

section1

Merge DataFrames on ‘Date’

```
[ ] def merge_dataframes_on_date(dataframes):
    """
    Merge multiple DataFrames on the 'Date' column.
    Assumes all DataFrames have a 'Date' column.
    """
    if not dataframes:
        return pd.DataFrame() # Return an empty DataFrame if no dataframes are provided

    merged_df = dataframes[0] # Start with the first DataFrame
    for df in dataframes[1:]:
        merged_df = pd.merge(merged_df, df, on='Date', how='outer') # Outer join to keep all dates
    return merged_df.set_index('Date') # Set 'Date' as the index for the final DataFrame
```

این تابع یک لیست از DataFrame‌های Pandas را بر اساس ستون 'Date' با استفاده از یک پیوند بیرونی (outer) ترکیب می‌کند تا تمام تاریخ‌ها از همه DataFrame‌ها شامل شوند. روش کار به صورت زیر است:

- بررسی خالی بودن لیست: اگر هیچ DataFrame‌ ارائه نشده باشد، یک DataFrame خالی بر می‌گرداند.
- شروع ترکیب: با اولین DataFrame در لیست شروع می‌کند.
- ترکیب متوالی DataFrame‌های دیگر: هر DataFrame بعدی را بر اساس ستون 'Date' ترکیب می‌کند، به طوری که تمام تاریخ‌ها حفظ شوند.
- تنظیم 'Date' به عنوان شاخص: پس از ترکیب، ستون 'Date' را به عنوان شاخص DataFrame می‌کند تا دستکاری و تحلیل داده‌ها آسان‌تر شود.

Fetch Cryptocurrency Data

```
def fetch_crypto_data(tickers, start_date, end_date):
    """
    Fetch historical price data for a list of cryptocurrencies.
    Returns a list of DataFrames, each with 'Date' as a column.
    """
    dataframes = []
    for ticker in tickers:
        try:
            print(f"Fetching data for {ticker}...")
            crypto_data = yf.download(ticker, start=start_date, end=end_date, progress=False)
            if not crypto_data.empty:
                # Keep only the 'Close' column and reset index to include 'Date'
                crypto_data = crypto_data[['Close']].reset_index()
                # Rename 'Close' column to the ticker name
                crypto_data.rename(columns={'Close': ticker}, inplace=True)
                dataframes.append(crypto_data)
        except Exception as e:
            print(f"Could not fetch data for {ticker}: {e}")
    return dataframes
```

این تابع داده‌های تاریخی قیمت را برای یک لیست از ارز‌های دیجیتال دریافت می‌کند و یک لیست از DataFrame‌ها را بر می‌گرداند که هر کدام شامل ستون 'Date' هستند.

- یک لیست خالی به نام `dataframes` ایجاد می‌کند.
- برای هر نماد (`ticker`) در `yf.download()` از کتابخانه `yf` از ستون 'Close' دانلود کند.
 - تلاش می‌کند داده‌ها را با استفاده از `yf.download()` از کتابخانه `yf` دانلود کند.
 - اگر داده‌ها خالی نباشند:
 - فقط ستون 'Close' را نگه می‌دارد و شاخص را ریست می‌کند تا 'Date' به عنوان یک ستون ظاهر شود.
 - نام ستون 'Close' را به نام نماد (`ticker`) تغییر می‌دهد.

اضافه می‌کند.

اگر خطای رخ دهد، پیام خطأ چاپ می‌شود.

در نهایت، لیست `DataFrame` را برمی‌گرداند.

Perform ADF Test

```
def adf_test(series):
    """
    Perform the Augmented Dickey-Fuller test on a time series.
    Returns the p-value, test statistic, and critical values.
    """
    result = adfuller(series, autolag='AIC')
    p_value = result[1]
    test_stat = result[0]
    critical_values = result[4]
    return p_value, test_stat, critical_values
```

این تابع آزمون دیکی-فولر تقویت شده (Augmented Dickey-Fuller) را بر روی یک سری زمانی انجام می‌دهد و مقدار `p-value`، آماره آزمون و مقادیر بحرانی را برمی‌گرداند.

● اجرای آزمون ADF با استفاده از تابع `adf_test`:

● آزمون را با انتخاب خودکار بهترین `autolag='AIC'`:

● تعداد وقایع بر اساس معیار اطلاعاتی آکائیک (AIC) انجام می‌دهد.

● استخراج نتایج مهم از خروجی آزمون:

● آماره آزمون دیکی-فولر: `test_stat`.

● مقدار احتمال مربوط به آماره آزمون: `p_value`.

● مقادیر بحرانی آزمون برای سطوح اطمینان مختلف (مانند 1٪ و 10٪).

Analyze Stationarity

```
def analyze_stationarity(data):
    """
    Analyze stationarity for each cryptocurrency in the dataset.
    """
    stationary_pvalue = []
    stationary_teststat = []
    for ticker in data.columns:
        series = data[ticker].dropna() # Drop missing values for the series
        if len(series) > 0: # Ensure the series is not empty
            p_value, test_stat, critical_values = adf_test(series)
            # Check stationarity based on p-value
            if p_value < 0.05:
                stationary_pvalue.append((ticker,p_value))
            # Check stationarity based on test statistic
            if test_stat < critical_values['1%']:
                stationary_teststat.append((ticker,p_value, f'{test_stat} < {critical_values["1%"]}') )
    return stationary_pvalue, stationary_teststat
```

این تابع برای هر ارز دیجیتال در مجموعه داده، ایستایی (stationarity) سری زمانی آن را تحلیل می‌کند.

● دو لیست خالی به نام‌های `stationary_pvalue` و `stationary_teststat` برای ذخیره نتایج ایجاد می‌کند.

● برای هر نماد (ticker) در ستون‌های `data`:

● سری زمانی مربوط به آن نماد را گرفته و مقادیر نادرست (`NaN`) را حذف می‌کند.

● اگر سری خالی نباشد:

● تابع `adf_test` را روی سری اجرا می‌کند تا مقدار `p-value`، آماره آزمون و مقادیر بحرانی را به دست آورد.

- بررسی ایستایی بر اساس **p-value**
- بررسی ایستایی بر اساس آماره آزمون
- در نهایت، دو لیست **stationary_teststat** و **stationary_pvalue** را برمی‌گرداند.

Main Execution Logic

```
# Define the top cryptocurrencies (use sample tickers for demonstration)
tickers = [
    "BTC-USD", "ETH-USD", "USDT-USD", "XRP-USD", "BNB-USD", "SOL-USD", "DOGE-USD", "USDC-USD", "STETH-USD", "ADA-USD",
    "WTRX-USD", "TRX-USD", "AVAX-USD", "WSTETH-USD", "LINK-USD", "TON11419-USD", "SHIB-USD", "WBTC-USD", "SUI20947-USD",
    "WETH-USD", "HBAR-USD", "XLM-USD", "DOT-USD", "BGB-USD", "BCH-USD", "LEO-USD", "LTC-USD", "UNI7083-USD",
    "PEPE24478-USD", "WEETH-USD", "WBETH-USD", "BTCB-USD", "NEAR-USD", "USDE29470-USD", "DAI-USD", "AAVE-USD",
    "APT21794-USD", "ICP-USD", "SUSDE-USD", "POL28321-USD", "CRO-USD", "MNT27075-USD", "ETC-USD",
    "VET-USD", "XMR-USD", "TA022974-USD", "OM-USD", "ARB11841-USD", "FET-USD", "FIL-USD", "VIRTUAL-USD",
    "OKB-USD", "ALGO-USD", "KAS-USD", "ENA-USD", "JITOSOL-USD", "FTM-USD", "ATOM-USD", "BONK-USD", "OP-USD",
    "STX4847-USD", "IMX10603-USD", "TIA22861-USD", "THETA-USD", "FDUSD-USD", "INJ-USD", "ONDO-USD",
    "GRT6719-USD", "WIF-USD", "JASMY-USD", "SEI-USD", "RETH-USD", "WLD-USD", "RSETH-USD", "RUNE-USD",
    "FLOKI-USD", "LDO-USD", "METH29035-USD", "FLR-USD", "FTN-USD",
    "RAY-USD", "XTZ-USD", "BBTC31369-USD", "GT-USD", "MKR-USD", "SAND-USD", "BEAM28298-USD", "QNT-USD", "PYTH-USD",
    "GALA-USD", "KCS-USD", "WZEDX-USD"
]

start_date = "2023-12-01"
end_date = "2024-12-01"

print("Fetching cryptocurrency data...")
dataframes = fetch_crypto_data(tickers, start_date, end_date)

if dataframes:
    merged_data = merge_dataframes_on_date(dataframes)
    print("Merged DataFrame:")
    print(merged_data.head())
else:
    print("No data fetched for the given tickers and date range.")

crypto_data = merged_data
crypto_data.columns = tickers

if crypto_data.empty:
    print("No data was fetched. Please check the tickers or the date range.")
else:
    # Analyze stationarity
    print("Analyzing stationarity...")
    stationary_pvalue, stationary_teststat = analyze_stationarity(crypto_data)
    # Compare results
    print("\nCryptocurrencies stationary based on p-value and their p-values:")
    print([f"\t{x[0]} => p-value: {x[1]} for x in stationary_pvalue]")
    print("\nCryptocurrencies stationary based on test statistic and their p-values & their test_stat compare to critical-1%-value:")
    print([f"\t{x[0]} => p-value: {x[1]}, test_stat compare to 1%-critical_value: {x[2]} for x in stationary_teststat] )
```

این بخش از کد داده‌های تاریخی ارزهای دیجیتال را برای لیستی از نمادها (tickers) دریافت می‌کند، آنها را ترکیب می‌کند و ایستایی (stationarity) سری‌های زمانی را تحلیل می‌کند.

- تعیین تاریخ شروع و پایان:
- "start_date = "2023-12-01"
- "end_date = "2024-12-01"
- دریافت داده‌ها:
- پیام "...Fetching cryptocurrency data" چاپ می‌شود.
- تابع `fetch_crypto_data` با استفاده از `start_date` و `end_date` و `tickers` نمایش داده می‌شود.
- فراخوانی می‌شود تا داده‌های قیمتی را دریافت کند.
- اگر داده‌ها دریافت شوند، آنها با استفاده از تابع `merge_dataframes_on_date` ترکیب می‌شوند.
- در صورت موفقیت، اولین سطرهای `merged_data` نمایش داده می‌شود.
- اگر داده‌ای دریافت نشود، پیام مناسبی چاپ می‌شود.
- تنظیم داده‌های کریپتو:

- `crypto_data = merged_data`
- ستون‌های `crypto_data` به `tickers` تغییر نام داده می‌شوند.
- بررسی خالی بودن داده‌ها:

 - اگر `crypto_data` خالی باشد، پیام هشدار چاپ می‌شود.
 - در غیر این صورت:

 - پیام "...Analyzing stationarity" چاپ می‌شود.
 - تابع `analyze_stationarity` برای تحلیل ایستایی داده‌ها فراخوانی می‌شود.
 - نتایج تحلیل ایستایی بر اساس `p-value` و آماره آزمون نمایش داده می‌شود.

section2

Hurst Exponent

```
def calculate_hurst_exponent(prices: pd.Series, max_lag: int = 100) -> float:
    """
    Calculate the Hurst exponent of a time series using rescaled range (R/S) analysis.
    Parameters:
        prices (pd.Series): A Pandas Series of closing prices.
        max_lag (int): The maximum lag to consider for the calculation.
    Returns:
        float: The Hurst exponent.
    """
    log_prices = np.log(prices)
    log_returns = log_prices.diff().dropna()
    N = len(log_returns)
    if N < 20:
        return np.nan # Not enough data to calculate Hurst exponent reliably
    max_lag = min(max_lag, N // 2)
    rs_values = []
    valid_lags = []
    for lag in range(2, max_lag):
        segments = [log_returns[i:i + lag] for i in range(0, len(log_returns), lag) if len(log_returns[i:i + lag]) == lag]
        if len(segments) == 0:
            continue
        rs_segment = []
        for segment in segments:
            mean = segment.mean()
            cumulative_deviations = np.cumsum(segment - mean)
            R = cumulative_deviations.max() - cumulative_deviations.min() # Range
            S = segment.std(ddof=1) # Sample standard deviation
            if S > 0:
                rs_segment.append(R / S)
        if len(rs_segment) > 0:
            rs_values.append(np.mean(rs_segment))
            valid_lags.append(lag)
    if len(rs_values) < 2:
        return np.nan
    log_lags = np.log(valid_lags)
    log_rs = np.log(rs_values)
    slope, intercept = np.polyfit(log_lags, log_rs, 1)
    hurst_exponent = slope # Slope of the line
    return hurst_exponent
```

این تابع نمای هرست (Hurst Exponent) را برای یک سری زمانی از قیمت‌ها با استفاده از تحلیل دامنه مقیاس شده (R/S) محاسبه می‌کند.

- محاسبه بازده‌های لگاریتمی:
- قیمت‌های ورودی را به لگاریتم تبدیل می‌کند.
- بازده‌های لگاریتمی را با تفاوت‌های متوالی محاسبه می‌کند و مقادیر نادرست را حذف می‌کند.
- بررسی کافی بودن داده‌ها:

 - اگر تعداد داده‌ها کمتر از ۲۰ باشد، مقدار `np.nan` را بر می‌گرداند زیرا داده‌های کافی برای محاسبه نمای هرست وجود ندارد.
 - اطمینان حاصل می‌کند که `max_lag` بیشتر از نصف طول داده‌ها نیست.
 - محاسبه مقادیر R/S برای لگهای مختلف:

 - برای هر لگ از ۲ تا `:max_lag` سری بازده‌ها را به بخش‌هایی با طول لگ تقسیم می‌کند.

- برای هر بخش:
- میانگین بخش را محاسبه می‌کند.
- انحراف تجمعی از میانگین را محاسبه می‌کند.
- دامنه \mathbb{R} و انحراف معیار (S) را محاسبه می‌کند.
- نسبت R/S را اگر انحراف معیار مثبت باشد، محاسبه و ذخیره می‌کند.
- مقادیر میانگین R/S را برای هر لگ ذخیره می‌کند.
- برآورد نمای هرست با رگرسیون خطی:
- اگر داده‌های کافی برای رگرسیون وجود داشته باشد:
- لگاریتم لگها و مقادیر R/S را محاسبه می‌کند.
- یک رگرسیون خطی بر روی داده‌های لگاریتمی انجام می‌دهد.
- شبیه خط رگرسیون نمای هرست است.
- اگر داده‌های کافی وجود نداشته باشد، مقدار `np.nan` را برمی‌گرداند.
- خروجی:
- مقدار عددی نمای هرست که رفتار سری زمانی را توصیف می‌کند.

اینتابع به تحلیل رفتار سری زمانی کمک می‌کند:

- اگر $H \approx 0.5$: سری زمانی شبیه به یک گام تصادفی است (بدون حافظه).
- اگر $H < 0.5$: سری زمانی تمایل به برگشت به میانگین دارد (ضد روند).
- اگر $H > 0.5$: سری زمانی تمایل به حفظ روند دارد (حافظه بلندمدت).

نمای هرست برای فهم و پیش‌بینی رفتار سری‌های زمانی مالی و دیگر داده‌های زمانی مفید است.

Half-life

```
def calculate_half_life(series):
    """
    Calculate the half-life of mean reversion for a time series.
    """
    # Clean the series (ensure no NaN or infinite values)
    series = series.replace([np.inf, -np.inf], np.nan).dropna()

    # Calculate lagged series
    lagged_series = series.shift(1).dropna()
    delta_series = series.diff().dropna()

    # Ensure the series lengths match
    if len(lagged_series) != len(delta_series):
        print("Series lengths do not match for half-life calculation.")
        return np.nan

    # Perform linear regression (without libraries)
    mean_lagged = lagged_series.mean()
    mean_delta = delta_series.mean()

    # Covariance and variance
    cov = ((lagged_series - mean_lagged) * (delta_series - mean_delta)).sum()
    var = ((lagged_series - mean_lagged) ** 2).sum()

    # Slope (beta)
    slope = cov / var if var > 0 else 0

    # Check if slope is valid
    if slope <= 0 or slope >= 1:
        print(f"Invalid slope for half-life calculation: {slope}")
        return np.nan

    # Calculate half-life
    half_life = np.log(2) / np.log(1 - slope)

    return half_life
```

اینتابع نیمه عمر بازگشت به میانگین را برای یک سری زمانی محاسبه می‌کند.

۳. پاکسازی داده‌ها:

- مقادیر بینهایت (`inf` یا `-inf`) را با `NaN` جایگزین می‌کند.
- تمامی مقادیر `NaN` را از سری حذف می‌کند.

- هدف این مرحله اطمینان از تمیز بودن داده‌ها برای محاسبات بعدی است.

ii. ایجاد سری‌های تاخیری و تغییرات:

- سری تاخیری: سری اصلی را یک واحد به جلو شیفت می‌دهد تا مقادیر آن با مقادیر قبلی خود مقایسه شوند.
- تغییرات سری: تفاوت بین مقادیر سری اصلی در زمان‌های متوالی را محاسبه می‌کند.
- این دو سری برای انجام رگرسیون خطی بین تغییرات سری و مقادیر قبلی آن استفاده می‌شوند.

iii. بررسی تطابق طول سری‌ها:

- اطمینان حاصل می‌کند که سری‌های تاخیری و تغییرات دارای طول برابر هستند.
- اگر طول‌ها برابر نباشند، به این معنی است که داده‌های کافی برای محاسبه وجود ندارد و تابع مقدار NaN را برمی‌گرداند.

iv. محاسبه شبیه رگرسیون خطی (Beta):

- میانگین سری‌های تاخیری و تغییرات را محاسبه می‌کند.
- کوواریانس بین سری‌های تاخیری و تغییرات را محاسبه می‌کند.
- واریانس سری تاخیری را محاسبه می‌کند.
- شبیه رگرسیون (یا همان Beta) را با تقسیم کوواریانس بر واریانس به دست می‌آورد.
- این شبیه نشان‌دهنده سرعت بازگشت سری زمانی به میانگین است.

v. محاسبه نیمه‌عمر بازگشت به میانگین:

- از فرمول نیمه‌عمر در فرآیندهای آرشه یکپارچه مرتبه اول (Ornstein-Uhlenbeck) استفاده می‌کند.

$$\text{فرمول: نیمه‌عمر} = (\ln(2) / \ln(1 - \text{Beta})$$

- این فرمول نشان می‌دهد که چه مدت زمان نیاز است تا انحراف از میانگین به نصف کاهش یابد.

Sort Time Series Based on Hurst Exponent

```
def analyze_hurst_and_half_life(data, stationary_pvalue):
    """
    Analyze Hurst exponent and half-life for stationary time series.
    Sort the results based on Hurst exponent in ascending order.
    """
    results = []

    for ticker, _ in stationary_pvalue:
        # Check if the ticker exists in the data
        if ticker not in data.columns:
            print(f"Warning: {ticker} not found in data. Skipping...")
            continue

        # Drop NaN values for the series
        series = data[ticker].dropna()

        # Ensure the series is not empty
        if series.empty:
            print(f"Warning: {ticker} series is empty after dropping NaNs. Skipping...")
            continue

        # Calculate Hurst exponent and half-life
        try:
            hurst = calculate_hurst_exponent(series)
            half_life = calculate_half_life(series)
            results.append((ticker, hurst, half_life))
        except Exception as e:
            print(f"Error processing {ticker}: {e}")
            continue

    # Convert results to a DataFrame for easier sorting and visualization
    results_df = pd.DataFrame(results, columns=["Ticker", "Hurst Exponent", "Half-Life"])
    results_df = results_df.sort_values(by="Half-Life", ascending=True)

    return results_df
```

این تابع به تحلیل نمای هرست (Hurst Exponent) و نیمه‌عمر بازگشت به میانگین (Half-Life) برای سری‌های زمانی ایستا می‌پردازد و نتایج را بر اساس نیمه‌عمر به ترتیب صعودی مرتب می‌کند

ایجاد یک لیست برای نتایج:

- یک لیست خالی به نام `results` برای ذخیره نتایج تحلیل ایجاد می‌کند.
- پردازش هر نماد ایستا:

 - برای هر نماد (`ticker`) در `:stationary_pvalue` بررسی وجود نماد در داده‌ها:
 - اگر نماد در ستون‌های `data` وجود نداشته باشد، هشدار می‌دهد و از آن عبور می‌کند.
 - دریافت سری زمانی قیمت:
 - سری زمانی مربوط به نماد را از `data` گرفته و مقادیر `NaN` را حذف می‌کند.
 - بررسی خالی نبودن سری زمانی:
 - اگر سری زمانی پس از حذف `NaN` خالی باشد، هشدار می‌دهد و از آن عبور می‌کند.
 - محاسبه نمای هرست و نیمه‌عمر:
 - سعی می‌کند نمای هرست را با استفاده ازتابع `calculate_hurst_exponent` محاسبه کند.
 - نیمه‌عمر بازگشت به میانگین را با استفاده ازتابع `calculate_half_life` محاسبه می‌کند.
 - نتایج شامل نماد، نمای هرست و نیمه‌عمر را به لیست `results` اضافه می‌کند.
 - مدیریت استثناهای:
 - اگر در حین محاسبات خطای رخ دهد، پیام خطأ چاپ شده و پردازش ادامه می‌یابد.

ایجاد یک DataFrame از نتایج:

- لیست `results` را به یک `DataFrame` تبدیل می‌کند که ستون‌های آن عبارتند از:

 - نام نماد ارز دیجیتال: "Ticker"
 - نمای هرست: "Hurst Exponent"
 - نیمه‌عمر بازگشت به میانگین محاسبه شده: "Half-Life"
 - نتایج را بر اساس ستون "Half-Life" به ترتیب صعودی مرتب می‌کند تا ارزهای دیجیتالی که سریع‌تر به میانگین بازمی‌گردند در بالای لیست قرار گیرند.

