

```
In [226]: import seaborn as sns
import matplotlib.pyplot as pyplot
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import math import sqrt
from datetime import timedelta

# Define list of stocks to conduct event analysis on.
symbols_list = ['AAPL', 'AMZN', 'GOOGL', 'MSFT', 'FB', 'NFLX', 'CRM', 'GM']
sy_in=5
```

```
In [227]: # get earnings announcements date
api = pl.EODH()
ev_data=api.get_earnings(symbols_list[sy_in], 'US')
ev_data.rename(columns={'date': 'event_date'}, inplace=True)
ev_data['code'] = symbols_list[sy_in]
ev_data.rename(columns={'Code': 'symbol'}, inplace=True)
ev_data.dropna()
```

```
Out[227]:
```

	actual	symbol	event_date	difference	estimate	percent	report_date
2	1.28	GM	2016-12-31	0.11	1.17	9.40	2017-02-07
3	1.70	GM	2017-03-31	0.22	1.48	14.86	2017-04-28
4	1.89	GM	2017-06-30	0.20	1.69	11.83	2017-07-25
5	1.32	GM	2017-09-30	0.20	1.12	17.86	2017-10-24
6	1.65	GM	2017-12-31	0.27	1.38	19.57	2018-02-06
7	1.43	GM	2018-03-31	0.19	1.24	15.32	2018-04-26
8	1.81	GM	2018-06-30	0.03	1.78	1.69	2018-07-25
9	1.87	GM	2018-09-30	0.82	1.05	49.60	2018-11-01
10	1.43	GM	2018-12-31	0.21	1.22	17.21	2019-02-06
11	1.41	GM	2019-03-31	0.30	1.11	27.03	2019-05-02
12	1.64	GM	2019-06-30	0.21	1.43	14.69	2019-08-01

```
In [228]: # get stock prices for the selected stock/benchmark and timeframe
# using DAILY and Adjusted Close
def getTickerDataEOD(ticker, exchange, stdate, enddate):
    api = pl.EODH()
    return api.get_eod_data_with_date(ticker, exchange, stdate, enddate)

# Start and End dates
dt_start = dt.datetime(2016, 1, 1)
dt_end = dt.datetime(2019, 1, 1)

data=getTickerDataEOD(symbols_list[sy_in], 'US', dt_start, dt_end)['Adjusted_Close'].rename(symbols_list[sy_in])
dataB=getTickerDataEOD('GSPC', 'INDX', dt_start, dt_end)['Adjusted_Close'].rename('GSPC')
data=data.dropna()
data.head()
```

```
Out[228]:
```

Date		
2016-01-04	28.3977	
2016-01-05	27.6474	
2016-01-06	26.6585	
2016-01-07	25.5672	
2016-01-08	25.1751	

Name: GM, dtype: float64

```
In [229]: # functions to extract close prices
def get_close_price(data, current_date, day_number):
    # If we're looking at day 0 just return the indexed date
    if day_number == 0:
        return data.loc[current_date]
    # Find the close price day_number away from the current date
    else:
        # If the close price is too far ahead, just get the last available
        total_date_index_length = len(data.index)
        # Find the closest date to the target date
        date_index = data.index.searchsorted(current_date + timedelta(days=day_number))
        # If the closest date is too far ahead, reset to the latest date possible
        date_index = total_date_index_length - 1 if date_index >= total_date_index_length else date_index
        # Use the index to return a close price that matches
        return data.iloc[date_index]

def get_first_price(data, starting_point, date):
    starting_day = date - timedelta(days=starting_point)
    date_index = data.index.searchsorted(starting_day)
    return data.iloc[date_index]

def remove_outliers(returns, num_std_devs):
    return returns[~((returns-returns.mean()).abs()>num_std_devs*returns.std())]

def get_returns(data, starting_point, date, day_num):
    # Get stock prices
    first_price = get_first_price(data, starting_point, date)
    close_price = get_close_price(data, date, day_num)

    # Calculate returns
    ret = (close_price - first_price)/(first_price + 0.0)
    return ret
```

```
