In [3]:
```
import numpy as np
import pandas as pd
import random
from datetime import datetime, timedelta
from bs4 import BeautifulSoup
from selenium import webdriver
import requests
from selenium.webdriver.common.by import By
from textblob import TextBlob
from selenium.common.exceptions import StaleElementReferenceException, Timeou
import quandl
import requests
import time
```

In [4]:
```
API_KEY = "G4xyNzxhrmq82gLxfES5"
```

In [5]:
```
snp = pd.read_csv("^GSPC.csv")
nasdaq = pd.read_csv("^IXIC.csv")
nyse = pd.read_csv("^NYA.csv")
```

In [6]:
```
def perc_change(row):
    return (row["Close"] - row["Open"])/row["Open"]

nyse["Change"] = nyse.apply(perc_change, axis=1)
nasdaq["Change"] = nasdaq.apply(perc_change, axis=1)
snp["Change"] = snp.apply(perc_change, axis=1)
```

In [7]:
```
# snp[snp["Date"] == "2014-10-20"]
nyse
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volum
0	2013-07-10	9340.730469	9372.639648	9314.410156	9340.679688	9340.679688	301101000
1	2013-07-11	9446.940430	9500.259766	9443.049805	9493.259766	9493.259766	344634000
2	2013-07-12	9488.799805	9498.500000	9465.580078	9498.500000	9498.500000	303907000
3	2013-07-15	9506.459961	9531.660156	9496.919922	9520.950195	9520.950195	262320000
4	2013-07-16	9521.200195	9525.040039	9462.559570	9489.000000	9489.000000	308171000
5	2013-07-17	9516.580078	9545.250000	9506.089844	9523.790039	9523.790039	315344000
6	2013-07-18	9537.769531	9601.080078	9537.769531	9587.200195	9587.200195	345237000
7	2013-07-19	9587.490234	9618.500000	9565.919922	9618.500000	9618.500000	330258000
8	2013-07-22	9626.009766	9658.080078	9621.969727	9650.580078	9650.580078	277913000
9	2013-07-23	9672.299805	9676.940430	9644.129883	9659.629883	9659.629883	309618000
10	2013-07-24	9681.469727	9682.110352	9586.389648	9605.040039	9605.040039	333612000
11	2013-07-25	9577.709961	9636.360352	9566.299805	9635.059570	9635.059570	332250000
12	2013-07-26	9600.190430	9621.059570	9540.959961	9620.129883	9620.129883	276277000
13	2013-07-29	9594.719727	9601.530273	9554.910156	9571.790039	9571.790039	284052000
14	2013-07-30	9585.830078	9605.190430	9536.480469	9556.169922	9556.169922	332053000
15	2013-07-31	9571.179688	9631.049805	9555.719727	9558.830078	9558.830078	384739000
16	2013-08-01	9617.290039	9679.919922	9617.290039	9673.400391	9673.400391	377517000
17	2013-08-02	9660.719727	9690.099609	9645.049805	9690.070313	9690.070313	313663000
18	2013-08-05	9669.799805	9679.919922	9646.120117	9671.599609	9671.599609	252930000
19	2013-08-06	9661.690430	9661.780273	9593.129883	9614.320313	9614.320313	314121000
20	2013-08-07	9581.669922	9584.839844	9550.509766	9568.259766	9568.259766	301023000

	Date	Open	High	Low	Close	Adj Close	Volum
21	2013-08-08	9613.360352	9645.750000	9575.790039	9634.700195	9634.700195	32716600(
22	2013-08-09	9626.089844	9649.519531	9589.250000	9622.110352	9622.110352	29576700(
23	2013-08-12	9585.219727	9618.219727	9577.500000	9609.030273	9609.030273	27891600(
24	2013-08-13	9616.440430	9644.519531	9577.389648	9630.570313	9630.570313	30355600(
25	2013-08-14	9628.040039	9636.419922	9589.790039	9593.500000	9593.500000	28714300(
26	2013-08-15	9524.759766	9524.759766	9452.139648	9489.290039	9489.290039	34266900(
27	2013-08-16	9489.280273	9501.360352	9449.730469	9465.589844	9465.589844	32114500(
28	2013-08-19	9455.000000	9459.209961	9385.009766	9385.889648	9385.889648	29045300(
29	2013-08-20	9385.889648	9451.990234	9374.910156	9421.559570	9421.559570	29940900(
...	...	...	...	...	...	...	
1481	2019-05-29	12398.240234	12405.110352	12309.129883	12386.209961	12386.209961	37000500(
1482	2019-05-30	12402.540039	12449.280273	12354.790039	12393.660156	12393.660156	32737900(
1483	2019-05-31	12280.780273	12318.459961	12238.400391	12264.490234	12264.490234	39810200(
1484	2019-06-03	12288.509766	12372.820313	12273.089844	12341.820313	12341.820313	39438100(
1485	2019-06-04	12341.820313	12564.370117	12341.820313	12560.580078	12560.580078	38104300(
1486	2019-06-05	12607.349609	12618.219727	12530.150391	12614.700195	12614.700195	35488300(
1487	2019-06-06	12631.089844	12710.669922	12606.120117	12675.660156	12675.660156	33964100(
1488	2019-06-07	12725.379883	12814.809570	12725.379883	12765.860352	12765.860352	32202500(
1489	2019-06-10	12821.490234	12859.629883	12798.290039	12802.030273	12802.030273	32092100(
1490	2019-06-11	12867.169922	12900.769531	12787.490234	12813.910156	12813.910156	35484200(
1491	2019-06-12	12797.860352	12818.599609	12772.330078	12784.830078	12784.830078	30341300(
1492	2019-06-13	12816.429688	12841.719727	12788.540039	12827.509766	12827.509766	30698100(
1493	2019-06-14	12808.889648	12813.000000	12767.849609	12787.240234	12787.240234	29223300(
1494	2019-06-17	12789.679688	12808.120117	12769.530273	12775.519531	12775.519531	28101400(

	Date	Open	High	Low	Close	Adj Close	Volum
1495	2019-06-18	12837.469727	12950.799805	12837.469727	12907.459961	12907.459961	34376200(
1496	2019-06-19	12924.269531	12972.480469	12908.410156	12954.080078	12954.080078	32878900(
1497	2019-06-20	12954.080078	13094.799805	12954.080078	13081.540039	13081.540039	39059400(
1498	2019-06-21	13068.360352	13095.570313	13037.879883	13047.240234	13047.240234	50001200(
1499	2019-06-24	13047.240234	13061.250000	13015.910156	13019.980469	13019.980469	31362500(
