Leveraging Graphical Structures in the Corporate World

Ryan Harty

4/21/2021

Leveraging Graphical Structures in the Corporate World

Authors: Ryan Harty, Ernest Kufuor, Aline Niyonsaba, Eric Ngabonzima

Advisors: Dr. Moise Busogi, Ph.D., Dr. Brian Junker, Ph.D.

Client: JP Morgan

Email Addresses: rjharty@andrew.cmu.edu (mailto:rjharty@andrew.cmu.edu), ekufor@andrew.cmu.edu (mailto:ekufor@andrew.cmu.edu), aniyonsa@andrew.cmu.edu (mailto:aniyonsa@andrew.cmu.edu), engabonz@andrew.cmu.edu (mailto:engabonz@andrew.cmu.edu)

Abstract:

This paper seeks to explore the development of graphical structures in order to build and analyze corporate relationships using a large amount of data. Namely, we are interested in community detection and information prediction . We have collected both news article data and financial performance data on 30 companies and the financial index fund they all belong to (the Dow Jones Industrial Average) for the purposes of graph creation. We have performed named entity recognition on our news article data to identify company co-mentions in news articles, mapped relationships between companies as graphs, with companies as nodes and relationships as edges, combined graphs by taking subsets of the edges in the graphs we have built, and predicted the change in stock price for these companies using node attribute predictions. We have succeeded in visualizing the graphs we have created, the communities we have detected, and the analysis of these graphs and developed a model for node attribute prediction that can be replicated. Finally, we have evidence that community detection possible and provides useful information on relationships between companies.

Introduction:

There is an abundance of data in the Financial Services industry- there may be quantitative data describing a firm's individual financial performance, qualitative data describing a company's relationships to other companies both similar and dissimilar in industry type, and other data sources on anything that could affect a company's financial well-being. Different datasets with different levels of structure are often used independently in machine learning tasks such as modeling and prediction, even though much more could be learned from building an ensemble model that can learn to account for interactions between different data sources. In this project, we are seeking to construct knowledge graphs, which are networks of relationships between publicly-traded companies that show how companies are related in areas such as stock price and co-mentions in news articles.

This is based on graph theory, where a graph G is defined by a set of edges E and nodes (or vertices) V, where nodes are connected to other nodes by edges formed between them [1]. In the knowledge graph we'd like to build out of news article co-mentions, our nodes are going to be different companies, and an edge between two nodes will represent a high frequency of news article co-mentions between them. Since co-mentions are nondirectional (both companies are treated equally in the co-mention process), our edge set will be non-directed in this case, which means that an edge reflects an equal connection for both nodes it joins.

These different knowledge graphs will each provide us with key insights into business relationships between companies, and combining them however possible should allow us to perform more accurate, informative clusters of similar companies from which we can infer different features of each company in a cluster. The applications for this are very open-ended, but some of the main ways that financial companies could benefit from this analysis are through enhancement of investment strategy and improvement in anomaly detection of companies that do not have strong relationships to many others. We have set our research questions as follows:

Research Question 1: What kinds of communities can be detected among different companies given financial and news article data?

Research Question 2: How can we use graphical structures governing relationships between different companies to determine future information about those companies?

Data:

There are two main datasets that we are incorporating here. The first is news article co-mentions data, where the goal is to identify which companies are mentioned together frequently in news articles as a basis for drawing edges in a graph. A single news article co-mention for two specified companies is considered to be a news article that mentions both companies. When building our news article dataset, we decided to use the GoogleNews python package to pull in news articles from GoogleNews, which itself pulls from several different news sources in compiling online news for those who are interested in reading it [2]. One of the main reasons for this is that we wanted to bring in news articles from numerous different sources to avoid single-publication bias affecting the relationships we notice, and another reason is the convenience with which several thousand news articles can be pulled at a time.

For this project, we decided to focus on the Dow Jones Industrial Average (DJIA), a stock market index that takes into account the stock performance of the largest 30 publicly-traded companies in the United States. The companies in our dataset were therefore set as the thirty companies most recently found in the DJIA, as well as the DJIA itself (as it as a stock price and news article mentions), giving us 31 entities in total. The entities, referred to as companies, are as follows: American Express, Apple, Amgen, Boeing, Caterpillar, Salesforce, Cisco, Chevron, Disney, Dow Chemical, Dow Jones Industrial Average, Goldman Sachs, Home Depot, Honeywell, International Business Machines, Intel, Johnson & Johnson, JPMorgan, Coca-Cola, McDonalds, 3M, Merck, Microsoft, Nike, Proctor & Gamble, Travelers, UnitedHealth, Visa, Verizon, Walgreens, and Walmart. We set this small cap on companies to build an interpretable graph with a visually-appealing number of individually-important nodes, and to simplify and expedite downstream data-processing tasks. Table 1 displays the news sources that GoogleNews pulled from in creating our article database.

	media	count
108	Fortune	37
287	The Motley Fool	33
304	TheStreet	32
224	Quartz	29
161	MarketWatch	27
218	Profit Confidential	22
277	The Guardian	21
48	Business Wire	20
107	Forbes	19
56	CNBC	18
233	Reuters	16
183	Nasdaq	15
330	Wall Street Journal	15
252	Smarter Analyst	13
313	USA Today	12
44	Business - Insider	12
199	PCMag	11
102	FierceBiotech	11
281	The Indian Express	11
180	NPR	11

Table 1: Article Sources for News Article Data

We conducted a GoogleNews search on the stock ticker of each company in question, and returned the 40 most popular news articles mentioning that company in each year from 2011-2020, and after removing duplicate articles we wound up with about 12000 news articles available for further processing, each with information on the date of the article, the title, the link to the article, and the full text content of the article. This was our news article dataset which was used, after some more processsing of article content, to establish news article co-mentions data for the purposes of creating a knowledge graph.

Our second main dataset was in terms of stock price correlations and transaction volume correlations between publicly-traded companies. The goal of this dataset was to build a knowledge graph using more quantitative financial relationship data rather than a graph constructed on news article co-mentions. To do this, we pulled stock price and transaction volume data from Yahoo Finance data, with our dataset also ranging from 2011-2020 and utilizing the same 30 companies used in the first dataset (as well as the DJIA index itself, which we treated as a company for research purposes). Both the stock price and transaction volume were calculated for each business day in the time period, adding up to about 264 days' worth of data per year per company. The recorded stock price and transaction volume were those measured at the close of business each day, giving us a time series dataset of each of these financial variables over time for each company. Overall, each of the companies had about 2600 data points on stock price and volume available, and with these data points we were able to calculate the change in price for each company on each day as well. Figure 1 shows the scatterplot for the correlation between JP Morgan and Goldman Sachs, where we see a moderate positive correlation indicating that stock prices at JP Morgan have some relationship with those at Goldman Sachs.

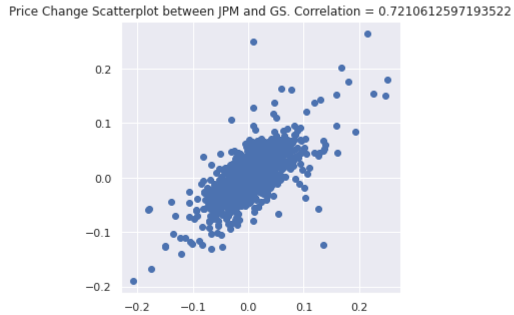


Figure 1: Correlation Between JP Morgan and Goldman Sachs

Methods:

One very important step in our process regarding our first dataset was to conduct Named Entity Recognition on the content of the articles we pulled in order to generate a list of co-mentions from our list of articles. The process here was to search the content of each article for entities, which in our case were companies represented by their stock ticker symbols. For each article, there was guaranteed to be at least one entity of interest since our news articles were pulled based on the companies we were interested in, but given that business articles usually discuss competition there were often mentions of competitors in a given company's articles. We used the flair package in python to perform Named Entity Recognition on the article content, since the flair package is a recently-developed solution that outperforms many older solutions for this process by utilizing a neural language model to assign tags to text data and learn which words in an article count as entities [3]. After performing this Named Entity Recognition for each article, we removed corporate entities that were not listed in our 31 entities of interest (the 30 companies currently in the DJIA, plus the DJIA itself) so that our graphs would be visually interpretable. The next step was to take the entities found in each article and create a list of co-mentions for each pair of entities found in the article; so, if 6 entities were found in an article, there would be 15 different co-mentions returned in order to connect all the entities found in that article to each other. In Figure 2 below, a frequency heat map is shown to show the number of co-mentions between different companies in our dataset.

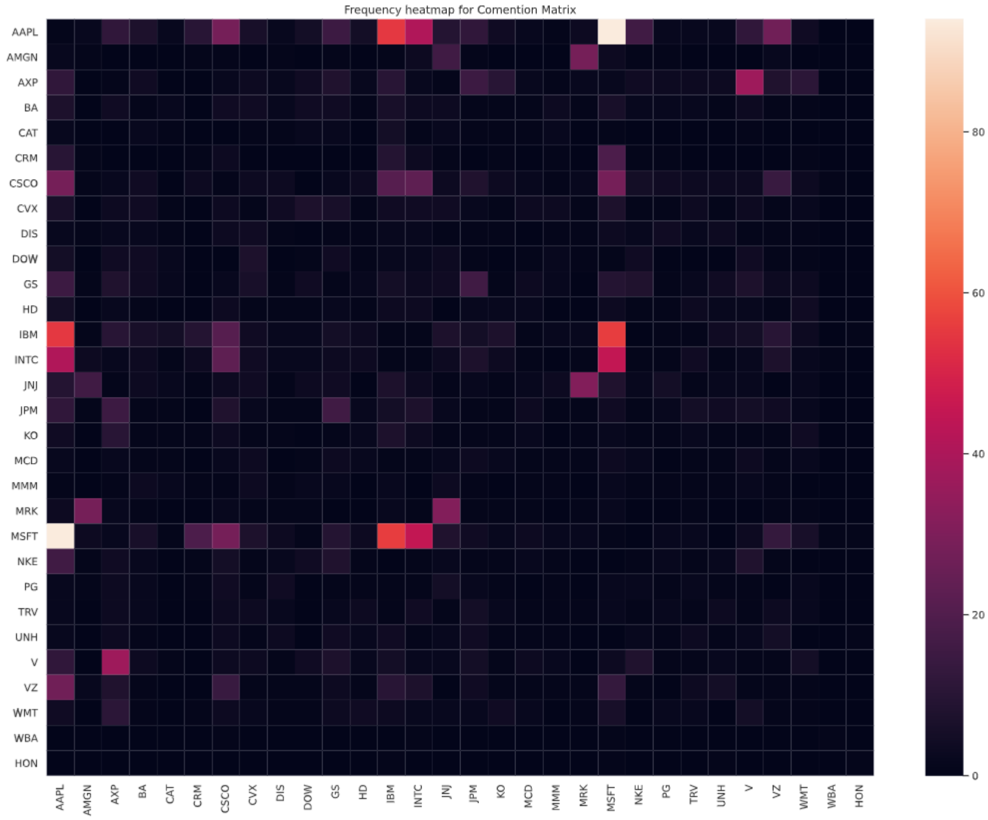


Figure 2: News Article Co-Mention Heat Map

After standardizing the co-mentions so that there were not separate co-mentions for a company and its stock ticker, the large list of co-mentions (about 3000 co-mentions with the two companies, the date, and the link to the article attached) was used for knowledge graph creation. Table 2 shows an extract of the database containing these co-mentions before any removal of duplicate and non-interest entities was applied, to show what an individual unit of data in this graph looks like and the information stored in each feature.

	from	to	title	date
0	AXP	American Express	Analysts Remain Positive on Starbucks Corporat...	Jan 22, 2016
1	AXP	American Express Company	Analysts Remain Positive on Starbucks Corporat...	Jan 22, 2016
2	AXP	Costco	Analysts Remain Positive on Starbucks Corporat...	Jan 22, 2016
3	AXP	Deutsche Bank	Analysts Remain Positive on Starbucks Corporat...	Jan 22, 2016
4	AXP	MarriottStarwood	Analysts Remain Positive on Starbucks Corporat...	Jan 22, 2016
...	...	...	...	...
73	USDA	Walmart	How Aldi is beating Walmart in the grocery aisle	Mar 29, 2016
74	USDA	WillardBishop	How Aldi is beating Walmart in the grocery aisle	Mar 29, 2016
75	WMT	Walmart	How Aldi is beating Walmart in the grocery aisle	Mar 29, 2016
76	WMT	WillardBishop	How Aldi is beating Walmart in the grocery aisle	Mar 29, 2016
77	Walmart	WillardBishop	How Aldi is beating Walmart in the grocery aisle	Mar 29, 2016

84760 rows × 4 columns

Table 2: Example Data for News Article Co-Mentions

We conducted Named Entity Recognition on the content of each article, a topic that will be more fully described in our Methods section, in order to establish a list of co-mentions between companies (with identical co-mentions allowed as long as they came from different articles). We then summed the identical co-mentions to develop a list of co-mention frequencies- the number of times each pair of companies had been mentioned together in the same article. This list of co-mention frequencies was ultimately stored as a 31 x 31 matrix, with each of the 31 rows and columns corresponding to a single company node (each company was deemed to have 0 co-mentions with itself for ease of usage, so the diagonal of the matrix was 0). Finally, when any two companies had a co-mention frequency greater than 6 (chosen as the cutoff to allow a visually-interpretable graph), an edge was drawn between the nodes representing those two companies in the graph that was drawn.

We wanted to create knowledge graphs creation for both the news article co-mention and financial data correlation datasets, and we decided to use the NetworkX package for this [4]. After completing Named Entity Recognition on the first dataset, we had a list of about 3000 instances of co-mentions in news articles between our 31 entities of interest. From there, we summed identical co-mentions to develop a list of co-mention frequencies- the number of times each pair of companies had been mentioned together across our entire dataset of articles. This list of co-mention frequencies was ultimately stored as a 31 x 31 matrix, with each of the 31 rows and columns corresponding to a single company node (each company was deemed to have 0 co-mentions with itself for ease of usage, so the diagonal of the matrix was 0). Finally, when any two companies had a co-mention frequency greater than 6 (chosen as the cutoff to allow a visually-interpretable graph), an edge was drawn between the nodes representing those two companies in the graph that was drawn. This created our first knowledge graph.

To create the knowledge graphs for the financial dataset, we began with the stock price and transaction volume for each business entity of interest. We calculated both the correlation in price change and the correlation in transaction volume for each pair of companies over the time period being measured, giving us two 31 x 31 matrices of correlation coefficients, one for price changes and one for transaction volume. These 31 x 31 correlation matrices are how we created our edges for this dataset- if you assign a number to each corporate entity we measured, then each entry in the matrix would reference one company by its row index and one company by its column index, so each of the calculated correlation coefficients became an entry in the corresponding graph. For each pair of company nodes in the graph, we drew an edge between the nodes if there was a moderate (0.40-0.60) correlation between the companies for both price changes and transaction volume, or a strong correlation (>0.60) for either of the metrics. These metrics were developed through trial and error and chosen for their ability to provide a structure where we did not have an abundance of connections between nodes, nor a sparsely-connected graph which would fare poorly in later stages of this project. These cutoffs could easily be tuned using machine learning and a validation set if this project had a larger time horizon. The 31 x 31 matrix for price changes used to create one of our knowledge graphs is shown in Figure 3 to aid in comprehension.

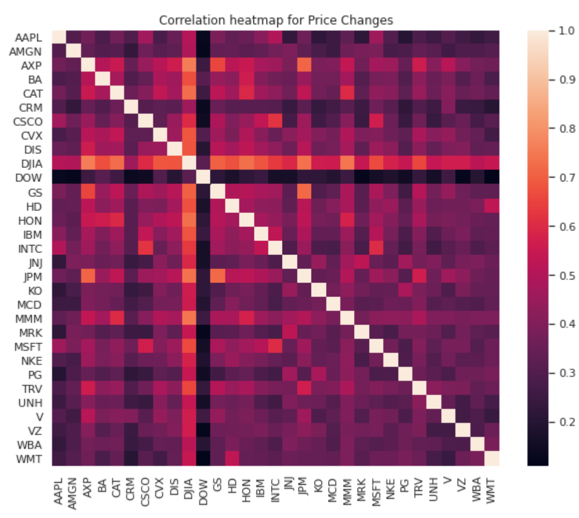


Figure 3: Price Change and Volume Correlation Matrices

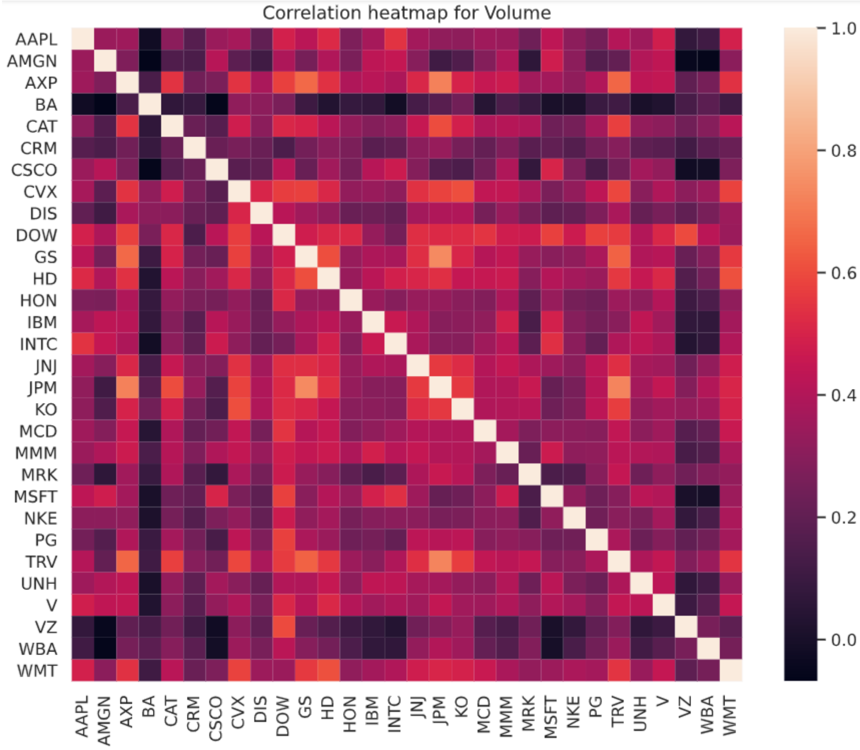


Figure 3: Price Change and Volume Correlation Matrices

Three more methods are currently being developed for this research project. The first is graph combination, where multiple graphs are fused through the use of edge regulation metrics or development of a multiplex graph. By using edge regulation metrics, two graphs with identical nodes can be combined into one graph by strictly delineating the requirements for an edge to be formed between two nodes. Since our knowledge graphs rely on news article co-mentions and financial correlations, we are currently testing different metrics for edge regulation to ensure the combined graph that we form will perform as well as possible in downstream community detection tasks. This part of our methods will be much better fleshed-out when we finish developing community detection methods and their evaluation tasks. Multiplex graphs, as shown in Figure 4, are singular graphs which combine multiple knowledge graphs by joining common nodes in the two graphs, and the edges connecting the nodes in separate graphs are an additional part of the graphical structure. The multiplex graph construction is also currently in development, and we are weighing whether it will be better suited to a visual enhancer for our project or a viable solution for community detection and prediction. A sample multiplex graph we have constructed is shown below:

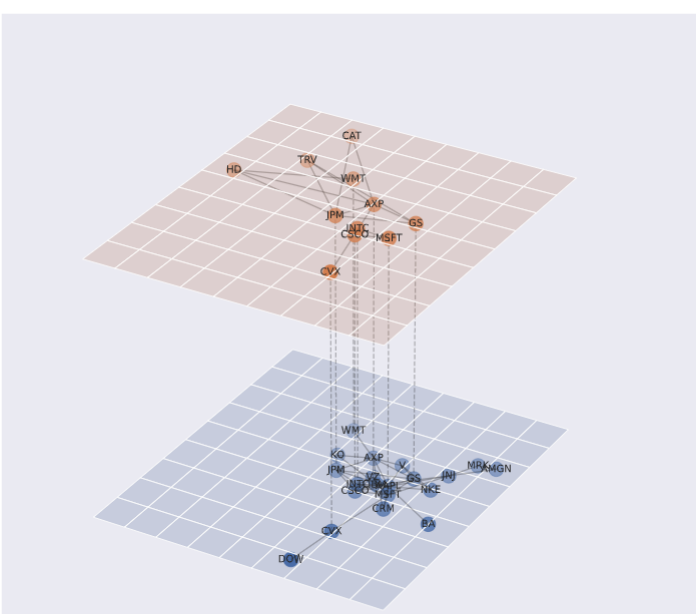


Figure 4: Sample Multiplex Graph

The second method mentioned above is community detection, where communities of nodes are formed from a knowledge graph to give us insight into which nodes are more connected than others. We detected communities by searching for groups of nodes within a graph which all had at least n edges among the nodes selected. After evaluation of which values of n would provide us with useful cliques for detecting communities, we then split the graph into the k cliques that result from the choice n. From there, we can evaluate the centrality of nodes in the graph, which is typically done by taking all multi-edge paths between nodes in the graphical structure and finding which nodes show up often as intermediate steps in connecting multiple nodes. This way, we identify nodes which have many connections to other nodes, as well as nodes which serve a role in connecting nodes that are not directly connected to each other. The formal methodology used is eigenvector centrality, which assigns scores to nodes using the eigenvectors of the adjacency matrix which defines the edge relationships in the graph and finds central nodes based on their connectivity to other central nodes.

