```
import requests from bs4 import BeautifulSoup import json import time import schedule from datetime
import datetime, timedelta from solana.rpc.api import Client from solana.keypair import Keypair from
solana.transaction import Transaction from solana.rpc.types import TxOpts
# Replace with your actual private key securely (recommend using .env in production) PRIVATE_KEY = [...] #
Your Solana private key as list of 64 integers
# Blockchain setup SOLANA_RPC = "https://api.mainnet-beta.solana.com" keypair =
Keypair.from_secret_key(bytes(PRIVATE_KEY)) client = Client(SOLANA_RPC)
# Token and wallet utility functions SOL_MINT = "So11111111111111111111111111111111111111112"
class PortfolioManager:    def __init__(self):        self.holdings = {} # token_address: {buy_price, amount,
time_bought}
   def record_buy(self, token_address, amount, price):        self.holdings[token_address] = {           'buy_price':
price,           'amount': amount,           'time_bought': datetime.utcnow()        }
   def monitor_prices(self, price_feed):        for token, data in list(self.holdings.items()):            current_price =
price_feed.get(token, 0)            if current_price >= 10 * data['buy_price']:                success = sell_token(token,
data['amount'], data['buy_price'], current_price)                if success:                    print(f"✅ Take profit executed
for {token}")                    del self.holdings[token]

def buy_token(token_address, amount_in_sol=1.0, slippage=0.15):     try:        lamports = int(amount_in_sol *
1e9)        print(f"🛒 Buying {amount_in_sol} SOL worth of {token_address} with {int(slippage*100)}%
slippage.")        # Placeholder: Add Jupiter swap integration here.        return True     except Exception as e:
  print(f"❌ Buy failed: {str(e)}")        return False

def sell_token(token_address, amount_held, buy_price, current_price, slippage=0.15):     try:        sell_amount =
amount_held * 0.85        print(f"💰 Selling {sell_amount} of {token_address} at price {current_price:.4f}")        #
Placeholder: Add Jupiter swap integration here.        return True     except Exception as e:        print(f"❌ Sell
failed: {str(e)}")        return False

class SolanaCoinAnalyzer:     def __init__(self):        self.headers = {            'User-Agent': 'Mozilla/5.0'        }
self.token_data = []        self.known_ruggers = {"ABC123...", "DEF456..."} # Populate with real addresses
   def get_solsniffer_score(self, mint_address):        try:            url =
f"https://api.solsniffer.com/audit/{mint_address}"            response = requests.get(url, headers=self.headers,
timeout=10)            if response.status_code == 200:                data = response.json()                return
data.get('score', 0), data.get('audit_passed', False)            return 0, False        except Exception as e:
print(f"SolSniffer error: {e}")            return 0, False
   def is_fake_volume(self, coin):        vol = coin['volume']        liq = coin['liquidity']        if liq > 0 and vol / liq >
10:            return True        if coin['age_hours'] < 2 and vol > 50000:            return True        return False
   def is_known_rugger(self, address):        return address in self.known_ruggers
   def fetch_pump_fun_coins(self):        try:            url = "https://api.pump.fun/coins"            response =
requests.get(url, headers=self.headers, timeout=10)            if response.status_code == 200:                coins =
response.json()                for coin in coins:                    self.token_data.append({                        'source':
'pump.fun',                        'symbol': coin['symbol'],                        'name': coin['name'],                        'address':
coin['mint'],                        'price': coin['price'],                        'liquidity': coin['liquidity'],                        'volume':
coin['volume'],                        'age_hours': (datetime.utcnow() - datetime.strptime(coin['created'], '%Y-%m-
%dT%H:%M:%S.%fZ')).total_seconds() / 3600,                        'creator_address': coin.get('creator', '')
})        except Exception as e:            print(f"Pump.fun error: {e}")
   def fetch_meteora_coins(self):        try:            url = "https://dlmm-api.meteora.ag/pair/all"            response
