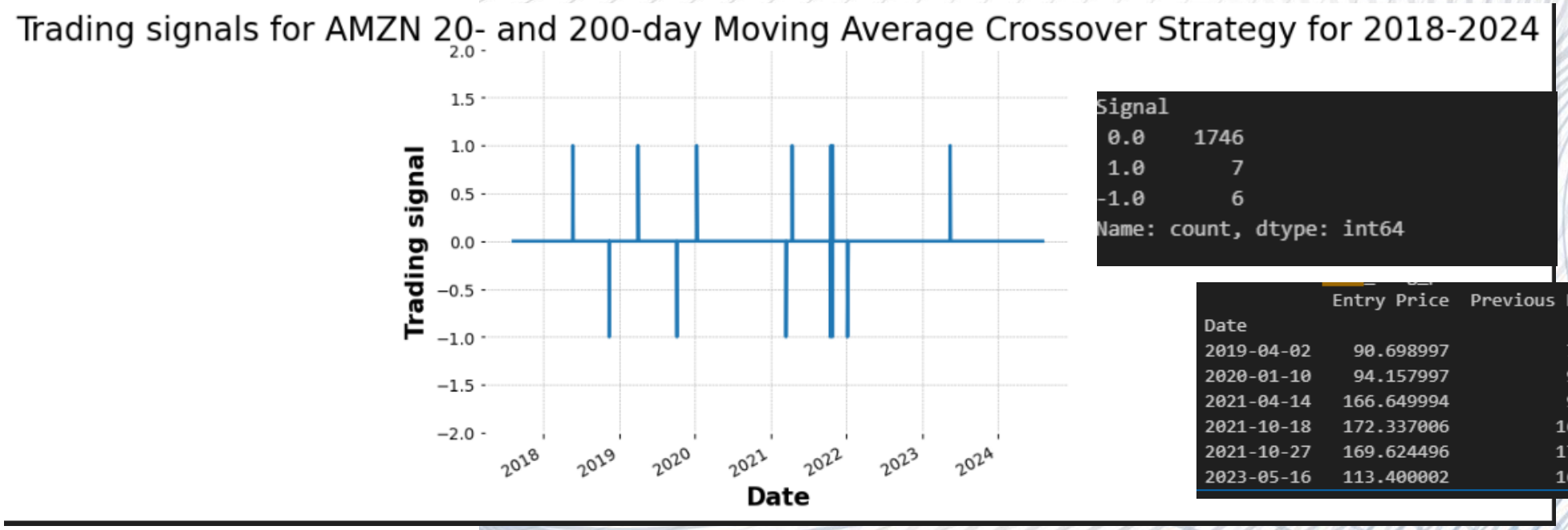
Date	Open	High	Low	Close	Adj Close	Volume	Ticker	20d	50d	200d	20d-200d
2024-07-30	184.720001	185.860001	179.380005	181.710007	181.710007	39508600	AMZN	190.16	187.04	167.81	22.35
2024-07-31	185.050003	187.940002	184.460007	186.979996	186.979996	41667300	AMZN	189.51	187.09	168.09	21.42
2024-08-01	189.289993	190.600006	181.869995	184.070007	184.070007	70435600	AMZN	188.83	187.10	168.36	20.47
2024-08-02	166.750000	168.770004	160.550003	167.899994	167.899994	141448400	AMZN	187.23	186.80	168.53	18.70
2024-08-05	154.210007	162.960007	151.610001	161.020004	161.020004	82961700	AMZN	185.31	186.35	168.68	16.63

For 1039 days the market was bullish, for 522 days it was bearish, and neutral for 199 days for the time period 2018-2024.



Date	Open	High	Low	Close	Adj Close	Volume	Ticker	20d	50d	200d	20d-200d	Regime	Signal
2024-07-30	184.720001	185.860001	179.380005	181.710007	181.710007	39508600	AMZN	190.16	187.04	167.81	22.35	1	0.0
2024-07-31	185.050003	187.940002	184.460007	186.979996	186.979996	41667300	AMZN	189.51	187.09	168.09	21.42	1	0.0
2024-08-01	189.289993	190.600006	181.869995	184.070007	184.070007	70435600	AMZN	188.83	187.10	168.36	20.47	1	0.0
2024-08-02	166.750000	168.770004	160.550003	167.899994	167.899994	141448400	AMZN	187.23	186.80	168.53	18.70	1	0.0
2024-08-05	154.210007	162.960007	151.610001	161.020004	161.020004	82961700	AMZN	185.31	186.35	168.68	16.63	1	0.0

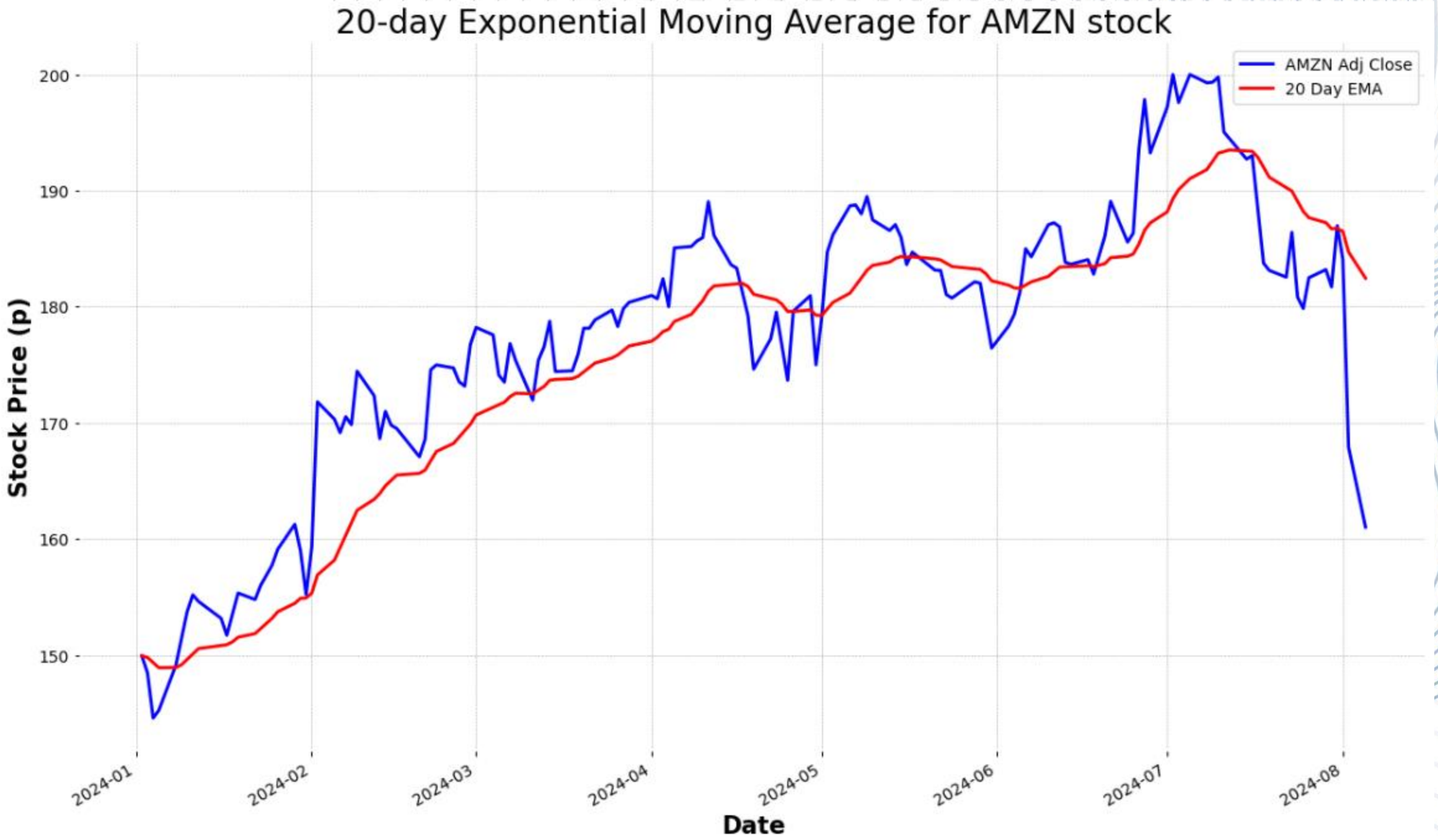
Date	Price	Regime	Signal
2018-05-22	79.070000	1	Buy
2018-11-14	79.950500	-1	Sell
2019-04-02	90.698997	1	Buy
2019-10-07	86.633003	-1	Sell
2020-01-10	94.157997	1	Buy
2021-03-16	154.593002	-1	Sell
2021-04-14	166.649994	1	Buy
2021-10-15	170.451004	-1	Sell
2021-10-18	172.337006	1	Buy
2021-10-25	166.018494	-1	Sell
2021-10-27	169.624496	1	Buy
2022-01-06	163.253998	-1	Sell
2023-05-16	113.400002	1	Buy

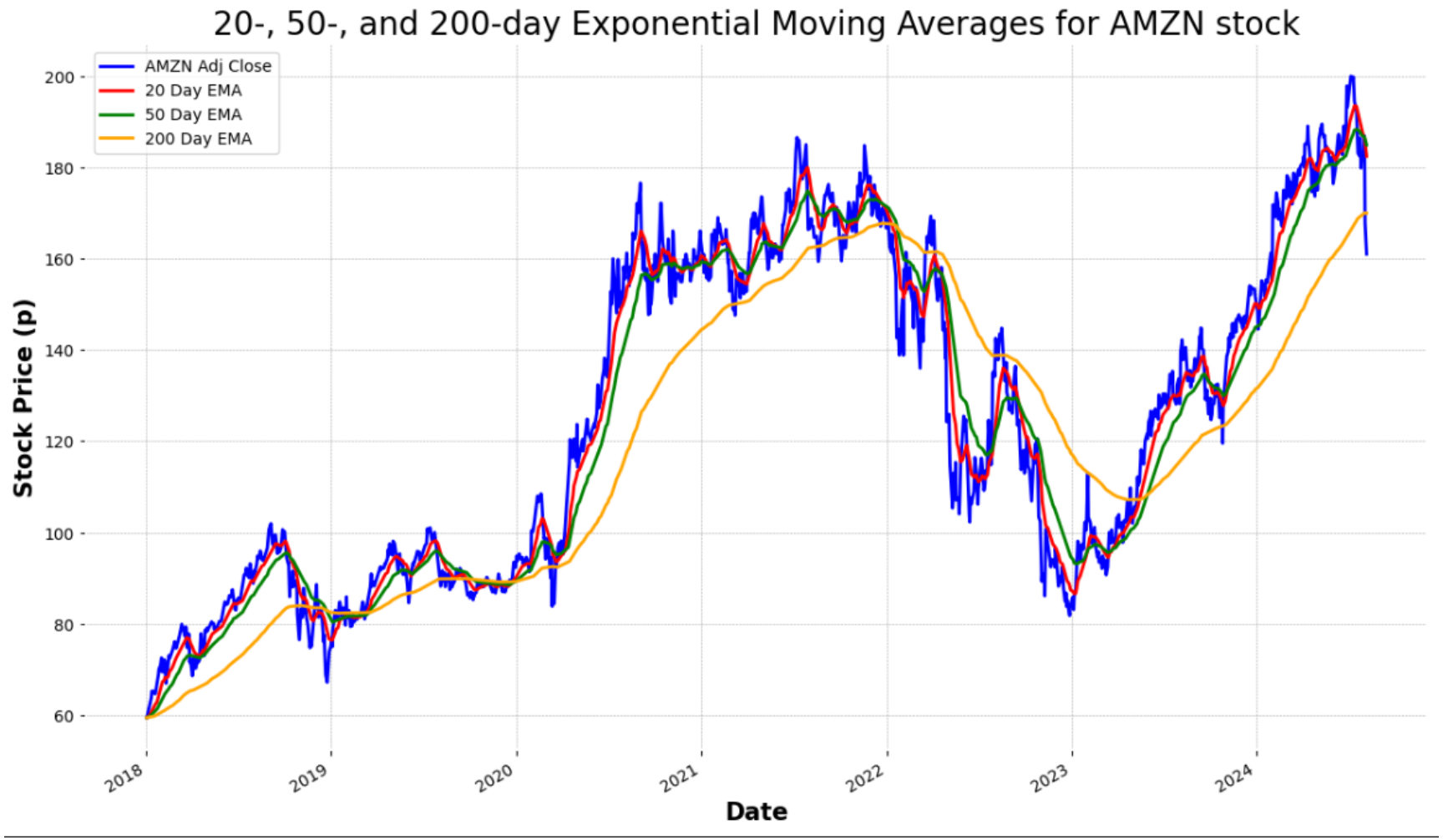



```
Signal
 0.0    1746
 1.0       7
-1.0       6
Name: count, dtype: int64
```

Date	Entry Price	Previous Buy Price	Profit	End Date
2019-04-02	90.698997	79.070000	11.628998	2019-04-02
2020-01-10	94.157997	90.698997	3.459000	2020-01-10
2021-04-14	166.649994	94.157997	72.491997	2021-04-14
2021-10-18	172.337006	166.649994	5.687012	2021-10-18
2021-10-27	169.624496	172.337006	-2.712509	2021-10-27
2023-05-16	113.400002	169.624496	-56.224495	2023-05-16

We would buy amzn stock 7 times and sell 6 times. If we only go long 6 trades will be engaged in over the 6-year period, while if we pivot from a long to a short position every time a long position is terminated, we would engage in 7 trades total. It is worth bearing in mind that trading more frequently isn’t necessarily good as trades are never free.


20-day Exponential Moving Average for AMZN stock
AMZN Adj Close
20 Day EMA
Stock Price (p)
200
190
180
170
160
150
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
Date

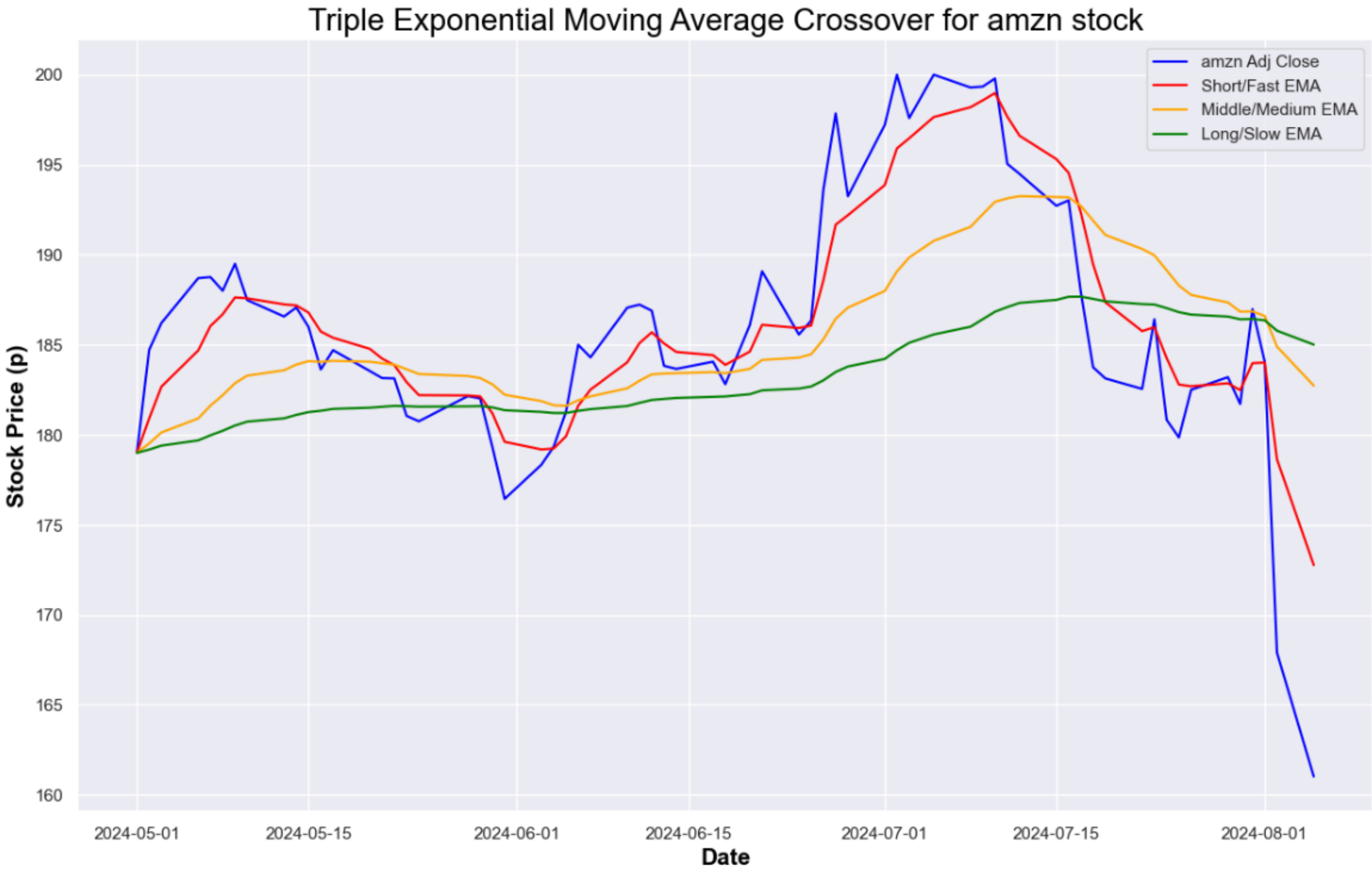


5.1.5 Triple Moving Average Crossover Strategy

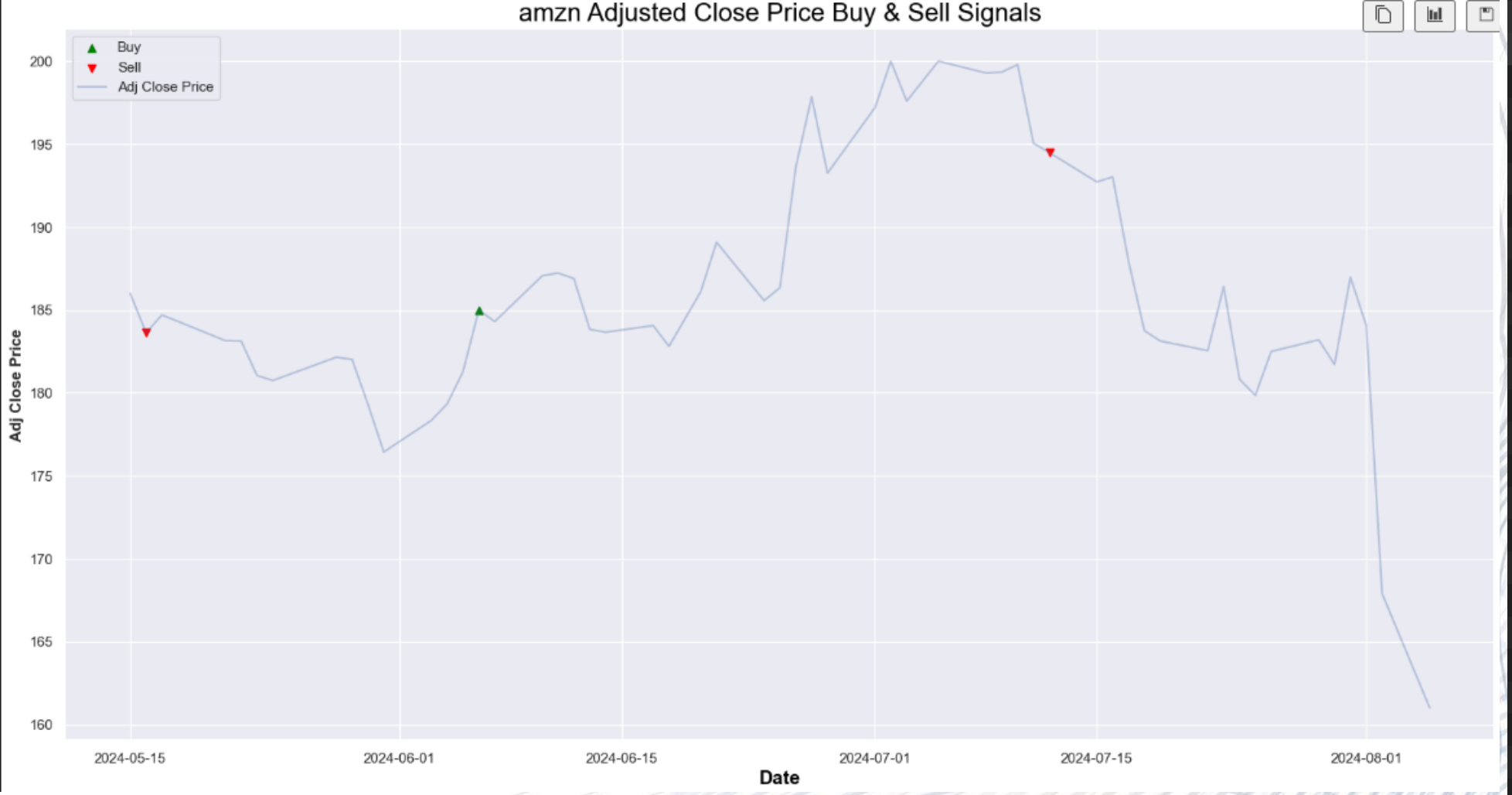
This strategy uses three moving moving averages - short/fast, middle/medium and long/slow - and has two buy and sell signals.

