

25_solana_buy_sell

Def #ez.py

```
 ez.py > ...
13
14  '''
15
16  from config import *
17  import nice_funcs as n
18  import time
19  from termcolor import colored, cprint
20  import schedule
21
22
23 ##### ASKING USER WHAT THEY WANNA DO - WILL REMOVE USER SOON AND REPLACE WITH BOT #####
24 buy = 0
25 print('slow down, dont trade by hand... moon dev told you so...')
26 buy = input('0 to close, 1 to buy, 2 eth trade, 4 pnl close, 5 market maker, 6 funding bot, 7 liq bot, 8 sdz bot, 9 batch sdz, : ')
27 print('you entered:', buy)
28 buy = int([buy])
29
30
31 def bot():
32
33     symbol = symbol1
34
35     while buy == 0:
36         print('closing position')
37         # get pos first
38         pos = n.get_position(symbol)
39         while pos > 0:
40             n.chunk_kill(symbol, max_usd_order_size, slippage)
41             pos = n.get_position(symbol)
42             time.sleep(1)
43
44         if pos < .9:
45             print('position closed thanks moon dev....')
46             time.sleep(876786)
```

```
ez.py > bot
28     buy = int(buy)
29
30
31     def bot():
32
33         symbol = symbol1
34
35         while buy == 0:
36             print('closing position')
37             # get pos first
38             pos = n.get_position(symbol)
39             while pos > 0:
40                 n.chunk_kill(symbol, max_usd_order_size, slippage)
41                 pos = n.get_position(symbol)
42                 time.sleep(1)
43
44             if pos < .9:
45                 print('position closed thanks moon dev....')
46                 time.sleep(876786)
47                 break
48
49
50         print('bot successfully closed position...')
51
52         while buy == 1:
53             from config import buy_under, sell_over
54             print('opening buying position')
55             pos = n.get_position(symbol)
56             price = n.token_price(symbol)
57             pos_usd = pos * price
58             size_needed = usd_size - pos_usd
59             if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
60             else: chunk_size = size_needed
61
62             chunk_size = int(chunk_size * 10**6)
63             chunk_size = str(chunk_size)
64
65             if pos_usd > (.97 * usd_size):
66                 print('position filled')
```

```

31 def bot():
30     print('bot successfully closed position...')
31
32     while buy == 1:
33         from config import buy_under, sell_over
34         print('opening buying position')
35         pos = n.get_position(symbol)
36         price = n.token_price(symbol)
37         pos_usd = pos * price
38         size_needed = usd_size - pos_usd
39         if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
40         else: chunk_size = size_needed
41
42         chunk_size = int(chunk_size * 10**6)
43         chunk_size = str(chunk_size)
44
45         if pos_usd > (.97 * usd_size):
46             print('position filled')
47             time.sleep(7867678)
48
49         while pos_usd < (.97 * usd_size):
50
51             print(f'position: {round(pos,2)} price: {round(price,8)} buy_under: {buy_under} pos_usd: ${round(pos_usd,2)}')
52
53             try:
54
55                 for i in range(orders_per_open):
56                     n.market_buy(symbol, chunk_size, slippage)
57                     # cprint green background black text
58                     cprint(f'chunk buy submitted of {symbol[-4:]} sz: {chunk_size} you my dawg moon dev', 'white', 'on_blue')
59                     time.sleep(1)
60
61                     time.sleep(tx_sleep)
62
63                     pos = n.get_position(symbol)
64                     price = n.token_price(symbol)
65                     pos_usd = pos * price
66                     size_needed = usd_size - pos_usd
67                     if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
68                     else: chunk_size = size_needed
69
70                     chunk_size = int(chunk_size * 10**6)
71                     chunk_size = str(chunk_size)
72
73             except:
74
75                 try:
76                     cprint(f'trying again to make the order in 30 seconds....', 'light_blue', 'on_light_magenta')
77                     time.sleep(30)
78                     for i in range(orders_per_open):
79                         n.market_buy(symbol, chunk_size, slippage)
80                         # cprint green background black text
81                         cprint(f'chunk buy submitted of {symbol[-4:]} sz: {chunk_size} you my dawg moon dev', 'white', 'on_blue')
82                         time.sleep(1)
83
84                         time.sleep(tx_sleep)
85                         pos = n.get_position(symbol)
86                         price = n.token_price(symbol)
87                         pos_usd = pos * price
88                         size_needed = usd_size - pos_usd
89                         if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
90                         else: chunk_size = size_needed
91
92                         chunk_size = int(chunk_size * 10**6)
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110