1500	2019-06-25	13019.980469	13022.730469	12934.900391	12936.650391	12936.650391	35780500(
1501	2019-06-26	12936.650391	12977.919922	12911.990234	12912.009766	12912.009766	34781300(
1502	2019-06-27	12912.059570	12976.230469	12912.059570	12965.320313	12965.320313	31229200(
1503	2019-06-28	12965.549805	13055.860352	12965.549805	13049.709961	13049.709961	54207000(
1504	2019-07-01	13049.709961	13191.900391	13049.709961	13127.690430	13127.690430	35132700(
1505	2019-07-02	13127.690430	13152.950195	13105.030273	13152.549805	13152.549805	32068400(
1506	2019-07-03	13152.549805	13152.549805	13152.549805	13152.549805	13152.549805	19637200(
1507	2019-07-05	13231.799805	13231.799805	13114.839844	13210.910156	13210.910156	24342100(
1508	2019-07-08	13210.870117	13210.870117	13136.219727	13147.219727	13147.219727	29045500(
1509	2019-07-09	13147.209961	13147.209961	13091.900391	13136.889648	13136.889648	30282100(
1510	2019-07-10	13137.959961	13221.059570	13137.959961	13173.259766	13173.259766	31542400(

1511 rows × 8 columns

In [9]:
```
dlydata = pd.read_csv("dlydata.csv")
dlydata.rename(columns={'A': 'Symbol', '2014-10-20': 'Date', '52.0700': 'Star
               '52.3500': 'Closing', '4942200': 'Volume', '201443': 'Week'}, ir
```

In [10]:
```
dlydata[dlydata['Symbol']=='Z']
```

Out[10]:

	Symbol	Date	Starting	High	Low	Closing	Volume	Week
1034481	Z	2014-10-20	104.66	108.66	104.3100	107.80	1045700	201443
1034482	Z	2014-10-21	109.10	110.40	107.6200	109.01	893600	201443
1034483	Z	2014-10-22	108.62	110.34	103.5400	104.43	1104400	201443
1034484	Z	2014-10-23	105.51	110.00	104.5000	108.45	1287200	201443
1034485	Z	2014-10-24	108.08	108.13	104.8000	106.07	849800	201443
1034486	Z	2014-10-27	105.75	107.83	104.7100	105.55	633700	201444
1034487	Z	2014-10-28	106.06	108.66	104.7200	107.18	1051400	201444
1034488	Z	2014-10-29	107.02	107.02	104.2500	105.52	967300	201444
1034489	Z	2014-10-30	100.50	105.65	98.6000	104.24	1674400	201444
1034490	Z	2014-10-31	106.80	109.66	105.4300	108.73	1373100	201444
1034491	Z	2014-11-03	109.15	111.34	106.5500	107.87	1345600	201445
1034492	Z	2014-11-04	107.17	108.67	106.3700	107.57	855900	201445
1034493	Z	2014-11-05	108.80	109.00	102.3600	103.77	2330800	201445
1034494	Z	2014-11-06	94.68	103.47	94.2300	99.37	4541900	201445
1034495	Z	2014-11-07	99.85	103.72	99.4800	103.20	1459800	201445
1034496	Z	2014-11-10	104.44	106.81	103.5300	106.07	1710000	201446
1034497	Z	2014-11-11	106.00	108.89	106.0000	108.77	996800	201446
1034498	Z	2014-11-12	108.45	109.61	105.6900	106.19	1144600	201446
1034499	Z	2014-11-13	107.00	107.11	101.5500	102.46	914400	201446
1034500	Z	2014-11-14	102.67	114.67	102.0500	113.71	2649100	201446
1034501	Z	2014-11-17	112.90	123.26	112.2600	117.86	4441000	201447
1034502	Z	2014-11-18	117.90	121.56	115.6200	118.96	1533500	201447
1034503	Z	2014-11-19	124.50	127.89	120.6500	120.96	3678500	201447
1034504	Z	2014-11-20	120.40	122.03	115.7500	116.36	1533400	201447
1034505	Z	2014-11-21	117.74	117.83	114.5000	115.02	1453700	201447
1034506	Z	2014-11-24	115.20	117.00	113.7500	116.62	1332000	201448
1034507	Z	2014-11-25	116.78	117.65	114.5600	115.83	775700	201448
1034508	Z	2014-11-26	116.08	120.21	115.5000	120.21	907200	201448
1034509	Z	2014-11-27	120.21	120.21	120.2100	120.21	0	201448
1034510	Z	2014-11-28	121.48	122.04	117.3900	118.36	542000	201448
...	...	...	...	...	...	...	...	...
1035552	Z	2019-04-05	37.35	37.74	36.9400	37.27	1546000	201914
1035553	Z	2019-04-08	38.56	38.94	37.4500	37.59	3486400	201915
1035554	Z	2019-04-09	37.56	37.82	36.9500	37.18	1780800	201915

	Symbol	Date	Starting	High	Low	Closing	Volume	Week
1035555	Z	2019-04-10	36.87	37.45	36.7500	37.08	1159400	201915
1035556	Z	2019-04-11	36.98	37.36	36.3400	37.10	2042000	201915
1035557	Z	2019-04-12	37.41	37.70	37.0600	37.42	1440700	201915
1035558	Z	2019-04-15	37.24	37.83	37.2300	37.47	1807900	201916
1035559	Z	2019-04-16	37.59	38.41	37.5287	38.17	1339500	201916
1035560	Z	2019-04-17	38.24	38.36	36.7900	36.98	1502000	201916
1035561	Z	2019-04-18	36.83	37.01	35.8500	36.28	1744000	201916
1035562	Z	2019-04-22	36.20	36.24	34.8700	34.92	2664000	201917
1035563	Z	2019-04-23	35.08	36.42	34.9700	35.67	2409900	201917
1035564	Z	2019-04-24	35.59	35.86	34.1400	34.57	3410600	201917
1035565	Z	2019-04-25	34.82	34.82	33.6700	34.11	2057100	201917
1035566	Z	2019-04-26	34.17	34.35	33.7300	34.25	1566500	201917
1035567	Z	2019-04-29	34.13	34.58	33.7500	33.93	3169600	201918
1035568	Z	2019-04-30	33.83	33.98	32.8500	33.40	6857300	201918
1035569	Z	2019-05-01	33.55	33.63	32.4800	32.49	2405300	201918
1035570	Z	2019-05-02	32.50	32.81	31.7500	32.33	3540300	201918
1035571	Z	2019-05-03	32.51	33.94	32.2700	33.85	2963800	201918
1035572	Z	2019-05-06	32.92	35.23	32.6400	34.43	3928500	201919
1035573	Z	2019-05-07	33.92	34.44	33.3200	33.81	2436900	201919
1035574	Z	2019-05-08	33.62	35.17	33.5500	34.85	3436100	201919
1035575	Z	2019-05-09	34.31	35.00	33.6700	34.27	6312300	201919
1035576	Z	2019-05-10	38.80	39.80	35.0800	36.00	20064400	201919
1035577	Z	2019-05-13	35.13	36.88	34.5500	35.86	4680800	201920
1035578	Z	2019-05-14	36.24	37.64	35.7500	36.90	3905400	201920
1035579	Z	2019-05-15	37.33	39.09	37.1200	38.56	4816800	201920
1035580	Z	2019-05-16	38.27	39.85	38.2100	39.13	5154500	201920
1035581	Z	2019-05-17	38.66	40.79	38.6000	40.25	3828200	201920

1101 rows × 8 columns

In [9]:

```
def find_nearest_date(future_str_date, date_list):
    while (future_str_date not in date_list.values):
        next_date = datetime.strptime(future_str_date,"%Y-%m-%d")
        future_date = next_date + timedelta(days = 1)
        future_str_date = future_date.strftime('%Y-%m-%d')
    index = list(date_list.values).index(future_str_date)
    return future_str_date, index
```

In [134]:
```
def calculate_averages(symbol,dlydata, num_days):
    data = dlydata[dlydata['Symbol']==symbol]
    date_list = data['Date']
    max_date = datetime.strptime(date_list.iloc[len(date_list)-1],"%Y-%m-%d")
    moving_averages = {}
    sum_averages = {}
    prev_end = None
    prev_iter_date = None
    for i in range(0, len(date_list)):
        if i == 0:
            str_date = date_list.iloc[i]
            this_date = datetime.strptime(str_date,"%Y-%m-%d")
            future_date = this_date + timedelta(days=num_days)
            future_str_date = future_date.strftime('%Y-%m-%d')
            if future_date + timedelta(3) <= max_date:
                future_str_date, index = find_nearest_date(future_str_date, d
                sum_avg = 0
                for j in range(i,index+1):
                    sum_avg += data[data['Date']==date_list.iloc[j]].iloc[0][
                moving_averages[future_str_date] = sum_avg/(index + 1 - i)
                sum_averages[future_str_date] = sum_avg
        else:
            curr_date = date_list.iloc[i]
            prev_date = date_list.iloc[i-1]
            time_diff = (datetime.strptime(curr_date, "%Y-%m-%d") - datetime.
            future_date = datetime.strptime(prev_iter_date, "%Y-%m-%d") + tim
            future_str_date = future_date.strftime('%Y-%m-%d')
            if future_date + timedelta(3) <= max_date:
                future_str_date, index = find_nearest_date(future_str_date, d
                add = 0
                for j in range(prev_end, index+1):
                    add += data[data['Date']==date_list.iloc[j]].iloc[0]['Clo
                subtract = data[data['Date']==date_list.iloc[prev_end-1]].ilo
                this_sum = sum_averages[prev_iter_date] - subtract + add
                sum_averages[future_str_date] = this_sum
                moving_averages[future_str_date] = this_sum/(index - i)
        prev_iter_date = future_str_date
        prev_end = index+1
    return moving_averages
```

In [11]:
```
def percent_positive(percent_change):
    if len(percent_change)==0:
        return 0
    return len([percent_change[i] for i in percent_change.keys() if percent_c
```

In [12]:
```
def create_dict(data):
    dict = {}
    for i,row in data.iterrows():
        dict[row["Date"]] = row["Closing"]
    return dict
```

In [132]:
```
def populate_df(dict, averages, data, num_days):
    new_averages = []
    for key in dict:
        if key in list(averages.keys()):
            new_averages.append(averages[key])
        else:
            new_averages.append(None)
    data[str(num_days) + " Day Moving Average"] = new_averages
    return data
```

In [14]:
```
def add_market_performance(data, snp, nyse, nasdaq):
    def format_snp(date, date_list, dset):
        this_date = date
        first = this_date[0:6]
        second = this_date[6:]
        d1 = first.index('-')
        d2 = second.index('-') + 6
        mid = this_date[d1+1:d2]
        end = this_date[d2+1:]
        if mid[0] == '0':
            mid = mid[1:]
        if end[0] == '0':
            end = end[1:]
        formatted = mid + "/" + end + "/" + this_date[0:d1]
        if formatted in date_list:
#             print (True)
#             print(dset[dset["Date"] == date].iloc[0]["Change"])
            return dset[dset["Date"] == formatted].iloc[0]["Change"]
        else:
            return None
    def format_other(date,date_list, dset):
        if date in date_list:
            return dset[dset["Date"] == date].iloc[0]["Change"]
        else:
            return None
    def snp_change(row):
        return format_snp(row["Date"], snp["Date"].values, snp)
    def nyse_change(row):
        return format_other(row["Date"],nyse["Date"].values, nyse)
    def nasdaq_change(row):
        return format_other(row["Date"],nasdaq["Date"].values, nasdaq)

