

Stocks selection

We evaluated the 129 stocks present on our database in order to select a subgroup for the network analysis.

Summary ¶

For the network analysis that will be performed for minimal covariance portfolio selection, some characteristics of the data should be present, namely:

1. The time series should not have too many missing values
2. The covariance matrix between the stocks should generate an irreducible graph, meaning that all the stocks should communicate with each other (so we can compute the betweenness)

Exploratory Data Analysis and Cleaning

Before we dive into the meat of our asset allocation model, we first explore, clean, and preprocess our historical price data for time-series analyses. In this section we complete the following.

- Observe how many rows and columns are in our dataset and what they mean
- Observe the datatypes of the columns and update them if needed
- Take note of how the data is structured and what preprocessing will be necessary for time-series analyses
- Deal with any missing data accordingly
- Test which time series satisfies the conditions for our model

In [1]:

```
#import data manipulation (pandas) and numerical manipulation (numpy) modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#silence warnings
import warnings
warnings.filterwarnings("ignore")
```

In [47]:

```
#reads the csv file into pandas DataFrame
# df = pd.read_csv("https://github.com/firmai/random-assets/blob/master/all_stocks.csv?raw=true")
# df = get_daily_from_db()
df = pd.read_csv(r"data/20130102_20200529_daily.csv", index_col=0)

#prints first 5 rows of the DataFrame
df.head()
```

Out[47]:

	Ticker	Open	Low	High	Close	Volume	Name
Day							
2013-01-02	ABCB4	14.00	14.00	14.27	14.15	5224632.0	ABC BRASIL
2013-01-02	ALPA4	15.10	14.98	15.30	15.16	2719722.0	ALPARGATAS
2013-01-02	AMAR3	32.55	32.54	33.01	32.63	7420976.0	LOJAS MARISA
2013-01-02	BBAS3	26.00	25.46	26.19	25.80	220234920.0	BRASIL
2013-01-02	BBDC3	34.30	34.30	35.43	35.11	39184241.0	BRADESCO

- Day : date (yyyy-mm-dd), index of the data frame
- Ticker : The ticker code for the stocks
- Open : daily opening prices (USD)
- Low : daily low prices (USD)
- High : daily high prices (USD)
- Close : daily closing prices (USD)
- Volume : daily volume (number of shares traded)
- Name : Short name of the company

In [41]:

```
#prints last 5 rows
df.tail()
```

Out[41]:

	Ticker	Open	Low	High	Close	Volume	Name
Day							
2020-05-29	VIVT4	47.39	46.39	47.53	47.14	2.268595e+08	TELEF BRASIL
2020-05-29	VVAR3	12.45	11.87	12.70	12.40	1.252818e+09	VIAVAREJO
2020-05-29	WEGE3	40.20	39.61	41.83	41.83	3.634596e+08	WEG
2020-05-29	YDUQ3	28.40	27.67	28.95	28.48	8.791632e+07	YDUQS PART
2020-05-29	OIBR3	0.73	0.71	0.75	0.75	1.037060e+08	OI

In [42]:

```
#prints information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 185853 entries, 2013-01-02 to 2020-05-29
Data columns (total 7 columns):
Ticker    185853 non-null object
Open      185853 non-null float64
Low       185853 non-null float64
High      185853 non-null float64
Close     185853 non-null float64
Volume    185853 non-null float64
Name      185853 non-null object
dtypes: float64(5), object(2)
memory usage: 11.3+ MB
```

In [43]:

```
#prints unique tickers in the Name column
print("There are {} unique stocks\n\n".format(len(df['Ticker'].unique())))
print(df['Ticker'].unique())
```

There are 124 unique stocks

```
[ 'ACB3' 'ALPA4' 'AMAR3' 'BBAS3' 'BBDC3' 'BBDC4' 'BEEF3' 'BRAP4' 'BRFS3'
 'BRML3' 'BRPR3' 'BTOW3' 'CCRO3' 'CIEL3' 'CMIG3' 'CMIG4' 'CPFE3' 'CSAN3'
 'CSMG3' 'CSNA3' 'CYRE3' 'DIRR3' 'DTEX3' 'ECOR3' 'ELET3' 'EMBR3' 'ENBR3'
 'EQTL3' 'EVEN3' 'EZTC3' 'FLRY3' 'GFS3' 'GGBR4' 'GOAU4' 'GOLL4' 'GRND3'
 'GUAR3' 'HBOR3' 'HGTX3' 'HYPE3' 'IGTA3' 'ITSA4' 'ITUB3' 'ITUB4' 'JBSS3'
 'JHSF3' 'JSLG3' 'KLBN4' 'LAME3' 'LAME4' 'LCAM3' 'LIGT3' 'LOGN3' 'LREN3'
 'MDIA3' 'MGLU3' 'MILS3' 'MRFG3' 'MRVE3' 'MULT3' 'MYPK3' 'OIBR3' 'PETR3'
 'PETR4' 'POMO4' 'POS13' 'PSSA3' 'QUAL3' 'RADL3' 'RAPT4' 'RENT3' 'SBSP3'
 'SLCE3' 'SMT03' 'TCSA3' 'TIMP3' 'TOTS3' 'TRIS3' 'TRPL4' 'TUPY3' 'UGPA3'
 'VALE3' 'VIVT4' 'WEGE3' 'VVAR3' 'LINX3' 'BBSE3' 'ENEV3' 'ANIM3' 'SEER3'
 'ABEV3' 'CVCB3' 'BPAN4' 'RLOG3' 'MEAL3' 'PRI03' 'PCAR3' 'EGIE3' 'STBP3'
 'MOVI3' 'RAIL3' 'AZUL4' 'TEND3' 'CRFB3' 'IRBR3' 'CAML3' 'SMLS3' 'SUZB3'
 'BRDT3' 'BKBR3' 'B3SA3' 'GNDI3' 'HAPV3' 'BIDI4' 'SQIA3' 'CNT03' 'ENAT3'
 'NEOE3' 'YDUQ3' 'ALS03' 'VIVA3' 'COGN3' 'CEAB3' 'NTC03' ]
```

Preprocessing for Time-Series Analysis

In this section we do the following.

1. Remove stocks from the same company (e.g.: ITUB4 - preferred type - and ITUB3 - ordinary type);
2. We create a separate DataFrame for the Open , High , Low , and Close time-series;
 - Pivot the tickers in the Ticker column of df to the column names of the above DataFrames and set the values as the daily prices
3. Transform each time-series so that it's stationary;
 - We do this by transforming the prices in returns with the pd.pct_change() method
4. Remove the missing data;
5. Examine the assumptions of our minimal risk portfolio selection model.

In [44]:

```
# Range of our data
print("The data contains stock prices from {} to {}".format(df.index.min(), df.index.max()))
```

The data contains stock prices from 2013-01-02 to 2020-05-29.

In [13]:

```
# creates a new column with the difference between the closing and opening prices
# Not creating this
# df['Price Range'] = df.loc[:, 'Close'] - df.loc[:, 'Open']
# df.head()
```

1. Removing tickers from the same company

In [66]:

```
# determine which companies have more than one Ticker
companies = df.Name.unique()
companies_df = pd.DataFrame({'Name': companies, 'n': [len(df.loc[df.Name == b].Ticker.unique()) for b in companies]})
companies_df.loc[companies_df.n > 1].Name
```

Out[66]:

```
4          BRADESCO
13         CEMIG
40        ITAUUNIBANCO
45       LOJAS AMERIC
58      PETROBRAS
Name: Name, dtype: object
```

In [69]:

```
# Getting the Tickers
comp_tickers = set(companies_df.loc[companies_df.n > 1].Name)
df.loc[df.Name.apply(lambda x: x in comp_tickers), "Ticker"].unique()
```

Out[69]:

```
array(['BBDC3', 'BBDC4', 'CMIG3', 'CMIG4', 'ITUB3', 'ITUB4', 'LAME3',
       'LAME4', 'PETR3', 'PETR4'], dtype=object)
```

We see that BRADESCO has the tickers BBDC3 and BBDC4 , and the same for all the 5 companies. We will select the stocks 4 (preferred)

In [70]:

```
print("Before: {}".format(df.shape))
ordinary_to_remove = set(['BBDC3', 'CMIG3', 'ITUB3', 'LAME3', 'PETR3'])
df = df.loc[df.Ticker.apply(lambda x: x not in ordinary_to_remove)]
print("After: {}".format(df.shape))
```

Before: (185853, 7)

After: (176693, 7)

2. Create data frames for each daily price

In [71]:

```
#creates a DataFrame for each time-series (see In [11])
df_close = df.pivot(columns='Ticker', values='Close')
df_open = df.pivot(columns='Ticker', values='Open')
# df_price_range = df.pivot(columns='Ticker', values='Price Range')
df_high = df.pivot(columns='Ticker', values='High')
df_low = df.pivot(columns='Ticker', values='Low')

df_close.head()
```

Out[71]:

Ticker	ABCB4	ABEV3	ALPA4	ALSO3	AMAR3	ANIM3	AZUL4	B3SA3	BBAS3	BBDC4	
Day											
2013-01-02	14.15	NaN	15.16	NaN	32.63	NaN	NaN	NaN	25.80	36.02	.
2013-01-03	14.19	NaN	15.15	NaN	32.05	NaN	NaN	NaN	26.31	38.12	.
2013-01-04	13.99	NaN	15.19	NaN	32.05	NaN	NaN	NaN	26.00	37.45	.
2013-01-07	14.10	NaN	14.85	NaN	32.09	NaN	NaN	NaN	26.15	37.29	.
2013-01-08	14.25	NaN	14.65	NaN	31.70	NaN	NaN	NaN	26.45	37.42	.