```

```

92     except:
93
94         try:
95             cprint(f'trying again to make the order in 30 seconds.....', 'light_blue', 'on_light_magenta')
96             time.sleep(30)
97             for i in range(orders_per_open):
98                 n.market_buy(symbol, chunk_size, slippage)
99                 # cprint green background black text
100                cprint(f'chunk buy submitted of {symbol[-4:]}, sz: {chunk_size} you my dawg moon dev', 'white', 'on_blue')
101                time.sleep(1)
102
103                time.sleep(tx_sleep)
104                pos = n.get_position(symbol)
105                price = n.token_price(symbol)
106                pos_usd = pos * price
107                size_needed = usd_size - pos_usd
108                if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
109                else: chunk_size = size_needed
110                chunk_size = int(chunk_size * 10**6)
111                chunk_size = str(chunk_size)
112
113
114        except:
115            cprint(f'Final Error in the buy, restart needed', 'white', 'on_red')
116            time.sleep(10)
117            break
118
119        pos = n.get_position(symbol)
120        price = n.token_price(symbol)
121        pos_usd = pos * price
122        size_needed = usd_size - pos_usd
123        if size_needed > max_usd_order_size: chunk_size = max_usd_order_size
124        else: chunk_size = size_needed
125        chunk_size = int(chunk_size * 10**6)
126        chunk_size = str(chunk_size)
127
128

```

```

129    # cprint white on greeen
130    cprint(f'position filled of {symbol[-4:]}, total: ${pos_usd}', 'white', 'on_green')
131    break
132
133
134 while buy == 2:
135
136     print('starting eth trade.. ')
137
138     eth_df = n.fetch_candle_data_with_smas('ETH', '1d', 200, sma_windows=[20, 41])

```

nicefuncs.py

#def_market_buy

```
209 def market_buy(token, amount, slippage):
210     import requests
211     import sys
212     import json
213     import base64
214     from solvers.keypair import Keypair
215     from solvers.transaction import VersionedTransaction
216     from solana.rpc.api import Client
217     from solana.rpc.types import TxOpts
218     import time
219
220     # Import the secret key and Ankr endpoint from dontshare
221     import dontshare.dontshare as d
222
223     KEY = Keypair.from_base58_string(d.sol_key)
224     SLIPPAGE = slippage # 5000 is 50%, 500 is 5% and 50 is .5%
225     PRIORITY_FEE = 5000 # Hardcoded prioritization fee (adjust as needed)
226     QUOTE_TOKEN = "EPjFWdd5AufqSSqeM2qN1xzyapC8G4wEGGkZwyTDt1v" # usdc
227
228     # Use Ankr's RPC endpoint
229     http_client = Client(d.ankr_key)
230     #http_client = Client("https://rpc.ankr.com/solana/b0d038027b60cf79aba1de4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")
231
232     quote_url = f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}'
233     swap_url = 'https://quote-api.jup.ag/v6/swap'
234
235     while True:
236         try:
237             quote = requests.get(quote_url).json()
238             # print(quote)
239
240             txRes = requests.post(swap_url,
241                                   headers={"Content-Type": "application/json"},
242                                   data=json.dumps({
243                                       "quoteResponse": quote,
244                                       "userPublicKey": str(KEY.pubkey()),
245                                       "prioritizationFeeLamports": PRIORITY_FEE # Hardcoded fee
246                                   }).json()
247
248             # Use Ankr's RPC endpoint
249             #http_client = Client(d.ankr_key) # QUICKNODE
250             http_client = Client("https://api.mainnet-beta.solana.com/")
251
252             quote_url = f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}'
253             swap_url = 'https://quote-api.jup.ag/v6/swap'
```

```

236     while True:
237         try:
238             quote = requests.get(quote_url).json()
239             # print(quote)
240
241             txRes = requests.post(swap_url,
242                                   headers={"Content-Type": "application/json"},
243                                   data=json.dumps({
244                                       "quoteResponse": quote,
245                                       "userPublicKey": str(KEY.pubkey()),
246                                       "prioritizationFeeLamports": PRIORITY_FEE  # Hardcoded fee
247                                   })).json()
248             # print(txRes)
249             swapTx = base64.b64decode(txRes['swapTransaction'])
250             # print(swapTx)
251             tx1 = VersionedTransaction.from_bytes(swapTx)
252             tx = VersionedTransaction(tx1.message, [KEY])
253             txId = http_client.send_raw_transaction(bytes(tx), TxOpts(skip_preflight=True)).value
254             print(f"https://solscan.io/tx/{str(txId)}")
255             break
256         except requests.exceptions.RequestException as e:
257             print(f"Request failed: {e}")
258             time.sleep(5)  # Wait for 5 seconds before retrying
259         except Exception as e:
260             print(f"An error occurred: {e}")
261             time.sleep(5)  # Wait for 5 seconds before retrying
262
263
264     def market_buy_juprpc(token, amount, slippage):
265
266         import requests
267         import sys
268         import json
269         from solders import Keypair

