
BLOCKCYPHER
TRADE ANAYTX

Block Cypher Trade Analytx
Autonomous AI Trading & Staking System on Ethereum

Executive Summary

BlockCypher Trade Analytx is an autonomous AI-driven trading and staking system operating natively on the Ethereum blockchain, designed to independently manage digital assets without continuous human intervention. The system functions as a self-governing on-chain agent that analyzes market conditions, executes trades, reallocates capital, and simultaneously stakes assets to generate continuous yield.
Unlike conventional trading bots or passive staking solutions, BlockCypher Trade Analytx treats trading and staking as a unified capital strategy. Capital allocation is dynamically adjusted based on market structure, volatility regimes, on-chain activity, and sector-specific momentum rather than fixed exposure rules or static portfolios.
The objective is not short-term speculation, but the creation of a self-learning capital allocation engine that:

- actively responds to changing market conditions,
- minimizes idle liquidity,
- and continuously optimizes risk-adjusted returns through adaptive decision-making.

Core Concept: Autonomous Capital Management

BlockCypher Trade Analytx is built around the principle that capital should remain productive under all market conditions.
Instead of separating active trading from passive yield generation, the system continuously evaluates whether capital should:

- remain liquid for active trading,
- be allocated to staking and yield-bearing mechanisms,
- be rotated across sectors,
- or be temporarily de-risked during unfavorable market regimes.

Human discretion is intentionally minimized to reduce emotional bias, delayed reactions, and inconsistent execution. All decisions are governed by predefined risk constraints, adaptive learning logic, and verifiable execution rules.

Strategic Focus: Layer, AI & Gaming Altcoins

The system deliberately focuses on innovation-driven altcoin sectors, rather than attempting to cover the entire cryptocurrency market. These segments were selected due to their structural characteristics:

- accelerated technological development,
- strong dependency on developer and community activity,
- measurable on-chain usage patterns,
- and historically asymmetric upside during expansion phases.

Primary target sectors

Layer Infrastructure
Protocols related to scaling, modular blockchain design, rollups, execution layers, and settlement infrastructure.
Artificial Intelligence Protocols
Decentralized AI networks, autonomous agents, compute coordination layers, and data marketplaces.
Gaming & Interactive Economies
On-chain gaming, digital ownership models, virtual economies, and user-driven blockchain activity.
Assets within these sectors are not statically defined. They are continuously evaluated, ranked, and reweighted as market conditions evolve.

Dynamic Asset Selection & Sector Rotation

BlockCypher Trade Analytx applies a multi-factor evaluation framework to all eligible assets within the targeted sectors. Ranking and allocation decisions incorporate:

- on-chain activity metrics,
- liquidity depth and volatility profiles,
- relative performance within sector benchmarks,
- developer activity and update frequency,
- ecosystem and community engagement indicators.

Capital is rotated dynamically within and across sectors, allowing the system to adapt to emerging trends while gradually reducing exposure to declining or structurally weakening protocols.

Integrated Trading & Staking Logic

A defining feature of BlockCypher Trade Analytx is its dual execution framework, where trading and staking operate as complementary components of a single strategy.

- During high-volatility or trend-driven phases, capital remains predominantly liquid to enable rapid repositioning and active trade execution.
- During range-bound, accumulation, or low-volatility environments, a larger share of capital is allocated to staking and yield-generating mechanisms.
- During sector-specific momentum phases, capital is rotated toward outperforming assets within the Layer, AI, or Gaming domains.

Staking decisions are evaluated not only by yield, but also by lock-up duration, liquidity risk, opportunity cost, and interaction with overall portfolio volatility.

Adaptive Learning & Continuous Optimization

The system incorporates adaptive learning mechanisms that allow it to evolve alongside changing market conditions. Rather than relying on static parameters, BlockCypher Trade Analytx continuously evaluates historical outcomes, performance deviations, and regime shifts.

Key objectives of the learning layer include:

- reducing overfitting to historical patterns,
- adjusting sensitivity to new volatility regimes,
- recalibrating risk thresholds,
- and improving long-term capital efficiency.

The system assumes markets are probabilistic and non-deterministic. No single indicator, model, or signal governs decisions.

Data Scope & Historical Training Framework

BlockCypher Trade Analytx is trained and calibrated using an extensive historical dataset covering multiple full market cycles within the cryptocurrency ecosystem.

