Crypto NLNTest

[1]:
```
# Silenting the warnings
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

Imports

[2]:
```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from datetime import datetime
import time
import math
import nlntest
```

Reading Data

We read the historical data of the 10 biggest crypto currencies using yahoo finance api.

[3]:
```
# Bitcoin, Ethereum, Tether, BNB, Dai, XRP, Dogecoin, Cardano, Polygon, Polkadot

crypto_list=    ['BTC', 'ETH', 'USDT', 'BNB', 'DAI',
                 'XRP', 'DOGE', 'ADA', 'MATIC', 'DOT']
currency = 'USD'
crypto_list = [f'{x}-{currency}' for x in crypto_list]

df_raw = yf.download(crypto_list, group_by = 'ticker', period = '100y', interval='1d')
```

```
[**********100%**********]  10 of 10 completed
```

Helper Functions

The `nlntstuniv` function in `nlntest` package returns 4 tests. We take vote of this 4 tests to decied weather the series is nonlinear or not.

[6]:
```
def get_is_nln(series) -> bool:
    nln_res = nlntest.nlntstuniv(np.array(series))
    p_val_tresh = 0.05
    is_nln = list(filter(lambda x: x<=p_val_tresh, nln_res)).__len__() >= 2
    return is_nln
```

This functions returns ta window of a dataframes by getting the window_size and offset.

[7]:
```
def rolling(df, window, offset) -> pd.DataFrame:
    return df[offset:offset+window]
```

Tests

We itterate over all 10 price series and test for nonlinearity for the return series in all possible windows_size and offsets.

[8]:
```
t = time.time()

results = []

for symbol in df_raw.columns.levels[0]:

    df_price = df_raw.loc[:, symbol].loc[:, 'Close']
    df_price = df_price.dropna()

    windows = [1, 2, 3, 5, math.floor(df_price.shape[0]/365)]
    step = 1

    for window in windows:
        num_of_ittr = math.floor(df_price.shape[0]/365) - window + 1
        num_of_ittr = max(num_of_ittr, 1)

        for i in range(num_of_ittr):
            prices_in_window = rolling(df_price, 365*window, i*(step)*365)
            ret_series = np.log(prices_in_window).diff()
            is_nln = get_is_nln(ret_series.dropna())
            results.append({
                'symbol': symbol.split('-')[0],
                'from': prices_in_window.index[0].date(),
                'to': prices_in_window.index[-1].date(),
                'window_size': window,
                'offsetY': i*step,
                'is_nln': is_nln,
            })

elapsed = time.time() - t
print('elapsed time:', elapsed)
```

```
elapsed time: 1.3930859565734863
```

Visualizing

[9]:
```
df_res = pd.DataFrame(results)
```

This function recievs a dataframe and save it in a figure like a table.

[10]:
```
def render_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
                     header_color='#40466e', row_colors=['#f1f1f2', 'w'],␣
 ↪edge_color='w',
                     bbox=[0, 0, 1, 1], header_columns=0,
                     ax=None, **kwargs):
    if ax is None:
        size = (np.array(data.shape[::-1]) + np.array([0, 1])) * np.array([col_width,␣
 ↪row_height])
```

```
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')
    mpl_table = ax.table(cellText=data.values, bbox=bbox, colLabels=data.columns,␣
↪**kwargs)
    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    for k, cell in mpl_table._cells.items():
        cell.set_edgecolor(edge_color)
        if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(weight='bold', color='w')
            cell.set_facecolor(header_color)
        else:
            cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
    return ax.get_figure(), ax
```

We plot the results. The output is in `./ouputs/` directory.

[12]:
```
plt.ioff()

for symbol in df_res['symbol'].unique():


    df_plt = df_res[df_res['symbol']==symbol].drop_duplicates().reset_index()
    xtickes = []
    positions = []
    fig = plt.figure()
    for i, row in df_plt.iterrows():
        frm = datetime.strptime(str(row['from']), '%Y-%m-%d')
        to = datetime.strptime(str(row['to']), '%Y-%m-%d')
        xtickes.append(f"{row['window_size']}w+{row['offsetY']}")
        positions.append(i+row['window_size'])
        x, y = [frm, to], [i+row['window_size'], i+row['window_size']]
        plt.plot(x, y, c='crimson' if row['is_nln'] else 'darkgreen', linewidth=2);
    red_patch = mpatches.Patch(color='crimson', label='Non-Linear')
    blue_patch = mpatches.Patch(color='darkgreen', label='Linear')
    plt.legend(handles=[red_patch, blue_patch])
    plt.yticks(positions, xtickes)
    for label in fig.axes[0].get_xticklabels():
        label.set_ha("right")
        label.set_rotation(30)
    plt.title(symbol);

    plt.savefig(f'./output/{symbol}_plot.png');
    plt.close()
    fig,ax = render_mpl_table(df_plt.drop('index', axis=1), header_columns=0,␣
↪col_width=2.0);
    fig.savefig(f"./output/{symbol}_table.png");
    fig, ax = None, None
```