```

#Def market_sell

```

307     def market_sell(QOTE_TOKEN, amount, slippage):
308
309         import requests
310         import sys
311         import json
312         import base64
313         from solders.keypair import Keypair
314         from solders.transaction import VersionedTransaction
315         from solana.rpc.api import Client
316         from solana.rpc.types import TxOpts
317         import dontshare.dontshare as d
318
319         print('inside market sell')
320
321         KEY = Keypair.from_base58_string(d.sol_key)
322         SLIPPAGE = slippage # 5000 is 5%, 500 is 5% and 50 is .5%
323         print(f'slippage is {SLIPPAGE}')
324         #PRIORITY_FEE = PRIORITY_FEE
325         #QUOTE_TOKEN = "So1111111111111111111111111111111111111111111111111111111112"
326         # QUOTE_TOKEN = "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v" # usdc
327
328         # token would be usdc for sell orders cause we are selling
329         token = "EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"
330
331         http_client = Client(d.ankr_key)
332         #http_client = Client("https://rpc.ankr.com/solana/b6d038027b60cf79abade4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")
333
334
335         quote = requests.get(f"https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}").json()
336         #print(quote)
337         txRes = requests.post('https://quote-api.jup.ag/v6/swap',
338                               headers={"Content-Type": "application/json"},
339                               data=json.dumps({
340                                   "quoteResponse": quote,
341                                   "userPublicKey": str(KEY.pubkey()),
342                               }))

```

```

341         "userPublicKey": str(KEY.pubkey()),
342         "prioritizationFeeLamports": PRIORITY_FEE # or replace 'auto' with your specific lamport value
343     }).json()
344     #txRes = requests.post('https://quote-api.up.ag/v6/swap',headers={"Content-Type": "application/json"}, data=json.dumps({"quoteResponse": quote, "userPublicKey": str(KEY.pubkey())}))
345     #print(txRes)
346     swapTx = base64.b64decode(txRes['swapTransaction'])
347     #print(swapTx)
348     tx1 = VersionedTransaction.from_bytes(swapTx)
349     #print(tx1)
350     tx = VersionedTransaction(tx1.message, [KEY])
351     #print(tx)
352     txId = http_client.send_raw_transaction(bytes(tx), TxOpts(skip_preflight=True)).value
353     print(f"https://solscan.io/tx/{str(txId)}")