The first is to buy when the middle/medium moving average crosses above the long/slow moving average and the short/fast moving average crosses above the middle/medium moving average. If we use this buy signal the strategy is to sell if the short/fast moving average crosses below the middle/medium moving average.

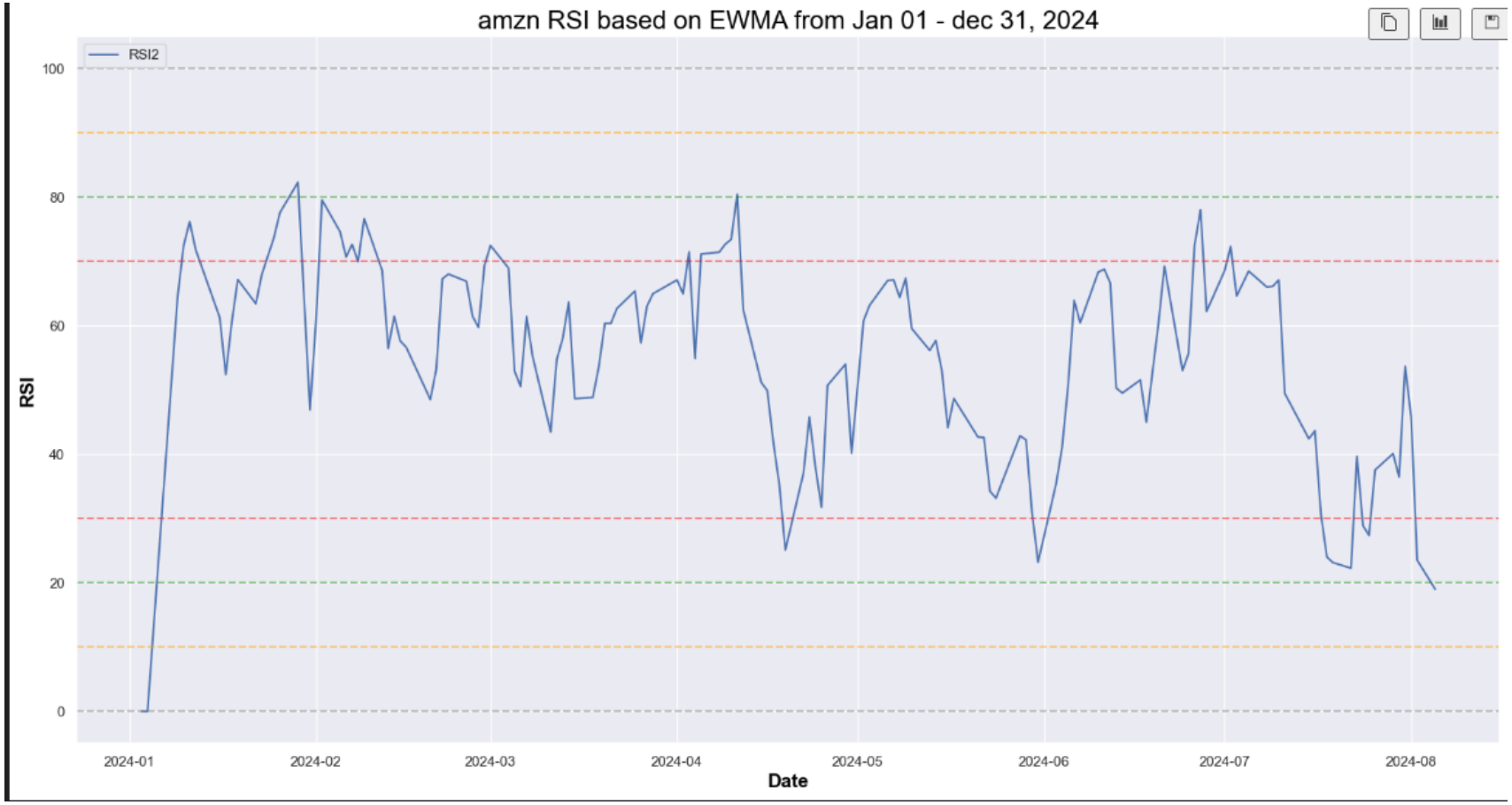
The second is to buy when the middle/medium moving average crosses below the long/slow moving average and the short/fast moving average crosses below the middle/medium moving average. If we use this buy signal the strategy is to sell if the short/fast moving average crosses above the middle/medium moving average.


Triple Exponential Moving Average Crossover for amzn stock
amzn Adj Close
Short/Fast EMA
Middle/Medium EMA
Long/Slow EMA
Stock Price (p)
200
195
190
185
180
175
170
165
160
2024-05-01
2024-05-15
2024-06-01
2024-06-15
2024-07-01
2024-07-15
2024-08-01
Date


Trading signals for amzn stock
amzn Adj Close
Short/Fast EMA
Middle/Medium EMA
Long/Slow EMA
Buy Signal
Sell Signal
Stock Price (p)
200
195
190
185
180
175
170
165
160
2024-05-01
2024-05-15
2024-06-01
2024-06-15
2024-07-01
2024-07-15
2024-08-01
Date



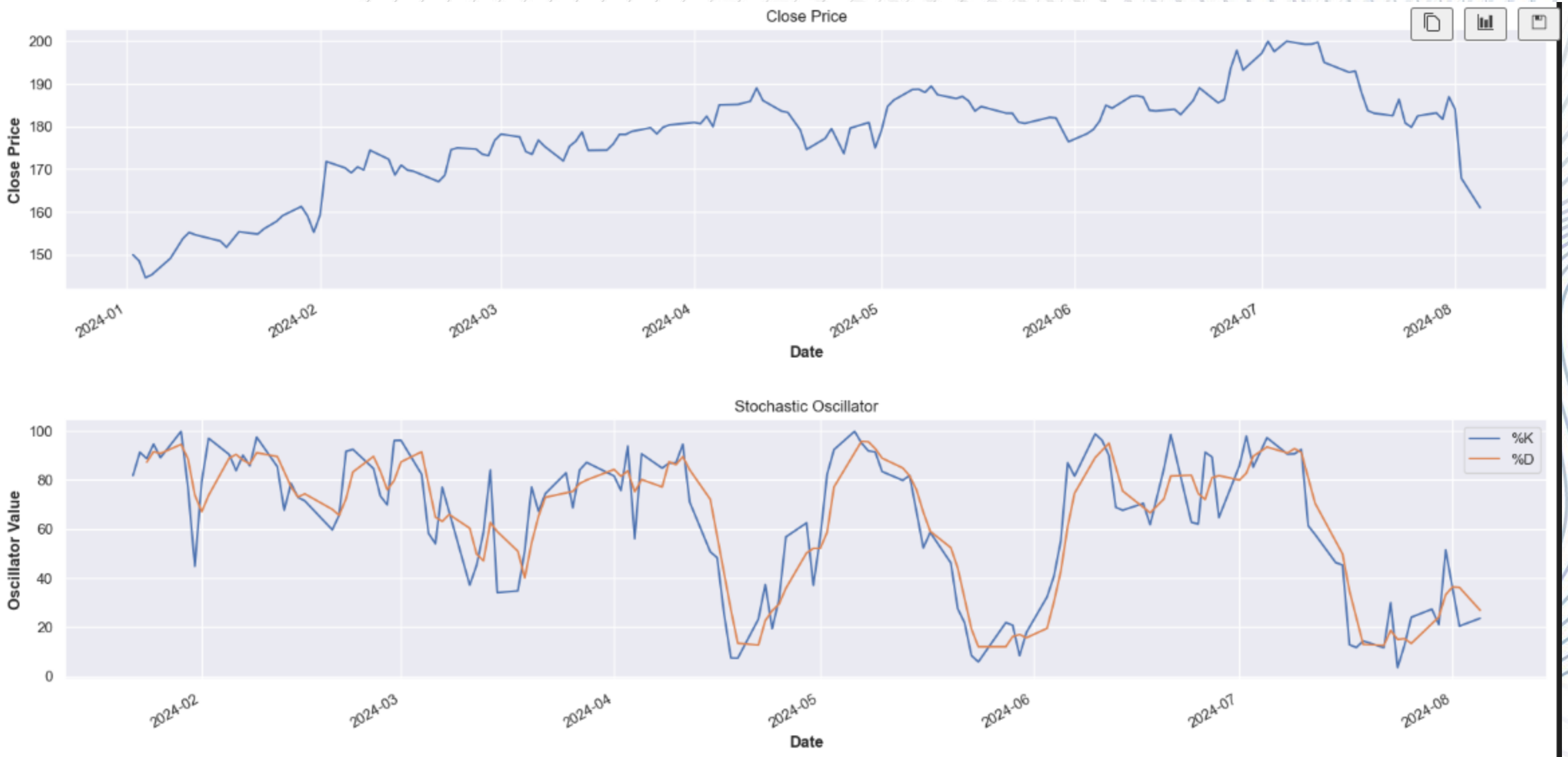
Date	Adj Close	MACD	Signal Line
2024-05-15	185.990005	0.000000	0.000000
2024-05-16	183.630005	-0.188262	-0.037652
2024-05-17	184.699997	-0.248260	-0.079774
2024-05-20	183.539993	-0.384974	-0.140814
2024-05-21	183.149994	-0.518809	-0.216413
2024-05-22	183.130005	-0.619349	-0.297000
2024-05-23	181.050003	-0.856987	-0.408997
2024-05-24	180.750000	-1.057336	-0.538665
2024-05-28	182.149994	-1.090575	-0.649047
2024-05-29	182.020004	-1.114558	-0.742149
2024-05-30	179.320007	-1.336031	-0.860926
2024-05-31	176.440002	-1.724069	-1.033554
2024-06-03	178.339996	-1.856873	-1.198218
2024-06-04	179.339996	-1.859988	-1.330572
2024-06-05	181.279999	-1.686475	-1.401753
2024-06-06	185.000000	-1.234560	-1.368314
2024-06-07	184.300003	-0.922267	-1.279105
2024-06-10	187.059998	-0.446912	-1.112666
2024-06-11	187.229996	-0.055829	-0.901299
2024-06-12	186.889999	0.224089	-0.676221
2024-06-13	183.830002	0.196742	-0.501628
2024-06-14	183.660004	0.159513	-0.369400
2024-06-17	184.059998	0.160436	-0.263433
2024-06-18	182.809998	0.059615	-0.198823
2024-06-20	186.100006	0.242396	-0.110579
2024-06-21	189.080002	0.620559	0.035648
2024-06-24	185.570007	0.629769	0.154472
2024-06-25	186.339996	0.691231	0.261824
2024-06-26	193.610001	1.311451	0.471750
2024-06-27	197.850006	2.120668	0.801533
2024-06-28	193.250000	2.363550	1.113937
2024-07-01	197.199997	2.842007	1.459551

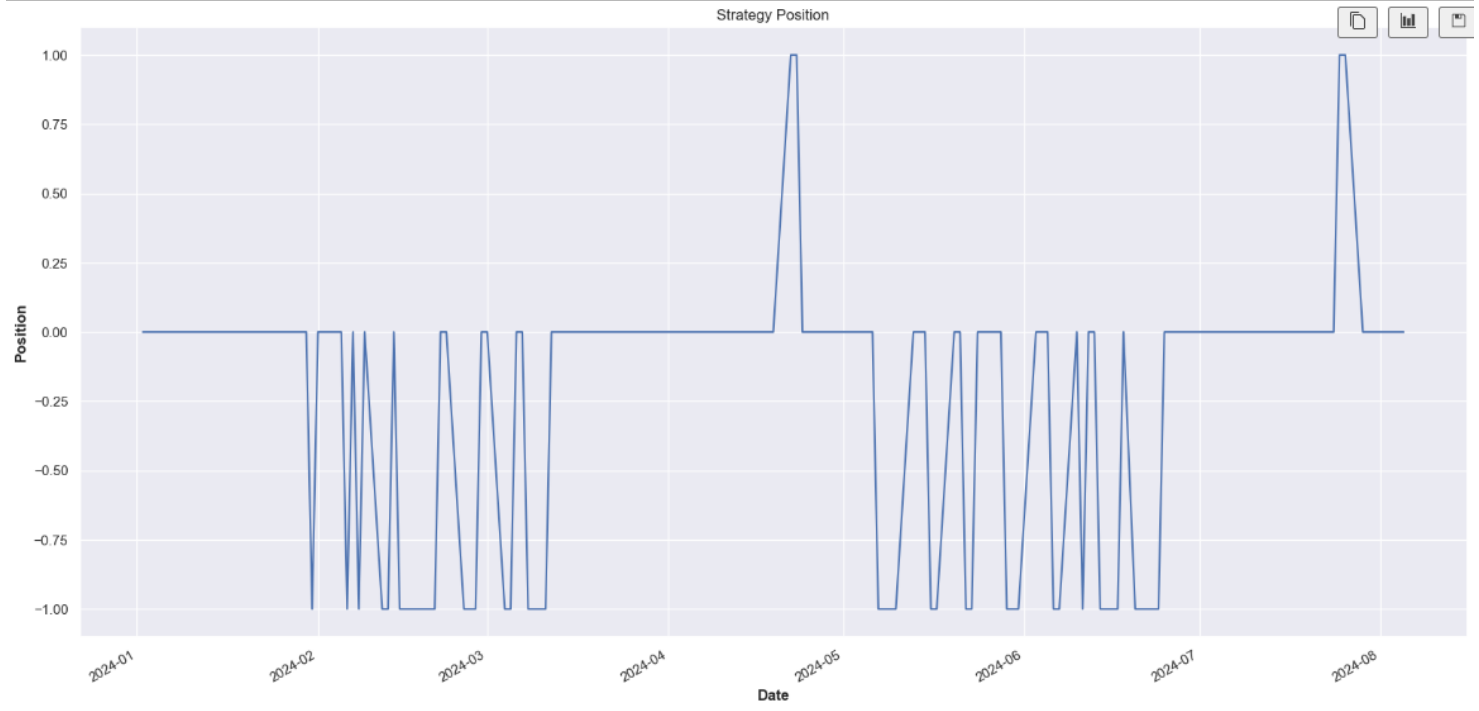


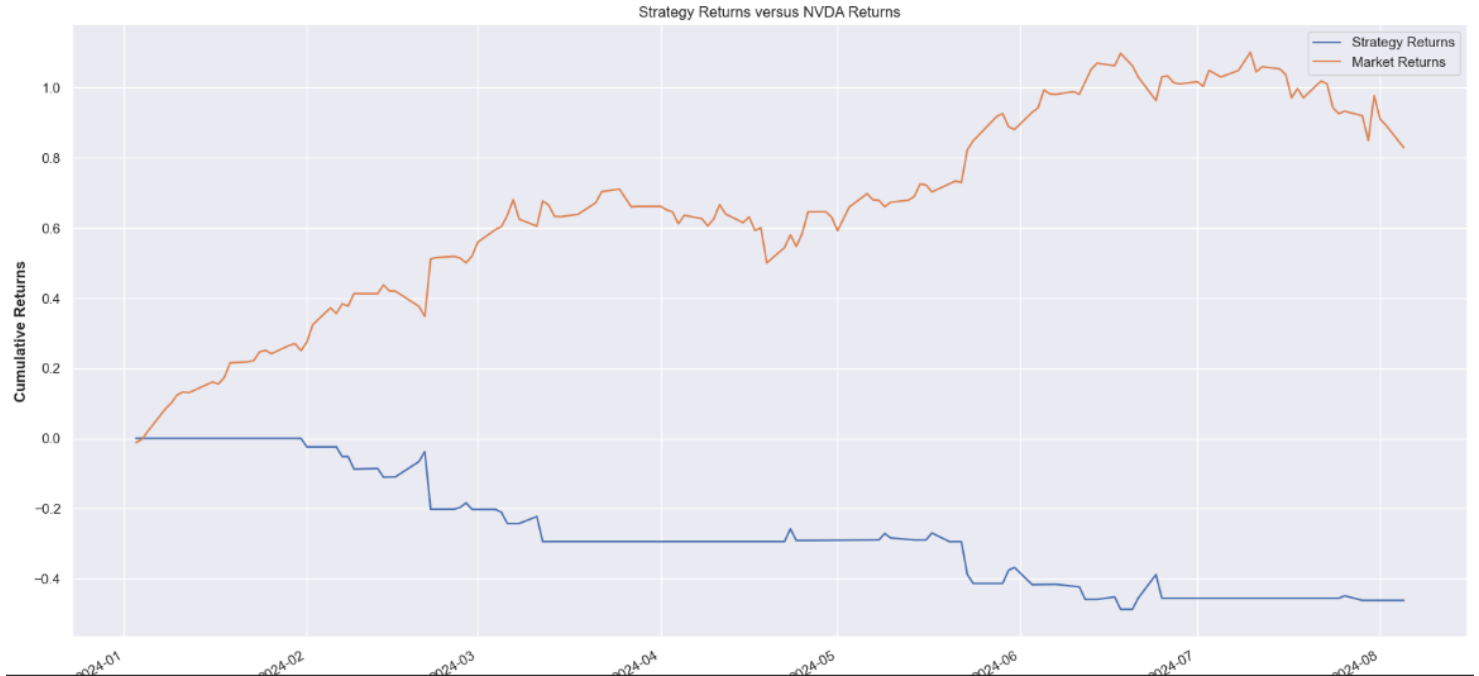
5.2.2 Money Flow Index (MFI)
Money Flow Index (MFI) is a technical oscillator, and momentum indicator, that uses price and volume data for identifying overbought or oversold signals in an asset. It can also be used to spot divergences which warn of a trend change in price. The oscillator moves between 0 and 100 and a reading of above 80 implies overbought conditions, and below 20 implies oversold conditions.

