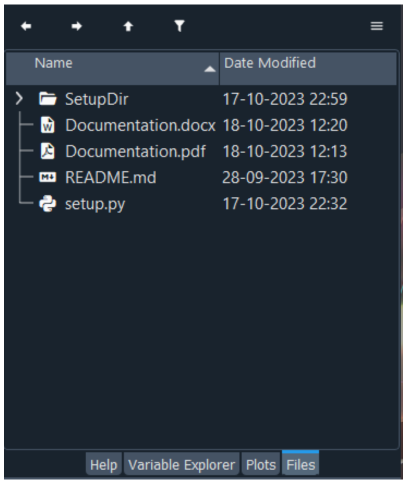
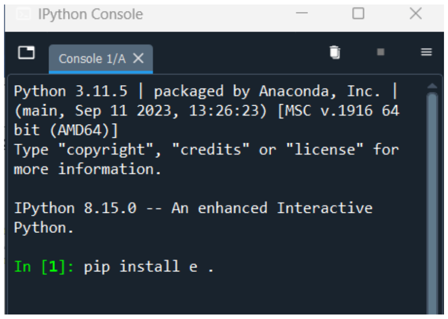
i. Installation of the Environment:

1. Download the folder from github and open the directory named "SingleStockLongOnlyIndiaEnv" in the python IDE as shown in following figure.



Name	Date Modified
SetupDir	17-10-2023 22:59
Documentation.docx	18-10-2023 12:20
Documentation.pdf	18-10-2023 12:13
README.md	28-09-2023 17:30
setup.py	17-10-2023 22:32

2. Next run the command, "pip install e ." in the console as shown in the following figure



```
Python 3.11.5 | packaged by Anaconda, Inc. |
(main, Sep 11 2023, 13:26:23) [MSC v.1916 64
bit (AMD64)]
Type "copyright", "credits" or "license" for
more information.

IPython 8.15.0 -- An enhanced Interactive
Python.

In [1]: pip install e .
```

ii. Call the Environment:

1. To check the environment run the .py file, "check_the_env.py" from the directory, "training_and_testing"

2. Stock datas are provided in the "stock_data" directory.

iii. Description of the Environment:

1. <u>Observation:</u>

At the tth time instant, an agent observes
1. Open Prices from (t-n) to t time steps
2. High Prices from (t-n-1) to (t-1) time steps
3. Low Prices from (t-n-1) to (t-1) time steps
4. Close Prices from (t-n-1) to (t-1) time steps
5. Volume from (t-n-1) to (t-1) time steps

6. Money reserve from (t-n-1) to (t-1) time steps
7. Stock holding from (t-n-1) to (t-1) time steps
8. Action taken from (t-n-1) to (t-1) time steps

Where, (n is the look back window)

Hence, the shape of the observation space is as follows
spaces.Box(low=-np.inf, high=np.inf, shape=(8, self.look_back_window), dtype=np.float64)

A sample observation will look like as follows
np.obs([Open Prices from (t-n) to t time steps],
[High Prices from (t-n-1) to (t-1) time steps],
[Low Prices from (t-n-1) to (t-1) time steps],
[Close Prices from (t-n-1) to (t-1) time steps],
[Volume from (t-n-1) to (t-1) time steps],
[Money reserve from (t-n-1) to (t-1) time steps],
[Stock holding from (t-n-1) to (t-1) time steps],
[Action taken from (t-n-1) to (t-1) time steps])

2. <u>Action</u>:

This is a long-only environment. No shorting action is allowed. The action is represented as an int number in the range of [-100, 100].

0 represents the Hold action.

A positive number represents the Buy action. The number represents the percentage of money reserve that will be used to buy stock. The target position is as follows

$$\text{target\_position\_in\_money} = (\text{action} \times \text{money\_reserve})/100$$

A negative number represents the Sell action. The number represents the percentage of stock reserve that will be sold. The target position is as follows

$$\text{target\_position\_in\_quantity} = (-\text{action} \times \text{stock\_holding\_in\_quantity})/100$$

The action space looks like as following
spaces.Discrete(201, start=-100)

3. <u>The trade execution mechanism (Indian Market)</u>:

3.1. Random selection of a stock: At every episode, a random stock is selected from the list of stocks, i.e., list_of_stock.

3.2. Stock data: The environment needs historic market data (.csv files) in OHLC format which is saved in the data directory, i.e., data_dir.

3.2.1. The index column must be in datetime format and named as ‘date’.

3.2.2. The input dataframe must have ‘open’, ‘high’, ‘low’, ‘close’, and ‘volume’ columns

3.3. Random time duration: At every episode, a random time duration of the episode length from the data frame is selected. The length of the episode (episode_length_in_steps) is user-defined.

3.4. Trading Fee: At every step, a trading fee is needed to pay. The total fee is ((trade_volume_in_money × fee_percentage)/100) or max_fee, whichever is the minimum.

3.5. Execution Price: The execution price is a random price between the high and low price of the time step.

3.6. Buy and Sell mechanism: In India, fractional stock buy and sell is not allowed.

3.6.1. Buy action execution: Buy action is executed when action_value > 0

The target position is given by,

target_position_in_money = self.money_reserve × (1 – (self.max_fee /100)) × (action_value/100)

The term, (1 – (self.max_fee /100)) ensures that there is enough money left to pay the fee. When the fee is much smaller this approximation should not affect the environment significantly.

The actual stock buy quantity is

stock_buy_in_quantity = int(target_position_in_money / execution_price)

3.6.2. Sell action execution: Sell action is executed when action_value < 0

The stock sell quantity is given by

stock_sell_in_quantity = int(self.stock_holdings_in_num × (-self.action/100))

3.7. Episode Ending: An episode ends is the maximum number of steps, i.e., (episode_length_in_steps) is over of the net_worth is reduced more than the acceptable loss, (max_acceptable_drop_down_percentage_episode_stopping).

4. <u>Reward</u>:

All the reward is calculated in the ratio of initial_fund_in_money to generalize the reward value. The reward function has the following components

4.1. Inflation: At every step, a negative reward is imposed to reflect inflation. It forces the agent to generate profit faster. The approximation is done as follows

The inflation percentage is x per year. In a year trading days are approximately 260 days. Every day there are (6×60)/y time steps. y is the timestamp of the input dataframe in min e.g., 5 min, 15 min.

Hence at each step the negative reward for inflation = $-xy/\ (6\times60\times260\times100)$

A simple linear approximation is adopted here.

4.2. Increase in Net Worth: The reward from net worth increase at t^{th} time instant is given by

(net_worth_in_money[t] – net_worth_in_money[t-1])/ initial_fund_in_money