

# **GRAPHICAL NETWORK ANALYSIS & ITS APPLICATION ON STOCK RETURNS DATA**



**PRINCETON UNIVERSITY**

**JULY 2020**



# Outline

---

## 1. Introduction

- Datasets
  - Basic Summary of Stock Data
  - Stock Return data (from CRSP via WRDS)
  - T-Bill Return data (from FRED)

## 2. Data Preprocessing

- Data cleaning and preparation

## 3. Graphical Network Analysis

- Algorithm explanation
  - Sparse Inverse Covariance Estimation via Graphical Lasso
  - Clustering via Affinity Propagation
  - Manifold Learning via Multidimensional scaling (MDS)
- Detailed codes written in *Graphical\_Analysis\_functions.py*
- Graphical Network Analysis (Control File)

# **Introduction**

# Dataset: Basic Summary of Stock Data

- 4 sectors:
  - Bank
  - Health
  - Energy
  - Tech
- 22 companies in total

	Ticker	Name	Sector	PERMNO
0	FRC	First Republic	Bank	12448
1	PNC	PNC Financial	Bank	60442
2	USB	US Bancorp	Bank	66157
3	JPM	JPMorgan Chase	Bank	47896
4	BAC	Bank of America	Bank	59408
5	C	Citigroup	Bank	70519
6	RY	Royal Bank of Canada	Bank	82654
7	WFC	Wells Fargo	Bank	38703
8	GS	Goldman Sachs	Bank	86868
9	MS	Morgan Stanley	Bank	69032
10	HSBC	HSBC Holdings plc	Bank	87033
11	JNJ	Johnson & Johnson	Health	22111
12	PFE	Pfizer	Health	21936
13	MRK	Merck & Co.	Health	22752
14	XOM	Exxon	Energy	11850
15	RDS	Royal Dutch Shell	Energy	90794
16	CVS	Chevron	Energy	14541
17	BP	BP	Energy	29890
18	AAPL	Apple	Tech	14593
19	GOOG	Alphabet	Tech	90319
20	MSFT	Microsoft	Tech	10107
21	BABA	Alibaba	Tech	14888



## Dataset: Stock Return Data

- Downloaded from CRSP via WRDS
  - Frequency: Daily
  - Date Range: 1999-12-31 to 2020-06-30
  - “RET” is the “Holding Period Return”

Out[3]:

	PERMNO	date	TICKER	COMNAM	CUSIP	RET
0	10107	1999/12/31	MSFT	MICROSOFT CORP	59491810	-0.007439
1	10107	2000/01/03	MSFT	MICROSOFT CORP	59491810	-0.001606
2	10107	2000/01/04	MSFT	MICROSOFT CORP	59491810	-0.033780
3	10107	2000/01/05	MSFT	MICROSOFT CORP	59491810	0.010544
4	10107	2000/01/06	MSFT	MICROSOFT CORP	59491810	-0.033498
...	...	...	...	...	...	...
104436	90794	2020/06/24	RDS	ROYAL DUTCH SHELL PLC	78025920	-0.049641
104437	90794	2020/06/25	RDS	ROYAL DUTCH SHELL PLC	78025920	0.023249
104438	90794	2020/06/26	RDS	ROYAL DUTCH SHELL PLC	78025920	-0.028917
104439	90794	2020/06/29	RDS	ROYAL DUTCH SHELL PLC	78025920	0.016104
104440	90794	2020/06/30	RDS	ROYAL DUTCH SHELL PLC	78025920	-0.022428

104441 rows × 6 columns



## Dataset: 3-Month T-Bill Data

- Downloaded from FRED  
(<https://fred.stlouisfed.org/series/WTB3MS#0>)
  - Frequency: Weekly, Ending on Friday
  - Date Range: max to 2020-06-26
  - Units: percent per annum

	DATE	WTB3MS
0	1954-01-08	1.30
1	1954-01-15	1.28
2	1954-01-22	1.11
3	1954-01-29	1.01
4	1954-02-05	0.99
...	...	...
3464	2020-05-29	0.15
3465	2020-06-05	0.15
3466	2020-06-12	0.17
3467	2020-06-19	0.17
3468	2020-06-26	0.15

3469 rows × 2 columns

# **Data Preprocessing**





# Data Cleaning

---

- We will do the following steps to clean data and make it ready for our graphical network analysis.
  1. Read and edit (Stock Returns) data from CRSP
  2. Convert daily dataset to weekly dataset
  3. Read and edit T-Bill data
  4. Save and export the (cleaned) data to an Excel file
  
- Detailed codes written in the notebook *Part 1 – Data Preprocessing.ipynb*



# 1. Read and Edit (Stock Returns) data from CRSP

Out[3]:

	PERMNO	date	TICKER	COMNAM	CUSIP	RET
0	10107	1999/12/31	MSFT	MICROSOFT CORP	59491810	-0.007439
1	10107	2000/01/03	MSFT	MICROSOFT CORP	59491810	-0.001606
2	10107	2000/01/04	MSFT	MICROSOFT CORP	59491810	-0.033780
3	10107	2000/01/05	MSFT	MICROSOFT CORP	59491810	0.010544
4	10107	2000/01/06	MSFT	MICROSOFT CORP	59491810	-0.033498
...	...	...	...	...	...	...
104436	90794	2020/06/24	RDS	ROYAL DUTCH SHELL PLC	78025920	-0.049641
104437	90794	2020/06/25	RDS	ROYAL DUTCH SHELL PLC	78025920	0.023249
104438	90794	2020/06/26	RDS	ROYAL DUTCH SHELL PLC	78025920	-0.028917
104439	90794	2020/06/29	RDS	ROYAL DUTCH SHELL PLC	78025920	0.016104
104440	90794	2020/06/30	RDS	ROYAL DUTCH SHELL PLC	78025920	-0.022428

104441 rows x 6 columns

```
In [5]: ## Check the unique tickers in the dataset
# Note that there is more tickers than the ones we specified in the summary information
df.TICKER.unique()
```

```
Out[5]: array(['MSFT', 'XOM', 'FRC', 'CHV', 'CVX', 'AAPL', 'BABA', 'PFE', 'JNJ',
        'MRK', 'BPA', 'BP', 'WFC', 'CMB', 'JPM', 'BAC', 'PNC', 'USB',
        'MWD', 'MS', 'C', 'RY', 'GS', 'HBC', 'HSBC', 'GOOG', 'GOOGL',
        'RDS'], dtype=object)
```

	Ticker	Name	Sector	PERMNO
0	FRC	First Republic	Bank	12448
1	PNC	PNC Financial	Bank	60442
2	USB	US Bancorp	Bank	66157
3	JPM	JPMorgan Chase	Bank	47896
4	BAC	Bank of America	Bank	59408
5	C	Citigroup	Bank	70519
6	RY	Royal Bank of Canada	Bank	82654
7	WFC	Wells Fargo	Bank	38703
8	GS	Goldman Sachs	Bank	86868
9	MS	Morgan Stanley	Bank	69032
10	HSBC	HSBC Holdings plc	Bank	87033
11	JNJ	Johnson & Johnson	Health	22111
12	PFE	Pfizer	Health	21936
13	MRK	Merck & Co.	Health	22752
14	XOM	Exxon	Energy	11850
15	RDS	Royal Dutch Shell	Energy	90794
16	CVS	Chevron	Energy	14541
17	BP	BP	Energy	29890
18	AAPL	Apple	Tech	14593
19	GOOG	Alphabet	Tech	90319
20	MSFT	Microsoft	Tech	10107
21	BABA	Alibaba	Tech	14888

# 1. Read and Edit (Stock Returns) Data from CRSP

```
In [6]: ## Rearrange the data: separate by the specified tickers and extract "RET" column only
RET_DFS = []
for i in range(len(firms_CRSP.PERMNO)):
    temp = firms_CRSP.PERMNO[i]
    firmTic = firms_CRSP.Ticker[i]
    df_temp = df[df.PERMNO == temp].copy()
    df_temp['date'] = pd.to_datetime(df_temp['date'])

    RET_data = pd.DataFrame(columns=['RET'], index=df_temp.date)
    RET_data['RET'] = pd.to_numeric(df_temp['RET'].values, errors='coerce')
    RET_DFS.append(RET_data)

    # Output the firm (ticker) and its shape
    print(firmTic, ': ', RET_data.shape)
```

```
In [7]: ## Put the return data together

len_RET_DFS = np.array([len(temp) for temp in RET_DFS])

Names = firms_CRSP.Ticker
data = pd.DataFrame(columns=Names, index=RET_DFS[np.argmax(len_RET_DFS)].index)
for i in range(len(Names)):
    data[Names[i]] = RET_DFS[i]
```

Out[8]:

Ticker date	FRC	PNC	USB	JPM	BAC	C	RY	WFC	GS	MS	...	PFE	MRK	XOM
1999-12-31	NaN	-0.005587	0.005277	-0.004804	-0.009864	0.006780	0.001418	-0.004615	0.021695	0.010619	...	-0.007648	-0.009217	0.001554
2000-01-03	NaN	-0.051966	-0.065617	-0.060338	-0.034869	-0.050505	-0.031161	-0.032457	-0.062376	-0.054291	...	-0.017341	0.006512	-0.029480
2000-01-04	NaN	-0.031111	-0.033708	-0.023493	-0.059355	-0.042553	-0.001462	-0.051118	-0.062987	-0.074074	...	-0.037255	-0.037893	-0.017586
2000-01-05	NaN	0.004587	-0.002907	-0.006173	0.010974	0.003704	-0.005857	-0.008418	-0.046828	-0.036500	...	0.016293	0.042267	0.054516
2000-01-06	NaN	0.047184	0.032070	0.014197	0.071913	0.065191	0.008837	0.044143	0.042789	0.019201	...	0.036072	0.008295	0.049383
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-06-24	-0.026556	-0.040379	-0.044819	-0.033391	-0.039532	-0.040415	-0.015222	-0.040764	-0.033011	-0.019628	...	-0.017699	-0.016209	-0.047109
2020-06-25	0.024723	0.029696	0.032004	0.034862	0.038219	0.036804	0.014863	0.047856	0.045854	0.039191	...	0.004039	0.009226	0.015056
2020-06-26	-0.040303	-0.051855	-0.052300	-0.054818	-0.063511	-0.058846	-0.023433	-0.074169	-0.086480	-0.035663	...	-0.008663	-0.018023	-0.034315
2020-06-29	0.013389	0.018825	0.019689	0.004428	0.010367	0.014320	0.010948	0.014207	0.022623	0.004038	...	0.018727	0.012369	0.016048
2020-06-30	0.007414	0.023145	0.012930	0.011398	0.015391	0.016107	0.005785	-0.003891	0.021450	0.022439	...	0.001838	0.015896	0.009025

5158 rows × 22 columns

## 2. Convert Daily Dataset to Weekly Dataset

### 2. Convert Daily dataset to Weekly Dataset

```
In [11]: ## Reorganize the data and convert it to weekly frequency

WEEKLY_DFS = []

for i in range(len(data.columns)):
    firm = data.columns[i]
    RET = data[firm].dropna(axis=0)
    RET_weekly = (RET+1).resample('W').prod() - 1 # Convert daily data to weekly
    WEEKLY_DFS.append(RET_weekly)

len_WEEKLY_DFS = np.array([len(temp) for temp in WEEKLY_DFS])
Names = data.columns
data_weekly = pd.DataFrame(columns=Names, index=WEEKLY_DFS[np.argmax(len_WEEKLY_DFS)].index)
for i in range(len(Names)):
    data_weekly[Names[i]] = WEEKLY_DFS[i]
```

Out[12]: data\_weekly

Out[12]:

	FRC	PNC	USB	JPM	BAC	C	RY	WFC	GS	MS	...	MRK	XOM	RDS
date														
2000-01-02	NaN	-0.005587	0.005277	-0.004804	-0.009864	0.006780	0.001418	-0.004615	0.021695	0.010619	...	-0.009217	0.001554	NaN
2000-01-09	NaN	-0.032304	-0.049869	-0.058139	-0.029888	-0.030302	-0.042493	-0.032457	-0.123426	-0.113398	...	0.115349	0.054307	NaN
2000-01-16	NaN	0.097264	0.019338	0.015465	0.034659	0.074074	-0.017752	0.065494	0.037093	0.088261	...	-0.010843	-0.013980	NaN
2000-01-23	NaN	-0.061498	-0.094850	-0.012690	-0.093051	-0.026939	-0.035181	-0.121438	0.002128	-0.035047	...	-0.022765	0.014925	NaN
2000-01-30	NaN	0.045583	0.005989	0.051414	0.001367	-0.005537	0.068037	0.058020	0.015318	-0.044340	...	0.054358	-0.072060	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-06-07	0.110382	0.122675	0.191507	0.143049	0.173256	0.228554	0.109022	0.196449	0.109063	0.120588	...	0.019078	0.167364	0.182160
2020-06-14	-0.111648	-0.134890	-0.114703	-0.102130	-0.118463	-0.112299	-0.064237	-0.116829	-0.074064	-0.065617	...	-0.065038	-0.111341	-0.087901
2020-06-21	0.038987	-0.022843	0.036257	-0.020626	0.018967	0.012823	0.010253	-0.013228	-0.000744	0.014045	...	0.020968	-0.025229	-0.014805
2020-06-28	-0.063504	-0.067449	-0.072293	-0.053368	-0.083168	-0.063113	-0.019268	-0.081885	-0.061697	0.002558	...	-0.034790	-0.051327	-0.030347
2020-07-05	0.020902	0.042406	0.032874	0.015876	0.025918	0.030658	0.016796	0.010261	0.044558	0.026568	...	0.028462	0.025218	-0.006685

1071 rows × 23 columns

### 3. Read and edit T-Bill data

	DATE	WTB3MS
0	1954-01-08	1.30
1	1954-01-15	1.28
2	1954-01-22	1.11
3	1954-01-29	1.01
4	1954-02-05	0.99
...	...	...
3464	2020-05-29	0.15
3465	2020-06-05	0.15
3466	2020-06-12	0.17
3467	2020-06-19	0.17
3468	2020-06-26	0.15

3469 rows × 2 columns

```
In [14]: ## Read and Edit data into the right format
file_name = 'WTB3MS.csv'
df = pd.read_csv(file_name)

df['DATE'] = pd.to_datetime(df['DATE'])
df['WTB3MS'] = pd.to_numeric(df['WTB3MS'], errors='coerce')

RET_data = pd.DataFrame(columns=['RET'], index=df.DATE)
RET_data['RET'] = (df['WTB3MS'].values/100 + 1) ** (1/52) - 1 # Edit the unit of the return data

RET_data_weekly = (RET_data['RET']+1).resample('W').prod() - 1 # Convert the data to the same weekly frequency as above
TBill = pd.DataFrame(columns=['T-Bill'], index=RET_data_weekly.index)
TBill['T-Bill'] = RET_data_weekly

TBill
```

Out[14]:

	T-Bill
DATE	
1954-01-10	0.000248
1954-01-17	0.000245
1954-01-24	0.000212
1954-01-31	0.000193
1954-02-07	0.000189
...	...
2020-05-31	0.000029
2020-06-07	0.000029
2020-06-14	0.000033
2020-06-21	0.000033
2020-06-28	0.000029

3469 rows × 1 columns

## 4. Save and export the (cleaned) data to an Excel file

firms\_CRSP

Out[16]:

	Name	Sector
Ticker		
FRC	First Republic	Bank
PNC	PNC Financial	Bank
USB	US Bancorp	Bank
JPM	JPMorgan Chase	Bank
BAC	Bank of America	Bank
C	Citigroup	Bank
RY	Royal Bank of Canada	Bank
WFC	Wells Fargo	Bank
GS	Goldman Sachs	Bank
MS	Morgan Stanley	Bank
HSBC	HSBC Holdings plc	Bank
JNJ	Johnson & Johnson	Health
PFE	Pfizer	Health
MRK	Merck & Co.	Health
XOM	Exxon	Energy
RDS	Royal Dutch Shell	Energy
CVS	Chevron	Energy
BP	BP	Energy
AAPL	Apple	Tech
GOOG	Alphabet	Tech
MSFT	Microsoft	Tech
BABA	Alibaba	Tech
SP500	S&P 500	Market Index

data\_weekly

Out[17]:

	FRC	PNC	USB	JPM	BAC	C	RY	WFC	GS	MS	...	MRK	XOM	RDS
date														
2000-01-09	NaN	-0.032304	-0.049869	-0.058139	-0.029888	-0.030302	-0.042493	-0.032457	-0.123426	-0.113398	...	0.115349	0.054307	NaN
2000-01-16	NaN	0.097264	0.019338	0.015465	0.034659	0.074074	-0.017752	0.065494	0.037093	0.088261	...	-0.010843	-0.013980	NaN
2000-01-23	NaN	-0.061498	-0.094850	-0.012690	-0.093051	-0.026939	-0.035181	-0.121438	0.002128	-0.035047	...	-0.022765	0.014925	NaN
2000-01-30	NaN	0.045583	0.005989	0.051414	0.001367	-0.005537	0.068037	0.058020	0.015318	-0.044340	...	0.054358	-0.072060	NaN
2000-02-06	NaN	-0.028609	0.014881	0.066015	-0.008197	-0.021727	-0.054815	0.000774	-0.026581	0.037512	...	-0.017185	0.025358	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-05-31	0.111717	0.090144	0.079866	0.087626	0.064430	0.086395	0.101580	0.095159	0.098984	0.101420	...	0.056960	0.019506	-0.008995

TBill

Out[18]:

T-Bill	
DATE	
2000-01-09	0.000986
2000-01-16	0.000986
2000-01-23	0.000995
2000-01-30	0.001014
2000-02-06	0.001030
...	...
2020-05-31	0.000029
2020-06-07	0.000029
2020-06-14	0.000033
2020-06-21	0.000033
2020-06-28	0.000029

In [19]: `## Save data as Excel file with multiple sheets`

```
writer = pd.ExcelWriter('CleanedData_Weekly.xlsx', engine='xlsxwriter')

firms_CRSP.to_excel(writer, sheet_name='Firms Info')
data_weekly.to_excel(writer, sheet_name='Stock Returns')
TBill.to_excel(writer, sheet_name='T-Bill')

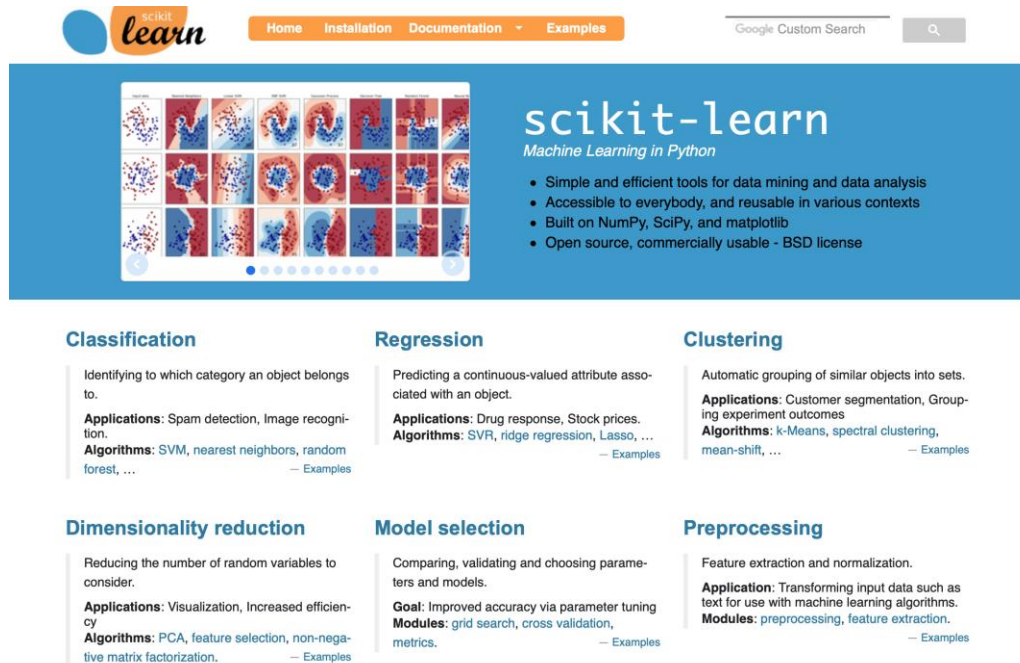
writer.save()
```

# Graphical Network Analysis

---

# scikit-learn Library

- [scikit-learn](https://scikit-learn.org) is a machine learning library in Python
- Involves tools for data analysis and machine learning algorithms



The screenshot shows the scikit-learn website homepage. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. A Google Custom Search bar is also present. The main header features the scikit-learn logo and the tagline "Machine Learning in Python". Below this, a grid of 16 small images displays various machine learning visualizations, such as scatter plots, decision boundaries, and heatmaps. To the right of the grid, a list of bullet points highlights the library's features: simple and efficient tools for data mining and data analysis, accessibility to everybody and reusability in various contexts, being built on NumPy, SciPy, and matplotlib, and being open source with a BSD license. The lower section of the page is organized into six columns, each representing a different machine learning task: Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing. Each column provides a brief description, applications, and algorithms used in that domain.

**scikit-learn**  
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

**Classification**  
Identifying to which category an object belongs to.  
**Applications:** Spam detection, Image recognition.  
**Algorithms:** SVM, nearest neighbors, random forest, ... — Examples

**Regression**  
Predicting a continuous-valued attribute associated with an object.  
**Applications:** Drug response, Stock prices.  
**Algorithms:** SVR, ridge regression, Lasso, ... — Examples

**Clustering**  
Automatic grouping of similar objects into sets.  
**Applications:** Customer segmentation, Grouping experiment outcomes  
**Algorithms:** k-Means, spectral clustering, mean-shift, ... — Examples

**Dimensionality reduction**  
Reducing the number of random variables to consider.  
**Applications:** Visualization, Increased efficiency  
**Algorithms:** PCA, feature selection, non-negative matrix factorization. — Examples

**Model selection**  
Comparing, validating and choosing parameters and models.  
**Goal:** Improved accuracy via parameter tuning  
**Modules:** grid search, cross validation, metrics. — Examples

**Preprocessing**  
Feature extraction and normalization.  
**Application:** Transforming input data such as text for use with machine learning algorithms.  
**Modules:** preprocessing, feature extraction. — Examples





# Algorithm Explanation

---

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

1. **Sparse Inverse Covariance Estimation** for learning the graphical structure
  - Graphical Lasso
2. **Affinity Propagation** for identifying clusters
3. **Manifold Learning** for node visualization
  - Multidimensional scaling (MDS)



# 1. Sparse Inverse Covariance Estimation: Graphical Lasso

---

- Precision Matrix:

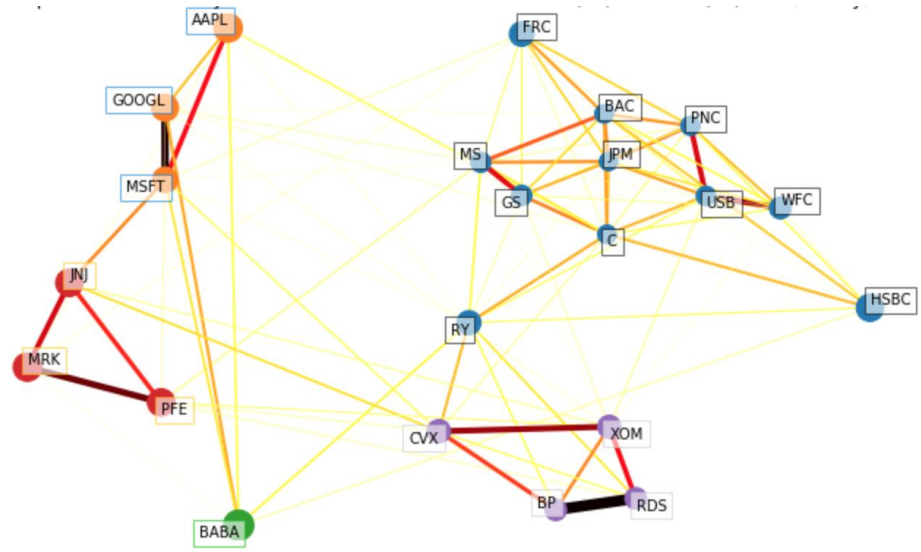
- The matrix inverse of the covariance matrix, often called the precision matrix, is proportional to the partial correlation matrix, and it gives the partial independence relationship among the data
- In other words, if two features are independent conditionally on the others, the corresponding coefficient in the precision matrix will be zero.