Plot

```

def plot_hurst_and_half_life(results_df):
    """
    Plot Hurst exponent and Half-Life separately, sorted by ascending Half-Life.
    """
    # Sort the results by Half-Life in ascending order
    results_df = results_df.sort_values(by="Half-Life", ascending=True)

    # Plot Hurst Exponent
    plt.figure(figsize=(12, 6))
    plt.plot(results_df["Ticker"], results_df["Hurst Exponent"], marker="o", label="Hurst Exponent", color="blue")
    plt.title("Hurst Exponent (Sorted by Half-Life)", fontsize=14)
    plt.xlabel("Cryptocurrency Ticker", fontsize=12)
    plt.ylabel("Hurst Exponent", fontsize=12)
    plt.xticks(rotation=90, fontsize=8) # Rotate tickers for better readability
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Plot Half-Life
    plt.figure(figsize=(12, 6))
    plt.plot(results_df["Ticker"], results_df["Half-Life"], marker="o", label="Half-Life", color="green")
    plt.title("Half-Life (Sorted by Half-Life)", fontsize=14)
    plt.xlabel("Cryptocurrency Ticker", fontsize=12)
    plt.ylabel("Half-Life", fontsize=12)
    plt.xticks(rotation=90, fontsize=8) # Rotate tickers for better readability
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

```

این تابع دو نمودار جدگانه برای نمایش نمای هرست (Hurst Exponent) و نیمه عمر بازگشت به میانگین (Half-Life) مجموعه‌ای از ارز‌های دیجیتال ایجاد می‌کند. نتایج بر اساس نیمه عمر به صورت صعودی مرتب می‌شوند تا مقایسه آسان‌تر باشد.

```

# Analyze Hurst exponent and half-life for stationary time series
hurst_results = analyze_hurst_and_half_life(crypto_data, stationary_pvalue)

# Display the sorted results
print(hurst_results)

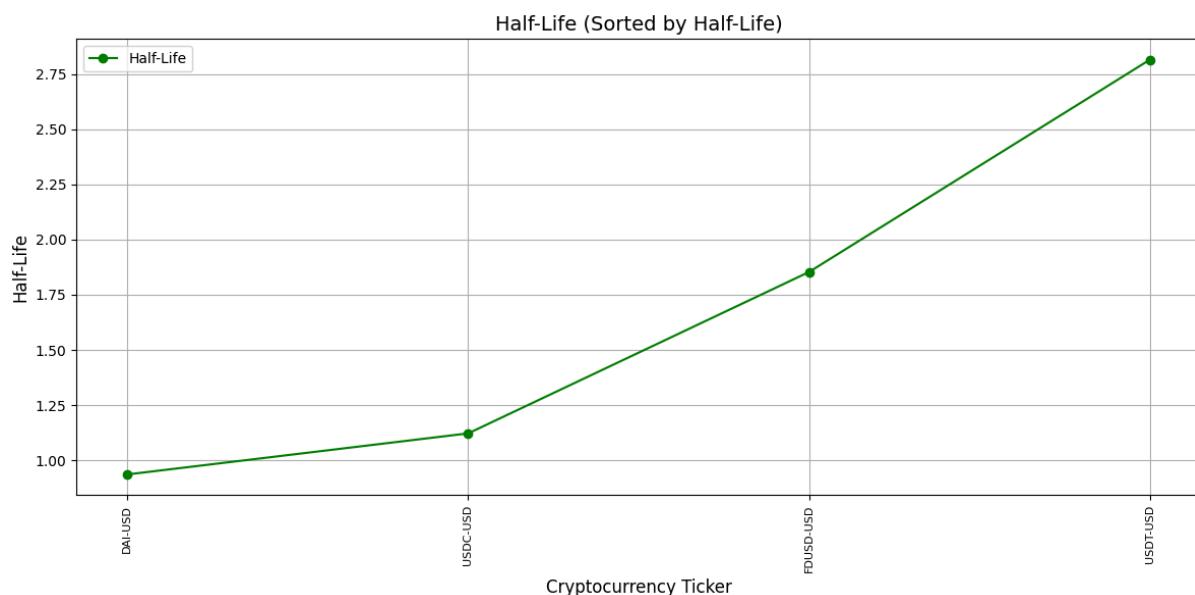
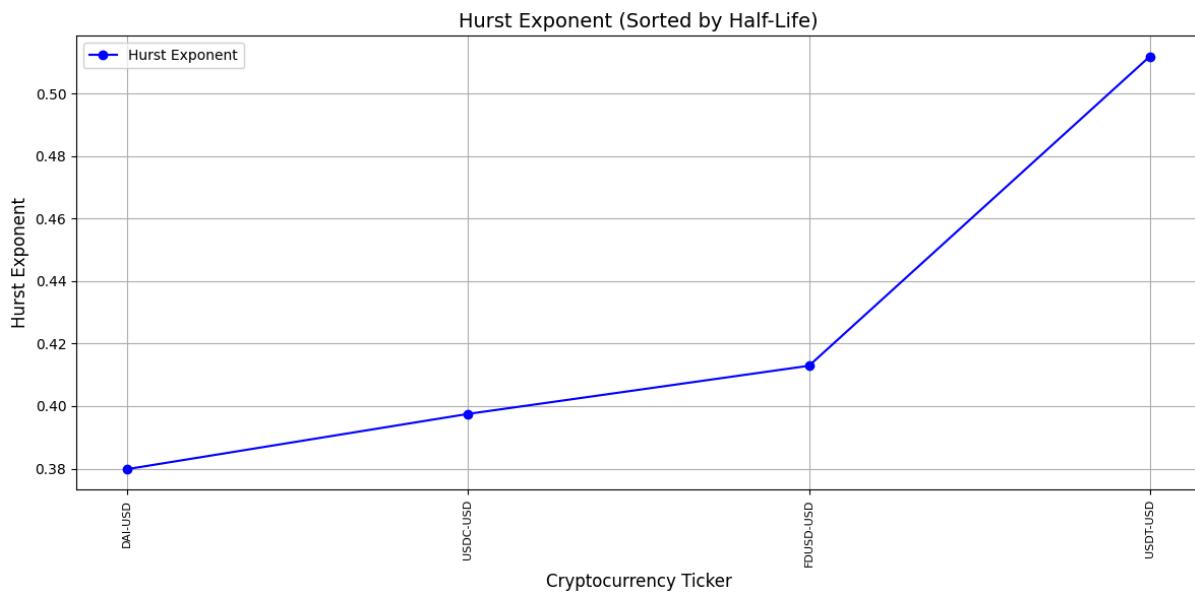
# Plot the relationship between Hurst exponent and half-life
plot_hurst_and_half_life(hurst_results)

```

این کد به تحلیل نمای هرست و نیمه عمر بازگشت به میانگین برای ارز‌های دیجیتالی که سری زمانی آن‌ها ایستا است، می‌پردازد. نتایج محاسبه شده نمایش داده شده و سپس نمودارهایی برای تجسم رابطه بین نمای هرست و نیمه عمر ایجاد می‌شوند. این تحلیل می‌تواند در شناسایی فرآنشهای معاملاتی و درک دینامیک بازار ارز‌های دیجیتال مفید باشد.

خروجی:

	Ticker	Hurst Exponent	Half-Life
2	DAI-USD	0.379830	0.935732
1	USDC-USD	0.397499	1.121796
3	FDUSD-USD	0.412898	1.853133
0	USDT-USD	0.511875	2.815123



۱. عوامل مشترک در نیمه عمر برای رمزارزهای کمتر مانا

از تحلیل و نمودارها، عوامل مشترک زیر در میان رمزارزهایی که کمتر مانا به نظر می‌رسند، مشاهده می‌شود:

نیمه عمر طولانی‌تر:

نمودار نیمه عمر نشان می‌دهد که رمزارزهای کمتر مانا (مثلًا USDT-USD و FDUSD-USD) در مقایسه با رمزارزهای ماناتر (مثلًا DAI-USD) نیمه عمر طولانی‌تری دارند.

نیمه عمر طولانی‌تر نشان‌دهنده بازگشت به میانگین ضعیفتر است، به این معنی که قیمت‌ها زمان بیشتری برای بازگشت به میانگین خود نیاز دارند، یا ممکن است اصلًا بازگشت نکند.

نمایه هرست بالاتر:

نمودار نمایه هرست نشان می‌دهد که رمざرزهای کمتر مانا مقادیر نمایه هرست بالاتری دارند (مثالاً USDT-USD با $(0.51 \approx H)$.

نمایه هرست بالاتر رفتار روندی یا پایدار را نشان می‌دهد، که با مشاهده نیمه عمرهای طولانی‌تر همخوانی دارد.

بازگشت به میانگین ضعیفتر:

رمざرزهایی با نیمه عمرهای طولانی‌تر و نمایه هرست بالاتر، دینامیک‌های بازگشت به میانگین ضعیفتری نشان می‌دهند، که آن‌ها را کمتر قابل پیش‌بینی و کمتر مانا می‌کند.

2. آچه نمایه هرست و نیمه عمر درباره سری‌های قیمتی بیان می‌کند

نمایه هرست:

نمایه هرست (H) حافظه یا پایداری یک سری زمانی را اندازه‌گیری می‌کند:

$H > 0.5$: رفتار بازگشت به میانگین (ضد پایدار). قیمت‌ها به میانگین خود بازمی‌گردند، که آن‌ها را برای استراتژی‌های بازگشت به میانگین مناسب می‌کند.

$H = 0.5$: رفتار گام تصادفی (بدون حافظه). قیمت‌ها غیرقابل پیش‌بینی هستند و یک فرآیند تصادفی را دنبال می‌کند.

$H < 0.5$: رفتار روندی (پایدار). قیمت‌ها به احتمال بیشتری به حرکت در همان جهت ادامه می‌دهند، که آن‌ها را برای استراتژی‌های مومنتوم مناسب می‌کند.

نیمه عمر:

نیمه عمر سرعت بازگشت به میانگین را اندازه‌گیری می‌کند:

نیمه عمر کوتاه: نشان‌دهنده بازگشت به میانگین قوی است، جایی که قیمت‌ها به سرعت به میانگین خود بازمی‌گردند. این رمざرزها برای استراتژی‌های بازگشت به میانگین ایده‌آل هستند.

نیمه عمر طولانی: نشان‌دهنده عدم بازگشت به میانگین ضعیف است، جایی که قیمت‌ها زمان بیشتری برای بازگشت به میانگین خود نیاز دارند. این رمざرزها ممکن است رفتار روندی یا گام تصادفی داشته باشند.

نیمه عمر نامشخص یا NaN : نشان‌دهنده عدم بازگشت به میانگین است، که اغلب با سری‌های گام تصادفی یا روندی مرتبط است.

3. بینش‌هایی از نمودارها

نمودار نمایه هرست:

نمودار نشان می‌دهد که با افزایش نیمه عمر (از DAI-USD تا USDT-USD)، نمایه هرست نیز افزایش می‌یابد.

این نشان می‌دهد که رمزارزهایی با نیمه عمر های طولانی‌تر تمایل دارند نمایه هرست بالاتری داشته باشند، که بیانگر مانایی ضعیفتر و رفتار روندی بیشتر است.

نمودار نیمه عمر:

نیمه عمر برای رمزارزهای کمتر مانا به طور قابل توجهی افزایش می‌یابد (مثلًا USDT-USD طولانی‌ترین نیمه عمر را دارد).

این تأیید می‌کند که رمزارزهای کمتر مانا زمان بیشتری برای بازگشت به میانگین خود نیاز دارند، یا ممکن است اصلاً بازگشت نکند.

4. نتیجه‌گیری

رمزارزهایی با نیمه عمر های طولانی‌تر و نمایه هرست بالاتر کمتر مانا هستند، بازگشت به میانگین ضعیفتر نشان می‌دهند، و ممکن است رفتار روندی یا گام تصادفی داشته باشند.

نمایه هرست و نیمه عمر معیارهای مهمی برای درک رفتار سری‌های قیمتی هستند:

نمایه هرست تعیین می‌کند که آیا یک سری بازگشت به میانگین دارد، تصادفی است، یا روندی است.

نیمه عمر سرعت بازگشت به میانگین را کمی می‌کند، که به ارزیابی قابلیت پیش‌بینی حرکت‌های قیمت کمک می‌کند.

این معیارها برای طراحی استراتژی‌های معاملاتی حیاتی هستند:

استراتژی‌های بازگشت به میانگین برای رمزارزهایی با نمایه هرست پایین و نیمه عمر های کوتاه بهترین کارایی را دارند.

استراتژی‌های مومنتوم برای رمزارزهایی با نمایه هرست بالا و نیمه عمر های طولانی‌تر مناسب‌تر هستند.

نمودارها به طور مؤثر این روابط را نشان می‌دهند و برای تحلیل بسیار مرتبط هستند.

section3

Conintegration of non-stationary pairs

```

import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import coint
from statsmodels.api import OLS, add_constant

# Example input: crypto_data (DataFrame with crypto prices), stationary_pvalue (list of stationary cryptos)
# crypto_data: A DataFrame where columns are cryptocurrency tickers and rows are price data.
# stationary_pvalue: A list of tuples [(ticker, p-value), ...] for stationary cryptocurrencies.

def find_cointegrated_pairs(non_stationary_cryptos):
    """
    Find the top 10 cointegrated pairs of non-stationary cryptocurrencies.

    Parameters:
        crypto_data (pd.DataFrame): Time series data for cryptocurrencies.
        stationary_pvalue (list): List of tuples [(ticker, p-value), ...] for stationary cryptocurrencies.

    Returns:
        pd.DataFrame: A DataFrame with the top 10 cointegrated pairs and their p-values.
    """

    # Step 1: Perform the CADF test on all pairs of non-stationary cryptocurrencies
    results = []
    for i, crypto1 in enumerate(non_stationary_cryptos):
        for j, crypto2 in enumerate(non_stationary_cryptos):
            if i < j: # Avoid duplicate pairs
                # Perform the CADF test
                series1 = crypto_data[crypto1].dropna()
                series2 = crypto_data[crypto2].dropna()

                # Ensure both series have the same length
                min_len = min(len(series1), len(series2))
                series1 = series1[-min_len:] # Align series to the same length
                series2 = series2[-min_len:]

                # CADF test
                coint_stat, p_value, crit_values = coint(series1, series2)

                # Save the result if p-value < 0.05
                if p_value < 0.05:
                    results.append((crypto1, crypto2, p_value, coint_stat))

```

```

# Step 2: Sort results by p-value and select the top 10 pairs
results = sorted(results, key=lambda x: x[2]) # Sort by p-value
top_10_pairs = results[:10] # Select top 10 pairs

# Step 3: Calculate coefficients for the top 10 pairs using OLS
final_results = []
for crypto1, crypto2, p_value, coint_stat in top_10_pairs:
    # Perform OLS regression
    series1 = crypto_data[crypto1].dropna()
    series2 = crypto_data[crypto2].dropna()

    # Align the series to the same length
    min_len = min(len(series1), len(series2))
    series1 = series1[-min_len:]
    series2 = series2[-min_len:]

    # OLS regression: series2 = beta * series1 + intercept
    X = add_constant(series1) # Add intercept to the model
    model = OLS(series2, X).fit()
    intercept, beta = model.params

    # Create the combined series (stationary series)
    combined_series = series2 - (intercept + beta * series1)

    # Append the results
    final_results.append({
        "Crypto 1": crypto1,
        "Crypto 2": crypto2,
        "Combined_series": combined_series,
        "P-Value": p_value,
        "Cointegration Statistic": coint_stat,
        "Intercept": intercept,
        "Beta": beta
    })

# Convert results to a DataFrame
final_results_df = pd.DataFrame(final_results)
return final_results_df

```

این تابع به منظور یافتن ۱۰ جفت ارز دیجیتال دارای همانباشته‌گی (Cointegration) در بین ارزهای دیجیتال غیرایستا طراحی شده است.

- آزمون همانباشته‌گی (Cointegration) بین تمامی جفت‌های ممکن:
- برای هر جفت منحصر به فرد از ارزهای دیجیتال غیرایستا:

- سری زمانی قیمت‌های هر دو ارز دیجیتال را دریافت می‌کند و مقادیر NaN را حذف می‌کند.
- اطمینان حاصل می‌کند که هر دو سری زمانی دارای طول یکسان هستند (با تراز کردن آخرین داده‌ها).
- از آزمون همانباشته‌گی Johansen استفاده می‌کند تا ببیند آیا ترکیبی خطی از دو سری زمانی ایستا است یا خیر.
- آزمون همانباشته‌گی (CADF) را انجام می‌دهد و مقدار آماره آزمون و مقدار p-value را به دست می‌آورد.
- اگر مقدار p-value کمتر از ۰.۰۵ باشد، به این معنی است که دو سری زمانی همانباشته هستند و نتیجه را ذخیره می‌کند.

ii. انتخاب ۱۰ جفت با کمترین p-value:

- تمامی نتایج آزمون‌های همانباشته‌گی را بر اساس مقدار p-value به صورت صعودی مرتب می‌کند.
- ۱۰ جفت برتر با کمترین مقدار p-value را انتخاب می‌کند که نشان‌دهنده قوی‌ترین همانباشته‌گی بین جفت ارزها است.

iii. محاسبه ضرایب رگرسیون خطی (OLS) برای ۱۰ جفت برتر:

- برای هر یک از ۱۰ جفت منتخب:
 - سری زمانی قیمت‌های هر دو ارز دیجیتال را دریافت و تراز می‌کند.
 - رگرسیون خطی حداقل مربعات معمولی (OLS) را بین سری قیمت‌ها انجام می‌دهد:
 - ضریب شیب (Beta) و عرض از مبدأ (Intercept) را از مدل رگرسیون به دست می‌آورد.
 - سری زمانی ترکیبی را با استفاده از این ضرایب محاسبه می‌کند
 - این سری ترکیبی باید ایستا باشد در صورتی که دو سری همانباشته باشند.
- نتایج شامل نمادهای دو ارز دیجیتال، سری ترکیبی، مقادیر p-value، آماره همانباشته‌گی، و ضرایب رگرسیون را ذخیره می‌کند.

iv. خروجی:

- یک DataFrame که شامل اطلاعات زیر برای هر یک از ۱۰ جفت ارز دیجیتال برتر است:
 - نماد ارز دیجیتال اول: Crypto 1
 - نماد ارز دیجیتال دوم: Crypto 2
 - سری زمانی ترکیبی ایستا: Combined_series
 - مقدار p-value آزمون همانباشته‌گی: P-Value
 - آماره آزمون همانباشته‌گی: Cointegration Statistic
 - عرض از مبدأ مدل رگرسیون: Intercept
 - ضریب شیب مدل رگرسیون: Beta

```
non_stationary_cryptos = list(set(crypto_data) - set(stationary_pvalue))
find_cointegrated_pairs(non_stationary_cryptos)

cointegrated_pairs = find_cointegrated_pairs(non_stationary_cryptos)
print(cointegrated_pairs)
```

- این کد ابتدا با تعریق مجموعه ارزهای ایستا از مجموعه کل ارزهای دیجیتال، لیست ارزهای دیجیتال غیرایستا را تعیین می‌کند.

- سپس با بررسی تمامی جفت‌های ممکن از ارزهای غیرایستا، جفت‌هایی را که رابطه همانباشته‌گی معنی‌داری دارند، شناسایی می‌کند.
- از بین این جفت‌ها، ۱۰ جفت برتر با قوی‌ترین رابطه همانباشته‌گی انتخاب می‌شوند.
- برای این جفت‌ها، رگرسیون خطی انجام شده و سری ترکیبی ایستا محاسبه می‌شود.
- نتایج شامل اطلاعات مهمی درباره جفت‌های همانباشته، مانند ضرایب رگرسیون و آماره‌های آزمون، در اختیار تحلیلگر قرار می‌گیرد.

```

import matplotlib.pyplot as plt

# Print the coefficients for each cointegrated pair
print("Cointegrated Pairs and Coefficients:")
for index, row in cointegrated_pairs.iterrows():
    print(f"Pair: {row['Crypto 1']} and {row['Crypto 2']}")
    print(f" P-Value: {row['P-Value']:.4f}")
    print(f" Cointegration Statistic: {row['Cointegration Statistic']:.4f}")
    print(f" Intercept: {row['Intercept']:.4f}")
    print(f" Beta: {row['Beta']:.4f}")
    print()

# Plot the time series for each cointegrated pair
crypto1 = row['Crypto 1']
crypto2 = row['Crypto 2']
intercept = row['Intercept']
beta = row['Beta']
combined_series = row['Combined_series']

# Get the time series data
series1 = crypto_data[crypto1].dropna()
series2 = crypto_data[crypto2].dropna()

# Plot the time series
plt.figure(figsize=(12, 6))
plt.plot(series1, label=f'{crypto1}', alpha=0.7)
plt.plot(series2, label=f'{crypto2}', alpha=0.7)
plt.plot(combined_series, label="Combined Series (Stationary)", linestyle="--", alpha=0.9)
plt.title(f"Cointegrated Pair: {crypto1} and {crypto2}")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.grid()
plt.show()

```

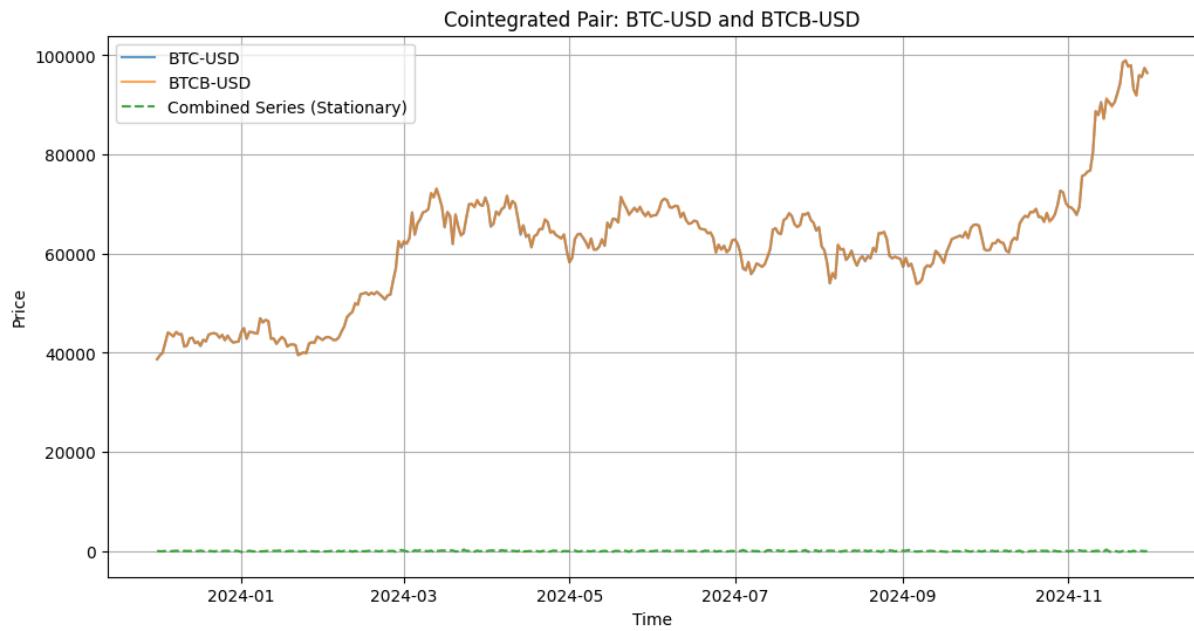
۱. نمایش ضرایب جفت‌های همانباشته:

- برای هر جفت همانباشته، اطلاعات زیر چاپ می‌شود:
 - نام دو ارز دیجیتال در جفت.
 - مقدار **p-value** و آماره همانباشته‌گی که نشان‌دهنده معنی‌داری آماری همانباشته‌گی بین دو ارز هستند.
 - عرض از مبدأ (**Intercept**) و ضریب شیب (**Beta**) مدل رگرسیون خطی، که رابطه خطی بین دو سری زمانی را توصیف می‌کنند.