```

\#Def Get_posistions from nicefuncs.py

```

nice_funcs.py > get_position
685 def get_position(token_mint_address):
686     """
687     Fetches the balance of a specific token given its mint address from a DataFrame.
688
689     Parameters:
690     - dataframe: A pandas DataFrame containing token balances with columns ['Mint Address', 'Amount'].
691     - token_mint_address: The mint address of the token to find the balance for.
692
693     Returns:
694     - The balance of the specified token if found, otherwise a message indicating the token is not in the wallet.
695     """
696
697     dataframe = fetch_wallet_token_single(address, token_mint_address)
698
699     #dataframe = pd.read_csv('data/token_per_addy.csv')
700
701     print('-----')
702     #print(dataframe)
703
704     #print(dataframe)
705
706     # Check if the DataFrame is empty
707     if dataframe.empty:
708         print("The DataFrame is empty. No positions to show.")
709         return 0 # Indicating no balance found
710
711     # Ensure 'Mint Address' column is treated as string for reliable comparison
712     dataframe['Mint Address'] = dataframe['Mint Address'].astype(str)
713
714     # Check if the token mint address exists in the DataFrame
715     if dataframe['Mint Address'].isin([token_mint_address]).any():
716         # Get the balance for the specified token
717         balance = dataframe.loc[dataframe['Mint Address'] == token_mint_address, 'Amount'].iloc[0]
718         #print(f"Balance for {token_mint_address[-4:]} token: {balance}")
719         return balance
720     else:
721         # If the token mint address is not found in the DataFrame, return a message indicating so
722         print("Token mint address not found in the wallet.")
723         return 0 # Indicating no balance found
724

```

#Def fetch_wllet_token_single

```
622 def fetch_wallet_holdings_og(address):
650     print(df)
651     # Assuming cprint is a function you have for printing in color
652     cprint(f'** Total USD balance is {df["USD Value"].sum()}', 'white', 'on_green')
653     # Save the filtered DataFrame to a CSV file
654     # TOKEN_PER_ADDY_CSV = 'filtered_wallet_holdings.csv' # Define your CSV file name
655     # df.to_csv(TOKEN_PER_ADDY_CSV, index=False)
656 else:
657     # If the DataFrame is empty, print a message or handle it as needed
658     cprint("No wallet holdings to display.", 'white', 'on_red')
659
660 return df
661
662 def fetch_wallet_token_single(address, token_mint_address):
663
664     df = fetch_wallet_holdings_og(address)
665
666     # filter by token mint address
667     df = df[df['Mint Address'] == token_mint_address]
668
669     return df
670
671
672 def token_price(address):
673     API_KEY = d.birdeye
674     url = f"https://public-api.birdeye.so/defi/price?address={address}"
675     headers = {"X-API-KEY": API_KEY}
676     response = requests.get(url, headers=headers)
677     price_data = response.json()
678
679     if price_data['success']:
680         return price_data['data']['value']
681     else:
682         return None
683
684
685 def get_position(token_mint_address):
686     """
```

#Def fetch_wallet_holdings

```
⚡ nice_funcs.py > ⚡ fetch_wallet_holdings_og
621
622     def fetch_wallet_holdings_og(address):
623
624         API_KEY = d.birdeye # Assume this is your API key; replace it with the actual one
625
626         # Initialize an empty DataFrame
627         df = pd.DataFrame(columns=['Mint Address', 'Amount', 'USD Value'])
628
629         url = f"https://public-api.birdeye.so/v1/wallet/token_list?wallet={address}"
630         headers = {"x-chain": "solana", "X-API-KEY": API_KEY}
631         response = requests.get(url, headers=headers)
632
633         if response.status_code == 200:
634             json_response = response.json()
635
636             if 'data' in json_response and 'items' in json_response['data']:
637                 df = pd.DataFrame(json_response['data']['items'])
638                 df = df[['address', 'uiAmount', 'valueUsd']]
639                 df = df.rename(columns={'address': 'Mint Address', 'uiAmount': 'Amount', 'valueUsd': 'USD Value'})
640                 df = df.dropna()
641                 df = df[df['USD Value'] > 0.05]
642             else:
643                 cprint("No data available in the response.", 'white', 'on_red')
644
645         else:
646             cprint(f"Failed to retrieve token list for {address}.", 'white', 'on_magenta')
647
648         # Print the DataFrame if it's not empty
649         if not df.empty:
650             print(df)
651             # Assuming cprint is a function you have for printing in color
652             cprint(f'* Total USD balance is {df["USD Value"].sum()}', 'white', 'on_green')
653             # Save the filtered DataFrame to a CSV file
654             # TOKEN_PER_ADDY_CSV = 'filtered_wallet_holdings.csv' # Define your CSV file name
655             # df.to_csv(TOKEN_PER_ADDY_CSV, index=False)
656         else:
657             # If the DataFrame is empty, print a message or handle it as needed
658             cprint("No wallet holdings to display.", 'white', 'on_red')
659
660     return df
```

