Visualizing the stock market structure:

Learning a graph structure

We use sparse inverse covariance estimation to find which quotes are correlated conditionally on the others. Specifically, sparse inverse covariance gives us a graph that is a list of connection. For each symbol, the symbols that it is connected too are those useful to explain its fluctuations.

Clustering

We use clustering to group together quotes that behave similarly. Here, amongst the Various Clustering available in the scikit-learn, we use Affinity Propagation as it does not enforce equal-size clusters, and it can choose automatically the number of clusters from the data.

Note that this gives us a different indication than the graph, as the graph reflects conditional relations between variables, while the clustering reflects marginal properties: variables clustered together can be considered as having a similar impact at the level of the full stock market.

Embedding in 2D space

For visualization purposes, we need to lay out the different symbols on a 2D canvas. For this we use Manifold learning techniques to retrieve 2D embedding.

Visualization

The output of the 3 models is combined in a 2D graph where nodes represent the stocks

- cluster labels are used to define the color of the nodes
- the sparse covariance model is used to display the strength of the edges
- the 2D embedding is used to position the nodes in the plan

This example has a fair amount of visualization-related code, as visualization is crucial here to display the graph. One of the challenges is to position the labels minimizing overlap. For this we use a heuristic based on the direction of the nearest neighbor along each axis.

Graphical Analysis

We will be applying graphical analysis algorithm on the weekly stock returns data. The algorithm employs several unsupervised learning techniques and combines them together to visualize the stock returns data.

The algorithm takes the weekly returns of the stocks as input, and feeds it into three unsupervised learning techniques in order. The three techniques are:

Sparse Inverse Covariance Estimation:

It is used on the returns data, and it outputs a sparse matrix whose elements represent the conditional correlation between any two stock returns.

It provides information about which stocks are correlated conditionally on the others, and we can use this information to construct the graphical network.

The estimator we used in this graphical analysis example is called **Graphical Lasso**, which uses an l1 penalty to enforce sparsity on the precision matrix.

Sparse inverse covariance:

The matrix inverse of the covariance matrix, often called the precision matrix, is proportional to the partial correlation matrix. It gives the partial independence relationship. In other words, if two features are independent conditionally on the others, the corresponding coefficient in the precision matrix will be zero. This is why it makes sense to estimate a sparse precision matrix: the estimation of the covariance matrix is better conditioned by learning independence relations from the data. This is known as *covariance selection*.

In the small-samples situation, in which n_samples is on the order of n_features or smaller, sparse inverse covariance estimators tend to work better than shrunk covariance estimators. However, in the opposite situation, or for very correlated data, they can be numerically unstable. In addition, unlike shrinkage estimators, sparse estimators are able to recover off-diagonal structure.

The **GraphicalLasso** estimator uses an l1 penalty to enforce sparsity on the precision matrix: the higher its alpha parameter, the more sparse the precision matrix. The corresponding **GraphicalLassoCV** object uses cross-validation to automatically set the alpha parameter.

Affinity Propagation

- ***Affinity Propagation***, like K-Means, is a clustering method that organizes the returns into clusters. It does not enforce equal-size clusters, and it can choose automatically the number of clusters from the data.
- Stock returns that are clustered together can be considered as having a similar impact at the level of the full stock market.

Affinity Propagation

Affinity Propagation creates clusters by sending messages between pairs of samples until convergence.

A dataset is then described using a small number of exemplars, which are identified as those most representative of other samples. The messages sent between pairs represent the suitability for one sample to be the exemplar of the other, which is updated in response to the values from other pairs.

This updating happens iteratively until convergence, at which point the final exemplars are chosen, and hence the final clustering is given.

Affinity Propagation can be interesting as it chooses the number of clusters based on the data provided. For this purpose, the two important parameters are the *preference*, which controls how many exemplars are used, and the *damping factor* which damps the responsibility and availability messages to avoid numerical oscillations when updating these messages.

The main drawback of Affinity Propagation is its complexity. The algorithm has a time complexity of the order O (N2T), where N is the number of samples and T is the number of iterations until convergence. Further, the memory complexity is of the order O(N2) if a dense similarity matrix is used, but reducible if a sparse similarity matrix is used. This makes Affinity Propagation most appropriate for small to medium sized datasets.

MANIFOLD LEARNING

- ***Manifold Learning*** is a non-linear embedding method that projects a high-dimensional graph onto 2D plane. The 2D graph is then constructed by combining the results from the above techniques in the following way:
- Each node corresponds to a stock, with color being defined by the cluster label from Affinity Propagation.
- Each edge links two stocks.

The **strength** of the edge is defined by the sparse inverse covariance matrix (i.e. the precision matrix). The higher the value is, the more conditional correlated it is for the two stocks.

The Maniford 2D embedding is used to position the nodes on the plane.