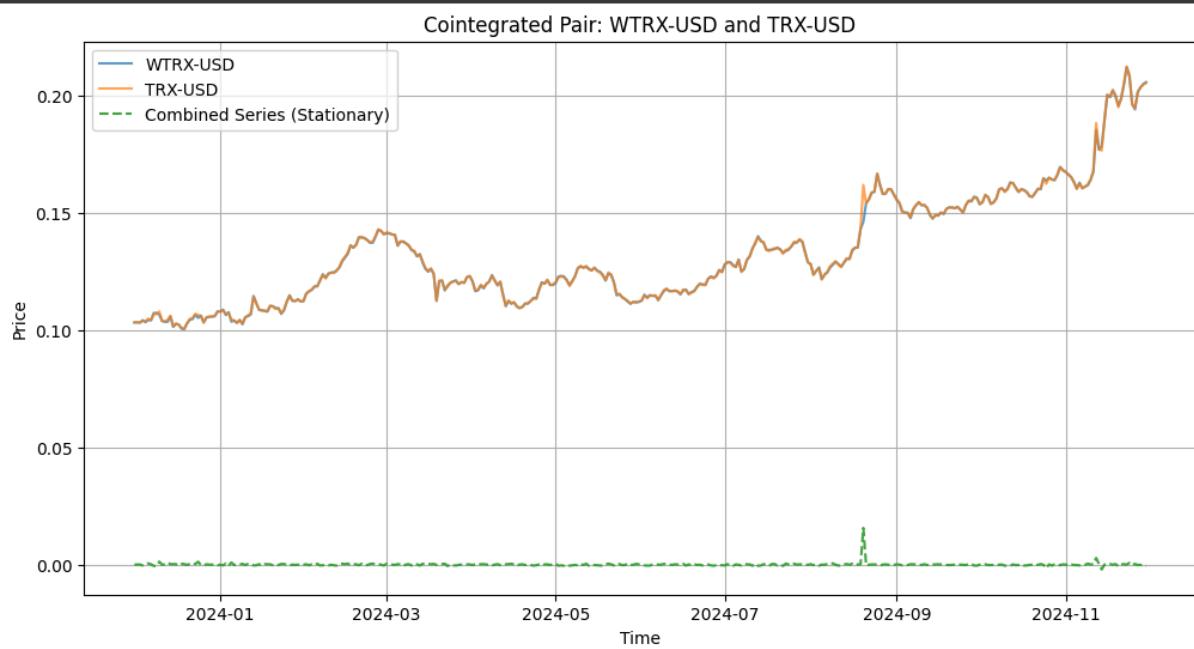
۲. ترسیم سری‌های زمانی برای هر جفت همانباشته:

- دریافت داده‌های قیمتی دو ارز دیجیتال و حذف مقادیر گمشده.
- محاسبه سری ترکیبی ایستا با استفاده از ضرایب رگرسیون بهدست‌آمده.
- ترسیم نمودار که شامل:
 - سری زمانی قیمت هر یک از دو ارز دیجیتال.
 - سری ترکیبی ایستا.
 - تنظیمات بصری مانند عنوان نمودار، برچسب‌های محورها، راهنمای و شبکه.
- نمایش نمودار برای مشاهده بصری رابطه بین دو ارز دیجیتال و سری ترکیبی.

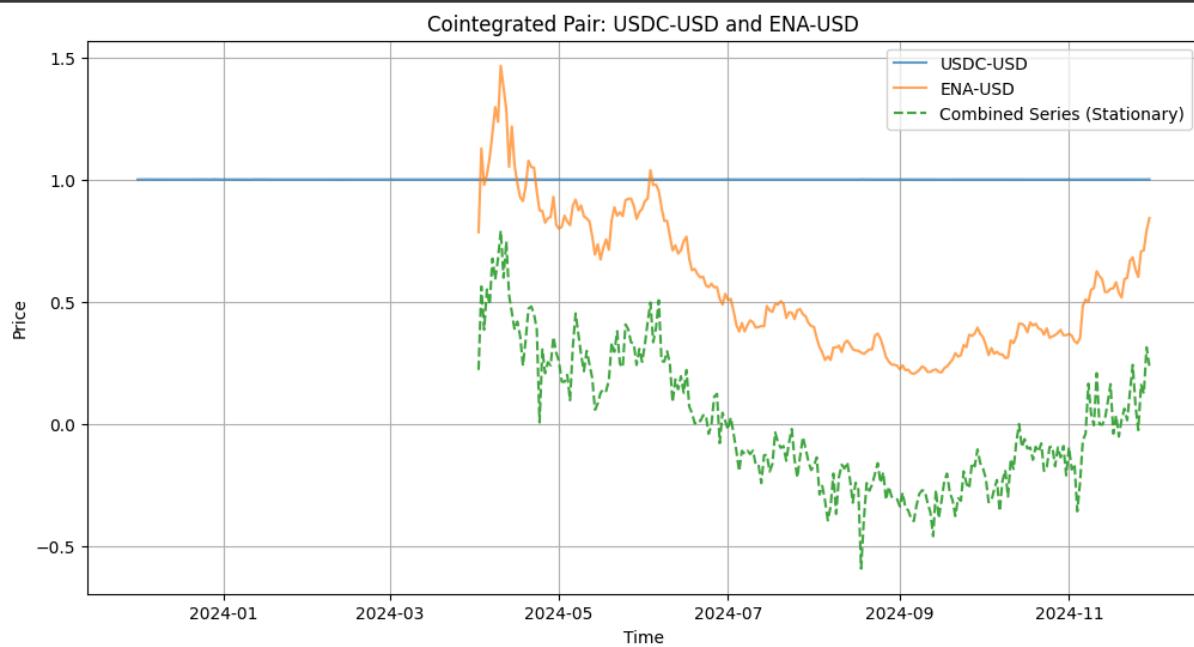
Cointegrated Pairs and Coefficients:
Pair: BTC-USD and BTCB-USD
P-Value: 0.0000
Cointegration Statistic: -20.6359
Intercept: 22.1630
Beta: 0.9997



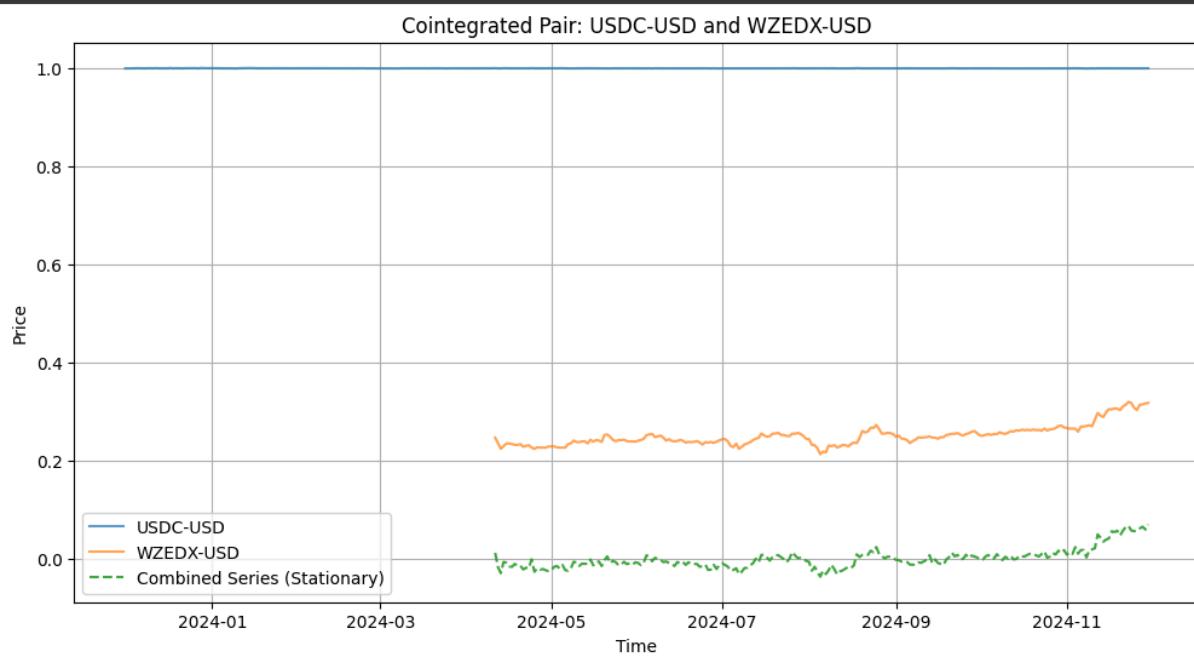
Pair: WTRX-USD and TRX-USD
P-Value: 0.0000
Cointegration Statistic: -19.0991
Intercept: 0.0000
Beta: 1.0004



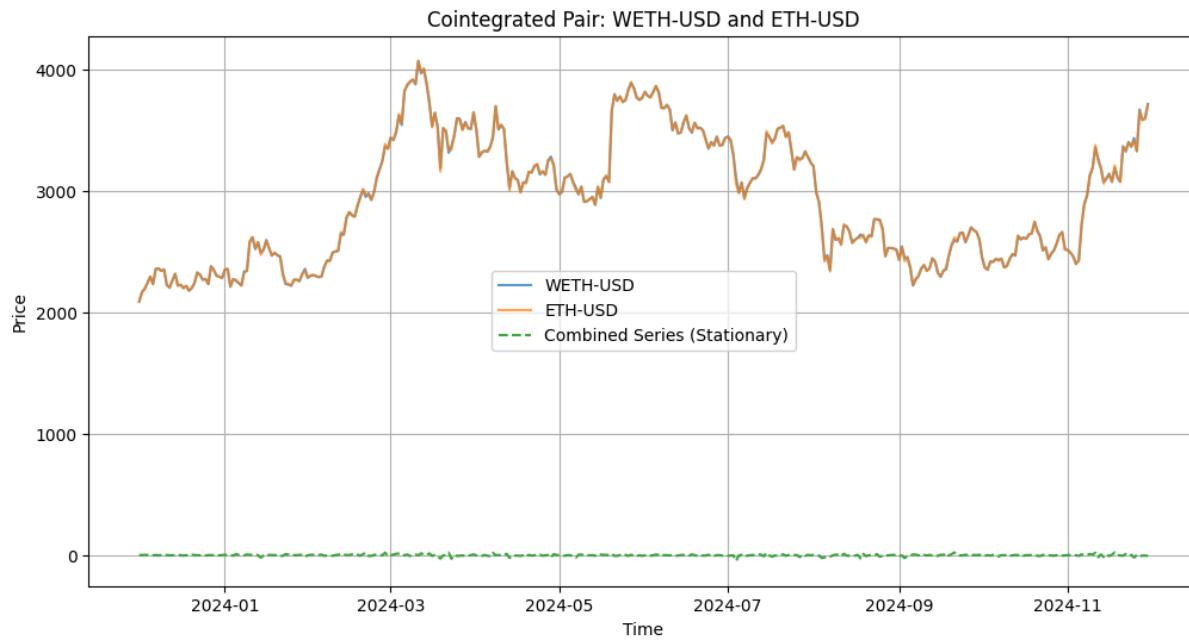
Pair: USDC-USD and ENA-USD
P-Value: 0.0000
Cointegration Statistic: -13.1270
Intercept: -667.9054
Beta: 668.4713



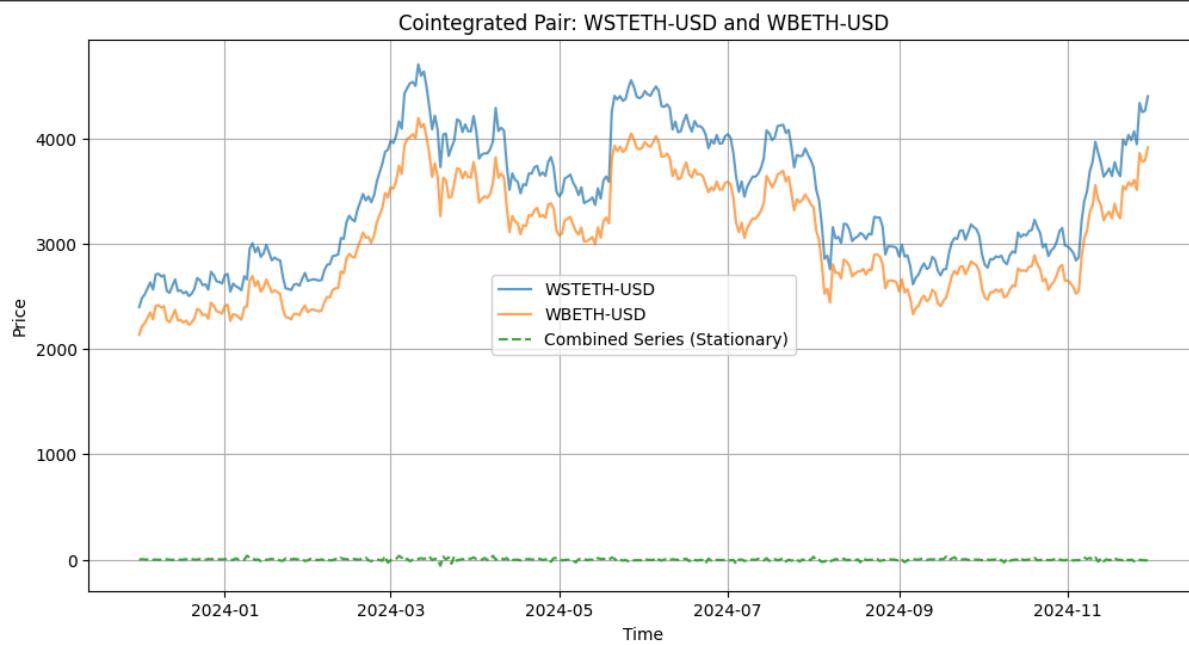
Pair: USDC-USD and WZEDX-USD
P-Value: 0.0000
Cointegration Statistic: -13.0883
Intercept: 50.4496
Beta: -50.1991



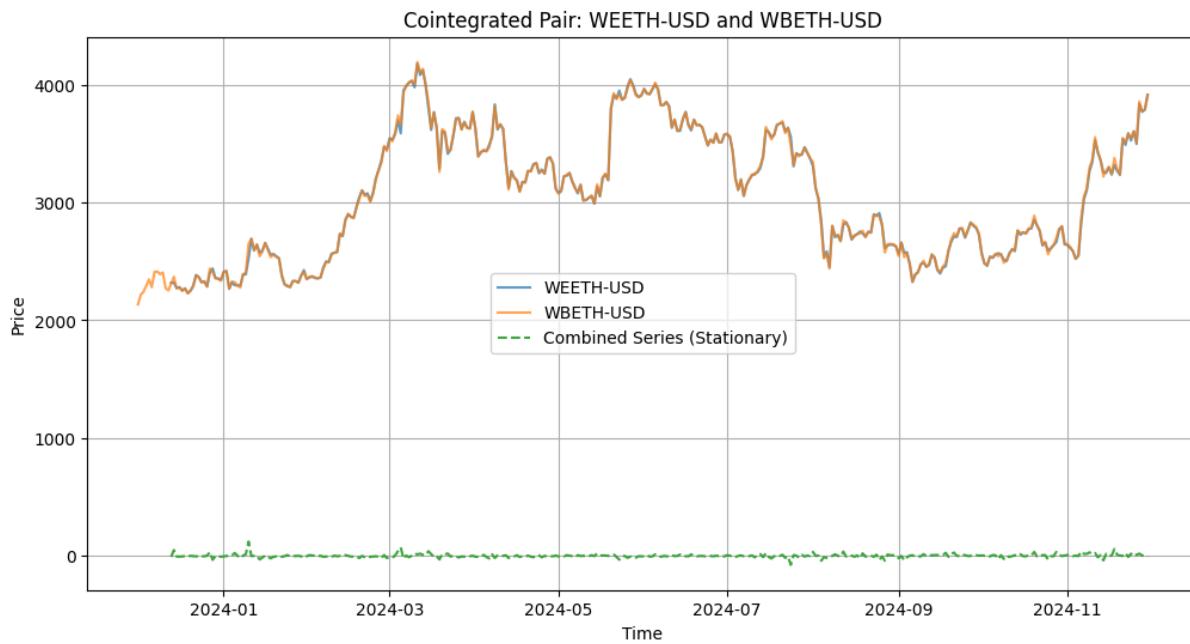
Pair: WETH-USD and ETH-USD
P-Value: 0.0000
Cointegration Statistic: -12.2947
Intercept: -0.4562
Beta: 1.0000



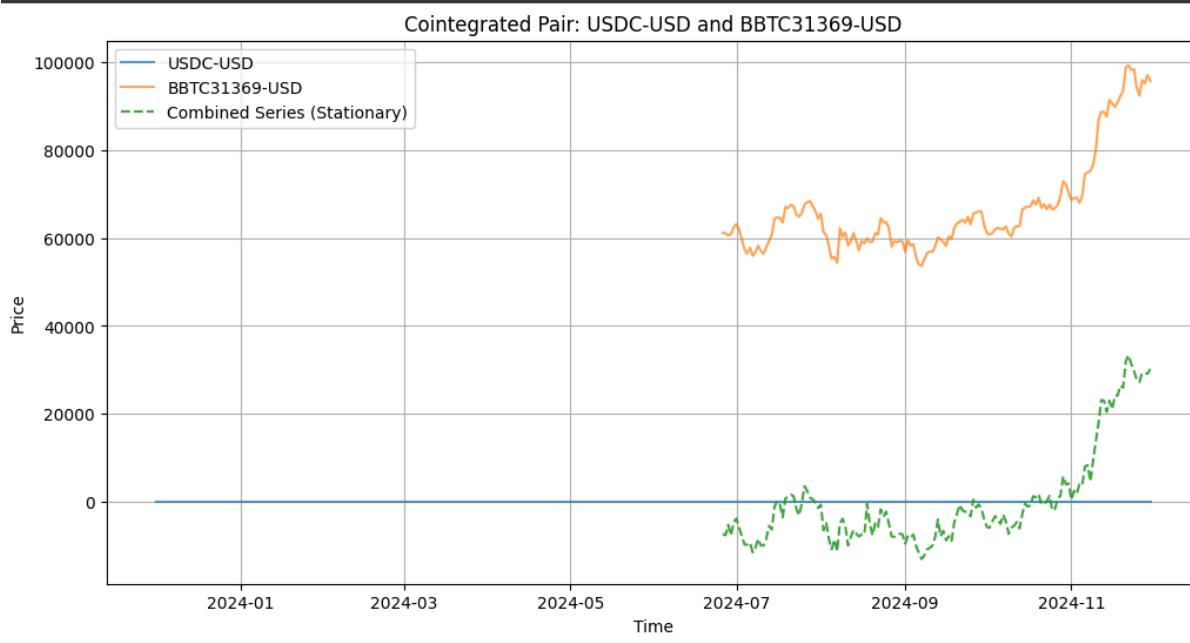
Pair: WSTETH-USD and WBETH-USD
P-Value: 0.0000
Cointegration Statistic: -12.0460
Intercept: -0.7139
Beta: 0.8908



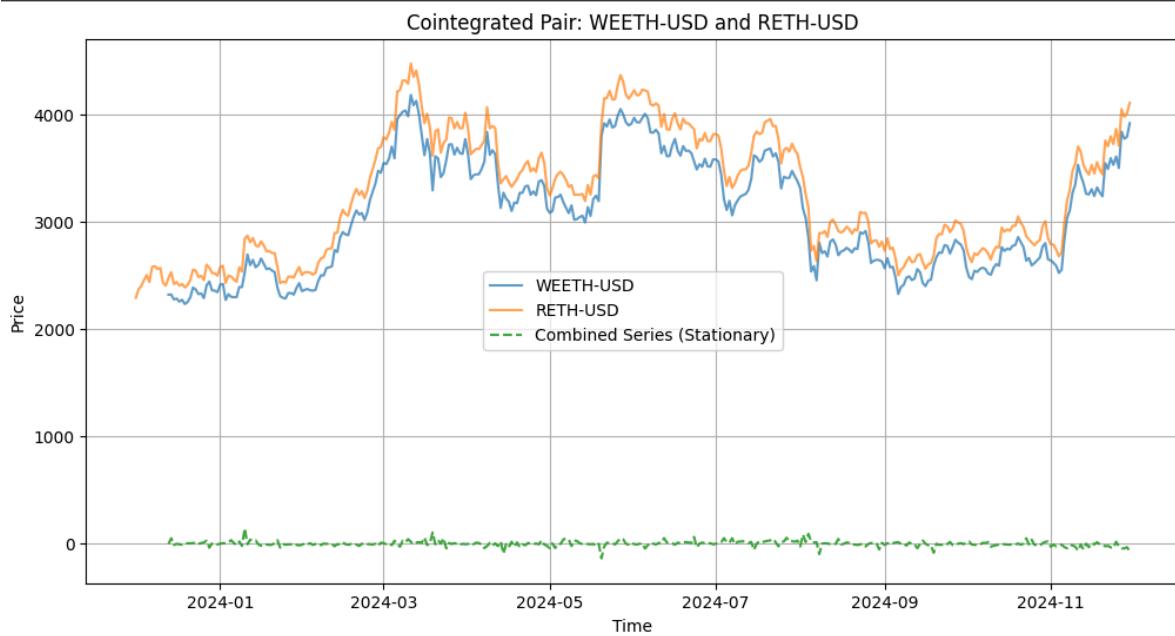
Pair: WEETH-USD and WBETH-USD
P-Value: 0.0000
Cointegration Statistic: -11.4935
Intercept: -1.9562
Beta: 1.0012



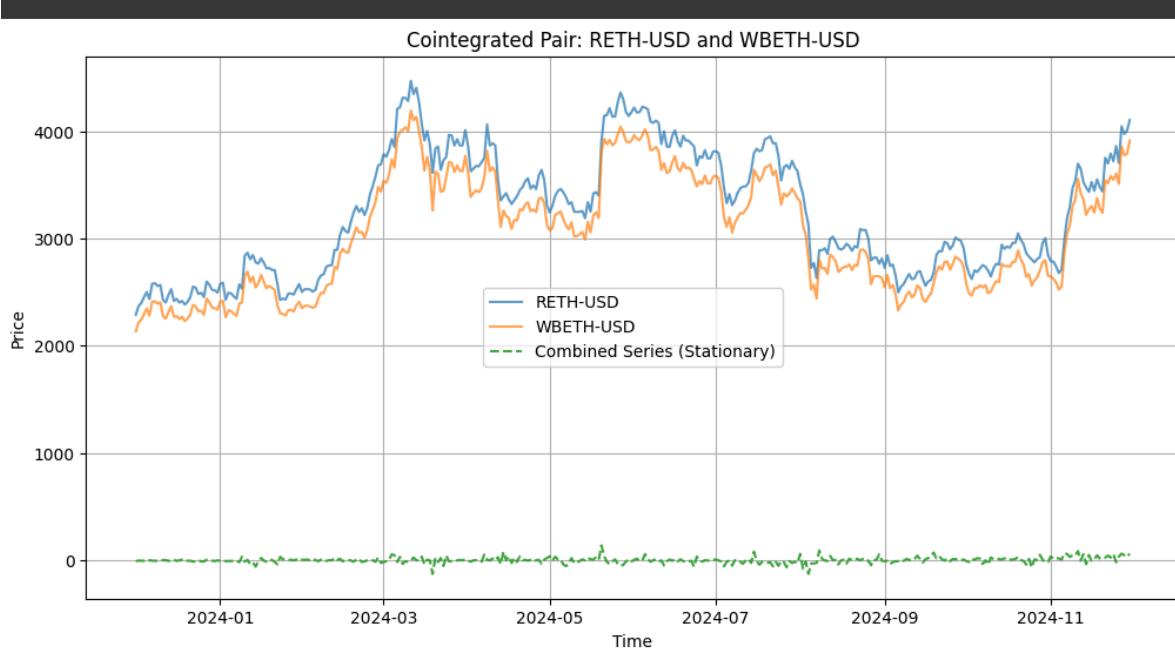
Pair: USDC-USD and BBTC31369-USD
P-Value: 0.0000
Cointegration Statistic: -10.0883
Intercept: 12520439.2734
Beta: -12454149.1839



Pair: WEETH-USD and RETH-USD
P-Value: 0.0000
Cointegration Statistic: -9.3612
Intercept: 13.1270
Beta: 1.0626



Pair: RETH-USD and WBETH-USD
P-Value: 0.0000
Cointegration Statistic: -9.3559
Intercept: -8.9565
Beta: 0.9406



Section4

Mean reversion strategy

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def mean_reversion_strategy(combined_series, half_life):
    """
    Apply the mean-reversion trading strategy on a stationary, mean-reverting series.