5 rows × 119 columns



3. Detrending and Additional Data Cleaning

In [72]:

```
# creates a list of stocks
stocks = df_close.columns.tolist()

#List of training DataFrames containing each time-series
df_list = [df_close, df_open, df_high, df_low]

# detrends each time-series for each DataFrame
for df in df_list:
    for s in stocks:
        df[s] = df[s].pct_change()
```

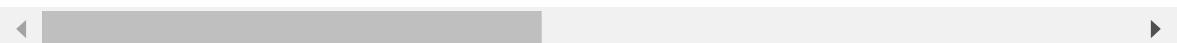
In [73]:

```
df_close.head()
```

Out[73]:

Ticker	ABCB4	ABEV3	ALPA4	ALSO3	AMAR3	ANIM3	AZUL4	B3SA3	BBAS3
Day									
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	0.002827	NaN	-0.000660	NaN	-0.017775	NaN	NaN	NaN	0.019767
2013-01-04	-0.014094	NaN	0.002640	NaN	0.000000	NaN	NaN	NaN	-0.011783
2013-01-07	0.007863	NaN	-0.022383	NaN	0.001248	NaN	NaN	NaN	0.005769
2013-01-08	0.010638	NaN	-0.013468	NaN	-0.012153	NaN	NaN	NaN	0.011472

5 rows × 119 columns



4. Remove missing data

In [74]:

```
#counts the missing values in each column
missing_values = pd.concat([a.isnull().sum() for a in df_list], axis=1)
missing_values['avg'] = missing_values.mean(axis=1)
missing_values
```

Out[74]:

Ticker	0	1	2	3	avg
ABCB4	1	1	1	1	1.0
ABEV3	217	217	217	217	217.0
ALPA4	1	1	1	1	1.0
ALSO3	1630	1630	1630	1630	1630.0
AMAR3	1	1	1	1	1.0
...
VIVA3	1677	1677	1677	1677	1677.0
VIVT4	1	1	1	1	1.0
VVAR3	2	2	2	2	2.0
WEGE3	1	1	1	1	1.0
YDUQ3	1620	1620	1620	1620	1620.0

119 rows × 5 columns

In [75]:

```
missing_values.avg.describe()
```

Out[75]:

count	119.000000
mean	333.210084
std	564.567269
min	1.000000
25%	1.000000
50%	1.000000
75%	497.500000
max	1724.000000
Name:	avg, dtype: float64

Many tickers have too many missing values. We will filter them out.

In [76]:

```
miss_avg = missing_values['avg']

# Remove Tickers with more than 3 errors (from 11 and more)
to_remove = miss_avg.loc[miss_avg > 3].index
print("{} will be removed, and {} will remain in the dataset".format(len(to_remove), len(miss_avg)-len(to_remove)))
to_remove
```

39 will be removed, and 80 will remain in the dataset

Out[76]:

```
Index(['ABEV3', 'ALSO3', 'ANIM3', 'AZUL4', 'B3SA3', 'BBSE3', 'BIDI4', 'BKB
R3',
       'BPAN4', 'BRDT3', 'CAML3', 'CEAB3', 'CNT03', 'COGN3', 'CRFB3', 'CVC
B3',
       'EGIE3', 'ENAT3', 'ENEV3', 'GNDI3', 'HAPV3', 'IRBR3', 'LINX3', 'MEA
L3',
       'MOVI3', 'NEOE3', 'NTC03', 'PCAR3', 'PRI03', 'RAIL3', 'RLOG3', 'SEE
R3',
       'SMLS3', 'SQIA3', 'STBP3', 'SUZB3', 'TEND3', 'VIVA3', 'YDUQ3'],
      dtype='object', name='Ticker')
```

In [77]:

```
# Remove from the data
print("Close price before: {}".format(df_close.shape))
for df in df_list:
    df.drop(to_remove, axis=1, inplace=True)

print("Close price after: {}".format(df_close.shape))
```

Close price before: (1832, 119)
Close price after: (1832, 80)

In [80]:

```
df_close.isnull().sum().describe()
```

Out[80]:

count	80.000000
mean	1.012500
std	0.111803
min	1.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	2.000000
dtype:	float64

Since the number of remaining missing values in every column is considerably small, we can safely drop them.

In [81]:

```
#drops all missing values in each DataFrame
for df in df_list:
    df.dropna(inplace=True)

# Update List of stocks
stocks = df_close.columns.tolist()
np.array(stocks) # Just because it prints better
```

Out[81]:

```
array(['ABCB4', 'ALPA4', 'AMAR3', 'BBAS3', 'BBDC4', 'BEEF3', 'BRAP4',
       'BRFS3', 'BRML3', 'BRPR3', 'BTOW3', 'CCRO3', 'CIEL3', 'CMIG4',
       'CPFE3', 'CSAN3', 'CSMG3', 'CSNA3', 'CYRE3', 'DIRR3', 'DTEX3',
       'ECOR3', 'ELET3', 'EMBR3', 'ENBR3', 'EQTL3', 'EVEN3', 'EZTC3',
       'FLRY3', 'GFS4', 'GGBR4', 'GOAU4', 'GOLL4', 'GRND3', 'GUAR3',
       'HBOR3', 'HGTX3', 'HYPE3', 'IGTA3', 'ITSA4', 'ITUB4', 'JBSS3',
       'JHSF3', 'JSLG3', 'KLBN4', 'LAME4', 'LCAM3', 'LIGT3', 'LOGN3',
       'LREN3', 'MDIA3', 'MGLU3', 'MILS3', 'MRFG3', 'MRVE3', 'MULT3',
       'MYPK3', 'OIBR3', 'PETR4', 'POMO4', 'POSI3', 'PSSA3', 'QUAL3',
       'RADL3', 'RAPT4', 'RENT3', 'SBSP3', 'SLCE3', 'SMT03', 'TCSA3',
       'TIMP3', 'TOTS3', 'TRIS3', 'TRPL4', 'TUPY3', 'UGPA3', 'VALE3',
       'VIVT4', 'VVAR3', 'WEGE3'], dtype='<U5')
```

Examining the conditions for our model

Now we already have a selected group of 80 stocks. We will examine if they can be used on our betweenness centrality model.

1. We compute the *distance correlation matrix* $\rho_D(X_i, X_j)$ for the Open, High, Low, and Close time series.
2. We create a master matrix with the average between the prices.
3. We use the NetworkX module to transform each distance correlation matrix into a weighted graph.
4. We adopt the winner-take-all method and remove edges with correlations below a threshold value that will be determined in order to maintain a connected graph. If the value is too small, we will discard additional stocks.

$$Cor_{ij} = \begin{cases} \rho_D(X_i, Y_j), & \rho \geq \rho_c \\ 0, & \text{otherwise.} \end{cases}$$

To find the threshold, we will compute the degree distribution of the nodes in the graph, and will iterate with different values until we find a value that is "big enough" (we want to limit the number of connections between low correlated stocks) and at the same time the corresponding graph is irreducible (fully connected). Algebraically, the degree of the i th vertex is given as,

$$\text{Deg}(i) = \sum_{j=1}^N A_{ij}$$

What on Earth is Distance Correlation and Why Should We Care?

Put simply, [Distance correlation \(\[https://en.wikipedia.org/wiki/Distance_correlation#cite_note-SR2007-2\]\(https://en.wikipedia.org/wiki/Distance_correlation#cite_note-SR2007-2\)\)](https://en.wikipedia.org/wiki/Distance_correlation#cite_note-SR2007-2) is a generalization of Pearson's correlation insofar as it (1) detects both linear and non-linear associations in the data and (2) can be applied to time series of unequal dimension.

Distance correlation varies between 0 and 1. A Distance correlation close to 0 indicates a pair of time series is **independent** where values close to 1 indicate a high degree of **dependence**. This is in contrast to Pearson's correlation which varies between -1 and 1 and can be 0 for time series that are dependent (see [Szekely, et al. \(2017\) \(<https://arxiv.org/pdf/0803.4101.pdf>\)](https://arxiv.org/pdf/0803.4101.pdf)).

What makes Distance correlation particularly appealing is the fact that it can be applied to time series of unequal dimension. If our ultimate goal is to scale the asset allocation algorithm to the *entire* market (with time series of many assets) and update it in real-time (which it is), the algo must be able to handle time series of arbitrary dimension. The penultimate goal is to observe how an asset correlation network *representative* of the global market evolves in real-time and update the allocation weights in response.

