نوید ابراهیمی ۹۹۵۲۱۰۰۱

در اینجا داده‌های موردنیار برای تحلیل پروژه را در بازه زمانی ۴ ساعته ذخیره می‌کنیم. چون بازه ۴ ساعته در intervalها موجود نیست از این ساختار برای این تبدیل استفاده می‌کنیم.

```
[ ]  def check_cointegration(df):
         result = coint_johansen(df, det_order=0, k_ar_diff=1)
         if np.all(result.lr1 > result.cvt[:, 1]):
             return result
         else:
             return None
```

با استفاده از این تابع سری‌های زمانیی که stationary هستند را مشخص می‌کنیم. اینکار به کمک coint_johnson انجام می‌گیرد. با بررسی اینکه آیا همه آمارهای trace از مقادیر بحرانی خود فراتر می روند (با استفاده از np.all)، این تابع تأیید می کند که یک ترکیب خطی stationary در بین متغیرها وجود دارد.

اگر سری مانا باشد، result را برمیگردانیم تا بتوانیم مقادیر weights را محاسبه کنیم.

Generate Combination of Time Series

```
df = pd.DataFrame(prices)

def generate_stationary_series(df, max_len=4):
    stationary_series = []
    cols = list(df.columns)
    for i in range(2, max_len+1):
        combinations = list(itertools.combinations(cols, i))
        for combination in combinations:
            result = check_cointegration(df[list(combination)])
            if result is not None:
                stationary_series.append(combination)
                normalized_weights = result.evec[:, 0] / result.evec[:, 0][0]
                weights[f'{combination}'] = normalized_weights
                results_of_cointegration[f'{combination}'] = result
    return stationary_series
```

در اینجا سری‌های زمانی جدید را تولید کردیم. به این صورت که با استفاده از itertools سری‌های زمانی را با تعداد معین ترکیب کردیم. سپس چک می‌کنیم که اگر سری مانا باشد، آن را به عنوان سری stationary ذخیره می‌کنیم. همچنین وزن‌های سری جدید و کلا result را برای استفاده در مراحل بعدی ذخیره می‌کنیم. دلیل normal کردن داده‌ها این است که معمولا یک عضو وزن‌ها را ۱ می‌گذارند و بقیه وزن‌ها با توجه به آن مقدار می‌گیرند.

Compute Price of Time Series

```
combination_of_timeseries = {}
for key, weight in weights.items():
  val = key.split(',')
  cleaned_list = [s.replace('(', '').replace(')', '').replace(' ', '').replace("'", "") for s in val]
  time_series = df[cleaned_list].dot(weights[key])
  if f'{cleaned_list}' not in combination_of_timeseries:
    combination_of_timeseries[f'{cleaned_list}'] = {}
  combination_of_timeseries[f'{cleaned_list}']['value'] = time_series
```

در اینجا قیمت سری زمانی را به دست آوردم. به این صورت که وزن‌ها را از قبل ذخیره کرده بودم و در اینجا آن را با قیمت سری‌های زمانی ضرب نقطه‌ای و سپس حاصل را به صورت یک دنباله اعداد که همان نرخ سری زمانی جدید است ذخیره کردم.

Compute P Value

```
[ ]  coint_series_pvalues = {}
     for key, val in combination_of_timeseries.items():
       ad_fuller_result = adfuller(combination_of_timeseries[key]['value'])
       series_pvalue = ad_fuller_result[1]
       combination_of_timeseries[key]['p_value'] = series_pvalue
```

در اینجا نیز مقدار p value را با استفاده از adfuller به دست آورده و ذخیره کردم.

Selection

```
[ ]  sorted_combination_of_timeseries = {k: v for k, v in sorted(combination_of_timeseries.items(), key=lambda item: item[1]['p_value'])}
     items = list(sorted_combination_of_timeseries.items())
     sliced_items = items[:10]
     sorted_combination_of_timeseries = dict(sliced_items)
     for key, val in sorted_combination_of_timeseries.items():
       print(sorted_combination_of_timeseries[key]['p_value'])
```

در اینجا سری زمانی با کمترین مقدارp value را انتخاب کردم و سایر محاسبات را فقط بر روی همین مقادیر انجام دادم.