In [230]: #Calculate average cumulative returns
# Dictionaries that I'm going to be storing calculated data in
all_returns = {}
all_std_devs = {}
total_sample_size = {}

# Create our range of day_numbers that will be used to calculate returns
starting_point = 30
# Looking from -starting_point till +starting_point which creates our timeframe band
day_numbers = [i for i in range(-starting_point, starting_point)]

for day_num in day_numbers:
    # Reset our returns and sample size each iteration
    returns = []
    sample_size = 0

    # Get the return compared to t=0
    for date, row in ev_data.iterrows():
        sid = row.symbol
        date = row.event_date

        # Make sure that data exists for the dates
        if date not in data.index:
            continue

        returns.append(get_returns(data, starting_point, date, day_num))
        sample_size += 1

    # Drop any Nans, remove outliers, find outliers and aggregate returns and std dev
    returns = pd.Series(returns).dropna()
    all_returns[day_num] = np.average(returns)
    all_std_devs[day_num] = np.std(returns)
    total_sample_size[day_num] = sample_size

# Take all the returns, stds, and sample sizes that I got and put that into a Series
all_returns = pd.Series(all_returns)
all_std_devs = pd.Series(all_std_devs)
N = np.average(pd.Series(total_sample_size))
```

```
In [231]: #Plotting event study graph
xticks = [d for d in day_numbers if d%2 == 0]
all_returns.plot(xticks=xticks, label="N=%s" % N)

pyplot.grid(b=None, which='major', axis='y')
pyplot.title(symbols_list[sy_in] + " Cumulative Return from Announcements before and after event")
pyplot.xlabel("Window Length (t)")
pyplot.ylabel("Cumulative Return (r)")
pyplot.legend()
```



```
In [232]: #Comparing with the benchmark's cumulative returns
all_returns = {}
benchmark_returns = {}

# Create our range of day_numbers that will be used to calculate returns
starting_point = 30
day_numbers = [i for i in range(-starting_point, starting_point)]

for day_num in day_numbers:
    # Reset our returns and sample size each iteration
    returns = []
    b_returns = []
    sample_size = 0

    # Get the return compared to t=0
    for date, row in ev_data.iterrows():
        sid = 0
        date = row.event_date

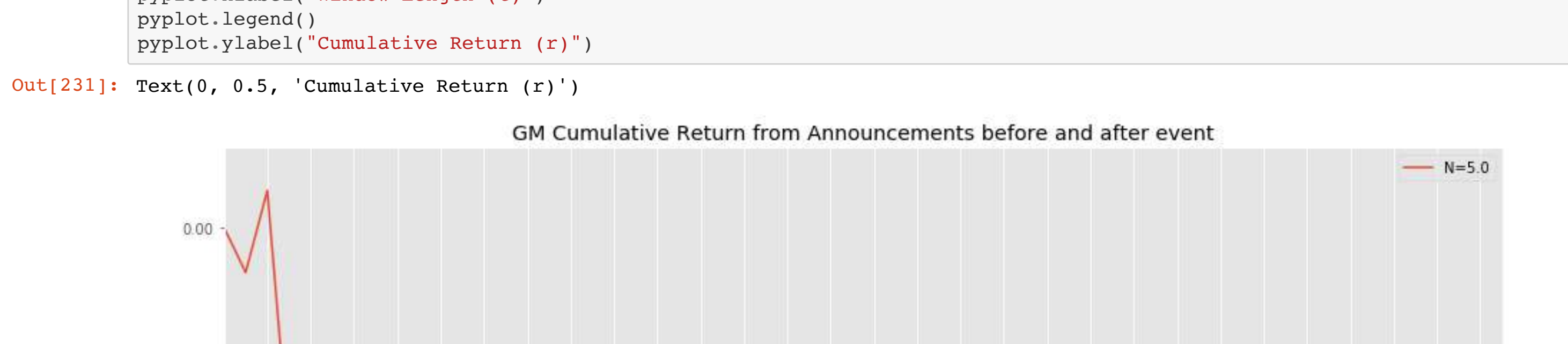
        # Make sure that data exists for the dates
        if date not in data.index:
            continue

        returns.append(get_returns(data, starting_point, date, day_num))
        # 8554 is the sid for the benchmark
        b_returns.append(get_returns(dataB, starting_point, date, day_num))

    # Drop any Nans, remove outliers, find outliers and aggregate returns and std dev
    all_returns[day_num] = np.average(remove_outliers(pd.Series(returns).dropna(), 2))
    benchmark_returns[day_num] = np.average(pd.Series(b_returns).dropna())

# Plot
xticks = [d for d in day_numbers if d%2 == 0]
all_returns = pd.Series(all_returns)
all_returns.plot(xticks=xticks, label=symbols_list[sy_in])
benchmark_returns = pd.Series(benchmark_returns)
benchmark_returns.plot(xticks=xticks, label="benchmark")

pyplot.title("Comparing the benchmark's average returns around that time to " + symbols_list[sy_in])
pyplot.ylabel("% Cumulative Return")
pyplot.xlabel("Time Window")
pyplot.legend()
pyplot.grid(b=None, which='major', axis='y')
```



```
In [233]: # Plotting strictly the abnormal returns using a rolling 30 day beta
def calc_beta(date, starting_point):