TODO: See if copulas can be an alternative (Kendal Tau for correlation matrix -> use negative values to "compensate" risk)

1. Calculating the Distance Correlation Matrix with dcor

In [82]:

```
#imports the dcor module to calculate distance correlation
import dcor

#function to compute the distance correlation (dcor) matrix from a DataFrame and output
a DataFrame
#of dcor values.
def df_distance_correlation(df):

    #initializes an empty DataFrame
    df_dcor = pd.DataFrame(index=stocks, columns=stocks)

    #initialzes a counter at zero
    k=0

    # iterates over the time series of eachstocks stock
    for i in stocks:

        # stores the ith time series as a vector
        v_i = df.loc[:, i].values

        # iterates over the time series of each stock subect to the counter k
        for j in stocks[k:]:

            # stores the jth time series as a vector
            v_j = df.loc[:, j].values

            # computes the dcor coefficient between the ith and jth vectors
            dcor_val = dcor.distance_correlation(v_i, v_j)

            # appends the dcor value at every ij entry of the empty DataFrame
            df_dcor.at[i,j] = dcor_val

            # appends the dcor value at every ji entry of the empty DataFrame
            df_dcor.at[j,i] = dcor_val

        # increments counter by 1
        k+=1

    # returns a DataFrame of dcor values for every pair of stocks
    return df_dcor
```

In [83]:

```
# Distance correlation for df_close, df_open, df_price_range, df_high, df_low
df_dcor_list = [df_distance_correlation(df) for df in df_list]
```

2. Creation of the average data

In [115]:

```
# initializes a DataFrame full of zeros
df_zeros = pd.DataFrame(index=stocks, columns=stocks).fillna(0)

# iterates over the length of the DataFrame list containing the Open, High, Low, Close, and Price Range
# time series
for i in range(len(df_list)):

    # Adds the distance correlation DataFrames of the Open, High, Low, Close, and Price Range
    # time series together
    df_zeros += df_dcor_list[i]

# Takes the average of the distance correlation DataFrames
df_master = df_zeros/len(df_list)
df_dcor_master = df_distance_correlation(df_master)
```

In [86]:

```
print("Check the resulting correlation matrix for Close price:\n\n")
df_dcor_list[0].head()
```

Check the resulting correlation matrix for Close price:

Out[86]:

	ABCB4	ALPA4	AMAR3	BBAS3	BBDC4	BEEF3	BRAP4	BRFS3	E
ABCB4	1	0.258812	0.242263	0.464469	0.483814	0.162692	0.253581	0.218237	0.3
ALPA4	0.258812	1	0.251698	0.29269	0.298628	0.15601	0.165893	0.193373	0.3
AMAR3	0.242263	0.251698	1	0.294654	0.292318	0.155758	0.171539	0.187853	0.3
BBAS3	0.464469	0.29269	0.294654	1	0.70144	0.163379	0.352104	0.300505	0.4
BBDC4	0.483814	0.298628	0.292318	0.70144	1	0.169016	0.371825	0.318448	0.5

5 rows × 80 columns



3. Building a Time-Series Correlation Network with Networkx

In [87]:

```
#imports the NetworkX module
import networkx as nx

# takes in a pre-processed dataframe and returns a time-series correlation
# network with pairwise distance correlation values as the edges
def build_corr_nx(df, corr_threshold=0.325):

    # converts the distance correlation dataframe to a numpy matrix with dtype float
    cor_matrix = df.values.astype('float')

    # Since dcor ranges between 0 and 1, (0 corresponding to independence and 1
    # corresponding to dependence), 1 - cor_matrix results in values closer to 0
    # indicating a higher degree of dependence where values close to 1 indicate a lower
    # degree of
    # dependence. This will result in a network with nodes in close proximity reflectin
    g the similarity
    # of their respective time-series and vice versa.
    sim_matrix = 1 - cor_matrix

    # transforms the similarity matrix into a graph
    G = nx.from_numpy_matrix(sim_matrix)

    # extracts the indices (i.e., the stock names from the dataframe)
    stock_names = df.index.values

    # relabels the nodes of the network with the stock names
    G = nx.relabel_nodes(G, lambda x: stock_names[x])

    # assigns the edges of the network weights (i.e., the dcor values)
    G.edges(data=True)

    # copies G
    ## we need this to delete edges or otherwise modify G
    H = G.copy()

    # iterates over the edges of H (the u-v pairs) and the weights (wt)
    for (u, v, wt) in G.edges.data('weight'):
        # selects edges with dcor values less than or equal to 0.33
        if wt >= 1 - corr_threshold:
            # removes the edges
            H.remove_edge(u, v)

        # selects self-edges
        if u == v:
            # removes the self-edges
            H.remove_edge(u, v)

    # returns the final stock correlation network
    return H
```

In [116]:

```
#builds the distance correlation networks for the Open, Close, High, Low, and Price Range time series
# Initially we will use corr_threshold=0.325
H_close = build_corr_nx(df_dcor_list[0], corr_threshold=0.325)
H_open = build_corr_nx(df_dcor_list[1], corr_threshold=0.325)
# H_price_range = build_corr_nx(df_dcor_list[2], corr_threshold=0.325)
H_high = build_corr_nx(df_dcor_list[2], corr_threshold=0.325)
H_low = build_corr_nx(df_dcor_list[3], corr_threshold=0.325)
# Builds the master network with the averaged distance correlation DataFrame
H_master = build_corr_nx(df_dcor_master, corr_threshold=0.325)
```

In [96]:

```
def is_irreducible(H):
    for node, weight in H.degree():
        if weight == 0:
            return False
    return True
```

In [97]:

```
[is_irreducible(a) for a in [H_close, H_open, H_high, H_low, H_master]]
```

Out[97]:

```
[False, False, False, False, False]
```

None of the graphs were irreducible using the threshold 0.325 . We will grid search for different values

In [108]:

```
def grid_search_threshold(df_dcor, threshold_list):
    for threshold in threshold_list:
        print("Testing for threshold {:.4f}: ".format(threshold))
        H = build_corr_nx(df_dcor, corr_threshold=threshold)
        print("Result: {}".format("Irreducible!" if is_irreducible(H) else "Not irreducible!"))
        print()
```

In [111]:

```
threshold_list = [0.0, 0.1, 0.15, 0.2, 0.25, 0.3]
print("Testing for Close price: \n")
grid_search_threshold(df_dcor_list[0], threshold_list)
```

Testing for Close price:

Testing for threshold 0.0000:
Result: Irreducible!

Testing for threshold 0.1000:
Result: Irreducible!

Testing for threshold 0.1500:
Result: Irreducible!

Testing for threshold 0.2000:
Result: Not irreducible!

Testing for threshold 0.2500:
Result: Not irreducible!

Testing for threshold 0.3000:
Result: Not irreducible!

In [112]:

```
print("Testing for Open price: \n")
# df_close, df_open, df_high, df_low
grid_search_threshold(df_dcor_list[1], threshold_list)
```

Testing for Open price:

Testing for threshold 0.0000:
Result: Irreducible!

Testing for threshold 0.1000:
Result: Irreducible!

Testing for threshold 0.1500:
Result: Irreducible!

Testing for threshold 0.2000:
Result: Not irreducible!

Testing for threshold 0.2500:
Result: Not irreducible!

Testing for threshold 0.3000:
Result: Not irreducible!

In [113]:

```
print("Testing for High price: \n")
# df_close, df_open, df_high, df_low
grid_search_threshold(df_dcor_list[2], threshold_list)
```

Testing for High price:

Testing for threshold 0.0000:
Result: Irreducible!

Testing for threshold 0.1000:
Result: Irreducible!

Testing for threshold 0.1500:
Result: Irreducible!

Testing for threshold 0.2000:
Result: Irreducible!

Testing for threshold 0.2500:
Result: Not irreducible!

Testing for threshold 0.3000:
Result: Not irreducible!

In [114]:

```
print("Testing for Low price: \n")
grid_search_threshold(df_dcor_list[3], threshold_list)
```

Testing for Low price:

Testing for threshold 0.0000:
Result: Irreducible!

Testing for threshold 0.1000:
Result: Irreducible!

Testing for threshold 0.1500:
Result: Irreducible!

Testing for threshold 0.2000:
Result: Irreducible!

Testing for threshold 0.2500:
Result: Not irreducible!

Testing for threshold 0.3000:
Result: Not irreducible!

In [117]:

```
print("Testing for the average price: \n")
grid_search_threshold(df_dcor_master, threshold_list)
```

Testing for the average price:

Testing for threshold 0.0000:
Result: Irreducible!

Testing for threshold 0.1000:
Result: Irreducible!

Testing for threshold 0.1500:
Result: Irreducible!

Testing for threshold 0.2000:
Result: Irreducible!

Testing for threshold 0.2500:
Result: Irreducible!

Testing for threshold 0.3000:
Result: Irreducible!

In [118]:

```
grid_search_threshold(df_dcor_master, [0.35, 0.4, 0.45, 0.5, 0.55])
```

Testing for threshold 0.3500:
Result: Irreducible!

Testing for threshold 0.4000:
Result: Irreducible!

Testing for threshold 0.4500:
Result: Irreducible!

Testing for threshold 0.5000:
Result: Not irreducible!