Nicefuncs.py

#def_chunk_hkill

```

❷ nice_funcs.py > ⌂ chunk_kill
957 def chunk_kill(token_mint_address, max_usd_sell_size, slippage):
958     """
959
960     this function closes the position in full after CHUNKING ORDERS
961
962     if the usd_size > 10k then it will chunk in 10k orders
963     ...
964
965
966     # if time is on the 5 minute do the balance check, if not grab from data/current_position.csv
967     balance = get_position(token_mint_address)
968     cprint(f'inside chunk... balance {balance}', 'white', 'on_black')
969
970     # get current price of token
971     price = token_price(token_mint_address)
972     price = float(price)
973     usd_value = balance * price
974
975     if usd_value < max_usd_sell_size:
976         sell_size = balance
977     else:
978         sell_size = max_usd_sell_size / price
979
980
981     cprint(f'balance is {balance} and selling this amount {sell_size}', 'white', 'on_black')
982
983
984     # round to 2 decimals
985     sell_size = round_down(sell_size, 2)
986     decimals = 0
987     decimals = get_decimals(token_mint_address)
988     sell_size = int(sell_size * 10 ** decimals)
989
990     #printf('bal: {balance} price: {price} usdVal: {usd_value} TP: {tp} sell size: {sell_size} decimals: {decimals}')
991
992     while usd_value > 0:
993         # 100 selling 70% ..... selling 30 left
994         #printf(f'for {token_mint_address[-4:]} closing position cause exit all positions is set to {EXIT_ALL_POSITIONS} and value is {usd_value}')
995         try:

```

```

957 def chunk_kill(token_mint_address, max_usd_sell_size, slippage):
958     #printf(f'for {token_mint_address[-4:]} closing position cause exit all positions is set to {EXIT_ALL_POSITIONS}')
959     try:
960
961         market_sell(token_mint_address, sell_size, slippage)
962         cprint(f'just made an order {token_mint_address[-4:]} selling {sell_size} ...', 'white', 'on_blue')
963         time.sleep(1)
964         market_sell(token_mint_address, sell_size, slippage)
965         cprint(f'just made an order {token_mint_address[-4:]} selling {sell_size} ...', 'white', 'on_blue')
966         time.sleep(1)
967         market_sell(token_mint_address, sell_size, slippage)
968         cprint(f'just made an order {token_mint_address[-4:]} selling {sell_size} ...', 'white', 'on_blue')
969         time.sleep(15)
970
971     except:
972         cprint('order error.. trying again', 'white', 'on_red')
973         # time.sleep(7)
974
975     balance = get_position(token_mint_address)
976     price = token_price(token_mint_address)
977     usd_value = balance * price
978     tp = sell_at_multiple * USDC_SIZE
979
980
981     if usd_value < max_usd_sell_size:
982         sell_size = balance
983     else:
984         sell_size = max_usd_sell_size / price
985
986
987     cprint(f'balance is {balance} and selling this amount {sell_size}', 'white', 'on_black')
988
989
990     # down downwards to 2 decimals
991     sell_size = round_down(sell_size, 2)
992
993     decimals = 0
994     decimals = get_decimals(token_mint_address)
995     #printf(f'xxxxxxxx for {token_mint_address[-4:]} decimals is {decimals}')
996     sell_size = int(sell_size * 10 ** decimals)
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032

```

```

1031     #print(f'xxxxxxxxx for {token_mint_address[-4:]} decimals is {decimals}')
1032     sell_size = int(sell_size * 10 **decimals)
1033
1034
1035     else:
1036         print(f'for {token_mint_address[-4:]} value is {usd_value} ')
1037         #time.sleep(10)
1038
1039     print('closing position in full...')
1040
1041 def kill_switch(token_mint_address, slippage):
1042
1043     ''' this function closes the position in full
1044