    data["snp_change"] = data.apply(snp_change,axis=1)
    data["nyse_change"] = data.apply(nyse_change,axis=1)
    data["nasdaq_change"] = data.apply(nasdaq_change,axis=1)

#     print(data["snp_change"])

    return data
```

In [26]:
```
def convertToDate(datetext):
    week_days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
    abbr_to_num = {"Jan":"01","Feb":"02","Mar":"03","Apr":"04","May":"05","Ju
                   "Sep": "09","Oct": "10","Nov": "11","Dec": "12"}
    year = None
    month = None
    day = None
    if datetext[0:5] == "Today":
        d = datetime.today()
        return d.strftime('%Y-%m-%d')
    if datetext[0:9] == "Yesterday":
        d = datetime.today()
        yest = d - timedelta(days=1)
        return yest.strftime('%Y-%m-%d')
    elif datetext[0:3] in week_days:
        year = datetime.now().strftime("%Y")
        comma_index_2 = datetext[5:].index(",") + 5
        month = abbr_to_num[datetext[5:8]]
        day = datetext[10:comma_index_2]
        if month == "05":
            day = datetext[9:comma_index_2]
    else:
        comma_index_1 = datetext.index(",")
        year = datetext[comma_index_1:][2:6]
        month = abbr_to_num[datetext[0:3]]
        day = datetext[5:comma_index_1]
        if month == "05":
            day = datetext[4:comma_index_1]
    corr_date = year + "-" + month + "-" + day
    return corr_date

def date_conversion(datetext):
    abbr_to_num = {"Jan":"01","Feb":"02","Mar":"03","Apr":"04","May":"05","Ju
                   "Sep": "09","Oct": "10","Nov": "11","Dec": "12"}
    comma_index_1 = datetext.index(",")
    year = datetext[comma_index_1:][2:6]
    month = abbr_to_num[datetext[0:3]]
    day = datetext[4:comma_index_1]
    corr_date = year + "-" + month + "-" + day
    return corr_date

date_conversion("Aug 06, 2019")
```

Out[26]: '2019-08-06'

In [27]:
```
def getSentiment(ticker, start_date):
    lower_ticker = ticker.lower()
    date_dict = {}
    url = 'https://seekingalpha.com/symbol/' + ticker + '?s=' + lower_ticker
    source = requests.get(url)

    driver = webdriver.Edge(executable_path="C:/Users/Adhithya/Desktop/Micros
    driver.get(url)
    # html = driver.execute_script("return document.documentElement.innerHTML