Testing for threshold 0.5500:
Result: Not irreducible!

For the individual graphs, we see that the threshold of 0.15 keeps the graph connected, while for the graph made on the average price we can use a threshold of 0.45

Plotting a Time-Series Correlation Network with Seaborn

In [147]:

```
sns.choose_cubehelix_palette()
```

Out[147]:

```
[[0.9312692223325372, 0.8201921796082118, 0.7971480974663592],  
 [0.8888663743660877, 0.7106793139856472, 0.7158661451411206],  
 [0.8314793143949643, 0.5987041921652179, 0.6530062709235388],  
 [0.7588951019517731, 0.49817117746394224, 0.6058723814510268],  
 [0.6672565752652589, 0.40671838146419587, 0.5620016466433286],  
 [0.5529215689527474, 0.3217924564263954, 0.5093718054521851],  
 [0.43082755198027817, 0.24984535814964698, 0.44393960899639856],  
 [0.29794615023641036, 0.18145907625614888, 0.35317781405034754],  
 [0.1750865648952205, 0.11840023306916837, 0.24215989137836502]]
```

We will plot the graphs of the average data for different thresholds to analyze the difference. Then we can decide if we will use the averaged price or if we prefer to use, for example, the close prices with a lower threshold.

In [174]:

```
# function to display the network from the distance correlation matrix
def plt_corr_nx(H, title):

    # creates a set of tuples: the edges of G and their corresponding weights
    edges, weights = zip(*nx.get_edge_attributes(H, "weight").items())

    # This draws the network with the Kamada-Kawai path-length cost-function.
    # Nodes are positioned by treating the network as a physical ball-and-spring system.
    # The locations of the nodes are such that the total energy of the system is minimized.
    pos = nx.kamada_kawai_layout(H)

    with sns.axes_style('whitegrid'):
        # figure size and style
        plt.figure(figsize=(16, 9))
        plt.title(title, size=16)

        # computes the degree (number of connections) of each node
        deg = H.degree

        # List of node names
        nodelist = []
        # List of node sizes
        node_sizes = []

        # iterates over deg and appends the node names and degrees
        for n, d in deg:
            nodelist.append(n)
            node_sizes.append(d)

        # draw nodes
        nx.draw_networkx_nodes(
            H,
            pos,
            node_color="blue", "#DA70D6",
            nodelist=nodelist,
            node_size=[(x+1) * 20 for x in node_sizes], #np.power(node_sizes, 2.33),
            alpha=0.8,
            font_weight="bold",
        )

        # node label styles
        nx.draw_networkx_labels(H, pos, font_size=13, font_family="sans-serif", font_weight='bold')

        # color map
        cmap = sns.cubehelix_palette(10, start=3.0, dark=0.1, light=0.7, as_cmap=True,
                                     reverse=True)

        # draw edges
        nx.draw_networkx_edges(
            H,
            pos,
            edge_list=edges,
            style="solid",
            edge_color=weights,
            edge_cmap=cmap,
            edge_vmin=min(weights),
            edge_vmax=max(weights),
```

```

        )

    # builds a colorbar
    sm = plt.cm.ScalarMappable(
        cmap=cmap,
        norm=plt.Normalize(vmin=min(weights),
                           vmax=max(weights))
    )
    sm._A = []
    plt.colorbar(sm)

    # displays network without axes
    plt.axis("off")

# function to visualize the degree distribution
def hist_plot(network, title, bins, xticks):

    # extracts the degrees of each vertex and stores them as a list
    deg_list = list(dict(network.degree).values())

    # sets local style
    with plt.style.context('fivethirtyeight'):
        # initializes a figure
        plt.figure(figsize=(9,6))

        # plots a pretty degree histogram with a kernel density estimator
        sns.distplot(
            deg_list,
            kde=True,
            bins = bins,
            color='darksalmon',
            hist_kws={'alpha': 0.7}

        );

        # turns the grid off
        plt.grid(False)

        # controls the number and spacing of xticks and yticks
        #xticks = range()
        plt.xticks(xticks, size=11)
        plt.yticks(size=11)

        # removes the figure spines
        sns.despine(left=True, right=True, bottom=True, top=True)

        # Labels the y and x axis
        plt.ylabel("Probability", size=15)
        plt.xlabel("Number of Connections", size=15)

        # sets the title
        plt.title(title, size=20);

        # draws a vertical line where the mean is
        plt.axvline(sum(deg_list)/len(deg_list),
                    color='darkorchid',
                    linewidth=3,
                    linestyle='--',
                    label='Mean = {:.2f}'.format(sum(deg_list)/len(deg_list)))
    )

```

```
# turns the Legend on
plt.legend(loc=0, fontsize=12)
```

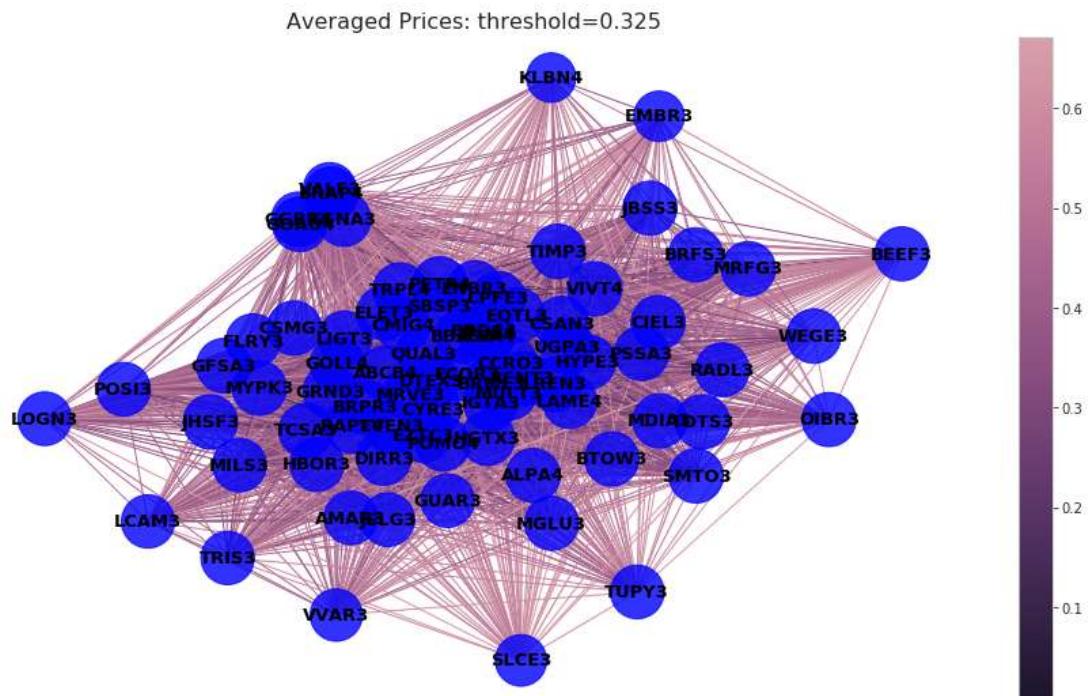
Visualizing How A Portfolio is Correlated with Itself (with Physics)

The following visualizations are rendered with the [Kamada-Kawai method](#) (<http://jgaa.info/accepted/2002/HarelKoren2002.6.3.pdf>), which treats each vertex of the graph as a mass and each edge as a spring. The graph is drawn by finding the list of vertex positions that minimize the total energy of the ball-spring system. The method treats the spring lengths as the weights of the graph, which is given by `1 - cor_matrix` where `cor_matrix` is the distance correlation matrix. Nodes separated by large distances reflect smaller correlations between their time series data, while nodes separated by small distances reflect larger correlations. The minimum energy configuration consists of vertices with few connections experiencing a repulsive force and vertices with many connections feeling an attractive force. As such, nodes with a larger degree (more correlations) fall towards the center of the visualization where nodes with a smaller degree (fewer correlations) are pushed outwards. For an overview of physics-based graph visualizations see the [Force-directed graph drawing](#) (https://en.wikipedia.org/wiki/Force-directed_graph_drawing) wiki.