The system incorporates market data from the 2017 and 2021 bull market cycles, including complete transitions from accumulation to expansion, distribution, and subsequent market contraction. These periods were deliberately selected due to their distinct structural characteristics, extreme volatility regimes, and diverse behavioral patterns across different asset classes.

In addition, the training and analysis framework includes all available historical chart data up to 10 January 2024 for every coin and token that has been listed on CoinMarketCap, including assets that are no longer actively traded or have since lost market relevance.

By including both successful and failed projects, the system reduces survivorship bias and learns from:

- high-growth expansion phases,
- speculative excess,
- prolonged drawdowns,
- and complete market exits.

This broad historical scope enables the AI agent to contextualize current market conditions within a multi-cycle, probabilistic framework rather than relying on isolated or recent data samples.

Design Philosophy & Constraints

BlockCypher Trade Analytx is not positioned as a guaranteed-profit system. It is designed under the assumption that financial markets are adaptive, competitive, and structurally evolving.

Accordingly:

- capital exposure is bounded by predefined risk limits,
- transparency and auditability are prioritized,
- execution logic is verifiable where possible,
- and failure modes are explicitly considered.

The system is designed to operate with discipline, consistency, and adaptability rather than predictive certainty.

Positioning

BlockCypher Trade Analytx represents an autonomous capital management layer at the intersection of:

- decentralized finance,
- artificial intelligence,
- and on-chain economic coordination.

Its role is to execute capital strategies with machine-level consistency and speed while remaining grounded in transparent, auditable, and risk-aware design principles.

In a world where the confluence of technology and finance is ever-evolving, the needfor advanced tools and scripts that can navigate the complex landscape of
cryptocurrency is paramount. The following text delves into the creation andimplementation of sophisticated scripts that serve a myriad of purposes in the realmof digital currencies.

From fetching historcal data and analyzing market trends to integrating cutting-edgemachine learning models, these scripts are the backbone of modern crypto tradingstrategies. They bridge the gap between raw data and actionable insights, using avariety of technical indicators, trading theories, and data sources to provide acomprehensive picture of the market.

Whether it's processing live metrics of the Ethereum blockchain, tracking transactionvolume and smart contract interactions, or analyzing social media for sentiment andtrends, these tools are designed to give traders, analysts, and enthusiasts an edge inthe highly competitive cryptocurrency space.

The intersection of Wyckof's theories, deep learning, and blockchain metrics createsa multifaceted approach to trading and market analysis. Each script embodies astrategic component of this approach, from the precise calculations of movingaverages and RSI to the predictive capabilities of LSTM networks.

As the industry grows and diversifies into NFTs, GameFi, Play-to-Earn, DeFi, and Al,the need for continuous innovation in data collection, integration, and analysisbecomes paramount. This text serves as a guide for those who aspire to master theart of cryptocurrency trading and analysis, offering a blueprint for building theadvanced tools required to thrive in this dynamic field.

Creating an indicator script

This script will fetch historical price data for a cryptocurrency pair from Binance,calculate the Simple Moving Average (SMA), Exponential Moving Average (EMA),and Relative Strength Index (RSI).

```
# Binance API for historical data
symbol="BTCUSDT"
interval="1d"
limit="100"  # Adjust based on the required period for your calculation

# Fetch historical candlestick data from Binance
data=$(curl -s "https://api.binance.com/api/v3/klines?symbol=${symbol}&

# Calculate Simple Moving Average (SMA)
calculate_sma() {
    echo $data | jq -r '.[].4' | awk '{sum+=$1} END {print sum/NR}'
}

# Calculate Exponential Moving Average (EMA)
calculate_ema() {
    period=10  # Example for a 10-day EMA
    multiplier=$(echo "2 / ($period + 1)" | bc -l)
    prices=$(echo $data | jq -r '.[].4')
    ema=0
    readarray -t price_array <<< "$prices"
    for ((i=0; i<${#price_array[@]}; i++)); do
        if (( i == 0 )); then
            ema=${price_array[i]}
        else
            ema=$(echo "$ema * (1 - $multiplier) + ${price_array[i]} *
        fi
    done
    echo $ema
}
```

```
# Calculate Relative Strength Index (RSI)
calculate_rsi() {
    period=14  # Standard period for RSI
    gains=0
    losses=0
    prices=$(echo $data | jq -r '.[].4')
    readarray -t price_array <<< "$prices"
    for ((i=1; i<=$period; i++)); do
        delta=$(echo "${price_array[i]} - ${price_array[i-1]}" | bc -l)
        if (( $(echo "$delta > 0" | bc -l) )); then
            gains=$(echo "$gains + $delta" | bc -l)
        else
            losses=$(echo "$losses + ${delta#-}" | bc -l)
        fi
    done
    avg_gain=$(echo "$gains / $period" | bc -l)
    avg_loss=$(echo "$losses / $period" | bc -l)
    rs=$(echo "$avg_gain / $avg_loss" | bc -l)
    rsi=$(echo "100 - (100 / (1 + $rs))" | bc -l)
    echo $rsi
}

# Output the calculated indicators
echo "Simple Moving Average (SMA):"
calculate_sma
echo "Exponential Moving Average (EMA):"
calculate_ema
echo "Relative Strength Index (RSI):"
calculate_rsi
```