    # Calculate beta by getting the last X days of data
    # 1. Create a DataFrame containing the data for the necessary days within that time frame
    # 2. Pass that DataFrame into our calc_beta function in order to spit out a beta
    # ...

    history_index = data.index.searchsorted(date)
    history_index_start = max((history_index - starting_point, 0))
    price_historystock = data.iloc[history_index_start:history_index]

    history_index = dataB.index.searchsorted(date)
    history_index_start = max((history_index - starting_point, 0))
    price_historybench = dataB.iloc[history_index_start:history_index]

    stock_prices = price_historystock.pct_change().dropna()
    bench_prices = price_historybench.pct_change().dropna()
    aligned_prices = bench_prices.align(stock_prices, join='inner')
    bench_prices = aligned_prices[0]
    stock_prices = aligned_prices[1]
    bench_prices = np.array(bench_prices.values)
    stock_prices = np.array(stock_prices.values)
    bench_prices = np.reshape(bench_prices, len(bench_prices))
    stock_prices = np.reshape(stock_prices, len(stock_prices))
    if len(stock_prices) == 0:
        return None
    m, b = np.polyfit(bench_prices, stock_prices, 1)
    return m

# Create our range of day_numbers that will be used to calculate returns
ab_all_returns = {}
ab_volatility = {}

starting_point = 30
day_numbers = [i for i in range(-starting_point, starting_point)]

for day_num in day_numbers:
    # Reset our returns and sample size each iteration
    returns = []
    b_returns = []
    sample_size = 0

    # Get the return compared to t=0
    for date, row in ev_data.iterrows():
        sid = 0
        date = row.event_date

        # Make sure that data exists for the dates
        if date not in data.index:
            continue

        ret = get_returns(data, starting_point, date, day_num)
        b_ret = get_returns(dataB, starting_point, date, day_num)

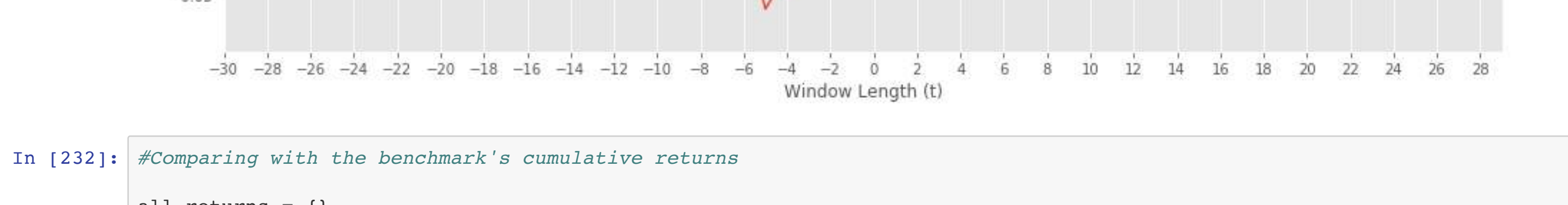
        # Calculate beta
        beta = calc_beta(date, starting_point)
        if beta is None:
            continue

        # Calculate abnormal returns
