Causality of Economic Uncertainty on Bitcoin Prices

School of Information Technology & School of Business
Monash University Malaysia

For Granger
https://towardsdatascience.com/granger-causality-and-vector-auto-regressive-model-for-time-series-forecasting-3226a64889a6 (https://towardsdatascience.com/granger-causality-and-vector-auto-regressive-model-for-time-series-forecasting-3226a64889a6)

Quick explanation of Granger
https://link.springer.com/chapter/10.1007/978-3-319-58895-7_27 (https://link.springer.com/chapter/10.1007/978-3-319-58895-7_27)

Steps

- Read dataset (BTC, China Economic, US Economic)
- Have new dataset Global EPU, Singapore EPU, UK EPU and US EPU (https://www.policyuncertainty.com/index.html (https://www.policyuncertainty.com/index.html))
- Wrangle and merge dataset
- Conduct Granger Test (statsmodels)

In [55]:
```
import numpy as np
import pandas as pd
#import matplotlib.pyplot as plt
```

Read datasets

In [56]:
```
globalDf = pd.read_csv("data/Global_Policy_Uncertainty_Data-Since1997.csv", encoding="latin-1")
btcDf = pd.read_csv("data/BTC-Monthly-Since2014.csv")
chinaDf = pd.read_csv("data/China-TPU-Monthly-Since2000.csv", encoding="latin-1") # due to some 0x9x characters
usDf = pd.read_csv("data/US_Policy_Index-Monthly-Since1985.csv")
```

Wrangle Global EPU dataset

In [57]:
```
globalDf.head()
```

Out[57]:

	Year	Month	GEPU_current	GEPU_ppp
0	1997	1.0	74.250916	76.827372
1	1997	2.0	77.455858	79.145625
2	1997	3.0	66.761655	64.889262
3	1997	4.0	71.868261	73.164048
4	1997	5.0	69.901362	70.204210

In [58]:
```
globalDf.dtypes
```

Out[58]:
```
Year             object
Month           float64
GEPU_current    float64
GEPU_ppp        float64
dtype: object
```

In [59]:
```
globalDf = globalDf.dropna()
```

In [60]:
```
globalDf.tail()
```

Out[60]:

	Year	Month	GEPU_current	GEPU_ppp
280	2020	5.0	412.826586	423.968339
281	2020	6.0	313.741755	333.653920
282	2020	7.0	337.384424	340.895162
283	2020	8.0	285.883907	306.856490
284	2020	9.0	286.915497	298.280919

In [61]:
```
globalDf['Year'] = globalDf['Year'].astype('int')
```

In [62]:
```
globalDf.drop(globalDf[globalDf['Year'] < 2014].index, inplace=True)
```

In [63]:
```
globalDf['Year'] = globalDf['Year'].astype('int')
```

In [64]:
```
globalDf['Month'] = globalDf['Month'].astype('int')
```

In [65]:
```
globalDf.head()
```

Out[65]:

	Year	Month	GEPU_current	GEPU_ppp
204	2014	1	107.324807	108.767217
205	2014	2	94.643427	92.150604
206	2014	3	109.436900	107.244238
207	2014	4	97.775491	101.434120
208	2014	5	100.257487	102.025873

In [66]:
```
len(globalDf)
```

Out[66]: 81

Wrangle BTC dataset

In [67]:
```
btcDf.head()
```

Out[67]:

	Date (based on 1st of Month)	Open	High	Low	Close	Volume
0	Sep-2014	465.864014	468.174011	372.239990	386.944000	4.108810e+08
1	Oct-2014	387.427002	411.697998	289.295990	338.321014	9.029944e+08
2	Nov-2014	338.649994	457.092987	320.626007	378.046997	6.597334e+08
3	Dec-2014	378.248993	384.037994	304.231995	320.192993	5.531023e+08
4	Jan-2015	320.434998	320.434998	171.509995	217.464005	1.098812e+09

In [68]:
```
btcDf.dtypes
```

Out[68]:
```
Date (based on 1st of Month)     object
Open                            float64
High                            float64
Low                             float64
Close                           float64
Volume                          float64
dtype: object
```

In [69]:
```
btcDf['Year'] = pd.DatetimeIndex(btcDf['Date (based on 1st of Month)']).year
btcDf['Month'] = pd.DatetimeIndex(btcDf['Date (based on 1st of Month)']).month
btcDf.head()
```