It is related to the Relative Strength Index (RSI) but incorporates volume, whereas the RSI only considers price.

It appears that RSI value dips below the 20 significant level in January 2024 indicating that the stock was oversold and presented a buying opportunity for an investor before a price rise.


Close Price
Close Price
200
190
180
170
160
150
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
Date
Stochastic Oscillator
Oscillator Value
100
80
60
40
20
0
%K
%D
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
Date


Strategy Position
Position
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
Date


Strategy Returns versus NVDA Returns
Strategy Returns
Market Returns
Cumulative Returns
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08

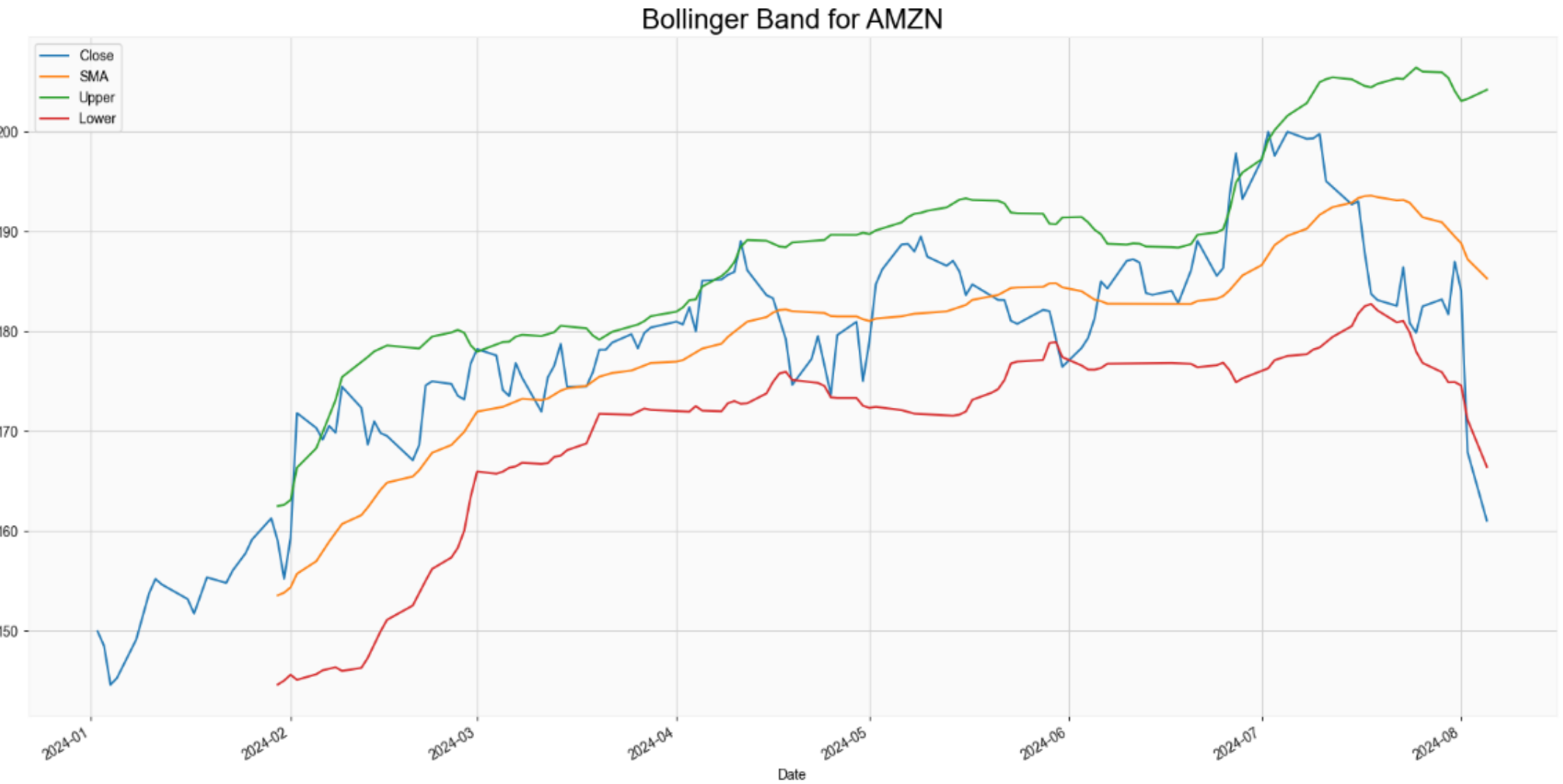

amzn Daily Price
Adj Closing Price
Price
200
190
180
170
160
9-Day ROC
% ROC
10.0%
5.0%
0.0%
-5.0%
-10.0%
-15.0%
Apr
May
Jun
Jul
Aug


amzn Daily Price
200
190
180
170
160
150
Price
150
100
50
Volume 10^6
Mar 13
Apr 11
May 09
Jun 07
Jul 09


amzn Daily Price
200
190
180
170
160
150
Price
150
100
50
10
Volume 10^6
10
0
−10
ROC
Mar 13
Apr 11
May 09
Jun 07
Jul 09

Volatility trading strategies

Volatility trading involves predicting the stability of an asset's value. Instead of trading on the price rising or falling, traders take a position on whether it will move in any direction.



5.3.1 Bollinger Bands

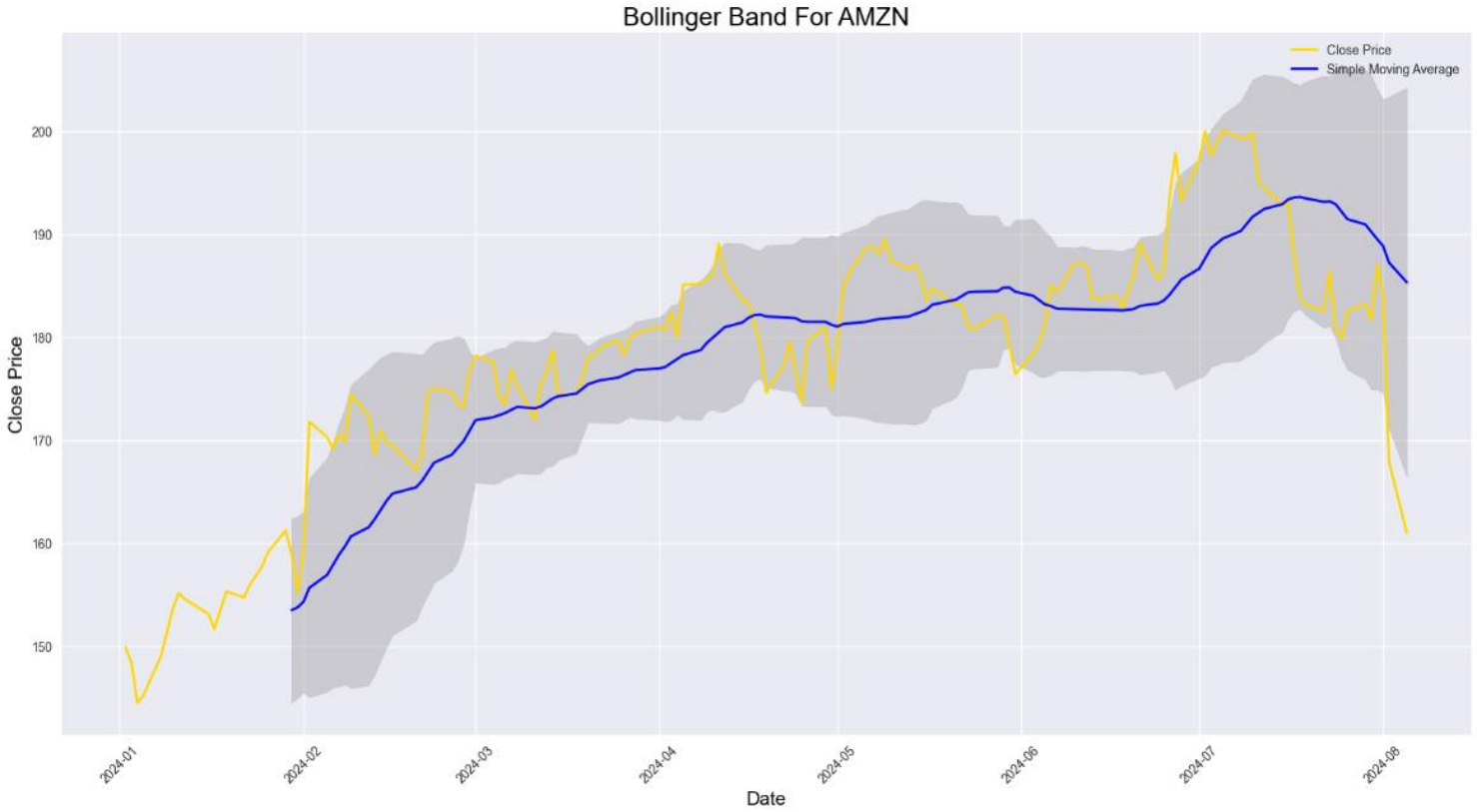
A Bollinger Band is a volatility indicator based on based on the correlation between the normal distribution and stock price and can be used to draw support and resistance curves. It is defined by a set of lines plotted two standard deviations (positively and negatively) away from a simple moving average (SMA) of the security's price, but can be adjusted to user preferences.

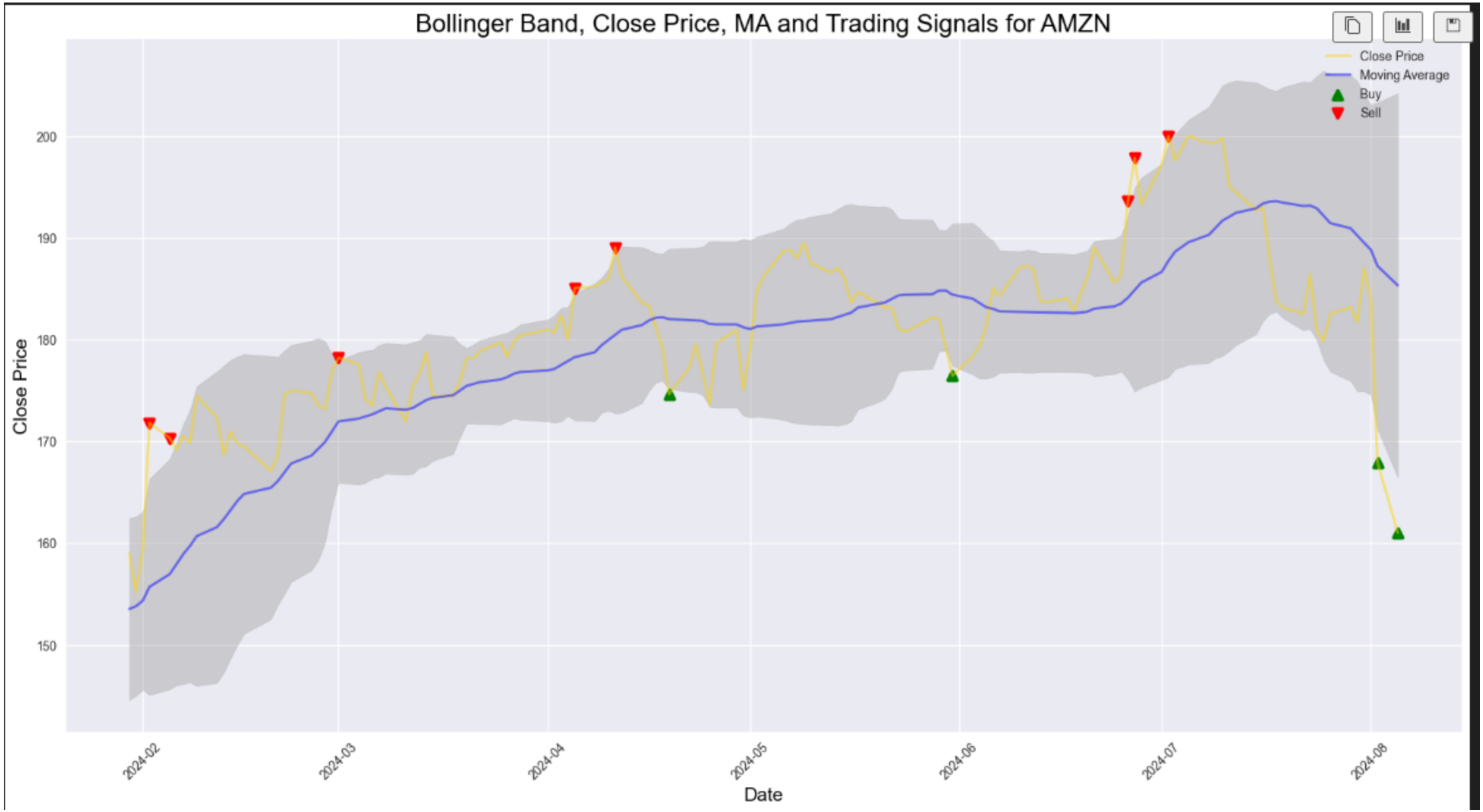
By default it calculates a 20-period SMA (the middle band), an upper band two standard deviations above the the moving average and a lower band two standard deviations below it.

If the price moves above the upper band this could indicate a good time to sell, and if it moves below the lower band it could be a good time to buy.

Whereas the RSI can only be used as a confirming factor inside a ranging market, not a trending market, by using Bollinger bands we can calculate the widening variable, or moving spread between the upper and the lower bands, that tells us if prices are about to trend and whether the RSI signals might not be that reliable.

Despite 90% of the price action happening between the bands, however, a breakout is not necessarily a trading signal as it provides no clue as to the direction and extent of future price movement.




Bollinger Band, Close Price, MA and Trading Signals for AMZN
Close Price
Moving Average
Buy
Sell
200
190
180
170
160
150
Close Price
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
Date

The Bollinger Bands technical indicator is an example of a mean reversion strategy.

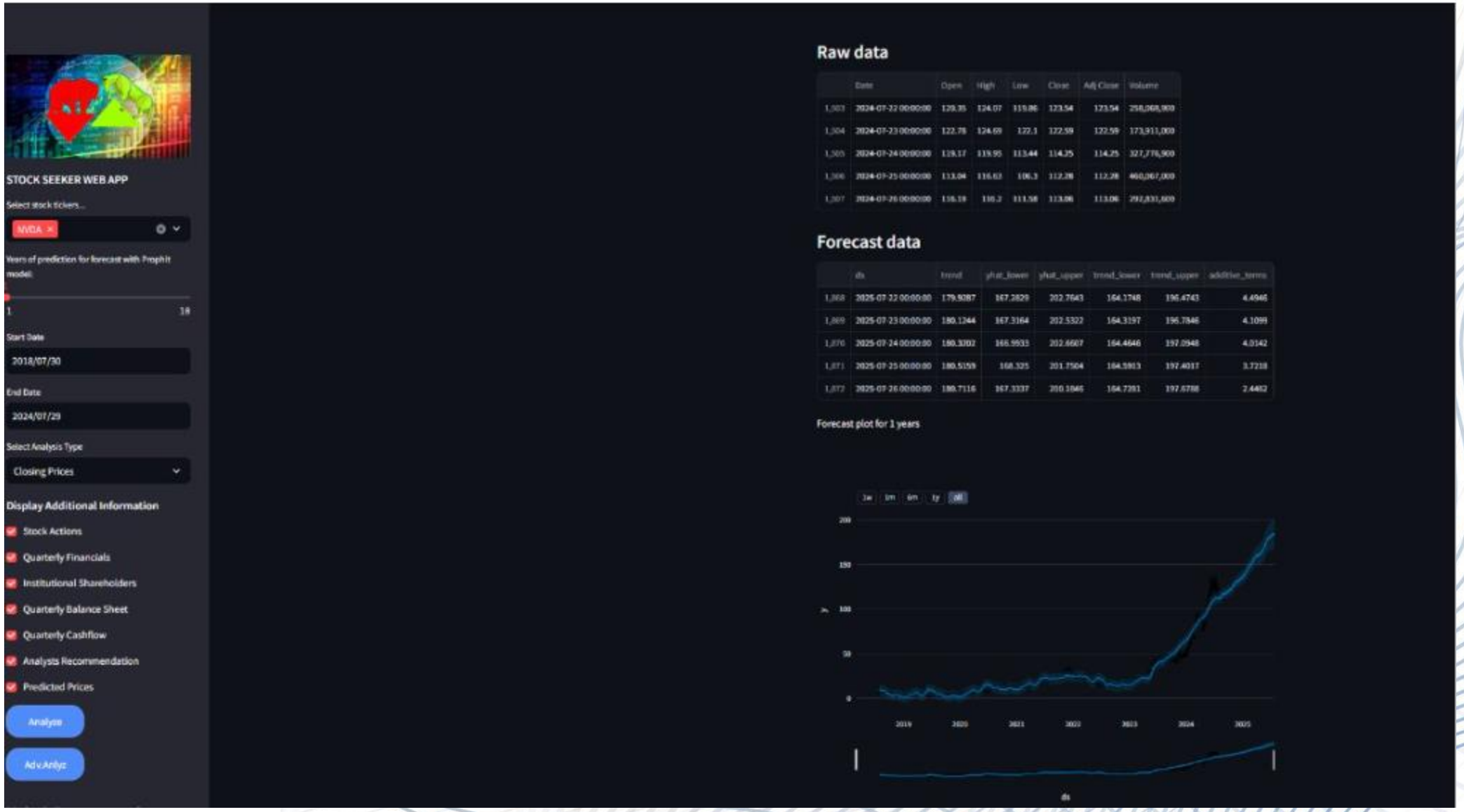
5.3.2 Mean reversion strategies

In mean reversion algorithmic trading strategies stocks return to their mean and we can exploit when it deviates from that mean.

These strategies usually involve selling into up moves and buying into down moves, a contrarian approach which assumes that the market has become oversold/overbought and prices will revert to their historical trends. This is almost the opposite of trend following where we enter in the direction of the strength and momentum, and momentum strategies such as buying stocks that have been showing an upward trend in hopes that the trend will continue, a continuation approach.

6. Conclusion

It is almost certainly better to choose technical indicators that complement each other, not just those that move in unison and generate the same signals. The intuition here is that the more indicators you have that confirm each other, the better your chances are to profit. This can be done by combining strategies to form a system, and looking for multiple signals.