#     driver.maximize_window()
    d_obj = datetime.today()
    print(d_obj)
    vars = None
    while d_obj > start_date:
        vars = driver.find_elements_by_class_name("mc_list_li")
        date = vars[len(vars)-1].find_element_by_class_name("date")
        corr_date = convertToDate(date.text)
        d_obj = datetime.strptime(corr_date, '%Y-%m-%d')
        driver.execute_script("window.scrollBy(0,document.body.scrollHeight)"


    for var in vars:
        # driver.execute_script("arguments[0].scrollIntoView();", var)
        list_items = var.find_elements_by_tag_name("li")
        avg_polarity = 0
        for l_item in list_items:
            analysis = TextBlob(l_item.text)
            avg_polarity += analysis.sentiment.polarity
        if len(list_items) == 0:
            avg_polarity = None
        else:
            avg_polarity = avg_polarity/len(list_items)
        date = var.find_element_by_class_name("date")
        corr_date = convertToDate(date.text)
        d_obj = datetime.strptime(corr_date,'%Y-%m-%d')
        date_dict[corr_date] = avg_polarity

    driver.quit()
    return date_dict
```

In [28]:
```
def add_sentiment(stock_df, sentiment_dict):

    def sentiment(date):
        if date not in sentiment_dict.keys():
            return None
        return sentiment_dict[date]
    stock_df["Sentiment"] = stock_df["Date"].apply(sentiment)
    return stock_df
```

In [18]:

```
def add_sector_performance(url, sector, sector_dict):
    stock_list = []
    driver = webdriver.Edge(executable_path="C:/Users/Adhithya/Desktop/Micros
    offset = 0
    ind = url.index("sector")
    prefix = url[0:ind]
    sector_part = url[(ind + 7):]
    url_base = prefix + "screener/predefined/" + sector_part
    while True:
        try:
#             url = sector_url + "&count=25&offset=" + str(offset)
            print(url)
            driver.get(url)
            print(offset)
            mytable = driver.find_element_by_css_selector("#scr-res-table > d
            rows = mytable.find_elements_by_tag_name("tr")
            offset += len(rows[1:])
            for row in rows[1:]:
                cells = row.find_elements_by_tag_name("td")
                ticker = cells[0].text
                new_line_index = ticker.index('\n')
                stock_list.append(ticker[0:new_line_index])
#                 print(cells[0].text)
            if len(rows) < 25:
                break
            if offset == 25:
                url = url_base + "?offset=25&count=25"
            else:
                url = url_base + "?offset=" + str(offset) + "&count=25"
        except Exception as e:
            print("exception")
            driver.quit()
#             if isinstance(e, NoSuchElementException):
#                 print("no such element")
#                 url = sector_url + "?count=25&offset=" + str(offset)
#             else:
            url = url_base + "?offset=" + str(offset) + "&count=25"
            driver = webdriver.Edge(executable_path="C:/Users/Adhithya/Deskto
            driver.get(url)
    sector_dict[sector] = stock_list
    driver.quit()
    return sector_dict


# https://finance.yahoo.com/screener/predefined/ms_utilities?offset=25&count=
```

In [19]:
```
sector_urls = {}
sector_urls["Basic Materials"] = "https://finance.yahoo.com/sector/ms_basic_m
sector_urls["Communication Services"] = "https://finance.yahoo.com/sector/ms_
sector_urls["Consumer Cyclical"] = "https://finance.yahoo.com/sector/ms_consu
sector_urls["Consumer Defensive"] = "https://finance.yahoo.com/sector/ms_cons
sector_urls["Energy"] = "https://finance.yahoo.com/sector/ms_energy"
sector_urls["Financial Services"] = "https://finance.yahoo.com/sector/ms_fina
sector_urls["Healthcare"] = "https://finance.yahoo.com/sector/ms_healthcare"
sector_urls["Industrials"] = "https://finance.yahoo.com/sector/ms_industrials
sector_urls["Real Estate"] = "https://finance.yahoo.com/sector/ms_real_estate
sector_urls["Technology"] = "https://finance.yahoo.com/sector/ms_technology"
sector_urls["Utilities"] = "https://finance.yahoo.com/sector/ms_utilities"
```

In [54]:
```
def populate_sector_stocks():
    sector_dictionary = {}
    for sector in sector_urls.keys():
        print("DONE")
        url = sector_urls[sector]
        sector_dictionary = add_sector_performance(url, sector, sector_dictic
    return sector_dictionary
```

In [55]:
```
sector_dictionary = populate_sector_stocks()
```

```
25
https://finance.yahoo.com/screener/predefined/ms_basic_materials?offset=
50&count=25 (https://finance.yahoo.com/screener/predefined/ms_basic_mate
rials?offset=50&count=25)
50
https://finance.yahoo.com/screener/predefined/ms_basic_materials?offset=
75&count=25 (https://finance.yahoo.com/screener/predefined/ms_basic_mate
rials?offset=75&count=25)
75
https://finance.yahoo.com/screener/predefined/ms_basic_materials?offset=
100&count=25 (https://finance.yahoo.com/screener/predefined/ms_basic_mat
erials?offset=100&count=25)
100
https://finance.yahoo.com/screener/predefined/ms_basic_materials?offset=
125&count=25 (https://finance.yahoo.com/screener/predefined/ms_basic_mat
erials?offset=125&count=25)
125
https://finance.yahoo.com/screener/predefined/ms_basic_materials?offset=
150&count=25 (https://finance.yahoo.com/screener/predefined/ms_basic_mat
erials?offset=150&count=25)
```

In [56]:
```
saved_sector_dict = sector_dictionary
```

In [107]:

```
def get_sector_day_averages(averages_dict, stock_list):
    curr_stock_dict = {}
    lowest_date = datetime.strptime("2014-01-01","%Y-%m-%d")

    api_key = "S52ZB5D8LU3M5FN0"

    for ticker in stock_list:
        print(ticker)
        print(key)
        API_URL = "https://www.alphavantage.co/query"

        data = {
            "function": "TIME_SERIES_DAILY",
            "symbol": ticker,
            "outputsize": "full",
            "datatype": "json",
            "apikey": api_key
            }

        response = requests.get(API_URL, data)
        json = response.json()

        if "Error Message" not in json.keys():
            json_data = json["Time Series (Daily)"]
            for date in json_data.keys():
                this_date = datetime.strptime(date,"%Y-%m-%d")
                if this_date > lowest_date:
                    price_data = json_data[date]
                    open = float(price_data['1. open'])
                    close = float(price_data['4. close'])
                    perc_change = (close - open)/open
                    if date in averages_dict.keys():
                        existing = averages_dict[date]
                        existing.append(perc_change)
                        averages_dict[date] = existing
                    else:
                        averages_dict[date] = [perc_change]

    return averages_dict


# get_sector_day_averages(['CTA-PA'])
```

In [106]:
```
sector_day_averages = {}
curr_index = 0
count = 0
key_index = 0
key = api_keys_list[key_index]
num_keys = len(api_keys_dict.keys())
for sector in sector_dictionary:
    print(sector)
    this_dict = {}
    sector_stocks = sector_dictionary[sector]
    symbol_list = random.sample(list(sector_stocks), 40)
    i = 0
    while i < len(symbol_list):
        sector_subgroup = symbol_list[i:i+5]
        this_dict = get_sector_day_averages(this_dict, sector_subgroup)
        i += 5
        time.sleep(60)
    sector_day_averages[sector] = this_dict
```

```
Basic Materials
OEC
G1B0U3OOHJIA9YUZ
AVD
G1B0U3OOHJIA9YUZ
GOLD
G1B0U3OOHJIA9YUZ
BLDR
G1B0U3OOHJIA9YUZ
BTU
G1B0U3OOHJIA9YUZ
DOOR
G1B0U3OOHJIA9YUZ
USLM
G1B0U3OOHJIA9YUZ
CLW
G1B0U3OOHJIA9YUZ
CBT
G1B0U3OOHJIA9YUZ
```

In [108]:
```
saved_sector_day_averages = sector_day_averages
```

In [110]:
```
new_sector_averages = {}
for sector in sector_day_averages:
    sector_dict = sector_day_averages[sector]
    this_dict = {}
    for date in sector_dict:
        day_values = sector_dict[date]
        mean = np.mean(day_values)
        this_dict[date] = mean
    new_sector_averages[sector] = this_dict
```

In [113]:
```
sector_day_averages = new_sector_averages
```

In [115]:
```
def calculations(data,moving_averages,dict):
    three_month = {}
    six_month = {}
    twelve_month = {}
    date_list = data['Date']
    max_date = datetime.strptime(list(moving_averages.keys())[len(moving_aver
    for key in list(moving_averages.keys()):
        key_date = datetime.strptime(key,'%Y-%m-%d')
        curr_price = dict[key]
        plus_three = key_date + timedelta(days=90)
        plus_three_str = plus_three.strftime('%Y-%m-%d')
        plus_six = key_date + timedelta(days=180)
        plus_six_str = plus_six.strftime('%Y-%m-%d')
        plus_twelve = key_date + timedelta(days=360)
        plus_twelve_str = plus_twelve.strftime('%Y-%m-%d')
        if plus_three < max_date:
            plus_three_str, index = find_nearest_date(plus_three_str, date_li
            plus_three_price = dict[plus_three_str]
            three_percent_change = (plus_three_price - curr_price)/curr_price
        if plus_six < max_date:
            plus_six_str, index = find_nearest_date(plus_six_str, date_list)
            plus_six_price = dict[plus_six_str]
            six_percent_change = (plus_six_price - curr_price)/curr_price
        if plus_twelve < max_date:
            plus_twelve_str, index = find_nearest_date(plus_twelve_str, date_
            plus_twelve_price = dict[plus_twelve_str]
            twelve_percent_change = (plus_twelve_price - curr_price)/curr_pri
        if curr_price > moving_averages[key]:
            three_month[key] = three_percent_change
            six_month[key] = six_percent_change
            twelve_month[key] = twelve_percent_change
    return three_month, six_month, twelve_month
```

In [210]:
```
def add_sectors(stock_df, sector_dictionary, sector_averages):

    def sector_name(df):
        for sector in sector_dictionary.keys():
            if df["Symbol"] in sector_dictionary[sector]:
                return sector
        return None

    def sector_data(df):
        symbol = df["Symbol"]
        date = df["Date"]
        symbol_sector = None
        for sector in sector_dictionary.keys():
            if symbol in sector_dictionary[sector]:
                symbol_sector = sector

        if symbol_sector is None:
            return None

        day_data = sector_averages[symbol_sector]
        if date not in day_data.keys():
            return None
        return day_data[date]

