In [58]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram , linkage, fcluster
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import os
import subprocess
import networkx as nx
from scipy.cluster import hierarchy
from scipy.cluster.hierarchy import dendrogram
```

In [59]:

```python
# df=pd.read_csv("Stocksforpython.csv")
# df=df.dropna()
df = pd.read_csv('Stocksforpython.csv', header = 0, index_col = 0)
print('shape of data'),df.shape
df
```

shape of data

Out[59]:

	AMD	BABA	NVDA	ROKU	U
Date					
21/06/2021	82.589996	211.059998	184.272507	382.730011	107.889999
22/06/2021	83.580002	211.320007	188.867493	403.500000	110.260002
23/06/2021	83.820000	214.860001	190.572495	421.700012	112.959999
24/06/2021	86.099998	218.380005	192.054993	423.579987	113.760002
25/06/2021	85.620003	228.500000	190.309998	430.940002	110.150002
...	...	...	...	...	...
13/06/2022	86.989998	98.519997	156.470001	73.250000	34.349998
14/06/2022	86.989998	105.230003	158.360001	74.290001	33.759998
15/06/2022	89.300003	108.029999	165.270004	83.000000	36.169998
16/06/2022	82.050003	101.449997	156.009995	78.580002	32.849998
17/06/2022	81.570000	102.239998	158.800003	82.419998	35.480000

252 rows × 5 columns

In [60]:

```python
returns = pd.DataFrame()
for i in df:
    returns[i] = np.log(df[i]).diff()
returns = returns[1:]
```

In [61]:

```python
returns.corr()
```

Out[61]:

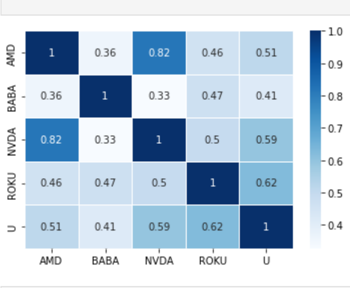
	AMD	BABA	NVDA	ROKU	U
AMD	1.000000	0.356762	0.818748	0.461033	0.513069
BABA	0.356762	1.000000	0.327675	0.470825	0.414495
NVDA	0.818748	0.327675	1.000000	0.497427	0.594451
ROKU	0.461033	0.470825	0.497427	1.000000	0.624803
U	0.513069	0.414495	0.594451	0.624803	1.000000

In [62]:

```python
corrMatrix = returns.corr()
```

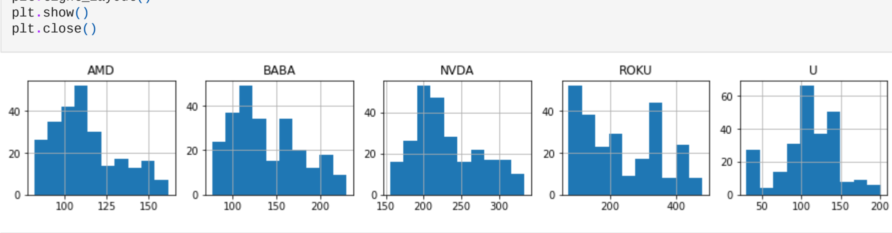
In [63]:

```python
# fig, ax = plt.subplots(figsize=(10,10))
# sn.heatmap(corrMatrix, annot=True)
# plt.yticks(rotation=45)
# plt.xticks(rotation=45)
# plt.title("Correlation matrix of stocks")
# plt.show()
# ή
heat_map = sn.heatmap(corrMatrix, linewidths = 1, annot = True, cmap = 'Blues')
fig = heat_map.get_figure()
# fig.savefig('Heatmap.jpg')
```



In [64]:

```python
df.hist(layout = (5,6), figsize = (15,10))
plt.tight_layout()
plt.show()
plt.close()
```



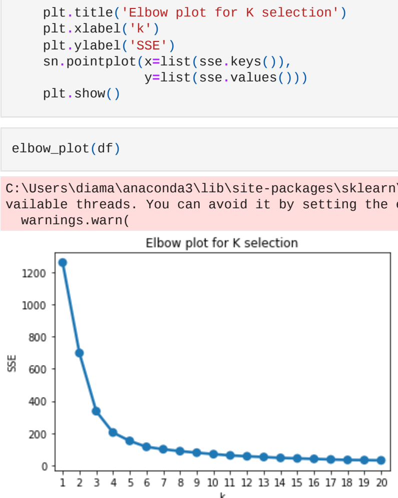
In [65]:

```python
def preprocess(df):
    """Preprocess data for KMeans clustering"""

    df_log = np.log1p(df)
    scaler = StandardScaler()
    scaler.fit(df_log)
    df_norm = scaler.transform(df_log)

    return df_norm

def elbow_plot(df):
    """Create elbow plot from normalized data"""

    df_norm = preprocess(df)

    sse = {}

    for k in range(1, 21):
        kmeans = KMeans(n_clusters=k, random_state=1)
        kmeans.fit(df_norm)
        sse[k] = kmeans.inertia_

    plt.title('Elbow plot for K selection')
    plt.xlabel('k')
    plt.ylabel('SSE')
    sn.pointplot(x=list(sse.keys()),
                 y=list(sse.values()))
    plt.show()
```

In [66]:

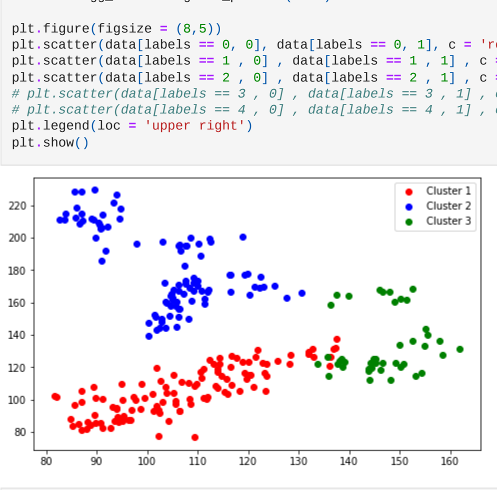
```python
elbow_plot(df)
```

```
C:\Users\diama\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than a
vailable threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [67]:

```python
agg_clustering = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
data = df.to_numpy()

labels = agg_clustering.fit_predict(data)

plt.figure(figsize = (8,5))
plt.scatter(data[labels == 0, 0], data[labels == 0, 1], c = 'red', label = 'Cluster 1')
plt.scatter(data[labels == 1 , 0] , data[labels == 1 , 1] , c = 'blue', label = 'Cluster 2')
plt.scatter(data[labels == 2 , 0] , data[labels == 2 , 1] , c = 'green', label = 'Cluster 3')
# plt.scatter(data[labels == 3 , 0] , data[labels == 3 , 1] , c = 'purple', label = 'Cluster 4')
# plt.scatter(data[labels == 4 , 0] , data[labels == 4 , 1] , c = 'yellow', label = 'Cluster 5')
plt.legend(loc = 'upper right')
plt.show()
```

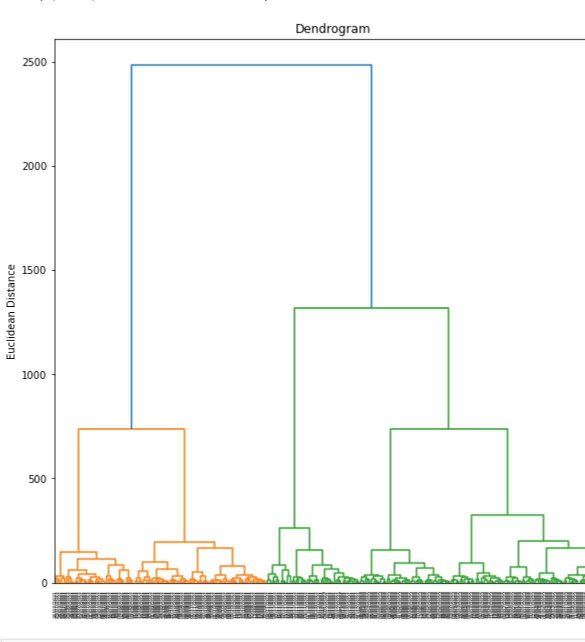


In [68]:

```python
# Dendrogramma ana hmera gia oles tis times kai twn 5 metoxwn
Z = linkage(data, method = 'ward')

plt.figure(figsize = (10,10))
names = df.index.to_list()
dendro = dendrogram(Z, labels = names)
plt.title('Dendrogram')
plt.ylabel('Euclidean Distance')
```