    Parameters:
        combined_series (pd.Series): The stationary combined series (residuals of cointegrated pair).
        half_life (int): The half-life of the series.

    Returns:
        trades (list): A list of executed trades (entry/exit points).
        pnl (float): The total profit/loss from the strategy.
    """
    # Adjust the look-back period
    look_back_period = max(20, int(np.ceil(half_life / 20.0)) * 20) # Ensure it's >= 20 and a multiple of half-life

    # Calculate EMA and SD
    ema = combined_series.ewm(span=look_back_period, adjust=False).mean()
    sd = combined_series.rolling(window=look_back_period).std()

    # Initialize variables for tracking positions and trades
    position = 0 # Current position size (positive for buy, negative for sell)
    trades = [] # List to store trade details
    pnl = 0 # Total profit/loss
    entry_price = 0 # Price at which the position was entered
```

```
# Iterate through the time series to apply the strategy
for t in range(look_back_period, len(combined_series)):
    price = combined_series.iloc[t]
    ema_t = ema.iloc[t]
    sd_t = sd.iloc[t]

    # Entry Conditions
    if price > ema_t + 2 * sd_t: # Price exceeds 2 SD above EMA
        if position == 0: # No current position
            position = -0.5 # Enter a half-unit sell position
            entry_price = price
            trades.append((combined_series.index[t], "Sell", price, position))
        elif position == -0.5 and price > ema_t + 3 * sd_t: # Price exceeds 3 SD above EMA
            position = -1.0 # Add to the sell position
            trades.append((combined_series.index[t], "Sell (Add)", price, position))

    elif price < ema_t - 2 * sd_t: # Price drops below 2 SD below EMA
        if position == 0: # No current position
            position = 0.5 # Enter a half-unit buy position
            entry_price = price
            trades.append((combined_series.index[t], "Buy", price, position))
        elif position == 0.5 and price < ema_t - 3 * sd_t: # Price drops below 3 SD below EMA
            position = 1.0 # Add to the buy position
            trades.append((combined_series.index[t], "Buy (Add)", price, position))

    # Exit Conditions
    elif position < 0 and price < ema_t + 1 * sd_t: # Close sell position
        pnl += abs(position) * (entry_price - price) # Profit from the sell position
        trades.append((combined_series.index[t], "Sell Exit", price, position))
        position = 0 # Close position

    elif position > 0 and price > ema_t - 1 * sd_t: # Close buy position
        pnl += abs(position) * (price - entry_price) # Profit from the buy position
        trades.append((combined_series.index[t], "Buy Exit", price, position))
        position = 0 # Close position

return trades, pnl
```

این تابع یک استراتژی معاملاتی مبتنی بر بازگشت به میانگین را بر روی یک سری زمانی ایستا و بازگشت‌پذیر به میانگین اجرا می‌کند.

1. تنظیم دوره بازبینی (Look-back Period)

- دوره بازبینی بر اساس نیمه عمر تنظیم می‌شود. این دوره حداقل 20 است و مضربی از نیمه عمر می‌باشد تا تحلیل بهینه باشد.

2. محاسبه میانگین متحرک نمایی (EMA) و انحراف معیار (SD)

- EMA: میانگین متحرک نمایی سری ترکیبی که روند میانگین سری را نشان می‌دهد.
- SD: انحراف معیار متحرک که میزان نوسانات سری را در دوره بازبینی نشان می‌دهد.

3. اجرای استراتژی معاملاتی:

- شرایط ورود به معامله:

- اگر قیمت از 2 انحراف معیار بالای EMA فراتر رود و موقعیتی باز نباشد، یک موقعیت فروش نصف واحد باز می‌شود.
- اگر قیمت از 2 انحراف معیار پایین EMA کمتر شود و موقعیتی باز نباشد، یک موقعیت خرید نصف واحد باز می‌شود.
- در صورت ادامه حرکت قیمت به سمت 3 انحراف معیار، موقعیت به یک واحد کامل افزایش می‌یابد.

● شرایط خروج از معامله:

- برای موقعیت‌های فروش، اگر قیمت به کمتر از 1 انحراف معیار بالای EMA بازگردد، معامله بسته می‌شود و سود/زیان محاسبه می‌شود.
- برای موقعیت‌های خرید، اگر قیمت به بیشتر از 1 انحراف معیار پایین EMA بازگردد، معامله بسته می‌شود و سود/زیان محاسبه می‌شود.

4. ثبت معاملات و محاسبه سود/زیان (PnL)

- تمامی معاملات انجامشده (نقطه ورود و خروج) در لیستی ذخیره می‌شوند.
- سود یا زیان کلی از تمامی معاملات محاسبه و برگردانده می‌شود.

Section5

```
def plot_trade_signals(combined_series, ema, sd, trades, title="Mean-Reversion Strategy"):  
    """  
    Plot the price series, EMA, SD bands, trade signals, and shaded areas for the strategy.  
  
    Parameters:  
        combined_series (pd.Series): The stationary combined series (residuals of cointegrated pair).  
        ema (pd.Series): Exponential Moving Average of the series.  
        sd (pd.Series): Standard Deviation of the series.  
        trades (list): List of executed trades (entry/exit points).  
        title (str): Title of the plot.  
    """  
    plt.figure(figsize=(12, 6))  
  
    plt.plot(combined_series.index, combined_series, label="Price", color="blue", alpha=0.7)  
  
    plt.plot(ema.index, ema, label="EMA", color="orange", linewidth=2)  
  
    plt.plot(ema.index, ema + 1 * sd, label="1 * SD", color="purple", linestyle="dotted")  
    plt.plot(ema.index, ema - 1 * sd, color="purple", linestyle="dotted")  
    plt.plot(ema.index, ema + 2 * sd, label="2 * SD", color="green", linestyle="dashed")  
    plt.plot(ema.index, ema - 2 * sd, color="green", linestyle="dashed")  
    plt.plot(ema.index, ema + 3 * sd, label="3 * SD", color="red", linestyle="dashdot")  
    plt.plot(ema.index, ema - 3 * sd, color="red", linestyle="dashdot")  
  
    plt.fill_between(ema.index, ema + 2 * sd, ema + 3 * sd, color="red", alpha=0.2, label="Sell Zone")  
    plt.fill_between(ema.index, ema + 3 * sd, combined_series.max(), color="red", alpha=0.3, label="Full Sell Zone")  
  
    for trade in trades:  
        t, action, price, position = trade  
        if "Sell" in action:  
            plt.scatter(t, price, color="red", label="Sell Signal" if "Sell Signal" not in plt.gca().get_legend_handles_labels()[1] else None)  
            plt.annotate(action, (t, price), textcoords="offset points", xytext=(-15, 10), ha='center', color="red")  
        elif "Buy" in action:  
            plt.scatter(t, price, color="green", label="Buy Signal" if "Buy Signal" not in plt.gca().get_legend_handles_labels()[1] else None)  
            plt.annotate(action, (t, price), textcoords="offset points", xytext=(-15, 10), ha='center', color="green")  
  
    plt.title(title)  
    plt.xlabel("Time")  
    plt.ylabel("Price")  
    plt.legend()  
    plt.grid()  
    plt.show()
```

این
تابع
برای

ترسیم سیگنال‌های معاملاتی استراتژی بازگشت به میانگین استفاده می‌شود. ورودی‌های آن شامل سری زمانی قیمت، میانگین متحرک نمایی (EMA)، انحراف معیار (SD) و لیستی از معاملات انجام‌شده است. تابع نموداری ایجاد می‌کند که در آن:

- سری زمانی قیمت ترسیم می‌شود.
- EMA و باندهای انحراف معیار (۱، ۲ و ۳ SD) نمایش داده می‌شوند.
- نواحی خرید و فروش با رنگ‌آمیزی مشخص می‌گردند.
- سیگنال‌های معاملاتی (نقاط ورود و خروج) با علامت‌ها و توضیحات روی نمودار نشان داده می‌شوند.

هدف این تابع، تجسم بصری استراتژی بازگشت به میانگین است تا تحلیلگر بتواند به سادگی عملکرد استراتژی و نقاط بهینه ورود و خروج را مشاهده و بررسی کند.

Sharpe ratio

```
def calculate_sharpe_ratio(strategy_returns, risk_free_rate=0):
    """
    Calculate the Sharpe Ratio for the strategy.

    Parameters:
        strategy_returns (pd.Series): The percentage returns of the strategy.
        risk_free_rate (float): The risk-free rate of return (default is 0).

    Returns:
        float: The Sharpe Ratio.
    """
    # Calculate the excess returns (strategy returns - risk-free rate)
    excess_returns = strategy_returns - risk_free_rate

    # Calculate the Sharpe Ratio
    sharpe_ratio = excess_returns.mean() / excess_returns.std()

    # Annualize the Sharpe Ratio if the returns are daily
    sharpe_ratio_annualized = sharpe_ratio * np.sqrt(252) # Assuming 252 trading days in a year

    return sharpe_ratio_annualized
```

محاسبه نسبت شارپ سالانه‌شده برای یک استراتژی معاملاتی.

1. بازده‌های مازاد را با کسر نرخ بدون ریسک از بازده‌های استراتژی محاسبه می‌کند.
2. نسبت شارپ را با تقسیم میانگین بازده‌های مازاد بر انحراف معیار آن‌ها محاسبه می‌کند.
3. نسبت شارپ را سالانه‌سازی می‌کند (فرض بر بازده‌های روزانه و ۲۵۲ روز معاملاتی در سال).

Plot equity

```
def plot_equity_curve(strategy_returns, pair_name):
    """
    Plot the Equity Curve for the strategy.

    Parameters:
        strategy_returns (pd.Series): The percentage returns of the strategy.
        pair_name (str): The name of the cointegrated pair.
    """
    # Calculate the Equity Curve (cumulative returns)
    equity_curve = (1 + strategy_returns).cumprod() # Compound returns

    # Plot the Equity Curve
    plt.figure(figsize=(12, 6))
    plt.plot(equity_curve.index, equity_curve, label="Equity Curve", color="blue", linewidth=2)
    plt.title(f"Equity Curve for {pair_name}")
    plt.xlabel("Time")
    plt.ylabel("Cumulative Returns")
    plt.grid()
    plt.legend()
    plt.show()
```

اینتابع به شما امکان می‌دهد تا عملکرد تاریخی استراتژی معاملاتی خود را تجسم کنید و ببینید چگونه سرمایه شما با گذشت زمان و با استفاده از استراتژی مورد نظر رشد کرده است. این تجسم به ارزیابی کارایی استراتژی و تصمیم‌گیری‌های آتی کمک می‌کند.

Run Strategy

```

results = []
for i, row in cointegrated_pairs.iterrows():
    combined_series = row['Combined_series'] # Assume you have the combined series for each pair

    # Clean the combined_series
    combined_series = combined_series.replace([np.inf, -np.inf], np.nan) # Replace inf/-inf with NaN
    combined_series = combined_series.dropna() # Drop NaN values

    # Check if the series is empty after cleaning
    if len(combined_series) == 0:
        print(f"Skipping pair {row['Crypto 1']} & {row['Crypto 2']} due to insufficient data.")
        continue # Skip this pair if the series is empty

    # Calculate the half-life of the cleaned series
    half_life = calculate_half_life(combined_series)

    # Apply the strategy
    trades, pnl = mean_reversion_strategy(combined_series, half_life)

    # Calculate daily returns from the trades
    # Assuming the `mean_reversion_strategy` returns a list of returns or you calculate it separately
    strategy_returns = pd.Series([trade[2] for trade in trades], index=combined_series.index[:len(trades)]) # Example

    # Calculate Sharpe Ratio
    sharpe_ratio = calculate_sharpe_ratio(strategy_returns)

    # Plot Equity Curve
    pair_name = f"{row['Crypto 1']} & {row['Crypto 2']}"

    # Store results
    results.append({
        "Pair": f"{row['Crypto 1']} & {row['Crypto 2']}",
        "Trades": trades,
        "Total PnL": pnl,
        "Sharpe Ratio": sharpe_ratio
    })

# Adjust the look-back period
look_back_period = max(20, int(np.ceil(half_life / 20.0)) * 20) # Ensure it's >= 20 and a multiple of half-life

# Calculate EMA and SD
ema = combined_series.ewm(span=look_back_period, adjust=False).mean()
sd = combined_series.rolling(window=look_back_period).std()

# Plot the results
plot_trade_signals(combined_series, ema, sd, trades, title=f"Signal Strategy for {row['Crypto 1']} & {row['Crypto 2']}")
plot_equity_curve(strategy_returns, pair_name)