- We use sparse inverse covariance estimation to find which stocks are correlated conditionally on the others.

- Specifically, sparse inverse covariance gives us a graph, or a list of connection, which can be used for constructing the graphical structure of the data.

# 1. Graphical Lasso: Overview

- Assumes that dataset fall under Gaussian Distribution
- Determines covariance based on all correlated pairs
- Solves for the precision matrix, or the inverse covariance matrix, through solving an optimization problem
- Contains  $L_1$  penalty to enforce sparsity on the precision matrix





# 1. Graphical Lasso: Optimization Problem

---

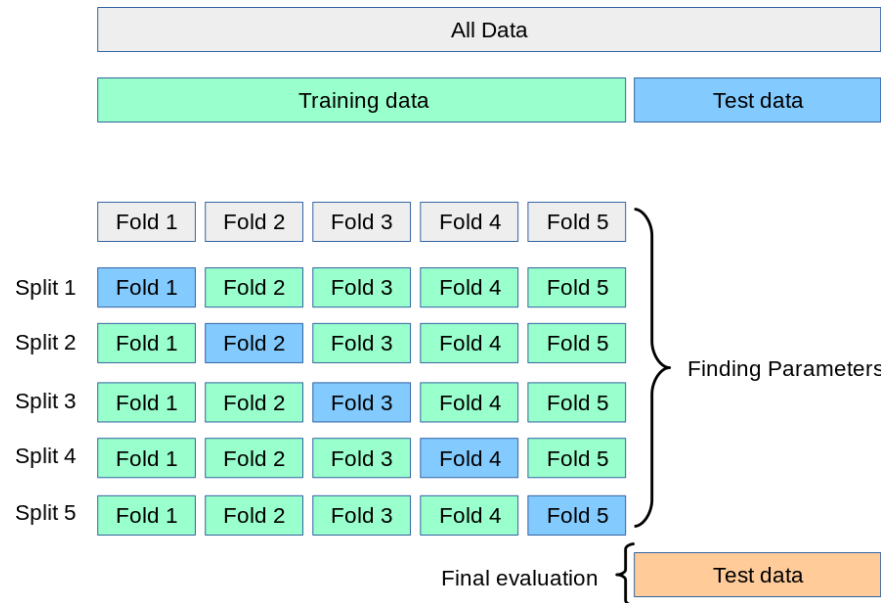
- Semidefinite Programming (SDP)
- Convex Program

$$\min_{\Theta \succeq 0} \text{tr}(S\Theta) - \log \det(\Theta) + \underbrace{\lambda \|\Theta\|_1}_{\text{lasso penalty}}$$

- $\Theta$ : precision matrix
  - $S$ : covariance matrix
  - $\lambda$ : shrinkage tuning parameter
- Solve for " $\Theta$ "
  - We will use [GraphicalLassoCV](#), which uses cross-validation to automatically set the lambda parameter

# 1. Review: Cross-Validation (CV)

- **Cross-validation (CV)** is a popular technique used in machine learning to find the best tuning parameters while lowering the risk of overfitting
- **k-fold CV** splits training data into k smaller sets. It follows the procedure for each k folds:
  - Train the model on k-1 folds of the training data
  - The resulting model is validated on the remaining part of the data



# 1. Example – GraphicalLassoCV.ipynb



## GraphicalLassoCV Example

Last Update: July 22nd, 2020

Reference: <https://scikit-learn.org/stable/modules/generated/sklearn.covariance.GraphicalLasso.html>

```
In [1]: # Import all the necessary packages
import numpy as np
import pandas as pd
from sklearn.covariance import GraphicalLassoCV
from sklearn.covariance import GraphicalLasso

In [2]: # Define true covariance matrix
true_cov = np.array([[0.8, 0.0, 0.2, 0.0],
                     [0.0, 0.4, 0.0, 0.0],
                     [0.2, 0.0, 0.3, 0.1],
                     [0.0, 0.0, 0.1, 0.7]])

# Set seed and generate X from multivariate norm with specified covariance
np.random.seed(0)
X = np.random.multivariate_normal(mean=[0, 0, 0, 0], cov=true_cov, size=200)

In [3]: # True precision matrix calculated from the inverse of true covariance matrix
true_prec = np.linalg.inv(true_cov)
true_prec

Out[3]: array([[ 1.51515152,  0.          , -1.06060606,  0.15151515],
               [ 0.          ,  2.5         ,  0.          ,  0.          ],
               [-1.06060606,  0.          ,  4.24242424, -0.60606061],
               [ 0.15151515,  0.          , -0.60606061,  1.51515152]])
```

# 1. Example – GraphicalLassoCV.ipynb

```
In [4]: # Fit the GraphicalLassoCV model
est = GraphicalLassoCV(max_iter = 1000).fit(X)
```

```
In [5]: # The estimated covariance matrix from GraphicalLassoCV
np.around(est.covariance_, decimals=3)
```

```
Out[5]: array([[0.816, 0.051, 0.22 , 0.017],
               [0.051, 0.364, 0.018, 0.036],
               [0.22 , 0.018, 0.322, 0.094],
               [0.017, 0.036, 0.094, 0.69 ]])
```

```
In [6]: # The estimated precision matrix from GraphicalLassoCV
np.around(est.precision_, decimals=3)
```

```
Out[6]: array([[ 1.521, -0.17 , -1.063,  0.116],
               [-0.17 ,  2.784, -0.   , -0.14 ],
               [-1.063, -0.   ,  3.982, -0.518],
               [ 0.116, -0.14 , -0.518,  1.524]])
```

```
In [7]: # The list of lambdas used in cross validation
est.cv_alphas_
```

```
Out[7]: [0.22813020648178522,
         0.049149163068849436,
         0.026598029308124188,
         0.014394042927746526,
         0.010588866190156311,
         0.008807435274025985,
         0.008488861740795783,
         0.008181811323310077,
         0.007885867219221498,
         0.007789617396298063,
         0.007600627702330532,
         0.007325705577266539,
         0.006093256496935091,
         0.005068150002186581,
         0.00421550355833272,
         0.0022813020648178514,
         0]
```

```
In [8]: # The lambda chosen by cross validation
est.alpha_
```

```
Out[8]: 0.008181811323310077
```

```
In [9]: # The index of the chosen lambda in the list of lambdas
ind_lambda = np.where(est.cv_alphas_ == est.alpha_)[0][0]
ind_lambda
```

```
Out[9]: 7
```



# 1. Example – GraphicalLassoCV.ipynb

Out[13]:

	lambdas	score_fold1	score_fold2	score_fold3	score_fold4	score_fold5	Total_score	Average_score	Num_Zeros
0	0.228130	-4.504907	-4.165553	-4.239870	-4.234588	-4.237572	-21.382489	-4.276498	12
1	0.049149	-4.314145	-4.044966	-4.167367	-4.174215	-4.164886	-20.865579	-4.173116	6
2	0.026598	-4.280548	-4.037276	-4.161012	-4.175371	-4.170713	-20.824920	-4.164984	2
3	0.014394	-4.263416	-4.032820	-4.161703	-4.174424	-4.177955	-20.810318	-4.162064	2
4	0.010589	-4.256945	-4.029620	-4.162758	-4.175143	-4.181156	-20.805621	-4.161124	2
5	0.008807	-4.254074	-4.028186	-4.163405	-4.175665	-4.182822	-20.804152	-4.160830	2
6	0.008489	-4.253571	-4.027934	-4.163531	-4.175771	-4.183131	-20.803939	-4.160788	2
7	0.008182	-4.253089	-4.027693	-4.163656	-4.175877	-4.183434	-20.803749	-4.160750	2
8	0.007886	-4.252628	-4.027461	-4.164057	-4.175983	-4.183728	-20.803858	-4.160772	2
9	0.007790	-4.252478	-4.027386	-4.164211	-4.176019	-4.183823	-20.803917	-4.160783	2
10	0.007601	-4.252186	-4.027239	-4.164515	-4.176089	-4.184014	-20.804043	-4.160809	2
11	0.007326	-4.251763	-4.027026	-4.164964	-4.176299	-4.184292	-20.804344	-4.160869	2
12	0.006093	-4.249896	-4.026083	-4.167059	-4.178038	-4.185575	-20.806651	-4.161330	2
13	0.005068	-4.248381	-4.025315	-4.168910	-4.179585	-4.186685	-20.808876	-4.161775	2
14	0.004216	-4.247147	-4.024688	-4.170529	-4.180942	-4.187637	-20.810943	-4.162189	2
15	0.002281	-4.244440	-4.023257	-4.174477	-4.184273	-4.189901	-20.816348	-4.163270	2
16	0.000000	-4.395458	-4.038115	-4.194629	-4.209178	-4.416023	-21.253404	-4.250681	0

```
[14]: > prec_mx_list[0]
```

```
Out[14]: array([[ 1.22596272, -0.          , -0.          , -0.          ],
                 [-0.          ,  2.74605582, -0.          , -0.          ],
                 [-0.          , -0.          ,  3.10477899, -0.          ],
                 [-0.          , -0.          , -0.          ,  1.44860969]])
```

```
[15]: > prec_mx_list[1]
```

```
Out[15]: array([[ 1.39660482, -0.03306993, -0.7758674 ,  0.          ],
                 [-0.03306993,  2.74694758, -0.          , -0.          ],
                 [-0.7758674 , -0.          ,  3.57628844, -0.24337161],
                 [ 0.          , -0.          , -0.24337161,  1.46744183]])
```

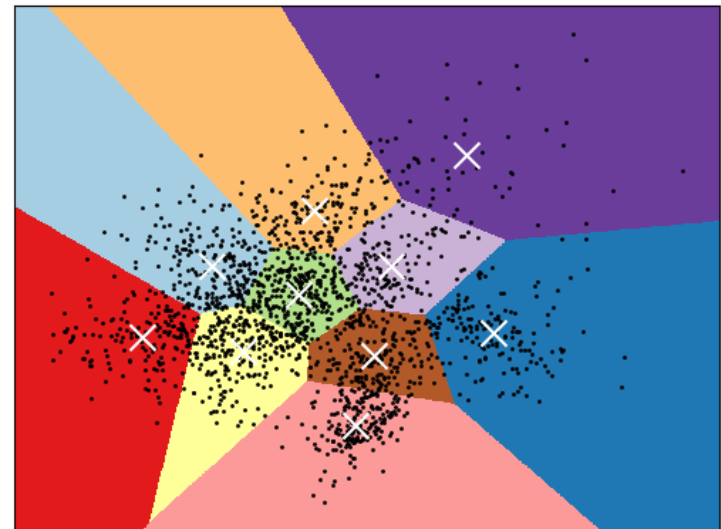
```
[16]: > prec_mx_list[ind_lambda]
```

```
Out[16]: array([[ 1.52069737, -0.17033761, -1.06283639,  0.11603294],
                 [-0.17033761,  2.78366128, -0.          , -0.14048774],
                 [-1.06283639, -0.          ,  3.98232618, -0.51782878],
                 [ 0.11603294, -0.14048774, -0.51782878,  1.52380577]])
```

## 2. Clustering: Overview

- **Clustering:** partition a set of objects (i.e. stock returns) into different clusters, such that objects in the same cluster behave more similarly
- There are various clustering methods:
  - K-Means Clustering
    - Separate samples into k clusters, in which each observation belongs to the cluster with the nearest mean (or cluster centroid), serving as a prototype of the cluster.
    - Requires the number of clusters to be specified
    - Scales well to large number of samples and has been used across a large range of application areas in many different fields.
  - Affinity Propagation
    - Used in our graphical analysis algorithm

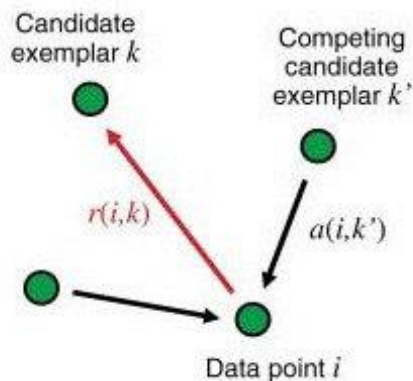
K-means clustering on the digits dataset (PCA-reduced data)  
Centroids are marked with white cross



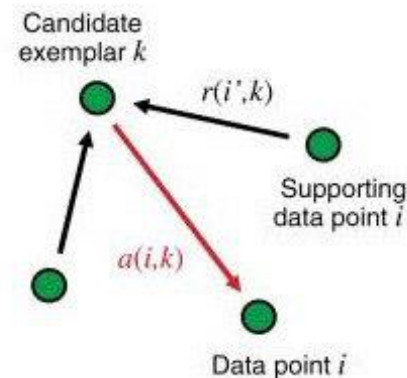
## 2. Affinity Propagation: Overview

- A clustering method; No need to know the number of clusters
- Finds exemplars which represent the clusters
- Introduces two intermediate variables: responsibility and availability
- Exemplars are chosen by samples if they are (1) similar enough to many samples and (2) chosen by many samples to be representative of themselves.
- Accepts dataset or matrix of similarity and returns cluster labels

Sending responsibilities



Sending availabilities



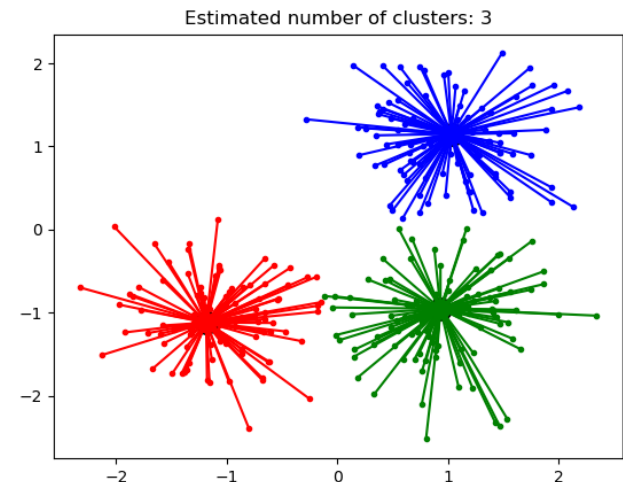
## 2. Affinity Propagation: Algorithm

- Initialize two zeros matrices: responsibility (r) and availability (a)
- Update two matrices using similarity matrix (s)

$$r(i, k) \leftarrow s(i, k) - \max[a(i, k') + s(i, k') \forall k' \neq k]$$

$$a(i, k) \leftarrow \min[0, r(k, k) + \sum_{i' \text{ s.t. } i' \notin \{i, k\}} r(i', k)]$$

- If similarity matrix is not given, apply Euclidean distance to calculate similarity matrix
- The sum of responsibility and availability matrix is criterion matrix which indicates the cluster label for each sample





## 2. Affinity Propagation: Damping

---

- The damping parameter ( $\lambda$ ) controls the rate of convergence
- Smaller damping parameter means that it converges faster

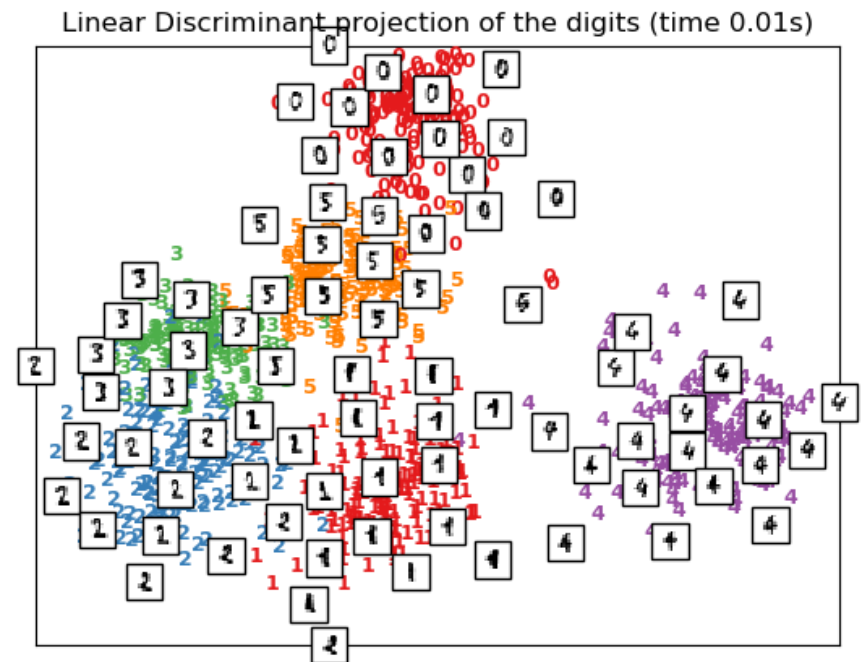
$$r_{t+1}(i, k) = \lambda \cdot r_t(i, k) + (1 - \lambda) \cdot r_{t+1}(i, k)$$

$$a_{t+1}(i, k) = \lambda \cdot a_t(i, k) + (1 - \lambda) \cdot a_{t+1}(i, k)$$

- The damping parameter prevents oscillation
- We will use [affinity propagation](#) to perform this affinity propagation clustering of data (with default setting: damping=0.5).

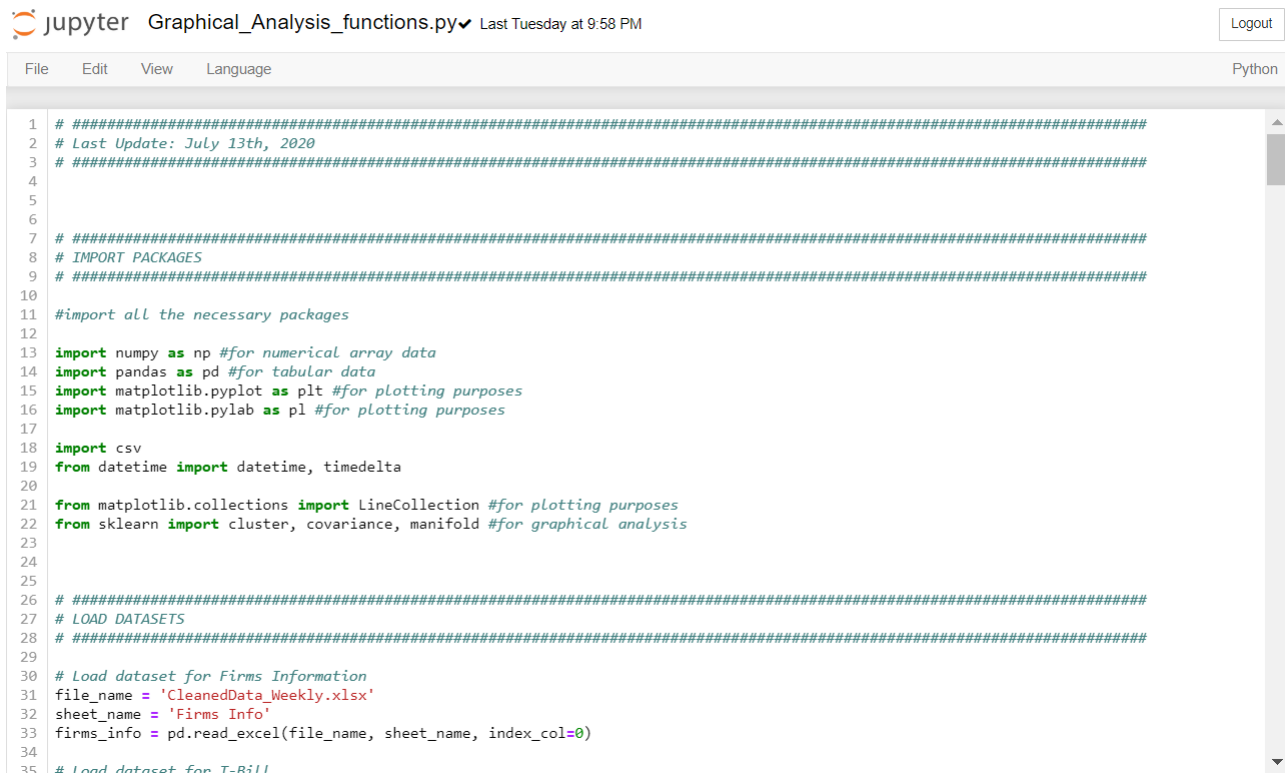
### 3. Multi-Dimensional Scaling: Overview

- Determines distances between samples and visualize them as nodes
- Performs dimensional reduction
- Preserves order of distances among the samples according to the given dataset
- We will use [MDS](#) to perform this multi-dimensional scaling technique for finding the 2D embedding.



# Graphical\_Analysis\_functions.py

- Contains the detailed codes for the self-defined functions which are used in the control file, including *getSumStat*, *graphicalAnalysis*, etc.
- Interested students can read through the code and see how the functions are defined.

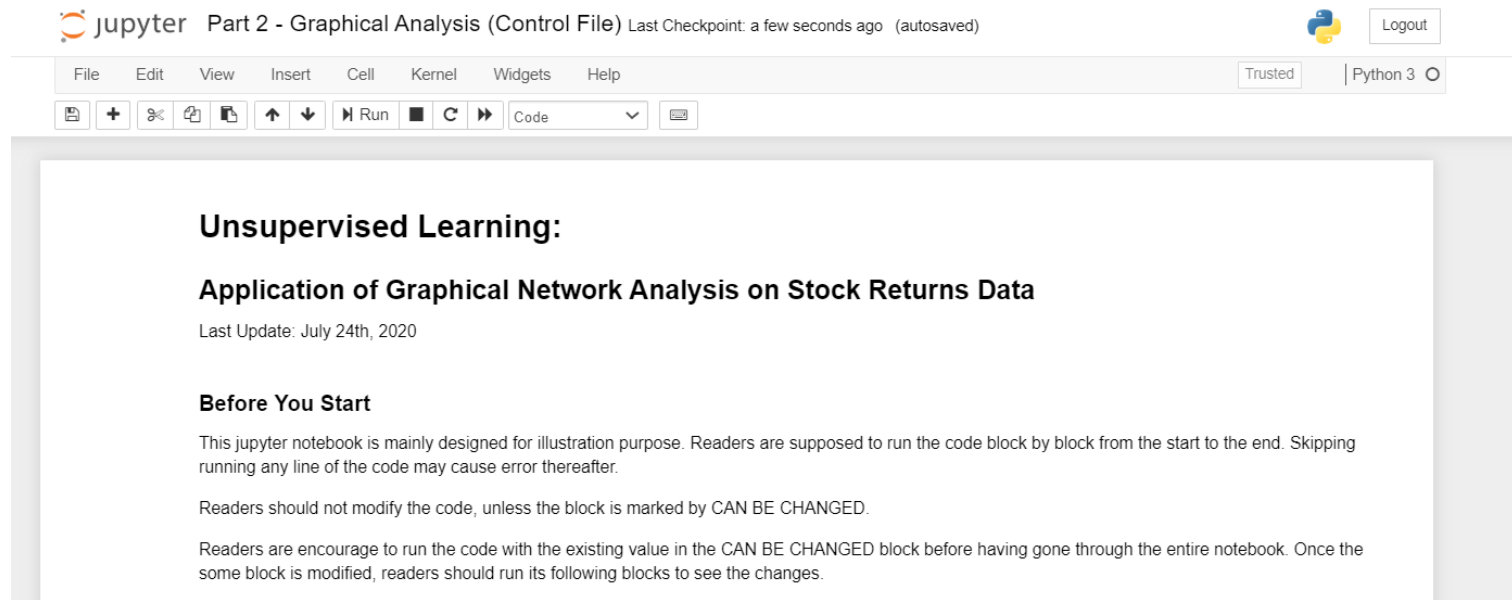


```
Jupyter Graphical_Analysis_functions.py Last Tuesday at 9:58 PM Logout
File Edit View Language Python
1 #####
2 # Last Update: July 13th, 2020
3 #####
4
5
6
7 #####
8 # IMPORT PACKAGES
9 #####
10
11 #import all the necessary packages
12
13 import numpy as np #for numerical array data
14 import pandas as pd #for tabular data
15 import matplotlib.pyplot as plt #for plotting purposes
16 import matplotlib.pylab as pl #for plotting purposes
17
18 import csv
19 from datetime import datetime, timedelta
20
21 from matplotlib.collections import LineCollection #for plotting purposes
22 from sklearn import cluster, covariance, manifold #for graphical analysis
23
24
25
26 #####
27 # LOAD DATASETS
28 #####
29
30 # Load dataset for Firms Information
31 file_name = 'CleanedData_Weekly.xlsx'
32 sheet_name = 'Firms Info'
33 firms_info = pd.read_excel(file_name, sheet_name, index_col=0)
34
35 # Load dataset for T-Bill
```



## Part 2 - Graphical Analysis (Control File).ipynb

- Our control file for doing graphical analysis on the stock returns data



The screenshot shows a Jupyter Notebook interface. At the top, the title bar reads "jupyter Part 2 - Graphical Analysis (Control File) Last Checkpoint: a few seconds ago (autosaved)". On the right, there is a "Logout" button and a "Trusted" status indicator. Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A toolbar below the menu bar contains icons for saving, adding, deleting, and running cells, as well as a "Code" dropdown menu. The main content area of the notebook displays the following text:

### Unsupervised Learning:

#### Application of Graphical Network Analysis on Stock Returns Data

Last Update: July 24th, 2020

#### Before You Start

This jupyter notebook is mainly designed for illustration purpose. Readers are supposed to run the code block by block from the start to the end. Skipping running any line of the code may cause error thereafter.

Readers should not modify the code, unless the block is marked by CAN BE CHANGED.

Readers are encourage to run the code with the existing value in the CAN BE CHANGED block before having gone through the entire notebook. Once the some block is modified, readers should run its following blocks to see the changes.

# Part 2 - Graphical Analysis (Control File).ipynb

## 1. Set Up Environment and Read Data

### 1.0 Model Setup

For the first step, let's import necessary packages and import our functions (for later use):

```
In [1]: # Import all the necessary packages

import numpy as np #for numerical array data
import pandas as pd #for tabular data
import matplotlib.pyplot as plt #for plotting purposes
import csv
from datetime import datetime, timedelta

from sklearn import cluster, covariance, manifold
from matplotlib.collections import LineCollection #for plotting purposes

import warnings
warnings.filterwarnings('ignore')

In [2]: # Import the self-defined function from the python file
from Graphical_Analysis_functions import *
```

### 1.1 Dataset Overview:

Below is a summary information of the stocks we picked for illustration:

```
In [3]: firms_info
```

Out[3]:

	Name	Sector
Ticker		

## Part 2 - Graphical Analysis (Control File).ipynb

### 1.1 Dataset Overview:

Below is a summary information of the stocks we picked for illustration:

```
In [3]: firms_info
```

```
Out[3]:
```

	Name	Sector
Ticker		
FRC	First Republic	Bank
PNC	PNC Financial	Bank
USB	US Bancorp	Bank
JPM	JPMorgan Chase	Bank
BAC	Bank of America	Bank
C	Citigroup	Bank
RY	Royal Bank of Canada	Bank
WFC	Wells Fargo	Bank
GS	Goldman Sachs	Bank
MS	Morgan Stanley	Bank
HSBC	HSBC Holdings plc	Bank
JNJ	Johnson & Johnson	Health
PFE	Pfizer	Health
MRK	Merck & Co.	Health
XOM	Exxon	Energy
RDS	Royal Dutch Shell	Energy
CVS	Chevron	Energy
BP	BP	Energy
AAPL	Apple	Tech
GOOG	Alphabet	Tech

```
In [4]: # Get and print the Sector Information
```

```
Sectors = firms_info.Sector.unique()  
print(Sectors)
```

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

## Part 2 - Graphical Analysis (Control File).ipynb

### 1.2 Load Data

Load the dataset from our Excel file. Plot and Examine the data.

```
In [5]: # 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()
```

```
In [6]: data
```

Out[6]:

	FRC	PNC	USB	JPM	BAC	C	RY	WFC	GS	MS	...	MRK	XOM	RDS	
date															
2000-01-09	NaN	-0.032304	-0.049869	-0.058139	-0.029888	-0.030302	-0.042493	-0.032457	-0.123426	-0.113398	...	0.115349	0.054307	NaN	0.0
2000-01-16	NaN	0.097264	0.019338	0.015465	0.034659	0.074074	-0.017752	0.065494	0.037093	0.088261	...	-0.010843	-0.013980	NaN	-0.0
2000-01-23	NaN	-0.061498	-0.094850	-0.012690	-0.093051	-0.026939	-0.035181	-0.121438	0.002128	-0.035047	...	-0.022765	0.014925	NaN	0.0
2000-01-30	NaN	0.045583	0.005989	0.051414	0.001367	-0.005537	0.068037	0.058020	0.015318	-0.044340	...	0.054358	-0.072060	NaN	-0.0
2000-02-06	NaN	-0.028609	0.014881	0.066015	-0.008197	-0.021727	-0.054815	0.000774	-0.026581	0.037512	...	-0.017185	0.025358	NaN	-0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-05-31	0.111717	0.090144	0.079866	0.087626	0.064430	0.086395	0.101580	0.095159	0.098984	0.101420	...	0.056960	0.019506	-0.008995	0.0
2020-06-07	0.110382	0.122675	0.191507	0.143049	0.173256	0.228554	0.109022	0.196449	0.109063	0.120588	...	0.019078	0.167364	0.182160	0.0
2020-06-14	-0.111648	-0.134890	-0.114703	-0.102130	-0.118463	-0.112299	-0.064237	-0.116829	-0.074064	-0.065617	...	-0.065038	-0.111341	-0.087901	-0.0
2020-06-21	0.038987	-0.022843	0.036257	-0.020626	0.018967	0.012823	0.010253	-0.013228	-0.000744	0.014045	...	0.020968	-0.025229	-0.014805	-0.0
2020-06-28	-0.063504	-0.067449	-0.072293	-0.053368	-0.083168	-0.063113	-0.019268	-0.081885	-0.061697	0.002558	...	-0.034790	-0.051327	-0.030347	-0.0

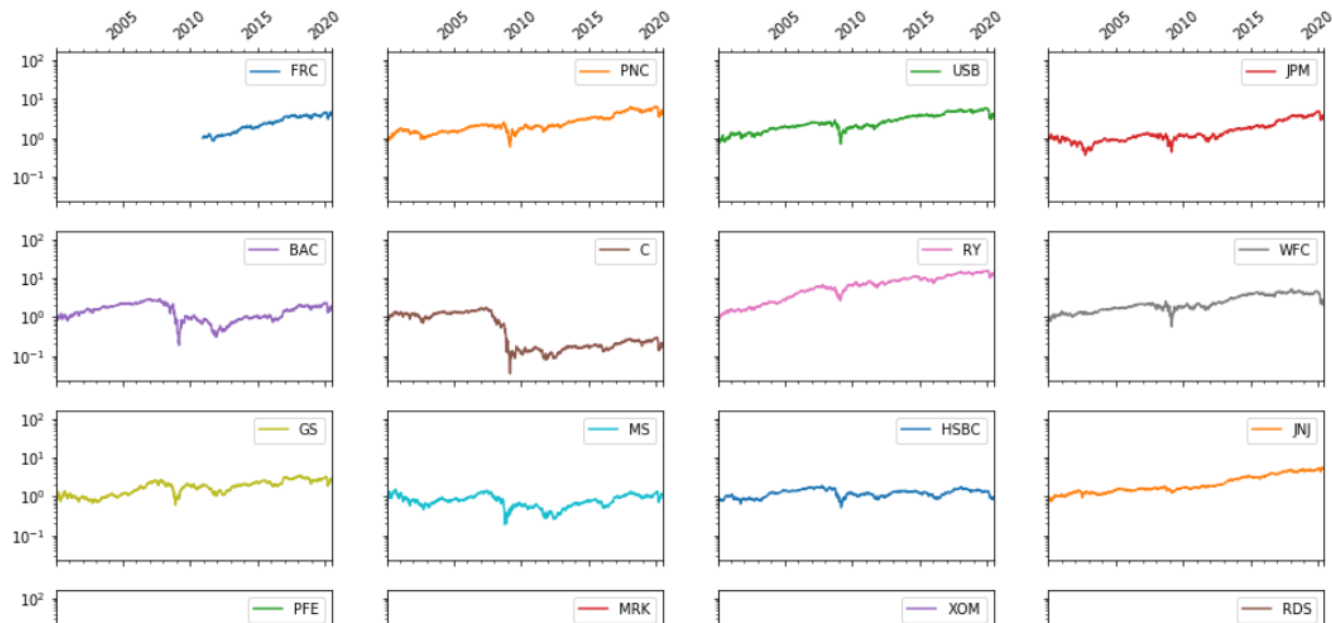
1069 rows × 23 columns

## Part 2 - Graphical Analysis (Control File).ipynb

```
In [8]: # Plot cumulative return for each firm
lo_col = 4
lo_row = int(np.ceil(len(df.columns)/4))
f_hei = lo_row * 2.5
f_wid = lo_col * 4
ax = (data+1).cumprod().plot(subplots=True, layout=(lo_row, lo_col), figsize=(f_wid, f_hei),
                             logy=True, sharex=True, sharey=True,
                             title='Cumulative Return for Individual Stock (Weekly, Log-scale)');

for i in range(lo_col):
    ax[0,i].xaxis.set_tick_params(which='both', top = True, labeltop=True, labelrotation=40)
plt.show()
```

Cumulative Return for Individual Stock (Weekly, Log-scale)



## Part 2 - Graphical Analysis (Control File).ipynb

### 1.3 Summary Statistics

For each firm, calculate and output the:

- Total Return
- Average return,
- Annualized Average return,
- Annualized standard deviation,
- Annualized Sharpe Ratio,
- Maximum drawdown.

Note: we used the 3-Month T-Bill Return as an indicator for risk-free rate when calculating for Sharpe Ratio. (Data Source: <https://fred.stlouisfed.org/series/WTB3MS>).

```
In [9]: # Output the summary statistics of the data for the whole time period
# Note: Firms with missing data will be dropped in this summary statistics calculation
getSumStat(data)
```

```
Summary Statistic Information from 01/03/2000 to 06/28/2020:
WARNING: Some firms have missing data during this time period!
Dropping firms:
FRC
RDS
GOOG
BABA
```

Out[9]:

	Sector	Total Return(%)	Ave Return(%)	Annu. Ave Return(%)	Annu. Std(%)	Annu. Sharpe Ratio	Max Drawdown(%)
PNC	Bank	311.54	0.25	14.05	35.74	0.34	-76.64
USB	Bank	260.58	0.23	12.75	34.18	0.32	-76.09
JPM	Bank	224.48	0.24	13.45	37.46	0.31	-71.51
BAC	Bank	56.75	0.24	13.07	46.04	0.25	-93.46
C	Bank	-81.53	0.10	5.61	53.84	0.07	-97.97
RY	Bank	1182.27	0.30	16.61	24.27	0.61	-60.28
WFC	Bank	125.06	0.21	11.28	37.52	0.25	-77.71
GS	Bank	153.40	0.22	11.88	37.01	0.27	-77.20
MS	Bank	12.70	0.24	13.19	49.77	0.23	-87.57



## Part 2 - Graphical Analysis (Control File).ipynb

### 2. Graphical Analysis

Below we will introduce a graphical analysis algorithm and apply it on the weekly stock returns data. The algorithm employs several unsupervised learning techniques and combine 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** 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  $l_1$  penalty to enforce sparsity on the precision matrix.
- **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.
- **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 Manifold 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

The source code is wrapped up in `Graphical_Analysis_functions.py`, and is beyond the scope of the course. Interested readers can find more information and read about the above three techniques.

References:

- [https://scikit-learn.org/stable/auto\\_examples/applications/plot\\_stock\\_market.html](https://scikit-learn.org/stable/auto_examples/applications/plot_stock_market.html)
- <https://scikit-learn.org/stable/modules/covariance.html#sparse-inverse-covariance>
- <https://scikit-learn.org/stable/modules/clustering.html#affinity-propagation>
- <https://scikit-learn.org/stable/modules/manifold.html#multi-dimensional-scaling-mds>

Recall that the sectors available in this dataset are:



## Part 2 - Graphical Analysis (Control File).ipynb

### First Data Analysis

Specifications:

- 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.
- By default (i.e. **display\_SumStat** = True, **display\_IndRet** = True), the summary statistics and the individual firm performance (individual firm's cumulative returns) will be displayed

```
In [15]: ##### CAN BE CHANGED #####

## Start and End date we are considering:
start_date = '2020-01-01'
end_date   = '2020-06-28'

## Sectors which we will be testing:
Sectors_chosen = ['Bank', 'Health', 'Energy', 'Tech']

## The list of firms we want to drop:
drop_firm = []

## Whether we want to display the summary statistics and/or individual firm performance:
display_SumStat = True
display_IndRet  = True

##### CAN BE CHANGED #####
```

## Part 2 - Graphical Analysis (Control File).ipynb

```
In [16]: print('Results over the time period ', start_date, ' to ', end_date, ':')
print()

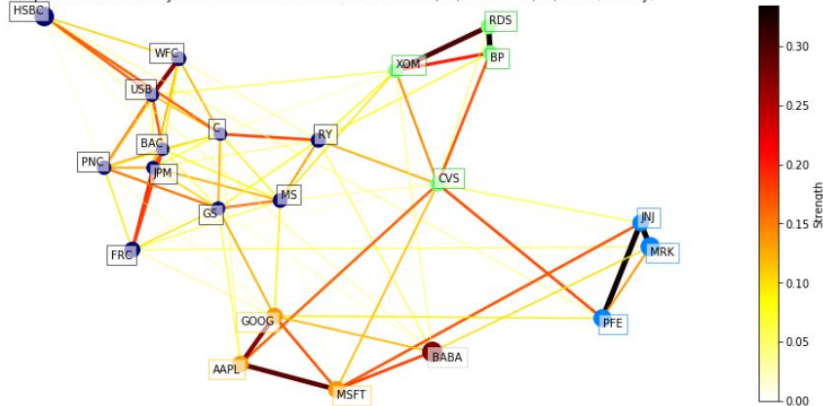
# Output the Clustering information, graphical network plot,
# as well as summary statistics (optional) and the individual firm performance (optional)
# Store the correlation matrix and precision matrix in "est" (stands for estimates)
# Store the plotting configuration information in "con_fig" which will be needed if we want to use the ZOOM functions
## Note: you can view the correlation matrix by looking at est[0] and view the precision matrix by looking at est[1]
est, con_fig = graphicalAnalysis(data, start_date, end_date,
                                Sectors_chosen, drop_firm,
                                display_SumStat, display_IndRet)
```

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

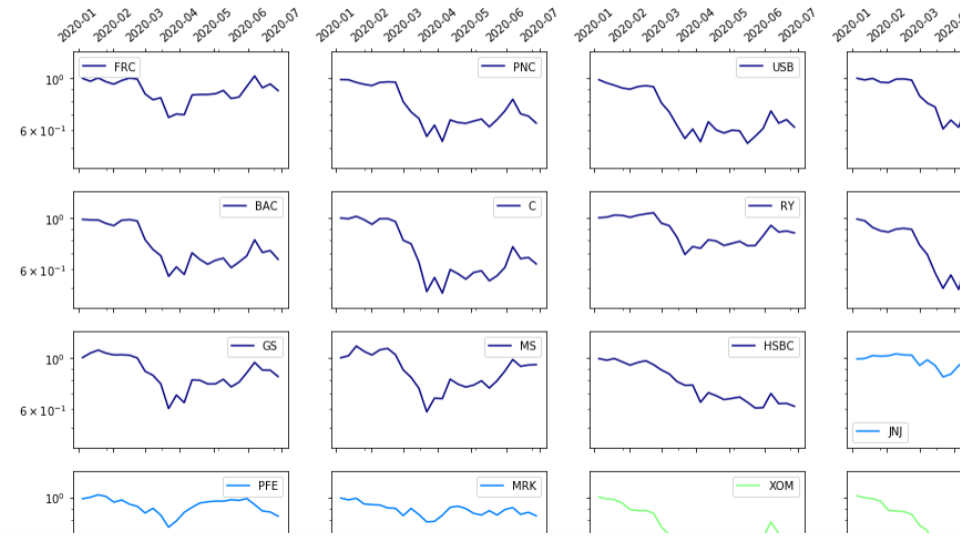
Sectors chosen in the Graphical Analysis are:  
['Bank', 'Health', 'Energy', 'Tech']

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

Graphical Network Analysis of Selected Firms over the Period 12/30/2019 to 06/28/2020 (Weekly)



Individual Stock Performance over the Period 12/30/2019 to 06/28/2020 (Weekly):



Summary Statistic Information from 12/30/2019 to 06/28/2020:

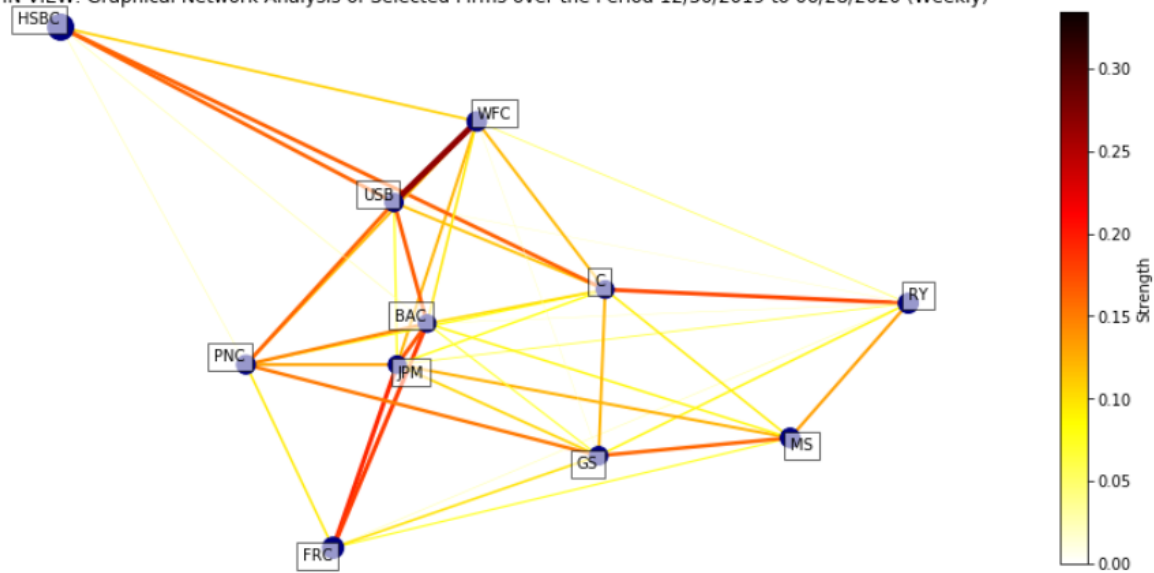
	Sector	Total Return(%)	Ave Return(%)	Annu. Ave Return(%)	Annu. Std(%)	Annu. Sharpe Ratio	Max Drawdown(%)
1	FRC Bank	-11.45	-0.16	-8.11	57.39	-0.15	-32.56
2	PNC Bank	-35.98	-1.27	-48.68	67.95	-0.74	-45.96
3	USB Bank	-38.49	-1.44	-52.93	67.65	-0.80	-46.95
4	JPM Bank	-32.30	-1.13	-44.75	62.17	-0.74	-39.64
5	BAC Bank	-33.63	-1.14	-44.79	68.50	-0.67	-43.26
6	C Bank	-36.64	-1.09	-43.39	83.90	-0.53	-53.47
7	RY Bank	-13.72	-0.38	-17.95	44.40	-0.42	-34.02
8	WFC Bank	-51.54	-2.22	-68.89	76.84	-0.92	-54.91

## Part 2 - Graphical Analysis (Control File).ipynb

We can use the following code to ZOOM IN and focus on a particular sector (or sectors) from the given network graph (above).

```
In [17]: ▶ ##### CAN BE CHANGED #####  
  
## The list of Sector(s) we want to look at more closely  
Sectors_list = ['Bank']  
  
##### CAN BE CHANGED #####  
  
In [18]: ▶ graphicalAnalysis_plot_ZOOM_bySector(Sectors_list, con_fig)
```

ZOOM IN VIEW: Graphical Network Analysis of Selected Firms over the Period 12/30/2019 to 06/28/2020 (Weekly)



We can also use the following code to ZOOM IN and focus on a particular set of firms from the given network graph.

## Part 2 - Graphical Analysis (Control File).ipynb

We can also use the following code to ZOOM IN and focus on a particular set of firms from the given network graph.

```
In [19]: ##### CAN BE CHANGED #####

## The List of firms we want to look at more closely
# Note: These firms don't have to be from the same sector
# (but they need to be in the "Sectors_chosen" list if "Sectors_chosen" is different from default setting)

firms_list = ['FRC', 'JPM', 'BAC', 'GS', 'MS', 'PNC', 'C']

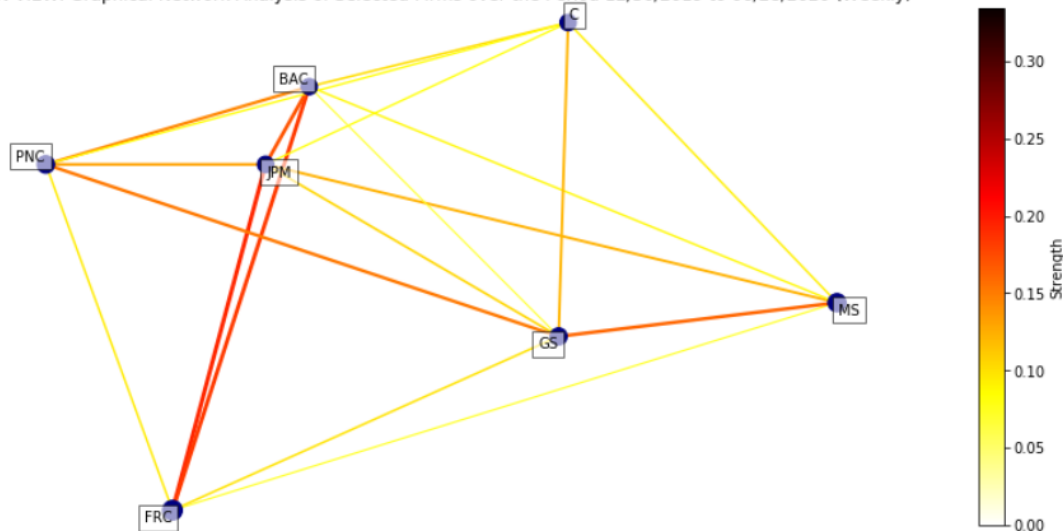
##### CAN BE CHANGED #####
```

```
In [20]: print('ZOOM IN View of the Selected Firms:')
print(firms_list)
graphicalAnalysis_plot_ZOOM_byFirm(firms_list, con_fig)
```

ZOOM IN View of the Selected Firms:

['FRC', 'JPM', 'BAC', 'GS', 'MS', 'PNC', 'C']

ZOOM IN VIEW: Graphical Network Analysis of Selected Firms over the Period 12/30/2019 to 06/28/2020 (Weekly)





# Conclusion

---

- Graphical Network Analysis
  - 1. Sparse Covariance Estimation via Graphical Lasso
    - find an estimate of precision matrix
  - 2. Clustering via Affinity Propagation
    - identify clusters
  - 3. Manifold Learning via Multidimensional Scaling (MDS)
    - Visualization
- We can apply this algorithm to extract the conditional dependency structure of the data and draw inferences based on the given network graph.

**Thank you!**



## References

---

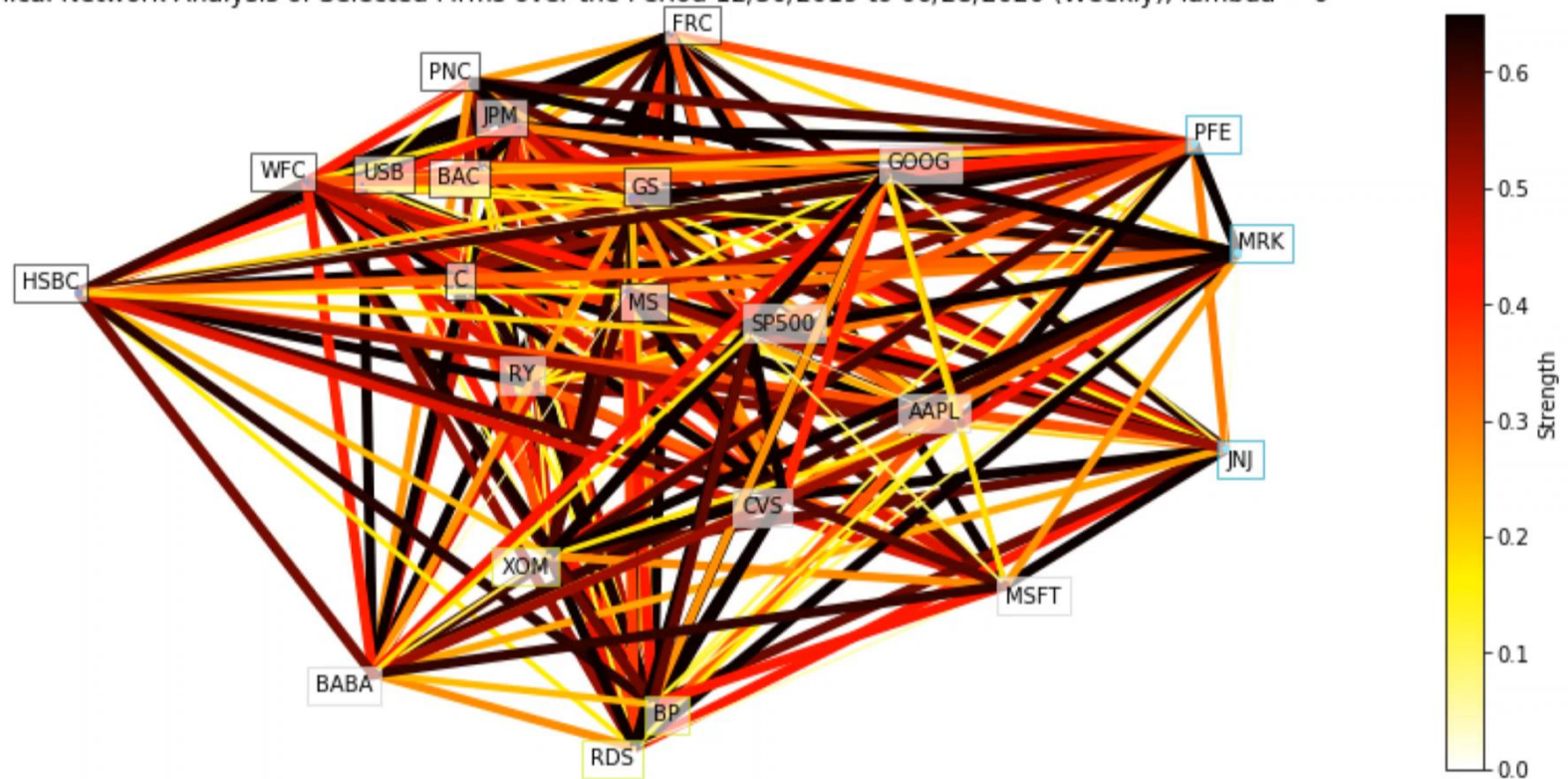
- Sklearn example: Visualizing the Stock Market Structure - [https://scikit-learn.org/stable/auto\\_examples/applications/plot\\_stock\\_market.html](https://scikit-learn.org/stable/auto_examples/applications/plot_stock_market.html)
- Sparse Inverse Covariance - <https://scikit-learn.org/stable/modules/covariance.html#sparse-inverse-covariance>
- Cross-Validation - [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)
- Affinity Propagation - <https://scikit-learn.org/stable/modules/clustering.html#affinity-propagation>
- Multidimensional Scaling (MDS) - <https://scikit-learn.org/stable/modules/manifold.html#multi-dimensional-scaling-mds>
- Graphical Lasso:
  - **J. Friedman, T. Hastie, and R. Tibshirani.** "Sparse inverse covariance estimation with the graphical lasso", *Biostatistics*, 2008.
  - [http://www.princeton.edu/~yc5/ele538b\\_sparsity/lectures/graphical\\_lasso.pdf](http://www.princeton.edu/~yc5/ele538b_sparsity/lectures/graphical_lasso.pdf)

# **Appendix**



# Varying Shrinkage Parameter in Graphical Lasso

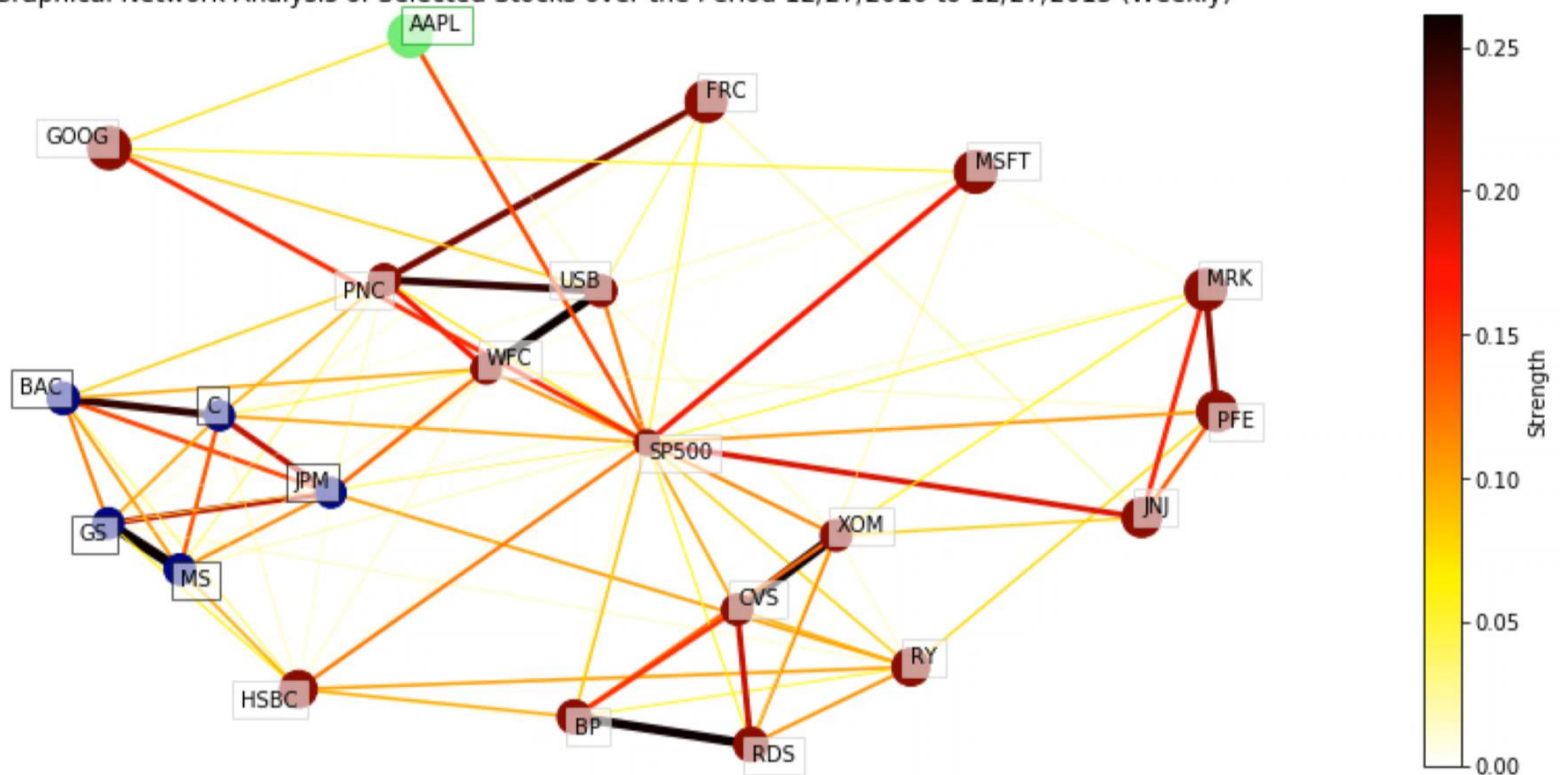
Graphical Network Analysis of Selected Firms over the Period 12/30/2019 to 06/28/2020 (Weekly),  $\lambda = 0$



# Time Lapse Video



Graphical Network Analysis of Selected Stocks over the Period 12/27/2010 to 12/27/2015 (Weekly)





# Semidefinite Program (SDP)

---

- A semidefinite program (SDP) is a convex optimization problem, and it is a natural generalization of linear programming (LP) but much richer expressive power.
- It minimizes a linear function subject to the constraint that is an affine combination of symmetric positive semidefinite matrix.

**Definition.** *A semidefinite program is an optimization problem of the form*

$$\begin{aligned} \min_{X \in S^{n \times n}} \quad & \text{Tr}(CX) \\ \text{s.t.} \quad & \text{Tr}(A_i X) = b_i, i = 1, \dots, m, \\ & X \succeq 0, \end{aligned}$$

*where the input data is  $C \in S^{n \times n}$ ,  $A_i \in S^{n \times n}$ ,  $i = 1, \dots, m$ ,  $b_i \in \mathbb{R}$ ,  $i = 1, \dots, m$ .*

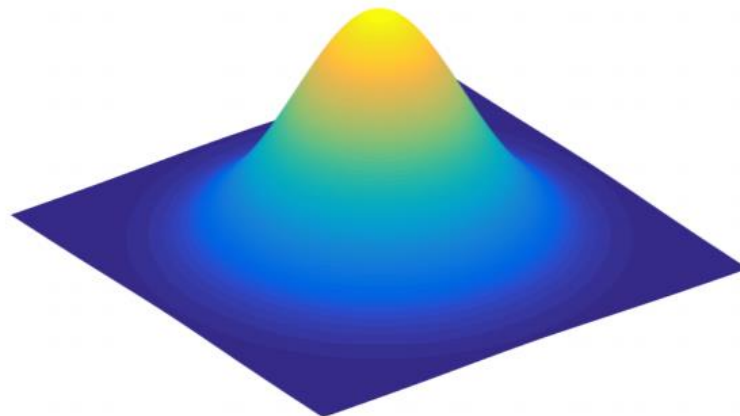
- $S^{n \times n}$  denotes the set of  $n \times n$  real symmetric matrices
- $\text{Tr}$  denotes the trace of a matrix; i.e. the sum of its diagonal elements (which also equals to the sum of its eigenvalues)

# Graphical Lasso: Background – Multivariate Gaussians

- We say that a random vector  $x$  has a multivariate gaussian distribution with zero mean and covariance  $\Sigma$ , i.e.  $x \sim N(0, \Sigma)$ , then its pdf takes the form:

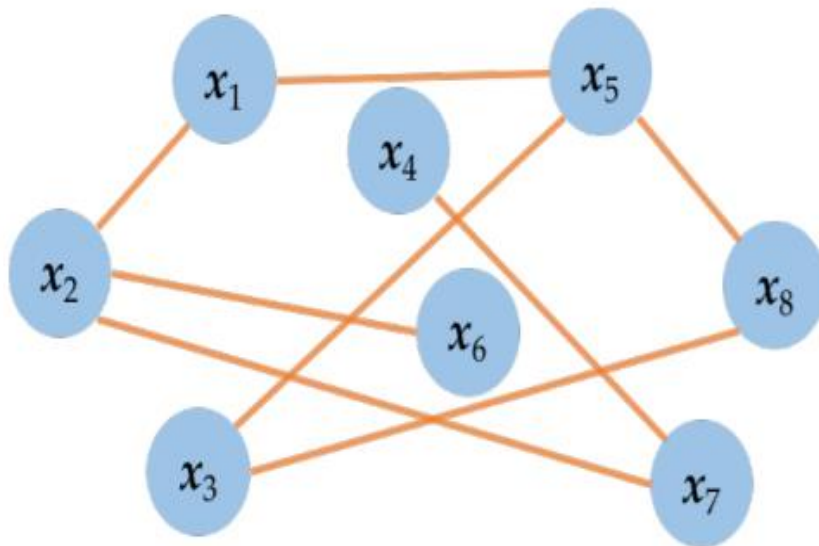
$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{(2\pi)^{p/2} \det(\Sigma)^{1/2}} \exp \left\{ -\frac{1}{2} \mathbf{x}^\top \Sigma^{-1} \mathbf{x} \right\} \\ &\propto \det(\Theta)^{1/2} \exp \left\{ -\frac{1}{2} \mathbf{x}^\top \Theta \mathbf{x} \right\} \end{aligned}$$

- $\Sigma = \mathbb{E}[xx^\top]$  denotes the covariance matrix, and it is positive definite
- $\Theta = \Sigma^{-1}$  is the inverse covariance matrix, or the precision matrix



# Graphical Lasso: Background – Gaussian Graphical Model

- Recall: if two features are independent conditionally on the others, the corresponding coefficient in the precision matrix will be zero
  - Conditional independence  $\Leftrightarrow$  sparsity
- Let  $x_1, \dots, x_n$  be i.i.d samples from the zero-mean Multivariate Gaussian distribution. The network graph (left) will have precision matrix (right).



$$\underbrace{\begin{bmatrix} * & * & 0 & 0 & * & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 & * & * & 0 \\ 0 & 0 & * & 0 & * & 0 & 0 & * \\ 0 & 0 & 0 & * & 0 & 0 & * & 0 \\ * & 0 & * & 0 & * & 0 & 0 & * \\ 0 & * & 0 & 0 & 0 & * & 0 & 0 \\ 0 & * & 0 & * & 0 & 0 & * & 0 \\ 0 & 0 & * & 0 & * & 0 & 0 & * \end{bmatrix}}_{\Theta}$$



## Graphical Lasso: Likelihoods of Gaussian Graphical Model

- Let  $x_1, \dots, x_n$  be i.i.d samples from the zero-mean Multivariate Gaussian distribution of dimension  $p$ , i.e.  $x_i \sim N(0, \Sigma)$  and  $x_i \in \mathbb{R}^p$ .
- Then, its log-likelihood is:

$$\begin{aligned}\ell(\Theta) &= \frac{1}{n} \sum_{i=1}^n \log f(x_i) = \frac{1}{2} \log \det(\Theta) - \frac{1}{2n} \sum_{i=1}^n x_i^T \Theta x_i \\ &= \frac{1}{2} \log \det(\Theta) - \frac{1}{2} \text{Tr}(S\Theta)\end{aligned}$$

- Where  $S$  is the sample covariance
- Hence, the maximum likelihood estimator for the precision matrix  $\Theta$  is

$$\max_{\Theta \succeq 0} \log \det(\Theta) - \text{Tr}(S\Theta)$$

Equivalently,

$$\min_{\Theta \succeq 0} \text{tr}(S\Theta) - \log \det(\Theta)$$





## Graphical Lasso (Friedman, Hastie, and Tibshirani, 2008)

- Knowing that many features are conditionally independent
  - $\Rightarrow$  many entries in the precision matrix are zeros
  - i.e. many missing link in the network graph
- Key idea: adding L1 (lasso) regularization to the objective
- Hence, the graphical lasso estimator is the solution to the following problem:

$$\min_{\Theta \succeq 0} \text{tr}(S\Theta) - \log \det(\Theta) + \underbrace{\lambda \|\Theta\|_1}_{\text{lasso penalty}}$$

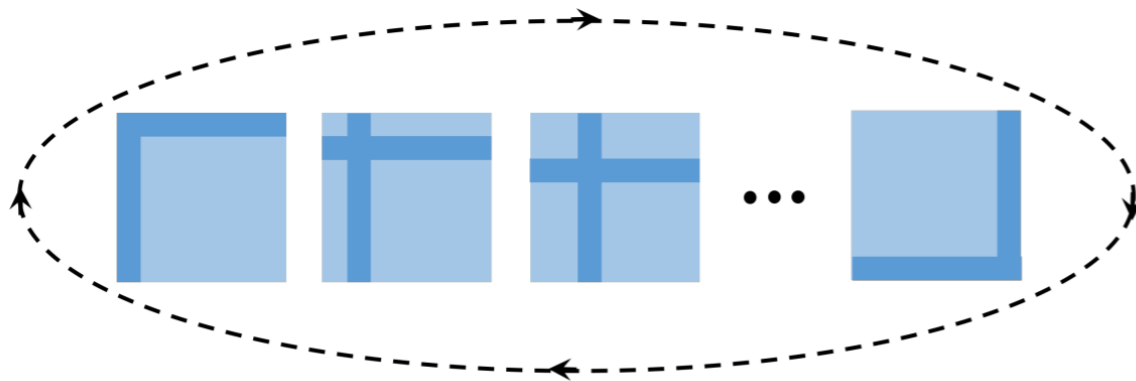
- First-order optimality condition gives:

$$\Theta^{-1} - S - \lambda \underbrace{\partial \|\Theta\|_1}_{\text{subgradient}} = 0$$

$$\Rightarrow (\Theta^{-1})_{i,i} = S_{i,i} + \lambda, \quad 1 \leq i \leq p$$

# Graphical Lasso: Blockwise Coordinate Descent

- The blockwise coordinate descent approach an efficient approach used to solve for an estimator of  $\Sigma = \Theta^{-1}$ ; one can then get the estimate for  $\Theta$  by taking its inverse.
- Idea: repeatedly cycle through all columns/rows, and optimize only one single column/row in each step



- Notation: Let  $W$  be the working version of  $\Theta^{-1}$ . Partition all matrices  $(\Theta, S, W)$  into 1 column and 1 row versus the rest, that is:

$$\Theta = \begin{bmatrix} \Theta_{11} & \theta_{12} \\ \theta_{12}^\top & \theta_{22} \end{bmatrix} \quad S = \begin{bmatrix} S_{11} & s_{12} \\ s_{12}^\top & s_{22} \end{bmatrix} \quad W = \begin{bmatrix} W_{11} & w_{12} \\ w_{12}^\top & w_{22} \end{bmatrix}$$





# Graphical Lasso: Blockwise Coordinate Descent

## ■ Blockwise step:

- Suppose we fix all but the last column/row.
- Using the **matrix inverse formula**, we have:

$$W = \Theta^{-1} = \begin{bmatrix} \Theta_{11} & \theta_{12} \\ \theta_{12}^T & \theta_{22} \end{bmatrix}^{-1} = \begin{bmatrix} * & -\frac{1}{\tilde{\theta}_{22}} \Theta_{11}^{-1} \theta_{12} \\ * & * \end{bmatrix} = \begin{bmatrix} * & -\frac{1}{\tilde{\theta}_{22}} W_{11} \theta_{12} \\ * & * \end{bmatrix} = \begin{bmatrix} * & W_{11} \beta \\ * & * \end{bmatrix}$$

$$\text{where } \tilde{\theta}_{22} = \theta_{22} - \theta_{12}^T \Theta_{11}^{-1} \theta_{12} > 0 \text{ and } \beta = -\frac{\theta_{12}}{\tilde{\theta}_{22}}$$

- Recall that the optimality condition is:

$$\Theta^{-1} - S - \lambda \partial \|\Theta\|_1 = 0$$

- Thus, focusing on the upper right block of the gradient equation, we get the following:

$$0 \in W_{11} \beta - s_{12} - \lambda \partial \|\theta_{12}\|_1 = W_{11} \beta - s_{12} + \lambda \partial \|\beta\|_1$$

- Note that the condition right above coincides with the optimality condition for the following optimization problem:

$$\text{minimize}_{\beta} \quad \frac{1}{2} \|W_{11}^{1/2} \beta - W_{11}^{-1/2} s_{12}\|_2^2 + \lambda \|\beta\|_1$$



## Graphical Lasso: Blockwise Coordinate Descent - Algorithm

---

**Algorithm.** Block coordinate descent for graphical lasso

---

**Initialize**  $\mathbf{W} = \mathbf{S} + \lambda \mathbf{I}$  and fix its diagonals  $\{w_{i,i}\}$ .

**Repeat until convergence:**

**for**  $t = 1, \dots, p$ :

    (i) Partition  $\mathbf{W}$  (resp.  $\mathbf{S}$ ) into 4 parts, where the upper-left part consists of all but the  $j$ th row / column

    (ii) Solve

$$\text{minimize}_{\boldsymbol{\beta}} \quad \frac{1}{2} \|\mathbf{W}_{11}^{1/2} \boldsymbol{\beta} - \mathbf{W}_{11}^{-1/2} \mathbf{s}_{12}\|^2 + \lambda \|\boldsymbol{\beta}\|_1$$

    (iii) Update  $\mathbf{w}_{12} = \mathbf{W}_{11} \boldsymbol{\beta}$

**Set**  $\hat{\boldsymbol{\theta}}_{12} = -\hat{\boldsymbol{\theta}}_{22} \boldsymbol{\beta}$  with  $\hat{\boldsymbol{\theta}}_{22} = 1 / (w_{22} - \mathbf{w}_{12}^\top \boldsymbol{\beta})$

---



## Graphical Lasso: Blockwise Coordinate Descent

- With the Blockwise Coordinate Descent approach, one can solve for an estimator of  $\Theta^{-1} = \Sigma$ .
- Validity check:  $W \succcurlyeq 0$ , i.e. is the estimator  $W$  positive semidefinite?
  - Yes, this is due to the following lemma:

### **Lemma. (Mazumder & Hastie, '12)**

*If we start with  $W \succ 0$  satisfying  $\|W - S\|_{\infty} \leq \lambda$ , then every row/column update maintains positive definiteness of  $W$ .*

- Since we initialized  $W^{(0)} = S + \lambda I$ , then our update in each step  $W^{(t)}$  will always be positive definite, so is the final estimate for  $\Theta^{-1} = \Sigma$ .