The Manifold learning method we used in this graphical analysis example is called **Multidimensional scaling (MDS)**, which seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space.

Multidimensional scaling

Multidimensional scaling (**MDS**) seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space.

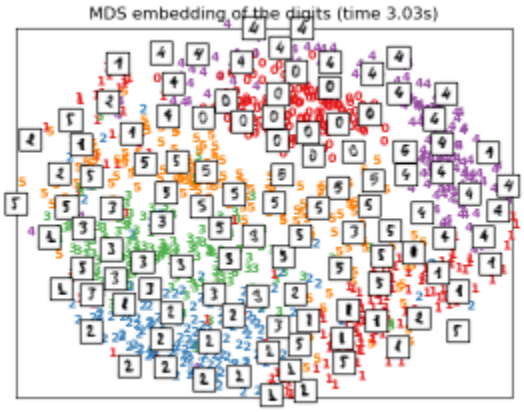
In general, **MDS** is a technique used for analyzing similarity or dissimilarity data. It attempts to model similarity or dissimilarity data as distances in a geometric spaces. The data can be ratings of similarity between objects, interaction frequencies of molecules, or trade indices between countries.

There exist two types of MDS algorithm: metric and non-metric.

In the scikit-learn, the class **MDS** implements both.

In Metric MDS, the input similarity matrix arises from a metric (and thus respects the triangular inequality), the distances between output two points are then set to be as close as possible to the similarity or dissimilarity data.

In the non-metric version, the algorithms will try to preserve the order of the distances, and hence seek for a monotonic relationship between the distances in the embedded space and the similarities/dissimilarities.



Let S be the similarity matrix, and X the coordinates of the n input points. Disparities $\hat{d}_{ij}$ are transformation of the similarities chosen in some optimal ways. The objective, called the stress, is then defined by $\sum_{i<j} d_{ij}(X) - \hat{d}_{ij}(X)$

2.2.8.1. Metric MDS

The simplest metric `MDS` model, called *absolute MDS*, disparities are defined by $\hat{d}_{ij} = S_{ij}$. With absolute MDS, the value S_{ij} should then correspond exactly to the distance between point i and j in the embedding point.

Most commonly, disparities are set to $\hat{d}_{ij} = bS_{ij}$.

2.2.8.2. Nonmetric MDS

Non metric `MDS` focuses on the ordination of the data. If $S_{ij} < S_{jk}$, then the embedding should enforce $d_{ij} < d_{jk}$. A simple algorithm to enforce that is to use a monotonic regression of d_{ij} on S_{ij}, yielding disparities $\hat{d}_{ij}$ in the same order as S_{ij}.

A trivial solution to this problem is to set all the points on the origin. In order to avoid that, the disparities $\hat{d}_{ij}$ are normalized.

Applying of Graphical Analysis using Python programming:

Here we have prepared one dataset which has info about all the firms employed for modeling and the sector that those firms belong to.

```
[ ] file_name = 'CleanedData_Weekly.xlsx'
    sheet_name = 'Firms Info'
    firms_info = pd.read_excel(file_name, sheet_name, index_col=0)
```

```
    firms_info.head(5)
```

Ticker	Name	Sector
FRC	First Republic	Bank
PNC	PNC Financial	Bank
USB	US Bancorp	Bank
JPM	JPMorgan Chase	Bank
BAC	Bank of America	Bank

```
[ ] # Get and print the Sector Information
    Sectors = firms_info.Sector.unique()
    print(Sectors)
```

```
['Bank' 'Health' 'Energy' 'Tech' 'Market Index']
```

Then we prepared other dataset which contains info about weakly stocks returns of these firms.

```
# Load Stock Return dataset
file_name = 'CleanedData_Weekly.xlsx'
sheet_name = 'Stock Returns'
df = pd.read_excel(file_name, sheet_name, index_col=0)
df.index=pd.to_datetime(df.index)
data = df.copy()
```

```
data.head(5)
```

date	FRC	PNC	USB	JPM	BAC	C	RY	WFC
2000-01-09	NaN	-0.032304	-0.049869	-0.058139	-0.029888	-0.030302	-0.042493	-0.032457
2000-01-16	NaN	0.097264	0.019338	0.015465	0.034659	0.074074	-0.017752	0.065494
2000-01-23	NaN	-0.061498	-0.094850	-0.012690	-0.093051	-0.026939	-0.035181	-0.121438
2000-01-30	NaN	0.045583	0.005989	0.051414	0.001367	-0.005537	0.068037	0.058020
2000-02-06	NaN	-0.028609	0.014881	0.066015	-0.008197	-0.021727	-0.054815	0.000774

✓ 13s completed at 6:54 PM

Specifications:

- We enter some specific dates in the "**start date**" and "**end date**" to specify the start and end date.
- By default (i.e. **Sectors chosen** = []), ALL sectors will be included in the graphical analysis.
- If certain sectors are entered in the "**Sectors chosen**" option, then only those sectors will be examined in the graphical network analysis.

  e.g.: Sectors chosen = ['Bank','Tech'] =➔only the Bank and Tech sectors will be examined.