Out[68]:

```
Text(0, 0.5, 'Euclidean Distance')
```

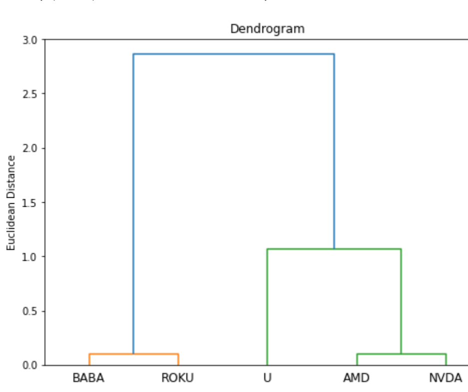


In [69]:

```python
Z = linkage(corr, method = 'ward')

plt.figure(figsize = (8,6))
names = df.columns.to_list()
dendrogram = hierarchy.dendrogram(Z, labels = names)
plt.title('Dendrogram')
plt.ylabel('Euclidean Distance')
```

Out[69]:

```
Text(0, 0.5, 'Euclidean Distance')
```



In [70]:

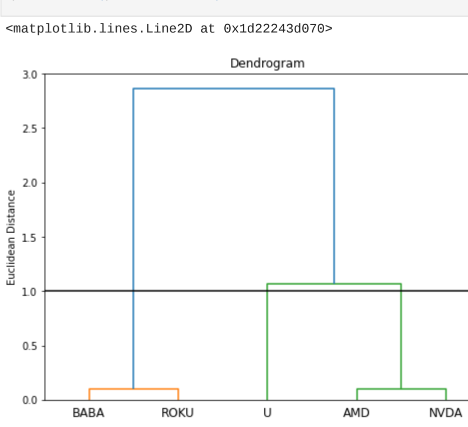
```python
corr_df=df.corr()
corr=corr_df.to_numpy()
```

In [71]:

```python
max_d = 1
Z = linkage(corr, method = 'ward')

plt.figure(figsize = (8,6))
names = df.columns.to_list()
dendrogram = hierarchy.dendrogram(Z, labels = names)
plt.title('Dendrogram')
plt.ylabel('Euclidean Distance')
plt.axhline(y=max_d, c='k')
```

Out[71]:

```
<matplotlib.lines.Line2D at 0x1d22243d070>
```

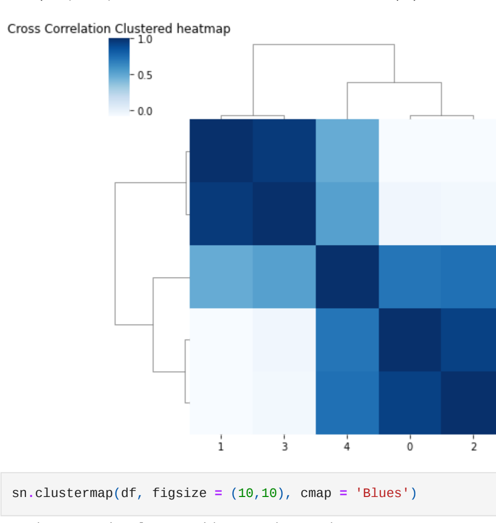


In [72]:

```python
sn.clustermap(corr, figsize = (6,6), cmap = 'Blues')
plt.title('Cross Correlation Clustered heatmap')
```