First, we plot the average prices with the threshold of 0.325:

In [149]:

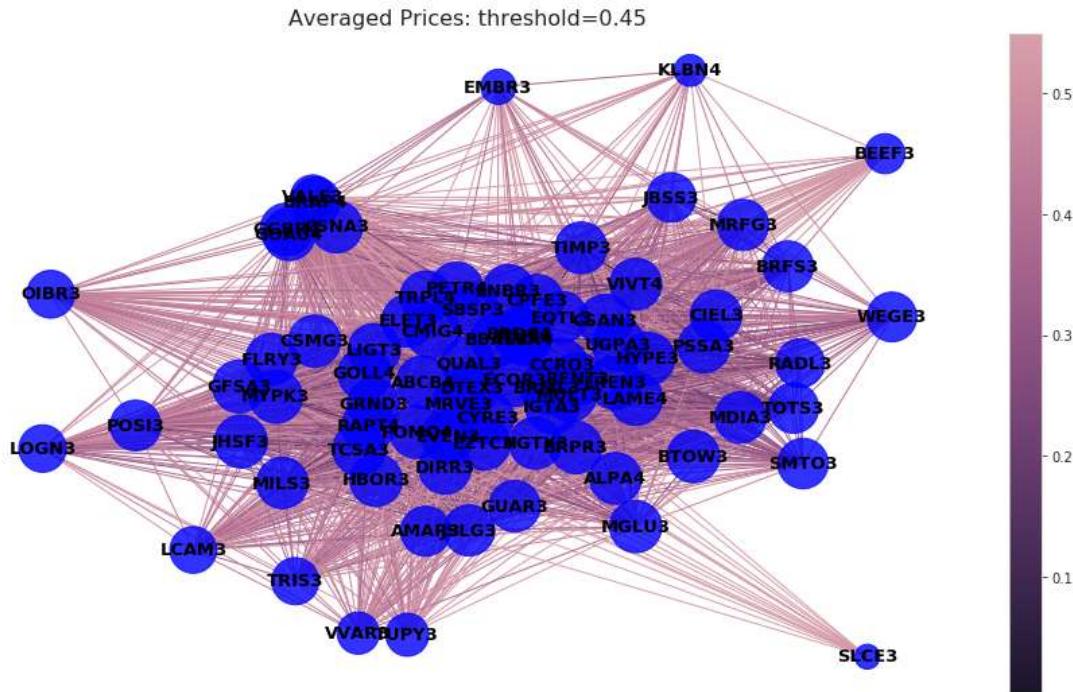
```
# plots the distance correlation network of the daily opening prices from 2006-2014
plt_corr_nx(H_master, title='Averaged Prices: threshold=0.325')
```



And now with a threshold of 0.45:

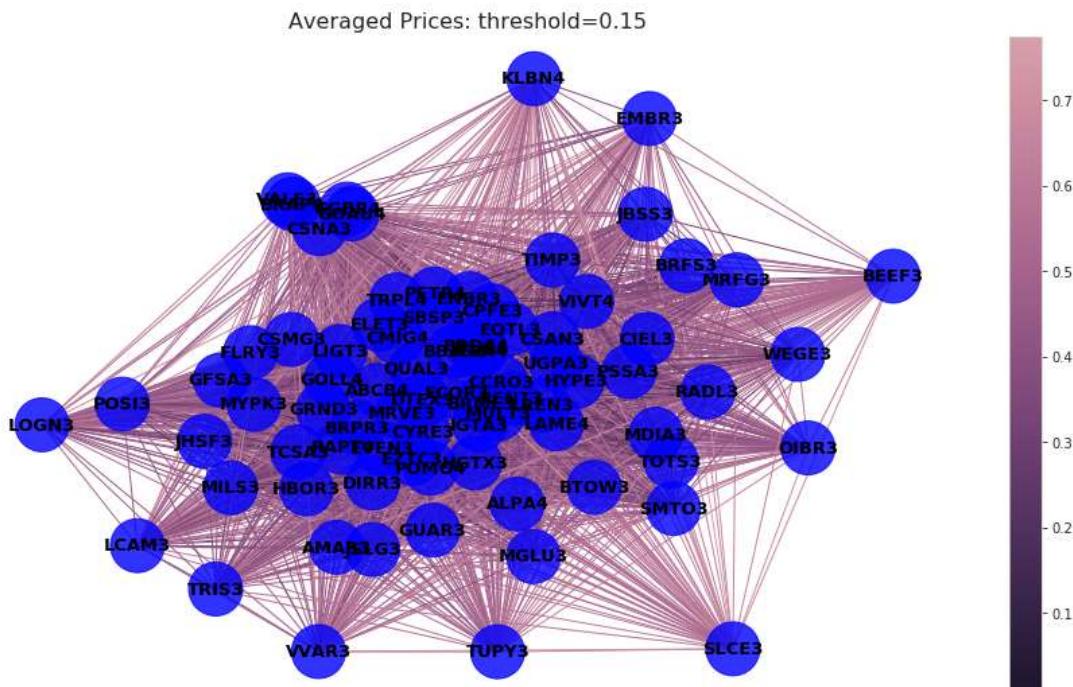
In [150]:

```
plt_corr_nx(build_corr_nx(df_dcor_master, corr_threshold=0.45), title='Averaged Prices: threshold=0.45')
```



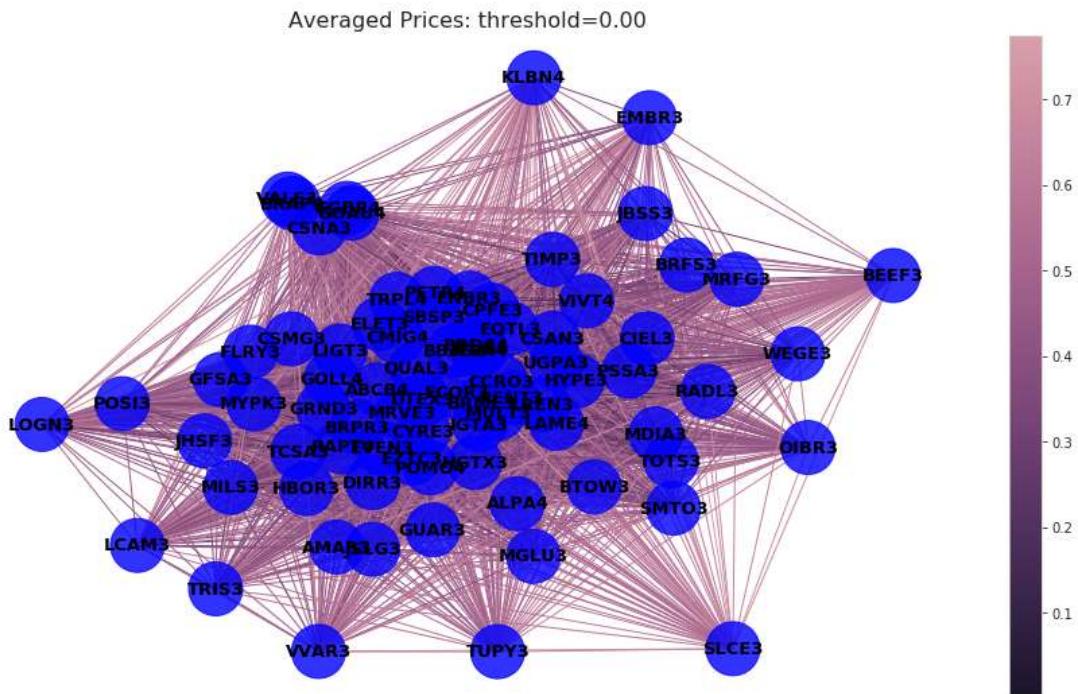
In [155]:

```
plt_corr_nx(build_corr_nx(df_dcor_master, corr_threshold=0.15), title='Averaged Prices: threshold=0.15')
```



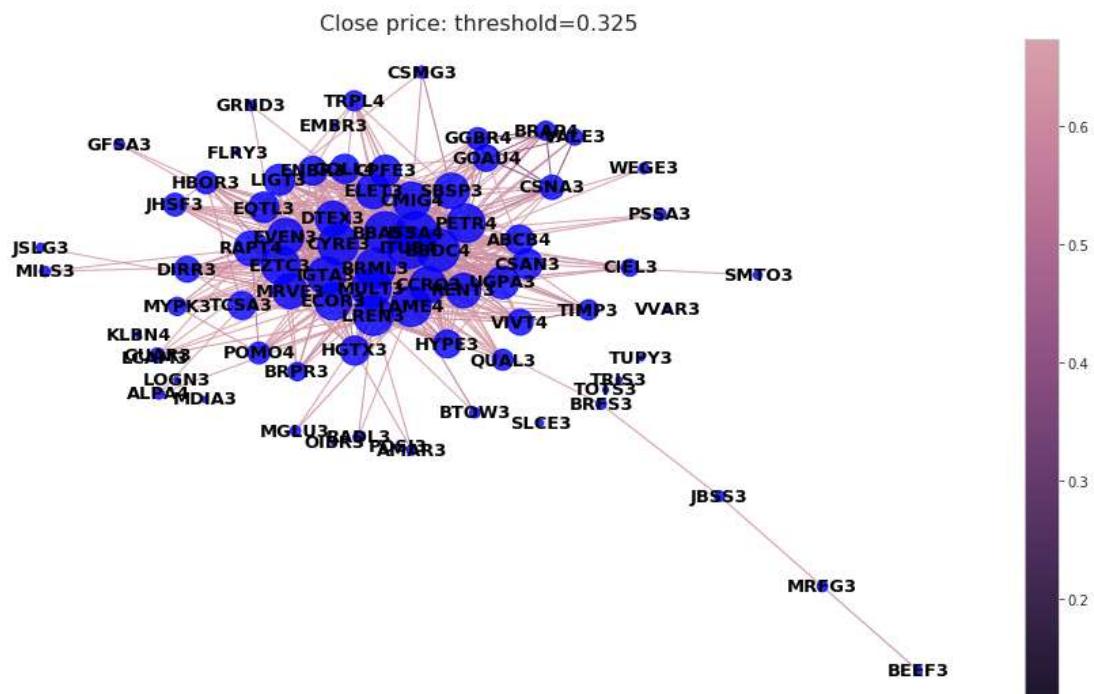
In [158]:

```
plt_corr_nx(build_corr_nx(df_dcor_master, corr_threshold=0.00), title='Averaged Prices: threshold=0.00')
```