The final method of interest here is node attribute prediction, where we take information about a node, combine it with structural information about relationships between companies built into the graph we have built, and use it to prediction future information about that node. In conducting node attribute prediction, we defined an experimental structure for evaluating whether the graphical structures we have available are useful in predicting information about the nodes in the graph. For the experiment, we wanted to see whether we could find a way to consistently take graphical data and predict something about the nodes of the graph itself, and this involved building many graphs for each company using the data at our disposal. In our experimental structure, we first divided both datasets at our disposal into 3-month windows (also known as business quarters) and developed both news and financial data graphs for each company using only the news articles and financial data from the chosen quarter. As both of our datasets run from 2011-2019, we wound up with 35 data windows for each of the 31 companies after excluding Q4 of 2019 from the dataset to improve code organization at a very small data opportunity cost. Since any successful implementation of a node attribute prediction model would likely involve predicting future information from past information, our training set was data from 2011-2017 (28 windows x 31 companies = 868 sets of graphs) and our testing set was data from 2018-2019 (7 windows x 31 companies = 217 sets of graphs). Each set of graphs contained the financial graph, the news article graph, a combined graph containing all the edges (as well as all the nodes) from the financial graph and the news article graph, a combined graph containing only the edges found in both the financial graph and the news article graph, and a baseline graph that connects every pair of nodes with an edge (uninformed by data). Let’s refer to these types of graphs as the financial graph, the news article graph, the all edges graph, the common edges graph, and the fully connected graph, respectively.

Our data was divided in this way to assist us in building a model for node attribute prediction. Since we had graphical structures that needed to be processed by our model, we decided on using a graph convolutional network in order to achieve this. The graph convolutional network converts the edge structure of a graph into a set of features for each node, so for each company in a given business quarter, its connections to other companies in our study (either in news or financial relationships) are encoded as a series of different values that can be uniquely identified as a set of edges between companies. While there is more than one way to achieve the stated conversion of edges into features, our team chose GCNConv in python over methods such as Node2Vec because of GCNConv’s linear scaling in terms of edges and ability to be quickly translated into a neural network for accurate prediction [5]. GCNConv allowed us to take our graphs constructed with the networkx package, and by simply transferring them to a different representation using the pytorch package, we could easily develop a neural network that could be trained in order to improve its prediction accuracy [6]. Furthermore, in order to visualize how our neural network was transforming the data, we utilized t-distributed stochastic neighborhood embeddings, which use standard distance metrics to compare how close high-dimensional data points are in their full-rank representations, as well as in low-rank approximations [7].

For our node attribute prediction task, we decided to measure whether a company's average daily stock price improved from the current business quarter to the next. If the company's average daily stock price increased from quarter to quarter, our binary response variable was encoded as a 1, and if it decreased it was encoded as a 0. We attached this value to each training and testing example in our data, where a single example is one of the sets of graphs mentioned earlier. Quarterly stock price change was the only node attribute measured in this way due to time constraints of the project, but in order to help in prediction, the stock price time series from the current quarter was added as a feature to each training and testing example as well. When it came to building a neural network using GCNConv, our model was trained on the training examples for only a specific type of graph (news article graph, common edges graph, etc.) in order to compare how different types of graphs performed against one another in node attribute prediction for our defined task [8]. The metric used for comparison will be accuracy on the testing set, comparing the binary predictions of our model (on whether stock price will increase or decrease quarter to quarter) to the historical labels of how stock price actually behaved for a given company in a certain quarter for instances not used to train the model. In addition to the types of graphs discussed above, another baseline for prediction accuracy will be the accuracy of a basic probabilitistic classifier, which uses the label proportions of the training set to guess at the labels for the testing set without learning anything about why these labels are assigned.

Results:

We have been able to build both news article and financial data graphs, and visualize them as well. Shown in Figure 5 is the visualization of a news article graph created with a selection of the entities of interest, using stricter edge criteria than normal in order to show the graph more clearly in this stage. The letters on each dot correspond to a company's stock ticker, so MSFT corresponds to Microsoft, and CAT to Caterpillar, for example.

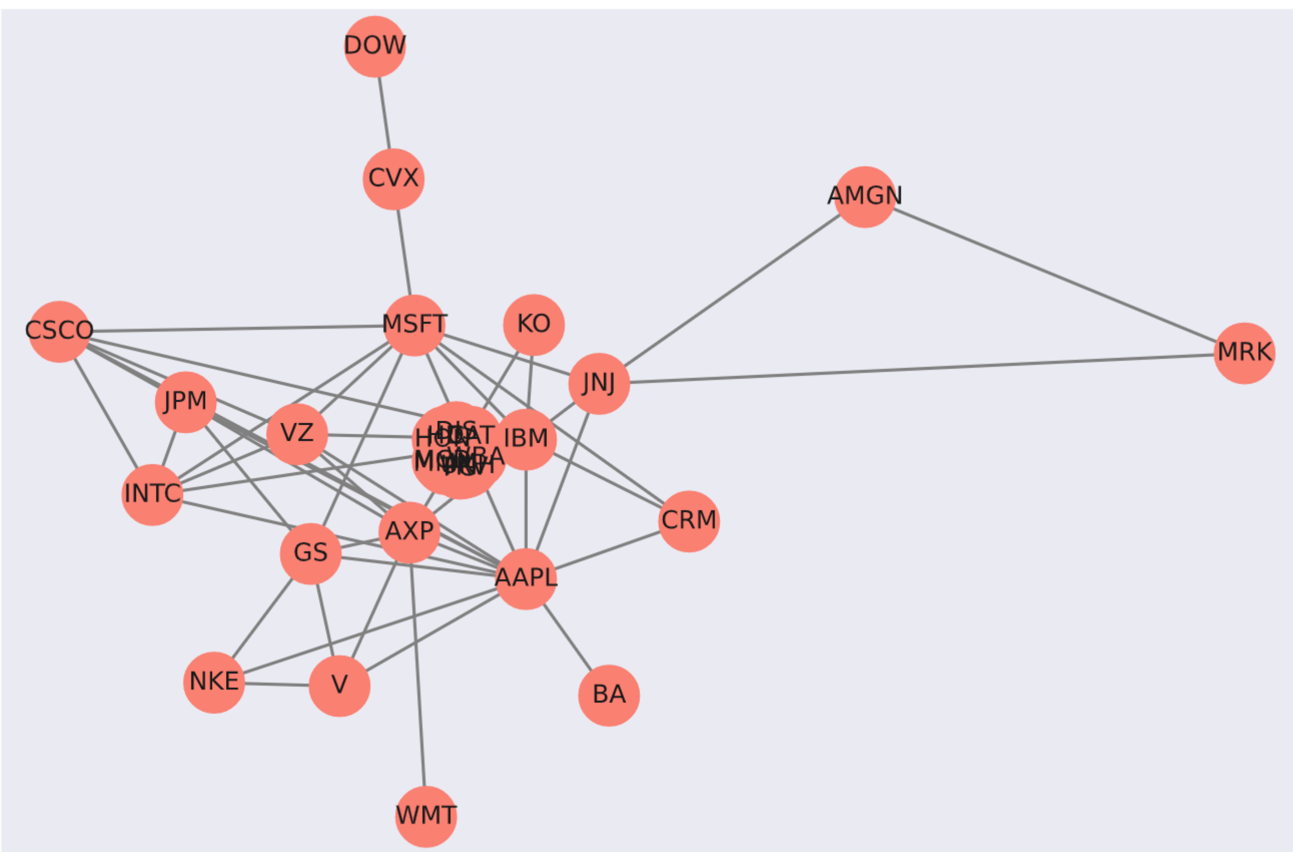


Figure 5: News Articles Co-Mentions Graphs

We also have the same graph for the financial correlations, formed from the stock price and transaction volume correlation matrices. This graph is much sparser in connections than the previous grap, as you can see from figure 6.

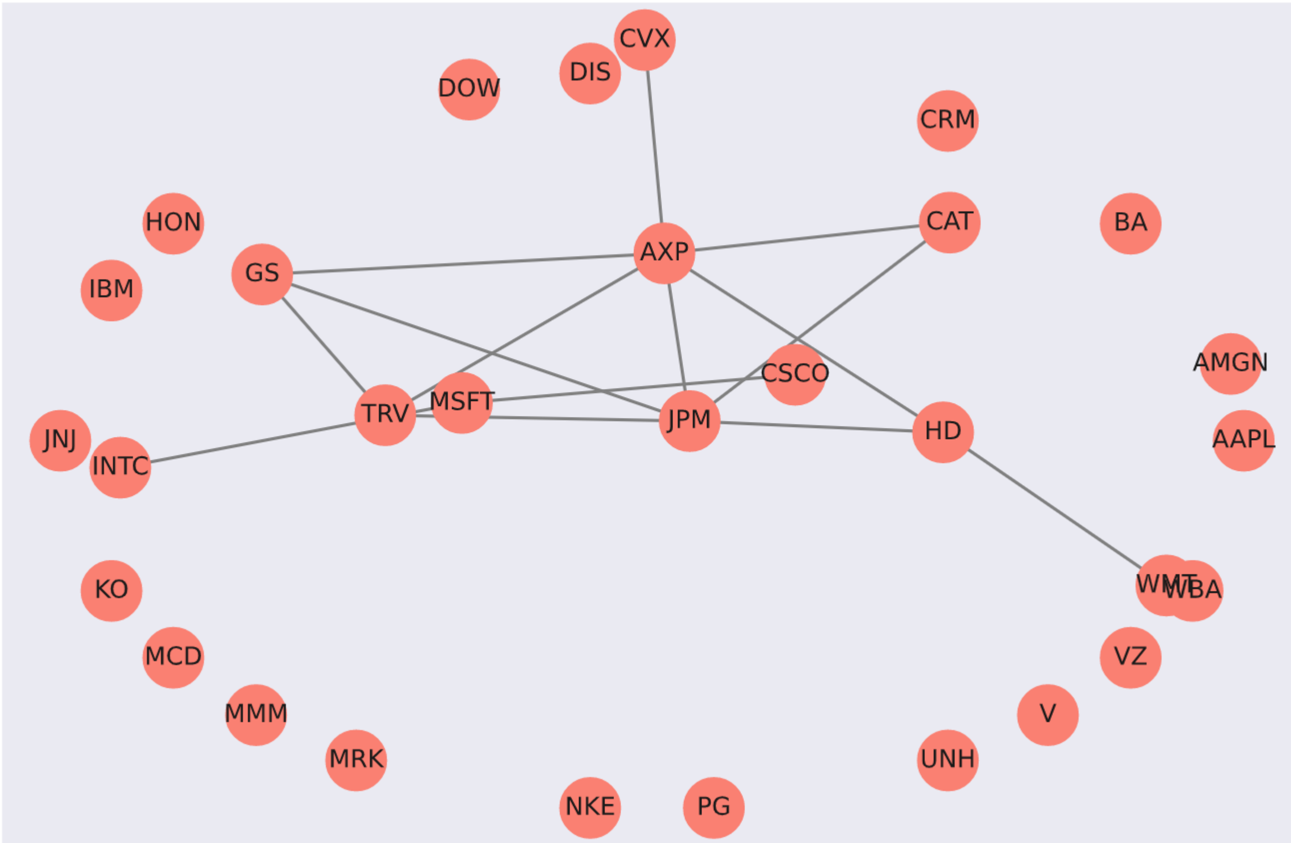


Figure 6: Price/Volume Correlation Graphs

We have several results on community detection. Using $n = 2$ in our community detection process, we were able to identify two distinct communities in a later version of our news article co-mentions graph. Only the largest community is shown in Figure 7, but we can see that there are both technology and financial companies included in this community, with larger bars on the right-hand side corresponding to more central nodes in the graph in terms of their eigenvector centrality score. Since our eigenvector centrality score is large for nodes that are highly connected to other high-scoring nodes, this method is perfect for showing groups of nodes with strong relationships.

Figure 7: Community Detection in the News Article Graph

Figure 8 shows is the largest community found in the financial data graph- and this one seems to show a much bigger breadth of industries included. Home goods, construction equipment, and insurance all appear here, highlighting the potential behind this approach- we may not have thought to look for connections between these industries if not for these graphical structures we created.

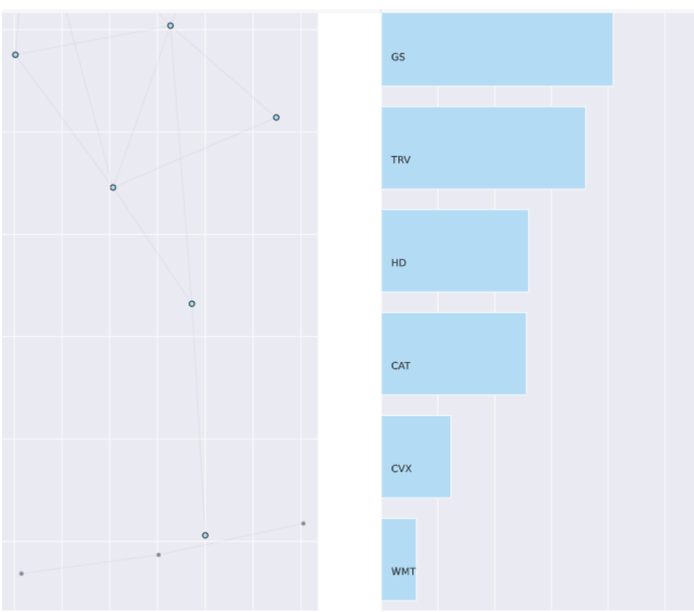


Figure 8: Community Detection in the Financial Data Graph

We also have results on which entities are most central to the graphs we create. In Figure 9, we can see that in the news article data graph, we have tech companies dominating- Apple, Microsoft, IBM, Intel, Cisco, Verizon, and Salesforce are all tech companies before we arrive at American Express. In the graphical structure we have created, there is evidence of tech companies being more dominant in the news than financial companies such as American Express and Goldman Sachs, as the more central nodes have more connections to other nodes in the graph and thus are related to more companies that we have sampled.

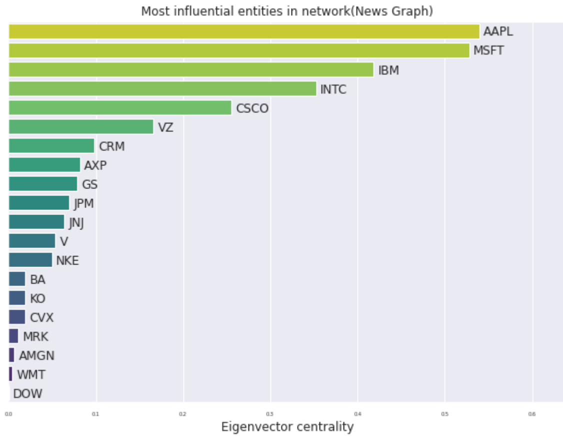


Figure 9: Eigenvector Centrality in the News Article Graph

In Figure 10 we notice a much different effect- in the financial data graph, financial companies are connected to many more nodes than other companies, which makes sense as they must invest in different industries. So, we have evidence that our two distinct knowledge graphs are going to provide us with two different perspectives on connections between companies of interest, and thus that our methods may be worth it for their potential applications to new data.

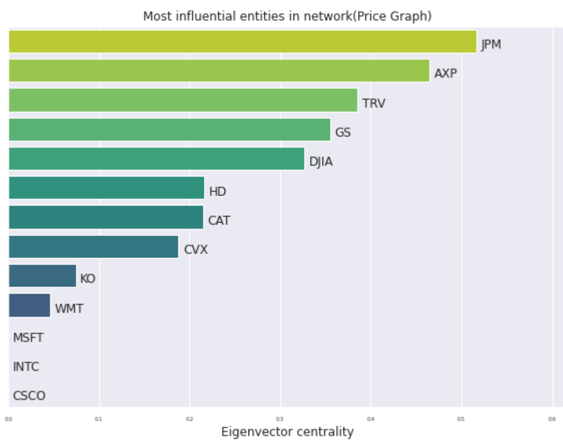


Figure 10: Eigenvector Centrality in the Financial Data Graph

Our results for node attribute prediction were very encouraging. We were successfully able to partition the data into training and testing sets as we described, and we were able to get a look at some of the graphs we were generating for each time window to make sure they were being formed reasonably. Shown in Figure 11 is the common edges graph for the fourth quarter of 2013, showing only the connections that were found in both the news article graph and the financial data.