Out[72]:

```
Text(0.5, 1.0, 'Cross Correlation Clustered heatmap')
```

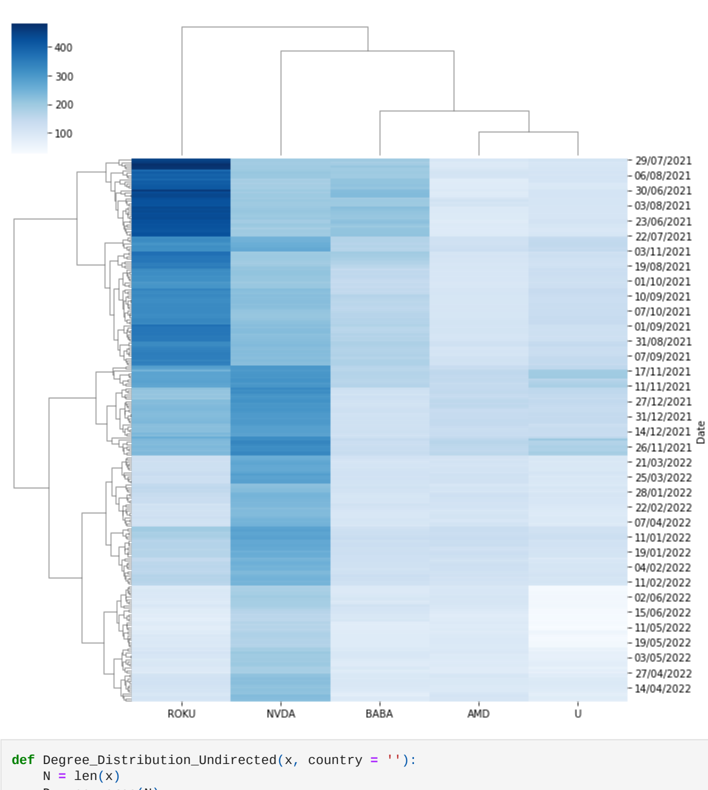


In [73]:

```python
sn.clustermap(df, figsize = (10,10), cmap = 'Blues')
```

Out[73]:

```
<seaborn.matrix.ClusterGrid at 0x1d2220c7dc0>
```

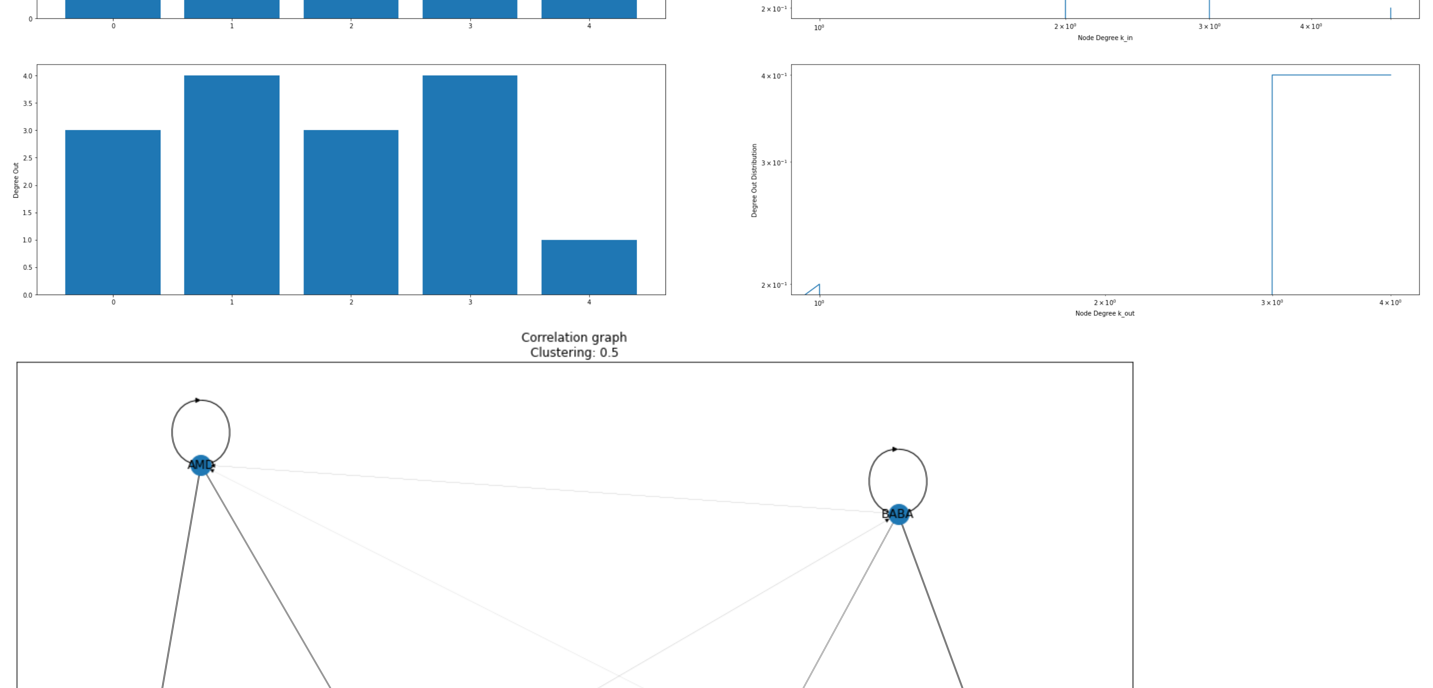


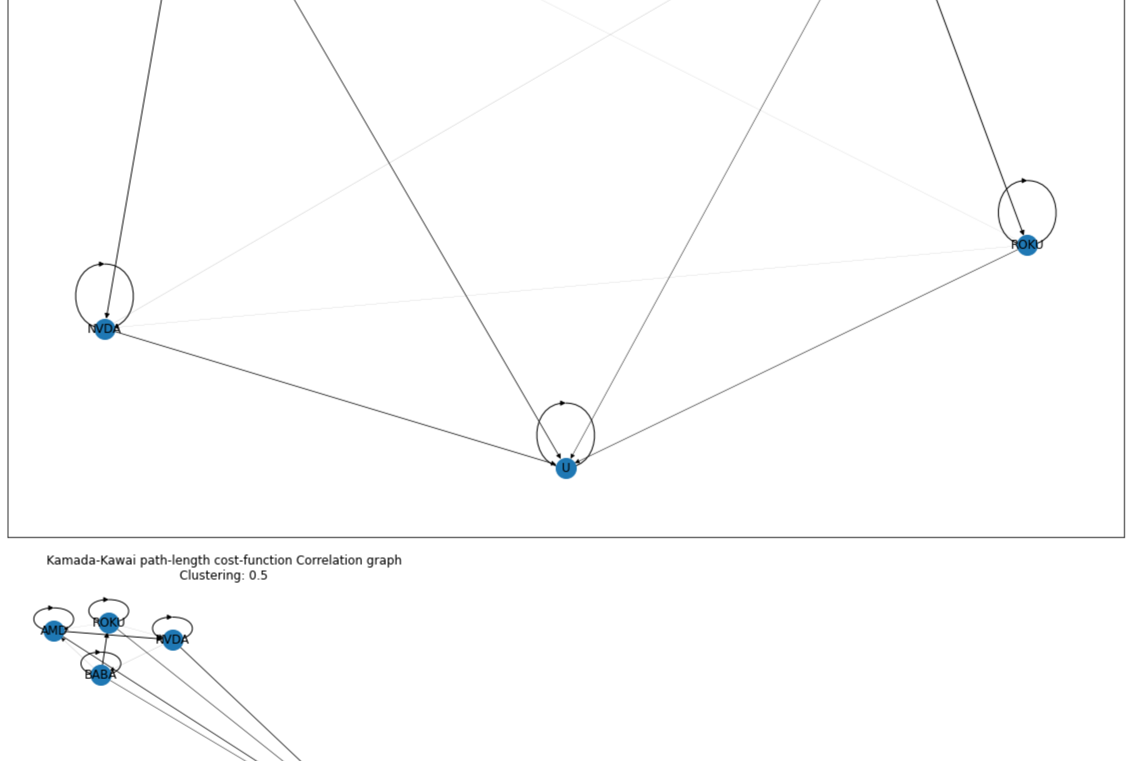
In [97]:

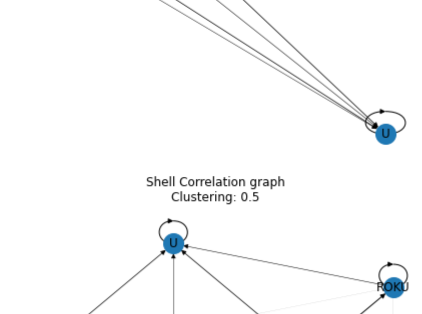
```python
def Degree_Distribution_Undirected(x, country = ''):
    N = len(x)
    D = np.zeros(N)
    for i in range(N):
        D[i] = sum(x[i])
    aver_D = np.mean(D)

    if sum(D) == 0:
        print('No connections in node')
        return
    else:
        fig, ax = plt.subplots(1, 2, figsize = (30,10))
        fig.tight_layout()
        ax[0].bar(range(0, N), D)
        ax[0].set(xlabel = 'Data', ylabel = 'Degree', title = 'Value of Degree Network of Time Series')
        plt.grid()
        M = np.max(D)
        P_D = np.zeros(int(M+1))
        for i in range(int(M+1)):
            P_D[i] = np.count_nonzero(D == i)
        my_sum = sum(P_D)
        P_D = P_D/my_sum
        ax[1].plot(P_D)
        ax[1].set(xlabel = 'Node Degree k', ylabel = 'Degree Distribution', title = 'Degree Distribution')
        if len(country) == 0:
            tit = 'Degree Distribution for Correlation Graph'
        else:
            tit = 'Degree Distribution for {}'.format(country)
        plt.grid()
        fig.subplots_adjust(top=0.88)
        fig.suptitle(tit, size = 16)
        plt.show()
        return D, aver_D

def Degree_Distribution_Directed(x, labels = ''):
    N = len(x)
    k_in = np.zeros(N)
    k_out = np.zeros(N)