= requests.get(url, headers=self.headers, timeout=15)            if response.status_code == 200:                pools =
response.json()                for pool in pools:                    if 'SOL' in [pool['tokenA']['symbol'], pool['tokenB']
['symbol']]:                        token = pool['tokenA'] if pool['tokenB']['symbol'] == 'SOL' else pool['tokenB']
   self.token_data.append({                            'source': 'meteora.ag',                            'symbol': token['symbol'],
                 'name': token['name'],                            'address': token['mint'],                            'price':
float(pool['reserveB']) / float(pool['reserveA']) if token == pool['tokenA'] else float(pool['reserveA']) /
float(pool['reserveB']),                            'liquidity': (float(pool['reserveA']) * float(pool['reserveB'])) ** 0.5,
        'volume': float(pool['volume24h']),                            'age_hours': 0,                            'creator_address': '' # If
known                        })        except Exception as e:            print(f"Meteora error: {e}")
   def fetch_letsbonk_coins(self):        try:            url = "https://letsbonk.fun/"            response =
requests.get(url, headers=self.headers, timeout=10)          if response.status_code == 200:          soup =
BeautifulSoup(response.text, 'html.parser')          table = soup.find('table', {'id': 'tokenTable'})          if
table and table.tbody:          for row in table.tbody.find_all('tr'):          cols = row.find_all('td')
      if len(cols) >= 6:          try:          token_link = cols[0].find('a')
token_name = token_link.find('div', class_='token-name').text.strip()          token_symbol =
token_link.find('div', class_='token-symbol').text.strip()          price =
float(cols[1].text.strip().replace('$', ").replace(',', "))          liquidity =
float(cols[2].text.strip().replace('$', ").replace(',', "))          volume =
float(cols[3].text.strip().replace('$', ").replace(',', "))          age_str = cols[5].text.strip().lower()
          if 'h' in age_str:          age_hours = float(age_str.split('h')[0])          elif 'd'
in age_str:          age_hours = float(age_str.split('d')[0]) * 24          else:
      age_hours = 0          self.token_data.append({          'source': 'letsbonk.fun',
          'symbol': token_symbol,          'name': token_name,
'address': token_link['href'].split('/')[-1],          'price': price,          'liquidity': liquidity,
          'volume': volume,          'age_hours': age_hours,
'creator_address': "          })          except Exception as e:
print(f"Parsing letsbonk row failed: {str(e)}")     except Exception as e:     print(f"LetsBonk error: {e}")
  def analyze_coins(self):     if not self.token_data:     return []
    filtered_coins = []     for coin in self.token_data:     sol_score, passed =
self.get_solsniffer_score(coin['address'])     coin['contract_score'] = sol_score     coin['audit_passed'] =
passed     if sol_score < 85:     continue     if self.is_fake_volume(coin):     continue
 if self.is_known_rugger(coin.get('creator_address', ")):     continue     filtered_coins.append(coin)
    if not filtered_coins:     return []
    max_volume = max(c['volume'] for c in filtered_coins)     max_liquidity = max(c['liquidity'] for c in
filtered_coins)     min_age = min(c['age_hours'] for c in filtered_coins if c['age_hours'] > 0) or 1
    for coin in filtered_coins:     vol_score = (coin['volume'] / max_volume) * 0.35 if max_volume > 0 else 0
      liq_score = (coin['liquidity'] / max_liquidity) * 0.30 if max_liquidity > 0 else 0     age_score = (1 -
min(1, coin['age_hours'] / (min_age * 10))) * 0.25     momentum_score = 0.10 # Placeholder
coin['score'] = vol_score + liq_score + age_score + momentum_score
    return sorted(filtered_coins, key=lambda x: x['score'], reverse=True)
  def run_analysis(self):     self.token_data = []     self.fetch_pump_fun_coins()
self.fetch_meteora_coins()     self.fetch_letsbonk_coins()     return self.analyze_coins()

def main():   analyzer = SolanaCoinAnalyzer()   portfolio = PortfolioManager()
  def job():     print(f"\n📈 Running analysis at {datetime.utcnow().isoformat()}...")     top_coins =
analyzer.run_analysis()     print(f"Found {len(top_coins)} candidates.")