Script for implementing Wyckoff Analysis

The script will attempt to identify potential accumulation or distribution phases basedon volume and price movements, which are central to Wyckoff's theories. We'll fetchhistorical data from Binance and other exchanges, then analyze volume and pricetrends to suggest which phase the market might be in.

```
# Define variables
symbol="BTCUSDT"
interval="1d"
limit="90"  # Adjust based on the required period for your calculations

# Fetch historical candlestick data from Binance
data=$(curl -s "https://api.binance.com/api/v3/klines?symbol=${symbol}&

# Analyze Accumulation and Distribution Phases based on Wyckoff's Theor
wyckoff_analysis() {
    readarray -t volumes <<< "$(echo $data | jq -r '.[].5')"
    readarray -t closes <<< "$(echo $data | jq -r '.[].4')"

    # Variables for trend detection
    increasing_volume_days=0
    decreasing_volume_days=0
    previous_volume=0
    previous_close=0
    trend=""

    for ((i=0; i<${#volumes[@]}; i++)); do
        current_volume=${volumes[i]}
        current_close=${closes[i]}

        # Check if volume is increasing or decreasing
        if (( $(echo "$current_volume > $previous_volume" | bc -l) ));
            ((increasing_volume_days++))
            decreasing_volume_days=0
        elif (( $(echo "$current_volume < $previous_volume" | bc -l) ))
            ((decreasing_volume_days++))
            increasing_volume_days=0
        fi
```

```
        # Analyze price movement
        if (( $(echo "$current_close > $previous_close" | bc -l) )); th
            # If price and volume are increasing, it might indicate Acc
            if (( increasing_volume_days > 3 )); then
                trend="Accumulation"
            fi
        elif (( $(echo "$current_close < $previous_close" | bc -l) ));
            # If price is falling but volume is increasing, it might in
            if (( increasing_volume_days > 3 )); then
                trend="Distribution"
            fi
        fi

        previous_volume=$current_volume
        previous_close=$current_close
    done

    echo "Market Trend based on Wyckoff's Theory: $trend"
}
```

Volume and Price Analysis: The script attempts to identify trends in volume andprice changes. An increasing trend in volume, coupled with price increases, mightsuggest an accumulation phase, whereas increasing volume and decreasing pricescould suggest a
distribution phase.

Intergrating deep learning into Data Collection

This process aims to incorporate trading indicators (such as MA, EMA, RSI) andstrategies (like Wyckoff's principles) into a cohesive system that can learn and adaptfrom ongoing market information.

Step 1: Data Collection

First, gather historical data for BTCUSD. You can use Binance's API for this purpose.This example assumes you have APl access and focuses on collecting close pricesand volumes, which are critical for calculating various trading indicators.

```
import requests
import pandas as pd

def fetch_historical_data(symbol="BTCUSDT", interval="1d", lookback="1
    # Construct the API URL
    endpoint = f"https://api.binance.com/api/v3/klines"
    params = {
        "symbol": symbol,
        "interval": interval,
        "limit": 1000,  # Adjust based on your needs
    }
    response = requests.get(endpoint, params=params)
    data = response.json()

    # Convert to DataFrame
    df = pd.DataFrame(data, columns=["Open Time", "Open", "High", "Low"
    df["Close"] = df["Close"].astype(float)
    df["Volume"] = df["Volume"].astype(float)
    df["Open Time"] = pd.to_datetime(df["Open Time"], unit='ms')
    df.set_index("Open Time", inplace=True)

    return df[["Close", "Volume"]]
```