STOCK SEEKER WEB APP
Select stock tickers...
NVDA
Years of prediction for forecast with Prophet model:
1
18
Start Date
2018/07/30
End Date
2024/07/29
Select Analysis Type
Closing Prices
Display Additional Information
Stock Actions
Quarterly Financials
Institutional Shareholders
Quarterly Balance Sheet
Quarterly Cashflow
Analysis Recommendation
Predicted Prices
Analyze
Adv Anlyz
Raw data
Date
Open
High
Low
Close
Adj Close
Volume
2024-07-22 00:00:00 129.35 124.07 119.06 123.54 123.54 258,068,900
2024-07-23 00:00:00 122.78 124.69 122.1 122.59 122.59 173,911,000
2024-07-24 00:00:00 119.17 119.95 113.44 114.25 114.25 327,776,900
2024-07-25 00:00:00 113.04 116.63 106.3 112.28 112.28 460,067,000
2024-07-26 00:00:00 116.19 110.2 111.58 113.06 113.06 292,831,600
Forecast data
ds
trend
yhat_lower
yhat_upper
trend_lower
trend_upper
additive_terms
2025-07-22 00:00:00 179.9087 167.2829 202.7643 164.1748 196.4743 4.4946
2025-07-23 00:00:00 180.1244 167.3164 202.5322 164.3197 196.7846 4.1099
2025-07-24 00:00:00 180.3302 166.9935 202.6607 164.4646 197.0948 4.0142
2025-07-25 00:00:00 180.5359 168.325 201.7504 164.5913 197.4017 3.7218
2025-07-26 00:00:00 180.7116 167.3337 200.1845 164.7281 197.6788 2.4402
Forecast plot for 1 years
1w
1m
6m
1y
all
200
150
100
50
0
2019
2020
2021
2022
2023
2024
2025
ds

STOCK SEEKER WEB APP

Select stock tickers...

NVDA ×

Years of prediction for forecast with Prophit model:

1 10

Start Date

2018/07/30

End Date

2024/07/29

Select Analysis Type

Closing Prices

Display Additional Information

- Stock Actions
- Quarterly Financials
- Institutional Shareholders
- Quarterly Balance Sheet
- Quarterly Cashflow
- Analysts Recommendation
- Predicted Prices

Analyze

Net Business Purchase And Sale	174,000,000	0	0
Purchase Of Business	-174,000,000	0	0
Net PPE Purchase And Sale	-369,000,000	-254,000,000	-278,000,000
Purchase Of PPE	-369,000,000	-254,000,000	-278,000,000

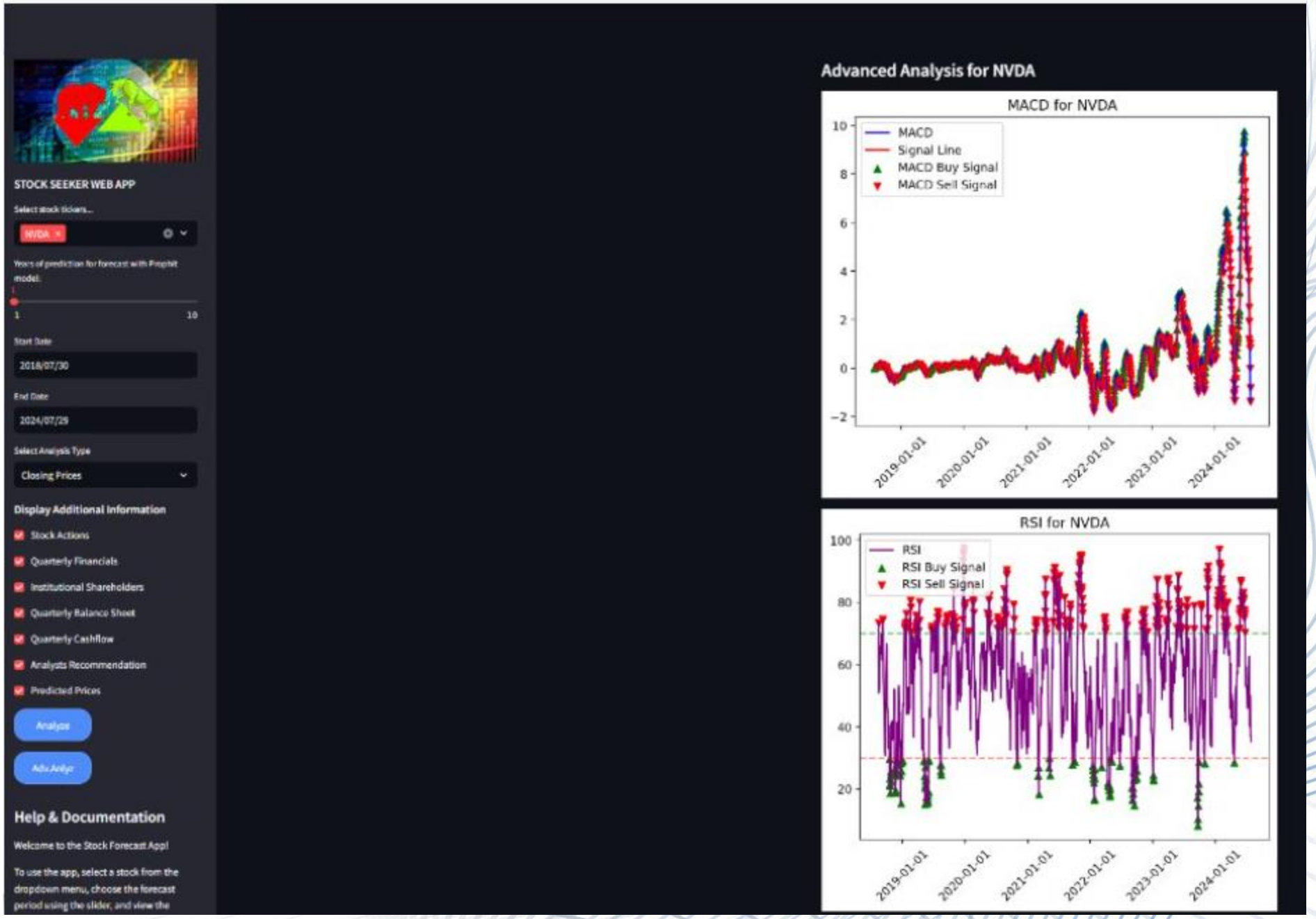
NVDA - Analysts Recommendation

	period	strongBuy	buy	hold	sell	strongSell
0	0m	7	14	15	1	1
1	-1m	19	34	5	0	0
2	-2m	20	33	4	0	0
3	-3m	12	21	12	2	0

NVDA - Predicted Prices

NVDA - Predicted Prices

NVDA Predicted Prices


STOCK SEEKER WEB APP
Select stock tickers...
NVDA
Years of prediction for forecast with Prophet model:
1
1
10
Start Date
2018/07/30
End Date
2024/07/29
Select Analysis Type
Closing Prices
Display Additional Information
Stock Actions
Quarterly Financials
Institutional Shareholders
Quarterly Balance Sheet
Quarterly Cashflow
Analysts Recommendation
Predicted Prices
Analyze
Adv.Analyz
Help & Documentation
Welcome to the Stock Forecast App!
To use the app, select a stock from the dropdown menu, choose the forecast period using the slider, and view the
Advanced Analysis for NVDA
MACD for NVDA
MACD
Signal Line
MACD Buy Signal
MACD Sell Signal
10
8
6
4
2
0
−2
2019-01-01
2020-01-01
2021-01-01
2022-01-01
2023-01-01
2024-01-01
RSI for NVDA
RSI
RSI Buy Signal
RSI Sell Signal
100
80
60
40
20
2019-01-01
2020-01-01
2021-01-01
2022-01-01
2023-01-01
2024-01-01

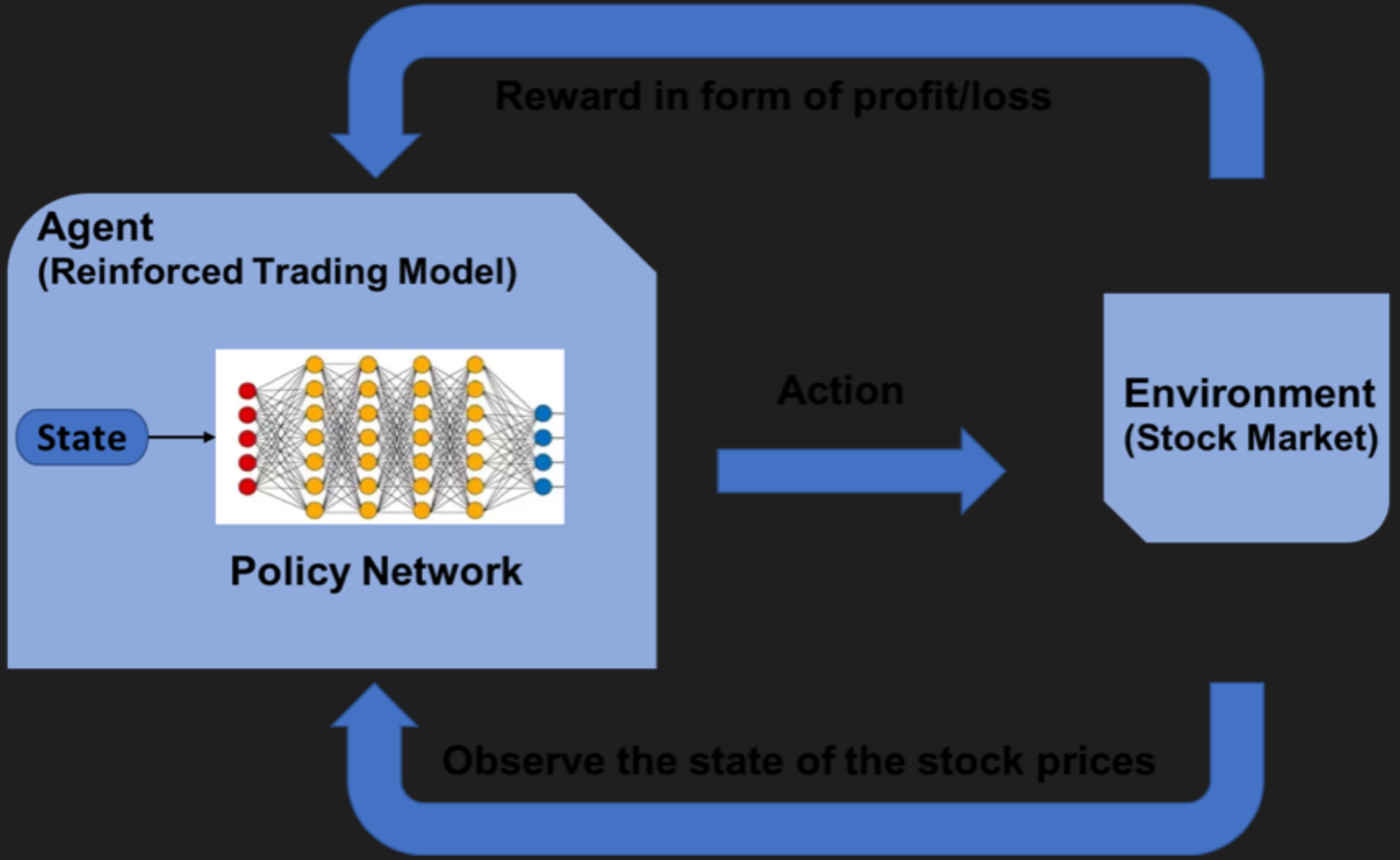
Reinforcement Learning

- State
- Take action

- Reward
- New State

- State (Trading price and all the features you can derive from it)
- Take action(Buy Sell or hold)

- Reward (Value of the portfolio)
- New State(new trading price and the values of the features used)


Reward in form of profit/loss
Agent
(Reinforced Trading Model)
State
Policy Network
Action
Environment
(Stock Market)
Observe the state of the stock prices

Techniques used to determine the action.

01

Deep Q-Learning

02

SARSA

03

Q-Learning wit
RNN-LSTM

This program defines a reinforcement learning agent to trade stocks using historical data. The agent makes buy, sell, or hold decisions based on a policy (greedy, egreedy, or softmax). It uses Double Q-learning, SARSA, Q-Learning with LSTM for updating its knowledge and **simulates multiple trading sessions to evaluate its performance.**

Q-Learning with Softmax Policy

Q-Learning is an off-policy learning method where the agent learns the value of the optimal policy, irrespective of the actions taken. The agent updates its Q-values using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

In the softmax policy, the probability $P(a|s)$ of choosing action a in state s is given by:

$$P(a|s) = \frac{e^{\beta Q(s,a)}}{\sum_{a'} e^{\beta Q(s,a')}}$$

Implementation:

1. **Action Selection:** Calculate probabilities for each action using the softmax function.
2. **Q-Value Update:** Update Q-values using the Q-learning update rule.
3. **Exploration vs. Exploitation:** The softmax policy balances exploration and exploitation based on the β parameter.

SARSA with Softmax Policy

SARSA is an on-policy learning method where the agent learns the value of the policy it is currently following. The Q-values are updated using the action actually taken by the agent:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

In the softmax policy, the action selection follows the same process as in Q-learning:

$$P(a|s) = \frac{e^{\beta Q(s,a)}}{\sum_{a'} e^{\beta Q(s,a')}}$$

Implementation:

1. **Action Selection:** Calculate probabilities for each action using the softmax function.
2. **Q-Value Update:** Update Q-values using the SARSA update rule.
3. **Exploration vs. Exploitation:** The softmax policy balances exploration and exploitation based on the β parameter.

Comparison

1. **Learning Approach:**
    - Q-Learning: Off-policy, learns the optimal policy independently of the agent's actions.
    - SARSA: On-policy, learns the policy that the agent is currently following.
2. **Action Selection:**
    - Both methods use the softmax policy to select actions, which probabilistically favors actions with higher Q-values based on the β parameter.

Conclusion

Using the softmax policy with both Q-Learning and SARSA allows the agent to balance between exploring new actions and exploiting known profitable actions. The β parameter is crucial in this balance, where a higher β value leads to more greedy actions, and a lower β value promotes exploration.

lead to more exploitation (greediness), while lower β values increase exploration.

Conclusion

Using the softmax policy with both Q-Learning and SARSA allows the agent to balance between exploring new actions and exploiting known profitable actions. The β parameter is crucial in this balance, where a higher β value leads to more greedy actions, and a lower β value promotes exploration.

DDQN :

```
# Fetch NVDA data
ticker = 'NVDA'
data = yf.download(ticker, start='2018-01-01', end='2024-01-01', interval='1d')
data.to_csv('NVDA_data.csv')
print(data.head())
```

✓ 0.2s

```
[**********100%%**********]  1 of 1 completed
                 Open     High      Low    Close  Adj Close     Volume
Date
2018-01-02  4.89450  4.98750  4.86250  4.98375   4.930644  355616000
2018-01-03  5.10250  5.34250  5.09375  5.31175   5.255147  914704000
2018-01-04  5.39400  5.45125  5.31725  5.33975   5.282849  583268000
2018-01-05  5.35475  5.42275  5.27700  5.38500   5.327617  580124000
2018-01-08  5.51000  5.62500  5.46450  5.55000   5.490859  881216000
```

Double Deep Q-Network (DDQN) is an advanced reinforcement learning algorithm designed to address the overestimation bias often found in Q-learning. It is an extension of Deep Q-Network (DQN) that decouples the action selection and action evaluation to improve learning stability and performance.

Key Components of DDQN

1.Experience Replay: Stores past experiences (state, action, reward, next state, done) to break correlations between consecutive samples, making the learning process more stable.

2.Target Network: A separate network, whose parameters are periodically updated, to stabilize the training process by providing consistent Q-value targets.

3.Double Q-Learning Update: Uses two sets of parameters to decouple action selection and action evaluation, reducing overestimation bias.

DDQN :

DDQN Algorithm Steps

1.Initialize Networks: Initialize the Q-network (with weights θ\thetaθ) and the target network (with weights θ−\theta^-θ−).

2.Experience Replay Memory: Create a replay memory to store experiences.

3.Action Selection (Epsilon-Greedy): Select an action using the epsilon-greedy policy.

4.Store Transition: Store the experience in replay memory.

5.Sample Mini-batch: Sample a mini-batch of experiences from the replay memory.

6. **Compute Target Q-value:**

$$Q_{\text{target}} = r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta^-)$$

7. **Update Q-Network:** Perform a gradient descent step on the loss:

$$\text{Loss} = (Q(s, a; \theta) - Q_{\text{target}})^2$$

8. **Update Target Network:** Periodically update the target network parameters:

$$\theta^- = \theta$$

Comparison with Q-Learning and SARSA

Q-Learning

Q-Learning is an off-policy algorithm that updates the Q-values using the maximum possible Q-value of the next state. This can lead to overestimation of action values.

- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- **Exploration-Exploitation:** Typically uses an epsilon-greedy policy to balance exploration and exploitation.

DDQN

Double DQN addresses the overestimation bias of Q-Learning by decoupling action selection and action evaluation using two sets of parameters. This leads to more stable learning.

- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s', \text{argmax}_{a'} Q(s',a';\theta);\theta^{-}) - Q(s,a) \right]$$

- **Exploration-Exploitation:** The code example uses an epsilon-greedy policy, but it can be adapted to use a softmax policy as well.

SARSA

SARSA is an on-policy algorithm that updates the Q-values using the actual action taken by the agent. This makes it more conservative compared to Q-Learning.

- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

- **Exploration-Exploitation:** Uses the epsilon-greedy policy for action selection.

Comparison with Q-Learning and SARSA

Q-Learning

Q-Learning is an off-policy algorithm that updates the Q-values using the maximum possible Q-value of the next state. This can lead to overestimation of action values.

- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- **Exploration-Exploitation:** Typically uses an epsilon-greedy policy to balance exploration and exploitation.

DDQN

Double DQN addresses the overestimation bias of Q-Learning by decoupling action selection and action evaluation using two sets of parameters. This leads to more stable learning.

- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s', \text{argmax}_{a'} Q(s',a';\theta);\theta^{-}) - Q(s,a) \right]$$

- **Exploration-Exploitation:** The code example uses an epsilon-greedy policy, but it can be adapted to use a softmax policy as well.

Conclusion

- **Q-Learning:** Suitable for scenarios where exploration needs to be balanced with exploiting known good actions, but it might overestimate action values.
- **SARSA:** More conservative as it learns the policy the agent is currently following, making it safer in uncertain environments.
- **DDQN:** Combines the best of both worlds by reducing overestimation bias and providing more stable learning, making it effective for complex environments like trading.

SARSA

SARSA is an on-policy algorithm that updates the Q-values using the actual action taken by the agent. This makes it more conservative compared to Q-Learning.

- **Update Rule:**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

- **Exploration-Exploitation:** Uses the epsilon-greedy policy for action selection.

Q-Learning

- **Pros:**
- Simpler to implement.
- Efficient in environments with a relatively small state space.
- Good for environments where exploration is crucial.
- **Cons:**
- Prone to overestimation bias, which can lead to unstable learning.
- May not perform well in highly volatile and complex environments like stock markets.

SARSA

- **Pros:**
- More conservative, leading to potentially safer decisions.
- Less prone to overestimation bias compared to Q-Learning.
- Suitable for environments where following the actual policy is crucial.
- **Cons:**
- Can be too conservative, potentially missing out on high-reward opportunities.
- Like Q-Learning, may struggle with very large state spaces and complex dynamics.