        abnormal_return = ret - (beta*b_ret)
        returns.append(abnormal_return)

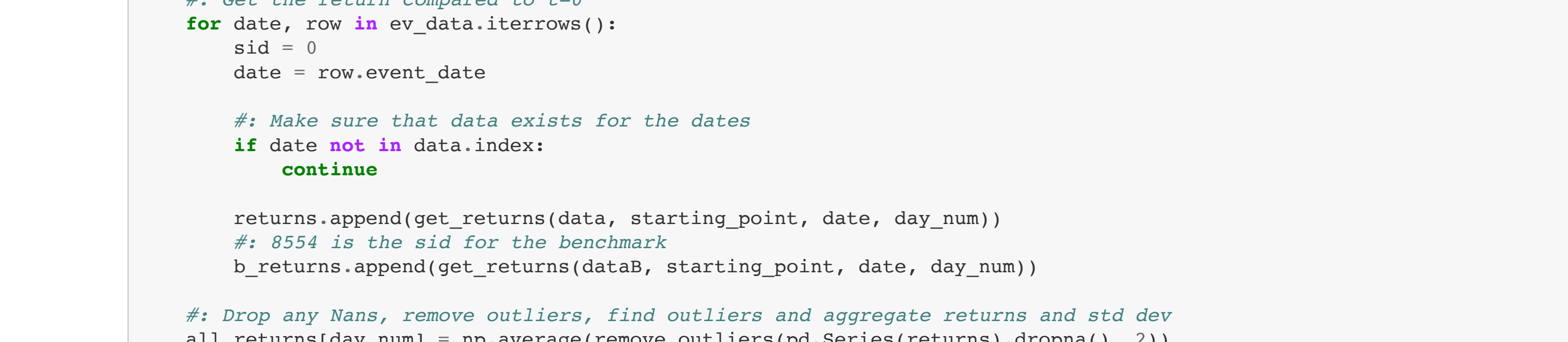
    # Drop any Nans, remove outliers, find outliers and aggregate returns and std dev
    returns = pd.Series(returns).dropna()
    ab_volatility[day_num] = np.std(returns)
    ab_all_returns[day_num] = np.average(returns)
```

```
In [234]: #Plotting cumulative abnormal returns
xticks = [d for d in day_numbers if d%2 == 0]
ab_all_returns = pd.Series(ab_all_returns)
ab_all_returns.plot(xticks=xticks, label="Abnormal Average Cumulative")
all_returns.plot(xticks=xticks, label="Simple Average Cumulative")

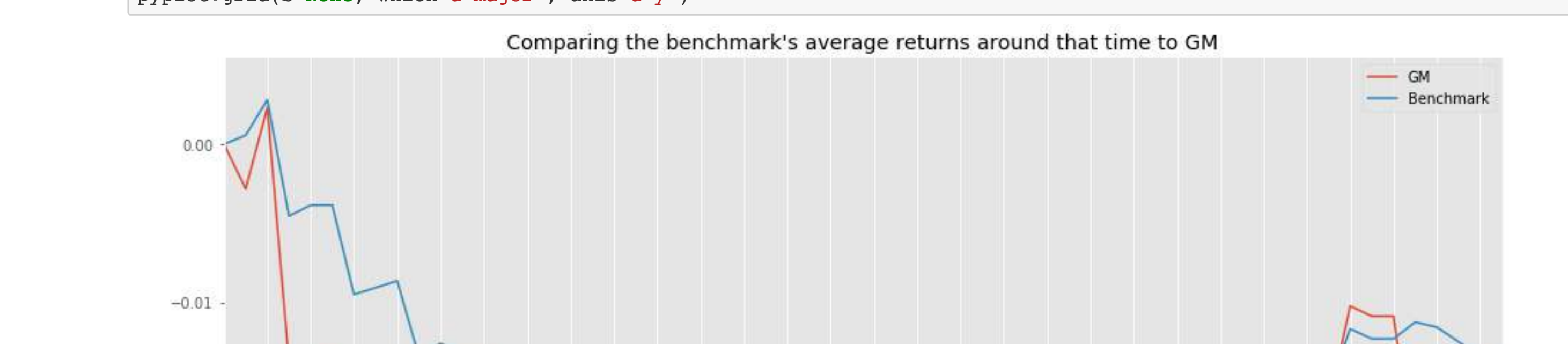
pyplot.axhline(y=ab_all_returns.loc[0], linestyle='--', color='black', alpha=.3, label='Drift')
pyplot.errorbar(ab_all_returns.index, all_returns, xerr=0, yerr=all_std_devs, label="N=%s" % N)
pyplot.title(symbols_list[sy_in] + " Cumulative Return from Announcements before and after event with error")
pyplot.xlabel("Window Length (t)")
pyplot.ylabel("Cumulative Return (r)")
pyplot.grid(b=None, which='major', axis='y')
pyplot.legend()
```



```
In [235]: #Plotting the same graph but with error bars
all_std_devs.loc[-1] = 0
pyplot.errorbar(ab_all_returns.index, all_returns, xerr=0, yerr=all_std_devs, label="N=%s" % N)
pyplot.grid(b=None, which='major', axis='y')
pyplot.title(symbols_list[sy_in] + " Cumulative Return from Announcements before and after event with error")
pyplot.xlabel("Window Length (t)")
pyplot.ylabel("Cumulative Return (r)")
pyplot.legend()
pyplot.show()
```



```
In [236]: # Plotting volatility of abnormal returns
ab_volatility = pd.Series(ab_volatility)
ab_all_returns = pd.Series(ab_all_returns)
ab_volatility.loc[-1] = 0
pyplot.errorbar(ab_all_returns.index, ab_all_returns, xerr=0, yerr=ab_volatility, label="N=%s" % N)
pyplot.grid(b=None, which='major', axis='y')
pyplot.title(symbols_list[sy_in] + " Cumulative Abnormal Returns from Announcements before and after event with error")
pyplot.xlabel("Window Length (t)")
pyplot.ylabel("Cumulative Return (r)")
pyplot.legend()
pyplot.show()
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```