print()
print(f"Sharpe Ratio for {row['Crypto 1']} & {row['Crypto 2']} is {sharpe_ratio}")
print()
print()

```

این کد برای هر جفت همانباشته در لیست `cointegrated_pairs` مراحل زیر را انجام می‌دهد:

1. داده‌پردازی سری ترکیبی:

- تمیز کردن داده‌ها: مقادیر بی‌نهایت (`inf`, `-inf`) را با `NaN` جایگزین می‌کند و سپس مقادیر گمشده را حذف می‌کند.
- بررسی داده‌ها: اگر پس از تمیز کردن، سری زمانی خالی باشد، جفت را رد می‌کند و به جفت بعدی می‌رود.

2. محاسبه نیمه عمر (Half-Life)

- از روی سری ترکیبی تمیز شده، نیمه عمر سری را محاسبه می‌کند که نشان‌دهنده سرعت بازگشت به میانگین است.

3. اجرای استراتژی بازگشت به میانگین:

- اعمال استراتژی: تابع `mean_reversion_strategy` را با سری ترکیبی و نیمه عمر اجرا می‌کند و معاملات (`trades`) و سود/زیان کل (`pnl`) را به دست می‌آورد.

4. محاسبه بازده و نسبت شارپ:

- بازده‌های استراتژی: بازده‌های روزانه را از معاملات استخراج یا محاسبه می‌کند.
- محاسبه نسبت شارپ: از طریق تابع `calculate_sharpe_ratio`، نسبت شارپ استراتژی را محاسبه می‌کند.

5. ذخیره و نمایش نتایج:

- ذخیره نتایج: اطلاعات مربوط به جفت، معاملات، سود/زیان کل و نسبت شارپ را در لیست `results` ذخیره می‌کند.

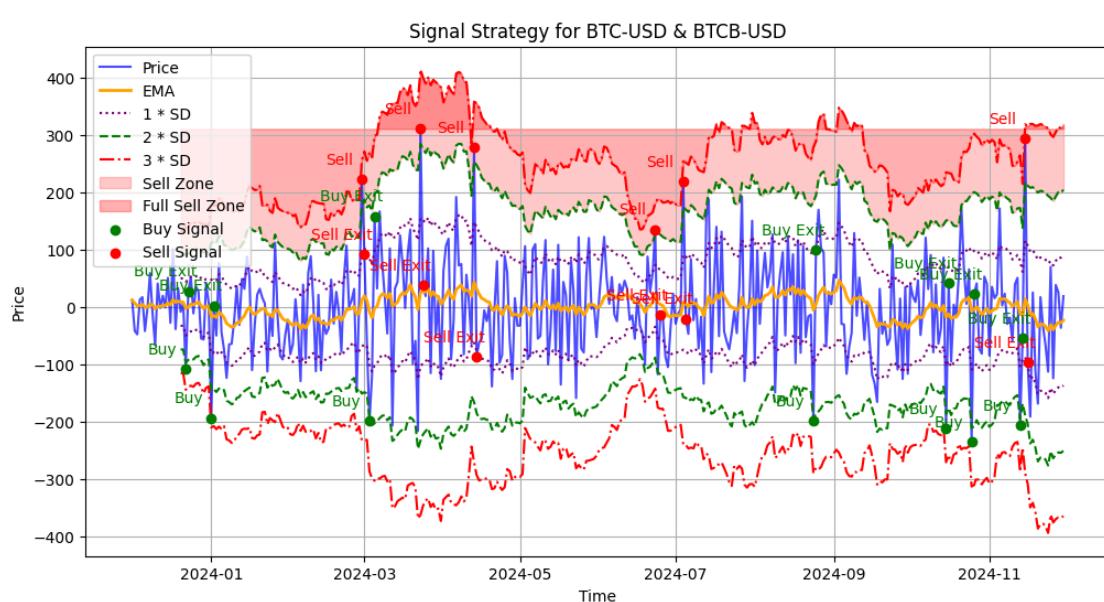
- محاسبه EMA و SD: دوره بازبینی را بر اساس نیمه عمر تنظیم کرده و میانگین متحرک نمایی و انحراف معیار را محاسبه می‌کند.
- ترسیم نمودارها:

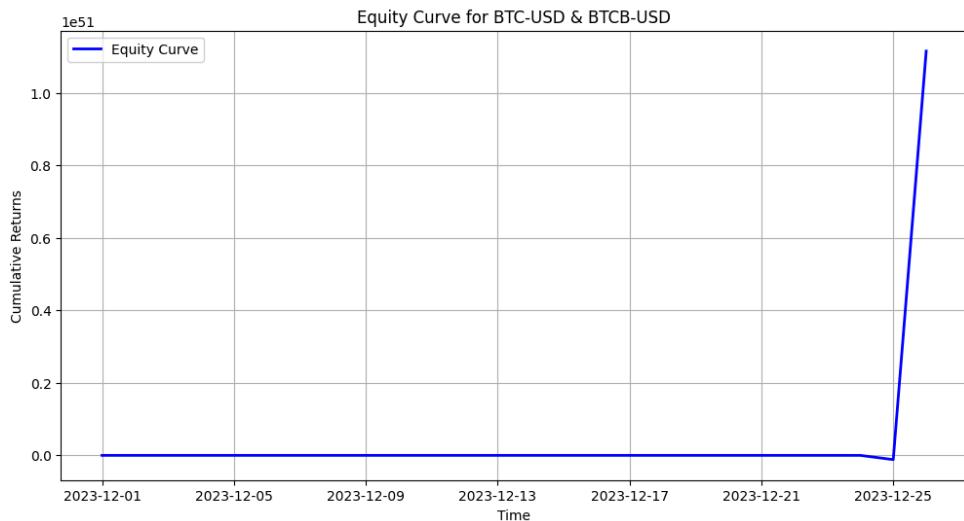
- سیگنال‌های معاملاتی: با استفاده از تابع `plot_trade_signals`

سیگنال‌های معاملاتی و باندهای انحراف معیار را ترسیم می‌کند.

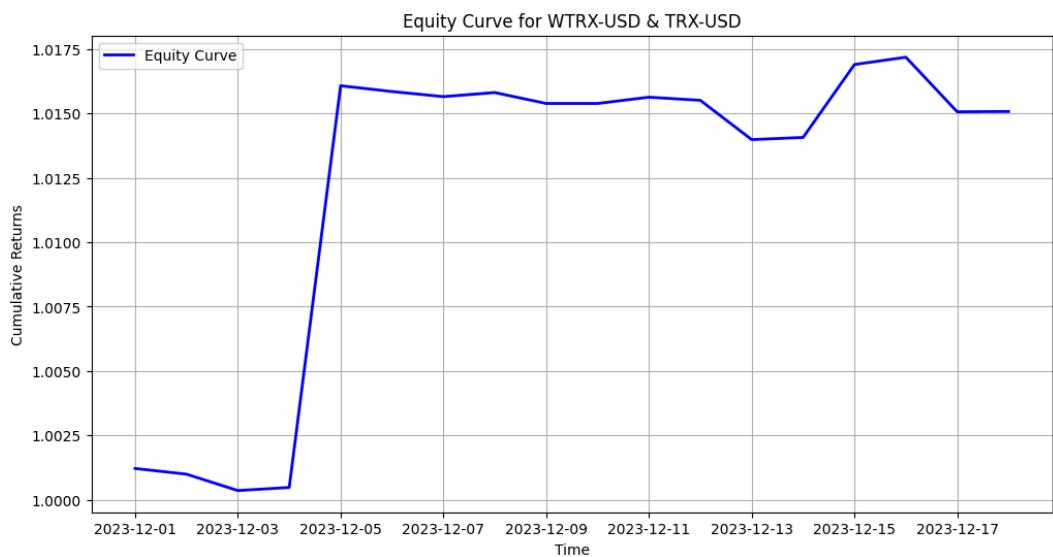
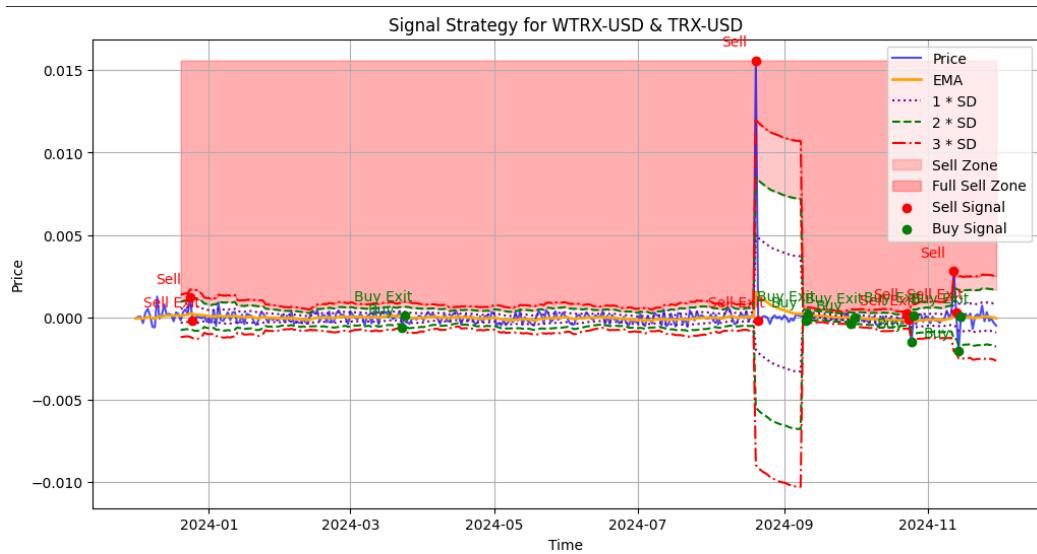
- منحنی ارزش دارایی: با تابع `plot_equity_curve`، منحنی رشد سرمایه را ترسیم می‌کند.

- نمایش اطلاعات: نسبت شارپ را برای هر جفت چاپ می‌کند.

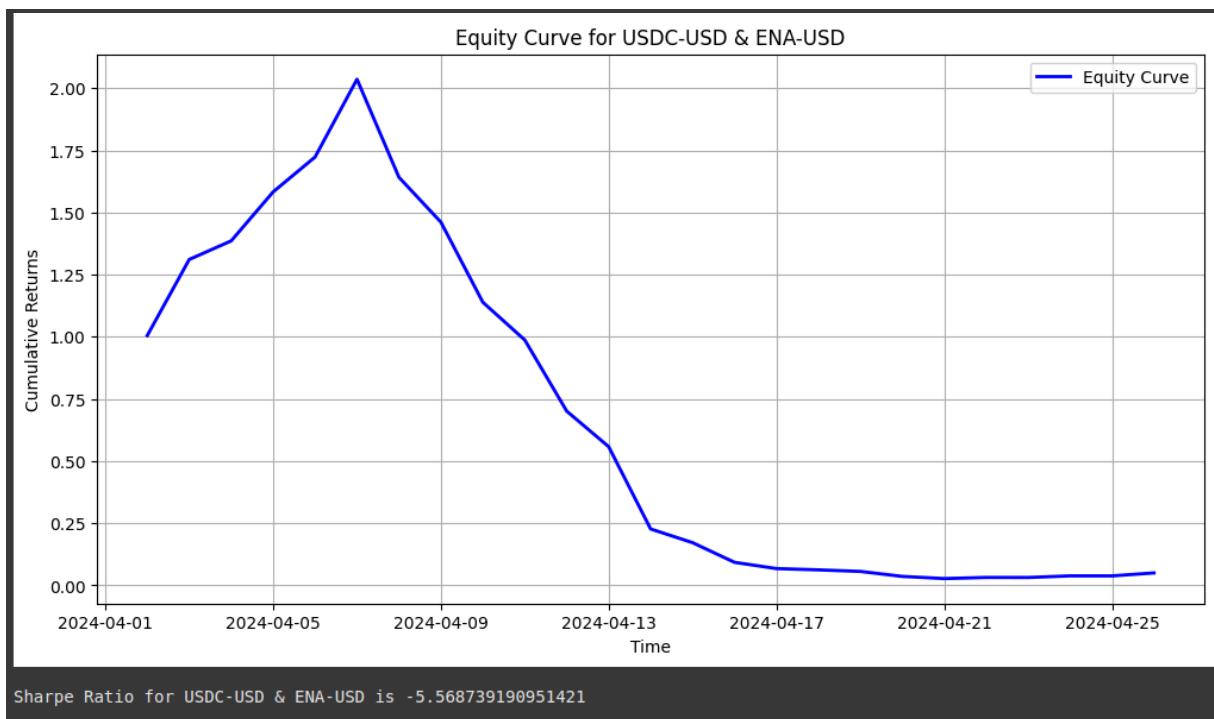
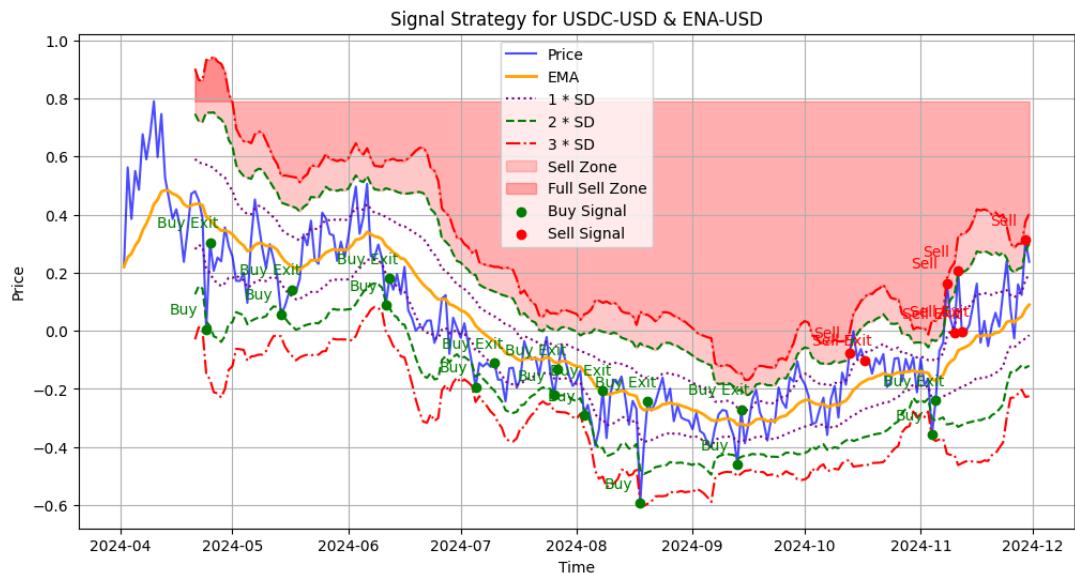


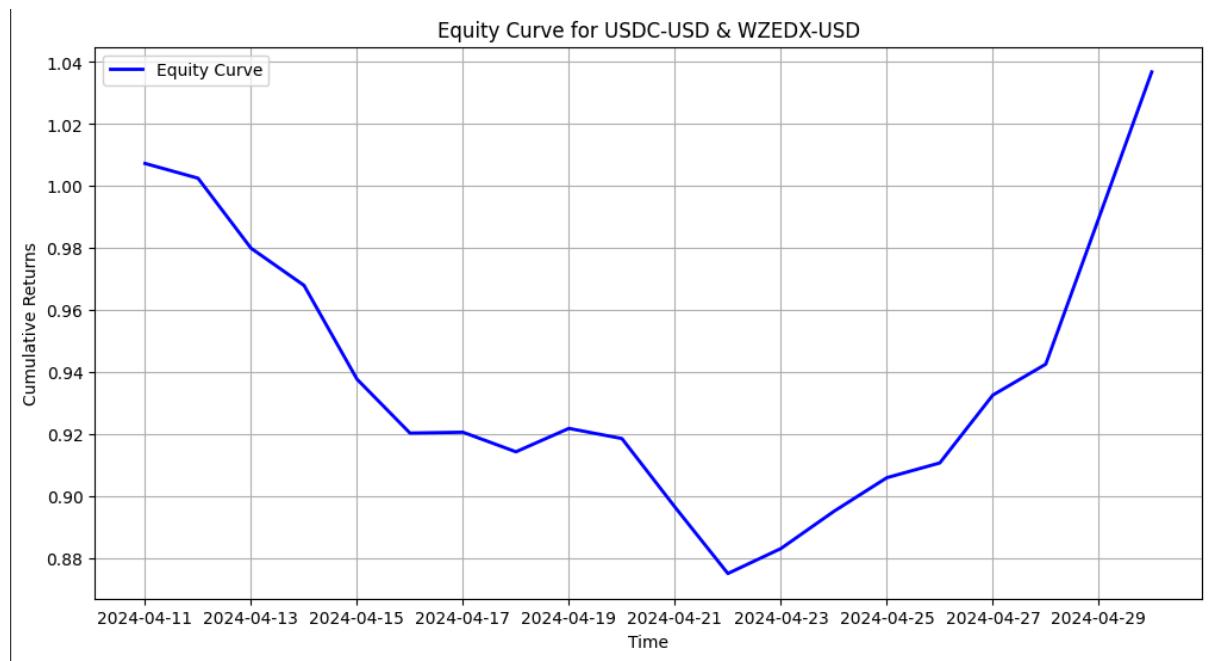
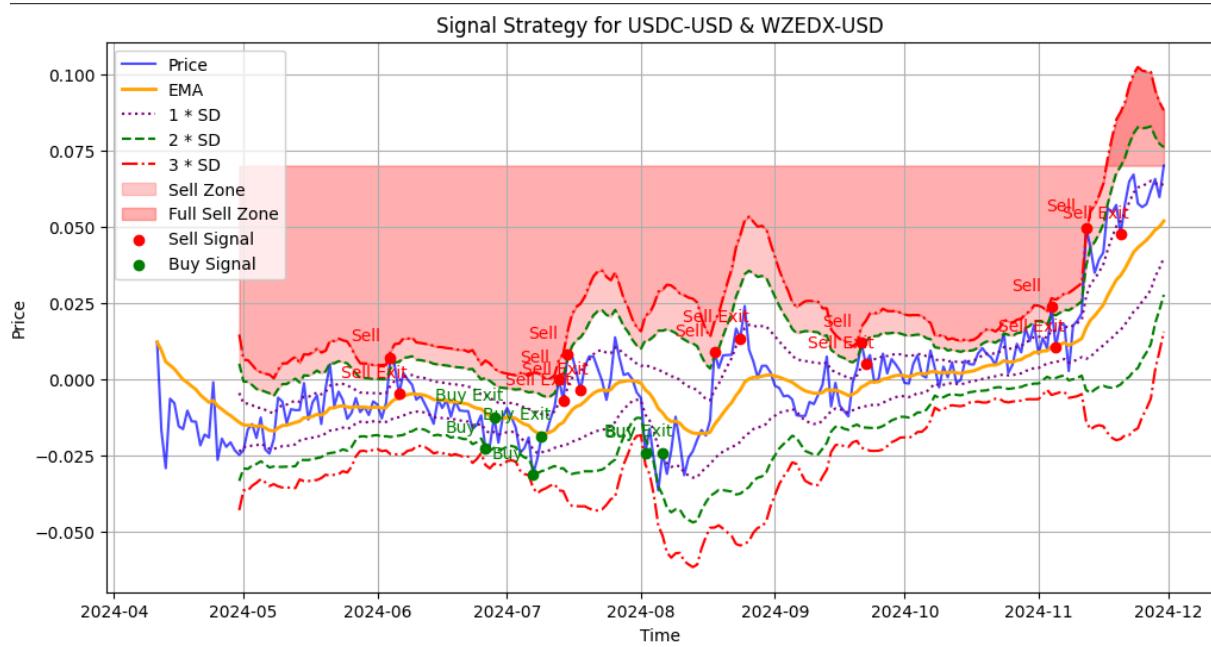


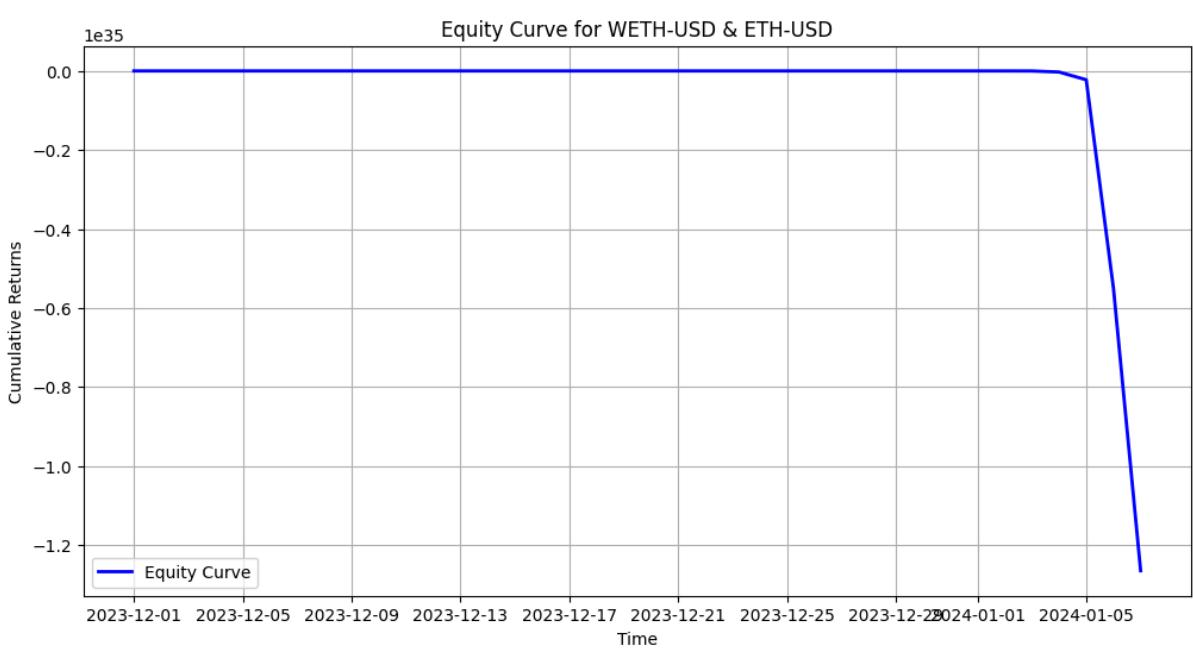
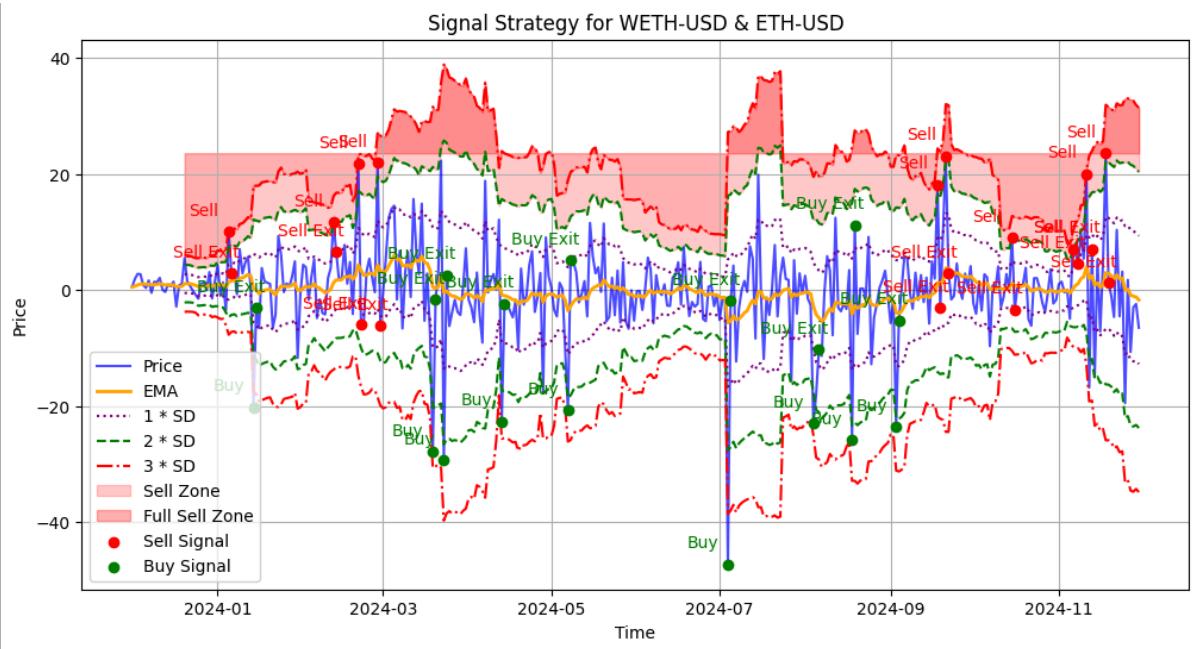
Sharpe Ratio for BTC-USD & BTCB-USD is 1.2052845680217883



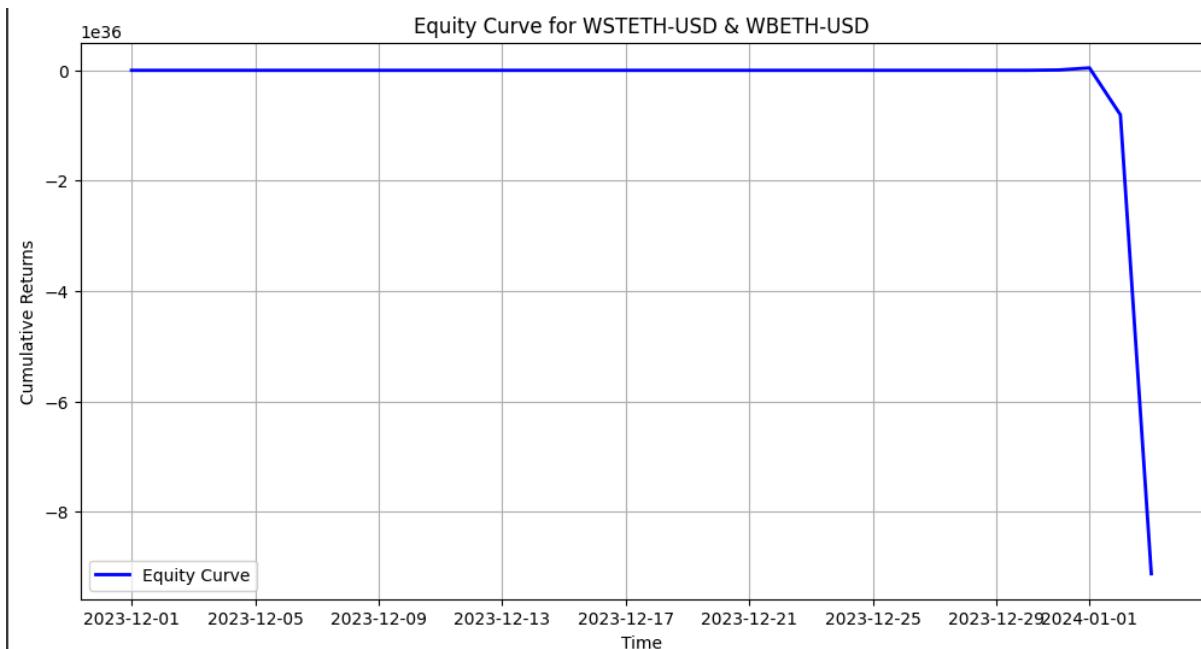
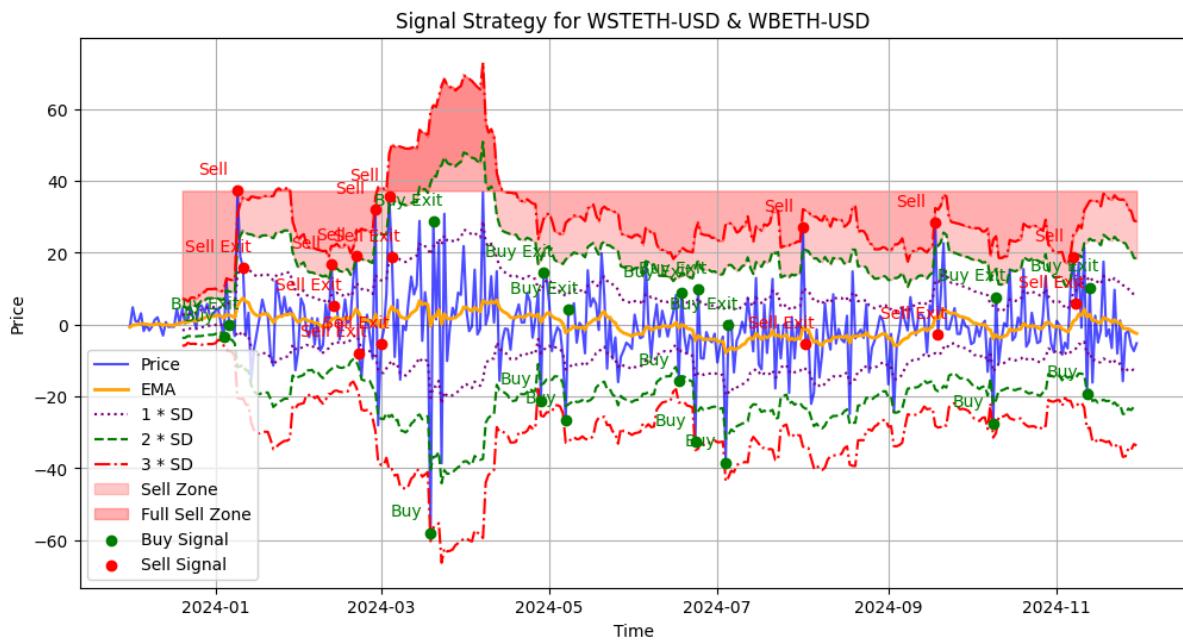
Sharpe Ratio for WTRX-USD & TRX-USD is 3.4909021819427766



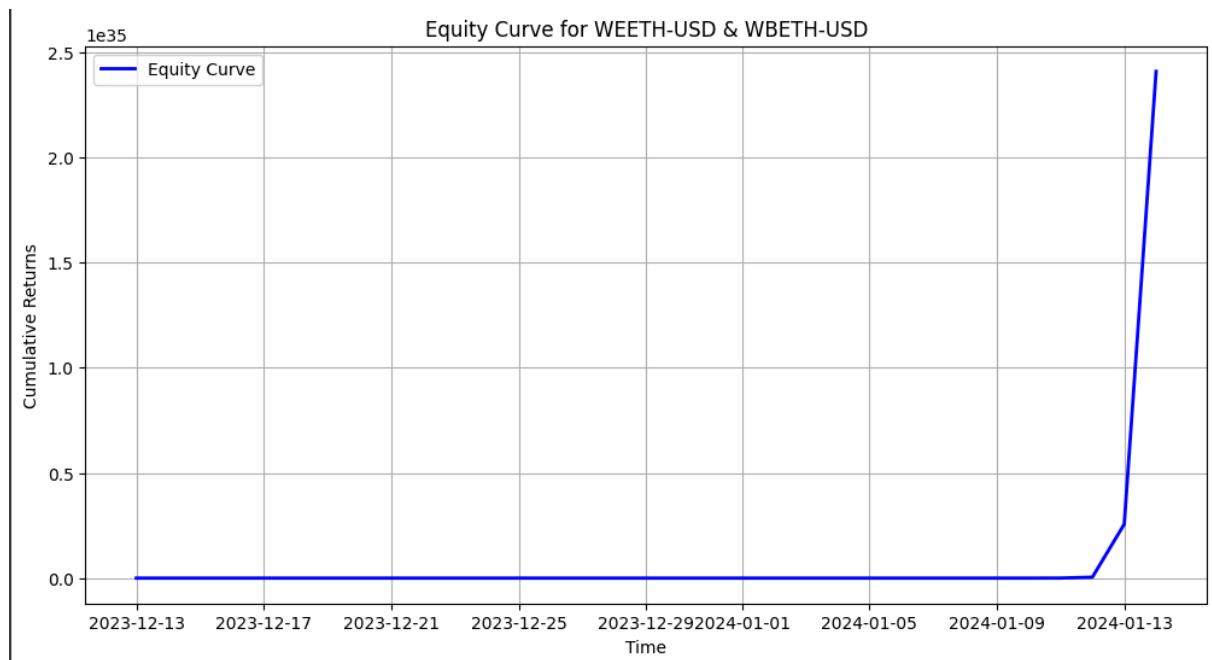
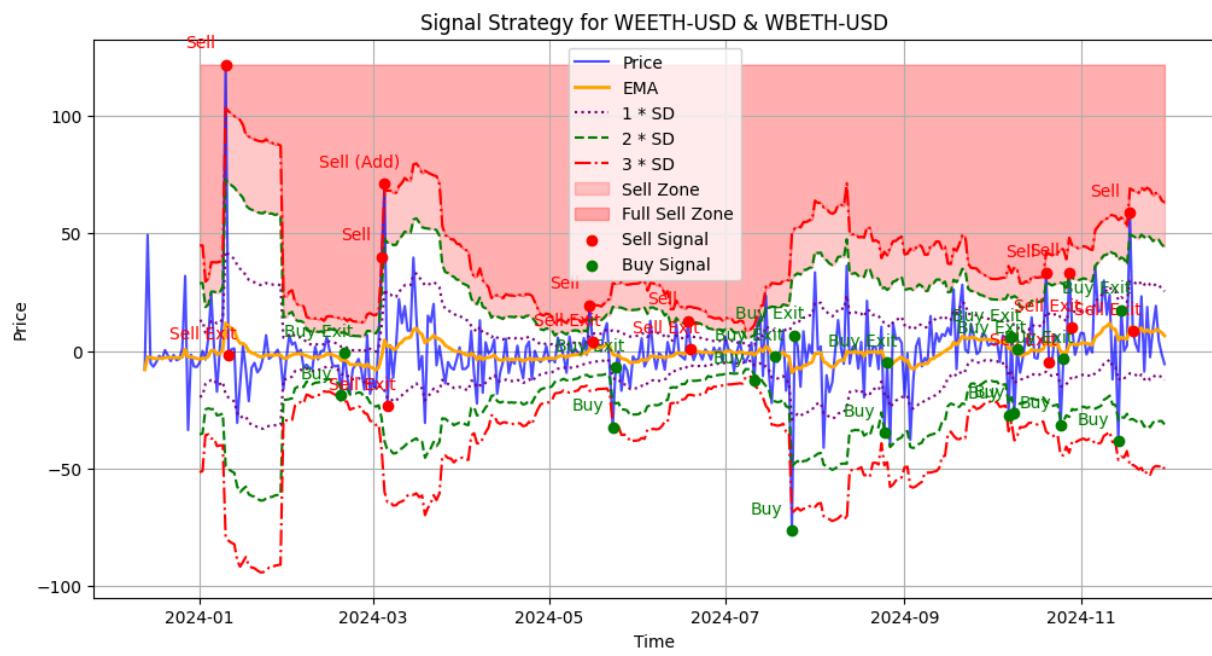




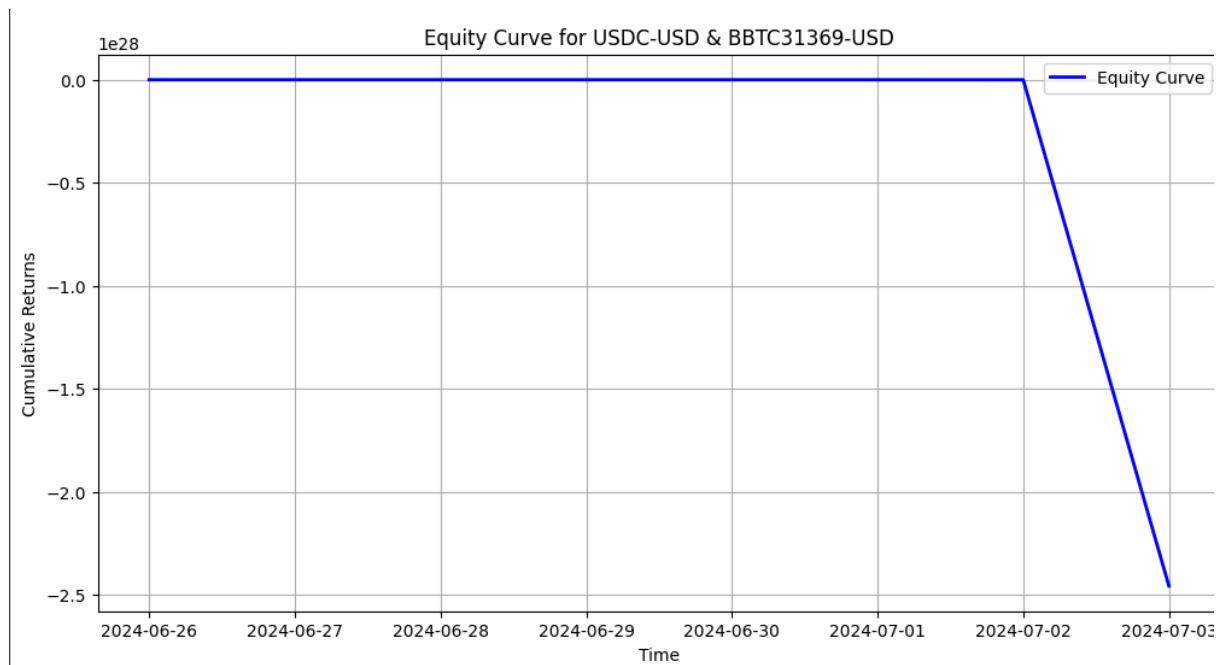
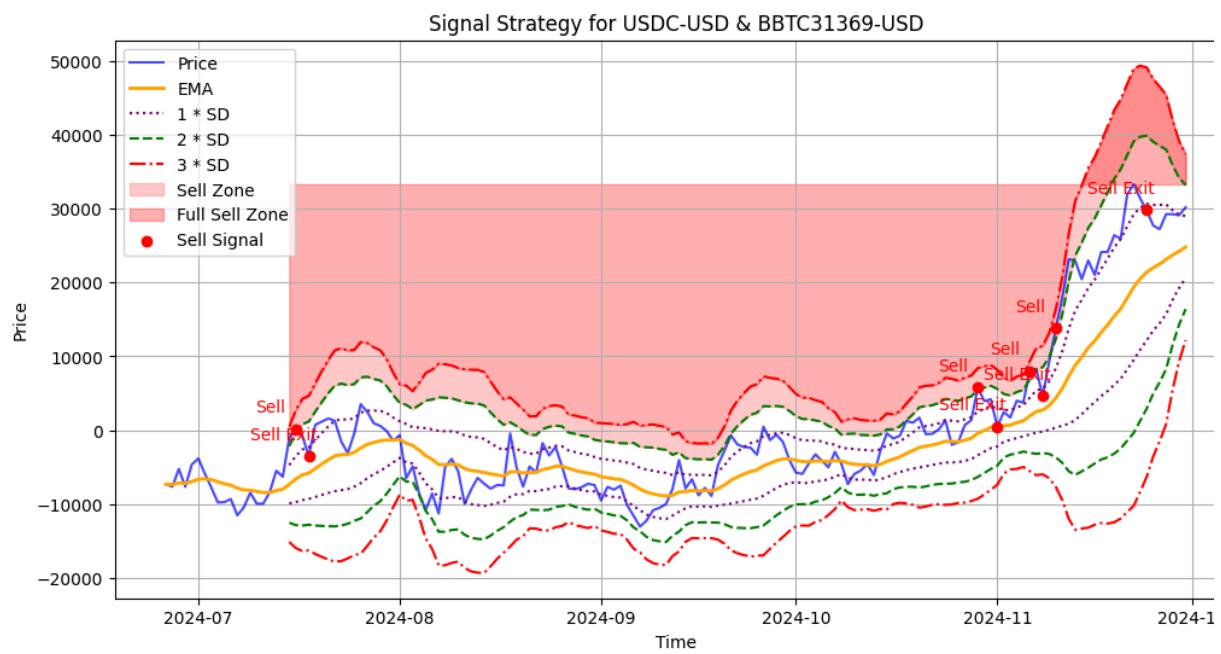
Sharpe Ratio for WETH-USD & ETH-USD is -1.802171925347042



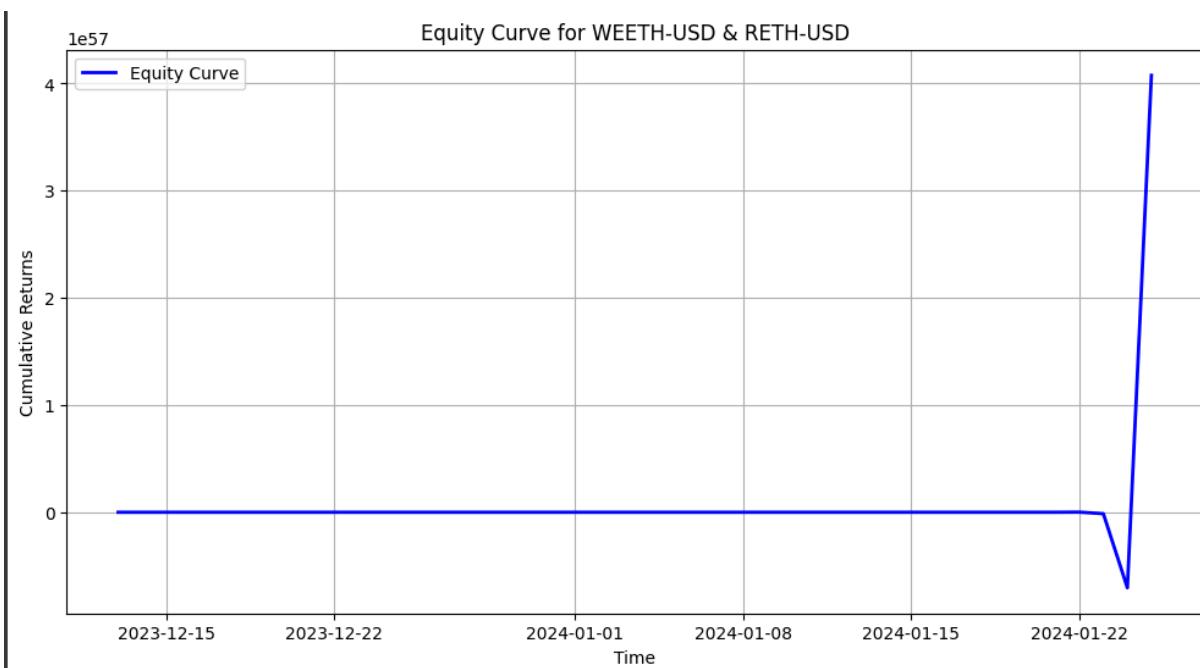
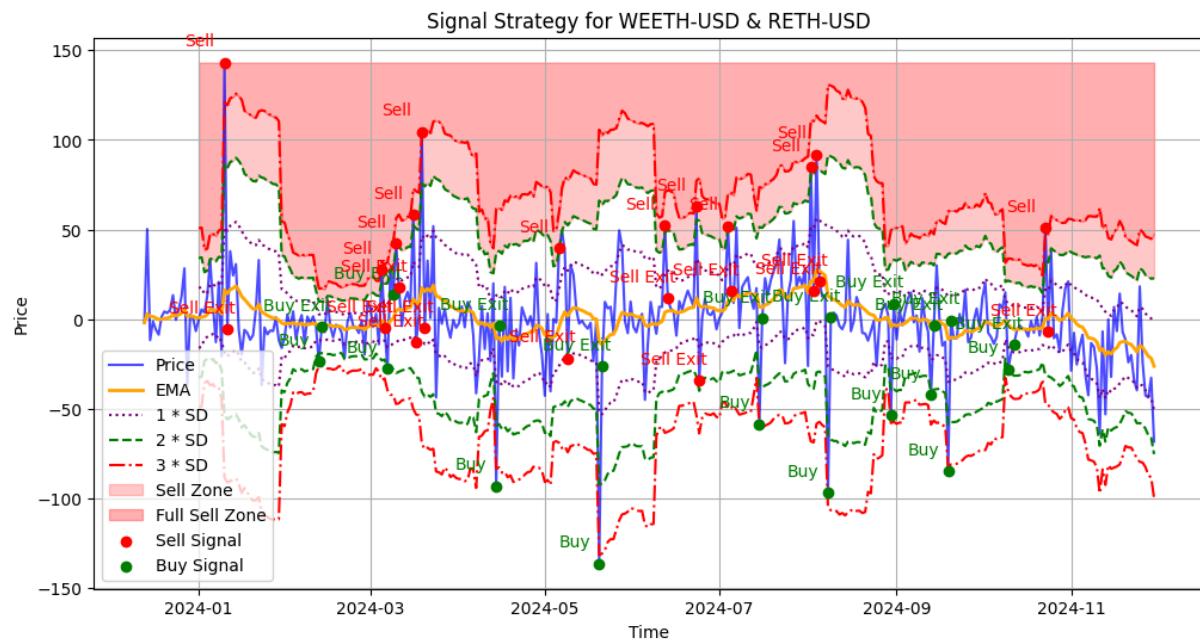
Sharpe Ratio for WSTETH-USD & WBETH-USD is 1.682604560079595



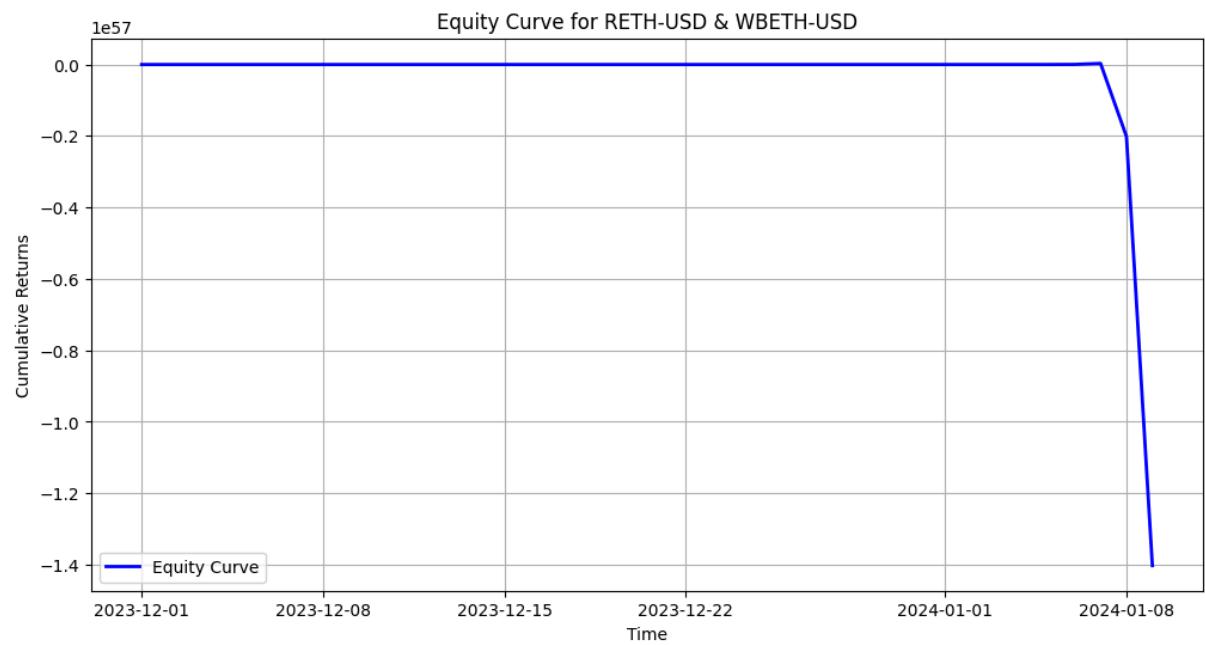
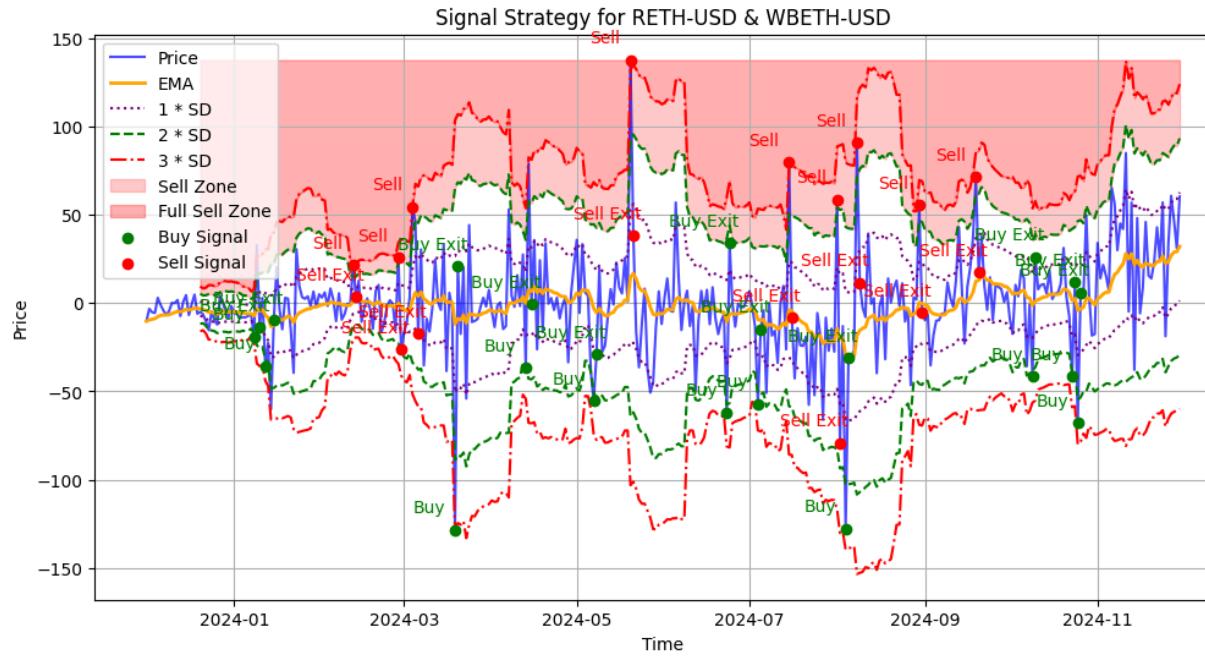
Sharpe Ratio for WEETH-USD & WBETH-USD is 1.2910315662347274



Sharpe Ratio for USDC-USD & BBTC31369-USD is 11.168523516559407



Sharpe Ratio for WEETH-USD & RETH-USD is 0.8264972243008658



Sharpe Ratio for RETH-USD & WBETH-USD is -1.0460491258958893