Step 2: Feature Engineering

Calculate indicators based on historical data. This involves transforming the raw datainto meaningful inputs (features) for your model.

```
def add_technical_indicators(df):
    # Moving Averages
    df['MA10'] = df['Close'].rolling(window=10).mean()
    df['MA50'] = df['Close'].rolling(window=50).mean()

    # Exponential Moving Average
    df['EMA10'] = df['Close'].ewm(span=10, adjust=False).mean()
    df['EMA50'] = df['Close'].ewm(span=50, adjust=False).mean()

    # Relative Strength Index (RSI)
    delta = df['Close'].diff(1)
    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
    rs = gain / loss
    df['RSI'] = 100 - (100 / (1 + rs))

    # Volume (as is, but could be processed into more meaningful featur
    # Other features can be added here based on Wyckoff's principles or

    # Clean NaN values
    df.dropna(inplace=True)
    return df
```

Step 3: Model Development

For a deep learning model, you might choose an LSTM (Long Short-Term Memory)network due to its ability to remember long-term dependencies, which is useful infinancial time series forecasting.

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

def build_lstm_model(input_shape):
    model = Sequential([
        LSTM(units=50, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        LSTM(units=50, return_sequences=False),
        Dropout(0.2),
        Dense(units=25),
        Dense(units=1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Step 4: Data Preprocessing

Before training, normalize your data and split it into training and testing datasets. Thisstep is crucial for effective model learning.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

def preprocess_data(df, feature_columns, label_column):
    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(df[feature_columns])

    # Create the training and testing data, labels
    X = []
    y = df[label_column].values

    for i in range(60, len(scaled_data)):
        X.append(scaled_data[i-60:i, :])

    X, y = np.array(X), np.array(y[60:])

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

    return X_train, X_test, y_train, y_test, scaler
```

Step 5:Training

Train the model with the preprocessed data. Here, you define the epochs and batchsize based on your experimental tuning.

```
def train_model(X_train, y_train):
    model = build_lstm_model((X_train.shape[1], X_train.shape[2]))
    model.fit(X_train, y_train, epochs=100, batch_size=32, validation_s
    return model
```

Step 6: Prediction and Strategy Execution

After training, use your model to make predictions. Based on these predictions andthe application of trading strategies (like those derived from Wyckoff's analysis), youcan make informed trading decisions

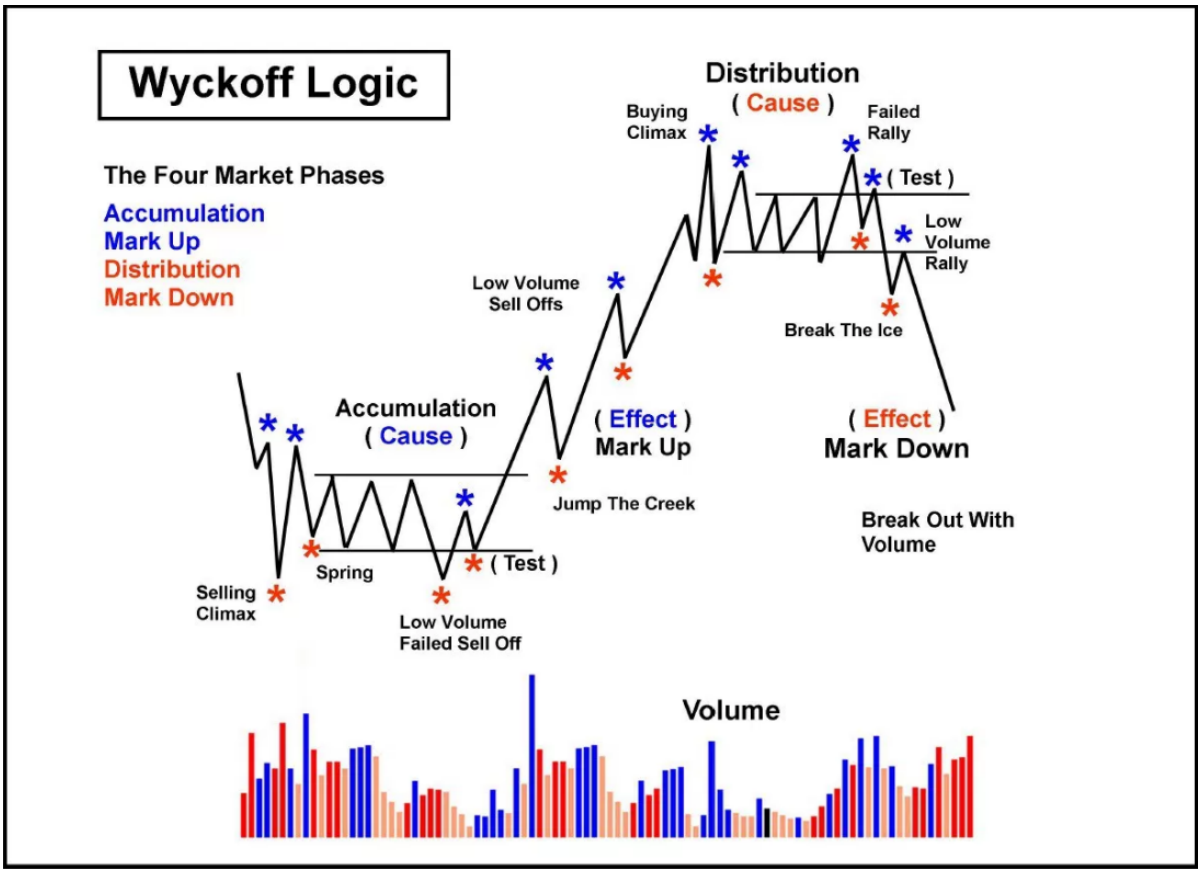
```
def make_predictions(model, X_test, scaler):
    predictions = model.predict(X_test)
    # Inverse transform if your target variable was scaled
    predictions = scaler.inverse_transform(predictions)
    return predictions