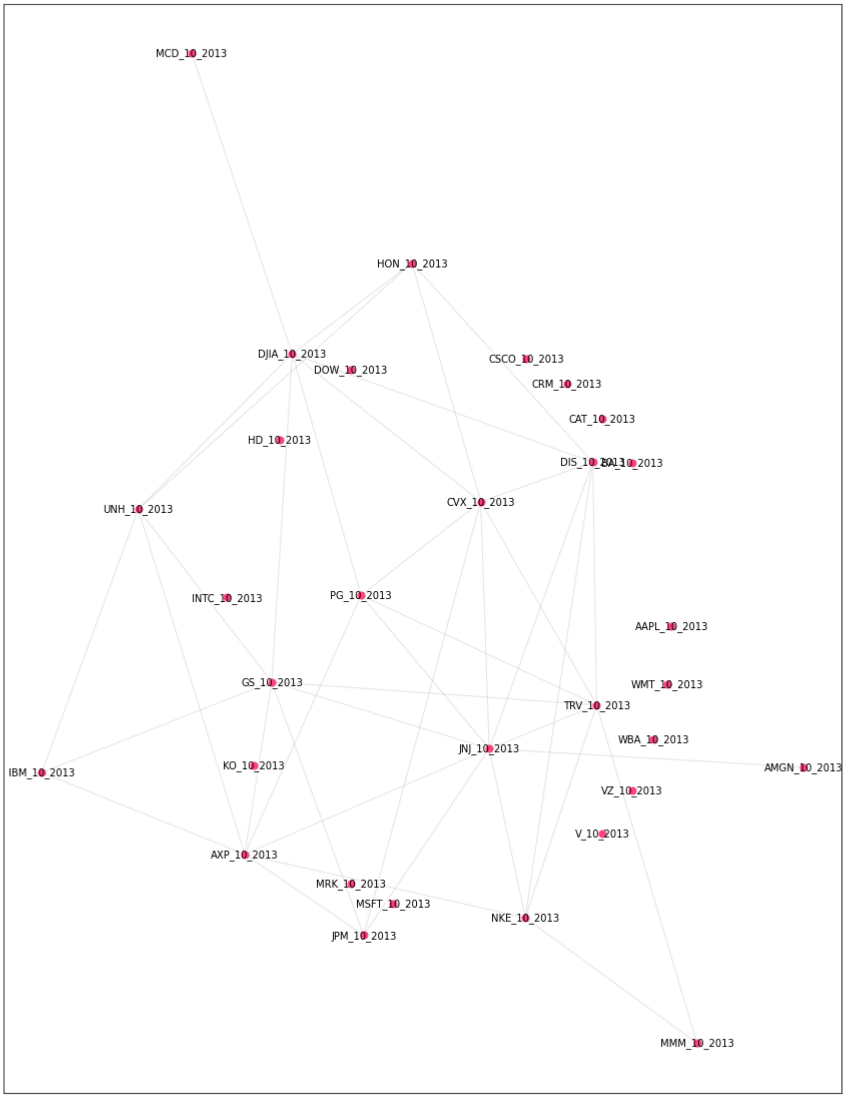


Figure 11: Common Edges Graph

After the edge sets in these graphs are converted into feature sets for the companies in those graphs using GCNConv, we are able to train our model on the data and observe the results of the training process. Shown in Figure 12 are the t-distributed stochastic neighbor embeddings (t-SNE) for our common edges graph training set after 1 epoch (or model training run), and after 200 epochs. These visualizations show that the data points (and specifically their features), in both full rank and low-rank approximations, are becoming separated on a basis that relates to their outcome in terms of average daily stock price. While the differently-labeled points are not entirely separate, this is understandable- we have taken up a difficult prediction task and it is not surprising to see an imperfect prediction system.

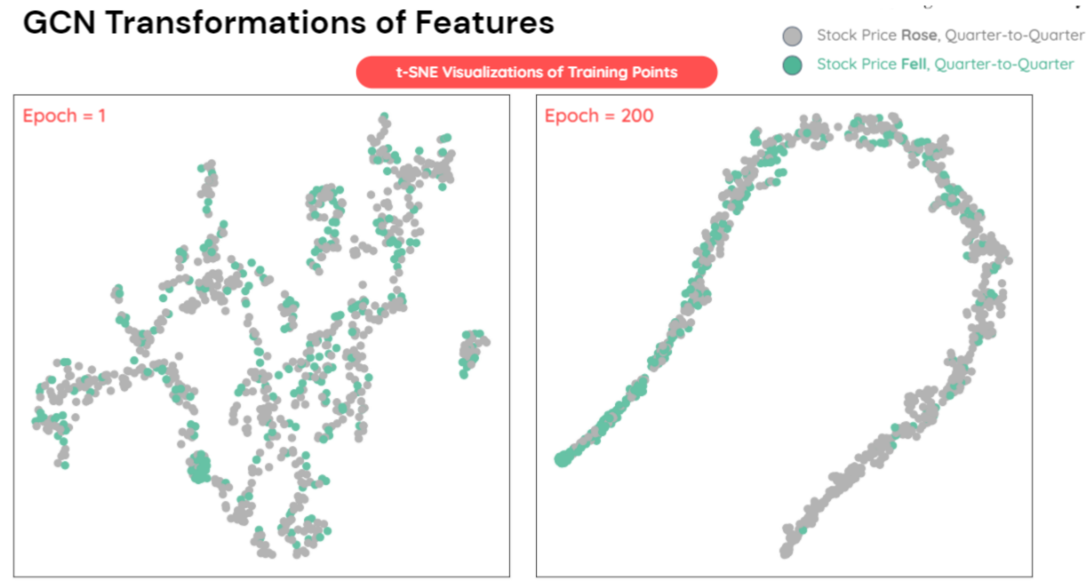


Figure 12: t-SNE Visualizations of Model Output

In Table 3 we can see the accuracy for our different prediction techniques in this context. We see our naive method, which uses the label probabilities of the train set to guess at the test set, and the model built from the fully connected graph both performing rather poorly, but we see great improvements in the performance of our two knowledge graphs (the price graph and the news graph), and the best performance overall in our common edges graph. These accuracy levels are all averaged over ten separate instances of each model- where a model was trained from scratch using a different random seed to begin its modeling decisions.

Metric: Mean Accuracy (10 Random Seeds)

	Method	Train Set (2011 – 2017)	Test Set (2018 -2019)
Baselines	Naive (Random Guess)	58.13 %	55.89 %
	Fully Connected	73.90 %	57.56%
Ours	Price Graph	73.89 %	64.66 %
	News Graph	72.94 %	**66.18 %**
	Combination - Union of edges	73.16 %	62.44 %
	Combination - Common edges	75.77 %	**66.73 %**

Table 3: Accuracy of Graphical Models

Discussion:

This project was fruitful in its ability to both detect communities of similar companies and use learned structures to develop predictions of future information of these companies. We were able to establish that the information found in the news helps us create communities of companies that favor the tech industry, as those companies have strong representation in the news cycle, and that the information found in financial data helps us understand how different companies' finances may be connected as well. We were also able to understand that Apple and Microsoft were massive players in the news cycle, while JP Morgan and American Express were large players in the financial cycle. While this seems to be a common sense conclusion, this not necessary bad news- in times when those relationships are unsure, these graphical structures built from recent data could establish which companies are most connected in markets such as news and finances.

For node attribute prediction, we have substantial proof that a graph built from multiple knowledge graphs (each of which with edges created from separate data sources) can outperform each of the individual knowledge graphs. This is even when our task heavily favored one structure, as a graph built from stock price and transaction correlations seems like it would be intuitively better at forecasting stock price changes- and still there was useful information for this task in the common edges graph that the financial graph alone lacked. The accuracy of our common edges model, given the restrictions of our experiment, establishes our node attribute prediction experiment as a proof of concept for the efficacy of graphical structures in predicting node attributes.

We had several limitations in this project, all of which provide strong directions for future improvement into this topic. We were limited in the number of news articles we could pull from the Google News API due to the built-in HTTP timeouts that Google News uses to prevent cyber attacks, and a clever solution to this data pulling problem could give us a much broader sample of news articles with which to draw connections between companies of interest. We were also limited in the amount of companies we could build a graph around and the number of articles we could analyze because of limited computing resources, which we had access to through Google Colab. Since we only used free accounts for this project, we were unable to handle larger computing jobs or run models which took more than a few hours, and the ability to experiment with some of these models may have been able to aid in our community detection methods as well as our node attribute prediction.

Additionally, if we had more time with this project, there are several other steps we would have pursued. One such step would involve evaluating more than just two different types of knowledge graphs, perhaps by pulling in data on SEC report filings for each company as well as sector-based data that could be used to create additional structures based on the industry a company does most of its business in. Another large step would be to extensively tune the edge cutoffs used in both the financial data graph and the news article graph, as well as tuning the hyperparameters of the neural network used in the node attribute prediction. This step would allow us to test the limits of the performance of our graphs and draw better quantitative conclusions on the percent increase in accuracy that could be expected when moving from knowledge graphs to combined graphs for prediction purposes.

Predicting new information is further off for us as a team. While we have made progress in Node Attribute Prediction, we are currently in the modeling stage for this data, so we are not yet able to say whether our graphs and communities are well-suited to predicting new information about their member nodes. This is our number one priority going forward and our main focus, but evaluation may be tricky- while we can likely develop strong accuracy levels given the right statistical model and evaluate them on a test set, we may have trouble discerning the graphical contribution to these results. We may need some creativity in order to measure this technique’s performance against more standard machine learning methods.

References

1. Ferencz, Marcell. 2020. “Building a Social Network from the News Using Graph Theory by Marcell Ferencz.” Medium. Retrieved April 23, 2021 (https://towardsdatascience.com/building-a-social-network-from-the-news-using-graph-theory-by-marcell-ferencz-9155d314e77f (https://towardsdatascience.com/building-a-social-network-from-the-news-using-graph-theory-by-marcell-ferencz-9155d314e77f)).

2. Hu, H. (2021). GoogleNews. PyPI. https://pypi.org/project/GoogleNews/ (https://pypi.org/project/GoogleNews/).

3. Akbik, A., Blythe, D., & Vollgraf, R. (2018, August). Contextual String Embeddings for Sequence Labeling. ACL Anthology. https://www.aclweb.org/anthology/C18-1139/ (https://www.aclweb.org/anthology/C18-1139/).

4. NetworkX Developers. (2014). NetworkX documentation. NetworkX. https://networkx.org/ (https://networkx.org/).

5. Kipf, T. N., & Welling, M. (2017, February 22). Semi-Supervised Classification with Graph Convolutional Networks. arXiv.org. https://arxiv.org/abs/1609.02907 (https://arxiv.org/abs/1609.02907).
6. Anon. 2021. “PyTorch Geometric Documentation¶.” PyTorch Geometric Documentation - pytorch_geometric 1.7.0 Documentation. Retrieved April 23, 2021 (https://pytorch-geometric.readthedocs.io/en/latest/ (https://pytorch-geometric.readthedocs.io/en/latest/)).
7. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
8. Kung-Hsiang, Huang (Steeve). 2019. “Hands on Graph Neural Networks with PyTorch & PyTorch Geometric.” Medium. Retrieved April 23, 2021 (https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geometric-359487e221a8 (https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geometric-359487e221a8)).

Techical Appendix

Open in Colab
(https://colab.research.google.com/github/jnrkufuor/apollo/blob/Ernest/notebooks/Graph.ipynb)

1. Load Packagaes

In [1]:

```
!pip install squarify

import pandas as pd
import numpy as np
import math
import networkx as nx
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import seaborn as sns
sns.set(rc={'figure.figsize':(20,15)})
import squarify
import statistics
#from google.colab import drive
#drive.mount('/content/drive')

tqdm.pandas()
```

```
Requirement already satisfied: squarify in /home/jay/.local/lib/pyt
hon3.8/site-packages (0.4.3)
```

2. Load data

2.1 Load News Data

In [2]:

```
df_links = pd.read_csv('../data/df_links_2011_2015.csv')
print(len(df_links.index))

#hyperparamaters
weight_criteria = 6
```

```
1591
```

1991

2.2 Load Financial data

In [3]:
```
df_prices =  pd.read_csv('../data/price_corr.csv')
df_vol =  pd.read_csv('../data/volume_corr.csv')

df_finance_nds = pd.DataFrame(columns = ["from", "to", "weight"])

df_prices = df_prices.drop(df_prices.columns[0], axis=1)
df_prices.index = df_prices.columns

df_vol = df_vol.drop(df_vol.columns[0], axis=1)
df_vol.index = df_vol.columns
```

2.3 Find Unique Pairs from Correlation Coefficient

In [4]:
```
# Get correlation pairs for Price and Volume
df_corr_price = df_prices[abs(df_prices) >= 0.0001].stack().reset_i
ndex()
df_corr_vol = df_vol[abs(df_vol) >= 0.0001].stack().reset_index()

#Take out lower triangle
#for price
df_corr_price  = df_corr_price[df_corr_price['level_0'].astype(str)
!=df_corr_price['level_1'].astype(str)]
df_corr_price['ordered-cols'] = df_corr_price.apply(lambda x: '-'.j
oin(sorted([x['level_0'],x['level_1']])),axis=1)

#for volume
df_corr_vol  = df_corr_vol[df_corr_vol['level_0'].astype(str)!=df_c
orr_vol['level_1'].astype(str)]
df_corr_vol['ordered-cols'] = df_corr_vol.apply(lambda x: '-'.join(
sorted([x['level_0'],x['level_1']])),axis=1)

#Remove duplicates and exclude self-correlated values
#for price
df_corr_price = df_corr_price.drop_duplicates(['ordered-cols'])
df_corr_price.reset_index(drop=True, inplace=True)
df_corr_price.drop(['ordered-cols'], axis=1, inplace=True)
```

```
#for volume
df_corr_vol = df_corr_vol.drop_duplicates(['ordered-cols'])
df_corr_vol.reset_index(drop=True, inplace=True)
df_corr_vol.drop(['ordered-cols'], axis=1, inplace=True)

#rename columns
df_corr_price.columns = ["from","to","correlation"]
df_corr_vol.columns = ["from","to","correlation"]

#pull out individual nodes
unique_nodes=[]
for row in df_corr_price.iterrows():
    if row[1]["from"] not in unique_nodes: unique_nodes.append(row[
1]["from"])
    if row[1]["to"] not in unique_nodes: unique_nodes.append(row[1]
["to"])
```

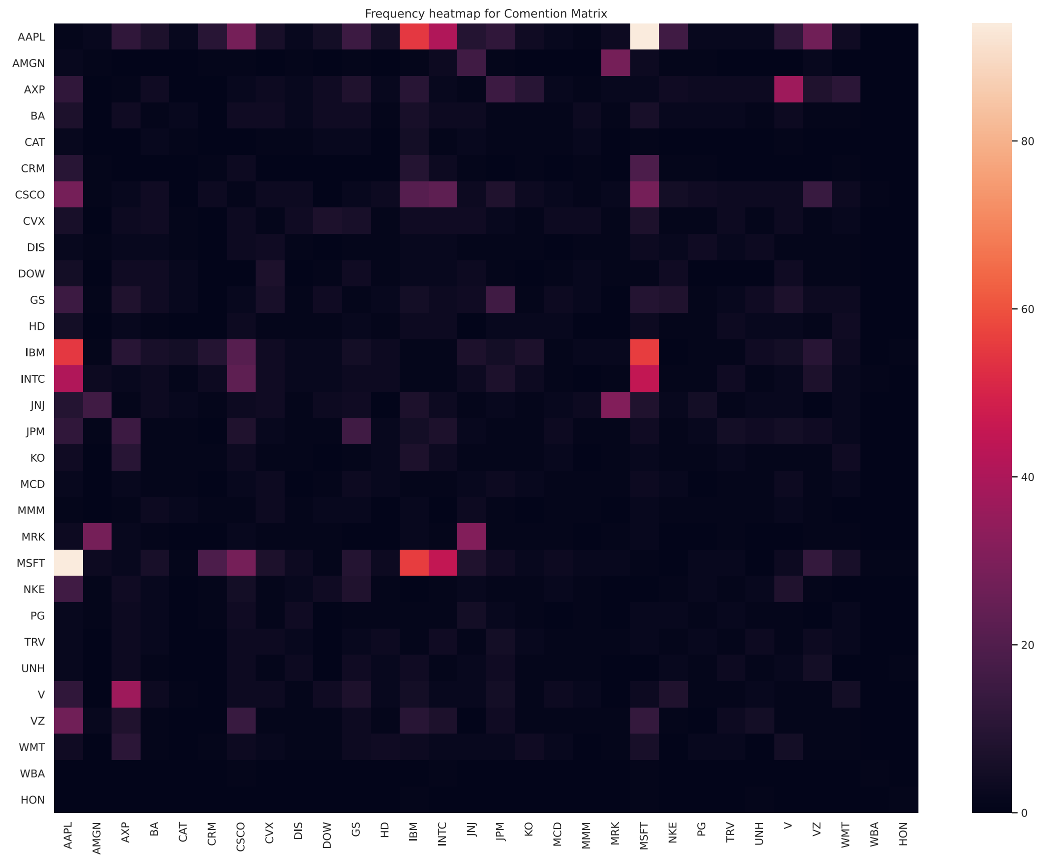
3. Subset data

In [5]:
```
#Subset mews data. Count all links and store under weight column
df_links = df_links.groupby(['from', 'to']).size().reset_index()
df_links.rename(columns={0: 'weight'}, inplace=True)
df_links.reset_index(drop=True, inplace=True)
```

In [6]:
```
sns.set(rc={'figure.figsize':(20,15)})

#Build Co-mention Matrix
df_links[['from', 'to', 'weight']].sort_values('weight', ascending=
False)
col=[]

#Extract Unique Columns
for row in df_links.iterrows():
    if row[1]['from'] not in col:
        col.append(row[1]['from'])
    if row[1]['to'] not in col:
        col.append(row[1]['to'])

df_matrix = pd.DataFrame(0,columns =col,index=col)
```

```
for row in df_links.iterrows():
    df_matrix[row[1]['from']][row[1]['to']] = row[1]['weight']
    df_matrix[row[1]['to']][row[1]['from']] = row[1]['weight']
    df_matrix[row[1]['from']][row[1]['from']] = 1
    df_matrix[row[1]['to']][row[1]['to']] = 1

#Construct Heatmap
sns.heatmap(df_matrix).set_title("Frequency heatmap for Comention M
atrix")
```

Out[6]: `Text(0.5, 1.0, 'Frequency heatmap for Comention Matrix')`



In [7]:

```
#normalize values and create co-mention matrix - Use Z-score normma
lization?
#df_links['weight'] =(df_links['weight']-df_links['weight'].min())/
(df_links['weight'].max()-df_links['weight'].min())

#Use Hyper parameter for now
```

```
#use hyper parameter for now
df_links = df_links[df_links['weight'] > weight_criteria]
df_links.head(10)
```

Out[7]:

	from	to	weight
1	AAPL	AXP	8
2	AAPL	BA	7
4	AAPL	CRM	10
5	AAPL	CSCO	28
9	AAPL	GS	15
11	AAPL	IBM	55
12	AAPL	INTC	41
13	AAPL	JNJ	9
14	AAPL	JPM	12
19	AAPL	MSFT	94

3.2 Subset Financial Nodes

In [99]:
```
df_corr_price[df_corr_price["to"] == "JPM"].head(10).sort_values("c
orrelation",ascending=False)
```

Out[99]:

	from	to	correlation
70	AXP	JPM	0.710496
121	CAT	JPM	0.528654
211	DIS	JPM	0.512087
190	CVX	JPM	0.470705
168	CSCO	JPM	0.455565
96	BA	JPM	0.453687
15	AAPL	JPM	0.362416

15	AAPL	JPM	0.362416
43	AMGN	JPM	0.330208
145	CRM	JPM	0.320565
231	DOW	JPM	0.223695

In [102]:
```
df_corr_vol[df_corr_price["to"] == "JPM"].head(10).sort_values("cor
relation",ascending=False)
```

Out[102]:

	from	to	correlation
70	AXP	JPM	0.719971
121	CAT	JPM	0.602653
190	CVX	JPM	0.579849
231	DOW	JPM	0.517871
211	DIS	JPM	0.393744
145	CRM	JPM	0.333988
15	AAPL	JPM	0.317996
96	BA	JPM	0.175565
168	CSCO	JPM	0.168824
43	AMGN	JPM	0.112505

In [8]:
```
#Subset financial Nodes using Stock Price and Volume Data
#If volume or price are above 0.8, add an edge between companies
#If volume and price are above 0.5 but less than 0.8, add an edge.

for i in range(1,len(df_corr_price)):
    if(abs(df_corr_price["correlation"][i]) > 0.8 or abs(df_corr_vo
l["correlation"][i]) > 0.8):
        df_finance_nds= df_finance_nds.append({"from" : df_corr_vol[
"from"][i], "to" : df_corr_vol["to"][i], "weight" : ((abs(df_corr_p
rice["correlation"][i])+abs(df_corr_price["correlation"][i]))/2)},i
gnore_index=True)
```

```
    elif (abs(df_corr_price["correlation"][i]) < 0.8 and abs(df_cor
r_vol["correlation"][i]) < 0.8):
        if (abs(df_corr_price["correlation"][i]) >= 0.5 and abs(df
_corr_vol["correlation"][i]) >= 0.5):
            df_finance_nds = df_finance_nds.append({"from" : df_co
rr_vol["from"][i], "to" : df_corr_vol["to"][i], "weight" : ((abs(df
_corr_price["correlation"][i])+abs(df_corr_price["correlation"][i
]))/2)},ignore_index=True)
df_finance_nds.head(10)
```