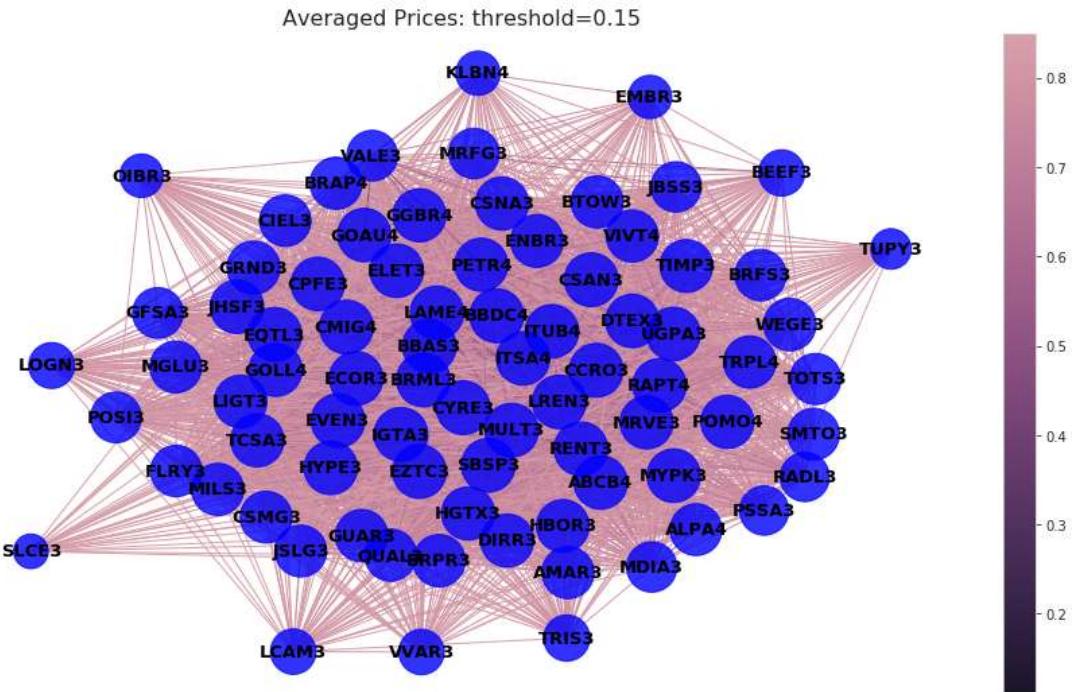
In [151]:

```
# plots the distance correlation network of the daily opening prices from 2006-2014
plt_corr_nx(H_close, title='Close price: threshold=0.325')
```



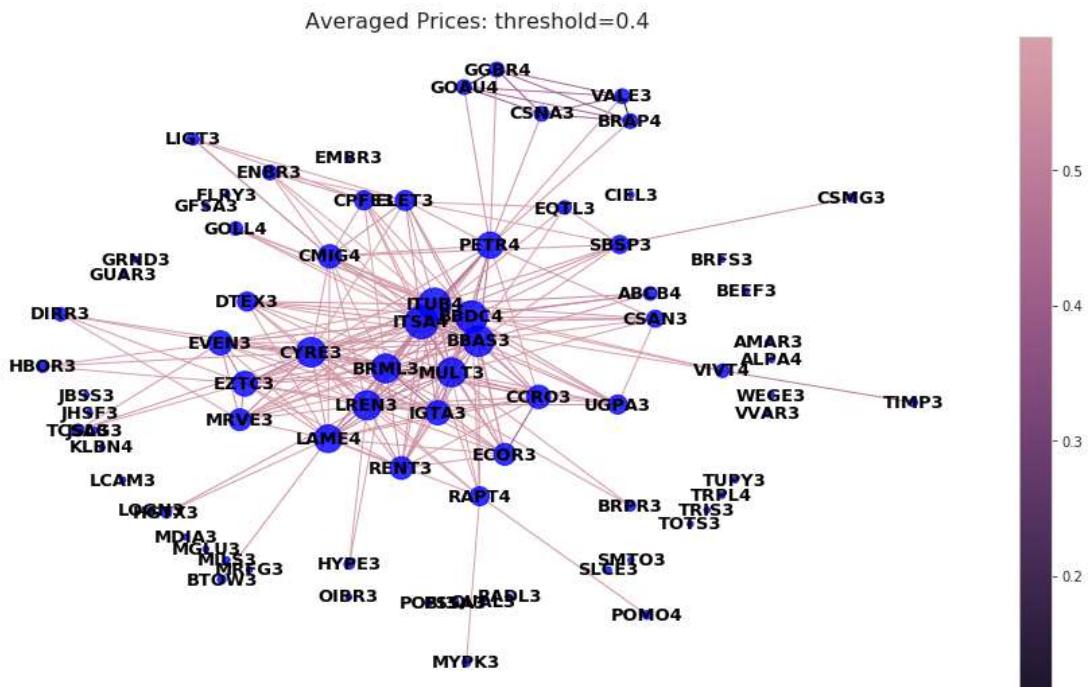
In [159]:

```
plt_corr_nx(build_corr_nx(df_dcor_list[0], corr_threshold=0.15), title='Averaged Prices: threshold=0.15')
```



In [165]:

```
plt_corr_nx(build_corr_nx(df_dcor_list[0], corr_threshold=0.4), title='Averaged Prices: threshold=0.4')
```



In the above visualizations, the sizes of the vertices are proportional to the number of connections they have. The colorbar to the right indicates the degree of dissimilarity (the distance) between the stocks. The larger the value (the lighter the color) the less similar the stocks are.

We compared the graphs for the averaged prices and the close price with different threshold values. We will perform our portfolio selection model with the close price and discard the averaged prices data because it seems to add artificial connections to the graph. We will use only one of the series (as usual we choose the close prices).

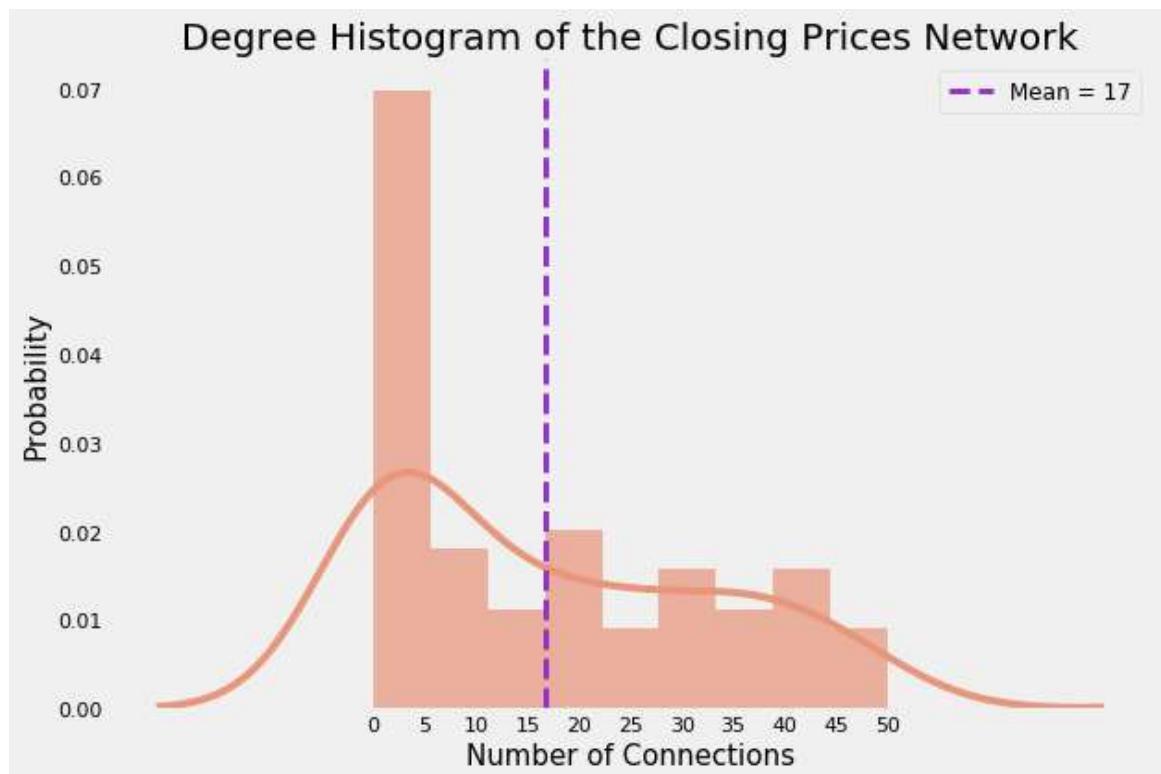
Also, by trying different values for the threshold we were able to identify the value of 0.4 to be of interest, because:

1. It allows us clearly visualize the network;
2. We obtain a highly connected network, but will have to remove a few stocks to obtain an irreducible graph.