```

# Display the results
for result in results:
    print(f"Pair: {result['Pair']}")
    print(f"Total PnL: {result['Total PnL']:.2f}")
    print("Trades:")
    for trade in result['Trades']:
        print(f" Time: {trade[0]}, Action: {trade[1]}, Price: {trade[2]:.2f}, Position: {trade[3]}")
    print()

```

این کد برای نمایش نتایج اجرای استراتژی معاملاتی بر روی جفت‌های همانباشته استفاده می‌شود

Section6

Johanson_test

```

def johansen_test(df, det_order=-1, k_ar_diff=1):
    """
    Perform the Johansen cointegration test on a DataFrame of time series.

    Parameters:
        df (pd.DataFrame): DataFrame containing the time series to test.
        det_order (int): Deterministic trend assumption (-1 for no trend).
        k_ar_diff (int): Number of lags to include in the test.

    Returns:
        bool: Whether the series are cointegrated.
        np.ndarray: The eigenvector for the stationary combination.
    """
    result = coint_johansen(df, det_order, k_ar_diff)
    trace_stat = result.lrl1 # Trace statistics
    crit_values = result.cvt[:, 1] # Critical values at the 5% level

    # Check if the first trace statistic exceeds the critical value
    if trace_stat[0] > crit_values[0]:
        return True, result.evec[:, 0] # Return the first eigenvector
    else:
        return False, None

```

اجرای آزمون یوهانسن برای تعیین همانباشته‌گی بین سری‌های زمانی در یک DataFrame.

- اجرای آزمون با `coint_johansen`.
- مقایسه آماره‌ی اثر با مقدار بحرانی در سطح ۵٪.
- تعیین وجود یا عدم وجود همانباشته‌گی.

find cointegrated

```
from statsmodels.tsa.vector_ar.vecm import coint_johansen
def find_cointegrated_triplets(non_stationary):
    """
    Find the top 5 stationary combined series from triplets of non-stationary cryptocurrencies.

    Parameters:
        crypto_data (pd.DataFrame): Time series data for cryptocurrencies.

    Returns:
        pd.DataFrame: A DataFrame with the top 5 stationary combined series and their Hurst exponents.
    """
    # Step 1: Perform the Johansen test on all triplets of non-stationary cryptocurrencies
    results = []
    tickers = non_stationary
    from itertools import combinations

    for triplet in combinations(tickers, 3): # Iterate through all combinations of 3 cryptos
        series1 = crypto_data[triplet[0]].dropna()
        series2 = crypto_data[triplet[1]].dropna()
        series3 = crypto_data[triplet[2]].dropna()

        # Align all three series to the same length
        min_len = min(len(series1), len(series2), len(series3))
        series1 = series1[-min_len:]
        series2 = series2[-min_len:]
        series3 = series3[-min_len:]

        # Create a DataFrame for the triplet
        triplet_df = pd.DataFrame({
            triplet[0]: series1,
            triplet[1]: series2,
            triplet[2]: series3
        })

        # Perform the Johansen test
        cointegrated, eigenvector = johansen_test(triplet_df)
```

```
if cointegrated:
    # Create the stationary combined series using the eigenvector
    combined_series = triplet_df.values @ eigenvector
    combined_series = pd.Series(combined_series, index=triplet_df.index)

    # Calculate the Hurst exponent for the combined series
    hurst = calculate_hurst_exponent(combined_series)

    # Append the results
    results.append({
        "Crypto 1": triplet[0],
        "Crypto 2": triplet[1],
        "Crypto 3": triplet[2],
        "Combined_series": combined_series,
        "Hurst Exponent": hurst,
        "Eigenvector": eigenvector
    })

# Step 2: Sort results by Hurst exponent and select the top 5 stationary series
results = sorted(results, key=lambda x: x["Hurst Exponent"]) # Sort by Hurst exponent
top_5_results = results[:5] # Select the top 5

# Step 3: Save the results as a DataFrame
final_results = []
for result in top_5_results:
    final_results.append({
        "Crypto 1": result["Crypto 1"],
        "Crypto 2": result["Crypto 2"],
        "Crypto 3": result["Crypto 3"],
        "Hurst Exponent": result["Hurst Exponent"],
        "Combined_series": result["Combined_series"],
        "Eigenvector": result["Eigenvector"]
    })

# Convert to DataFrame
final_results_df = pd.DataFrame(final_results)
return final_results_df
```

هدف: یافتن ۵ سری ترکیبی ایستایی برتر از میان سه‌تایی‌های ارز‌های دیجیتال غیر ایستا با استفاده از آزمون همانباشتگی یوهانسن.

۱. تولید ترکیبات سه‌تایی:

- تمام ترکیبات ممکن سه‌تایی از ارز‌های دیجیتال غیر ایستا را ایجاد می‌کند.

۲. آزمون همانباشتگی یوهانسن:

- برای هر سه‌تایی:

- سری‌های زمانی را تراز (Align) می‌کند تا طول یکسانی داشته باشد.

- آزمون یوهانسن را برای بررسی همانباشتگی بین سه سری اجرا می‌کند.

- اگر همانباشتگی وجود داشته باشد:

- سری ترکیبی ایستا را با استفاده از بردار ویژه

- (Eigenvector) به دست آمده از آزمون ایجاد می‌کند.

- نمایه‌ی هرست (Hurst Exponent) سری ترکیبی را

- محاسبه می‌کند.

- نتایج را ذخیره می‌کند:

- نام ارز‌های دیجیتال.

- سری ترکیبی ایستا.

- نمایه‌ی هرست.

- بردار ویژه.

۳. انتخاب بهترین سری‌های ترکیبی:

- نتایج را بر اساس نمایه‌ی هرست مرتب می‌کند (کم به زیاد).

- ۵ سری ترکیبی ایستایی برتر را انتخاب می‌کند (با کمترین نمایه‌ی هرست).

۴. تهیه DataFrame نهایی:

- نتایج منتخب را در یک DataFrame جمع‌آوری می‌کند.

- DataFrame شامل:

- نام سه ارز دیجیتال.

- سری ترکیبی ایستا.

- نمایه‌ی هرست.

- بردار ویژه.

```
three_cointegrated = find_cointegrated_triplets(non_stationary_cryptos[:26])
```

یافتن سه‌تایی‌های همانباشتگی از میان ۲۶ ارز دیجیتال غیر ایستا.

```
print(three_cointegrated["Eigenvector"])
```

این کد، ستون "Eigenvector" را از DataFrame three_cointegrated انتخاب کرده و محتوای آن را چاپ می‌کند.

