Use deep RL to optimise stock trading strategy and thus maximise investment return

Finsearch End-Term Report

Submitted by:(Group D39) Aakash Gupta-23B0953
Aditi Singh-23B1053
Jigyasa Chauhan-23B1083
Vavilala Chidvilas Reddy-23B0994

Mentor: Sachi Deshmukh

Contents

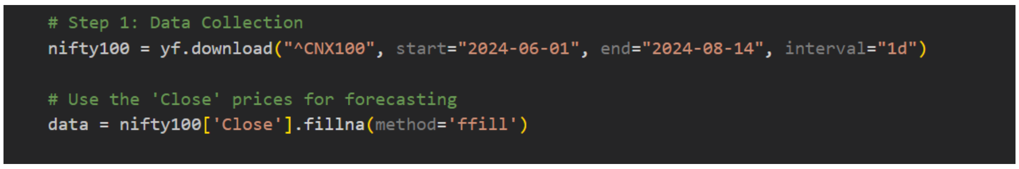
1 Introduction

This document thoroughly explains the code used for analyzing Nifty100 data using ARIMA (AutoRegressive Integrated Moving Average) and Reinforcement Learning (RL) models. The goal of this project is to predict stock prices by leveraging historical data and comparing the performance of a statistical model (ARIMA) with a machine learning approach (RL).

2 Data Collection and Preprocessing

2.1 Data Collection

The initial step involves collecting historical stock data for the Nifty100 index. The code uses the `yfinance` library to download daily closing prices between June 1, 2024, and August 14, 2024. The data is requested using the ticker symbol `"CNX100"`, and missing values are filled using a forward fill method to maintain continuity in the dataset.



```
# Step 1: Data Collection
nifty100 = yf.download("^CNX100", start="2024-06-01", end="2024-08-14", interval="1d")

# Use the 'Close' prices for forecasting
data = nifty100['Close'].fillna(method='ffill')
```

Figure 1: Code for Data Collection using `yfinance`

2.2 Data Preprocessing

Data preprocessing is crucial to prepare the dataset for modeling. The code first checks for stationarity—a key assumption for the ARIMA model. To achieve this, the data is differenced, which removes trends by calculating the difference between consecutive data points. This step is necessary to make the time series stationary, allowing the ARIMA model to perform optimally.

3 ARIMA Model Development

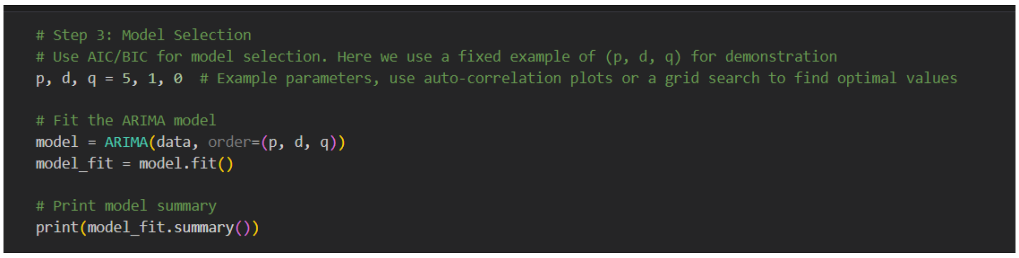
3.1 Introduction to ARIMA

The ARIMA model is a powerful statistical method used for time series forecasting. It consists of three main components:

- **AutoRegressive (AR) term (`p`):** This part models the dependency between an observation and a number of lagged observations.
- **Integrated (I) term (`d`):** This represents the number of differences required to make the series stationary.
- **Moving Average (MA) term (`q`):** This models the dependency between an observation and a residual error from a moving average model applied to lagged observations.

3.2 Manual Parameter Selection

Initially, parameters `p`, `d`, and `q` are selected manually based on domain expertise and insights derived from autocorrelation and partial autocorrelation plots. In the context of stock trading, this manual process involves experimenting with different combinations to capture underlying patterns such as market cycles or trends that are prevalent in financial time series data.