Degree Histogram

In [188]:

```
# plots the degree histogram of the closing prices network
hist_plot(
    H_close,
    'Degree Histogram of the Closing Prices Network',
    bins=9,
    xticks=range(0, 51, 5)
)
```



Observations

- The degree distribution is right-skewed;
- Most of the network is connected to less than 15 nodes;
- The average node is connected to 21% of the network;
- The kernel density estimation is not a good fit;

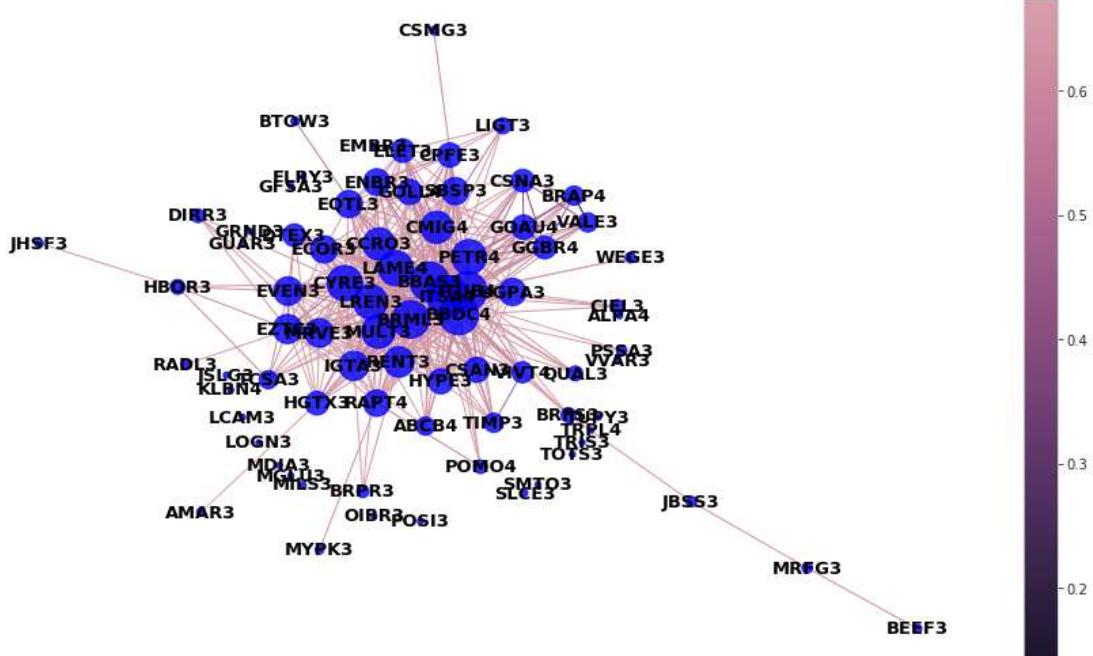
CHECK:

- By eyeballing the plot, the degrees appear to follow an *inverse power-law* distribution. (This would be consistent with the findings of [Tse, et al. \(2010\)](http://cktse.eie.polyu.edu.hk/pdf-paper/JoEF-1009.pdf) (<http://cktse.eie.polyu.edu.hk/pdf-paper/JoEF-1009.pdf>)).

In [186]:

```
plt_corr_nx(
    H_open,
    title='Distance Correlation Network of the Daily Opening Prices (train range)'
)
```

Distance Correlation Network of the Daily Opening Prices (train range)

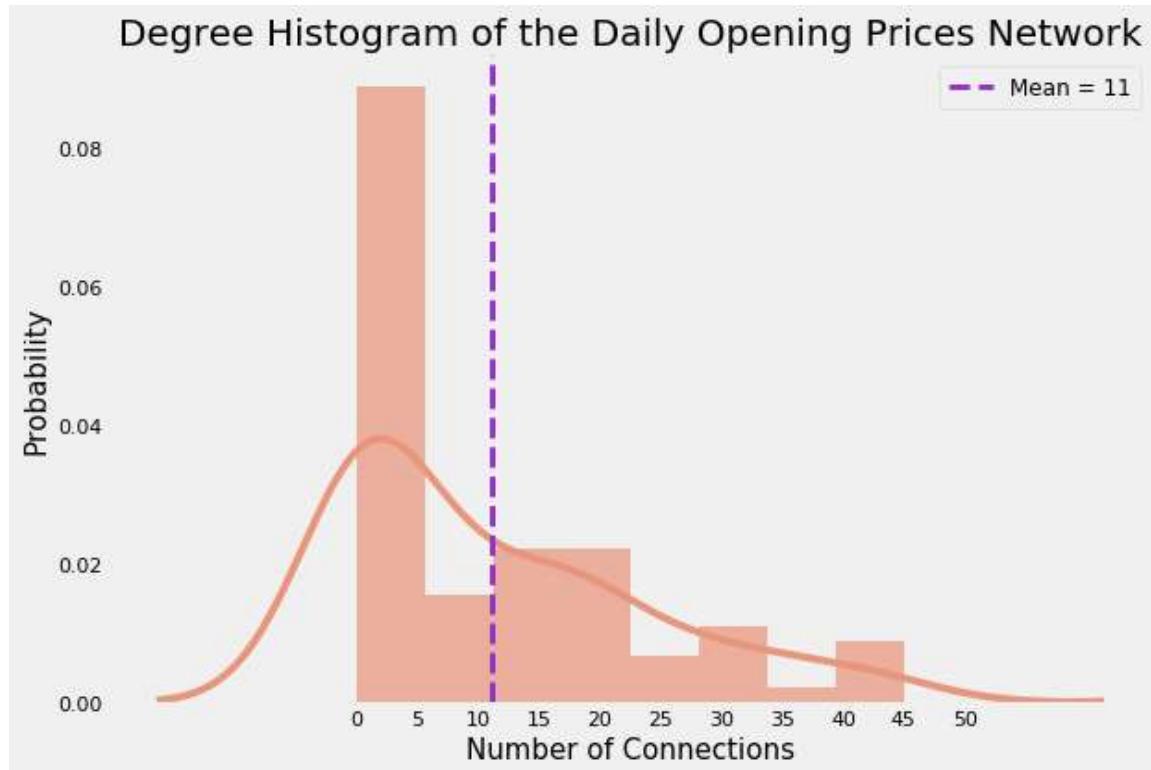


Observations

- ...

In [189]:

```
hist_plot(  
    H_open,  
    'Degree Histogram of the Daily Opening Prices Network',  
    bins=8,  
    xticks=range(0, 51, 5)  
)
```



Find non-communicating nodes

In [190]:

```
H_close_04 = build_corr_nx(df_dcor_list[0], corr_threshold=0.4)
zero_degree = []
nonzero_degree = []
for t, d in H_close_04.degree():
    if d == 0:
        zero_degree.append(t)
    else:
        nonzero_degree.append(t)

print(zero_degree)
print(len(zero_degree))
print(nonzero_degree)
print(len(nonzero_degree))

['ALPA4', 'AMAR3', 'BEEF3', 'BRFS3', 'CIEL3', 'EMBR3', 'FLRY3', 'GFS3',
'GRND3', 'GUAR3', 'JBSS3', 'JHSF3', 'JSLG3', 'KLBN4', 'LCAM3', 'LOGN3', 'M
DIA3', 'MGLU3', 'MILS3', 'MRFG3', 'OIBR3', 'POSI3', 'PSSA3', 'QUAL3', 'RAD
L3', 'SLCE3', 'SMT03', 'TOTS3', 'TRIS3', 'TRPL4', 'TUPY3', 'VVAR3', 'WEGE
3']
33
['ABCB4', 'BBAS3', 'BBDC4', 'BRAP4', 'BRML3', 'BRPR3', 'BTOW3', 'CCRO3',
'CMIG4', 'CPFE3', 'CSAN3', 'CSMG3', 'CSNA3', 'CYRE3', 'DIRR3', 'DTEX3', 'E
COR3', 'ELET3', 'ENBR3', 'EQTL3', 'EVEN3', 'EZTC3', 'GGBR4', 'GOAU4', 'GOL
L4', 'HBOR3', 'HGX3', 'HYPE3', 'IGTA3', 'ITSA4', 'ITUB4', 'LAME4', 'LIGHT
3', 'LREN3', 'MRVE3', 'MULT3', 'MYPK3', 'PETR4', 'POMO4', 'RAPT4', 'RENT
3', 'SBSP3', 'TCSA3', 'TIMP3', 'UGPA3', 'VALE3', 'VIVT4']
47
```

we found 33 nodes that are not communicating with the graph, and 47 remaining.