```

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Assuming 'final_results_df' is your DataFrame from the previous steps
# Replace 'crypto_data' with your actual DataFrame containing the price data

# Print and plot the cointegrated triplets
print("Cointegrated Triplets:")
for index, row in three_cointegrated.iterrows():
    crypto1 = row['Crypto 1']
    crypto2 = row['Crypto 2']
    crypto3 = row['Crypto 3']
    hurst_exponent = row['Hurst Exponent']
    combined_series = row['Combined_series']
    Eigenvector = row['Eigenvector']

    print()
    print()
    print(f"Triplet: {crypto1}, {crypto2}, {crypto3}")
    print(f"Hurst Exponent: {hurst_exponent:.4f}")
    print(f"Eigenvector (coefficients): {Eigenvector}")
    print()

    # Get the time series data
    # Ensure that 'crypto_data' is your DataFrame containing the price data
    series1 = crypto_data[crypto1].dropna()
    series2 = crypto_data[crypto2].dropna()
    series3 = crypto_data[crypto3].dropna()

    # Align all three series and the combined series to the same length
    min_len = min(len(series1), len(series2), len(series3), len(combined_series))
    series1 = series1[-min_len:]
    series2 = series2[-min_len:]
    series3 = series3[-min_len:]
    combined_series = combined_series[-min_len:]

```

```

# Create a DataFrame for plotting
plot_df = pd.DataFrame({
    crypto1: series1.values,
    crypto2: series2.values,
    crypto3: series3.values,
    'Combined Series': combined_series.values
}, index=series1.index)

# Plot the time series
plt.figure(figsize=(12, 6))
plt.plot(plot_df[crypto1], label=f"{crypto1}")
plt.plot(plot_df[crypto2], label=f"{crypto2}")
plt.plot(plot_df[crypto3], label=f"{crypto3}")

# Plot the combined stationary series on a secondary y-axis
ax1 = plt.gca()
ax2 = ax1.twinx()
ax2.plot(plot_df['Combined Series'], label="Combined Series (Stationary)", linestyle="--", color='black', alpha=0.9)

# Combine legends from both y-axes
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, labels1 + labels2, loc='best')

plt.title(f"Cointegrated Triplet: {crypto1}, {crypto2}, {crypto3}")
plt.xlabel("Time")
ax1.set_ylabel("Price")
ax2.set_ylabel("Combined Series Value")
plt.grid()
plt.show()

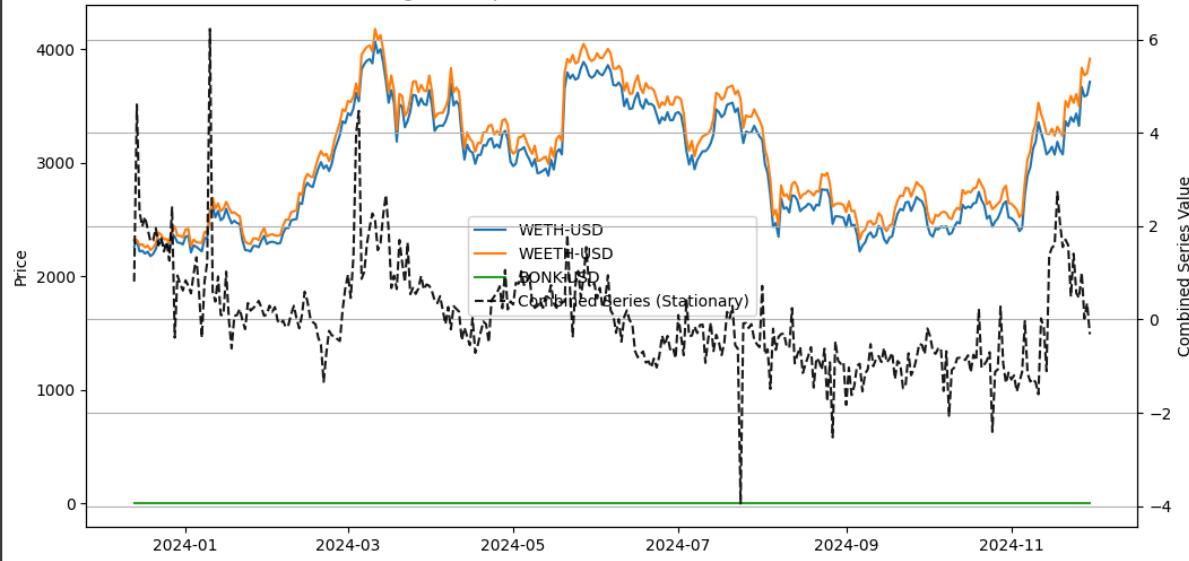
```

Triplet: WETH-USD, WEETH-USD, BONK-USD

Hurst Exponent: 0.4291

Eigenvector (coefficients): [4.40397700e-02 -4.33311134e-02 1.29524652e+05]

Cointegrated Triplet: WETH-USD, WEETH-USD, BONK-USD

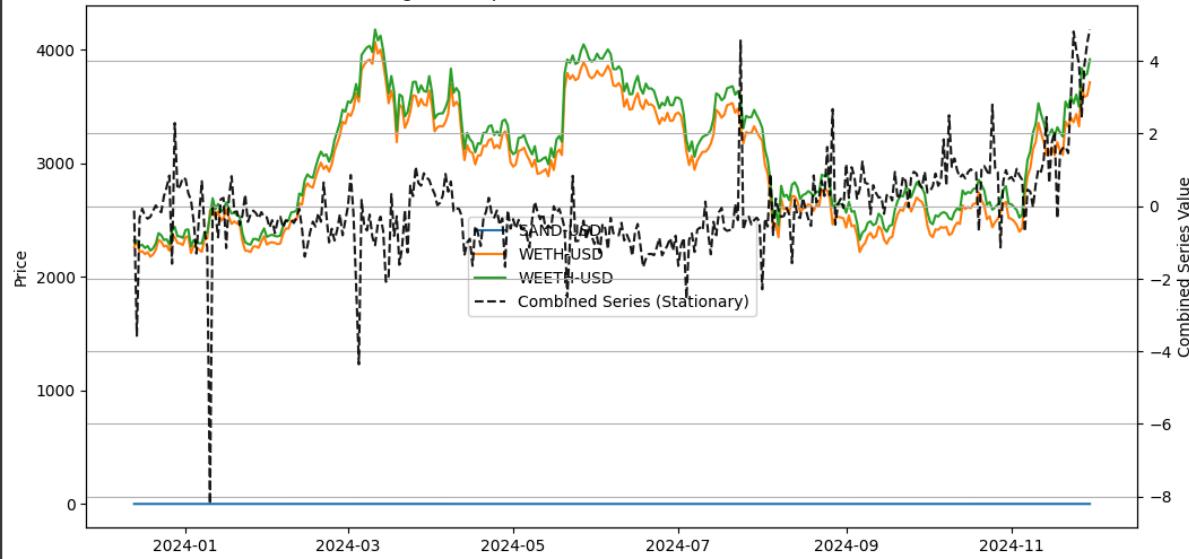


Triplet: SAND-USD, WETH-USD, WEETH-USD

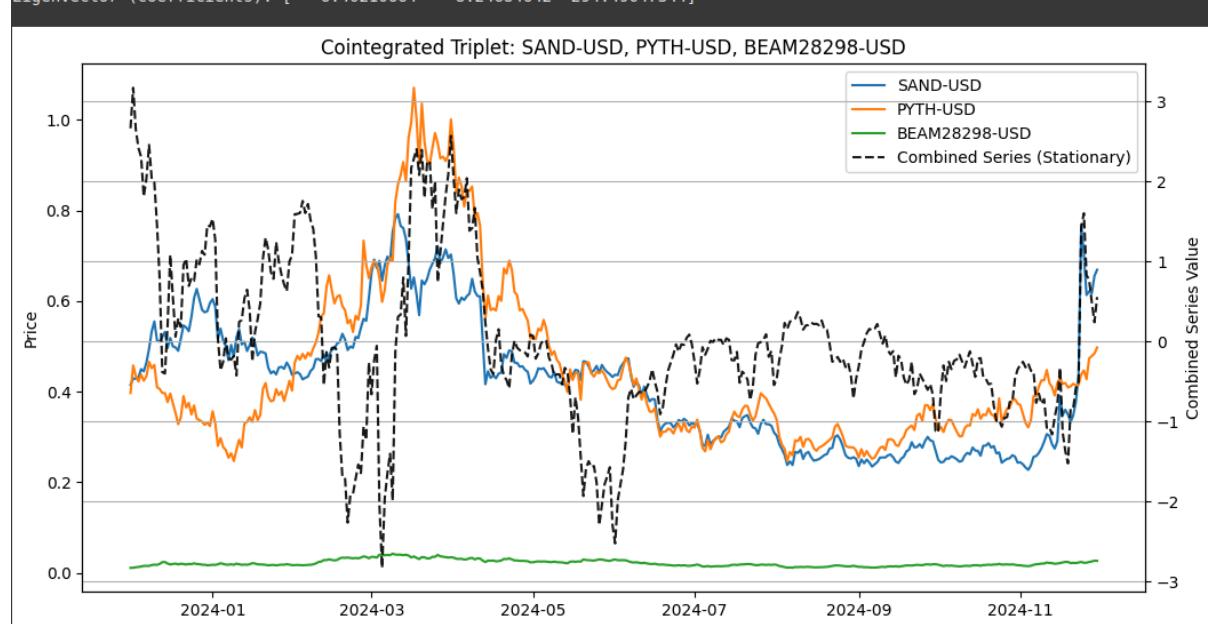
Hurst Exponent: 0.4678

Eigenvector (coefficients): [9.13691829 -0.06590022 0.06217397]

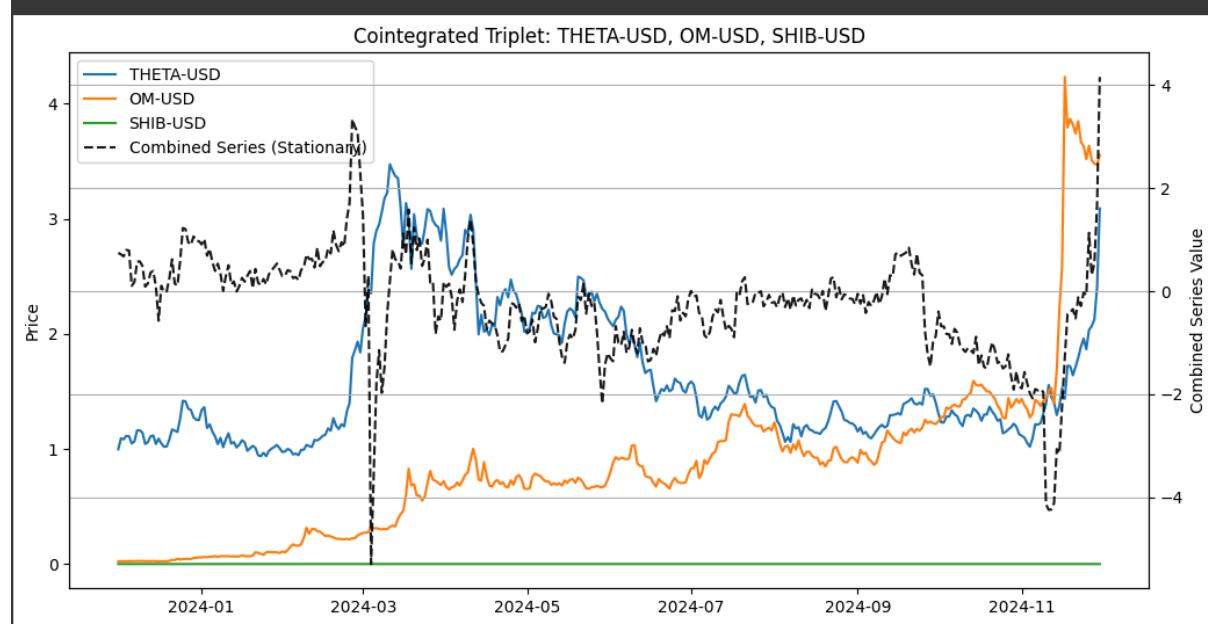
Cointegrated Triplet: SAND-USD, WETH-USD, WEETH-USD



Triplet: SAND-USD, PYTH-USD, BEAM28298-USD
Hurst Exponent: 0.4807
Eigenvector (coefficients): [6.40210664 8.24684642 -294.49047344]



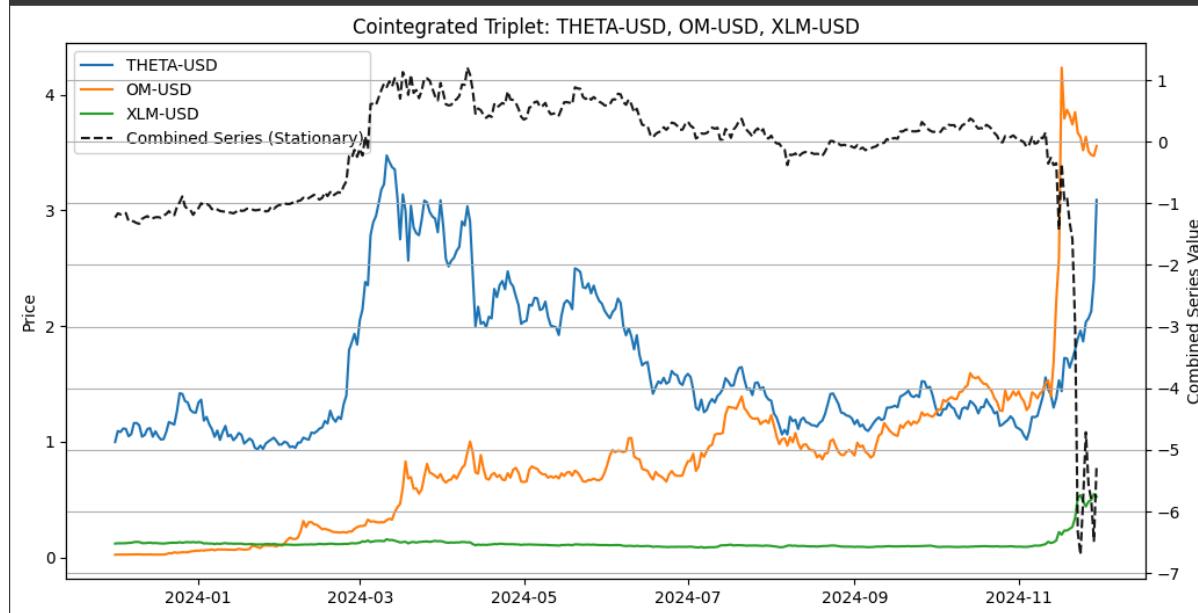
Triplet: THETA-USD, OM-USD, SHIB-USD
Hurst Exponent: 0.4898
Eigenvector (coefficients): [4.13330588e+00 8.01442213e-01 -4.24459742e+05]



```

Triplet: THETA-USD, OM-USD, XLM-USD
Hurst Exponent: 0.4912
Eigenvector (coefficients): [ 1.09442313  0.44815288 -19.49169674]

```



Run strategy

```

results = []
for i, row in three_cointegrated.iterrows():
    combined_series = row['Combined_series'] # Assume you have the combined series for each pair

    # Clean the combined_series
    combined_series = combined_series.replace([np.inf, -np.inf], np.nan) # Replace inf/-inf with NaN
    combined_series = combined_series.dropna() # Drop NaN values

    # Check if the series is empty after cleaning
    if len(combined_series) == 0:
        print(f"Skipping pair {row['Crypto 1']} & {row['Crypto 2']} due to insufficient data.")
        continue # Skip this pair if the series is empty

    # Calculate the half-life of the cleaned series
    half_life = calculate_half_life(combined_series)

    # Apply the strategy
    trades, pnl = mean_reversion_strategy(combined_series, half_life)

    # Calculate daily returns from the trades
    # Assuming the 'mean_reversion_strategy' returns a list of returns or you calculate it separately
    strategy_returns = pd.Series([trade[2] for trade in trades], index=combined_series.index[:len(trades)]) # Example

    # Calculate Sharpe Ratio
    sharpe_ratio = calculate_sharpe_ratio(strategy_returns)

    # Plot Equity Curve
    pair_name = f"{row['Crypto 1']} & {row['Crypto 2']} & {row['Crypto 3']}"

    # Store results
    results.append({
        "Pair": f"{row['Crypto 1']} & {row['Crypto 2']}",
        "Trades": trades,
        "Total PnL": pnl,
        "Sharpe Ratio": sharpe_ratio
    })