    for coin in top_coins[:3]: # Limit buys to top 3     print(f"{coin['symbol']} - Price: ${coin['price']:.6f} |
Score: {coin['score']:.2f}")     success = buy_token(coin['address'], amount_in_sol=1.0)     if success:
      portfolio.record_buy(coin['address'], 1.0, coin['price'])
    # Monitor for take-profit conditions     price_feed = {c['address']: c['price'] for c in analyzer.token_data}
  portfolio.monitor_prices(price_feed)
  job()   schedule.every(15).minutes.do(job)
  while True:     schedule.run_pending()     time.sleep(60)
if __name__ == "__main__":   main()

import requests from bs4 import BeautifulSoup import json import time import schedule from datetime
import datetime, timedelta from solana.rpc.api import Client from solana.keypair import Keypair from
solana.transaction import Transaction from solana.rpc.types import TxOpts # Replace with your actual
private key securely (recommend using .env in production) PRIVATE_KEY = [...] # Your Solana private key as
list of 64 integers # Blockchain setup SOLANA_RPC = "https://api.mainnet-beta.solana.com" keypair =
Keypair.from_secret_key(bytes(PRIVATE_KEY)) client = Client(SOLANA_RPC) # Token and wallet utility
functions SOL_MINT = "So11111111111111111111111111111111111111112" class PortfolioManager:   def
__init__(self):     self.holdings = {} # token_address: {buy_price, amount, time_bought}   def record_buy(self,
token_address, amount, price):     self.holdings[token_address] = {     'buy_price': price,     'amount':
amount,     'time_bought': datetime.utcnow()     }   def monitor_prices(self, price_feed):     for token,
data in list(self.holdings.items()):     current_price = price_feed.get(token, 0)     if current_price >= 10 *
data['buy_price']:     success = sell_token(token, data['amount'], data['buy_price'], current_price)
if success:     print(f"✅ Take profit executed for {token}")     del self.holdings[token] def
buy_token(token_address, amount_in_sol=1.0, slippage=0.15):   try:   lamports = int(amount_in_sol * 1e9)
```

```
    print(f"🛒 Buying {amount_in_sol} SOL worth of {token_address} with {int(slippage*100)}% slippage.")
# Placeholder: Add Jupiter swap integration here.        return True     except Exception as e:        print(f"❌ Buy
failed: {str(e)}")        return False def sell_token(token_address, amount_held, buy_price, current_price,
slippage=0.15):     try:        sell_amount = amount_held * 0.85        print(f"💰 Selling {sell_amount} of
{token_address} at price {current_price:.4f}")        # Placeholder: Add Jupiter swap integration here.        return
True     except Exception as e:        print(f"❌ Sell failed: {str(e)}")        return False class SolanaCoinAnalyzer:
  def __init__(self):        self.headers = {            'User-Agent': 'Mozilla/5.0'        }        self.token_data = []
self.known_ruggers = {"ABC123...", "DEF456..."} # Populate with real addresses     def get_solsniffer_score(self,
mint_address):        try:            url = f"https://api.solsniffer.com/audit/{mint_address}"            response =
requests.get(url, headers=self.headers, timeout=10)            if response.status_code == 200:                data =
response.json()                return data.get('score', 0), data.get('audit_passed', False)            return 0, False
except Exception as e:            print(f"SolSniffer error: {e}")            return 0, False     def is_fake_volume(self,
coin):        vol = coin['volume']        liq = coin['liquidity']        if liq > 0 and vol / liq > 10:            return True        if
coin['age_hours'] < 2 and vol > 50000:            return True        return False     def is_known_rugger(self,
address):        return address in self.known_ruggers     def fetch_pump_fun_coins(self):        try:            url =
"https://api.pump.fun/coins"            response = requests.get(url, headers=self.headers, timeout=10)            if
response.status_code == 200:                coins = response.json()                for coin in coins:
self.token_data.append({                        'source': 'pump.fun',                        'symbol': coin['symbol'],
'name': coin['name'],                        'address': coin['mint'],                        'price': coin['price'],
'liquidity': coin['liquidity'],                        'volume': coin['volume'],                        'age_hours': (datetime.utcnow() -
datetime.strptime(coin['created'], '%Y-%m-%dT%H:%M:%S.%fZ')).total_seconds() / 3600,
'creator_address': coin.get('creator', ")                    })        except Exception as e:            print(f"Pump.fun error:
{e}")    def fetch_meteora_coins(self):        try:            url = "https://dlmm-api.meteora.ag/pair/all"
response = requests.get(url, headers=self.headers, timeout=15)            if response.status_code == 200:
   pools = response.json()                for pool in pools:                    if 'SOL' in [pool['tokenA']['symbol'],
pool['tokenB']['symbol']]:                        token = pool['tokenA'] if pool['tokenB']['symbol'] == 'SOL' else
pool['tokenB']                        self.token_data.append({                            'source': 'meteora.ag',