# After predictions, compare with actual prices, apply your trading log
```

Integrating with Trading Indicators and Wyckoff's Theories

Data Collection and Feature Engineering: The integration starts from the Verybeginning, where you collect data and engineer features based on traditionalindicators and potentially patterns indicative of Wyckoff's phases.

Model Input: Use these features as inputs to your LSTM model, allowing the modelto learn from both price movements and the theoretical underpinnings of marketpsychology.

Continuous Learning: Regularly update your model with new data, recalculatingindicators and retraining to adapt to new market conditions.



Creating a trading script for example with the Binance API that utilizes technical indicatorssuch as the Exponential Moving Average (EMA), Simple Moving Average (MA), MovingAverage Convergence Divergence (MACD), and Relative Strength Index (RSI) across bothhourly and weekly time frames.

API Setup & Fetching Historical Data

To analyze and generate signals, you need historical price data for the coins you're interestedin. Binance API provides this data.

```
from binance.client import Client

# Replace these with your actual API keys
api_key = 'YOUR_API_KEY'
api_secret = 'YOUR_SECRET_KEY'

client = Client(api_key, api_secret)

# Function to fetch historical data
def get_historical_data(symbol, interval, start_str):
    klines = client.get_historical_klines(symbol, interval, start_str)
    return klines

# Example: Fetch hourly data for BTCUSDT
historical_data_hourly = get_historical_data('BTCUSDT', Client.KLINE_IN
# Example: Fetch weekly data for BTCUSDT
historical_data_weekly = get_historical_data('BTCUSDT', Client.KLINE_IN
```

Calculating Indicators

```
import talib
import numpy as np
import pandas as pd

# Convert data to DataFrame
df_hourly = pd.DataFrame(historical_data_hourly, columns=['timestamp',
df_hourly['close'] = pd.to_numeric(df_hourly['close'])

# Calculate indicators for hourly data
df_hourly['EMA14'] = talib.EMA(df_hourly['close'], timeperiod=14)
df_hourly['SMA14'] = talib.SMA(df_hourly['close'], timeperiod=14)
df_hourly['MACD'], df_hourly['MACD_signal'], _ = talib.MACD(df_hourly['
df_hourly['RSI14'] = talib.RSI(df_hourly['close'], timeperiod=14)

