```
import os
from swarms import Agent
from swarm_models import OpenAIChat
from web3 import Web3
from typing import Dict, Optional, Any
from datetime import datetime
import asyncio
from loguru import logger
from dotenv import load_dotenv
import csv
import requests
import time

BLOCKCHAIN_AGENT_PROMPT = """
You are an expert blockchain and cryptocurrency analyst with deep knowledge of Ethereum markets
and DeFi ecosystems.
You have access to real-time ETH price data and transaction information.

For each transaction, analyze:

1. MARKET CONTEXT
- Current ETH price and what this transaction means in USD terms
- How this movement compares to typical market volumes
- Whether this could impact ETH price

2. BEHAVIORAL ANALYSIS
```

- Whether this appears to be institutional, whale, or protocol movement

- If this fits any known wallet patterns or behaviors

- Signs of smart contract interaction or DeFi activity

3. RISK & IMPLICATIONS

- Potential market impact or price influence

- Signs of potential market manipulation or unusual activity

- Protocol or DeFi risks if applicable

4. STRATEGIC INSIGHTS

- What traders should know about this movement

- Potential chain reactions or follow-up effects

- Market opportunities or risks created

Write naturally but precisely. Focus on actionable insights and important patterns.

Your analysis helps traders and researchers understand significant market movements in real-time."""

```
class EthereumAnalyzer:
    def __init__(self, min_value_eth: float = 100.0):
        load_dotenv()

        logger.add(
            "eth_analysis.log",
            rotation="500 MB",
```

```
    retention="10 days",
    level="INFO",
    format="{time:YYYY-MM-DD at HH:mm:ss} | {level} | {message}",
)

self.w3 = Web3(
    Web3.HTTPProvider(
        "https://mainnet.infura.io/v3/9aa3d95b3bc440fa88ea12eaa4456161"
    )
)
if not self.w3.is_connected():
    raise ConnectionError(
        "Failed to connect to Ethereum network"
    )

self.min_value_eth = min_value_eth
self.last_processed_block = self.w3.eth.block_number
self.eth_price = self.get_eth_price()
self.last_price_update = time.time()

# Initialize AI agent
api_key = os.getenv("OPENAI_API_KEY")
if not api_key:
    raise ValueError(
        "OpenAI API key not found in environment variables"
    )
```

```
model = OpenAIChat(
    openai_api_key=api_key,
    model_name="gpt-4",
    temperature=0.1,
)

self.agent = Agent(
    agent_name="Ethereum-Analysis-Agent",
    system_prompt=BLOCKCHAIN_AGENT_PROMPT,
    llm=model,
    max_loops=1,
    autosave=True,
    dashboard=False,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="eth_agent.json",
    user_name="eth_analyzer",
    retry_attempts=1,
    context_length=200000,
    output_type="string",
    streaming_on=False,
)

self.csv_filename = "ethereum_analysis.csv"
self.initialize_csv()
```

```
def get_eth_price(self) -> float:
    """Get current ETH price from CoinGecko API."""
    try:
        response = requests.get(
            "https://api.coingecko.com/api/v3/simple/price",
            params={"ids": "ethereum", "vs_currencies": "usd"},
        )
        return float(response.json()["ethereum"]["usd"])
    except Exception as e:
        logger.error(f"Error fetching ETH price: {str(e)}")
        return 0.0

def update_eth_price(self):
    """Update ETH price if more than 5 minutes have passed."""
    if time.time() - self.last_price_update > 300:  # 5 minutes
        self.eth_price = self.get_eth_price()
        self.last_price_update = time.time()
        logger.info(f"Updated ETH price: ${self.eth_price:,.2f}")

def initialize_csv(self):
    """Initialize CSV file with headers."""
    headers = [
        "timestamp",
        "transaction_hash",
        "from_address",
```

```
            "to_address",
            "value_eth",
            "value_usd",
            "eth_price",
            "gas_used",
            "gas_price_gwei",
            "block_number",
            "analysis",
        ]

        if not os.path.exists(self.csv_filename):
            with open(self.csv_filename, "w", newline="") as f:
                writer = csv.writer(f)
                writer.writerow(headers)

    async def analyze_transaction(
        self, tx_hash: str
    ) -> Optional[Dict[str, Any]]:
        """Analyze a single transaction."""
        try:
            tx = self.w3.eth.get_transaction(tx_hash)
            receipt = self.w3.eth.get_transaction_receipt(tx_hash)

            value_eth = float(self.w3.from_wei(tx.value, "ether"))

            if value_eth < self.min_value_eth:
```

```
    return None

block = self.w3.eth.get_block(tx.blockNumber)

# Update ETH price if needed
self.update_eth_price()

value_usd = value_eth * self.eth_price

analysis = {