Out[8]:

	from	to	weight
0	AXP	CAT	0.548812
1	AXP	CVX	0.510701
2	AXP	GS	0.653400
3	AXP	HD	0.519912
4	AXP	JPM	0.710496
5	AXP	TRV	0.559720
6	CAT	JPM	0.528654
7	CSCO	MSFT	0.567961
8	GS	JPM	0.721061
9	GS	TRV	0.506627

4. Plot Edges

In [9]:

```
#create plot variables

#for news
df_plot_news = df_links
df_plot_news.reset_index(inplace=True, drop=True)

#for finance
df_plot_fin = df_finance_nds
df_plot_fin.reset_index(inplace=True, drop=True)
```

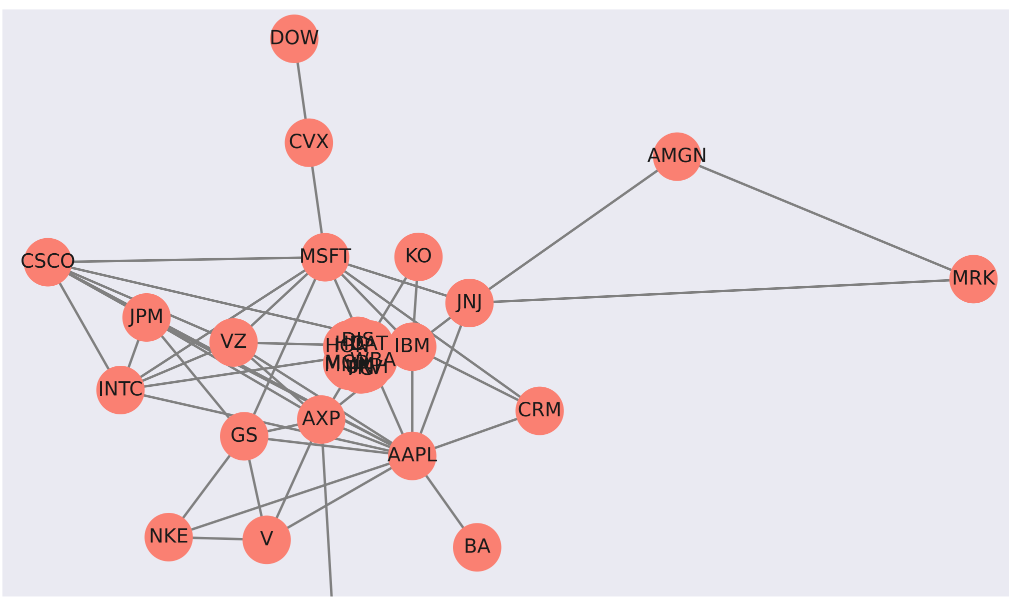
```
df_plots =[df_plot_news,df_plot_fin]

#Build Graph Variables
gr_news = nx.Graph() #news graph
gr_price = nx.Graph() #finacial graph
graph = [gr_news,gr_price]

#get a list of common nodes
joint_nodes = unique_nodes

#add edges and nodes to graph

graph_num=0

#add nodes
for g in graph:
    for nd in unique_nodes:
        g.add_node(nd)

#add edges
for df_plot in df_plots:
    for link in tqdm(df_plot.index):
        graph[graph_num].add_edge(df_plot.iloc[link]['from'],
                    df_plot.iloc[link]['to'],
                    weight=df_plot.iloc[link]['weight'])
    graph_num+=1
```

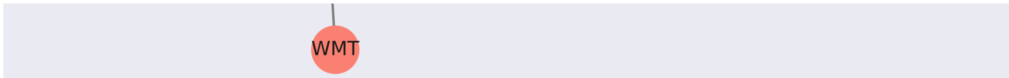
```
100%|██████████| 50/50 [00:00<00:00, 2174.81it/s]
100%|██████████| 14/14 [00:00<00:00, 1723.32it/s]
```

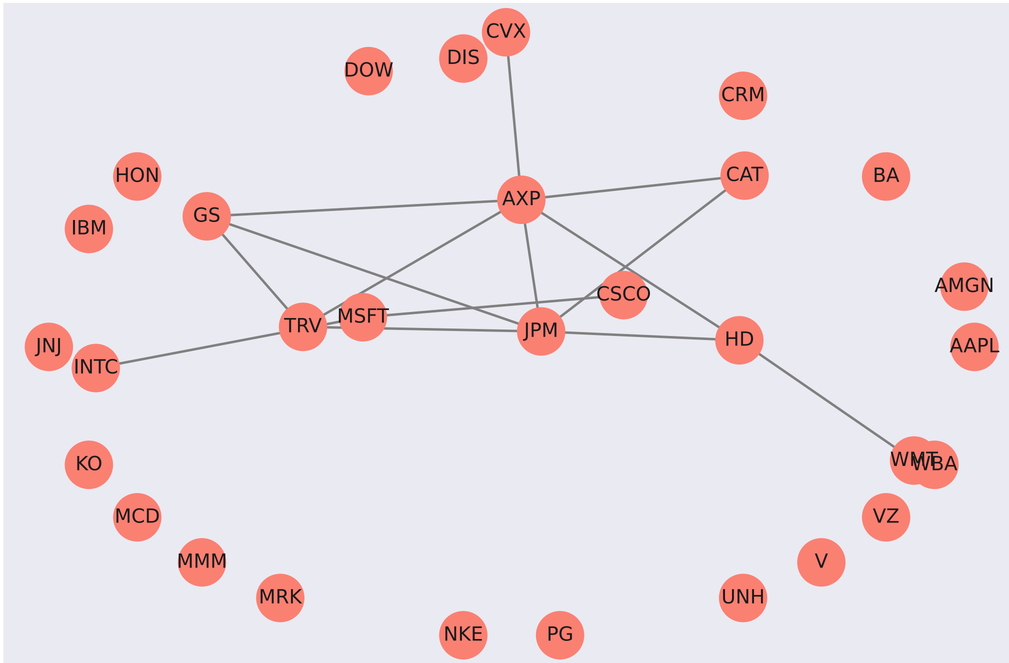
In [10]:

```
graph = [gr_news,gr_price]
node_labels = {}
nodes_multi_layer={}
node_type=["t1","t2"]
type_count=0
for G in graph:
    pos = nx.kamada_kawai_layout(G)
    nodes = G.nodes()
    fig,axs = plt.subplots(1,1,figsize=(15,10))

    el =nx.draw_networkx_nodes(G, pos, nodelist=nodes, node_size=15
00, node_color='salmon', alpha=1, )
    nl=nx.draw_networkx_edges(G, pos, edge_color='grey', width=2,)
    ll=nx.draw_networkx_labels(G, pos, font_size=16, font_family='s
```

```
    ll = nx.draw_networkx_labels(G, pos, font_size=10, font_family='s
ans-serif')
    axs.grid(False)

    #el = nx.draw_networkx_edges(G, pos, alpha=0.1, ax=axs)
    #nl = nx.draw_networkx_nodes(G, pos, nodelist=nodes, node_color
='#FF427b',
                                # node_size=50, ax=axs)
    #ll = nx.draw_networkx_labels(G, pos, font_size=10, font_family
='sans-serif')

    #createdictionary of nodes and labels
    node_count =0
    for node in G.nodes():
        #set the node name as the key and the label as its value
        node_labels[node] = node

        #create nodes for multilayered graph
        nodes_multi_layer[node_count]={"node": node,"type":node_typ
e[type_count]}

        #get like nodes


        node_count+=1
    type_count+=1
```





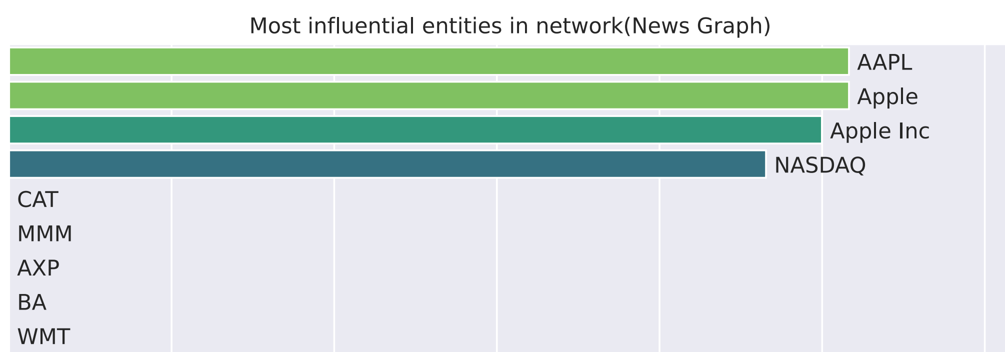


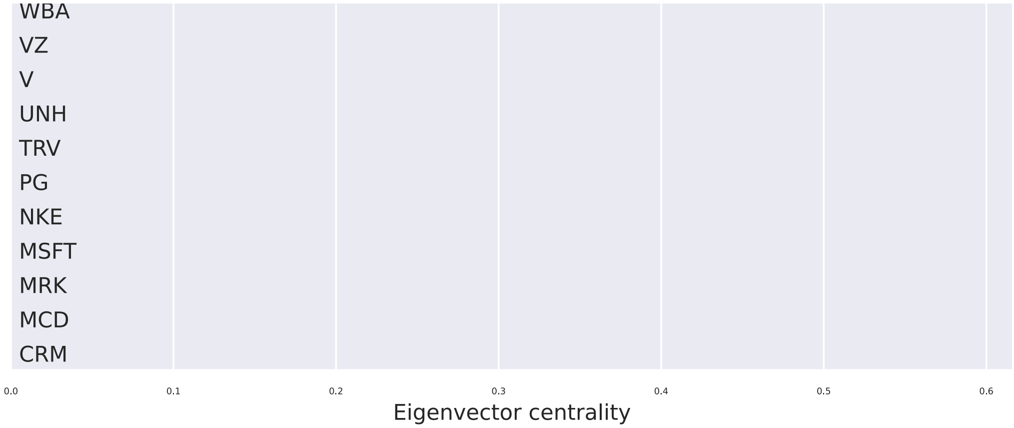
5. Find Subgraphs

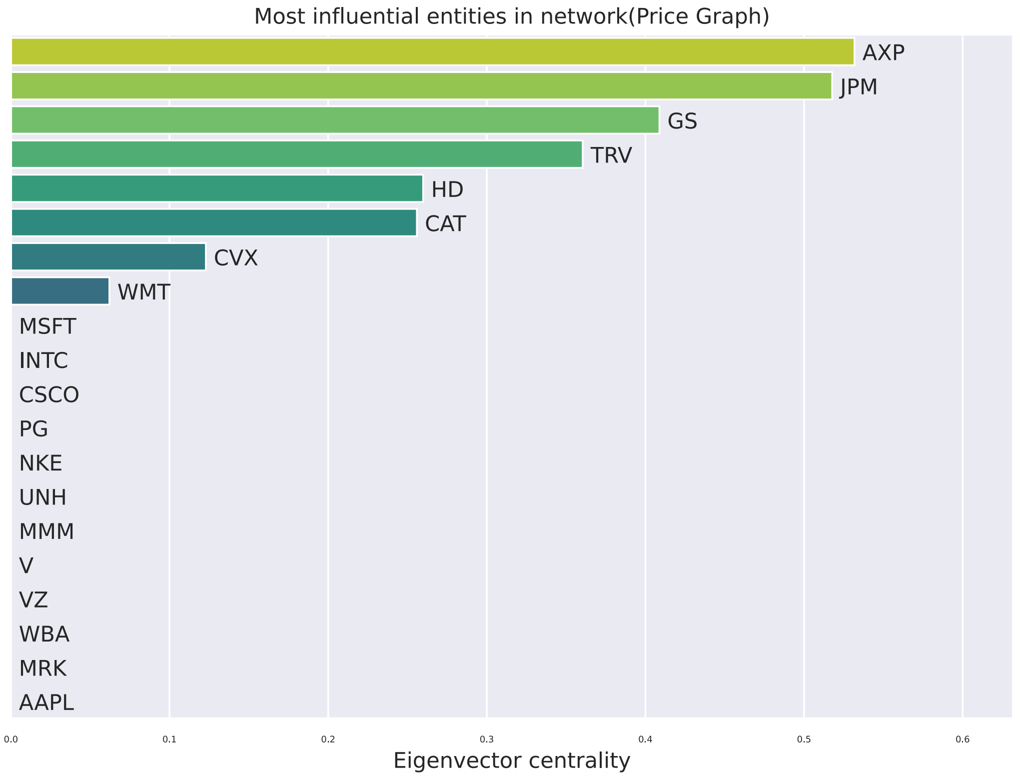
In [31]:
```
txt ="(News Graph)"
for G in graph:

    nodes = []
    eigenvector_cents = []
    ec_dict = nx.eigenvector_centrality(G, max_iter=1000, weight='w
eight')
    for node in tqdm(G.nodes()):
      nodes.append(node)
      eigenvector_cents.append(ec_dict[node])

    df_centralities = pd.DataFrame(data={'entity': nodes,
                                        'eigenvector': eigenvector_
cents})
```

```
    df_cent_top = df_centralities.sort_values('eigenvector', ascend
ing=False).head(20)
    df_cent_top.reset_index(inplace=True, drop=True)
    fig, axs = plt.subplots(figsize=(10,7))
    g = sns.barplot(data=df_cent_top,
                x='eigenvector',
                y='entity',
                dodge=False,
                orient='h',
                hue='eigenvector',
                palette='viridis',)

    g.set_yticks([])
    g.set_title('Most influential entities in network'+txt)
    g.set_xlabel('Eigenvector centrality')
    g.set_ylabel('')
    g.set_xlim(0, max(df_cent_top['eigenvector'])+0.1)
    g.legend_.remove()
    g.tick_params(labelsize=5)

    for i in df_cent_top.index:
        g.text(df_cent_top.iloc[i]['eigenvector']+0.005, i+0.25, df
_cent_top.iloc[i]['entity'])

    #sns.despine()
    g.get_figure().savefig('cent_plot.png', dpi=1000)
    txt ="(Price Graph)"
    nodes = []
    eigenvector_cents=[]
```

```
100%|██████████| 33/33 [00:00<00:00, 187804.66it/s]
100%|██████████| 30/30 [00:00<00:00, 564256.14it/s]
```







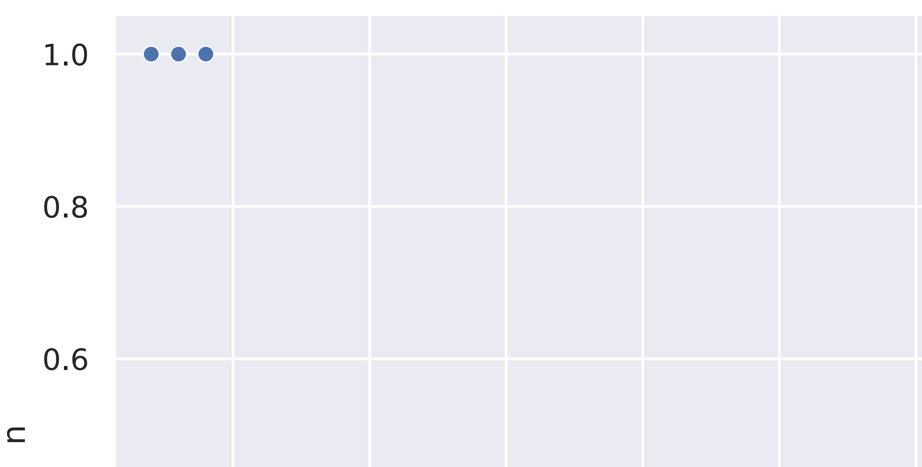
6. Cliques

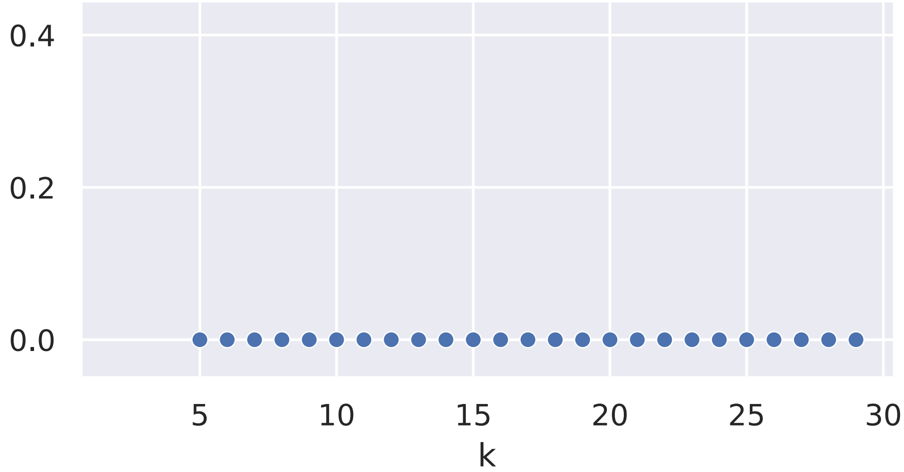
Finding the optimal number

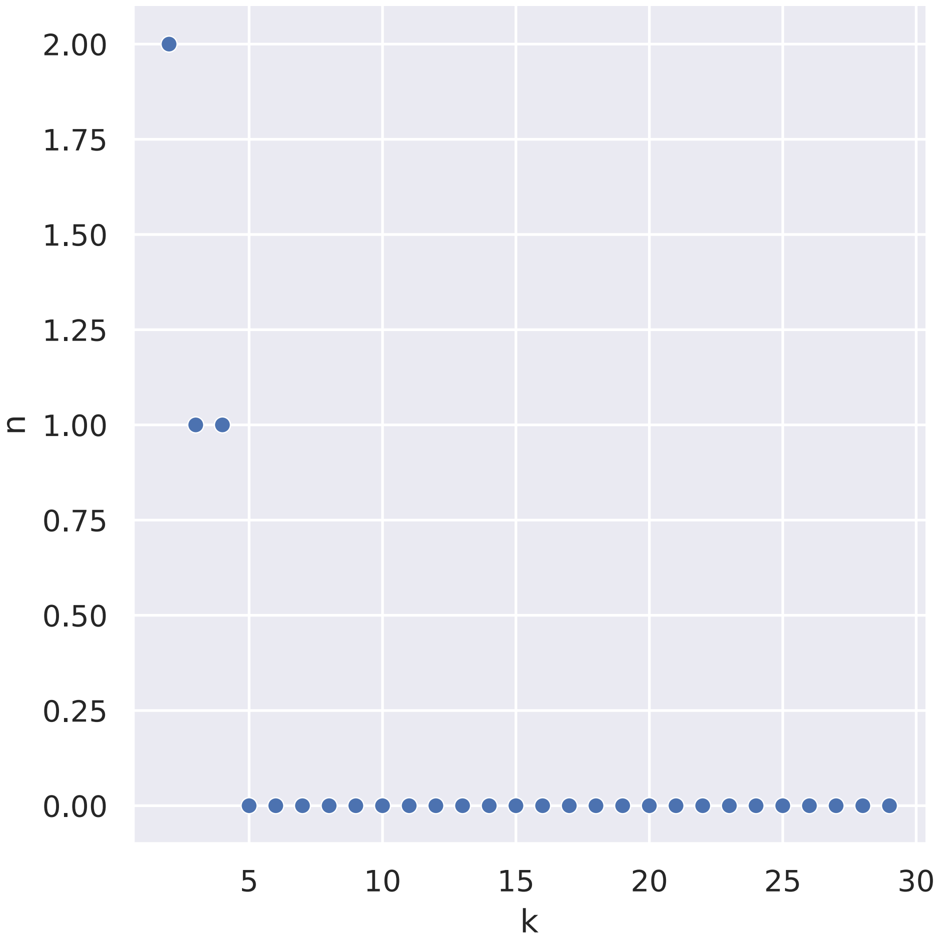
In [32]:
```
from networkx.algorithms.community.kclique import k_clique_communit
ies
```