    stock_df["Sector"] = stock_df.apply(sector_name, axis=1)
    stock_df["Sector Change"] = stock_df.apply(sector_data, axis=1)
    return stock_df
```

In [15]:
```
stocks = dlydata['Symbol'].unique()
# symbol_list = random.sample(list(stocks), 3)
symbol_list = stocks[0:int(len(stocks)/2)]
```

In [16]:
```
stock_df = pd.DataFrame(columns=['Symbol', 'Date', 'Starting', 'High', 'Low',
                                 'nyse_change','nasdaq_change'])
```

In [17]:
```
symbol_list
```

Out[17]:
```
array(['A', 'AA', 'AAL', 'AAP', 'AAPL', 'ABBV', 'ABC', 'ABMD', 'ABT',
       'ACAD', 'ACC', 'ACGL', 'ACHC', 'ACM', 'ACN', 'ADBE', 'ADI', 'ADM',
       'ADNT', 'ADP', 'ADS', 'ADSK', 'AEE', 'AEP', 'AES', 'AET', 'AFG',
       'AFL', 'AGCO', 'AGIO', 'AGN', 'AGNC', 'AGO', 'AGR', 'AHL', 'AIG',
       'AIV', 'AIZ', 'AJG', 'AKAM', 'AKRX', 'AL', 'ALB', 'ALGN', 'ALK',
       'ALKS', 'ALL', 'ALLE', 'ALLY', 'ALNY', 'ALR', 'ALSN', 'ALXN',
       'AMAT', 'AMCX', 'AMD', 'AME', 'AMG', 'AMGN', 'AMH', 'AMP', 'AMT',
       'AMTD', 'AMZN', 'AN', 'ANAT', 'ANET', 'ANSS', 'ANTM', 'AON', 'AOS',
       'APA', 'APC', 'APD', 'APH', 'APLE', 'AR', 'ARD', 'ARE', 'ARMK',
       'ARNC', 'ARRS', 'ARW', 'ASB', 'ASH', 'ATH', 'ATHN', 'ATO', 'ATR',
       'ATVI', 'AVB', 'AVGO', 'AVT', 'AVY', 'AWH', 'AWI', 'AWK', 'AXP',
       'AXS', 'AXTA', 'AYI', 'AZO', 'BA', 'BAC', 'BAH', 'BAX', 'BBBY',
       'BBT', 'BBY', 'BC', 'BCR', 'BDN', 'BDX', 'BEN', 'BERY', 'BF.A',
       'BF.B', 'BFAM', 'BG', 'BGCP', 'BHI', 'BIIB', 'BIO', 'BIVV', 'BK',
       'BKD', 'BKFS', 'BKU', 'BLK', 'BLL', 'BMRN', 'BMS', 'BMY', 'BOH',
       'BOKF', 'BPOP', 'BR', 'BRCD', 'BRK.B', 'BRKR', 'BRO', 'BRX', 'BSX',
       'BUFF', 'BURL', 'BWA', 'BWXT', 'BXP', 'C', 'CA', 'CAA', 'CAB',
       'CABO', 'CACC', 'CAG', 'CAH', 'CASY', 'CAT', 'CAVM', 'CB', 'CBG',
       'CBOE', 'CBS', 'CBSH', 'CBT', 'CC', 'CCI', 'CCK', 'CCL', 'CDEV',
       'CDK', 'CDNS', 'CDW', 'CE', 'CELG', 'CERN', 'CF', 'CFG', 'CFR',
       'CFX', 'CGNX', 'CHD', 'CHH', 'CHK', 'CHRW', 'CHTR', 'CI', 'CIM',
       'CINF', 'CIT', 'CL', 'CLGX', 'CLH', 'CLNS', 'CLR', 'CLX', 'CMA',
       'CMCSA', 'CME', 'CMG', 'CMI', 'CMS', 'CNA', 'CNC', 'CNDT', 'CNK',
       'CNP', 'CNX', 'COF', 'COG', 'COH', 'COHR', 'COL', 'COMM', 'CONE',
       'COO', 'COP', 'COR', 'COST', 'COTY', 'CPA', 'CPB', 'CPN', 'CPRT',
       'CPT', 'CR', 'CRI', 'CRL', 'CRM', 'CSCO', 'CSGP', 'CSL', 'CSRA',
       'CST', 'CSX', 'CTAS', 'CTL', 'CTSH', 'CTXS', 'CUBE', 'CVS', 'CVX',
       'CXO', 'CXP', 'CXW', 'CY', 'D', 'DAL', 'DATA', 'DCI', 'DCT', 'DD',
       'DDR', 'DE', 'DEI', 'DFS', 'DFT', 'DG', 'DGX', 'DHI', 'DHR', 'DIS',
       'DISCA', 'DISCK', 'DISH', 'DKS', 'DLB', 'DLPH', 'DLR', 'DLTR',
       'DNB', 'DNKN', 'DOV', 'DOW', 'DOX', 'DPS', 'DPZ', 'DRE', 'DRI',
       'DST', 'DTE', 'DUK', 'DVA', 'DVMT', 'DVN', 'DXC', 'DXCM', 'EA',
       'EBAY', 'ECL', 'ED', 'EEFT', 'EFX', 'EGN', 'EIX', 'EL', 'ELS',
       'EMN', 'EMR', 'ENDP', 'ENR', 'EOG', 'EPC', 'EPR', 'EQC', 'EQIX',
       'EQR', 'EQT', 'ERIE', 'ES', 'ESRT', 'ESRX', 'ESS', 'ETFC', 'ETN',
       'ETR', 'EV', 'EVHC', 'EW', 'EWBC', 'EXC', 'EXEL', 'EXP', 'EXPD',
       'EXPE', 'EXR', 'F', 'FAF', 'FANG', 'FAST', 'FB', 'FBHS', 'FCE.A',
       'FCX', 'FDC', 'FDS', 'FDX', 'FE', 'FEYE', 'FFIV', 'FHB', 'FHN',
       'FII', 'FIS', 'FISV', 'FITB', 'FL', 'FLIR', 'FLO', 'FLR', 'FLS',
       'FLT', 'FMC', 'FNB', 'FND', 'FNF', 'FOX', 'FOXA', 'FRC', 'FRT',
       'FSLR', 'FTNT', 'FTV', 'FWONA', 'FWONK', 'G', 'GD', 'GDDY', 'GDI',
       'GE', 'GGG', 'GGP', 'GHC', 'GILD', 'GIS', 'GLPI', 'GLW', 'GM',
       'GME', 'GNTX', 'GOOG', 'GOOGL', 'GPC', 'GPK', 'GPN', 'GPOR', 'GPS',
       'GRA', 'GRMN', 'GS', 'GT', 'GWR', 'GWRE', 'GWW', 'GXP', 'H',
       'HAIN', 'HAL', 'HAS', 'HBAN', 'HBI', 'HCA', 'HCN', 'HCP', 'HD',
       'HDS', 'HE', 'HEI', 'HEI.A', 'HES', 'HFC', 'HGV', 'HHC', 'HIG',
       'HII', 'HIW', 'HLF', 'HLT', 'HOG', 'HOLX', 'HON', 'HP', 'HPE',
       'HPP', 'HPQ', 'HPT', 'HRB', 'HRC', 'HRL', 'HRS', 'HSIC', 'HST',
       'HSY', 'HTA', 'HUBB', 'HUM', 'HUN', 'HXL', 'IAC', 'IBKR', 'IBM',
       'ICE', 'ICPT', 'IDXX', 'IEX', 'IFF', 'IGT', 'ILMN', 'INCY', 'INFO',
       'INGR', 'INT', 'INTC', 'INTU', 'INVH', 'IONS', 'IP', 'IPG', 'IPGP',
       'IR', 'IRM', 'ISRG', 'IT', 'ITT', 'ITW', 'IVZ', 'JBHT', 'JBL',
       'JBLU', 'JCI', 'JEC', 'JKHY', 'JLL', 'JNJ', 'JNPR', 'JPM', 'JUNO',
       'JW.A', 'JWN', 'K', 'KAR', 'KATE', 'KEX', 'KEY', 'KEYS', 'KHC',
```