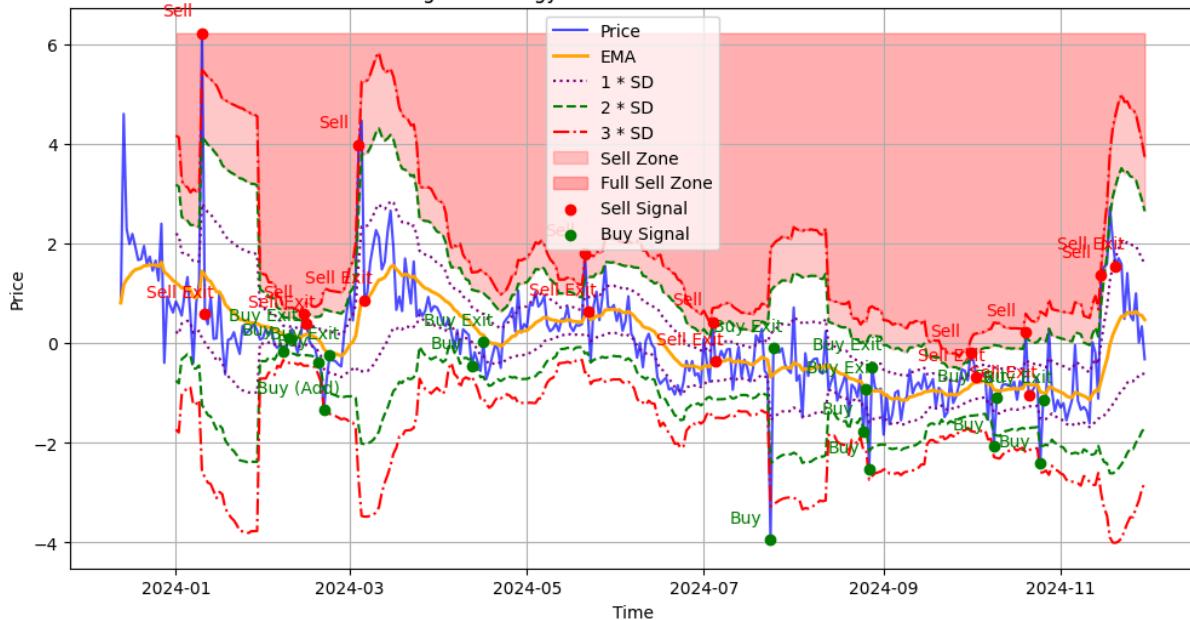
```

هدف: اجرای استراتژی معاملاتی بازگشت به میانگین بر روی سری‌های ترکیبی ایستای حاصل از سه‌تایی‌های همانباشتگی و ارزیابی عملکرد آن‌ها.

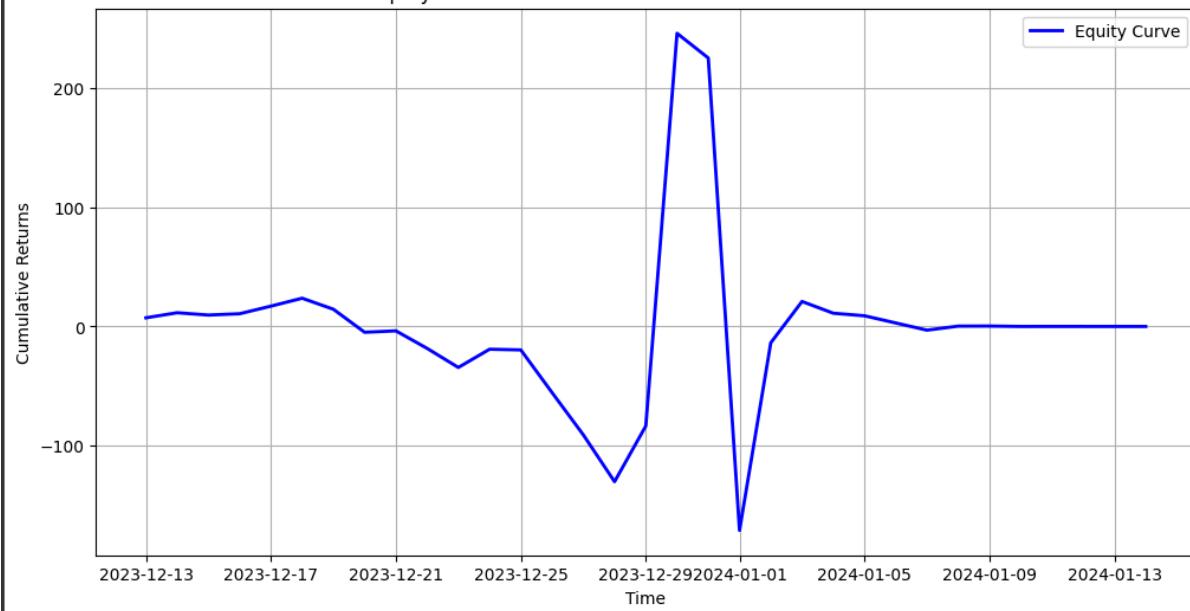
- تمیز کردن و آماده‌سازی داده‌ها.
- محاسبه پارامترهای کلیدی مانند نیمه‌عمر.

- اجرای استراتژی معاملاتی و جمع آوری نتایج.
- ارزیابی عملکرد با معیارهایی مثل نسبت شارپ.
- تجسم نتایج برای تحلیل بهتر.
- ذخیره و نمایش نتایج برای تحلیل‌های بیشتر.

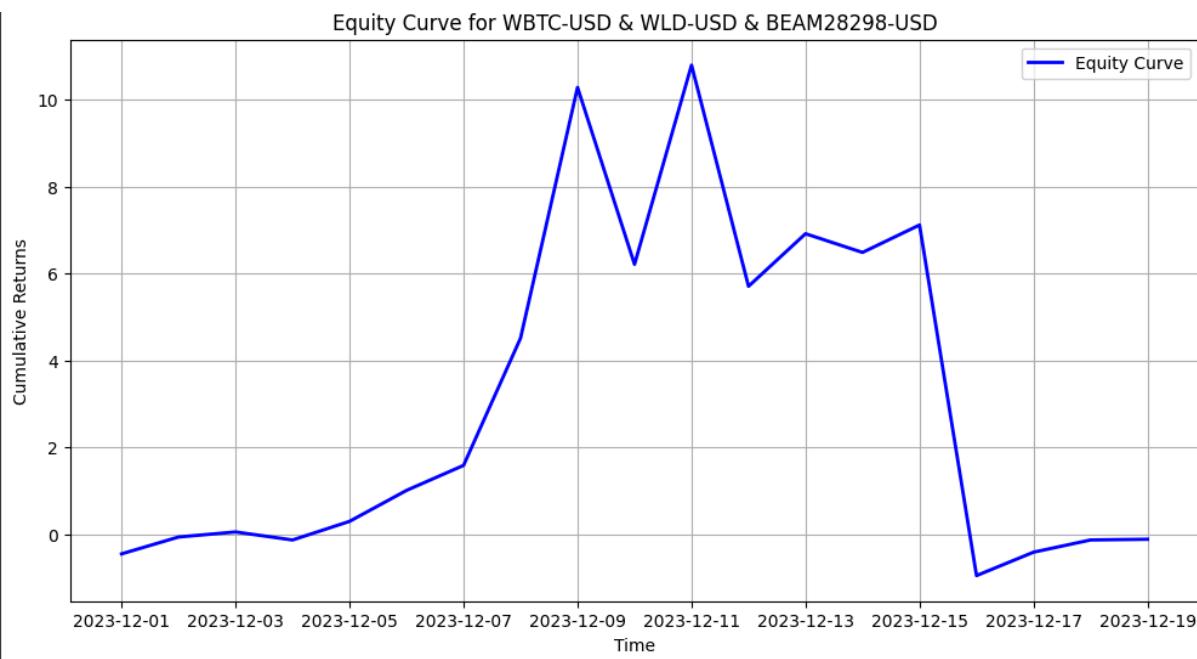
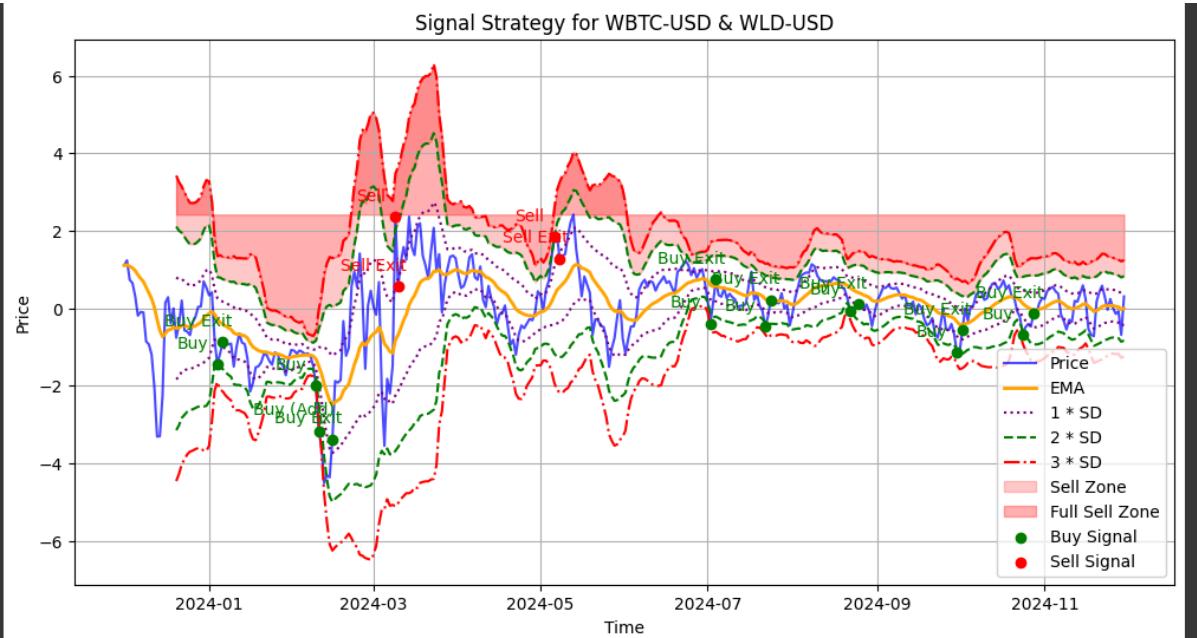
Signal Strategy for WETH-USD & WEETH-USD



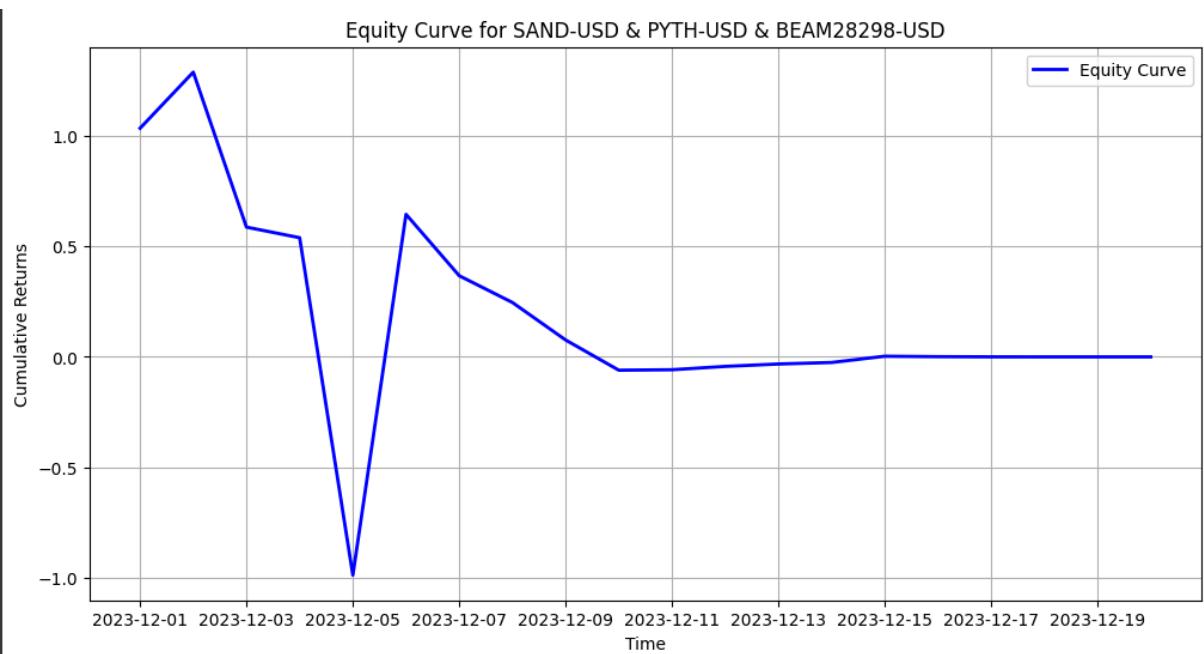
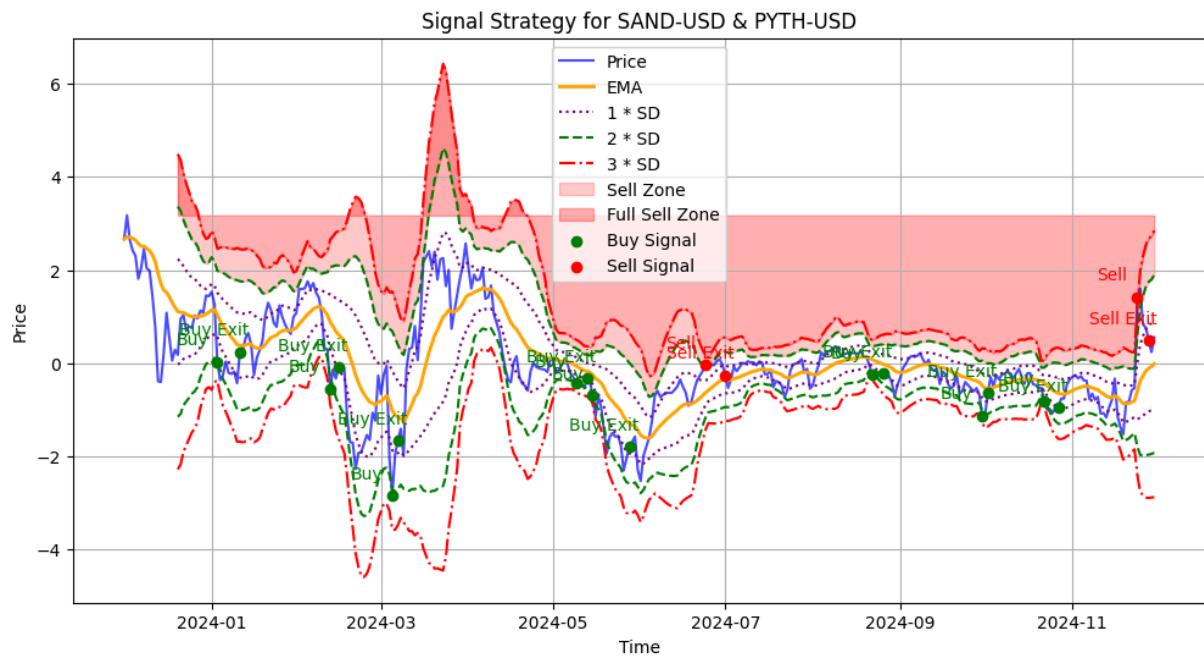
Equity Curve for WETH-USD & WEETH-USD & BONK-USD



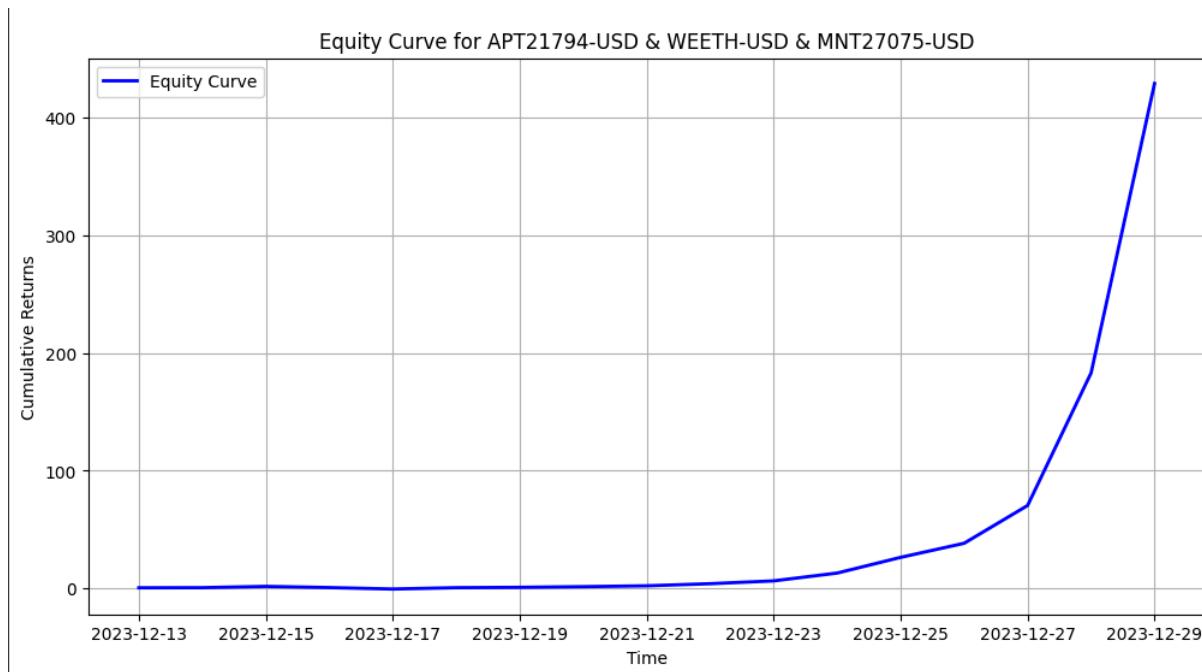
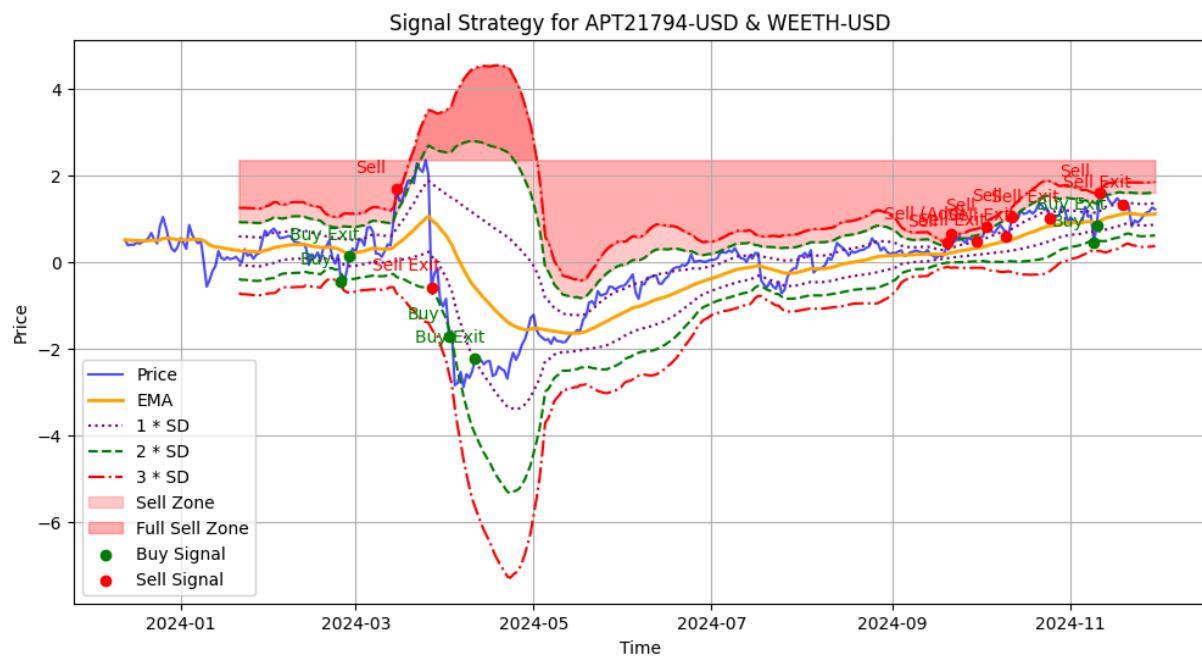
Sharpe Ratio for WETH-USD & WEETH-USD is -0.6252301563420902

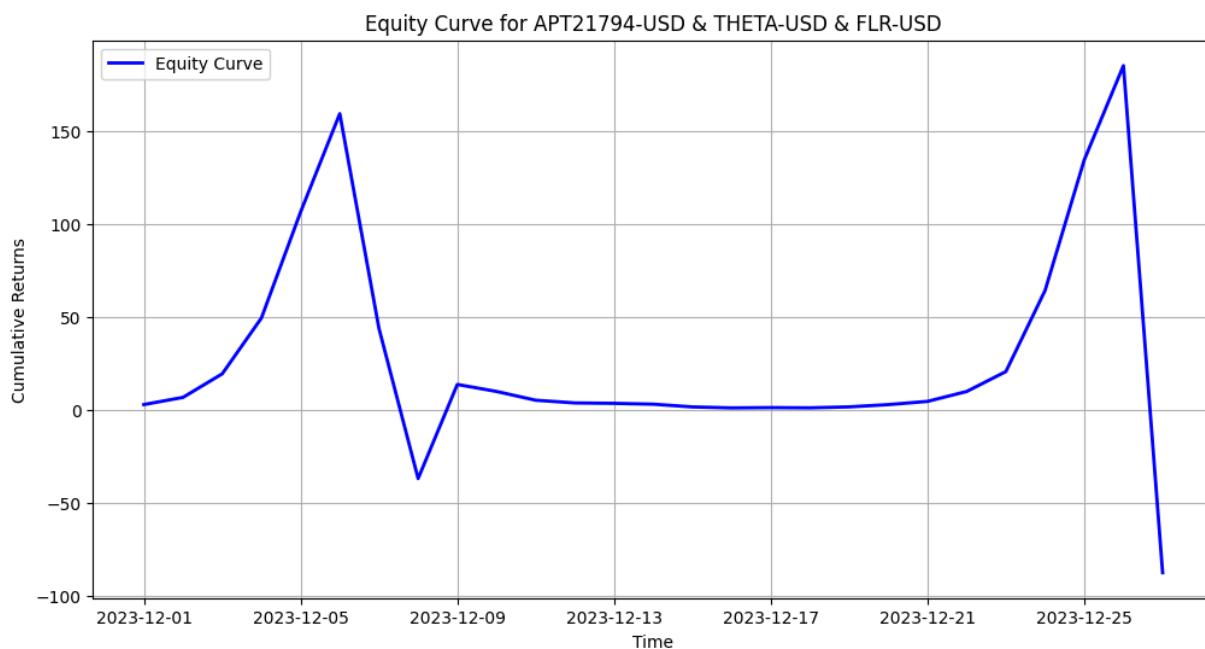
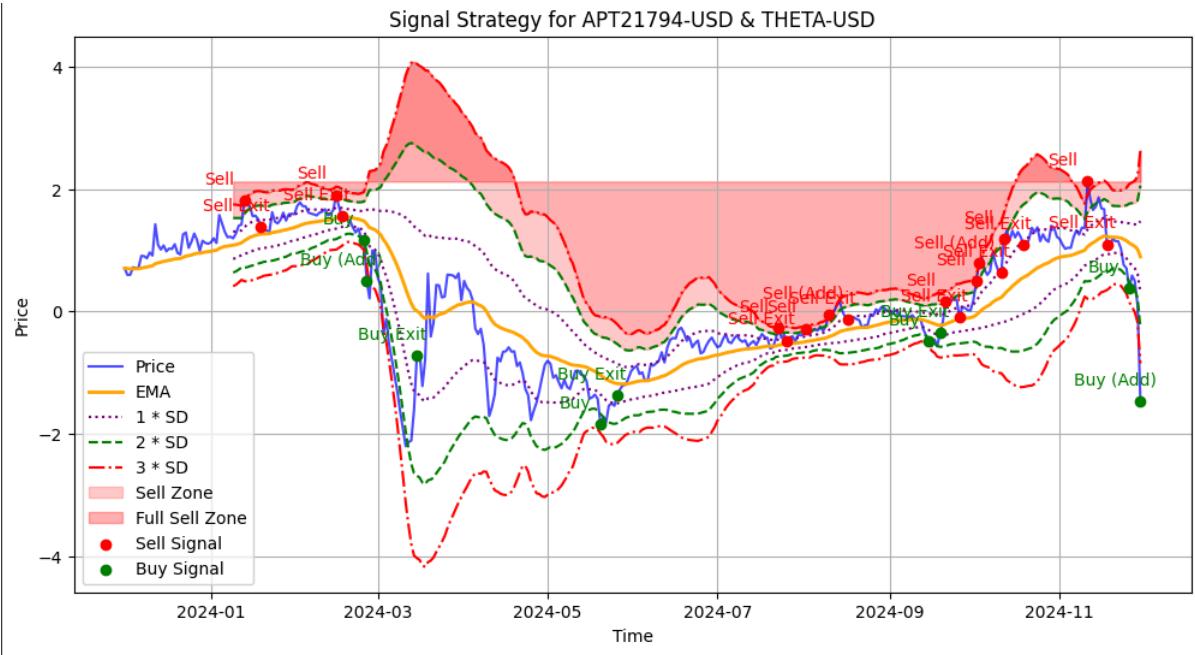


Sharpe Ratio for WBTC-USD & WLD-USD is -4.069950931402535



Sharpe Ratio for SAND-USD & PYTH-USD is -9.221990073226698





Sharpe Ratio for APT21794-USD & THETA-USD is 4.8524734264562905

Print trades

```
for result in results:
    print(f"Pair: {result['Pair']}")
    print(f"Total PnL: {result['Total PnL']:.2f}")
    print("Trades:")
    for trade in result['Trades']:
        print(f" Time: {trade[0]}, Action: {trade[1]}, Price: {trade[2]:.2f}, Position: {trade[3]}")
    print()
```

علامت ضرایب و تاثیر آن ها

درک اهمیت علامت ضرایب در آزمون جوهانسن

هنگامی که آزمون همانباشتگی جوهانسن را بر روی چند سری زمانی مانند قیمت‌های رمزارزها انجام می‌دهید، این آزمون ضرایب همانباشتگی (از بردارهای ویژه) را تولید می‌کند که ترکیب‌های خطی ایستا از متغیرها را تشکیل می‌دهند. این ضرایب مهم هستند زیرا رابطه تعادلی بلندمدت بین متغیرها را تعریف می‌کنند.

مثبت یا منفی بودن ضرایب نشان‌دهنده چیست؟

جهت تاثیر: علامت هر ضریب نشان می‌دهد که هر متغیر چگونه به انحرافات از تعادل کمک می‌کند.

ضریب مثبت: نشان می‌دهد که متغیر به همان جهت با رابطه تعادلی حرکت می‌کند.

ضریب منفی: نشان می‌دهد که متغیر به جهت مخالف رابطه تعادلی حرکت می‌کند.

تعادل در رابطه همانباشتگی: متغیرهایی با علامت مخالف یکدیگر را متعادل می‌کنند تا تعادل بلندمدت حفظ شود.

تفسیر ضرایب همانباشتگی

یک معادله همانباشتگی را در نظر بگیرید که از آزمون جوهانسن به دست آمده است:

$$[\beta_1 \cdot X_1 + \beta_2 \cdot X_2 + \beta_3 \cdot X_3 + \dots] = \text{سری ترکیبی}$$

که در آن:

(β_i) ضرایب همانباشتگی هستند.

(X_i) سری‌های زمانی فردی (مثلًاً قیمت‌های رمزارزها).

ضریب مثبت ($\beta_i > 0$):

تفسیر: متغیر (X_i) به صورت مثبت به سری ترکیبی کمک می‌کند.

مفهوم: افزایش در (X_i) منجر به افزایش در سری ترکیبی می‌شود و در صورت عدم جبران توسط سایر متغیرها، ممکن است آن را از تعادل دور کند.

ضریب منفی ($\beta_i < 0$):

تفسیر: متغیر (X_i) به صورت منفی به سری ترکیبی کمک می‌کند.

مفهوم: افزایش در (X_i) منجر به کاهش در سری ترکیبی می‌شود و به بازگشت آن به تعادل کمک می‌کند.

مثال

فرض کنید ضرایب همانباشتگی نرمال شده زیر را برای یک سه‌گانه از رمزارزها دارید:

ضرایب همانباشتگی:

رمزارز A (BTC-USD): 1.0000

رمزارز B (ETH-USD): -0.8000

رمزارز C (XRP-USD): -0.5000

معادله همانباشتگی:

$$\text{cdot} (\text{\text{BTC-USD}}) - 0.8 \text{cdot} (\text{\text{ETH-USD}}) - 0.5 \text{cdot} 1.0 = \text{\text{XRP-USD}}$$

تفسیر:

دارای BTC-USD ضریب مثبت است:

افزایش در قیمت BTC-USD به افزایش سری ترکیبی کمک می‌کند.

دارای XRP-USD و ETH-USD ضرایب منفی هستند:

افزایش در قیمت‌های XRP-USD یا ETH-USD به کاهش سری ترکیبی کمک می‌کند.

مفاهیم:

عملکرد متقابل: افزایش در BTC-USD با افزایش در XRP-USD و ETH-USD متعادل می‌شود تا تعادل حفظ شود.

حرکات مخالف: اگر قیمت BTC-USD افزایش باید، قیمت‌های XRP-USD و ETH-USD ممکن است نیاز به افزایش داشته باشند (با وجود ضرایب منفی) تا سری ترکیبی ثابت بماند، به دلیل اثر تغیریق در معادله.

بله، ضرایبی که از آزمون جوهانسن بدست می‌آیند در واقع از بردارهای ویژه ماتریس ضرایب سیستم استخراج می‌شوند.

کاربرد: این ضرایب (از بردارهای ویژه) برای ساخت سری ترکیبی ایستا و درک روابط تعادلی بلندمدت بین رمزارزها استفاده می‌شوند.

اهمیت: درک این موضوع به شما کمک می‌کند تا خروجی‌های آزمون جوهانسن را به درستی تفسیر کرده و آن‌ها را بهطور مؤثر در تحلیل و استراتژی‌های معاملاتی خود به کار ببرید.