Double Deep Q-Network (DDQN)

- **Pros:**
- Reduces overestimation bias, leading to more stable learning.
- Effective in complex environments with large state spaces, such as stock markets.
- Can leverage deep learning to model intricate patterns in stock data.
- Well-suited for high-dimensional state spaces and continuous action spaces.
- **Cons:**
- More complex to implement and requires more computational resources.
- Requires careful tuning of hyperparameters.

Recommendation for Stock Market Prediction

Double Deep Q-Network (DDQN) is generally the most suitable for stock market prediction among the three. Here's why:

1. **Complexity Handling:** The stock market is a highly complex and dynamic environment with large state spaces. DDQN, with its deep learning capabilities, can effectively model complex patterns and relationships in stock data.
2. **Stability:** DDQN's reduction of overestimation bias through the use of a target network and double Q-learning leads to more stable and reliable learning, which is crucial for the unpredictable nature of stock markets.
3. **Scalability:** DDQN can scale to handle the vast amount of data and features typically involved in stock market prediction, making it more robust for real-world applications.

3. Initialize Trading Environment

```python
from gym import Env
from gym.spaces import Box, Discrete

class TradingEnvironment(Env):
    def __init__(self, data, trading_cost_bps=1e-3):
        super(TradingEnvironment, self).__init__()
        self.data = data
        self.trading_cost_bps = trading_cost_bps
        self.action_space = Discrete(3)  # Buy, Hold, Sell
        self.observation_space = Box(low=-np.inf, high=np.inf, shape=(len(data.columns),)
        self.reset()

    def reset(self):
        self.current_step = 0
        self.done = False
        return self.data.iloc[self.current_step].values

    def step(self, action):
        self.current_step += 1
        if self.current_step >= len(self.data):
            self.done = True
        next_state = self.data.iloc[self.current_step].values
        reward = 0  # Define reward logic based on your trading strategy
        return next_state, reward, self.done, {}

    def render(self, mode='human'):
        pass

# Register environment
from gym.envs.registration import register

register(
    id='trading-v0',
    entry_point='__main__:TradingEnvironment',
    max_episode_steps=252  # Example: 252 trading days in a year
)
```

Python

6. Create DDQN Agent

```
tf.keras.backend.clear_session()

# Define hyperparameters
learning_rate = 0.001
gamma = 0.99
epsilon_start = 1.0
epsilon_end = 0.1
epsilon_decay_steps = 10000
epsilon_exponential_decay = 0.99
replay_capacity = 100000
architecture = [
    {'type': 'Dense', 'units': 64, 'activation': 'relu'},
    {'type': 'Dense', 'units': 64, 'activation': 'relu'},
    {'type': 'Dropout', 'rate': 0.1}
]
l2_reg = 1e-6
tau = 1e-3
batch_size = 64

ddqn = DDQNAgent(state_dim=state_dim,
                 num_actions=num_actions,
                 learning_rate=learning_rate,
                 gamma=gamma,
                 epsilon_start=epsilon_start,
                 epsilon_end=epsilon_end,
                 epsilon_decay_steps=epsilon_decay_steps,
                 epsilon_exponential_decay=epsilon_exponential_decay,
                 replay_capacity=replay_capacity,
                 architecture=architecture,
                 l2_reg=l2_reg,
                 tau=tau,
                 batch_size=batch_size,
                 data = data)

ddqn.online_network.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	448
dense_1 (Dense)	(None, 64)	4,160
dropout (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 3)	195

```
Total params: 4,803 (18.76 KB)

Trainable params: 4,803 (18.76 KB)

Non-trainable params: 0 (0.00 B)
```

```
otal_steps = 0
max_episodes = 1000

episode_time, navs, market_navs, diffs, episode_eps = [], [], [], [], []
```

```
class DDQNAgent:
    def __init__(self, alpha, gamma, epsilon, beta, data, assets, policy, window_size):
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.beta = beta
        self.data = data
        self.assets = assets
        self.policy = policy
        self.window_size = window_size
        self.memory = []  # Assuming a list for memory
        self.is_training = True  # Assuming a flag for training

    def epsilon_greedy_policy(self, state):
        # Dummy implementation for example purposes
        if np.random.rand() < self.epsilon:
            return np.random.choice(range(2))  # Assuming binary actions
        else:
            return np.argmax([0, 0])  # Placeholder for action selection

    def memorize_transition(self, state, action, reward, next_state, done):
        # Store the transition in memory
        self.memory.append((state, action, reward, next_state, done))

    def experience_replay(self):
        # Placeholder for experience replay logic
        pass

    def is_training_enabled(self):
        return self.is_training
```

```python
class TradingEnvironment:
    def __init__(self):
        # Dummy implementation of environment
        pass

    def reset(self):
        # Reset environment to initial state
        return np.zeros(10)  # Placeholder state

    def step(self, action):
        # Simulate taking an action in the environment
        next_state = np.zeros(10)  # Placeholder next state
        reward = 0.0  # Placeholder reward
        done = np.random.rand() < 0.1  # Randomly end episode
        return next_state, reward, done, {}

    def close(self):
        # Close the environment
        pass

    @property
    def state_dim(self):
        # Define the state dimension
        return 10  # Example dimension

    @property
    def env(self):
        # Example environment property
        class EnvSimulator:
            def result(self):
                # Placeholder result function
                import pandas as pd
                return pd.DataFrame({
                    'nav': [1],
                    'strategy_return': [0.01],
                    'market_nav': [1]
                })
        return type('Env', (), {'simulator': EnvSimulator()})()
```

```python
def track_results(episode, mean_nav, recent_nav, mean_market_nav, recent_market_nav, p
    print(f"Episode: {episode}")
    print(f"Mean NAV (last 100 episodes): {mean_nav}")
    print(f"Recent NAV (last 10 episodes): {recent_nav}")
    print(f"Mean Market NAV (last 100 episodes): {mean_market_nav}")
    print(f"Recent Market NAV (last 10 episodes): {recent_market_nav}")
    print(f"Positive Differences (last 100 episodes): {positive_diffs}")
    print(f"Elapsed Time: {elapsed_time:.2f} seconds")
    print(f"Epsilon: {epsilon}")

# Initialize parameters
max_episodes = 1000
max_episode_steps = 200

# Create environment and agent
trading_environment = TradingEnvironment()
ddqn = DDQNAgent(alpha=0.6, gamma=0.6, epsilon=0.2, beta=0.5, data=None, assets=1000,

# Obtain state dimension from environment
state_dim = trading_environment.state_dim
navs, market_navs, diffs = [], [], []

start = time()
```

```python
for episode in range(1, max_episodes + 1):
    this_state = trading_environment.reset()
    for episode_step in range(max_episode_steps):
        action = ddqn.epsilon_greedy_policy(this_state.reshape(-1, state_dim))
        next_state, reward, done, _ = trading_environment.step(action)

        ddqn.memorize_transition(this_state, action, reward, next_state, 0.0 if done else

        if ddqn.is_training_enabled():
            ddqn.experience_replay()

        if done:
            break
        this_state = next_state

    # Get and store results
    result = trading_environment.env.simulator.result()
    final = result.iloc[-1]
    nav = final.nav * (1 + final.strategy_return)
    navs.append(nav)
    market_nav = final.market_nav
    market_navs.append(market_nav)
    diff = nav - market_nav
    diffs.append(diff)

    if episode % 10 == 0:
        track_results(
            episode,
            np.mean(navs[-100:]),
            np.mean(navs[-10:]),
            np.mean(market_navs[-100:]),
            np.mean(market_navs[-10:]),
            np.sum([s > 0 for s in diffs[-100:]])/min(len(diffs), 100),
            time() - start,
            ddqn.epsilon
        )
    if len(diffs) > 25 and all([r > 0 for r in diffs[-25:]]):
        print(result.tail())
        break

trading_environment.close()
```

```
Episode: 10
Mean NAV (last 100 episodes): 1.01
Recent NAV (last 10 episodes): 1.01
Mean Market NAV (last 100 episodes): 1.0
Recent Market NAV (last 10 episodes): 1.0
Positive Differences (last 100 episodes): 1.0
Elapsed Time: 0.01 seconds
Epsilon: 0.2
Episode: 20
Mean NAV (last 100 episodes): 1.0100000000000002
Recent NAV (last 10 episodes): 1.01
Mean Market NAV (last 100 episodes): 1.0
Recent Market NAV (last 10 episodes): 1.0
Positive Differences (last 100 episodes): 1.0
Elapsed Time: 0.01 seconds
Epsilon: 0.2
   nav  strategy_return  market_nav
0    1             0.01           1
```

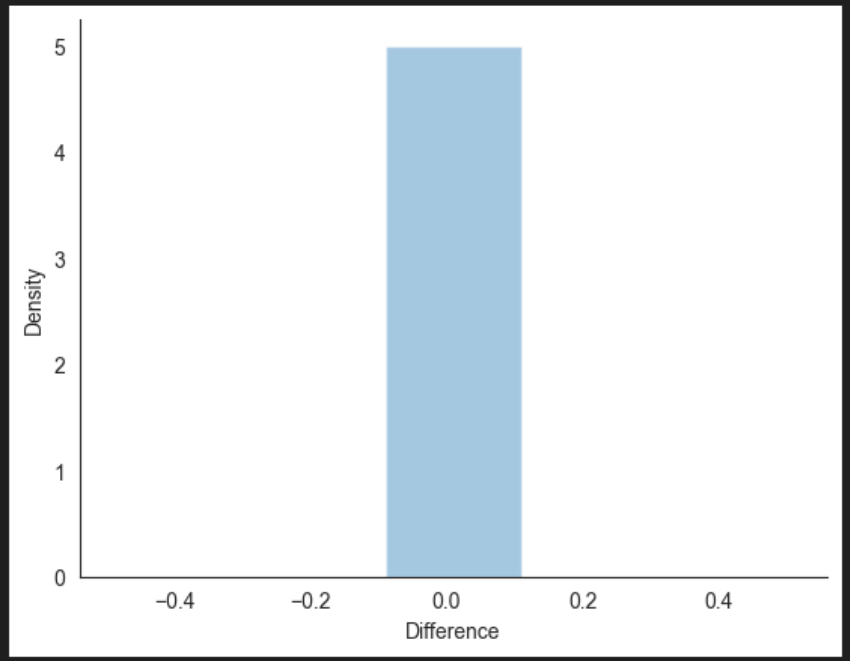
```python
    results = pd.DataFrame({'Episode': list(range(1, episode+1)),
                            'Agent': navs,
                            'Market': market_navs,
                            'Difference': diffs}).set_index('Episode')

    results['Strategy Wins (%)'] = (results.Difference > 0).rolling(100).sum()
    results.info()
    results.to_csv(results_path / 'results.csv', index=False)

    with sns.axes_style('white'):
        sns.distplot(results.Difference)
        sns.despine()
```

Python

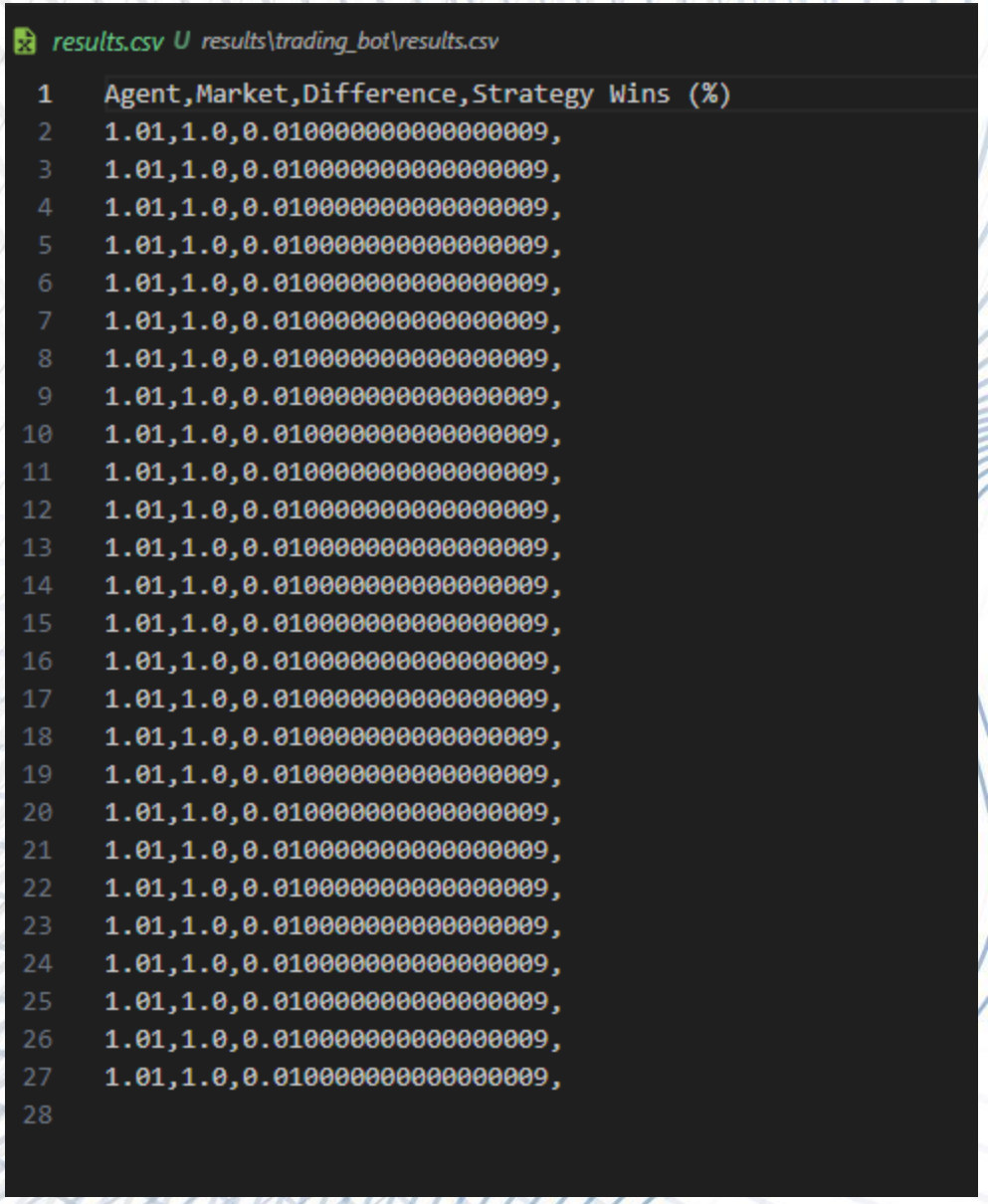
```
<class 'pandas.core.frame.DataFrame'>
Index: 26 entries, 1 to 26
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Agent              26 non-null     float64
 1   Market             26 non-null     float64
 2   Difference         26 non-null     float64
 3   Strategy Wins (%)  0 non-null      float64
dtypes: float64(4)
memory usage: 1.0 KB
```



NAV (Net Asset Value)

CSV File Analysis Headers

- **Episode**: Episode number
- **Mean_NAV_100**: Average Net Asset Value (NAV) over the last 100 episodes
- **Mean_NAV_10**: Average NAV over the last 10 episodes
- **Mean_Market_NAV_100**: Average Market NAV over the last 100 episodes
- **Mean_Market_NAV_10**: Average Market NAV over the last 10 episodes
- **Win_Ratio_100**: Win ratio over the last 100 episodes
- **Elapsed_Time**: Time elapsed
- **Epsilon**: Exploration rate



Data Section

- **Agent**: NAV value or performance of the agent
- **Market**: NAV value or performance of the market
- **Difference**: Difference between the agent's NAV and the market NAV
- **Strategy Wins (%)**: Percentage of wins for the strategy

results.csv U results\trading_bot\results.csv

```
Agent,Market,Difference,Strategy Wins (%)
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
1.01,1.0,0.010000000000000009,
```

ddqn_trading_results.csv U results\trading_bot\ddqn_trading_results.csv

```
Episode,Mean_NAV_100,Mean_NAV_10,Mean_Market_NAV_100,Mean_Market_NAV_10,Win_Ratio_100,Elapsed_Time,Epsilon
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
,,,,,,,
```

Summary

- **Header Information**: The CSV file tracks various performance metrics for episodes of a trading strategy.
- **Empty Rows**: Empty rows indicate missing or incomplete data. This might be due to issues in data collection or recording.
- **Data Insights**: The data shows small differences between the agent's performance and the market performance, indicating minimal discrepancy in strategy effectiveness.

SARSA
Stock Trend and Actions

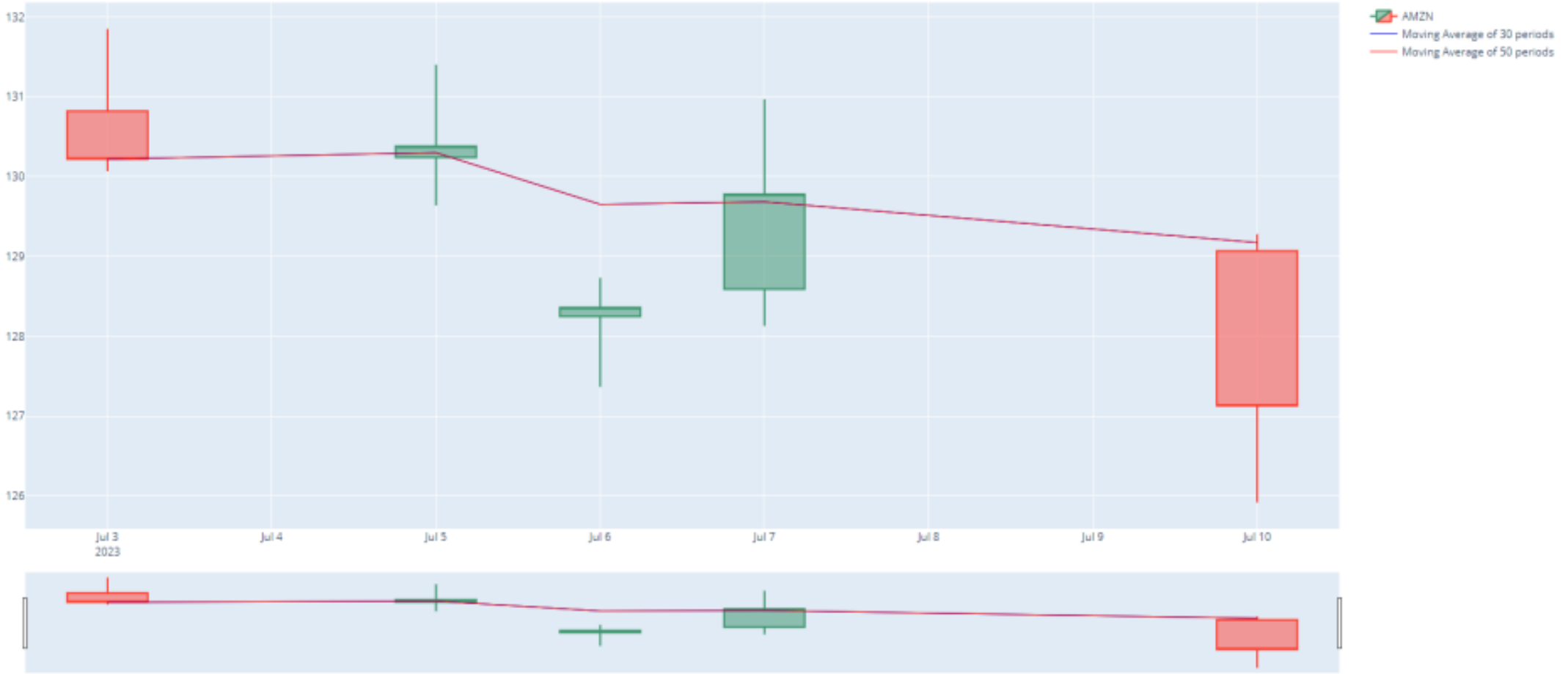
```
def step(self, action):
    self.previous_inventory = self.inventory.copy()
    if action == "buy":
        self.buy()
    elif action == "sell" and self.inventory and self.inventory['AMZN'] >
        self.sell()
    else:
        self.hold()
    while True:
        self.curr_date += pd.Timedelta(days=1)
        if self.check_date_validity():
            break
        else:
            self.skipped_days += 1
```

SARSA
Stock Trend and Actions