```
       'KIM', 'KLAC', 'KMB', 'KMI', 'KMX', 'KO', 'KORS', 'KOS', 'KR'],
      dtype=object)
```

In [204]:

```
three_list = {}
six_list = {}
twelve_list = {}
for symbol in symbol_list:
    print(symbol)
    symbol_data = dlydata[dlydata['Symbol']==symbol]
    first_date = symbol_data.iloc[0]["Date"]
#     print(first_date)
    dict = create_dict(symbol_data)
    averages200 = calculate_averages(symbol, dlydata, 200)
    averages90 = calculate_averages(symbol, dlydata, 90)
    averages30 = calculate_averages(symbol, dlydata, 30)
    data = populate_df(dict, averages200, symbol_data, 200)
    data = populate_df(dict, averages90, data, 90)
    data = populate_df(dict, averages30, data, 30)
    data = add_market_performance(data,snp, nyse, nasdaq)
    stock_df = stock_df.append(data, sort=False)


#     three_month, six_month, twelve_month = calculations(symbol_data, averag
#     three_percent, six_percent, twelve_percent = percent_positive(three_mor
#     three_list[symbol] = three_percent
#     six_list[symbol] = six_percent
#     twelve_list[symbol] = twelve_percent
df = add_sectors(stock_df, symbol, sector_dictionary, sector_day_averages)
df.to_csv("C:/Users/Adhithya/Desktop/output.csv", index=True, header=True)
```

```
A
```

In [211]:

```
df = add_sectors(stock_df, sector_dictionary, sector_day_averages)
```

In [213]:
```
df.to_csv("C:/Users/Adhithya/Desktop/sector_output.csv", index=True, header=T
```

In [11]:
```
csv = pd.read_csv("C:/Users/Adhithya/Desktop/sector_output.csv")
```

In [12]:
```
csv
```

Out[12]:

	Unnamed: 0	Symbol	Date	Starting	High	Low	Closing	Volume	Week
0	0	A	2014-10-21	52.86	54.0300	52.860	54.00	2231800	201443
1	1	A	2014-10-22	53.77	53.8600	52.990	53.05	2161000	201443
2	2	A	2014-10-23	53.59	54.2100	53.590	53.71	1754700	201443
3	3	A	2014-10-24	53.70	54.1100	53.510	54.05	1508100	201443
4	4	A	2014-10-27	53.94	54.1100	53.600	53.99	1105200	201444
5	5	A	2014-10-28	54.12	54.6700	53.880	54.66	1714300	201444

In [ ]:
```

```