- By default (i.e. **drop firm** = []), ALL firms in the given sectors (specified in "**Sectors chosen**") will be included in the graphical analysis.
- If certain firms are entered in the "**drop firm**" option, then only those firms will NOT be examined in the graphical network analysis.

  e.g.: drop firm = ['FRC','RY']=➔'FRC' and 'RY' will not be examined.

OUTPUT1:

Applying Graphical analysis putting all specification as gien above,we get the following Output:

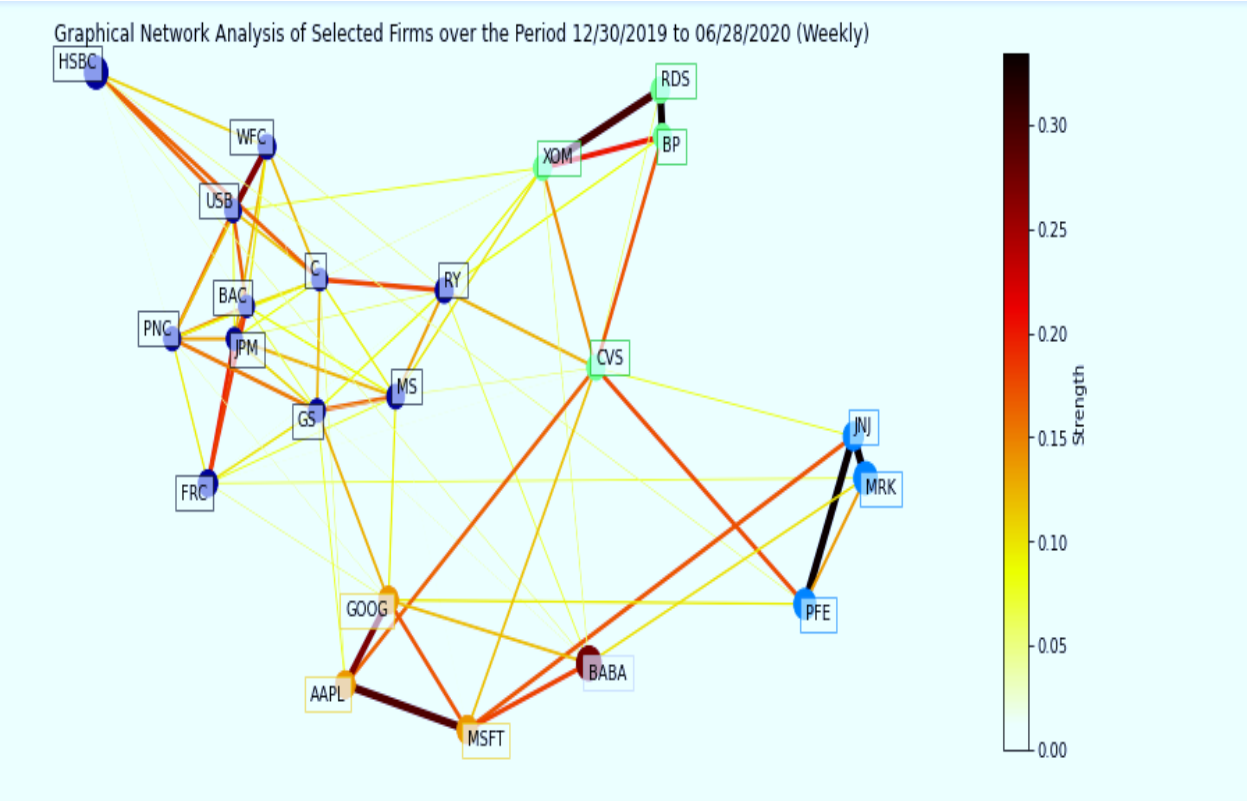
```
graphicalAnalysis(data, start_date, end_date, Sectors_chosen, drop_firm)

Results over the time period  2020-01-01  to  2020-06-28 :

Sectors choosen in the Graphical Analysis are:
['Bank', 'Health', 'Energy', 'Tech']

Number of firms examined: 22
Cluster 1: FRC, PNC, USB, JPM, BAC, C, RY, WFC, GS, MS, HSBC
Cluster 2: JNJ, PFE, MRK
Cluster 3: XOM, RDS, CVS, BP
Cluster 4: AAPL, GOOG, MSFT
Cluster 5: BABA
```

Plotting the graphical network graph we get the following Graph:



We can Explore Graphical Analysis further to understand how the select firms related with respect to all other firms or how one particular selected firm is located with respect to others. There can be multiple ways we can perform the factor Analysis of our stock portfolios using Graphical Analysis.