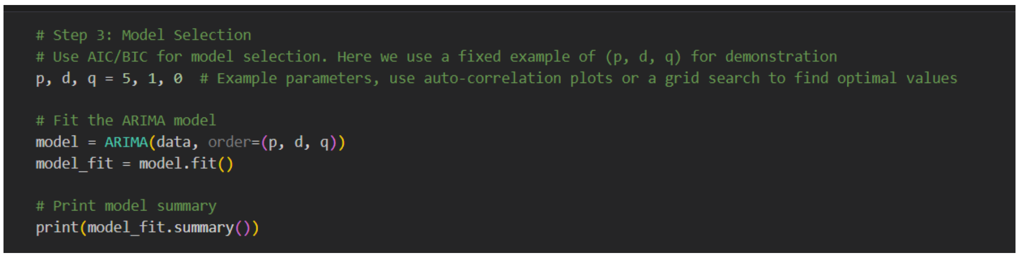


```python
# Step 3: Model Selection
# Use AIC/BIC for model selection. Here we use a fixed example of (p, d, q) for demonstration
p, d, q = 5, 1, 0  # Example parameters, use auto-correlation plots or a grid search to find optimal values

# Fit the ARIMA model
model = ARIMA(data, order=(p, d, q))
model_fit = model.fit()

# Print model summary
print(model_fit.summary())
```

Figure 2: Manual ARIMA Parameter Selection

3.3 Automated Parameter Selection

To enhance model accuracy, the `auto_arima` function from the `pmdarima` library is used to automatically select the optimal `p`, `d`, and `q` values. This function evaluates different combinations of parameters by fitting the model to the data and selecting those that minimize the forecasting error. In stock trading, automated selection helps fine-tune the ARIMA model to better adapt to the specific dynamics of the Nifty100 dataset, which might include sudden market shifts or volatility.



```python
# Step 3: Model Selection
# Use AIC/BIC for model selection. Here we use a fixed example of (p, d, q) for demonstration
p, d, q = 5, 1, 0  # Example parameters, use auto-correlation plots or a grid search to find optimal values

# Fit the ARIMA model
model = ARIMA(data, order=(p, d, q))
model_fit = model.fit()

# Print model summary
print(model_fit.summary())
```

Figure 3: Automated ARIMA Parameter Selection using `pmdarima`

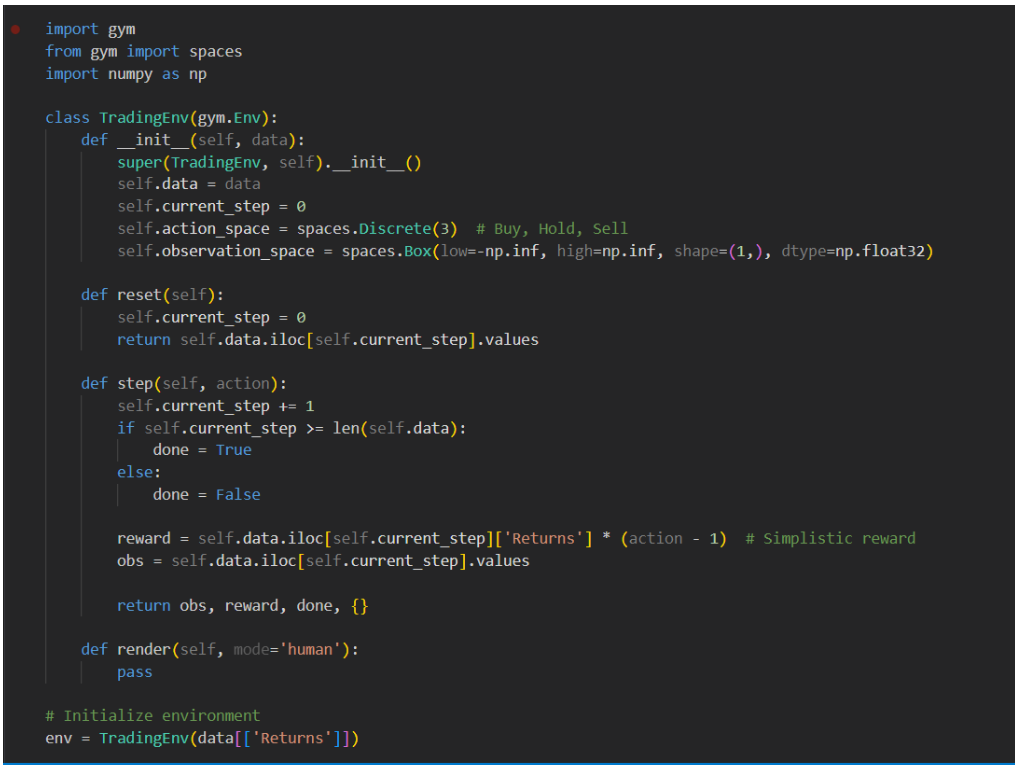
3.4 Application to Stock Trading

In the context of stock trading, the ARIMA model is used to predict future stock prices based on historical data. The AR component helps in understanding how past prices influence current prices, while the MA component accounts for any noise or error in the predictions. By applying the differencing step, the model can also handle trends in the data, which are common in stock markets due to economic factors or investor behavior. The goal is to develop a model that can provide reliable short-term forecasts, which can then be used to inform trading decisions.