```

#Def get_token_price

```

@ nice_funcs.py > token_price
622 def fetch_wallet_holdings_og(address):
623     return df
624
625
626 def fetch_wallet_token_single(address, token_mint_address):
627
628     df = fetch_wallet_holdings_og(address)
629
630     # filter by token mint address
631     df = df[df['Mint Address'] == token_mint_address]
632
633     return df
634
635
636 def token_price(address):
637     API_KEY = d.birdeye
638     url = f"https://public-api.birdeye.so/defi/price?address={address}"
639     headers = {"X-API-KEY": API_KEY}
640     response = requests.get(url, headers=headers)
641     price_data = response.json()
642
643     if price_data['success']:
644         return price_data['data']['value']
645     else:
646         return None
647
648
649 def get_position(token_mint_address):
650     """
651         Fetches the balance of a specific token given its mint address from a DataFrame.
652
653     Parameters:
654         - dataframe: A pandas DataFrame containing token balances with columns ['Mint Address', 'Amount'].
655         - token_mint_address: The mint address of the token to find the balance for.
656
657     Returns:
658         - The balance of the specified token if found, otherwise a message indicating the token is not in the wallet.
659     """

```

#Def Round_down

```
⚡ nice_funcs.py > ⚡ round_down
447     return None
448
449 def get_time_range():
450
451     now = datetime.now()
452     ten_days_earlier = now - timedelta(days=10)
453     time_to = int(now.timestamp())
454     time_from = int(ten_days_earlier.timestamp())
455     #print(time_from, time_to)
456
457     return time_from, time_to
458
459 import math
460 def round_down(value, decimals):
461     factor = 10 ** decimals
462     return math.floor(value * factor) / factor
463
464
465 def get_time_range(days_back=10):
466
467     now = datetime.now()
468     ten_days_earlier = now - timedelta(days=days_back)
469     time_to = int(now.timestamp())
470     time_from = int(ten_days_earlier.timestamp())
471     #print(time_from, time_to)
472
473     return time_from, time_to
474
475
476
477 def get_data(address, days_back_4_data, timeframe):
478
479     time_from, time_to = get_time_range(days_back_4_data)
480
481     url = f"https://public-api.birdeye.so/defi/ohlcv?address={address}&type={timeframe}&time_from={time_from}&time_to={time_to}"
482     #url = f"https://public-api.birdeye.so/defi/ohlcv?address={address}&type={timeframe}&time_from=1706826067&time_to=1706827067"
483     # 1706827420-1707501420
```

#Def get_decimals

```
725
726 def get_decimals(token_mint_address):
727     import requests
728     import base64
729     import json
730     # Solana Mainnet RPC endpoint
731     url = "https://api.mainnet-beta.solana.com/"
732     headers = {"Content-Type": "application/json"}
733
734     # Request payload to fetch account information
735     payload = json.dumps({
736         "jsonrpc": "2.0",
737         "id": 1,
738         "method": "getAccountInfo",
739         "params": [
740             token_mint_address,
741             {
742                 "encoding": "jsonParsed"
743             }
744         ]
745     })
746
747     # Make the request to Solana RPC
748     response = requests.post(url, headers=headers, data=payload)
749     response_json = response.json()
750
751     # Parse the response to extract the number of decimals
752     decimals = response_json['result']['value']['data']['parsed']['info']['decimals']
753     #print(f"Decimals for {token_mint_address[-4:]} token: {decimals}")
754
755     return decimals
756
757 def pnl_close(token_mint_address):
758     ''' this will check to see if price is > sell 1, sell 2, sell 3 and sell accordingly '''
759
```