Out[69]:

	Date (based on 1st of Month)	Open	High	Low	Close	Volume	Year	Month
0	Sep-2014	465.864014	468.174011	372.239990	386.944000	4.108810e+08	2014	9
1	Oct-2014	387.427002	411.697998	289.295990	338.321014	9.029944e+08	2014	10
2	Nov-2014	338.649994	457.092987	320.626007	378.046997	6.597334e+08	2014	11
3	Dec-2014	378.248993	384.037994	304.231995	320.192993	5.531023e+08	2014	12
4	Jan-2015	320.434998	320.434998	171.509995	217.464005	1.098812e+09	2015	1

In [70]:
```
# Estatics only, I just wanted the year and month in front
cols = btcDf.columns.tolist()
cols = cols[-1:] + cols[:-1]; cols = cols[-1:] + cols[:-1]
btcDf = btcDf[cols]
# Delete the not needed column
del btcDf['Date (based on 1st of Month)']
```

In [71]:
```
btcDf.reset_index()
btcDf.head()
```

Out[71]:

	Year	Month	Open	High	Low	Close	Volume
0	2014	9	465.864014	468.174011	372.239990	386.944000	4.108810e+08
1	2014	10	387.427002	411.697998	289.295990	338.321014	9.029944e+08
2	2014	11	338.649994	457.092987	320.626007	378.046997	6.597334e+08
3	2014	12	378.248993	384.037994	304.231995	320.192993	5.531023e+08
4	2015	1	320.434998	320.434998	171.509995	217.464005	1.098812e+09

In [72]:
```
btcDf.tail()
```

Out[72]:

	Year	Month	Open	High	Low	Close	Volume
67	2020	4	6437.319336	9440.650391	6202.373535	8658.553711	1.156130e+12
68	2020	5	8672.782227	9996.743164	8374.323242	9461.058594	1.286370e+12
69	2020	6	9463.605469	10199.565430	8975.525391	9137.993164	6.509130e+11
70	2020	7	9145.985352	11415.864260	8977.015625	11323.466800	5.458130e+11
71	2020	8	11322.570310	12034.144530	11012.415040	11675.739260	1.797630e+11

In [73]:
```
len(btcDf)
```

Out[73]: 72

Wrangle China dataset

In [74]:
```
chinaDf.head()
```

Out[74]:

	year	month	TPU	Unnamed: 3	Unnamed: 4	Source: "Economic Policy Uncertainty in China Since 1949: The View from Mainland Newspapers,□ by Steven J. Davis, Dingqian Liu and Xuguang S. Sheng, 2019.
0	2000	1	38.2	NaN	NaN	These data can be used freely with attribution...
1	2000	2	14.7	NaN	NaN	NaN
2	2000	3	8.9	NaN	NaN	NaN
3	2000	4	8.9	NaN	NaN	NaN
4	2000	5	0.0	NaN	NaN	NaN

In [75]:
```
# Reformat by dropping last 3 columns (or rather, just take the first 3 columns)
chinaDf = chinaDf.loc[:,['year','month','TPU']]
```

In [76]:
```
chinaDf.columns = ['Year','Month','TPU']
```

In [77]:
```
chinaDf.drop(chinaDf[chinaDf['Year'] < 2014].index, inplace=True)
```

In [78]:
```
chinaDf.reset_index(drop = True, inplace = True)
chinaDf.head()
```

Out[78]:

	Year	Month	TPU
0	2014	1	39.0
1	2014	2	24.0
2	2014	3	99.9
3	2014	4	67.9
4	2014	5	15.7

In [79]:
```
len(chinaDf)
```

Out[79]: 79

In [80]:
```
chinaDf.tail()
```

Out[80]:

	Year	Month	TPU
74	2020	3	105.9
75	2020	4	193.1
76	2020	5	547.3
77	2020	6	473.9
78	2020	7	340.8

Wrangle US dataset

In [81]:
```
usDf.head()
```

Out[81]:

	Year	Month	News_Based_Policy_Uncert_Index	FedStateLocal_Ex_disagreement	CPI_disagreement	Tax_expiration
0	1985	1.0	103.748802	94.195557	204.033661	13.494806
1	1985	2.0	78.313202	131.445221	136.022430	13.494806
2	1985	3.0	100.761482	131.683533	136.022430	13.494806
3	1985	4.0	84.778870	131.495529	136.022430	13.494806
4	1985	5.0	98.053658	139.016907	170.028061	13.494806

In [82]:
```
usDf.dtypes
```

Out[82]:
```
Year                              object
Month                            float64
News_Based_Policy_Uncert_Index   float64
FedStateLocal_Ex_disagreement    float64
CPI_disagreement                 float64
Tax_expiration                   float64
dtype: object
```

In [83]:
```
usDf['Year'] = pd.to_numeric(usDf['Year'], errors='coerce')
```

In [84]:
```
usDf.drop(usDf[usDf['Year'] < 2014].index, inplace=True)
```

In [85]:
```
usDf = usDf.dropna()
```

In [86]:
```
usDf['Year'] = usDf['Year'].astype('int'); usDf['Month'] = usDf['Month'].astype('int')
```

In [87]:
```
usDf.dtypes
```

Out[87]:
```
Year                              int32
Month                             int32
News_Based_Policy_Uncert_Index  float64
FedStateLocal_Ex_disagreement   float64
CPI_disagreement                float64
Tax_expiration                  float64
dtype: object
```

In [88]:
```
usDf.reset_index(drop = True, inplace = True)
usDf.head()
```

Out[88]:

	Year	Month	News_Based_Policy_Uncert_Index	FedStateLocal_Ex_disagreement	CPI_disagreement	Tax_expiration
0	2014	1	107.705139	83.794785	71.942268	18.917648
1	2014	2	93.369286	87.217278	85.585320	18.917648
2	2014	3	101.018730	87.261520	85.585320	18.917648
3	2014	4	96.993431	87.300034	85.585320	18.917648
4	2014	5	102.015045	91.901459	68.011215	18.917648

In [89]:
```
len(usDf)
```

Out[89]: 79

In [90]:
```
usDf.tail()
```

Out[90]:

	Year	Month	News_Based_Policy_Uncert_Index	FedStateLocal_Ex_disagreement	CPI_disagreement	Tax_expiratio
74	2020	3	425.779205	69.506531	72.472755	282.28414
75	2020	4	400.944733	69.506531	72.472755	282.28414
76	2020	5	503.963348	171.697784	98.969925	282.28414
77	2020	6	300.400940	171.697784	98.969925	282.28414
78	2020	7	409.611176	171.697784	98.969925	282.28414

Merge Data

As the BTC dataset only starts from September 2014 and the China and US dataset is only up to July 2020, we will need to filter them before combining all of them together in one dataframe.

In [91]:
```
globalDf = globalDf[ ~((globalDf['Year'] == 2020) & (globalDf['Month'] > 7)) ]
globalDf = globalDf[ ~((globalDf['Year'] == 2014) & (globalDf['Month'] < 9)) ]
```

In [92]:
```
btcDf = btcDf[ ~((btcDf['Year'] == 2020) & (btcDf['Month'] > 7)) ]
chinaDf = chinaDf[ ~((chinaDf['Year'] == 2014) & (chinaDf['Month'] < 9)) ]
usDf = usDf[ ~((usDf['Year'] == 2014) & (usDf['Month'] < 9)) ]
```

In [93]:
```
if (len(btcDf) == len(chinaDf) == len(usDf)):
    print("Data frames ready to merge")
```

```
Data frames ready to merge
```

In [94]:
```
allDf = globalDf.merge(btcDf.merge(chinaDf).merge(usDf))
```

In [95]:
```
allDf.head()
```

Out[95]:

	Year	Month	GEPU_current	GEPU_ppp	Open	High	Low	Close	Volume	TPU	I
0	2014	9	119.004523	123.848461	465.864014	468.174011	372.239990	386.944000	4.108810e+08	66.0	
1	2014	10	115.001499	117.841802	387.427002	411.697998	289.295990	338.321014	9.029944e+08	47.9	
2	2014	11	109.575411	115.329683	338.649994	457.092987	320.626007	378.046997	6.597334e+08	108.4	
3	2014	12	107.930762	109.408125	378.248993	384.037994	304.231995	320.192993	5.531023e+08	18.7	
4	2015	1	131.733360	138.663842	320.434998	320.434998	171.509995	217.464005	1.098812e+09	22.8	

In [96]:
```
# Let's save this for a bit
allDf.to_csv('data/allData.csv')
```

In [97]:
```
# No NaN values, if there are, best practise is to pad it
```

We use only the BTC closing price.