    "timestamp": datetime.fromtimestamp(
        block.timestamp
    ).isoformat(),
    "transaction_hash": tx_hash.hex(),
    "from_address": tx["from"],
    "to_address": tx.to if tx.to else "Contract Creation",
    "value_eth": value_eth,
    "value_usd": value_usd,
    "eth_price": self.eth_price,
    "gas_used": receipt.gasUsed,
    "gas_price_gwei": float(
        self.w3.from_wei(tx.gasPrice, "gwei")
    ),
    "block_number": tx.blockNumber,
}
```

```
            # Check if it's a contract
            if tx.to:
                code = self.w3.eth.get_code(tx.to)
                analysis["is_contract"] = len(code) > 0

                # Get contract events
                if analysis["is_contract"]:
                    analysis["events"] = receipt.logs

            return analysis

        except Exception as e:
            logger.error(
                f"Error analyzing transaction {tx_hash}: {str(e)}"
            )
            return None

    def prepare_analysis_prompt(self, tx_data: Dict[str, Any]) -> str:
        """Prepare detailed analysis prompt including price context."""
        value_usd = tx_data["value_usd"]
        eth_price = tx_data["eth_price"]

        prompt = f"""Analyze this Ethereum transaction in current market context:

Transaction Details:
- Value: {tx_data['value_eth']:.2f} ETH (${value_usd:,.2f} at current price)
```

```
- Current ETH Price: ${eth_price:,.2f}
- From: {tx_data['from_address']}
- To: {tx_data['to_address']}
- Contract Interaction: {tx_data.get('is_contract', False)}
- Gas Used: {tx_data['gas_used']:,} units
- Gas Price: {tx_data['gas_price_gwei']:.2f} Gwei
- Block: {tx_data['block_number']}
- Timestamp: {tx_data['timestamp']}

{f"Event Count: {len(tx_data['events'])} events" if tx_data.get('events') else "No contract events"}

Consider the transaction's significance given the current ETH price of ${eth_price:,.2f} and total USD
value of ${value_usd:,.2f}.
Analyze market impact, patterns, risks, and strategic implications."""

        return prompt

    def save_to_csv(self, tx_data: Dict[str, Any], ai_analysis: str):
        """Save transaction data and analysis to CSV."""
        row = [
            tx_data["timestamp"],
            tx_data["transaction_hash"],
            tx_data["from_address"],
            tx_data["to_address"],
            tx_data["value_eth"],
            tx_data["value_usd"],
```

```
            tx_data["eth_price"],
            tx_data["gas_used"],
            tx_data["gas_price_gwei"],
            tx_data["block_number"],
            ai_analysis.replace("\n", " "),
        ]

        with open(self.csv_filename, "a", newline="") as f:
            writer = csv.writer(f)
            writer.writerow(row)

    async def monitor_transactions(self):
        """Monitor and analyze transactions one at a time."""
        logger.info(
            f"Starting transaction monitor (minimum value: {self.min_value_eth} ETH)"
        )

        while True:
            try:
                current_block = self.w3.eth.block_number
                block = self.w3.eth.get_block(
                    current_block, full_transactions=True
                )

                for tx in block.transactions:
                    tx_analysis = await self.analyze_transaction(
```

```
            tx.hash
        )

        if tx_analysis:
            # Get AI analysis
            analysis_prompt = (
                self.prepare_analysis_prompt(tx_analysis)
            )
            ai_analysis = self.agent.run(analysis_prompt)
            print(ai_analysis)

            # Save to CSV
            self.save_to_csv(tx_analysis, ai_analysis)

            # Print analysis
            print("\n" + "=" * 50)
            print("New Transaction Analysis")
            print(
                f"Hash: {tx_analysis['transaction_hash']}"
            )
            print(
                f"Value: {tx_analysis['value_eth']:.2f} ETH (${tx_analysis['value_usd']:,.2f})"
            )
            print(
                f"Current ETH Price: ${self.eth_price:,.2f}"
            )
```

```
                print("=" * 50)
                print(ai_analysis)
                print("=" * 50 + "\n")

            await asyncio.sleep(1)  # Wait for next block

        except Exception as e:
            logger.error(f"Error in monitoring loop: {str(e)}")
            await asyncio.sleep(1)


async def main():
    """Entry point for the analysis system."""
    analyzer = EthereumAnalyzer(min_value_eth=100.0)
    await analyzer.monitor_transactions()


if __name__ == "__main__":
    print("Starting Ethereum Transaction Analyzer...")
    print("Saving results to ethereum_analysis.csv")
    print("Press Ctrl+C to stop")
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\nStopping analyzer...")
```