In [33]:
```
#Explore what the clique size per the nunmber of cliques for each g
raoh
clique_sizes = range(2, 30)
optimal_clique = [] #will hold the optimal clique size for each gra
ph
for G in graph:
  n_cliques = []
  for k in tqdm(clique_sizes):
    n_cliques.append(len(list(k_clique_communities(G, k))))

  optimal_clique.append(2+(n_cliques.index(max(n_cliques)))) #cliqu
e sizes should be greater than one, hence least clique size is 2

  df_relplot = pd.DataFrame(data={'k': clique_sizes,
                                  'n': n_cliques})
  print(n_cliques)
  sns.relplot(data=df_relplot,
              x='k',
              y='n')
```

```
100%|██████████| 28/28 [00:00<00:00, 12205.42it/s]
100%|██████████| 28/28 [00:00<00:00, 14538.32it/s]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0]
[2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0]
```




0.4
0.2
0.0
5
10
15
20
25
30
k


2.00
1.75
1.50
1.25
n
1.00
0.75
0.50
0.25
0.00
5
10
15
20
25
30
k

In [34]:
```
#General cliques using optimal minimum clique size array

cliques=[]
num=0
for G in graph:
    cliques.append(list(k_clique_communities(G, optimal_clique[num
])))
    num+=1
```

Find centralities in cliques

In [24]:
```
#Find centralities within each clique
clx=[]
num = 0
for G in graph:
    eigenvector_cents = []
    entities = []
    clique_ids = []
    for id, clique in enumerate(cliques[num]):
      sg = G.subgraph(list(clique))

      nodes = sg.nodes()

      clique_ids.extend(np.repeat(id, len(nodes)))
      entities.extend(nodes)

      ec_dict = nx.eigenvector_centrality(sg, max_iter=1000, weight
='weight')

      for entity in nodes:
        eigenvector_cents.append(ec_dict[entity])
    df_cliques = pd.DataFrame(data={
        'clique': clique_ids,
        'entity': entities,
        'centrality': eigenvector_cents
    })
    clx.append(df_cliques)
    num+=1
len(clx[0]['clique'].unique())#index 0 = news graph, index 1 = pric
e/vol graph
```

Out[24]: 2

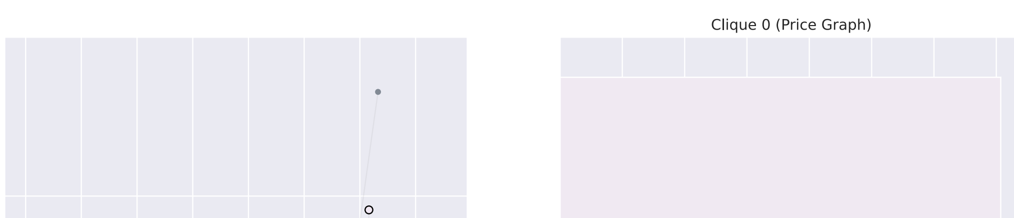
Out[24]: 2

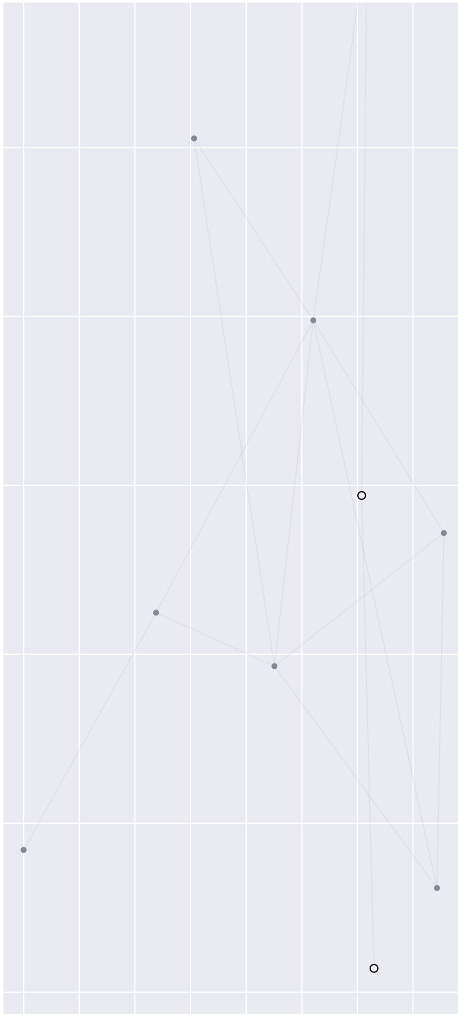
In [43]:
```
#Color pallete for cliques
col_pal = {0: '#F1E8F3',
           1: '#A8DDFF',
           2: '#FF8A5B',
           3: '#74D3AE',
           4: '#93B7BE',
           5: '#D1B1CB',
           6: '#BAF2BB',
           7: '#FFA69E',
           8: '#97EAD2',
           9: '#34E4EA',
           10: '#B95F89',
           99:'#828A95'}
```

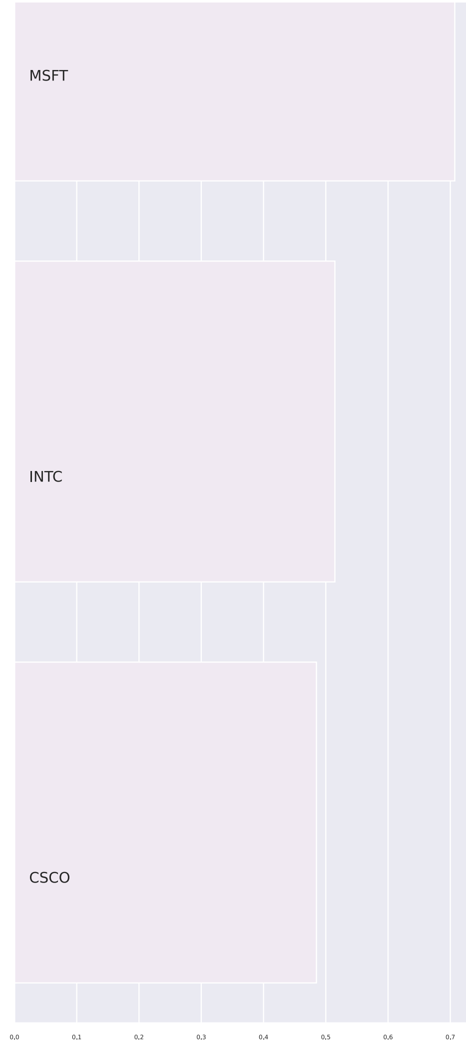
Plot Cliques

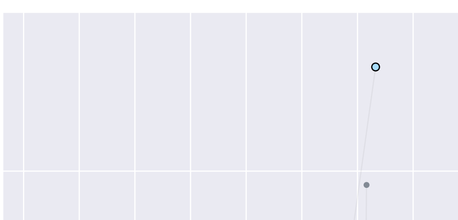
In [60]:
```
#Plot Cliques
clique_num=0
txt ="(News Graph)"
for G in graph:

    df_cliques = clx[clique_num]
    G_clique = G.subgraph(df_cliques['entity'].unique())
    pos = nx.kamada_kawai_layout(G_clique)
    nodes = G_clique.nodes()

    if(len(df_cliques['clique'].unique())>1):
      fig, axs = plt.subplots(max(df_cliques['clique'])+1, 2, figsi
ze=(15,40))

    for clique in range(max(df_cliques['clique'])+1):
      if(len(df_cliques['clique'].unique())<2):
        break
      node_colors = [col_pal[clique] if node in df_cliques[df_cliqu
es['clique']==clique]['entity'].values else col_pal[99] for node in
nodes]
      sizes = [40 if node in df_cliques[df_cliques['clique']==cliqu
e]['entity'].values else 15 for node in nodes]
      edge_colors = ['black' if node in df_cliques[df_cliques['cliq
ue']==clique]['entity'].values else col_pal[99] for node in nodes]
```

```
      ec = nx.draw_networkx_edges(G_clique, pos, alpha=0.05,ax=axs[
clique, 0] )
      nc = nx.draw_networkx_nodes(G_clique, pos, nodelist=nodes, no
de_color=node_colors,
                                  node_size=sizes,ax=axs[clique, 0
],
                                  edgecolors=edge_colors)

      df_clique_ind = df_cliques[df_cliques['clique']==clique]
      df_clique_ind = df_clique_ind.sort_values('centrality', ascen
ding=False).head(15)
      df_clique_ind.reset_index(inplace=True, drop=True)

      g = sns.barplot(data=df_clique_ind,
                  x='centrality',
                  y='entity',
                  hue='clique',
                  palette=col_pal,
                  dodge=False,
                  orient='h',
                  ax=axs[clique, 1])

      g.set_yticks([])
      g.set_title(f'Clique {clique} {txt}')
      g.set_xlabel('')
      g.set_ylabel('')
      g.legend_.remove()
      g.tick_params(labelsize=5)


      for i in df_clique_ind.index:
        g.text(max(df_clique_ind['centrality'])/30, i+0.15, df_cliq
ue_ind.iloc[i]['entity'])

    txt ="(Price Graph)"
    clique_num = clique_num+1
```

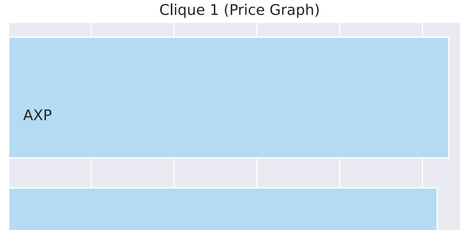


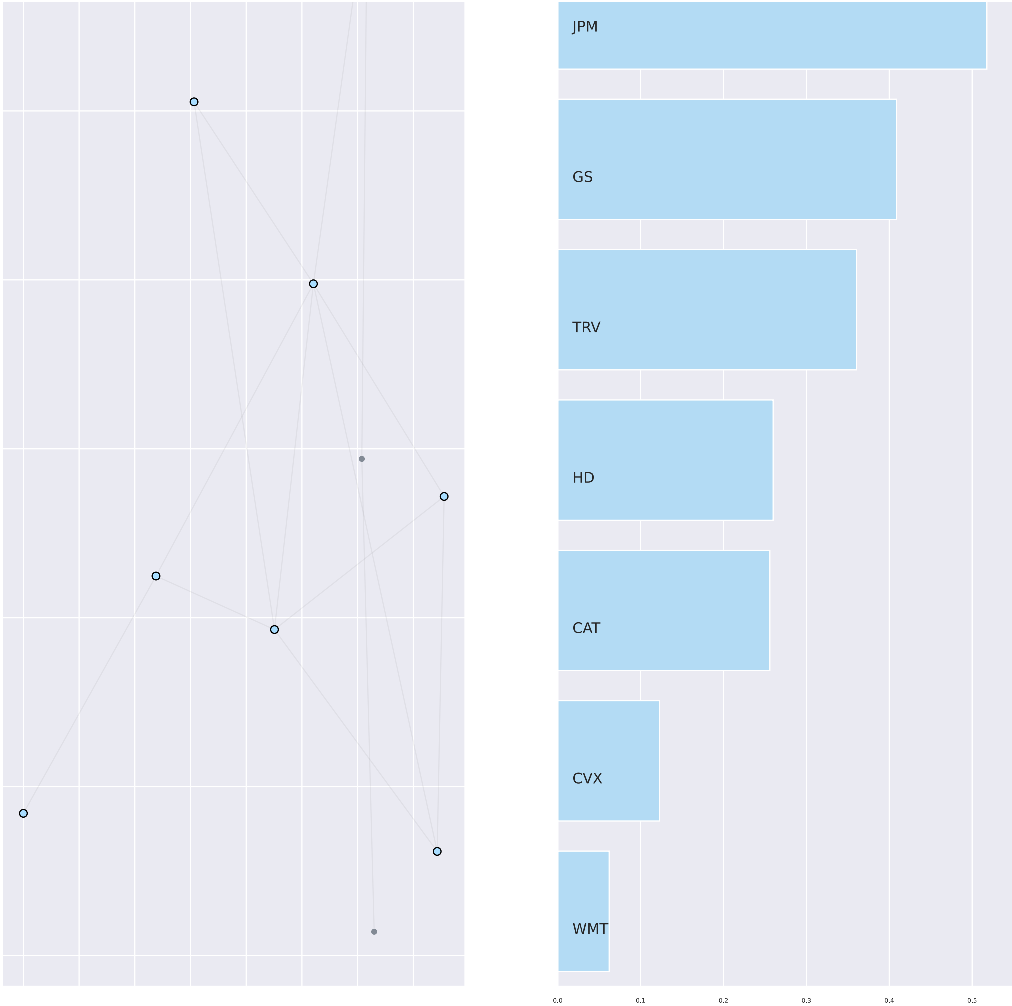






Clique 1 (Price Graph)





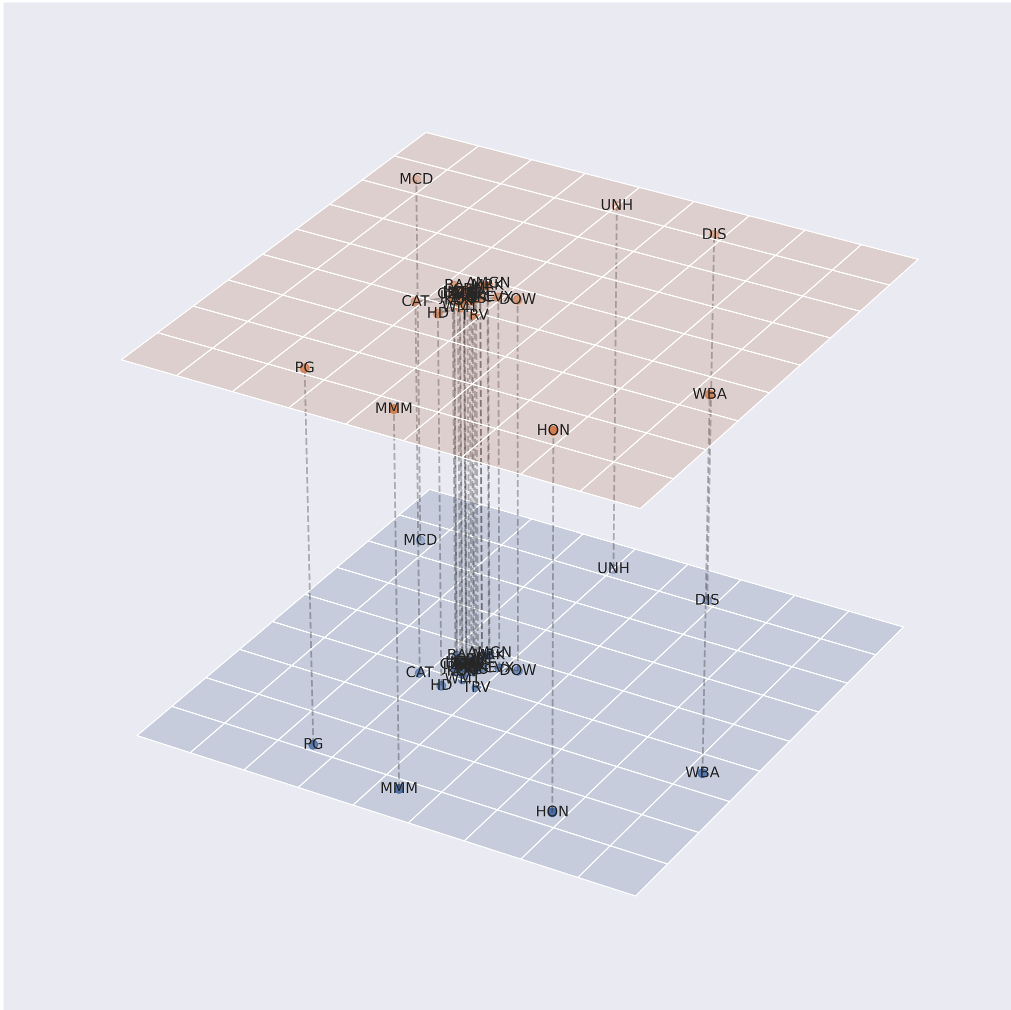
In [ ]:

4 Implement Multiplex Graph

In [11]:

```
#install recommended packages
from LayeredNetworkGraph import LayeredNetworkGraph

# initialise figure and plot
fig = plt.figure(figsize=(15,20))
ax = fig.add_subplot(111, projection='3d')
```

```
LayeredNetworkGraph([graph[0],graph[1]], node_labels=node_labels ,a
x=ax, layout=nx.spring_layout)
ax.set_axis_off()
plt.show()
fig.savefig('graph_images/multilayered.png', dpi=1000)
```



Multiplex Graph using py3plex

In [12]:
```
!pip install py3plex
!pip install leidenalg
!pip install python-igraph
```

Installing Required Packages

```
!pip install squarify

!pip install -q torch-scatter -f https://pytorch-geometric.com/whl/torch-1.8.0+cu101.html
!pip install -q torch-sparse -f https://pytorch-geometric.com/whl/torch-1.8.0+cu101.html
!pip install -q torch-geometric

import torch
import torch_geometric.utils as tgu

import re
import pandas as pd
import numpy as np
import math
import networkx as nx
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import squarify
import statistics
from google.colab import drive
drive.mount('/content/drive')

import random

import pandas_datareader.data as web
from datetime import datetime

tqdm.pandas()
```

```
Collecting squarify
  Downloading https://files.pythonhosted.org/packages/0b/2b/2e77c35326efec19819cd1d729540d4d235e6c2a3f37658288a363a67da
Installing collected packages: squarify
Successfully installed squarify-0.4.3
```

```
     |████████████████████████████████| 2.6MB 28.4MB/s
     |████████████████████████████████| 1.5MB 26.4MB/s
     |████████████████████████████████| 215kB 24.1MB/s
     |████████████████████████████████| 235kB 31.6MB/s
     |████████████████████████████████| 2.2MB 38.6MB/s
     |████████████████████████████████| 51kB 5.7MB/s
  Building wheel for torch-geometric (setup.py) ... done
Mounted at /content/drive
/usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Panel class is removed from pandas. Accessir
  from pandas import Panel
```