Compute Hurst Exponents

```
[ ]  hurst_exponents = {}
     for key, val in sorted_combination_of_timeseries.items():
       H, c, data = compute_Hc(sorted_combination_of_timeseries[key]['value'])
       sorted_combination_of_timeseries[key]['hurst_exponents'] = H
```

در اینجا نیز Hurst Exponent را با استفاده از تابع از پیش آماده compute_Hc محاسبه و سپس ذخیره کردم.

Compute Half Life Time

```
[ ]  def compute_half_life(series):
         price = pd.Series(series)
         lagged_price = price.shift(1).fillna(method="bfill")
         delta = price - lagged_price
         beta = np.polyfit(lagged_price, delta, 1)[0]
         half_life = -np.log(2) / beta
         return half_life
```

تابع ابتدا سری ورودی را به یک pandas.Series تبدیل می کند. سپس، سری lagged price را با جابجایی سری قیمت به اندازه یک دوره و پر کردن مقادیر Nan محاسبه می کند. این سری قیمت با تاخیر در متغیر lagged_price ذخیره می شود.

سپس تابع delta را برای هر دوره با کم کردن lagged price از قیمت فعلی محاسبه می کند. این در متغیر delta ذخیره می شود.

در مرحله بعد، تابع یک مدل رگرسیون خطی از شکل delta = beta * lagged_price + epsilon را با سری lagged_price و delta با استفاده از تابع polyfit مطابقت می دهد. ضریب بتا که نشان دهنده شیب خط رگرسیون است در متغیر بتا ذخیره می شود.

در نهایت، تابع نیمه عمر سری زمانی را با استفاده از فرمول -np.log(2) / beta محاسبه می کند. این فرمول مبتنی بر این واقعیت است که نیمه عمر زمانی است که طول می کشد تا مقدار سری به نصف از مقدار اولیه آن کاهش یابد. مدل فروپاشی نمایی فرض می کند که نرخ کاهش متناسب با مقدار فعلی است که توسط ضریب بتا گرفته می شود.

	['XRP-USD', 'USDC-USD', 'DAI-USD', 'LEO-USD']	['XRP-USD', 'USDC-USD', 'DAI-USD', 'LTC-USD']	['USDC-USD', 'DAI-USD', 'LTC-USD', 'ETC-USD']	['ETH-USD', 'USDC-USD', 'DAI-USD', 'CRO-USD']	['USDC-USD', 'STETH-USD', 'DAI-USD', 'CRO-USD']	['USDC-USD', 'DAI-USD', 'LEO-USD', 'XLM-USD']	['USDC-USD', 'DAI-USD', 'OKB-USD', 'XMR-USD']	['USDC-USD', 'AVAX-USD', 'DAI-USD', 'NEAR-USD']	['USDC-USD', 'DAI-USD', 'OKB-USD', 'STX4847-USD']	['XRP-USD', 'USDC-USD', 'DAI-USD', 'XLM-USD']
value	Datetime 2022-11-01 00:00:00+00:00 105.7272...	Datetime 2022-11-01 00:00:00+00:00 168.9372...	Datetime 2022-11-01 00:00:00+00:00 -0.127528...	Datetime 2022-11-01 00:00:00+00:00 303450.5...	Datetime 2022-11-01 00:00:00+00:00 -0.126828...	Datetime 2022-11-01 00:00:00+00:00 -0.125378...	Datetime 2022-11-01 00:00:00+00:00 -0.125610...	Datetime 2022-11-01 00:00:00+00:00 -0.127816...	Datetime 2022-11-01 00:00:00+00:00 -0.128788...	Datetime 2022-11-01 00:00:00+00:00 82.10259...
p_value	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
hurst_exponents	0.490125	0.489499	0.490151	0.490556	0.49042	0.490527	0.490698	0.489763	0.489541	0.490164
half_lives	0.970305	0.968121	0.968485	0.973068	0.971763	0.97612	0.975788	0.972203	0.969534	0.975963

نتایج نهایی فرایند (مقادیر p_value صفر در واقع e^{-28} هستند که در جدول 0 نوشته شده‌اند)

همانطور که مشخص است چون hurst exponent از ۰,۵ کوچک‌تر است، پس سری ماناست.