#Def get_market_sell

```

❷ nice_funcs.py > market_sell
305
306
307 def market_sell(QUOTE_TOKEN, amount, slippage):
308
309     import requests
310     import sys
311     import json
312     import base64
313     from solders.keypair import Keypair
314     from solders.transaction import VersionedTransaction
315     from solana.rpc.api import Client
316     from solana.rpc.types import TxOpts
317     import dontshare.dontshare as d
318
319     print('inside market sell')
320
321     KEY = Keypair.from_base58_string(d.sol_key)
322     SLIPPAGE = slippage # 5000 is 50%, 500 is 5% and 50 is .5%
323     print(f'slippage is {SLIPPAGE}!')
324     #PRIORITY_FEE = PRIORITY_FEE
325     #QUOTE_TOKEN = "So1111111111111111111111111111111111111111111111111111111111111112"
326     # QUOTE_TOKEN = "EPjFwdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v" # usdc
327
328     # token would be usdc for sell orders cause we are selling
329     token = "EPjFwdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"
330
331     http_client = Client(d.ankr_key)
332     #http_client = Client("https://rpc.ankr.com/solana/b0d038027b60cf79aba1de4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")
333
334
335     quote = requests.get(f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}')
336     #print(quote)
337     txRes = requests.post('https://quote-api.jup.ag/v6/swap',
338                           headers={"Content-Type": "application/json"},
339                           data=json.dumps({
340                               "quoteResponse": quote,
341                               "userPublicKey": str(KEY.pubkey()),
342                               "prioritizationFeeLamports": PRIORITY_FEE # or replace 'auto' with your specific lamport value
343
344     # token would be usdc for sell orders cause we are selling
345     token = "EPjFwdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"
346
347     http_client = Client(d.ankr_key)
348     #http_client = Client("https://rpc.ankr.com/solana/b0d038027b60cf79aba1de4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")
349
350
351     quote = requests.get(f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}')
352     #print(quote)
353     txRes = requests.post('https://quote-api.jup.ag/v6/swap',
354                           headers={"Content-Type": "application/json"}, data=json.dumps({
355                               "quoteResponse": quote,
356                               "userPublicKey": str(KEY.pubkey()),
357                               "prioritizationFeeLamports": PRIORITY_FEE # or replace 'auto' with your specific lamport value
358
359     #txRes = requests.post('https://quote-api.jup.ag/v6/swap', headers={"Content-Type": "application/json"}, data=json.dumps({"quoteResponse": quote, "userPublicKey": str(KEY.pubkey())}))
360     #print(txRes)
361     swapTx = base64.b64decode(txRes['swapTransaction'])
362     #print(swapTx)
363     tx1 = VersionedTransaction.from_bytes(swapTx)
364     #print(tx1)
365     tx = VersionedTransaction(tx1.message, [KEY])
366     #print(tx)
367     txId = http_client.send_raw_transaction(bytes(tx), TxOpts(skip_preflight=True)).value
368     print(f"https://solscan.io/tx/{str(txId)}")
369
370     from openai import OpenAI
371     import time
372     import re
373
374     client = OpenAI(api_key=d.openai_key)

```

```

328     # token would be usdc for sell orders cause we are selling
329     token = "EPjFwdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v"
330
331     http_client = Client(d.ankr_key)
332     #http_client = Client("https://rpc.ankr.com/solana/b0d038027b60cf79aba1de4eb52d9b0c0df3a0b7066a68ebfb3edb43be7138ee")
333
334
335     quote = requests.get(f'https://quote-api.jup.ag/v6/quote?inputMint={QUOTE_TOKEN}&outputMint={token}&amount={amount}&slippageBps={SLIPPAGE}')
336     #print(quote)
337     txRes = requests.post('https://quote-api.jup.ag/v6/swap',
338                           headers={"Content-Type": "application/json"}, data=json.dumps({
339                               "quoteResponse": quote,
340                               "userPublicKey": str(KEY.pubkey()),
341                               "prioritizationFeeLamports": PRIORITY_FEE # or replace 'auto' with your specific lamport value
342
343
344     #txRes = requests.post('https://quote-api.jup.ag/v6/swap', headers={"Content-Type": "application/json"}, data=json.dumps({"quoteResponse": quote, "userPublicKey": str(KEY.pubkey())}))
345     #print(txRes)
346     swapTx = base64.b64decode(txRes['swapTransaction'])
347     #print(swapTx)
348     tx1 = VersionedTransaction.from_bytes(swapTx)
349     #print(tx1)
350     tx = VersionedTransaction(tx1.message, [KEY])
351     #print(tx)
352     txId = http_client.send_raw_transaction(bytes(tx), TxOpts(skip_preflight=True)).value
353     print(f"https://solscan.io/tx/{str(txId)}")
354
355     from openai import OpenAI
356     import time
357     import re
358
359     client = OpenAI(api_key=d.openai_key)