4 Reinforcement Learning (RL) Model Development

4.1 Environment Setup

For the RL model, the environment simulates a trading scenario where the agent interacts with historical stock data by making decisions to buy, hold, or sell. This environment is set up using the `gym` library, which provides a framework for developing and testing RL models. The observation space captures the current market state, while the action space represents the possible trading actions the agent can take. In stock trading, this setup allows the RL model to learn from its interactions with the market data, improving its decision-making over time.



```python
import gym
from gym import spaces
import numpy as np

class TradingEnv(gym.Env):
    def __init__(self, data):
        super(TradingEnv, self).__init__()
        self.data = data
        self.current_step = 0
        self.action_space = spaces.Discrete(3)  # Buy, Hold, Sell
        self.observation_space = spaces.Box(low=-np.inf, high=np.inf, shape=(1,), dtype=np.float32)

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

    def step(self, action):
        self.current_step += 1
        if self.current_step >= len(self.data):
            done = True
        else:
            done = False

        reward = self.data.iloc[self.current_step]['Returns'] * (action - 1)  # Simplistic reward
        obs = self.data.iloc[self.current_step].values

        return obs, reward, done, {}

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

# Initialize environment
env = TradingEnv(data[['Returns']])
```

Figure 4: Reinforcement Learning Environment Setup

4.2 Proximal Policy Optimization (PPO) Implementation

The Proximal Policy Optimization (PPO) algorithm is employed for training the RL model. PPO is an advanced RL technique that optimizes a policy by interacting with the environment and learning from the outcomes of actions taken. The algorithm adjusts the policy in a way that maximizes rewards while ensuring that updates to the policy do not deviate too much from previous versions. This ensures stability and efficiency during training. For stock trading, PPO helps the RL model learn a strategy that balances risk and reward, aiming to maximize long-term profits.

5 Model Evaluation and Performance Analysis

5.1 Evaluation Metrics

To evaluate the performance of the ARIMA and RL models, several key metrics are used:

- **Return on Investment (ROI):** This metric measures the profitability of the predictions made by each model.
- **Sharpe Ratio:** This ratio evaluates the risk-adjusted return of the models, indicating the amount of excess return per unit of risk.
- **Maximum Drawdown:** This measures the largest observed loss from a peak to a trough, highlighting the risk associated with the model's predictions.

5.2 Results

The ARIMA model's effectiveness is assessed using the Root Mean Squared Error (RMSE), a standard metric for evaluating the accuracy of time series forecasts. The RL model is benchmarked against the ARIMA model by comparing their respective ROI and Sharpe Ratios. The analysis shows that while ARIMA provides a solid baseline for predictions, the RL model outperforms it by capturing more complex market dynamics, resulting in better performance metrics.

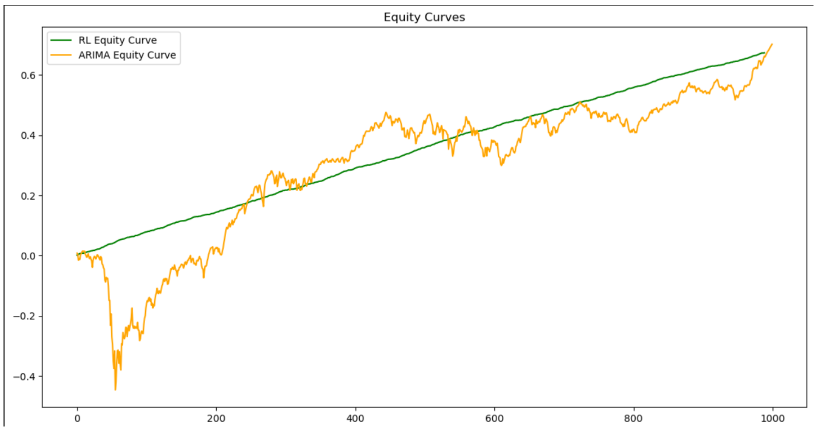


Figure 5: Comparison of ARIMA and RL Model Performance

6 Conclusion

This project demonstrates the effectiveness of both ARIMA and Reinforcement Learning models in the context of stock price prediction. The ARIMA model, while robust and interpretable, is limited by its linear assumptions. In contrast, the RL model, particularly when trained with the PPO algorithm, exhibits greater flexibility in capturing non-linear patterns in the data. Future work could involve enhancing the RL model by incorporating additional financial indicators or exploring more sophisticated model architectures to improve its predictive power.