    for i in range(N):
        for j in range(i, N):
            if x[i,j] == 0:
                continue
            elif x[i,j] > 0:
                k_out[i] += 1
                k_in[j] += 1
            else:
                k_in[i] += 1
                k_out[j] += 1
    M_in = np.max(k_in)
    M_out = np.max(k_out)
    P_in = np.zeros(int(M_in+1))
    P_out = np.zeros(int(M_out+1))
    for i in range(int(M_in + 1)):
        P_in[i] = np.count_nonzero(k_in == i)
    for i in range(int(M_out + 1)):
        P_out[i] = np.count_nonzero(k_out == i)
    my_sum = sum(P_in)
    P_in = P_in/my_sum
    my_sum = sum(P_out)
    P_out = P_out/my_sum

    fig, ax = plt.subplots(2, 2, figsize = (40,15))
    ax[0,0].bar(range(0, N), k_in)
    ax[0,0].set(ylabel = 'Degree In')
    if labels != '':
        ax[0,0].set_xticks(ticks = range(N), labels = labels, rotation = 45)
    plt.grid()
    ax[0,1].loglog(P_in)
    ax[0,1].set(xlabel = 'Node Degree k_in', ylabel = 'Degree In Distribution')
    tit = 'Degree In Distribution for Correlation Graph'
    plt.grid()
    fig.subplots_adjust(top=0.88)
    fig.suptitle(tit, size = 16)

    ax[1,0].bar(range(0, N), k_out)
    ax[1,0].set(ylabel = 'Degree Out')
    if labels != '':
        ax[1,0].set_xticks(ticks = range(N), labels = labels, rotation = 45)
    plt.grid()
    ax[1,1].loglog(P_out)
    ax[1,1].set(xlabel = 'Node Degree k_out', ylabel = 'Degree Out Distribution')
    tit = 'Degree Out Distribution for Correlation Graph'
    plt.grid()
    fig.subplots_adjust(top=0.88)
    fig.suptitle(tit, size = 16)
    plt.show()
    return k_in, k_out
```

