Robinhood cryptocurrency high frequency PROPRTS

(predictive reactive observational pattern recognition trading system)

layout & procedure

Author: Braden Lee

Github link: https://github.com/0x00C0DE/robinhood_crypto_bot

Introduction:

When it comes to trading cryptocurrency, the crypto markets are active 24/7. Therefore in my own opinion, it's reasonable to develop a complex system that can take advantage of the ever evolving crypto market along with its periods of high volatility that could occur at any moment via monitoring and trading the coins through automation. It's my belief that it's better to put money to work in the market rather than in a fund. Because nowadays the markets are so volatile that I believe it would be advantageous to exploit the dramatic shifts in prices to not only scalp profit when prices are higher than your average cost, but also reinvest the scalped profit via dollar cost averaging when the prices are lower than your average cost. What this method provides is a means of automatically increasing shares over time while attempting to retain the average cost of which the user paid for the total of the coins. This acts as a safe measure such that, even if the prices of the coins are declining, as long as the user keeps accumulating more coins over time, the decline in account balance will be mitigated. While it's implemented as a safe measure, it could also be seen as a means to provide exponential returns. If the price of the coins are increasing, the user will sell off portions of their total coins for scalp profit. Then the moment the price of the coin dips, the user would buy back a higher ratio of shares to price, such that the amount of shares are more than the user sold off for scalp profit and the price of the coins when bought is lower than what the user sold at. Exponentially increasing profit via increasing total stockpile of shares and waiting for those shares to increase in value.

- The ***goal*** of this program is to provide higher ROI than traditional savings for retirement like IRAs and 401k plans.

    1. Array to hold historical data and perform multiple algorithms before making a decision

    2. Price comparison percentage change difference algorithm
        a. (generalized reactive observational method for real life high volatility scenarios)
        b. Uses average cost ceiling (ensures there is a large enough disparity between the live price and the required minimal amount that is greater than the live price to obtain profitability when selling)

    c. Uses average cost floor (ensures there is a large enough disparity between the live price and the required minimal amount that is lower than the live price to obtain profitability when buying)

3. Calculate slope of each combination of paired historical data prices from the array. Then analyze the sequential steepness of each of the slopes for patterns. (m=(y2-y1)/(x2-x1) (needs a fixed sized plot for array of historical data to fit correctly)
    a. (assists in helping make accurate trading decision)

4. Utilize linear regression on the array of historical data prices to help predict future price movements. Record the steepness of the linear regression, after a half of the elements in the array are replaced with new values re-calculate the linear regression and record. Redo this process until there is a quarter of linear regression data element from the total array of historical data prices.
    a. Uses time as a variable to predict the dependent variable price

5.
   Analyze the array of linear regression data to predict future price movements and makes more accurate decisions when utilized along with the other filtering algorithms.
    a. (Allows for informative predictions to be made)

6. Utilize standard deviation among the historical set of data for a period of 1 day, 1 week, and 1 month. Compare the standard deviations
    a. (Checks clusters of datasets (1 day, 1 week, 1 month… etc) for patterns)

Files:

[Temporary files are the latest/prototype versions]

(you will have to change script_path pathing inside PROPRTS-1-proto-V1.py)
(you will have to change cv2.imread pathing inside proprtsImageViewer-1-protoV1.py)

1. Run PROPRTS-1-protoV1.py until "done writing new data to txt file" appears or when PROPRTS-1-LR-data.txt appears in the working directory
2. Read the beginning instructions of PROPRTS-main-proto-V1.py, then run

Temporary files:

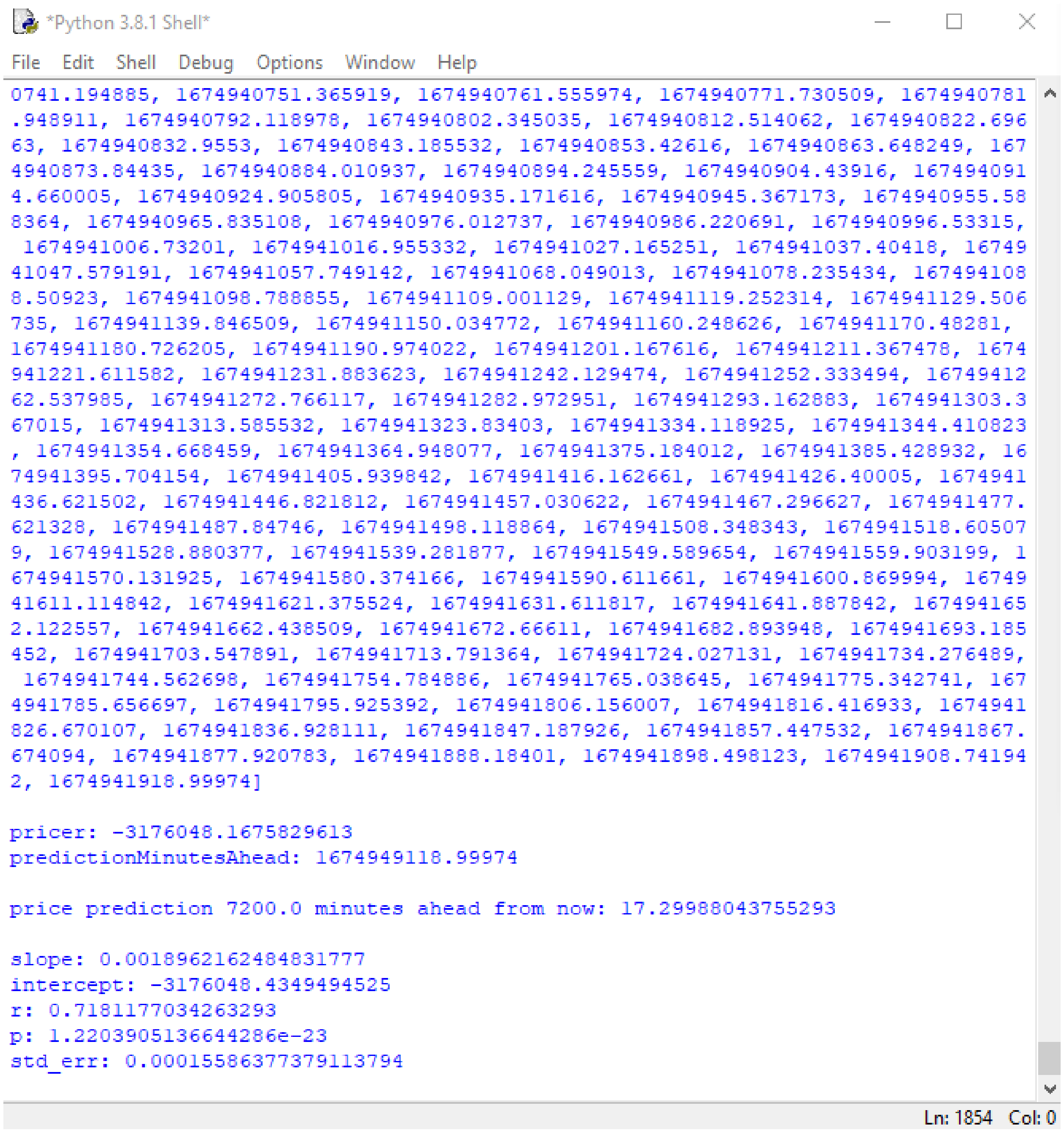
- PROPRTS-main-protoV1.py
    - Temporary main program
- proprtsImageViewer-1-protoV1.py
    - Temporary image viewer

    - PROPRTS-1-protoV1.py
        - Temporary data collector and writer

- **PROPRTS-main.py**
    - Main program that initiates/launches all other programs to run through running this single script
    - Reads data from text files generated from other programs
    - Responsible for making all final trade decisions utilizing the data from other programs
- **proprts-linear-regression.py**
    - Linear regression algorithm used to assist the main program in making mathematical decisions
    - Responsible for future price predictions
- **PROPRTS-1.py**
    - Linear regression algorithm for every 1 minutes
    - Writes data to text files for the main program
        - PROPRTS-1-LR-data.txt
- **PROPRTS-5.py**
    - Linear regression algorithm for every 5 minutes
    - Writes data to text files for the main program
        - PROPRTS-5-LR-data.txt
- **PROPRTS-15.py**
    - Linear regression algorithm for every 15 minutes
    - Writes data to text files for the main program
        - PROPRTS-15-LR-data.txt
- **PROPRTS-30.py**
    - Linear regression algorithm for every 30 minutes
    - Writes data to text files for the main program
        - PROPRTS-30-LR-data.txt
- **PROPRTS-60.py**
    - Linear regression algorithm for every 60 minutes
    - Writes data to text files for the main program
        - PROPRTS-60-LR-data.txt
- **PROPRTS-120.py**
    - Linear regression algorithm for every 120 minutes
    - Writes data to text files for the main program
        - PROPRTS-120-LR-data.txt
- **proprtsImageViewer.py**
    - Visualizes the data for linear regression, showing a real time graph with the best fit line

Example:

https://www.youtube.com/watch?v=t-YDShxDQoQ

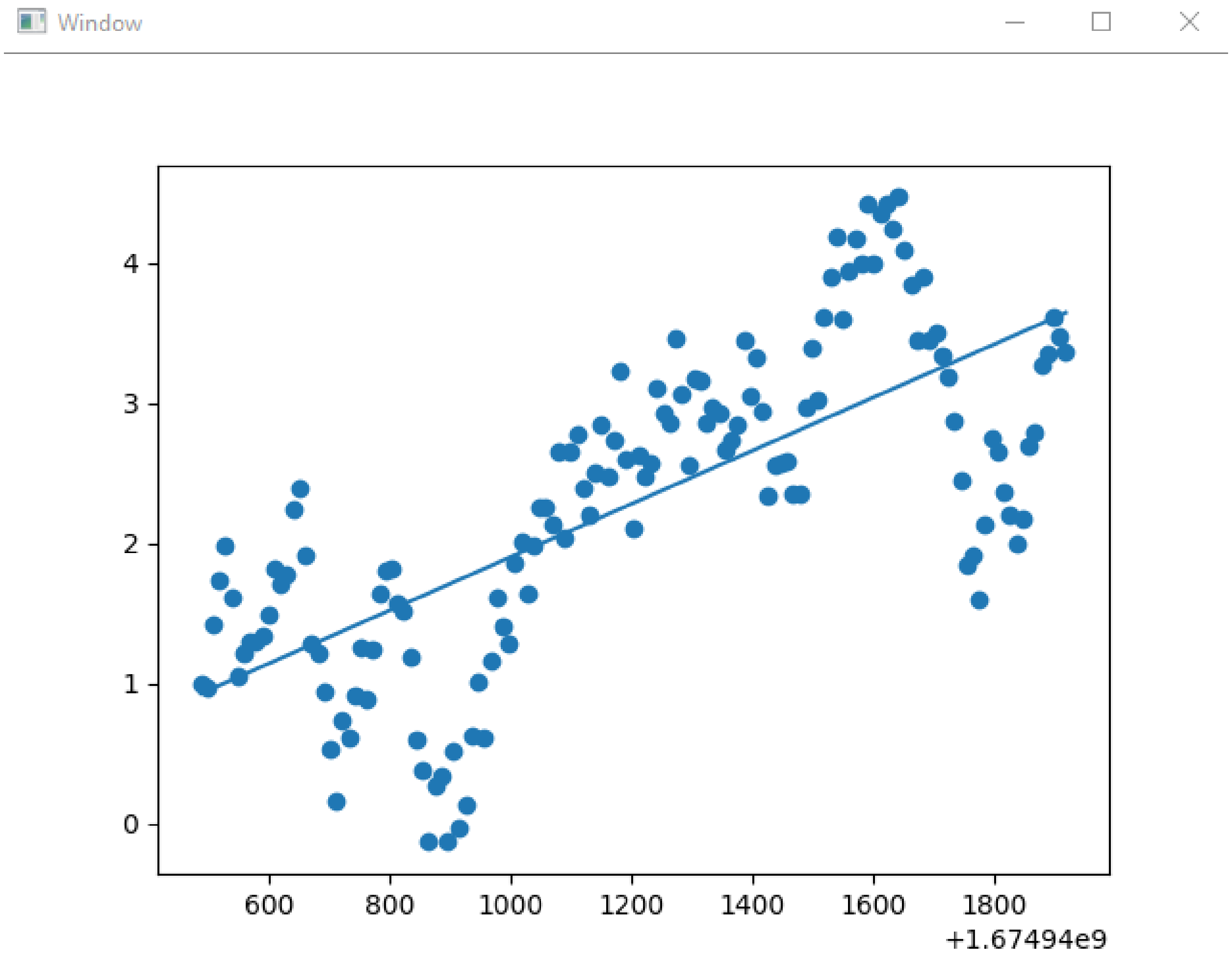


```
0741.194885, 1674940751.365919, 1674940761.555974, 1674940771.730509, 1674940781
.948911, 1674940792.118978, 1674940802.345035, 1674940812.514062, 1674940822.696
63, 1674940832.9553, 1674940843.185532, 1674940853.42616, 1674940863.648249, 167
4940873.84435, 1674940884.010937, 1674940894.245559, 1674940904.43916, 167494091
4.660005, 1674940924.905805, 1674940935.171616, 1674940945.367173, 1674940955.58
8364, 1674940965.835108, 1674940976.012737, 1674940986.220691, 1674940996.53315,
 1674941006.73201, 1674941016.955332, 1674941027.165251, 1674941037.40418, 16749
41047.579191, 1674941057.749142, 1674941068.049013, 1674941078.235434, 167494108
8.50923, 1674941098.788855, 1674941109.001129, 1674941119.252314, 1674941129.506
735, 1674941139.846509, 1674941150.034772, 1674941160.248626, 1674941170.48281,
1674941180.726205, 1674941190.974022, 1674941201.167616, 1674941211.367478, 1674
941221.611582, 1674941231.883623, 1674941242.129474, 1674941252.333494, 16749412
62.537985, 1674941272.766117, 1674941282.972951, 1674941293.162883, 1674941303.3
67015, 1674941313.585532, 1674941323.83403, 1674941334.118925, 1674941344.410823
, 1674941354.668459, 1674941364.948077, 1674941375.184012, 1674941385.428932, 16
74941395.704154, 1674941405.939842, 1674941416.162661, 1674941426.40005, 1674941
436.621502, 1674941446.821812, 1674941457.030622, 1674941467.296627, 1674941477.
621328, 1674941487.84746, 1674941498.118864, 1674941508.348343, 1674941518.60507
9, 1674941528.880377, 1674941539.281877, 1674941549.589654, 1674941559.903199, 1
674941570.131925, 1674941580.374166, 1674941590.611661, 1674941600.869994, 16749
41611.114842, 1674941621.375524, 1674941631.611817, 1674941641.887842, 167494165
2.122557, 1674941662.438509, 1674941672.66611, 1674941682.893948, 1674941693.185
452, 1674941703.547891, 1674941713.791364, 1674941724.027131, 1674941734.276489,
 1674941744.562698, 1674941754.784886, 1674941765.038645, 1674941775.342741, 167
4941785.656697, 1674941795.925392, 1674941806.156007, 1674941816.416933, 1674941
826.670107, 1674941836.928111, 1674941847.187926, 1674941857.447532, 1674941867.
674094, 1674941877.920783, 1674941888.18401, 1674941898.498123, 1674941908.74194
2, 1674941918.99974]

pricer: -3176048.1675829613
predictionMinutesAhead: 1674949118.99974

price prediction 7200.0 minutes ahead from now: 17.29988043755293

slope: 0.0018962162484831777
intercept: -3176048.4349494525
r: 0.7181177034263293
p: 1.2203905136644286e-23
std_err: 0.00015586377379113794
```


Window
4
3
2
1
0
600
800
1000
1200
1400
1600
1800
+1.67494e9