'symbol': token['symbol'],                            'name': token['name'],                            'address': token['mint'],
         'price': float(pool['reserveB']) / float(pool['reserveA']) if token == pool['tokenA'] else float(pool['reserveA'])
/ float(pool['reserveB']),                            'liquidity': (float(pool['reserveA']) * float(pool['reserveB'])) ** 0.5,
          'volume': float(pool['volume24h']),                            'age_hours': 0,                            'creator_address': " #
If known                        })        except Exception as e:            print(f"Meteora error: {e}")     def
fetch_letsbonk_coins(self):        try:            url = "https://letsbonk.fun/"            response = requests.get(url,
headers=self.headers, timeout=10)            if response.status_code == 200:                soup =
BeautifulSoup(response.text, 'html.parser')                table = soup.find('table', {'id': 'tokenTable'})                if
table and table.tbody:                    for row in table.tbody.find_all('tr'):                        cols = row.find_all('td')
         if len(cols) >= 6:                            try:                                token_link = cols[0].find('a')
token_name = token_link.find('div', class_='token-name').text.strip()                                token_symbol =
token_link.find('div', class_='token-symbol').text.strip()                                price =
float(cols[1].text.strip().replace('$', ").replace(',', "))                                liquidity =
float(cols[2].text.strip().replace('$', ").replace(',', "))                                volume =
float(cols[3].text.strip().replace('$', ").replace(',', "))                                age_str = cols[5].text.strip().lower()
                if 'h' in age_str:                                    age_hours = float(age_str.split('h')[0])                                elif 'd'
in age_str:                                    age_hours = float(age_str.split('d')[0]) * 24                                else:
         age_hours = 0                                self.token_data.append({                                    'source': 'letsbonk.fun',
                  'symbol': token_symbol,                                    'name': token_name,
'address': token_link['href'].split('/')[-1],                                    'price': price,                                    'liquidity': liquidity,
                     'volume': volume,                                    'age_hours': age_hours,
'creator_address': "                                })                            except Exception as e:
print(f"Parsing letsbonk row failed: {str(e)}")        except Exception as e:            print(f"LetsBonk error: {e}")
def analyze_coins(self):        if not self.token_data:            return []        filtered_coins = []        for coin in
self.token_data:            sol_score, passed = self.get_solsniffer_score(coin['address'])
coin['contract_score'] = sol_score            coin['audit_passed'] = passed            if sol_score < 85:
continue            if self.is_fake_volume(coin):                continue            if
self.is_known_rugger(coin.get('creator_address', ")):                continue            filtered_coins.append(coin)
if not filtered_coins:            return []        max_volume = max(c['volume'] for c in filtered_coins)
max_liquidity = max(c['liquidity'] for c in filtered_coins)        min_age = min(c['age_hours'] for c in filtered_coins
if c['age_hours'] > 0) or 1        for coin in filtered_coins:            vol_score = (coin['volume'] / max_volume) * 0.35
if max_volume > 0 else 0            liq_score = (coin['liquidity'] / max_liquidity) * 0.30 if max_liquidity > 0 else 0
```

```
    age_score = (1 - min(1, coin['age_hours'] / (min_age * 10))) * 0.25         momentum_score = 0.10 #
Placeholder          coin['score'] = vol_score + liq_score + age_score + momentum_score        return
sorted(filtered_coins, key=lambda x: x['score'], reverse=True)    def run_analysis(self):       self.token_data = []
     self.fetch_pump_fun_coins()       self.fetch_meteora_coins()       self.fetch_letsbonk_coins()       return
self.analyze_coins() def main():    analyzer = SolanaCoinAnalyzer()    portfolio = PortfolioManager()    def
job():       print(f"\n📈 Running analysis at {datetime.utcnow().isoformat()}...")       top_coins =
analyzer.run_analysis()       print(f"Found {len(top_coins)} candidates.")       for coin in top_coins[:3]: # Limit
buys to top 3          print(f"{coin['symbol']} - Price: ${coin['price']:.6f} | Score: {coin['score']:.2f}")          success
= buy_token(coin['address'], amount_in_sol=1.0)          if success:
portfolio.record_buy(coin['address'], 1.0, coin['price'])       # Monitor for take-profit conditions       price_feed =
{c['address']: c['price'] for c in analyzer.token_data}       portfolio.monitor_prices(price_feed)    job()
schedule.every(15).minutes.do(job)    while True:       schedule.run_pending()       time.sleep(60) if __name__
== "__main__":    main()
```