```
def run_policy(self, previous_stock_price):
    action = "hold"
    self.punish = False
    if self.policy == "greedy" or self.policy == "egreedy":
        if np.random.random() < self.epsilon:
            action = np.random.choice(self.actions, size=1)[0]
        else:
            choices = np.array(self.q_table[self.stock_history_idx(), :,
            action = str(np.random.choice(self.actions[choices.flatten()]
    elif self.policy == 'softmax':
        pvals = np.exp(self.beta * self.q_table[self.stock_history_idx(),
        action = np.random.choice(self.actions, size=1, p=pvals.flatten()


    if (action == "sell" and (n self.inventory or self.inventory['AMZN']
        self.punish = True
```

SARSA


AMZN
Moving Average of 30 periods
Moving Average of 50 periods
132
131
130
129
128
127
126
Jul 3
2023
Jul 4
Jul 5
Jul 6
Jul 7
Jul 8
Jul 9
Jul 10

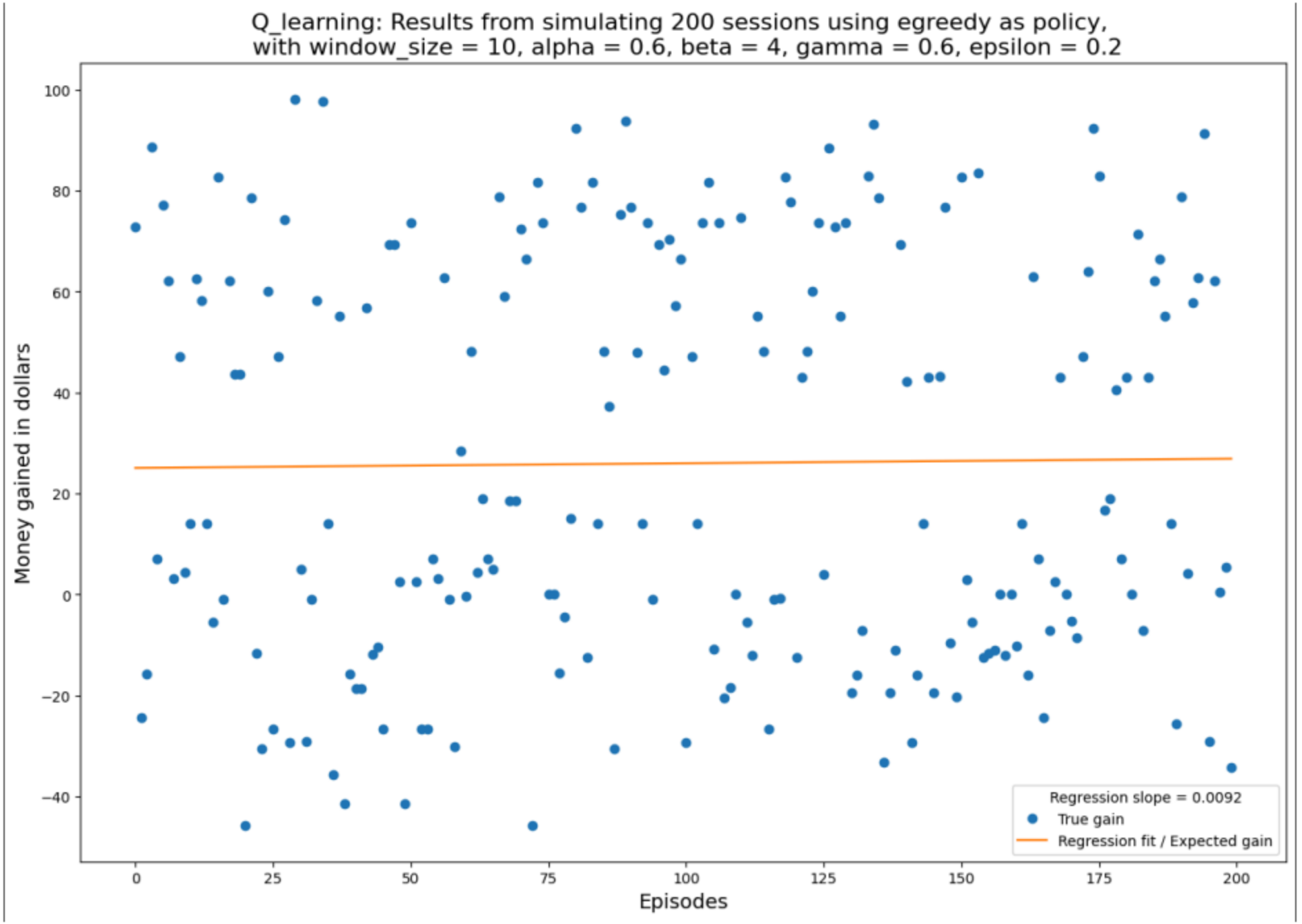
SARSA

Date	Open	High	Low	Close	Adj Close	Volume
2023-07-03	130.820007	131.850006	130.070007	130.220001	130.220001	28264800
2023-07-05	130.240005	131.399994	129.639999	130.380005	130.380005	35895400
2023-07-06	128.250000	128.729996	127.370003	128.360001	128.360001	40639900
2023-07-07	128.589996	130.970001	128.130005	129.779999	129.779999	41928700
2023-07-10	129.070007	129.279999	125.919998	127.129997	127.129997	61889300

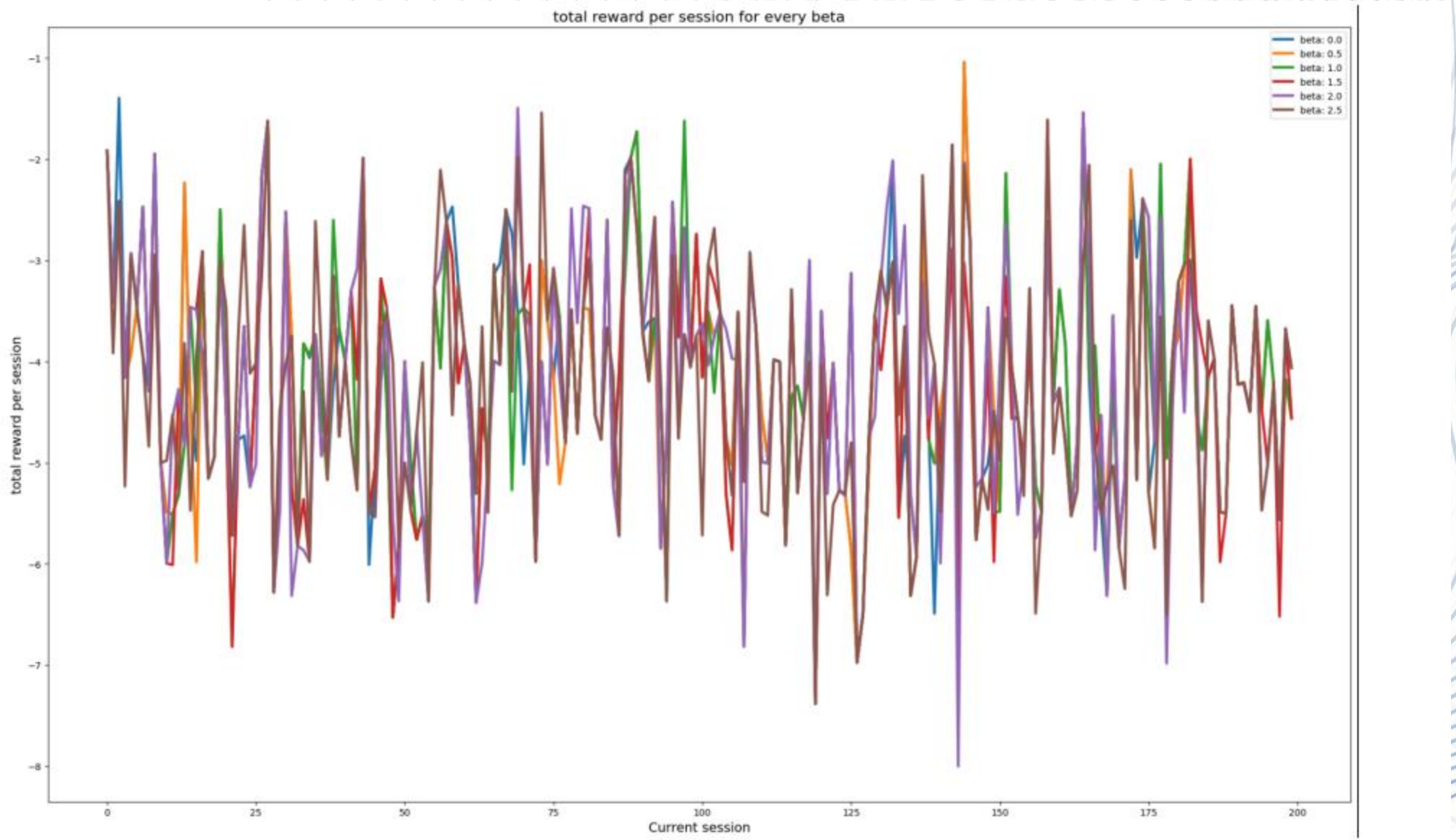
```
# Create RL_AGENT
rl_agent = RL_Agent(alpha = 0.6, gamma = 0.6, epsilon = 0.2, beta=4, sarsa=False, data=data_window, assets=1000 , policy="egreedy", window_size = 10)
# The line below can be used to peform a simulation
rewards, gains, m, b = rl_agent.simulate()
```

✓ 4.4s

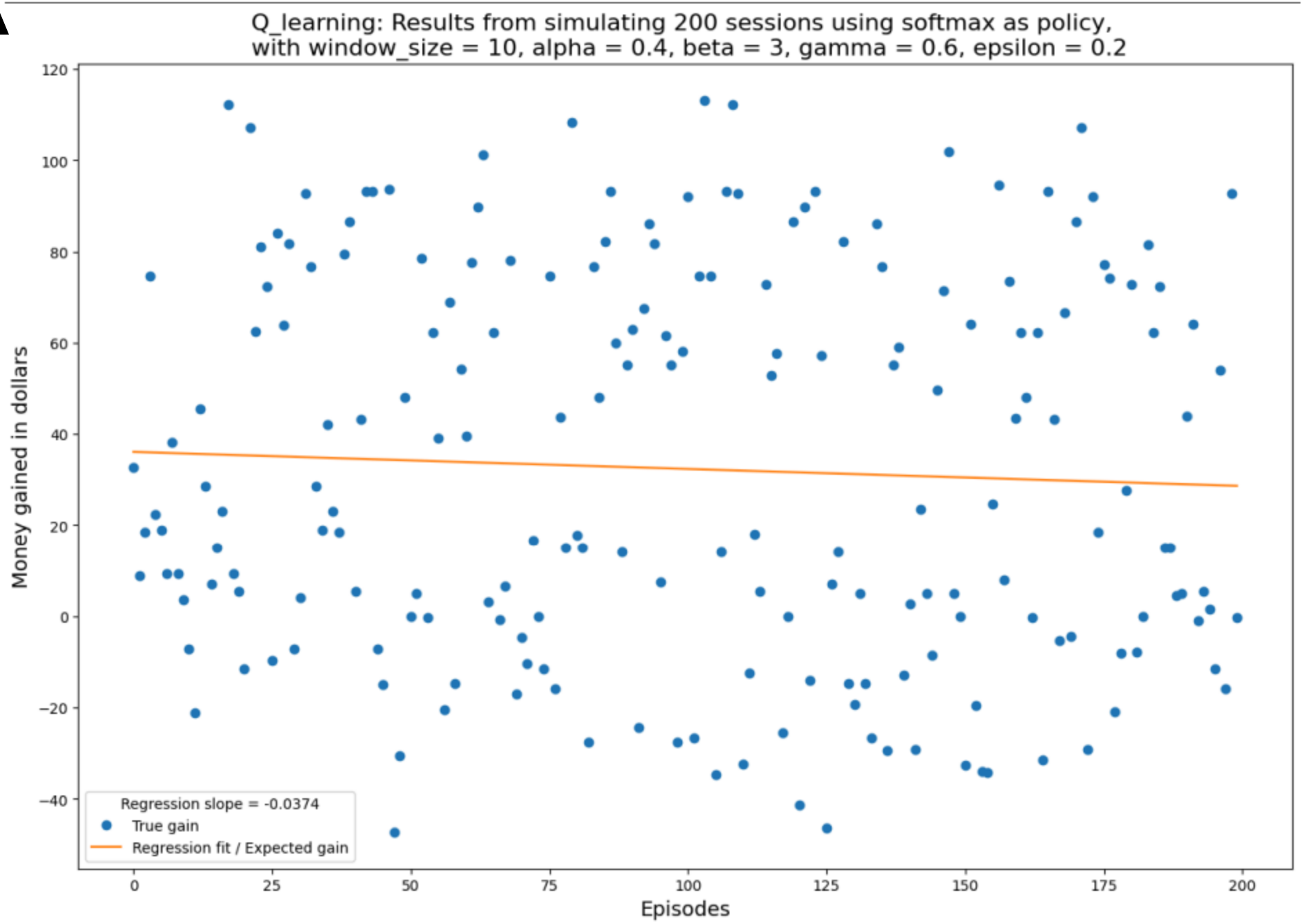
SARSA


Q_learning: Results from simulating 200 sessions using egreedy as policy,
with window_size = 10, alpha = 0.6, beta = 4, gamma = 0.6, epsilon = 0.2
Money gained in dollars
Episodes
100
80
60
40
20
0
−20
−40
0
25
50
75
100
125
150
175
200
Regression slope = 0.0092
True gain
Regression fit / Expected gain

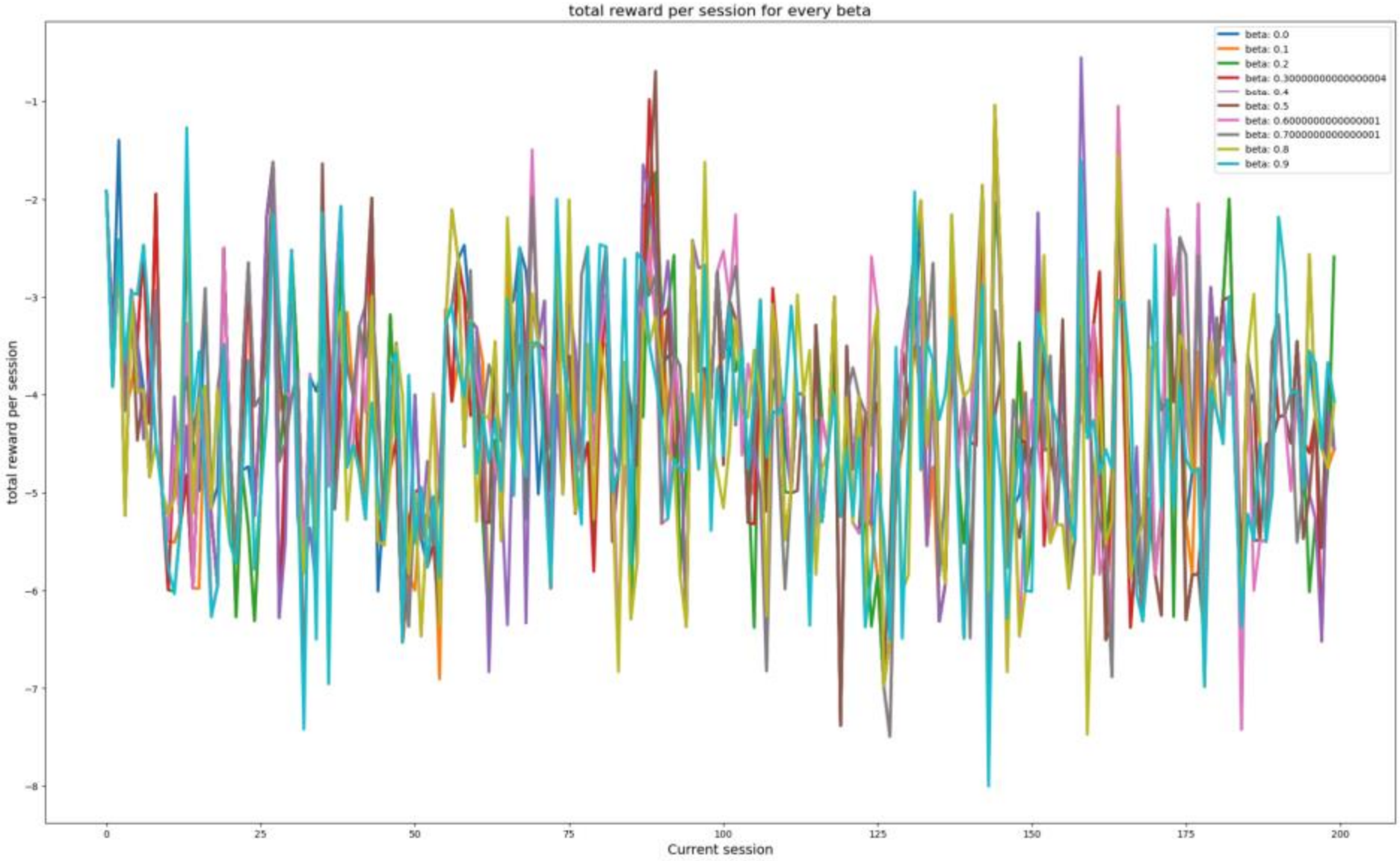
SARSA


total reward per session for every beta
beta: 0.0
beta: 0.5
beta: 1.0
beta: 1.5
beta: 2.0
beta: 2.5
total reward per session
Current session
−1
−2
−3
−4
−5