In [98]:
```
gDf = allDf.loc[:,['GEPU_current','Close', 'TPU', 'News_Based_Policy_Uncert_Index', 'FedState
Local_Ex_disagreement', 'CPI_disagreement', 'Tax_expiration']]
gDf.columns = ['GlobalEPU', 'BTC','TPU','News', 'FedState', 'CPI','Tax']
gDf.head()
```

Out[98]:

	GlobalEPU	BTC	TPU	News	FedState	CPI	Tax
0	119.004523	386.944000	66.0	86.216591	90.738510	81.613464	18.917648
1	115.001499	338.321014	47.9	113.334579	90.757492	81.613464	18.917648
2	109.575411	378.046997	108.4	93.185951	91.726883	81.613464	18.917648
3	107.930762	320.192993	18.7	87.415321	91.753418	81.613464	18.917648
4	131.733360	217.464005	22.8	120.544228	91.764519	81.613464	620.767761

Do we need to know the distribution, kurtosis and skewness of the data? For this moment, I think we don't need to do so.

In [99]:
```
from scipy import stats

stat, p = stats.normaltest(gDf.BTC)
print("Statistics = %.3f, p=%.3f" % (stat,p))

alpha = 0.05
if p > alpha:
    print('Data looks Gaussian (fail to reject null hypothesis)')
else:
    print('Data looks non-Gaussian (reject null hypothesis)')
```

```
Statistics = 19.829, p=0.000
Data looks non-Gaussian (reject null hypothesis)
```

In [100]:
```
# Kurtosis and Skewness
```

In [101]:
```
# To be coded, not important for the Granger test, just for us to get a feel of the data.
# Anyway, we can simply plot it and have a look.
```

Granger's Causality Test

Data Preparation

Transform

We transform the data to achieve stationary data by conductung a differencing. The code below does differencing (order 1) with the first element padded with a zero difference.

In [102]:
```
gDf['GlobalEPU'] = np.diff(gDf['GlobalEPU'], prepend=gDf['GlobalEPU'][0])
gDf['BTC'] = np.diff(gDf['BTC'], prepend=gDf['BTC'][0])
gDf['TPU'] = np.diff(gDf['TPU'], prepend=gDf['TPU'][0])
gDf['News'] = np.diff(gDf['News'], prepend=gDf['News'][0])
gDf['FedState'] = np.diff(gDf['FedState'], prepend=gDf['FedState'][0])
gDf['CPI'] = np.diff(gDf['CPI'], prepend=gDf['CPI'][0])
gDf['Tax'] = np.diff(gDf['Tax'], prepend=gDf['Tax'][0])
```

Stationary Check

Granger assumes that the data is stationary and hence we need to ensure, by checking, that the time series is stationary after differencing. Occassinally, after differencing, the data is not stationary and different order of differencing may be required.

The Augmented Dickey-Fuller test will need to reject the null hypothesis of the existance of a unit root (a feature that will cause problems in statistical inference). A negative number result is expected and the more negative it is, the better it is.