Running the Experiment for News and Financial Data

```
wiki  = pd.read_html('https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average#Components')
wiki_table  = wiki[1]
symbols = (wiki_table.Symbol.values.tolist()) + ['DJIA']
df = pd.DataFrame(symbols, columns=['Symbol'])
start_dates = pd.date_range(start='2011-01-01', end='2019-12-01', freq='MS')
end_dates = pd.date_range(start='2011-01-31', end='2019-12-31', freq='M')
news_graphs_vec=[]
price_graphs_vec=[]
comb1_graphs_vec=[]
comb2_graphs_vec=[]
comb3_graphs_vec=[]
comb4_graphs_vec=[]
conn_graphs_vec=[]
emp_graphs_vec=[]

pull_start = '2011-01-01'
pull_end  = '2019-12-31'
df = pd.DataFrame(symbols, columns=['Symbol'])
symbols = sorted(symbols)

for i, symbol in enumerate(symbols):
    try:
        df = web.DataReader(symbol,'yahoo', pull_start, pull_end)
```

```
        df = df[['Adj Close','Volume']]
        df.to_csv('/content/drive/My Drive/JPM_financial_data/' + "{}.csv".format(symbol))
    except KeyError:
      print("Error for {}".format(symbol))
      pass
#df_price[(df_price.index > '2011-03-31') & (df_price.index > '2011-05-30')]

index = pd.date_range(start=pull_start, end=pull_end, freq='D')     # initialize an empty DateTime Index
df_price = pd.DataFrame(index=index, columns=symbols)               # initialize empty dataframes
df_volume = pd.DataFrame(index=index, columns=symbols)

for symbol in symbols:
    symbol_df = pd.read_csv('/content/drive/My Drive/JPM_financial_data/' + symbol+".csv").set_index('Date')
    symbol_df.index = pd.to_datetime(symbol_df.index)

    df_price[symbol] = symbol_df['Adj Close']
    df_volume[symbol] = symbol_df['Volume']

df_price.dropna(how='all', inplace=True)
df_volume.dropna(how='all', inplace=True)
assert((df_price.index == df_volume.index).all())
df_price = df_price.bfill(axis='rows')
df_price = df_price.ffill(axis='rows')

df_links=pd.read_csv('/content/drive/My Drive/df_links_2011_2019.csv')

new_dates=[]

for date in df_links['date']:
    reg_date=re.sub("^.*?([A-Z])", "\\1", date)
    temp_date=datetime.strptime(reg_date,"%b %d, %Y")
    new_dates.append(pd.to_datetime(datetime.strftime(temp_date, "%Y-%m-%d")))

df_links['date']=new_dates
df_news_lengths=[None]*35
df_news_lengths2=[None]*35

for i in range(0, 35, 1):
```

```
df_links_pres = df_links[(df_links['date'] >= start_dates[3*i]) & (df_links['date'] <= start_dates[3*i+2])]
df_links_pres = df_links_pres.groupby(['from', 'to']).size().reset_index()
df_links_pres.rename(columns={0: 'weight'}, inplace=True)
df_links_pres.reset_index(drop=True, inplace=True)
df_links_pres = df_links_pres[df_links_pres['weight'] > 0]
# df_news_lengths[i]=len(df_links_pres)
# df_links_pres = df_links_pres[df_links_pres['weight'] > 1]
# df_news_lengths2[i]=len(df_links_pres)

df_price_pres = df_price[(df_price.index >= start_dates[3*i]) & (df_price.index <= end_dates[3*i+2])]
df_price_next = df_price[(df_price.index >= start_dates[3*(i+1)]) & (df_price.index <= end_dates[3*(i+1)+2])]
df_volume_pres = df_volume[(df_volume.index >= start_dates[3*i]) & (df_volume.index <= end_dates[3*i+2])]
#code for one-month intervals below
#df_volume_pres = df_volume[(df_volume.index >= start_dates[i]) & (df_volume.index <= end_dates[i])]
#df_price_pres = df_price[(df_price.index >= start_dates[i]) & (df_price.index <= end_dates[i])]
#df_price_next = df_price[(df_price.index >= start_dates[i+1]) & (df_price.index <= end_dates[i+1])]

df_price_pct_pres = df_price_pres.pct_change().dropna(how='all')
df_price_pct_next = df_price_next.pct_change().dropna(how='all')
df_volume_pct_pres = df_volume_pres.pct_change().dropna(how='all')

#added next period's info

price_corr = df_price_pct_pres.corr()
volume_corr = df_volume_pres.corr()

df_finance_nds = pd.DataFrame(columns = ["from", "to", "weight"])
price_corr.index = price_corr.columns
#*******************
volume_corr.index = volume_corr.columns

# Get correlation pairs for Price and Volume
df_corr_price = price_corr[abs(price_corr) >= 0.000001].stack().reset_index()
df_corr_vol = volume_corr[abs(volume_corr) >= 0.000001].stack().reset_index()

#Take out lower triangle
#for price
df_corr_price  = df_corr_price[df_corr_price['level_0'].astype(str)!=df_corr_price['level_1'].astype(str)]
df_corr_price['ordered-cols'] = df_corr_price.apply(lambda x: '-'.join(sorted([x['level_0'],x['level_1']])),axis=1)
```

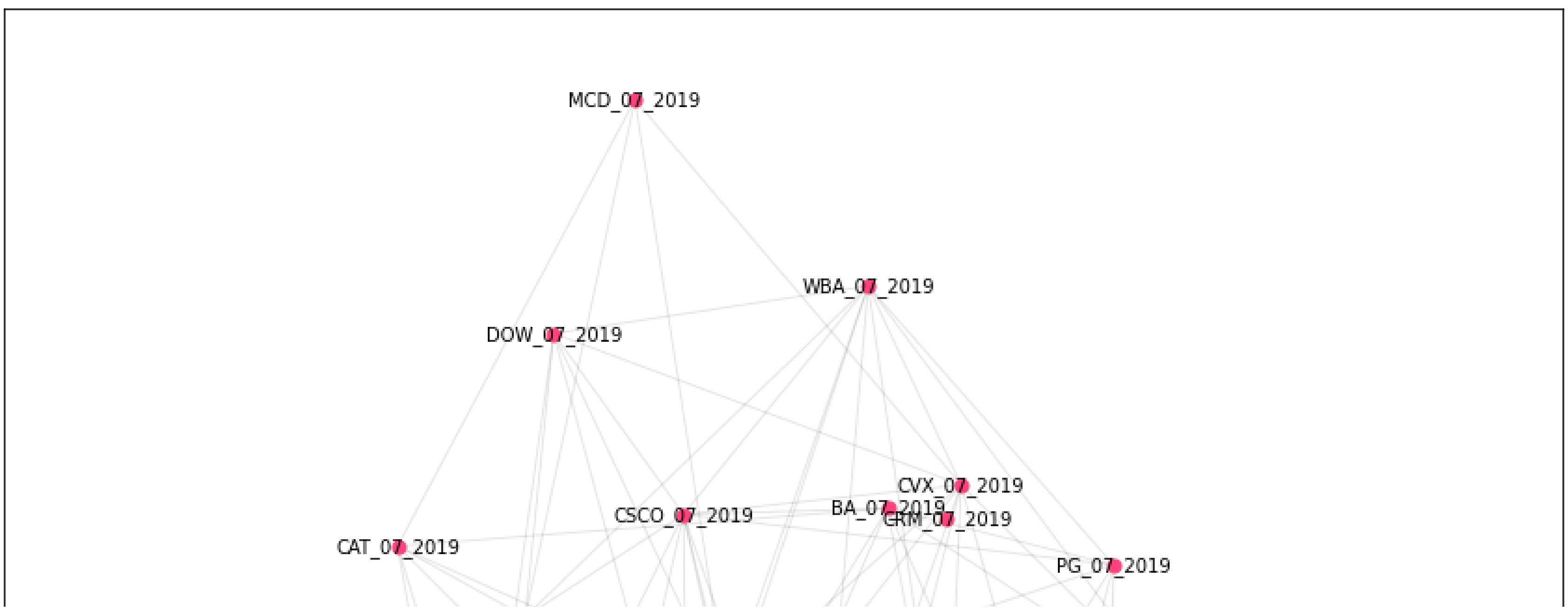
```
    #for volume
    df_corr_vol  = df_corr_vol[df_corr_vol['level_0'].astype(str)!=df_corr_vol['level_1'].astype(str)]
    df_corr_vol['ordered-cols'] = df_corr_vol.apply(lambda x: '-'.join(sorted([x['level_0'],x['level_1']])),axis=1)

    #Remove duplicates and exclude self-correlated values
    #for price
    df_corr_price = df_corr_price.drop_duplicates(['ordered-cols'])
    df_corr_price.reset_index(drop=True, inplace=True)
    df_corr_price.drop(['ordered-cols'], axis=1, inplace=True)


    #for volume
    df_corr_vol = df_corr_vol.drop_duplicates(['ordered-cols'])
    df_corr_vol.reset_index(drop=True, inplace=True)
    df_corr_vol.drop(['ordered-cols'], axis=1, inplace=True)

    #rename columns
    df_corr_price.columns = ["from","to","correlation"]
    df_corr_vol.columns = ["from","to","correlation"]

    unique_nodes=[]
    for colname in df_price.columns:
        if colname not in unique_nodes:
          unique_nodes.append(colname)
        else:
          continue

    df_news_nodes=df_finance_nds

    for j in range(0,len(df_corr_price)):
        if(abs(df_corr_price["correlation"][j]) > 0.6 or abs(df_corr_vol["correlation"][j]) > 0.6):
          df_finance_nds= df_finance_nds.append({"from" : df_corr_vol["from"][j], "to" : df_corr_vol["to"][j], "weight" : ((ab
        elif (abs(df_corr_price["correlation"][j]) < 0.6 and abs(df_corr_vol["correlation"][j]) < 0.6):
            if (abs(df_corr_price["correlation"][j]) >= 0.4 and abs(df_corr_vol["correlation"][j]) >= 0.4):
                df_finance_nds = df_finance_nds.append({"from" : df_corr_vol["from"][j], "to" : df_corr_vol["to"][j], "weight"
    #should consider making these edges directed if we have time
    #negative correlation is VERY different than positive correlation for our predictions
    #Update: I've investigated this and we don't have any edges drawn for negative correlations
```

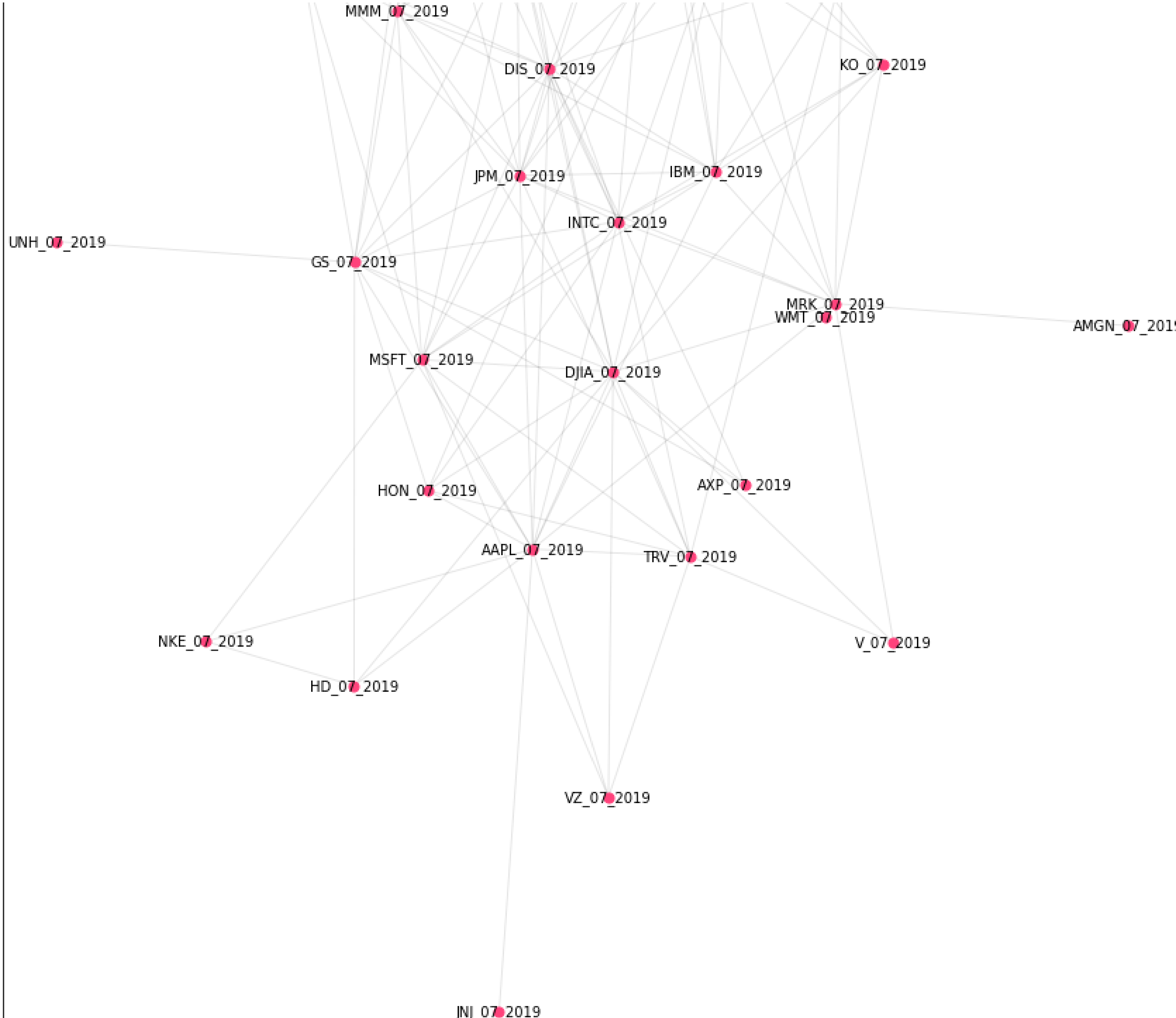
```
#Update: I've investigated this and we don't have any edges drawn for negative correlations
#Ideally this should be fixed going forward but right now it isn't affecting modeling

#for finance*****
df_plot_fin = df_finance_nds
df_plot_fin.reset_index(inplace=True, drop=True)

df_plot_news = df_links_pres
df_plot_news.reset_index(inplace=True, drop=True)

#combined graph 1- all edges in either graph
df_plot_comb1=df_finance_nds[['from','to']].append(df_links_pres[['from','to']])
df_plot_comb1.reset_index(inplace=True, drop=True)

#combined graph 2- 50% random sample of all edges in either graph
df_plot_comb2=(df_finance_nds[['from','to']].sample(n = int(0.5*round(len(df_finance_nds['from'])))).append(
    df_links_pres[['from','to']].sample(n = int(0.5*round(len(df_links_pres['from']))))
)
df_plot_comb2.reset_index(inplace=True, drop=True)

#combined graph 3- all edges shared between both graphs
df_plot_comb3=df_links_pres[['from','to']].merge(df_finance_nds[['from','to']], how='inner', on=['from', 'to'])
df_plot_comb3.reset_index(inplace=True, drop=True)

#combiend graph 4- all edges shared between both, 50% random sample of others
df_plot_comb4=df_plot_comb3[['from','to']]
dfpc4_tempf = df_finance_nds[['from','to']].merge(df_links_pres[['from','to']], how = 'outer' ,indicator=True).loc[lambda 
dfpc4_tempn = df_links_pres[['from','to']].merge(df_finance_nds[['from','to']], how = 'outer' ,indicator=True).loc[lambda 
dfpc4_tempb = dfpc4_tempf[['from','to']].append(dfpc4_tempn[['from','to']])
df_plot_comb4 = df_plot_comb4.append(dfpc4_tempb.sample(n = int(0.5*round(len(dfpc4_tempb['from'])))))
df_plot_comb4.reset_index(inplace=True, drop=True)

df_plot_conn = df_corr_vol.iloc[:,0:2]

df_plot_emp = pd.DataFrame()

df_plots = [df_plot_fin, df_plot_news, df_plot_comb1, df_plot_comb2, df_plot_comb3, df_plot_comb4, df_plot_conn, df_plot_e
#Build Graph Variables

gr_price = nx.Graph() #financial graph
```

```
gr_price = nx.Graph() #financial graph
gr_news = nx.Graph()
gr_comb1 = nx.Graph() #creating 3 attempts at a combined-edge graph
gr_comb2 = nx.Graph()
gr_comb3 = nx.Graph()
gr_comb4 = nx.Graph()
gr_conn = nx.Graph()
gr_emp = nx.Graph()
graph = [gr_price,gr_news,gr_comb1,gr_comb2,gr_comb3, gr_comb4, gr_conn, gr_emp]
#add edges and nodes to graph

#add nodes

for nd in unique_nodes:
   for g in graph:
     g.add_node(nd)

for plot_num in range(0,len(df_plots),1):
  for link in tqdm(df_plots[plot_num].index):
      graph[plot_num].add_edge(df_plots[plot_num].iloc[link]['from'],
                df_plots[plot_num].iloc[link]['to'])
               #weight=df_plots[plot_num].iloc[link]['weight'])
               #commented because we aren't using the weights and weights become trickier (but probably still doable) with
  #**

node_labels = {}
nodes_multi_layer={}
node_type=["t1","t2"]
type_count=0


month_pct_chg=df_price_next.mean(axis=0) - df_price_pres.mean(axis=0)
month_chg_label=pd.Series(np.zeros(len(month_pct_chg)))
for index in range(0, len(month_pct_chg),1):
  if month_pct_chg[index] > 0:
    month_chg_label[index]=1
  else:
    month_chg_label[index]=0

month_chg_label.index=month_pct_chg.index
```

```
  month_chg_label.index=month_pct_chg.index
  for g in graph:
    nx.set_node_attributes(g, month_chg_label, name='y')
    nx.set_node_attributes(g, df_price_pct_pres.iloc[0:57], name='x')
  #chosen y- percent change in 3-month avg from 1 period to next, 1 is increase 2 is decrease
  #index on df_price_pct_pres added here because Dataloader appears to need homogeneous length
  #ideally would be bfilled to equal length of 63 but that's a potential task
  for g_ind in range(0,len(graph),1):
    graph[g_ind]=nx.relabel.relabel_nodes(graph[g_ind], lambda x: x + str('_') + str(start_dates[3*i])[5:7] + str('_') + str

  #above line separates nodes from different years
  price_graphs_vec.append(graph[0])
  news_graphs_vec.append(graph[1])
  comb1_graphs_vec.append(graph[2])
  comb2_graphs_vec.append(graph[3])
  comb3_graphs_vec.append(graph[4])
  comb4_graphs_vec.append(graph[5])
  conn_graphs_vec.append(graph[6])
  emp_graphs_vec.append(graph[7])



#createdictionary of nodes and labels
node_count =0
for node in comb4_graphs_vec[34].nodes():
    #set the node name as the key and the label as its value
    node_labels[node] = node

    node_count+=1

pos = nx.kamada_kawai_layout(comb4_graphs_vec[34])
fig, axs = plt.subplots(1, 1, figsize=(15,20))

el = nx.draw_networkx_edges(comb4_graphs_vec[34], pos, alpha=0.1, ax=axs)
nl = nx.draw_networkx_nodes(comb4_graphs_vec[34], pos, nodelist=comb4_graphs_vec[34].nodes, node_color='#FF427b',
                            node_size=50, ax=axs)
ll = nx.draw_networkx_labels(comb4_graphs_vec[34], pos, font_size=10, font_family='sans-serif')
```