−6
−7
−8
0
25
50
75
100
125
150
175
200

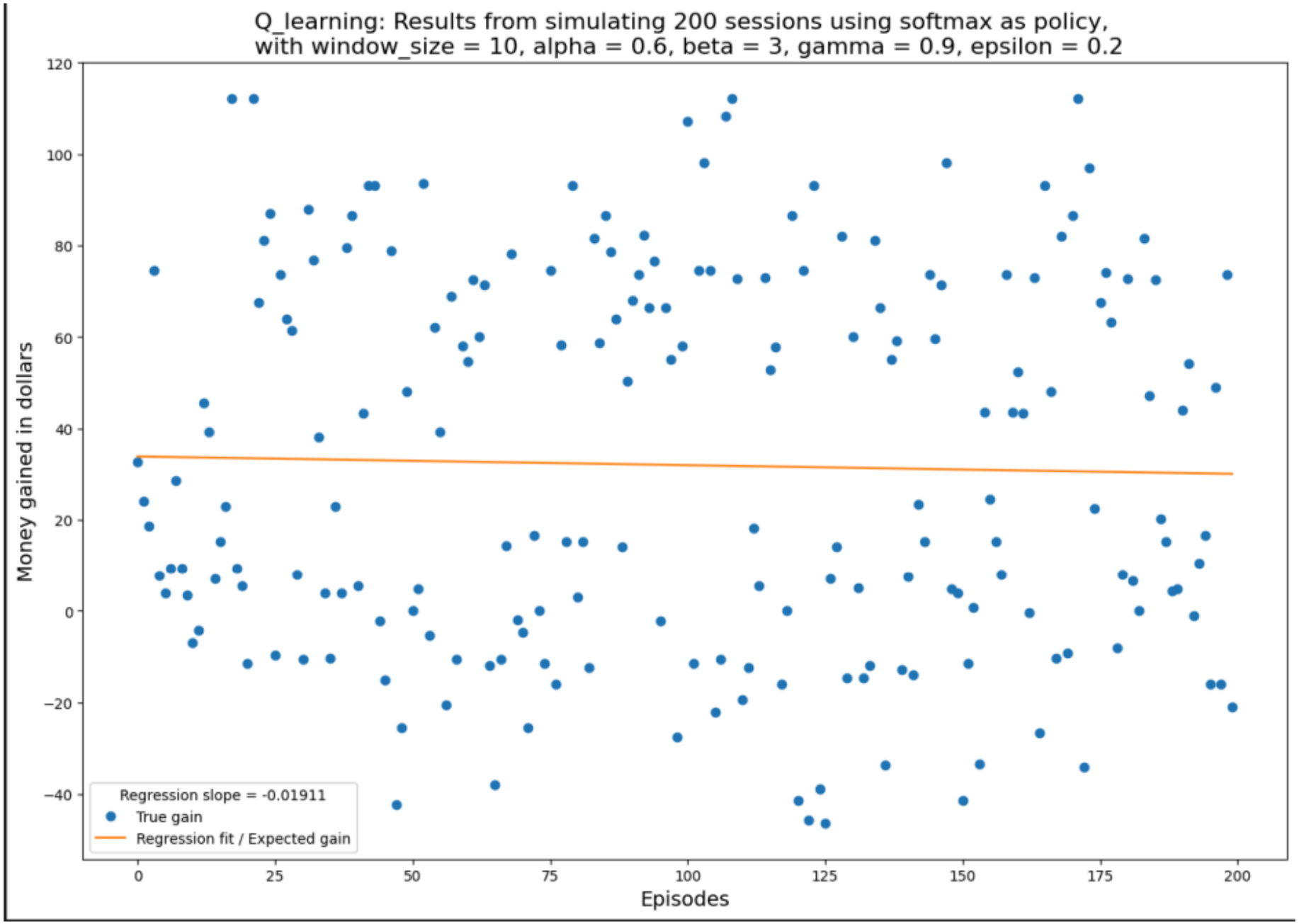
SARSA


Q_learning: Results from simulating 200 sessions using softmax as policy,
with window_size = 10, alpha = 0.4, beta = 3, gamma = 0.6, epsilon = 0.2
Money gained in dollars
Episodes
Regression slope = -0.0374
True gain
Regression fit / Expected gain
120
100
80
60
40
20
0
−20
−40
0
25
50
75
100
125
150
175
200

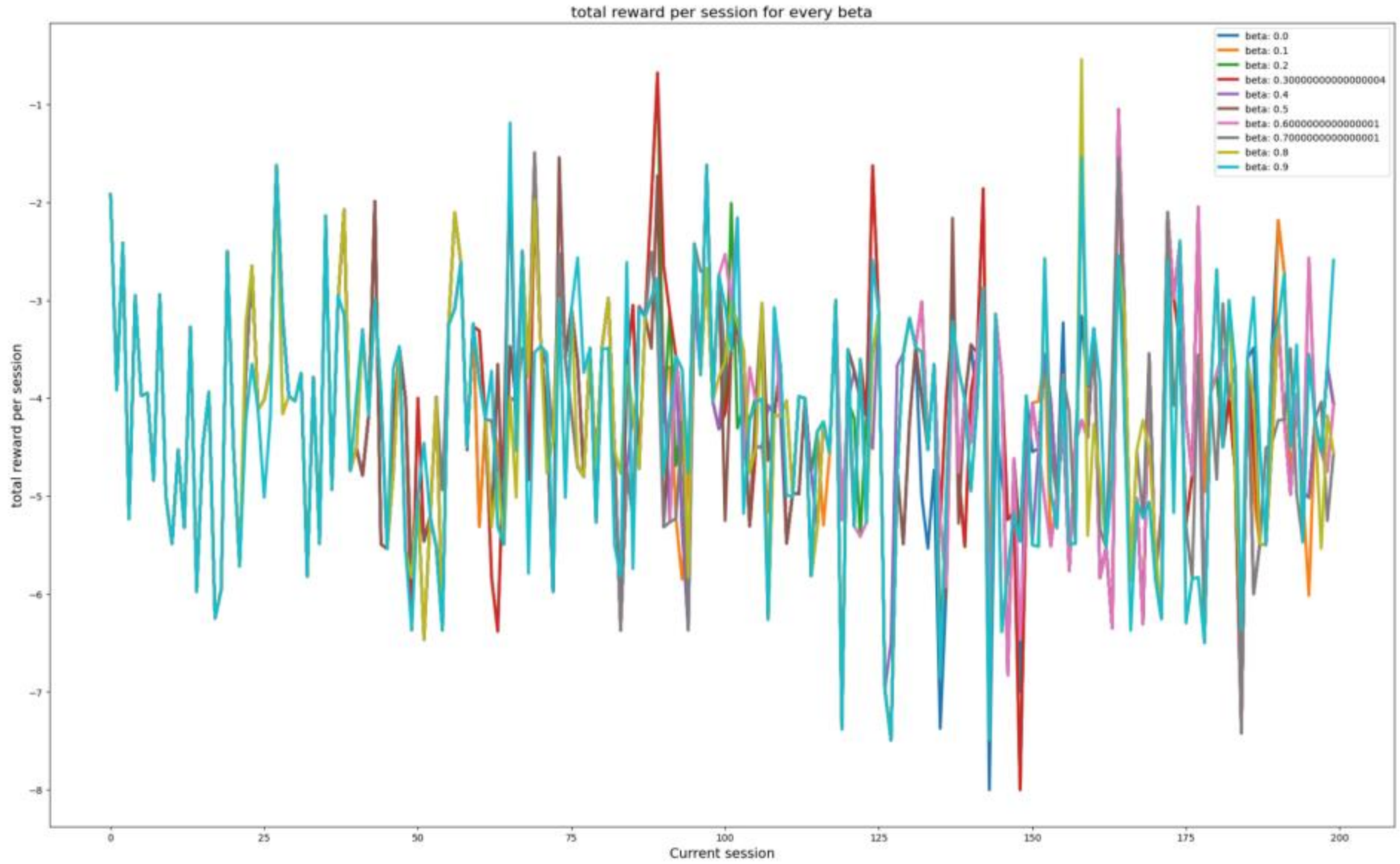
SARSA


total reward per session for every beta
beta: 0.0
beta: 0.1
beta: 0.2
beta: 0.30000000000000004
beta: 0.4
beta: 0.5
beta: 0.6000000000000001
beta: 0.7000000000000001
beta: 0.8
beta: 0.9
total reward per session
Current session
−1
−2
−3
−4
−5
−6
−7
−8
0
25
50
75
100
125
150
175
200

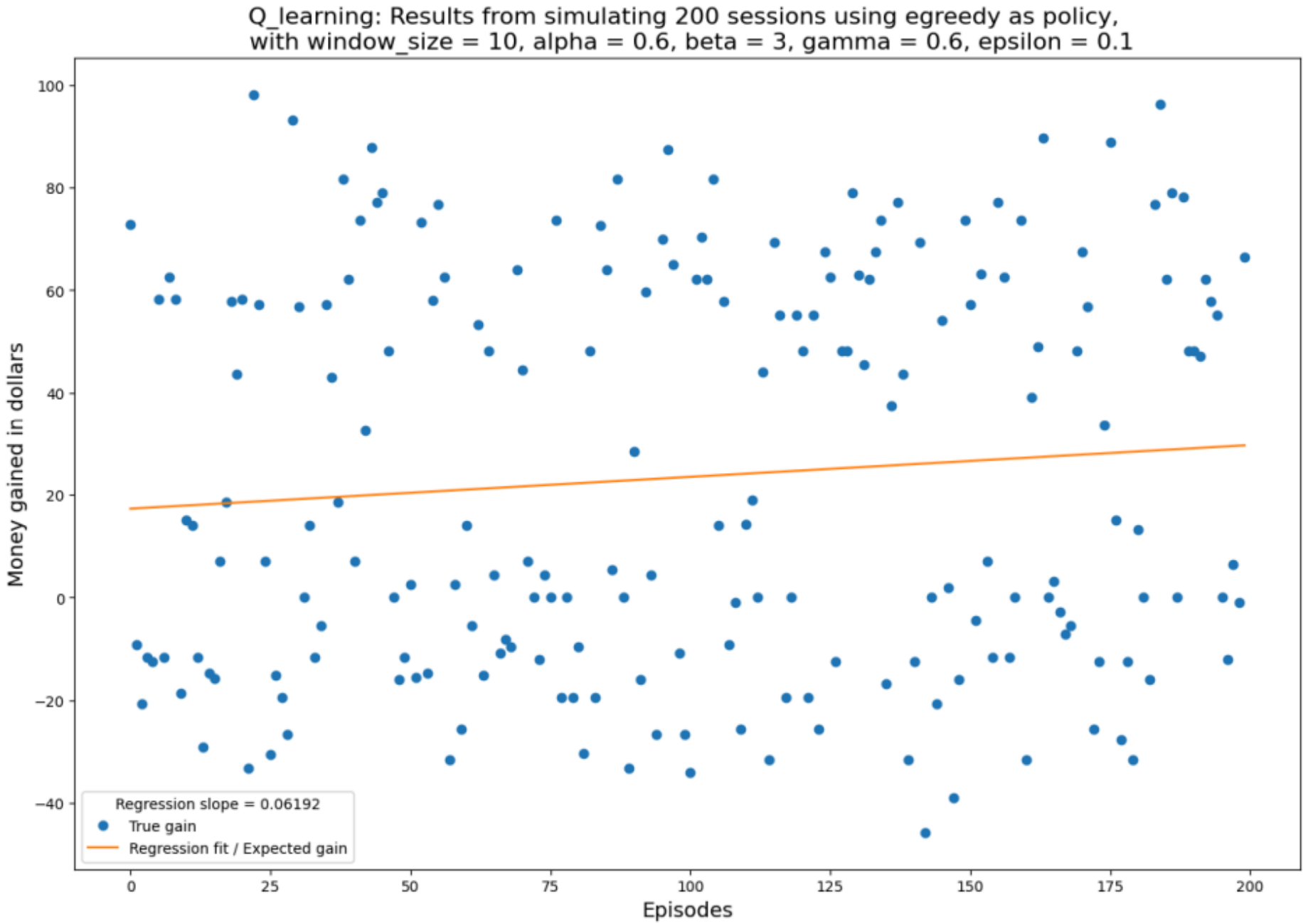
SARSA


Q_learning: Results from simulating 200 sessions using softmax as policy,
with window_size = 10, alpha = 0.6, beta = 3, gamma = 0.9, epsilon = 0.2
Money gained in dollars
Episodes
120
100
80
60
40
20
0
−20
−40
0
25
50
75
100
125
150
175
200
Regression slope = -0.01911
True gain
Regression fit / Expected gain

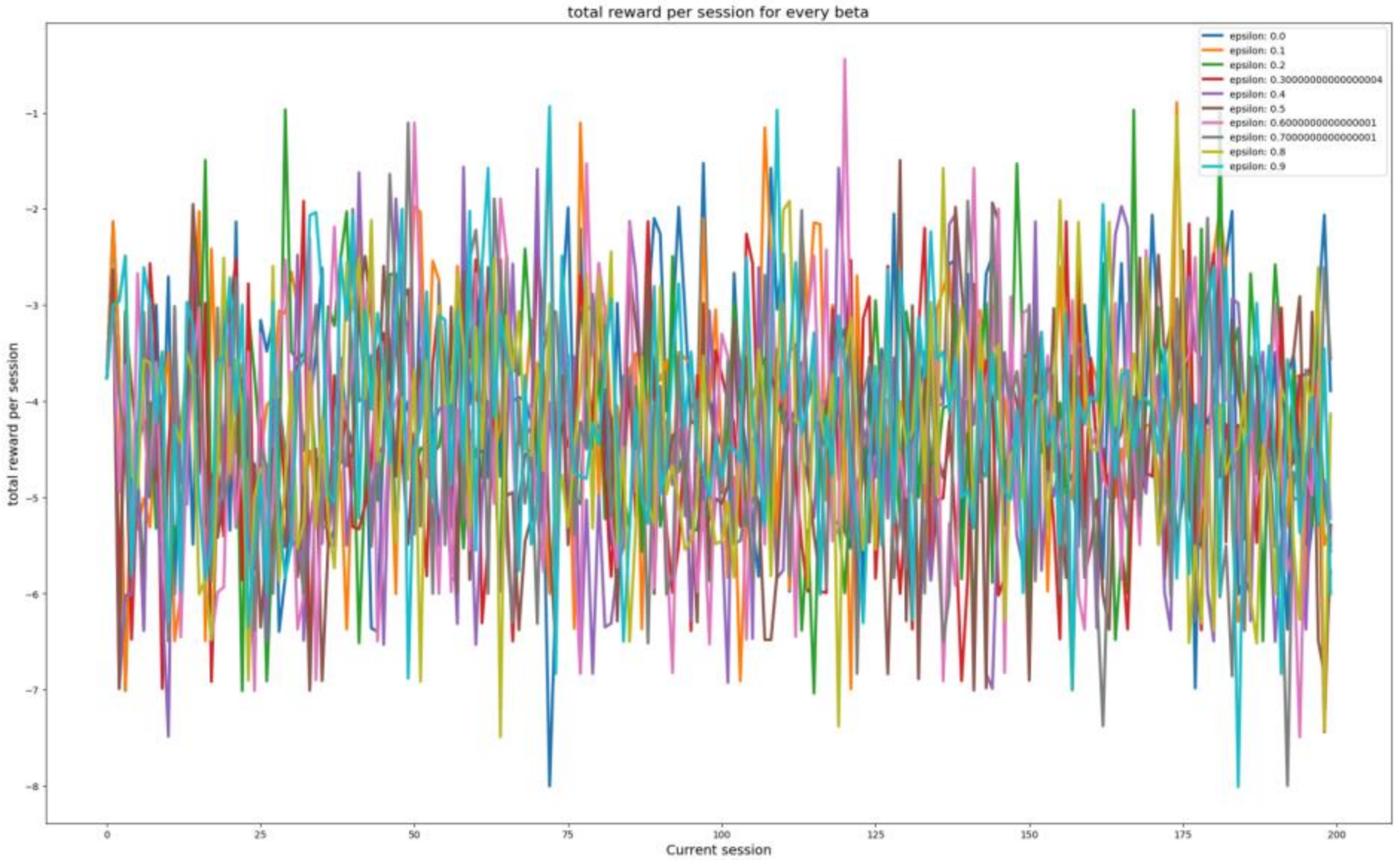
SARSA


total reward per session for every beta
beta: 0.0
beta: 0.1
beta: 0.2
beta: 0.30000000000000004
beta: 0.4
beta: 0.5
beta: 0.6000000000000001
beta: 0.7000000000000001
beta: 0.8
beta: 0.9
total reward per session
Current session
−1
−2
−3
−4
−5
−6
−7
−8
0
25
50
75
100
125
150
175
200

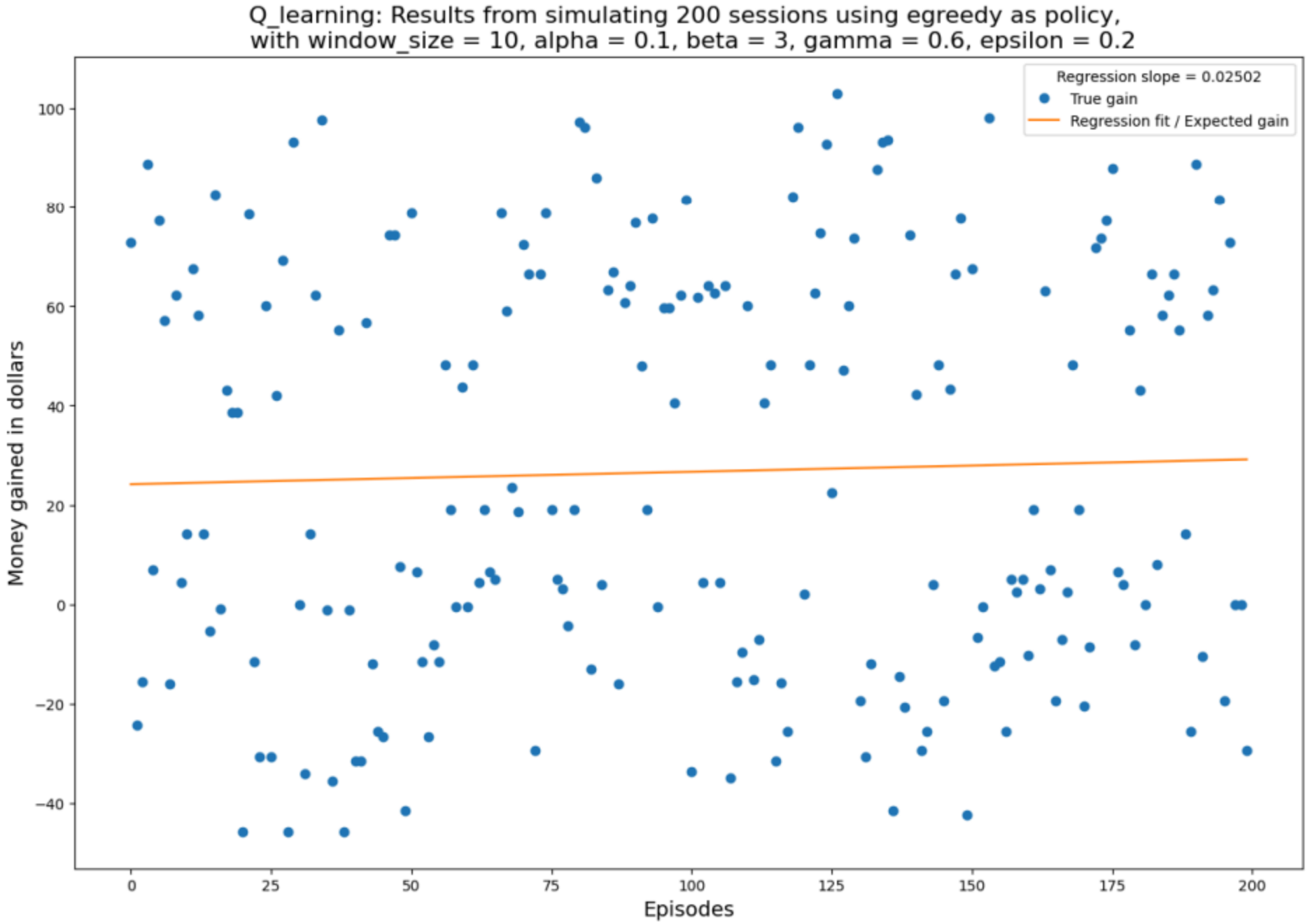
SARSA


Q_learning: Results from simulating 200 sessions using egreedy as policy,
with window_size = 10, alpha = 0.6, beta = 3, gamma = 0.6, epsilon = 0.1
Money gained in dollars
Episodes
Regression slope = 0.06192
True gain
Regression fit / Expected gain

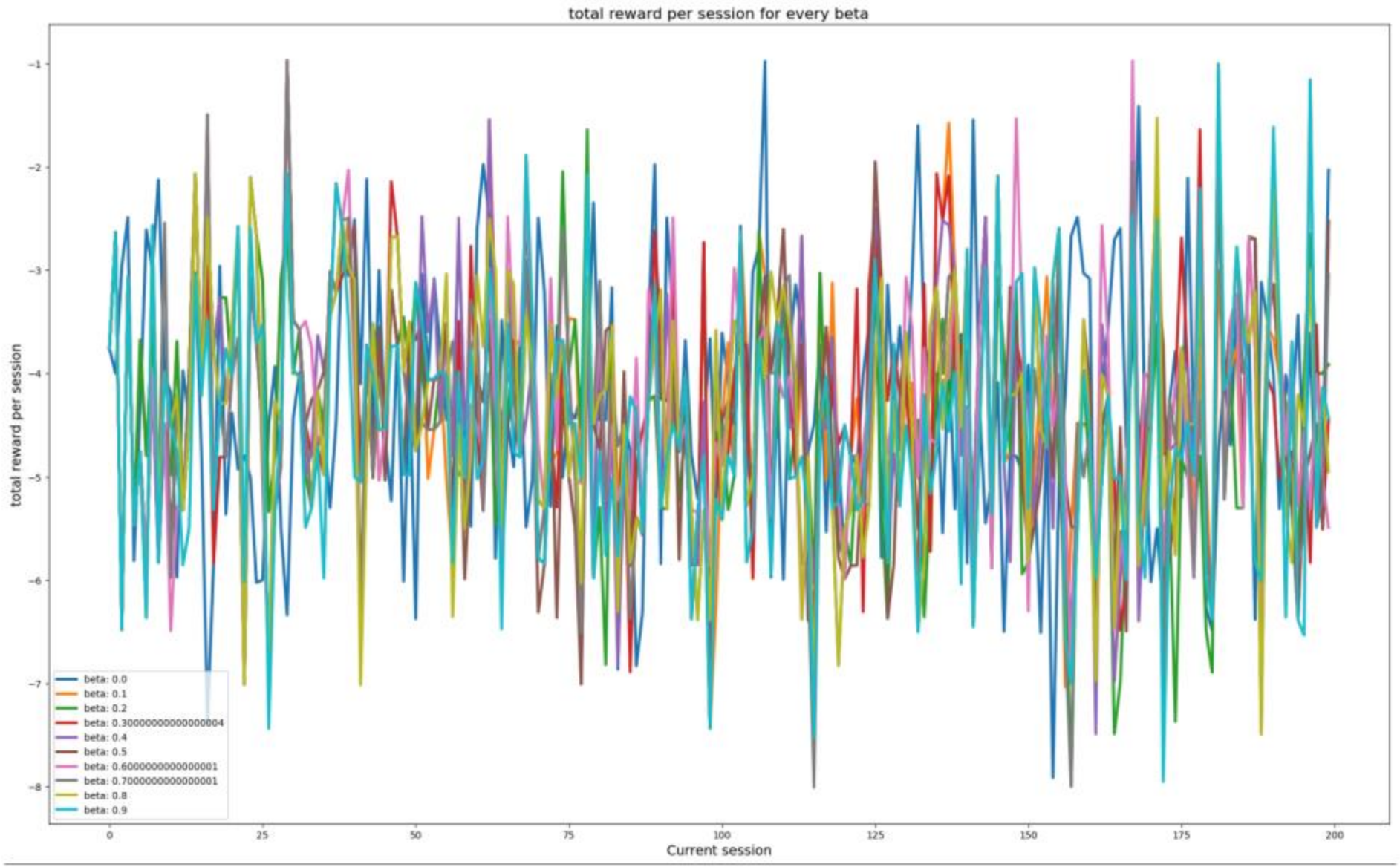
SARSA


total reward per session for every beta
epsilon: 0.0
epsilon: 0.1
epsilon: 0.2
epsilon: 0.30000000000000004
epsilon: 0.4
epsilon: 0.5
epsilon: 0.6000000000000001
epsilon: 0.7000000000000001
epsilon: 0.8
epsilon: 0.9
total reward per session
Current session
−1
−2
−3
−4
−5
−6
−7
−8
0
25
50
75
100
125
150
175
200

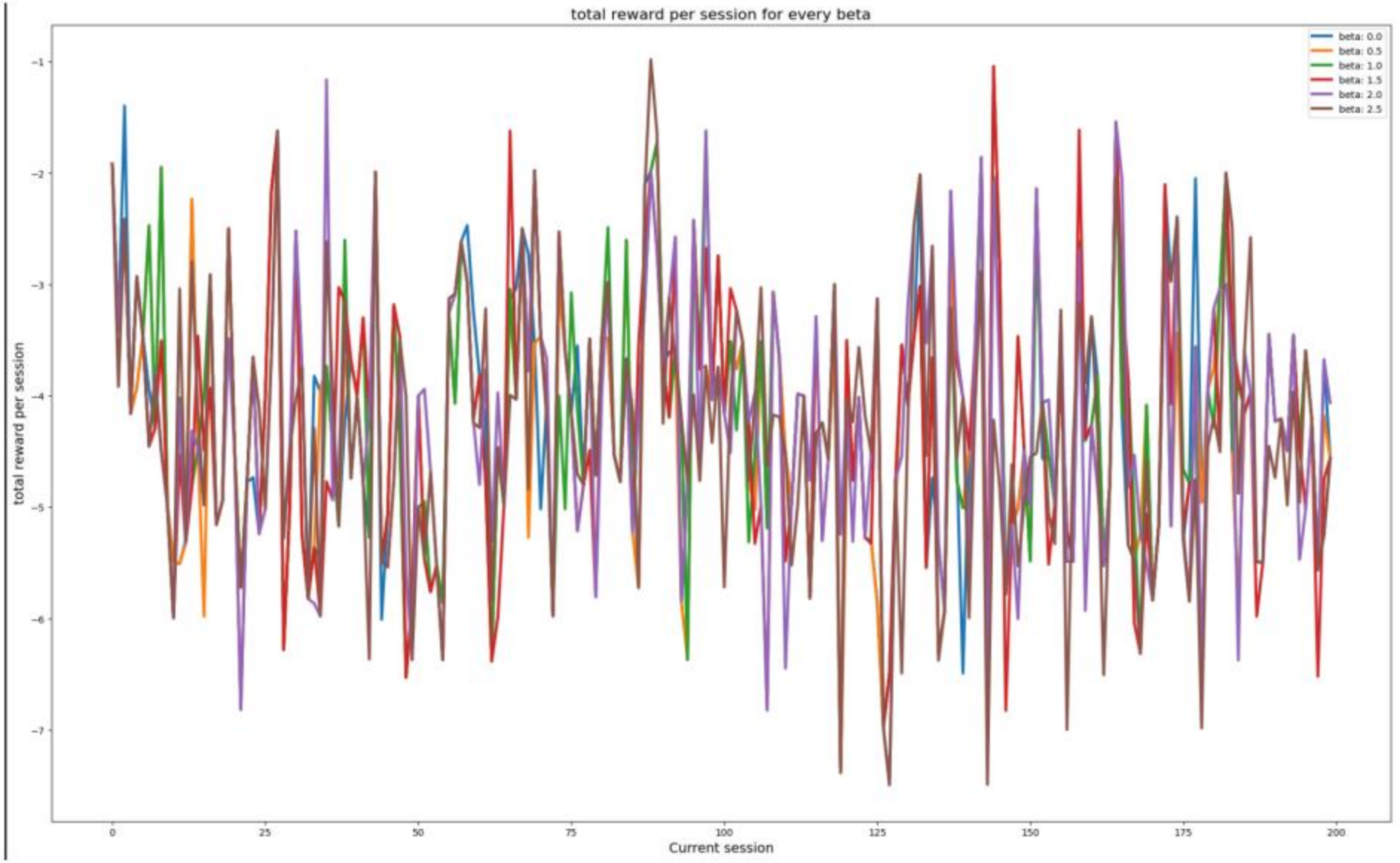
SARSA


Q_learning: Results from simulating 200 sessions using egreedy as policy,