```

25_Config file

```
5 ##### main.py configurations #####
6
7 #symbol1 = 'EjmDTt8G3T725eFSV7oWmGD8J848guo3LZ1EB3RfwGSw' # harold
8 #symbol1 = '3psH1Mj1f7yUfaD5gh6Zj7epE8hhrMkMETgv5TshQA4o' # boden
9 symbol1 = 'EQGG5muKhviWmWJwy4Fi9TeeNpJUX7RpHAwkQMnTAyAj' # indian
10 #symbol1 = '6D7NaB2xsLd7cauWu1wKk6KBsJohJmP2qZH9GEfVi5Ui' # shark
11 #symbol1 = 'EKpQGSJtjMFqKZ9KQanSqYXrcF8fBopzLHYxdM65zcjm' # wif
12 #symbol1 = '5LafQUrVco6o7KMz42eqVEJ9LW31StPyGjeeu5sKoMtA' # mumu
13 #symbol1 = '89KX9c3rNoW5sVnwMQvPPFgYKxpABupCSqXUtB1CW5J2'
14 #symbol1 = '5mbK36SZ7J19An8jFochhQS4of8g6BwUjbeCSxBSoWdp' # michi
15 #symbol1 = '2ee2GwuKBEGWUxSwQeSxT3NjG3rak3ULx8khjVUR26Jv' # glub
16 #symbol1 = '3hbKad9n27siYjhgSmQBP5r6C7n2aGYrzS2RKfpw4M8e' #cc
17 #symbol1 = '3hbKad9n27siYjhgSmQBP5r6C7n2aGYrzS2RKfpw4M8e'
18 #symbol1 = 'JCAbmex5ddCmQsJ2sKc1WJPbb1NUhg4eygvZbay2H2r9'
19 #symbol1 = '4o7piaDSo9bGFup7fbti3AsXthgSPnkaignGVU9sNxJV'
20 #symbol1 = 'EZUFNJMJZTBpungQX2czEb9ZyCMjtdzsDGMK4UywDUa1F'
21 #symbol1 = 'AfLLKxJHeb xoLG57iWABV71vRuSKPJLwMWbirMtkq2jD'
22 #symbol1 = '8H2t2Fx nCt9VNmt3ReLP7eQhjzMVxb59Bj3BPsF41oqC'
23 #symbol1 = '3dD6qoZPjAT3fNDVd9CWvWEy3rigJdVP2WdG2Fe23SPd'
24 #symbol1 = 'GJgHsc1HU4ibmzW6oWQr8L2RRT95ATc1BoNuLkp94AwU' # WBS
25 #symbol1 = '5ibZoZneHrus9fThCHPWeH5tHJQ5HC1epicUEUi2pXnG'
26 #symbol1 = '6k33czoudK1Z5YNnWPUzw8uvBrsRjwHLJALERNch5Y4R'
```

```
28
29 # Define the token batch as a list of strings
30 token_batch = [
31     "dekNoN3D8mXa4JHLwTbVXz8aPAyJUkk443UjcSpJKi4",
32     "JD3S1oqnjG5trRTZXplmKnS7LsKsppyFa51rPBBMwogv",
33     "5VgY2RzbB2adNpengMACREuRfePwwZ5L6RqBj3rwDUj3",
34     '6e6rViDzavLRv56nvZye5UofrKDg36mf6dTQrMCoPTW9'
35 ]
36
37 usd_size = 350 # .001 * size = exchange fee $15 -- .015
```

```
usd_size = 350 # .001 * size = exchange fee $15 --- .015
max_usd_order_size = 7 # max usd size to sell in one chunk
tx_sleep = 5 # for entering. closing we spam 3 in a row. hard coded.
slippage = 199 # 5000 = 50% slippage, so 500 = 5% and 50 = .5% slippage
PRIORITY_FEE = 100000 # 200000 is about .035 usd at 150 sol, after a bit of testing 100000 is sufficient and is .02 usd
orders_per_open = 5

# PNL CLOSE
lowest_balance = 1400
target_balance = 3000

# LIQ BOT
long_liq_threshold = 1000
short_liq_threshold = 1000
```