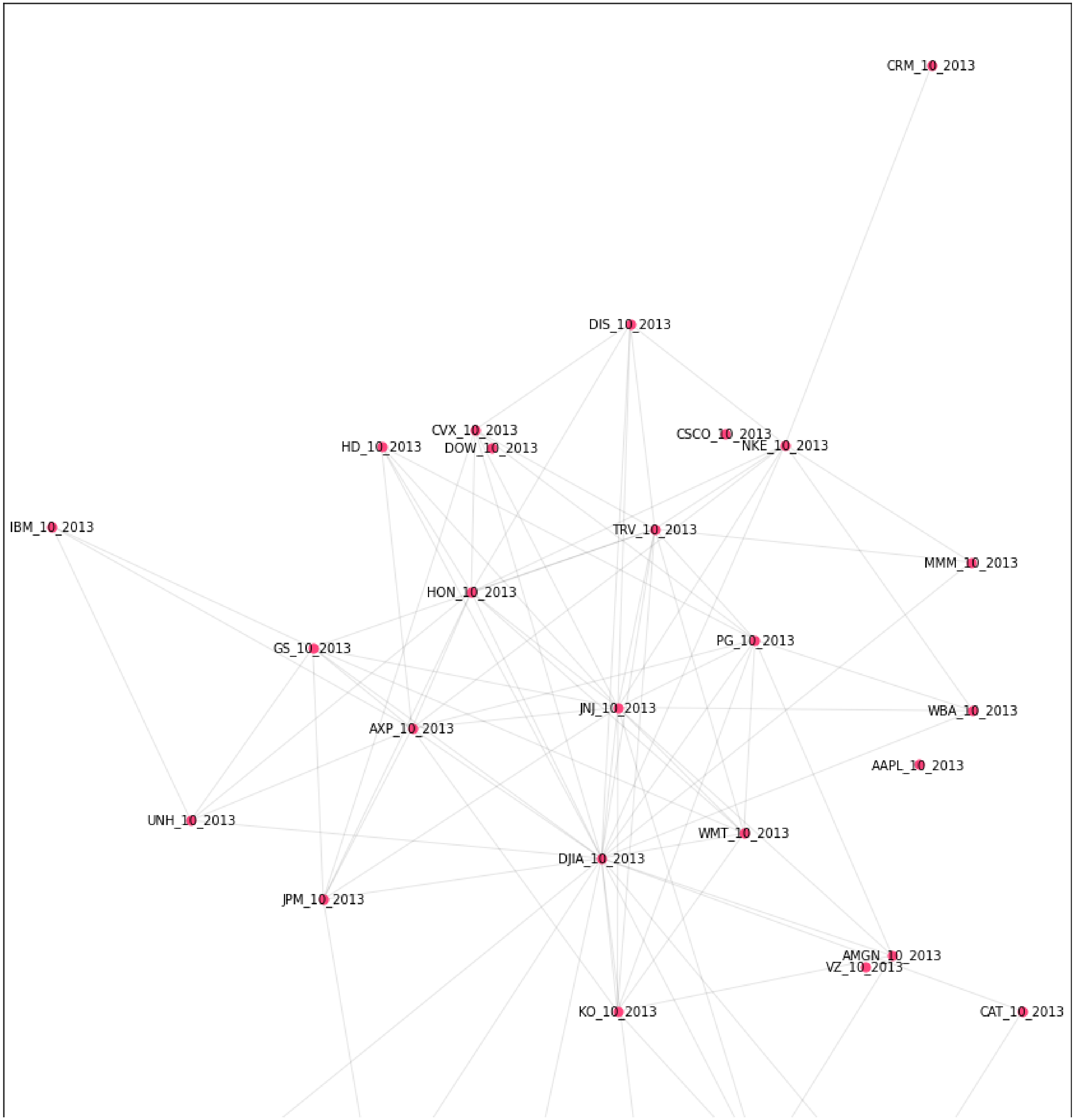
```
100%|██████████| 89/89 [00:00<00:00, 2660.24it/s]
100%|██████████| 16/16 [00:00<00:00, 1502.93it/s]
100%|██████████| 105/105 [00:00<00:00, 4951.28it/s]
100%|██████████| 52/52 [00:00<00:00, 2940.95it/s]
100%|██████████| 4/4 [00:00<00:00, 1234.25it/s]
100%|██████████| 52/52 [00:00<00:00, 3566.88it/s]
100%|██████████| 465/465 [00:00<00:00, 4900.32it/s]
0it [00:00, ?it/s]
100%|██████████| 56/56 [00:00<00:00, 2756.46it/s]
100%|██████████| 46/46 [00:00<00:00, 2128.24it/s]
100%|██████████| 102/102 [00:00<00:00, 3299.27it/s]
100%|██████████| 51/51 [00:00<00:00, 3171.71it/s]
100%|██████████| 12/12 [00:00<00:00, 2186.34it/s]
100%|██████████| 51/51 [00:00<00:00, 4247.94it/s]
100%|██████████| 465/465 [00:00<00:00, 5174.83it/s]
0it [00:00, ?it/s]
100%|██████████| 128/128 [00:00<00:00, 3225.90it/s]
100%|██████████| 115/115 [00:00<00:00, 3151.47it/s]
100%|██████████| 243/243 [00:00<00:00, 4188.51it/s]
100%|██████████| 121/121 [00:00<00:00, 4256.07it/s]
100%|██████████| 32/32 [00:00<00:00, 2731.17it/s]
100%|██████████| 121/121 [00:00<00:00, 4456.58it/s]
100%|██████████| 465/465 [00:00<00:00, 4890.77it/s]
0it [00:00, ?it/s]
```

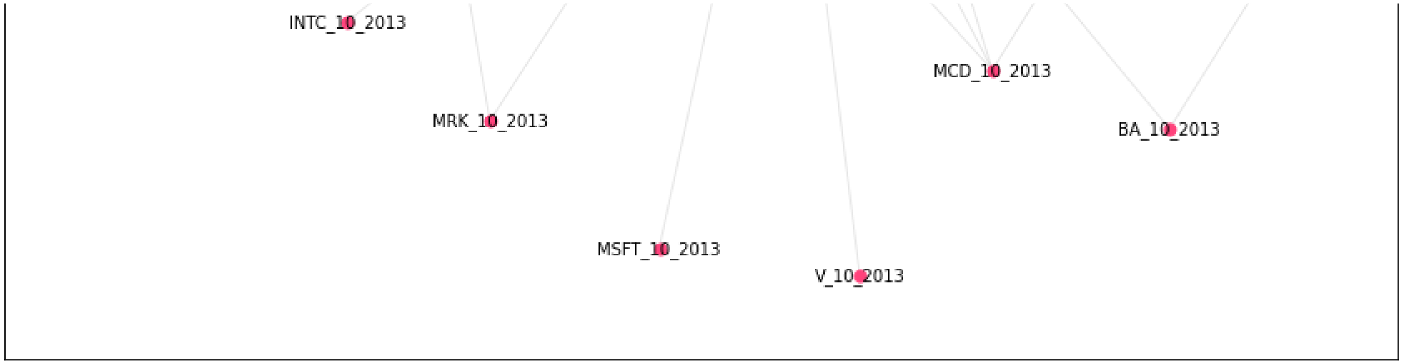



MMM_07_2019
DIS_07_2019
KO_07_2019
JPM_07_2019
IBM_07_2019
INTC_07_2019
UNH_07_2019
GS_07_2019
MRK_07_2019
WMT_07_2019
AMGN_07_2019
MSFT_07_2019
DJIA_07_2019
HON_07_2019
AXP_07_2019
AAPL_07_2019
TRV_07_2019
NKE_07_2019
V_07_2019
HD_07_2019
VZ_07_2019
JNJ_07_2019

Stock Price Correlations Graph

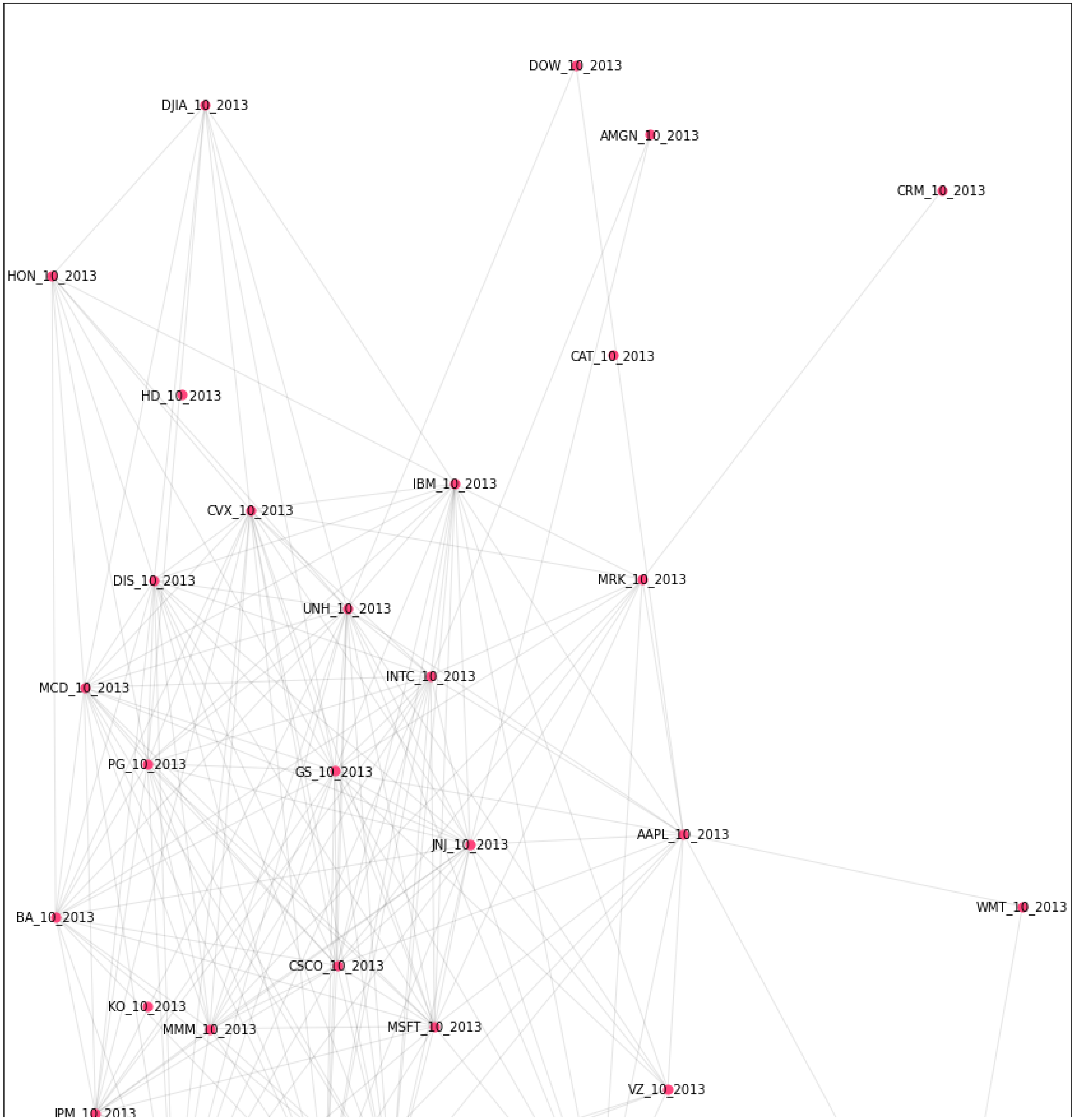
```
node_count =0
for node in price_graphs_vec[11].nodes():
    #set the node name as the key and the label as its value
    node_labels[node] = node

    node_count+=1

pos = nx.kamada_kawai_layout(price_graphs_vec[11])
fig, axs = plt.subplots(1, 1, figsize=(15,20))

el = nx.draw_networkx_edges(price_graphs_vec[11], pos, alpha=0.1, ax=axs)
nl = nx.draw_networkx_nodes(price_graphs_vec[11], pos, nodelist=price_graphs_vec[11].nodes, node_color='#FF427b',
                            node_size=50, ax=axs)
ll = nx.draw_networkx_labels(price_graphs_vec[11], pos, font_size=10, font_family='sans-serif')
```

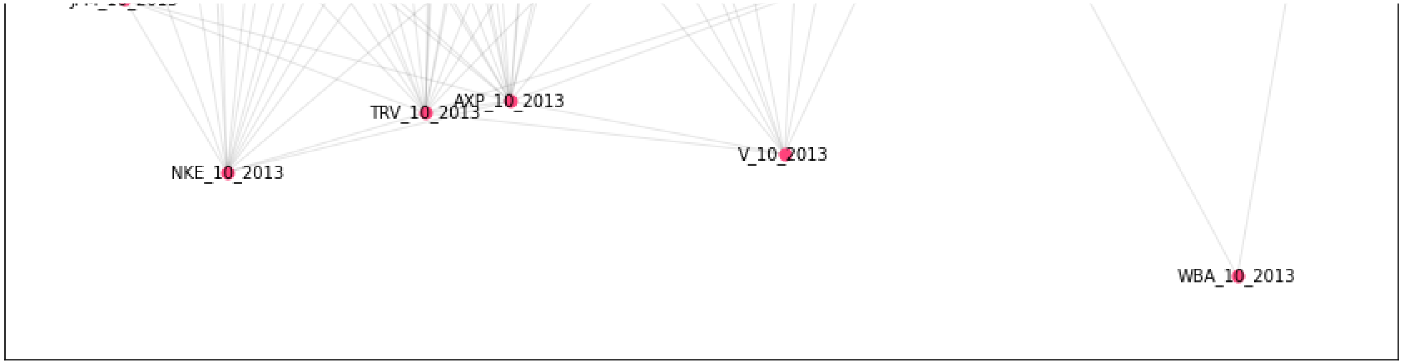

CRM_10_2013
DIS_10_2013
CVX_10_2013
HD_10_2013
DOW_10_2013
CSCO_10_2013
NKE_10_2013
IBM_10_2013
TRV_10_2013
MMM_10_2013
HON_10_2013
GS_10_2013
PG_10_2013
JNJ_10_2013
WBA_10_2013
AXP_10_2013
AAPL_10_2013
UNH_10_2013
WMT_10_2013
DJIA_10_2013
JPM_10_2013
AMGN_10_2013
VZ_10_2013
KO_10_2013
CAT_10_2013



News Article Co-Mentions Graph

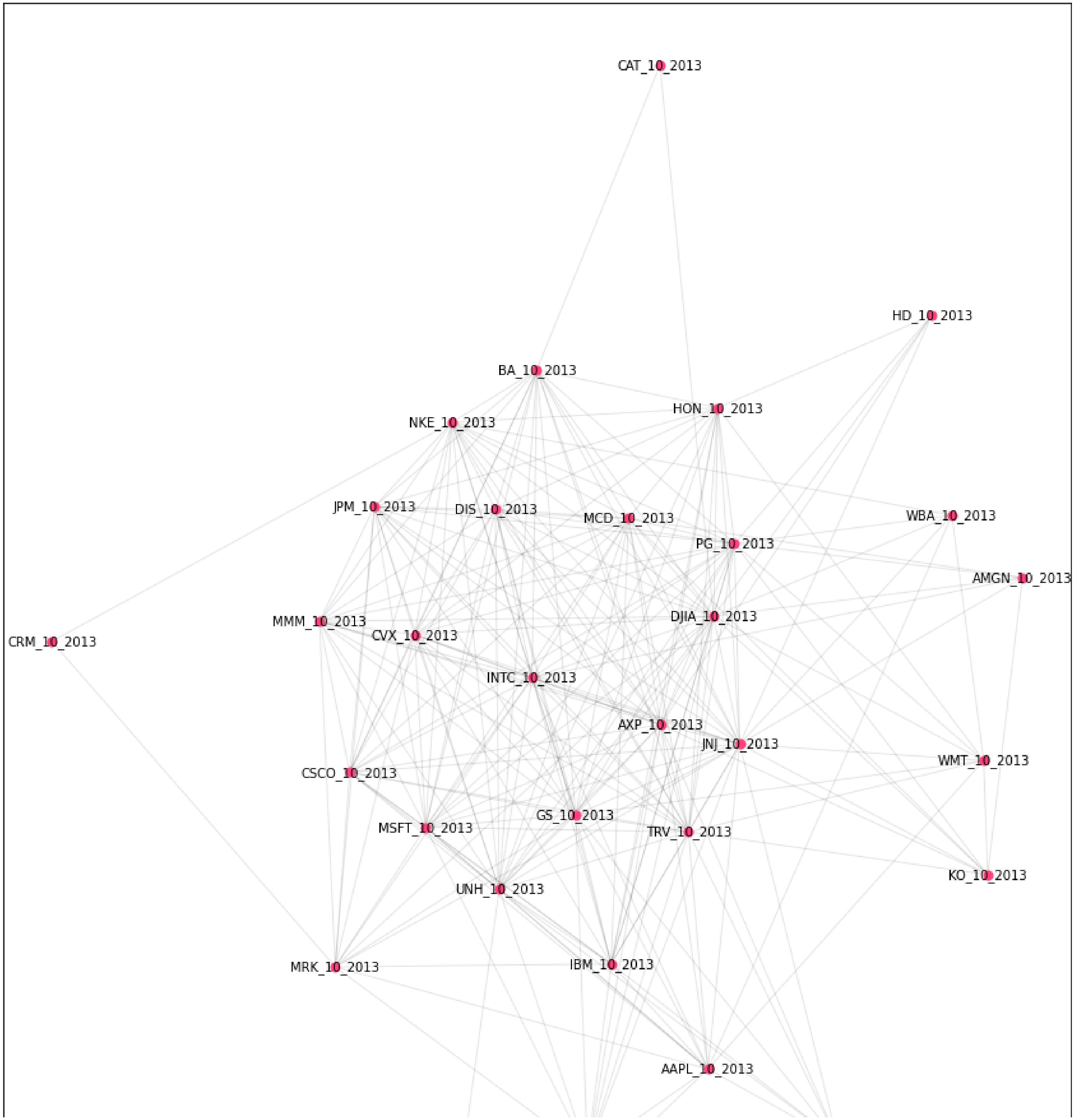
```
node_count =0
for node in news_graphs_vec[11].nodes():
    #set the node name as the key and the label as its value
    node_labels[node] = node

    node_count+=1

pos = nx.kamada_kawai_layout(news_graphs_vec[11])
fig, axs = plt.subplots(1, 1, figsize=(15,20))

el = nx.draw_networkx_edges(news_graphs_vec[11], pos, alpha=0.1, ax=axs)
nl = nx.draw_networkx_nodes(news_graphs_vec[11], pos, nodelist=news_graphs_vec[11].nodes, node_color='#FF427b',
                            node_size=50, ax=axs)
ll = nx.draw_networkx_labels(news_graphs_vec[11], pos, font_size=10, font_family='sans-serif')
```

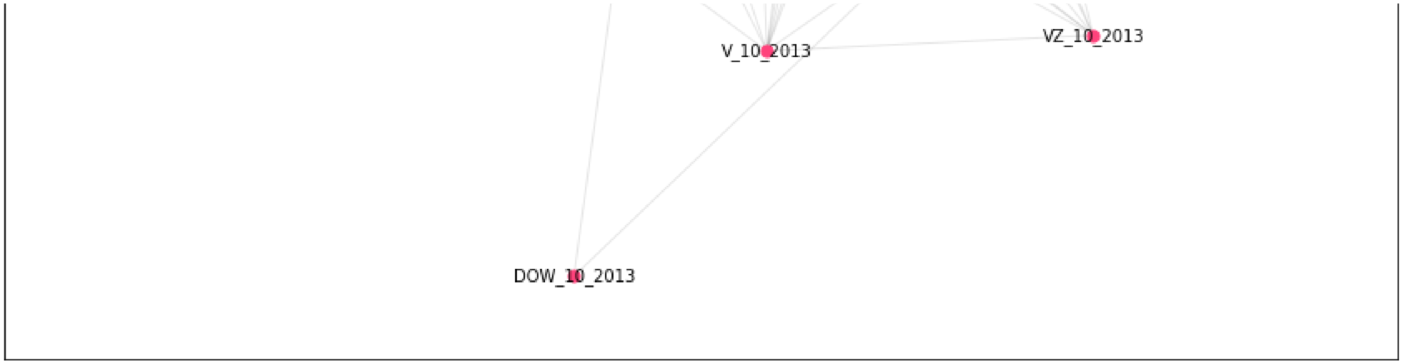

DOW_10_2013
DJIA_10_2013
AMGN_10_2013
CRM_10_2013
HON_10_2013
CAT_10_2013
HD_10_2013
IBM_10_2013
CVX_10_2013
DIS_10_2013
MRK_10_2013
UNH_10_2013
INTC_10_2013
MCD_10_2013
PG_10_2013
GS_10_2013
AAPL_10_2013
JNJ_10_2013
WMT_10_2013
BA_10_2013
CSCO_10_2013
KO_10_2013
MMM_10_2013
MSFT_10_2013
VZ_10_2013



Combined Graph- All edges in price and news graphs

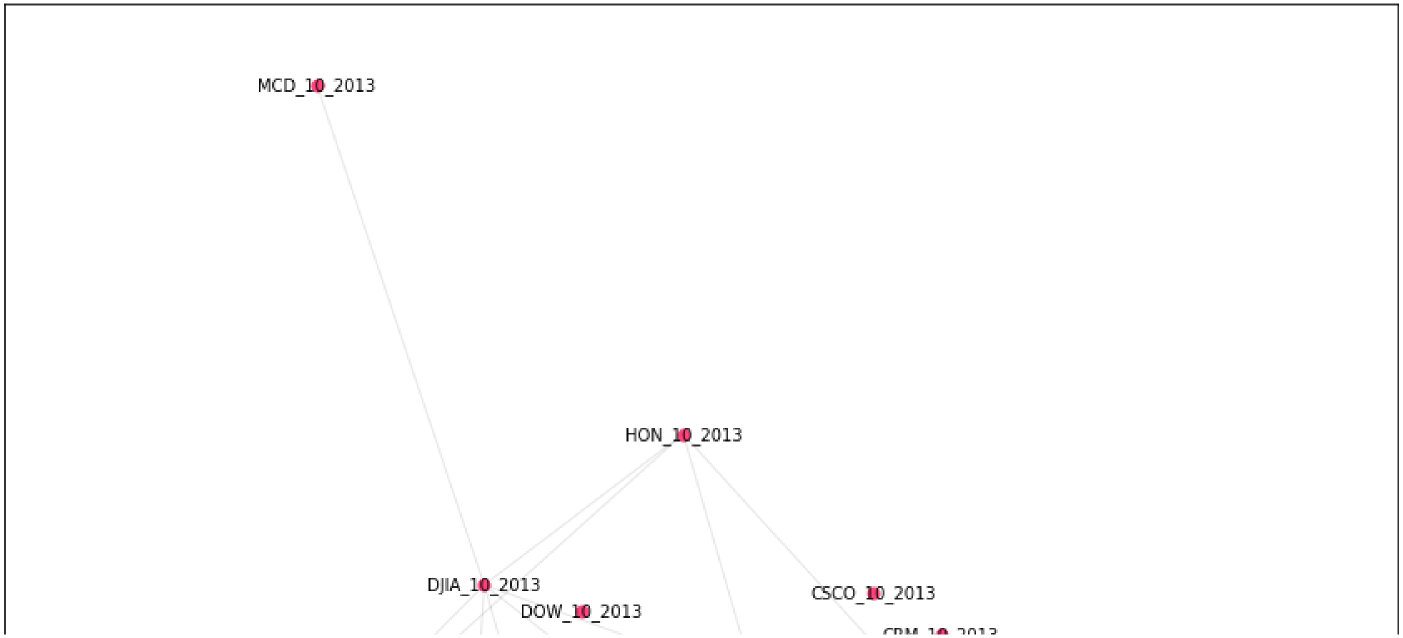
```
node_count =0
for node in comb1_graphs_vec[11].nodes():
    #set the node name as the key and the label as its value
    node_labels[node] = node

    node_count+=1

pos = nx.kamada_kawai_layout(comb1_graphs_vec[11])
fig, axs = plt.subplots(1, 1, figsize=(15,20))

el = nx.draw_networkx_edges(comb1_graphs_vec[11], pos, alpha=0.1, ax=axs)
nl = nx.draw_networkx_nodes(comb1_graphs_vec[11], pos, nodelist=comb1_graphs_vec[11].nodes, node_color='#FF427b',
                            node_size=50, ax=axs)
ll = nx.draw_networkx_labels(comb1_graphs_vec[11], pos, font_size=10, font_family='sans-serif')
```


CAT_10_2013
HD_10_2013
BA_10_2013
HON_10_2013
NKE_10_2013
JPM_10_2013
DIS_10_2013
MCD_10_2013
WBA_10_2013
PG_10_2013
AMGN_10_2013
DJIA_10_2013
MMM_10_2013
CVX_10_2013
CRM_10_2013
INTC_10_2013
AXP_10_2013
JNJ_10_2013
WMT_10_2013
CSCO_10_2013
GS_10_2013
MSFT_10_2013
TRV_10_2013
KO_10_2013
UNH_10_2013
IBM_10_2013
MRK_10_2013
AAPL_10_2013



Combined Graph- Common edges between price and news graphs

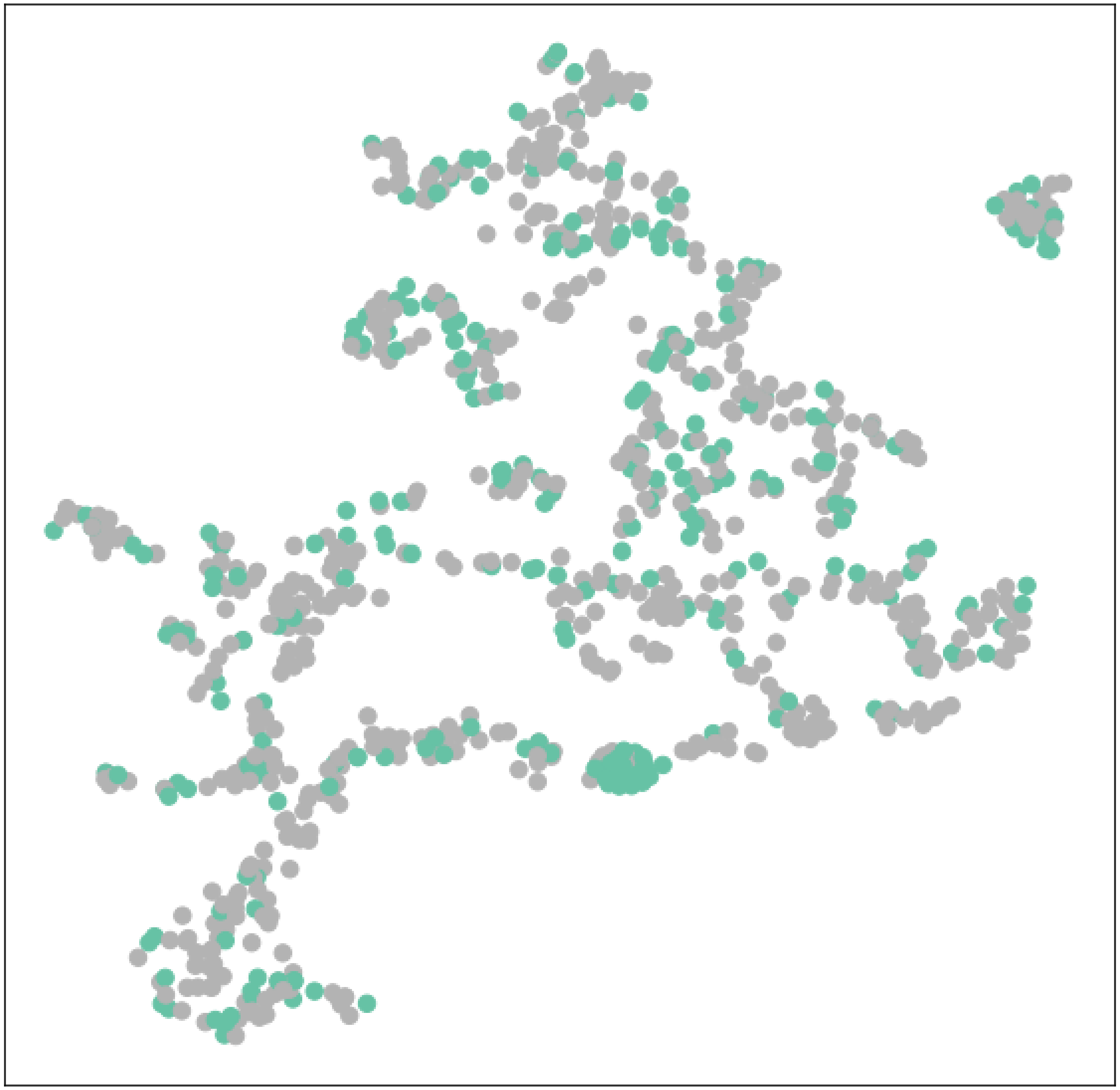
```
node_count =0
for node in comb3_graphs_vec[11].nodes():
    #set the node name as the key and the label as its value
    node_labels[node] = node

    node_count+=1

pos = nx.kamada_kawai_layout(comb3_graphs_vec[11])
fig, axs = plt.subplots(1, 1, figsize=(15,20))

el = nx.draw_networkx_edges(comb3_graphs_vec[11], pos, alpha=0.1, ax=axs)
nl = nx.draw_networkx_nodes(comb3_graphs_vec[11], pos, nodelist=comb3_graphs_vec[11].nodes, node_color='#FF427b',
                            node_size=50, ax=axs)
ll = nx.draw_networkx_labels(comb3_graphs_vec[11], pos, font_size=10, font_family='sans-serif')
```



Building Graph Convolutional Network for Node Attribute Prediction

```
from torch_geometric.nn import GCNConv
from torch.nn import Linear
import torch.nn.functional as F
import torch_geometric.data as tgd

%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

def visualize(h, color):
    z = TSNE(n_components=2).fit_transform(out.detach().cpu().numpy())

    plt.figure(figsize=(10,10))
    plt.xticks([])
    plt.yticks([])

    plt.scatter(z[:, 0], z[:, 1], s=70, c=color, cmap="Set2")
```

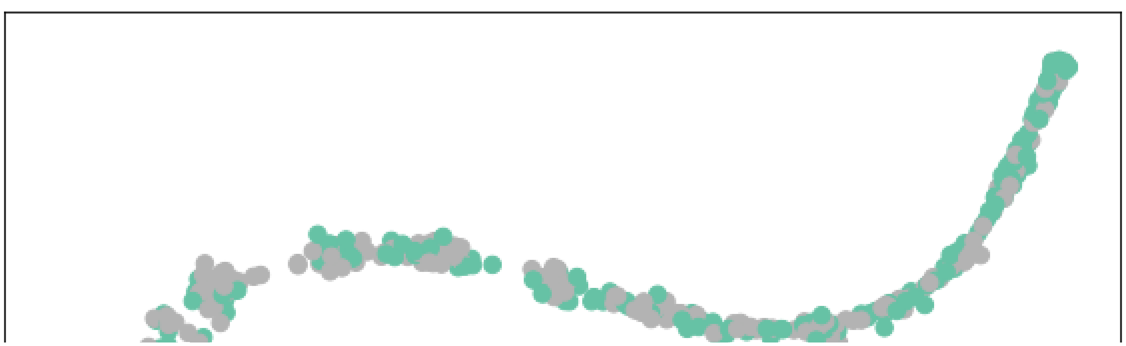
```
    plt.show()

class GCN_Mult(torch.nn.Module):
    def __init__(self, hidden_channels, num_feats, seed_num):
        super(GCN_Mult, self).__init__()
        torch.manual_seed(seed_num)
        num_labels=2
        self.conv1 = GCNConv(num_feats, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, num_labels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return x


nx_gvec=[news_graphs_vec, price_graphs_vec, comb1_graphs_vec, comb2_graphs_vec, comb3_graphs_vec, comb4_graphs_vec, conn_gra

pyt_vec=[None]*len(nx_gvec)
train_vec=[None]*len(nx_gvec)
test_vec=[None]*len(nx_gvec)
#out_vec=[None]*len(nx_gvec)
for gtype_index in range(0, len(nx_gvec), 1):
  pyt_vec[gtype_index]=list(map(tgu.from_networkx, nx_gvec[gtype_index]))
  train_vec[gtype_index]=tgd.Batch.from_data_list(pyt_vec[gtype_index][0:28])
  test_vec[gtype_index]=tgd.Batch.from_data_list(pyt_vec[gtype_index][28:35])
  #model = GCN_Mult(hidden_channels=16, num_feats=train_vec[gtype_index].num_features).double()
  #out_vec[gtype_index]=model(train_vec[gtype_index].x.double(), train_vec[gtype_index].edge_index)

#Experimental diff- we don't have 2020 data, need to either pull it (possible) or discard
#should use 2019/2020 as validation, likely for hyper parameters
#I am not sure I will be able to run find the best parameters via a hyperloop given the time we have
#fix classifier to be average price in next 3-month period

model = GCN_Mult(hidden_channels=16, num_feats=train_vec[0].num_features, seed_num=12345).double()
out = model(train_vec[0].x.double(), train_vec[0].edge_index)
visualize(out, color=train_vec[0].y)
```

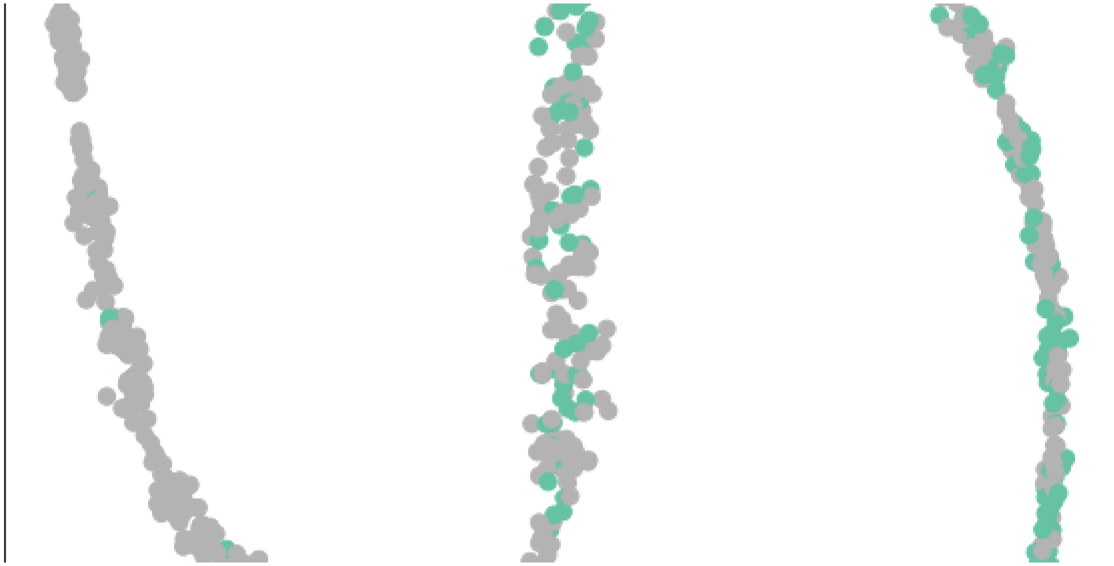


Training the Model and Testing Accuracy

```
from IPython.display import Javascript  # Restrict height of output cell.
from sklearn.model_selection import ShuffleSplit
import random
#num = random.randrange(10000, 99999)
#num
display(Javascript('''google.colab.output.setIframeHeight(0, true, {maxHeight: 300})'''))

test_acc_df=pd.DataFrame(np.zeros( (10, len(pyt_vec) )))
train_acc_df=pd.DataFrame(np.zeros( (10, len(pyt_vec) )))
#Using 10 random seeds

for samp_num in range(0,10,1):

  test_acc_vec=['None']*len(pyt_vec)
  train_acc_vec=['None']*len(pyt_vec)
  rand_num=random.randrange(10000, 99999)

  for gvec_ind in range(0,len(pyt_vec),1):


    model = GCN_Mult(hidden_channels=16,num_feats=train_vec[gvec_ind].num_features, seed_num=rand_num).double()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
    criterion = torch.nn.CrossEntropyLoss()
    #smoothL1 loss if we go to regression loss. investigate other losses for classification
    #potential sigmoid layer as well

    def train():
          model.train()
          optimizer.zero_grad()  # Clear gradients.
          out = model(train_vec[gvec_ind].x.double(), train_vec[gvec_ind].edge_index)  # Perform a single forward pass.
          loss = criterion(out, train_vec[gvec_ind].y.long())  # Compute the loss solely based on the training nodes.
          loss.backward()  # Derive gradients.
          optimizer.step()  # Update parameters based on gradients.
          return loss

    def test():
          model.eval()
          out_test = model(test_vec[gvec_ind].x.double(), test_vec[gvec_ind].edge_index).double()
          pred_test = out_test.argmax(dim=1)  # Use the class with highest probability.
```

```
        test_correct = pred_test == test_vec[gvec_ind].y.double()  # Check against ground-truth labels.
        test_acc = int(test_correct.sum()) / len(test_vec[gvec_ind].y)  # Derive ratio of correct predictions.

        out_train = model(train_vec[gvec_ind].x.double(), train_vec[gvec_ind].edge_index).double()
        pred_train = out_train.argmax(dim=1)  # Use the class with highest probability.
        train_correct = pred_train == train_vec[gvec_ind].y.double()  # Check against ground-truth labels.
        train_acc = int(train_correct.sum()) / len(train_vec[gvec_ind].y)  # Derive ratio of correct predictions.

        tt_acc=[test_acc, train_acc]
        return tt_acc
  if gvec_ind==5:
    for epoch in range(1, 201):
        loss = train()

    out = model(train_vec[gvec_ind].x.double(), train_vec[gvec_ind].edge_index).double()
    visualize(out, color=train_vec[gvec_ind].y)
        #print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}')
  else:
    for epoch in range(1, 201):
        loss = train()
        #print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}')

  tt_acc = test()
  test_acc_vec[gvec_ind]=round(tt_acc[0],4)
  train_acc_vec[gvec_ind]=round(tt_acc[1],4)

test_acc_df.iloc[samp_num]=test_acc_vec
train_acc_df.iloc[samp_num]=train_acc_vec
#print(f'Test Accuracy: {test_acc:.4f}')
```





```
graph_names_vec=['Price Graph', 'News Graph', 'All Edges', 'All Edges 50% Sample',
                 'Common Edges', 'Common Edges + 50% Sample', 'Fully Connected Graph', 'Fully Disconnected Graph', 'Naive Cl
test_acc_vec.append(0.6455)
df_results=pd.DataFrame()
df_results['Prediction Tool']=graph_names_vec
df_results['Mean Test Accuracy']=test_acc_df.mean(axis=0)
df_results['Mean Train Accuracy']=train_acc_df.mean(axis=0)
df_results.iloc[8]=['Naive Classifier', 0.6455, 0.7016]
df_results
#sum(test_vec[3].y)/len(test_vec[3].y)
#benchmark is 65.44% currently for test, 70.16% for train
```

	Prediction Tool	Mean Test Accuracy	Mean Train Accuracy
0	Price Graph	0.64980	0.73006
1	News Graph	0.65900	0.72674

```
0    0.64655
1    0.66176
2    0.62442
3    0.65440
4    0.66728
5    0.64887
6    0.57556
7    0.65992
dtype: float64
```

Running Experiment for Financial Data

```
# wiki  = pd.read_html('https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average#Components')
# wiki_table  = wiki[1]
# symbols = (wiki_table.Symbol.values.tolist()) + ['DJIA']
# df = pd.DataFrame(symbols, columns=['Symbol'])
# start_dates = pd.date_range(start='2011-01-01', end='2019-12-01', freq='MS')
# end_dates = pd.date_range(start='2011-01-31', end='2019-12-31', freq='M')
# graphs_vec=[]

# pull_start = '2011-01-01'
# pull_end  = '2019-12-31'
# df = pd.DataFrame(symbols, columns=['Symbol'])
# symbols = sorted(symbols)

# for i, symbol in enumerate(symbols):
#     try:
#         df = web.DataReader(symbol,'yahoo', pull_start, pull_end)
#         df = df[['Adj Close', 'Volume']]
```