# Repeat the calculation for weekly data as needed
```

Defining Signal Logic

The logic for generating buy or sell signals depends on your trading strategy. For example:

Buy Signal: A potential buy signal could be when the short-term EMA crosses above thelong-term EMA, and the RSI is below 30, indicating that the asset may be oversold.

Sell Signal: A potential sell signal could be when the short-term EMA crosses below thelong-term EMA,and the RSI is above 70, indicating that the asset may be overbought.

Executing Trades Based on Signals

```
# Example of executing a trade (pseudo-code)
for index, signal in buy_signals.iteritems():
    if signal:  # If buy signal is True
        # Place a buy order (simplified)
        order = client.order_limit_buy(
            symbol='BTCUSDT',
            quantity=0.01,  # Define quantity according to your strateg
            price='current_market_price'  # You need to fetch the curre
        )
```

Creating an Algorythm for comparing and indetifying metrics and trends in the blockchain industry, divided into groups of NFT, GameFi, Play-to-Earn, DeFi, and Al, using data sources like GitHub, Twitter, and Discord:

Data Collection: Set up data pipelines to continuously gather data from various sources like GitHub (for code commits and updates), Twitter (for tweets, retweets, and sentiment analysis), and Discord (for chat messages and community activity).Consider using APIs or web scraping techniques.

Data Integration: Integrate the collected data into a central database or data warehouse, ensuring that it is properly structured and indexed for efficient querying.

Data Preprocessing: Clean and preprocess the data, which may include handlingmissing values, removing duplicates, and normalizing text data for sentimentanalysis.

Feature Extraction: Extract relevant features from the data, such as daily counts ofcode commits, tweet volumes, and sentiment scores. You may also want toincorporate additional metrics like market capitalization, trading volumes, and tokenprices.

Sentiment Analysis (for Twitter): Utilize natural language processing (NLP)techniques to perform sentiment analysis on Twitter data to gauge market sentimentfor each category.

Metric Calculation: Calculate various metrics specific to each category. Thesemetrics might include averages, growth rates, volatility measures, and more. Developa weighting system to assign importance to each metric based on its relevance to thecategory.

Trend Analysis: Implement time-series analysis techniques to identfy trends andpatterns in the data. This could involve using statistical methods, moving averages,and other time-series forecasting techniques.

Visualization and Dashboarding: Create interactive dashboards and visualizationsusing tools like Tableau, Power Bl, or custom-built web applications. Thesedashboards should provide real-time insights into the metrics and trends for eachcategory.

Machine Learning Models (Optional): Ising machine learning models for predictiveanalytics, anomaly detection, or clustering to gain deeper insights into the data.

Continuous Monitoring and Updates: Ensure that the system is set up forcontinuous monitoring and updates to keep pace with the rapidly changing
cryptocurrency industry.

User Interface (Ul) Design: Design user-friendly interfaces for stakeholders tointeract with the data and insights generated by the algorithm.

Security and Data Privacy: Implement robust security measures to protect thecollected data and ensure compliance with data privacy regulations. Performance

Optimization: Continuously optimize the system's performance, including dataprocessing speed, storage, and scalability.

Developing a comprehensive Python script addressing two main objectives:

Processing Live Metrics of the Ethereum Blockchain: The script will fetch live datasuch as the current price of Ethereum, gas fees, and block time.
Tracking the Number of Transactions and Smart Contract Interactions on Ethereum: It will capture the number of transactions in the latest blocks and interactions withselected smart contracts.

Part 1: Processing Live Metrics of the Ethereum Blockchain

```
import requests

ETHERSCAN_API_KEY = 'YOUR_ETHERSCAN_API_KEY'
INFURA_PROJECT_ID = 'YOUR_INFURA_PROJECT_ID'

def get_ethereum_metrics():
    etherscan_url = f"https://api.etherscan.io/api?module=stats&action=
    eth_price_response = requests.get(etherscan_url).json()

    if eth_price_response['status'] == '1':
        eth_price = eth_price_response['result']['ethusd']
        print(f"Current Ethereum Price: ${eth_price}")
    else:
        print("Error fetching Ethereum price")

    gas_oracle_url = f"https://api.etherscan.io/api?module=gastracker&a
    gas_response = requests.get(gas_oracle_url).json()

    if gas_response['status'] == '1':
        fast_gas = gas_response['result']['FastGasPrice']
        print(f"Fast Gas Price: {fast_gas} Gwei")
    else:
        print("Error fetching gas prices")

if __name__ == "__main__":
    get_ethereum_metrics()
```

Part 2: Tracking Transactions and Smart Contract Interactions

For this part, we'"l use Web3.py, a Python library that facilitates interaction with theEthereum blockchain.