In [103]:

```python
from statsmodels.tsa.stattools import adfuller

def augmented_dickey_fuller_statistics(time_series):
    result = adfuller(time_series.values)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

print('Augmented Dickey-Fuller Test: Global EPU Time Series')
augmented_dickey_fuller_statistics(gDf['GlobalEPU'])

print('Augmented Dickey-Fuller Test: BTC Price Time Series')
augmented_dickey_fuller_statistics(gDf['BTC'])

print('Augmented Dickey-Fuller Test: China TPU Time Series')
augmented_dickey_fuller_statistics(gDf['TPU'])

print('Augmented Dickey-Fuller Test: US News Index Time Series')
augmented_dickey_fuller_statistics(gDf['News'])

print('Augmented Dickey-Fuller Test: US Federal and State Disagreement Index Time Series')
augmented_dickey_fuller_statistics(gDf['FedState'])

print('Augmented Dickey-Fuller Test: US CPI Time Series')
augmented_dickey_fuller_statistics(gDf['CPI'])

print('Augmented Dickey-Fuller Test: US Tax Time Series')
augmented_dickey_fuller_statistics(gDf['Tax'])
```

```
Augmented Dickey-Fuller Test: Global EPU Time Series
ADF Statistic: -6.507074
p-value: 0.000000
Critical Values:
        1%: -3.532
        5%: -2.906
        10%: -2.590
Augmented Dickey-Fuller Test: BTC Price Time Series
ADF Statistic: -8.247094
p-value: 0.000000
Critical Values:
        1%: -3.527
        5%: -2.904
        10%: -2.589
Augmented Dickey-Fuller Test: China TPU Time Series
ADF Statistic: -9.652311
p-value: 0.000000
Critical Values:
        1%: -3.529
        5%: -2.904
        10%: -2.590
Augmented Dickey-Fuller Test: US News Index Time Series
ADF Statistic: -6.769517
p-value: 0.000000
Critical Values:
        1%: -3.532
        5%: -2.906
        10%: -2.590
Augmented Dickey-Fuller Test: US Federal and State Disagreement Index Time Series
ADF Statistic: -8.299840
p-value: 0.000000
Critical Values:
        1%: -3.527
        5%: -2.904
        10%: -2.589
Augmented Dickey-Fuller Test: US CPI Time Series
ADF Statistic: -8.428358
p-value: 0.000000
Critical Values:
        1%: -3.530
        5%: -2.905
        10%: -2.590
Augmented Dickey-Fuller Test: US Tax Time Series
ADF Statistic: -4.657849
p-value: 0.000101
Critical Values:
        1%: -3.546
        5%: -2.912
        10%: -2.594
```

It looks like there isn't an existance of any unit roots in any of the time series above.

Granger Test Parameters Setting

In [104]:
```
from statsmodels.tsa.stattools import grangercausalitytests
```

The null hypothesis for all four test is that the coefficients corresponding to past values of the second time series are zero.

- “params_ftest”, “ssr_ftest” are based on F distribution
- “ssr_chi2test”, “lrtest” are based on chi-square distribution

In [105]:
```
test = 'ssr-chi2test'
```

We use a maximum lag of 12, as we have 12 months in a year. Possibly seasonal effects and also economic indicators are generally "stationary" on a yearly basis. May want to try using different lags, such as 1, 3, 6 (representing months in the data sets that we are using).

In [106]:
```
maxlag=12
```

Function to create a matrix to show the null hypothesis

In [107]:
```
def grangers_causality_matrix(X, variables, test = 'ssr_chi2test', verbose=False):
    dataset = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, ind
ex=variables)
    for c in dataset.columns:
        for r in dataset.index:
            test_result = grangercausalitytests(X[[r,c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            if verbose:
                print(f'Y = {r}, X = {c}, P Values = {p_values}')
            min_p_value = np.min(p_values)
            dataset.loc[r,c] = min_p_value
    dataset.columns = [var + '_x' for var in variables]
    dataset.index = [var + '_y' for var in variables]
    return dataset
```

Run the test

In [108]:
```
grangers_causality_matrix(gDf, variables = gDf.columns)
```

Out[108]:

	GlobalEPU_x	BTC_x	TPU_x	News_x	FedState_x	CPI_x	Tax_x
GlobalEPU_y	1.0000	0.0461	0.0094	0.0435	0.0133	0.0145	0.0004
BTC_y	0.0034	1.0000	0.0000	0.5195	0.0019	0.0000	0.8999
TPU_y	0.2463	0.0000	1.0000	0.3040	0.0396	0.0669	0.6760
News_y	0.2293	0.2645	0.0000	1.0000	0.0094	0.0952	0.0048
FedState_y	0.0102	0.0059	0.0000	0.0000	1.0000	0.0000	0.6959
CPI_y	0.0005	0.0071	0.0213	0.0086	0.0000	1.0000	0.0050
Tax_y	0.3014	0.8924	0.8929	0.4728	0.0016	0.0397	1.0000

H_0 is "Y does not Granger-cause X" --> 0 means reject, means Y causes X

Hence BTC price is impacted by Global EPU (0.0461), China TPU (0.000), US FedState disagreement (0.0059) and US CPI (0.0071)

In [109]:
```
#Ian: I am still unsure of the statement I made above.
```