We remove the non-communicating nodes from the graph

In [191]:

```
H_close_04.remove_nodes_from(zero_degree)
nx.communicability_betweenness_centrality(H_close_04)
```

Out[191]:

```
{'ABCB4': 0.059702855162930044,
 'BBAS3': 0.6845025171511212,
 'BBDC4': 0.7182441949280415,
 'BRAP4': 0.016089130583318678,
 'BRML3': 0.6170057160035335,
 'BRPR3': 0.011557952534177492,
 'BTOW3': 0.0025418949694964834,
 'CCRO3': 0.4129272855082728,
 'CMIG4': 0.3320914844826067,
 'CPFE3': 0.14059070674890686,
 'CSAN3': 0.13461079881332477,
 'CSMG3': 0.0005022321239919119,
 'CSNA3': 0.016089130583318935,
 'CYRE3': 0.6328111710639369,
 'DIRR3': 0.02984031442078398,
 'DTEX3': 0.2396248804549056,
 'ECOR3': 0.3471676956280092,
 'ELET3': 0.18580917162079022,
 'ENBR3': 0.04583270919716699,
 'EQTL3': 0.027879160377364973,
 'EVEN3': 0.38688471074276504,
 'EZTC3': 0.42067481515748006,
 'GGBR4': 0.01608913058331914,
 'GOAU4': 0.016089130583319136,
 'GOLL4': 0.059702855162929926,
 'HBOR3': 0.018793299012399477,
 'HGTX3': 0.011127073282953096,
 'HYPE3': 0.011793664707415196,
 'IGTA3': 0.4593462503336171,
 'ITSA4': 0.739967298965135,
 'ITUB4': 0.7295800686445043,
 'LAME4': 0.5679859819184467,
 'LIGT3': 0.007817081343691058,
 'LREN3': 0.6226401599863867,
 'MRVE3': 0.32326766705609605,
 'MULT3': 0.6062196808405862,
 'MYPK3': 0.0008425775024474112,
 'PETR4': 0.4908137199414152,
 'POMO4': 0.0008425775024473136,
 'RAPT4': 0.22998214816242096,
 'RENT3': 0.38308909887428605,
 'SBSP3': 0.14258082888165796,
 'TCSA3': 0.01879329901239938,
 'TIMP3': 0.000266503000946313,
 'UGPA3': 0.20594832317471606,
 'VALE3': 0.016089130583318782,
 'VIVT4': 0.07679279170503697}
```

In [192]:

```
is_irreducible(H_close_04)
```

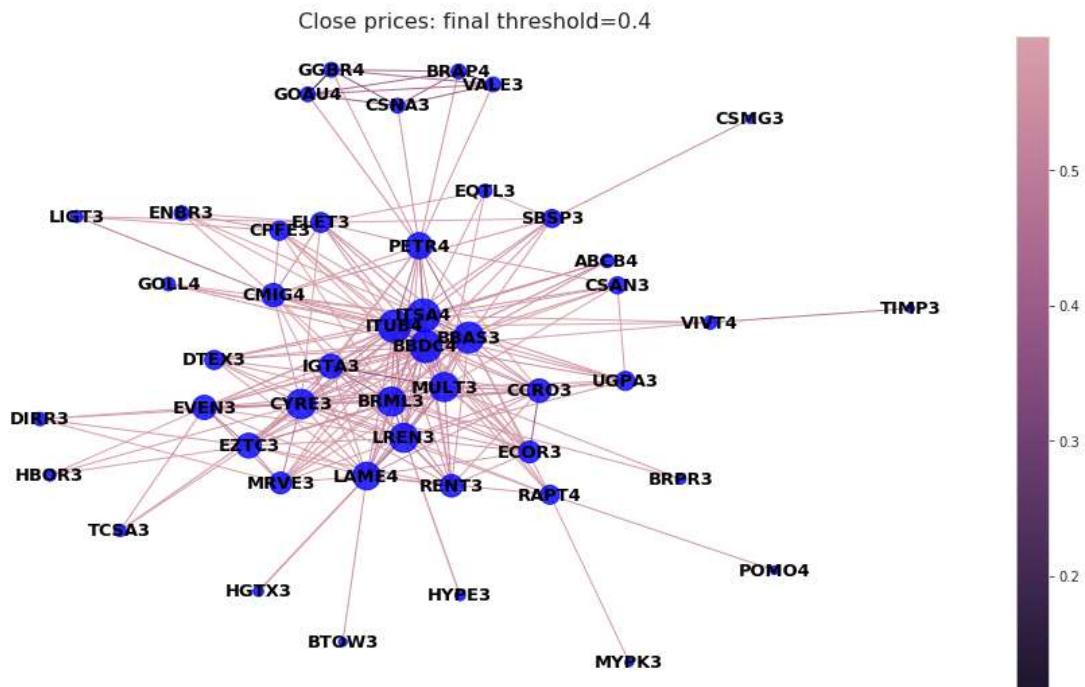
Out[192]:

True

Now we plot the resulting graph

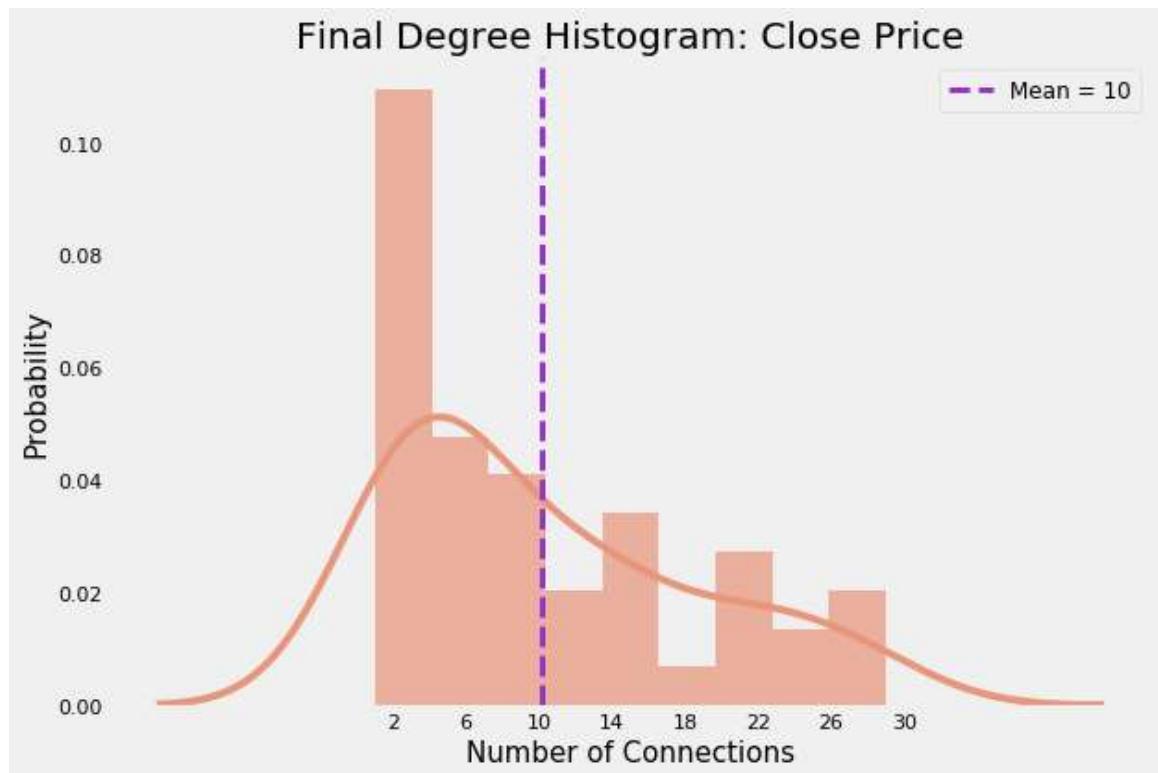
In [194]:

```
plt_corr_nx(  
    H_close_04,  
    title='Close prices: final threshold=0.4'  
)
```



In [198]:

```
hist_plot(  
    H_close_04,  
    'Final Degree Histogram: Close Price',  
    bins=9,  
    xticks=range(2, 31, 4)  
)
```



Saving the data

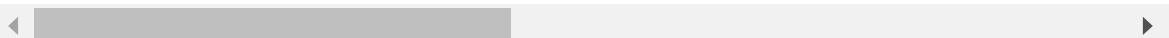
In [199]:

```
df_dcor_list[0].head()
```

Out[199]:

	ABCB4	ALPA4	AMAR3	BBAS3	BBDC4	BEEF3	BRAP4	BRFS3	B
ABCB4	1	0.258812	0.242263	0.464469	0.483814	0.162692	0.253581	0.218237	0.3
ALPA4	0.258812	1	0.251698	0.29269	0.298628	0.15601	0.165893	0.193373	0.3
AMAR3	0.242263	0.251698	1	0.294654	0.292318	0.155758	0.171539	0.187853	0.3
BBAS3	0.464469	0.29269	0.294654	1	0.70144	0.163379	0.352104	0.300505	0.4
BBDC4	0.483814	0.298628	0.292318	0.70144	1	0.169016	0.371825	0.318448	0.5

5 rows × 80 columns



In [200]:

```
# Save the non-zero degree list
with(open(r"data/selected_tickers.txt", "w")) as f:
    f.write(",".join(nonzero_degree))

# Save the close data frame
df_dcor_list[0].to_csv(r"data/close_prices_dcor.csv")
```

In []:

