```
import sys
from datetime import datetime

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
from six.moves.urllib.request import urlopen
from six.moves.urllib.parse import urlencode
from sklearn import cluster, covariance, manifold
import quandl
import pandas as pd
import datetime
from pandas_datareader import data as pdr
import fix_yahoo_finance as yf
from itertools import compress
from helper import read_txt

print(__doc__)

def retry(f, n_attempts=3):
    "Wrapper function to retry function calls in case of exceptions"
    def wrapper(*args, **kwargs):
        for i in range(n_attempts):
            try:
                return f(*args, **kwargs)
            except Exception:
                if i == n_attempts - 1:
                    raise
    return wrapper

def quotes_historical_google(symbol, start_date, end_date):

    format = '%Y-%m-%d'  # Formatting directives
    start = start_date.strftime(format)
    end = end_date.strftime(format)

    yf.pdr_override()  # <== that's all it takes :-)
    data = pdr.get_data_yahoo(symbol, start=start, end=end)

    return data



symbol_dict = {
    'NYSE:TOT': 'Total',
    'NYSE:XOM': 'Exxon',
    'NYSE:CVX': 'Chevron',
    'NYSE:COP': 'ConocoPhillips',
    'NYSE:VLO': 'Valero Energy',
```

```
    'NASDAQ:MSFT': 'Microsoft',
    'NYSE:IBM': 'IBM',
    'NYSE:TWX': 'Time Warner',
    'NASDAQ:CMCSA': 'Comcast',
    'NYSE:CVC': 'Cablevision',
    'NASDAQ:YHOO': 'Yahoo',
    'NASDAQ:DELL': 'Dell',
    'NYSE:HPQ': 'HP',
    'NASDAQ:AMZN': 'Amazon',
    'NYSE:TM': 'Toyota',
    'NYSE:CAJ': 'Canon',
    'NYSE:SNE': 'Sony',
    'NYSE:F': 'Ford',
    'NYSE:HMC': 'Honda',
    'NYSE:NAV': 'Navistar',
    'NYSE:NOC': 'Northrop Grumman',
    'NYSE:BA': 'Boeing',
    'NYSE:KO': 'Coca Cola',
    'NYSE:MMM': '3M',
    'NYSE:MCD': 'McDonald\'s',
    'NYSE:PEP': 'Pepsi',
    'NYSE:K': 'Kellogg',
    'NYSE:UN': 'Unilever',
    'NASDAQ:MAR': 'Marriott',
    'NYSE:PG': 'Procter Gamble',
    'NYSE:CL': 'Colgate-Palmolive',
    'NYSE:GE': 'General Electrics',
    'NYSE:WFC': 'Wells Fargo',
    'NYSE:JPM': 'JPMorgan Chase',
    'NYSE:AIG': 'AIG',
    'NYSE:AXP': 'American express',
    'NYSE:BAC': 'Bank of America',
    'NYSE:GS': 'Goldman Sachs',
    'NASDAQ:AAPL': 'Apple',
    'NYSE:SAP': 'SAP',
    'NASDAQ:CSCO': 'Cisco',
    'NASDAQ:TXN': 'Texas Instruments',
    'NYSE:XRX': 'Xerox',
    'NYSE:WMT': 'Wal-Mart',
    'NYSE:HD': 'Home Depot',
    'NYSE:GSK': 'GlaxoSmithKline',
    'NYSE:PFE': 'Pfizer',
    'NYSE:SNY': 'Sanofi-Aventis',
    'NYSE:NVS': 'Novartis',
    'NYSE:KMB': 'Kimberly-Clark',
    'NYSE:R': 'Ryder',
    'NYSE:GD': 'General Dynamics',
    'NYSE:RTN': 'Raytheon',
    'NYSE:CVS': 'CVS',
    'NYSE:CAT': 'Caterpillar',
```

```
100     'NYSE:DD': 'DuPont de Nemours'
101     }
102
103 # ########################################################
    ######################
104 symbols, names = np.array(sorted(symbol_dict.items())).T
105
106 symbols_new = [] #symbols without "NYSE"
107 for n in range(len(symbols)):
108     symbols_new.append(symbols[n].split(":")[1])
109 print symbols_new
110
111
112 start = datetime.datetime(2017, 1, 1)
113 end = datetime.datetime(2018, 4, 1)
114
115 quotes = []
116
117 for symbol in symbols_new:
118     print('Fetching quote history for %r' % symbol)
119     quotes.append(quotes_historical_google(
120         symbol, start, end))
121 print type(quotes[0])
122
123 empty_dataframe = []
124 for q in quotes:
125     empty_dataframe.append(q.empty)
126
127 symbols_valid = []
128 name_valid = []
129
130 for i,j in enumerate(empty_dataframe):
131     if j == False:
132         symbols_valid.append(symbols_new[i])
133         name_valid.append(names[i])
134
135 print "drop the empty dataframe and then download the
    stock price data...."
136
137 # ########################################################
    ######################
138 quotes = []
139
140 for symbol in symbols_valid:
141     print('Fetching quote history for %r' % symbol)
142     quotes.append(quotes_historical_google(
143         symbol, start, end))
144
145 print "complete fetching the stock price data ...."
146
```

```
147
148 close_prices = np.vstack([q.iloc[:,4] for q in quotes]) #4
     stands for Adj Close
149 open_prices = np.vstack([q.iloc[:,0] for q in quotes]) #0
    stands for Open
150
151 # The daily variations of the quotes are what carry most
    information
152 variation = close_prices - open_prices
153
154
155 # ##############################################################
    ######################
156
157 # Learn a graphical structure from the correlations
158 edge_model = covariance.GraphLassoCV()
159
160 # standardize the time series: using correlations rather
    than covariance
161 # is more efficient for structure recovery
162 X = variation.copy().T #copy and then transpose
163 X /= X.std(axis=0) #the std of each column(stock), thus if
     there two stocks, then the X.std(axis=0) only have two
    values.
164 edge_model.fit(X)
165
166 # ##############################################################
    ######################
167 # Cluster using affinity propagation
168 #
169 _, labels = cluster.affinity_propagation(edge_model.
    covariance_)
170 n_labels = labels.max()
171 #
172
173 for i in range(n_labels + 1):
174     print 'Cluster %i: %s' % ((i + 1), ', '.join(list(
    compress(name_valid,labels == i))))
175
176
177
178 # ##############################################################
    ######################
179 # Find a low-dimension embedding for visualization: find
    the best position of
180 # the nodes (the stocks) on a 2D plane
181
182 # We use a dense eigen_solver to achieve reproducibility (
    arpack is
183 # initiated with random vectors that we don't control). In
```

```
 addition, we
# use a large number of neighbors to capture the large-scale structure.
node_position_model = manifold.LocallyLinearEmbedding(
    n_components=3, eigen_solver='dense', n_neighbors=6)

embedding = node_position_model.fit_transform(X.T).T

# ######################################################################
# Visualization
plt.figure(1, facecolor='w', figsize=(10, 8))
plt.clf()
ax = plt.axes([0., 0., 1., 1.])
plt.axis('off')

# Display a graph of the partial correlations
partial_correlations = edge_model.precision_.copy()
d = 1 / np.sqrt(np.diag(partial_correlations))
partial_correlations *= d
partial_correlations *= d[:, np.newaxis]
non_zero = (np.abs(np.triu(partial_correlations, k=1)) > 0.02)

# Plot the nodes using the coordinates of our embedding
plt.scatter(embedding[0], embedding[1], s=100 * d ** 2, c=labels,
            cmap=plt.cm.spectral)

# Plot the edges
start_idx, end_idx = np.where(non_zero)
# a sequence of (*line0*, *line1*, *line2*), where::
#            linen = (x0, y0), (x1, y1), ... (xm, ym)
segments = [[embedding[:, start], embedding[:, stop]]
            for start, stop in zip(start_idx, end_idx)]
values = np.abs(partial_correlations[non_zero])
lc = LineCollection(segments,
                    zorder=0, cmap=plt.cm.hot_r,
                    norm=plt.Normalize(0, .7 * values.max()))
lc.set_array(values)
lc.set_linewidths(15 * values)
ax.add_collection(lc)

# Add a label to each node. The challenge here is that we want to
# position the labels to avoid overlap with other labels
for index, (name, label, (x, y)) in enumerate(
        zip(names, labels, embedding.T)):

```

```
227     dx = x - embedding[0]
228     dx[index] = 1
229     dy = y - embedding[1]
230     dy[index] = 1
231     this_dx = dx[np.argmin(np.abs(dy))]
232     this_dy = dy[np.argmin(np.abs(dx))]
233     if this_dx > 0:
234         horizontalalignment = 'left'
235         x = x + .002
236     else:
237         horizontalalignment = 'right'
238         x = x - .002
239     if this_dy > 0:
240         verticalalignment = 'bottom'
241         y = y + .002
242     else:
243         verticalalignment = 'top'
244         y = y - .002
245     plt.text(x, y, name, size=10,
246              horizontalalignment=horizontalalignment,
247              verticalalignment=verticalalignment,
248              bbox=dict(facecolor='w',
249                        edgecolor=plt.cm.spectral(label /
    float(n_labels)),
250                        alpha=.6))
251
252 plt.xlim(embedding[0].min() - .15 * embedding[0].ptp(),
253          embedding[0].max() + .10 * embedding[0].ptp(),)
254 plt.ylim(embedding[1].min() - .03 * embedding[1].ptp(),
255          embedding[1].max() + .03 * embedding[1].ptp())
256
257 plt.show()
258
```