with window_size = 10, alpha = 0.1, beta = 3, gamma = 0.6, epsilon = 0.2
Regression slope = 0.02502
True gain
Regression fit / Expected gain
Money gained in dollars
Episodes
100
80
60
40
20
0
−20
−40
0
25
50
75
100
125
150
175
200

SARSA


total reward per session for every beta
total reward per session
Current session
beta: 0.0
beta: 0.1
beta: 0.2
beta: 0.30000000000000004
beta: 0.4
beta: 0.5
beta: 0.6000000000000001
beta: 0.7000000000000001
beta: 0.8
beta: 0.9

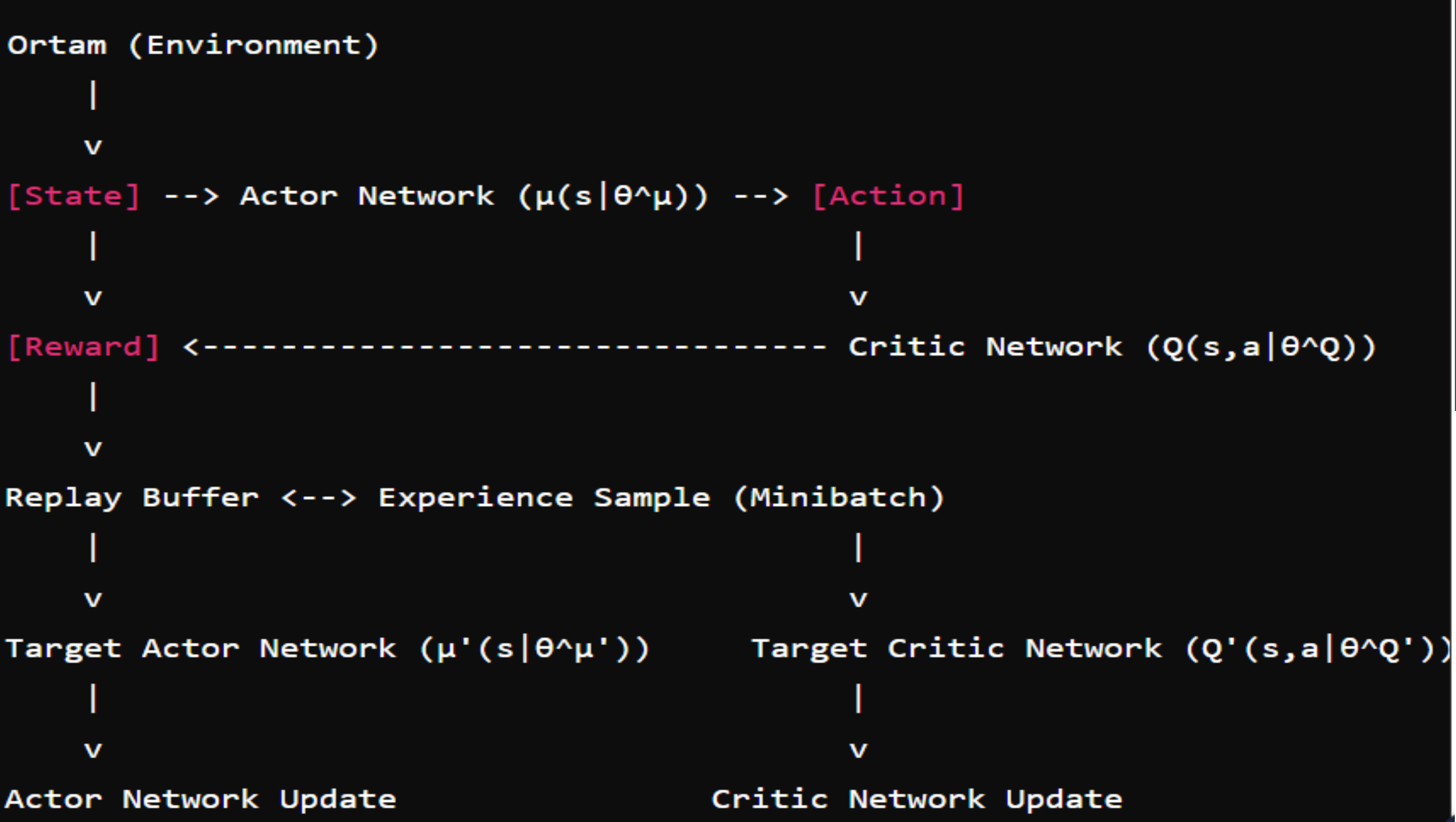
SARSA


total reward per session for every beta
beta: 0.0
beta: 0.5
beta: 1.0
beta: 1.5
beta: 2.0
beta: 2.5
total reward per session
Current session
−1
−2
−3
−4
−5
−6
−7
0
25
50
75
100
125
150
175
200

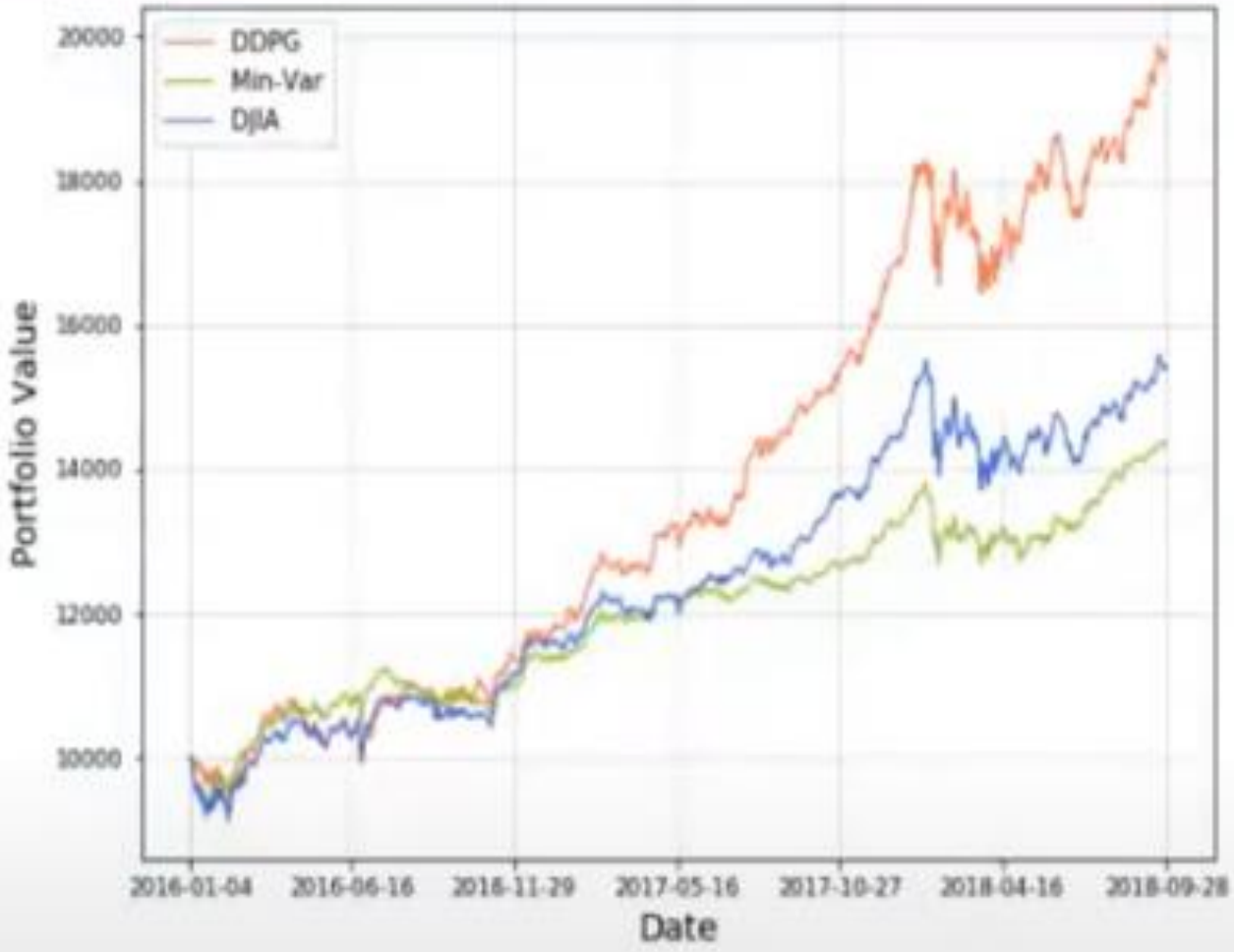
RL STOCK MARKET FORECAST:

DDPG (Deep Deterministic Policy Gradient) is an algorithm used in reinforcement learning for continuous action spaces. It combines the Actor-Critic method with deep learning. The DDPG schema generally includes the following components and workflow:

This folowing schema outlines the main components and workflow of the DDPG algorithm. DDPG facilitates the learning of optimal policies in continuous action spaces by leveraging the combined strengths of Actor-Critic methods and deep learning.


Ortam (Environment)
[State] --> Actor Network (μ(s|θ^μ)) --> [Action]
[Reward] <---------- Critic Network (Q(s,a|θ^Q))
Replay Buffer <--> Experience Sample (Minibatch)
Target Actor Network (μ'(s|θ^μ'))
Target Critic Network (Q'(s,a|θ^Q'))
Actor Network Update
Critic Network Update

Evaluation


DDPG
Min-Var
DJIA
20000
18000
16000
14000
12000
10000
Portfolio Value
2016-01-04
2016-06-16
2016-11-29
2017-05-16
2017-10-27
2018-04-16
2018-09-28
Date

Benefits

01

The decision-making algorithm can easily be updated.

02

Reduces human error.

03

It's easier to diversify your portfolio.

By implementing these advanced analytics tools, including Reinforcement Learning (RL) and Artificial Intelligence (AI), the Stock Forecast & Analysis App aims to empower financial advisors and finance students to make informed investment decisions.

Q&A?

We invite you to join us in this initiative to revolutionize financial data analysis through AI.