```
from web3 import Web3

w3 = Web3(Web3.HTTPProvider(f"https://mainnet.infura.io/v3/{INFURA_PROJ

def get_latest_transactions():
    latest_block = w3.eth.get_block('latest')
    print(f"Latest Block Number: {latest_block.number}")
    transactions = latest_block.transactions[:10]  # Get the first 10 t
    for tx_hash in transactions:
        tx = w3.eth.get_transaction(tx_hash)
        print(f"Transaction {tx.hash.hex()} | From: {tx['from']} To: {t

def monitor_contract_interactions(contract_address):
    latest_block = w3.eth.get_block('latest')
    block_number = latest_block.number
    filter = w3.eth.filter({'fromBlock': block_number-100, 'toBlock': '
    entries = filter.get_all_entries()
    print(f"Found {len(entries)} interactions with contract {contract_a

if __name__ == "__main__":
    get_latest_transactions()
    monitor_contract_interactions('CONTRACT_ADDRESS_HERE')  # Replace w
```

This script offers a comprehensive overview of interacting with the Ethereum blockchain by fetching both general live metrics and specific transaction data.

Analysing and comparing Google trends

Region-specific analysis to see how nterest varies by geography.
Category filtering to ensure we focus on financial/investment trends.
Monthly resolution for finer granularity over time.
Comparison of related queries to dentify other trending topics associated withthese cryptocurrencies.

Script for Detailed Comparison of "Bitcoin"vs. "Ethereum"

```
from pytrends.request import TrendReq
import matplotlib.pyplot as plt
import pandas as pd

# Initialize pytrends with English language setting
pytrends = TrendReq(hl='en-US')

# Define the keywords for comparison
keywords = ["Bitcoin", "Ethereum"]

# Set the timeframe from the beginning of 2015 to present
timeframe = '2015-01-01 today'

# Specify the category for cryptocurrencies (can be found on Google Tre
category = 7  # This is an example; the actual category ID for finance/

# Initialize a DataFrame to hold the collected data
trends_data = pd.DataFrame()

for keyword in keywords:
    # Retrieve the data with monthly resolution
    pytrends.build_payload([keyword], timeframe=timeframe, cat=category
    data = pytrends.interest_over_time()

    if not data.empty:
        trends_data[keyword] = data[keyword]
```

```
# Visualize the data with monthly resolution
plt.figure(figsize=(15, 7))
for keyword in keywords:
    if keyword in trends_data.columns:
        plt.plot(trends_data.index, trends_data[keyword], label=keyword

plt.title('Detailed Google Trends Comparison: Bitcoin vs. Ethereum (Sin
plt.xlabel('Year')
plt.ylabel('Interest Over Time')
plt.legend()
plt.grid(True)
plt.show()

# Additionally, analyze top and rising related queries for both keyword
for keyword in keywords:
    related_queries = pytrends.related_queries()
    top_queries = related_queries[keyword]['top']
    rising_queries = related_queries[keyword]['rising']
    print(f"Top related queries for {keyword}:\n", top_queries)
    print(f"Rising related queries for {keyword}:\n", rising_queries)
```

Enhancements Explained:

Region-Specific Analysis: This script is set up for a global analysis but can easilybe adapted to analyze trends in specific regions by setting the geo parameter inbuild_payload to the appropriate country/region code (e.g., US for the United States).

Category Filtering: By specifying a category, we focus our analysis on the contextmost relevant to cryptocurrencies, which might help filter out noise from unrelatedtrends that happen to share a keyword.

Monthly Resolution: This provides a more granular look at how interest haschanged over time, allowing for the identification of more subtle trends or reactions tospecific events.

Related Queries Analysis: Understanding related top and rising queries can offerinsights into what specific aspects of "Bitcoin" and Ethereum" people are interested in, which might indicate shifts in the public's perception or interest areas over time.