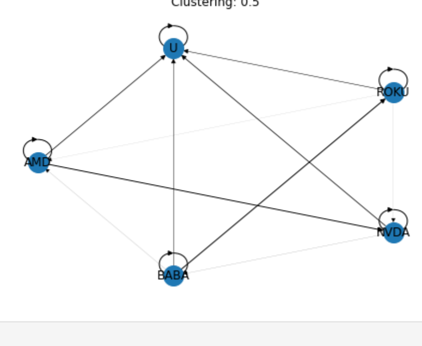
In [98]:

```python
Degree_Distribution_Directed(corr)
tit = 'Degree Distribution for Correlation Graph'
plt.show()
df = pd.DataFrame(columns = ['Source', 'Target', 'Type', 'weight'])
typ = 'Directed'
for i in range(len(corr)):
    for j in range(i, len(corr)):
        w = corr[i,j]
        if w == 0:
            continue
        elif w > 0:
            source = corr_df.index[i]
            target = corr_df.columns[j]
        else:
            source = corr_df.index[j]
            target = corr_df.columns[i]
            w = abs(w)
        tmp_df = pd.DataFrame([[source, target, typ, w]], columns = ['Source', 'Target', 'Type', 'weight'])
        df = pd.concat([df, tmp_df], ignore_index = None)
G1 = nx.from_pandas_edgelist(df, source = 'Source', target = 'Target', edge_attr = 'weight', create_using = nx.DiGraph())
clus = nx.average_clustering(G1)
widths = nx.get_edge_attributes(G1, 'weight')
# for key in widths:
#     widths[key] *= 4/3
options = {
    'node_size': 400,
    'width': list(widths.values()),
    'with_labels': True,
    'font_size': 12
}
fig = plt.figure(1, figsize = (20,15))
nx.draw_networkx(G1, **options)
tit = 'Correlation graph\nClustering: {}'.format(clus)
plt.title(tit)
tit = 'Correlation graph'
plt.show()
plt.clf()

nx.draw_circular(G1, **options)
tit = 'Circular Correlation graph\nClustering: {}'.format(clus)
plt.title(tit)
tit = 'Circular Correlation graph'
# plt.show()
plt.clf()

nx.draw_kamada_kawai(G1, **options)
tit = 'Kamada-Kawai path-length cost-function Correlation graph\nClustering: {}'.format(clus)
plt.title(tit)
tit = 'Kamada-Kawai path-length cost-function Correlation graph'
plt.show()
plt.clf()

nx.draw_shell(G1, **options)
tit = 'Shell Correlation graph\nClustering: {}'.format(clus)
plt.title(tit)
tit = 'Shell Correlation graph'
plt.show()
plt.clf()

plt.close()
```









In [ ]: