In [201]:
```python
import seaborn as sns
import matplotlib.pyplot as pyplot
import numpy as np
import pandas as pd
import datetime as dt
import Plib as pl
import matplotlib
import matplotlib.pyplot as plt
import Plib as pl
from pandas_datareader.data import DataReader
from pandas.tseries.offsets import *
from collections import defaultdict
from collections import OrderedDict
from math import sqrt
from datetime import timedelta


# Define list of stocks to conduct event analysis on.
symbols_list = ['AAPL', 'AET', 'AFL', 'AVP', 'CLX', 'GM']
sy_in=2
```

In [202]:
```python
# 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['event_date'] = pd.to_datetime(ev_data['event_date'])
ev_data.dropna()
```

Out[202]:

	actual	symbol	event_date	difference	estimate	percent	report_date
3	1.67	AFL	2017-03-31	0.05	1.62	3.09	2017-04-27
4	1.83	AFL	2017-06-30	0.17	1.66	10.24	2017-07-27
5	1.70	AFL	2017-09-30	0.07	1.63	4.29	2017-10-25
6	0.82	AFL	2017-12-31	0.04	0.78	5.13	2018-02-01
7	1.05	AFL	2018-03-31	0.08	0.97	8.25	2018-04-25
8	1.07	AFL	2018-06-30	0.08	0.99	8.08	2018-07-26
9	1.03	AFL	2018-09-30	0.04	0.99	4.04	2018-10-24
10	1.02	AFL	2018-12-31	0.08	0.94	8.51	2019-02-01
11	1.12	AFL	2019-03-31	0.06	1.06	5.66	2019-04-25
12	1.13	AFL	2019-06-30	0.06	1.07	5.61	2019-07-25

In [203]:
```python
# get stock prices for the selected stock/benchmark and timeframe
# using DAILY and Adjusted Close prices

def getTickerDataEOD(ticker,exchange,stdate,endate):
    api = pl.EODH()
    return api.get_eod_data_with_date(ticker,exchange,stdate,endate)

# 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[203]:
```
Date
2016-01-04    27.2874
2016-01-05    27.2506
2016-01-06    26.8355
2016-01-07    26.3790
2016-01-08    26.1485
Name: AFL, dtype: float64
```

In [204]:
```python
# 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(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(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 [ ]:

In [205]:
```python
#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()
    returns = remove_outliers(returns, 2)
    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 [206]:
```python
#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=u'major', axis=u'y')
pyplot.title(symbols_list[sy_in] + " Cumulative Return from Announcements before and after event")
pyplot.xlabel("Window Length (t)")
pyplot.legend()
pyplot.ylabel("Cumulative Return (r)")
```

Out[206]: Text(0, 0.5, 'Cumulative Return (r)')

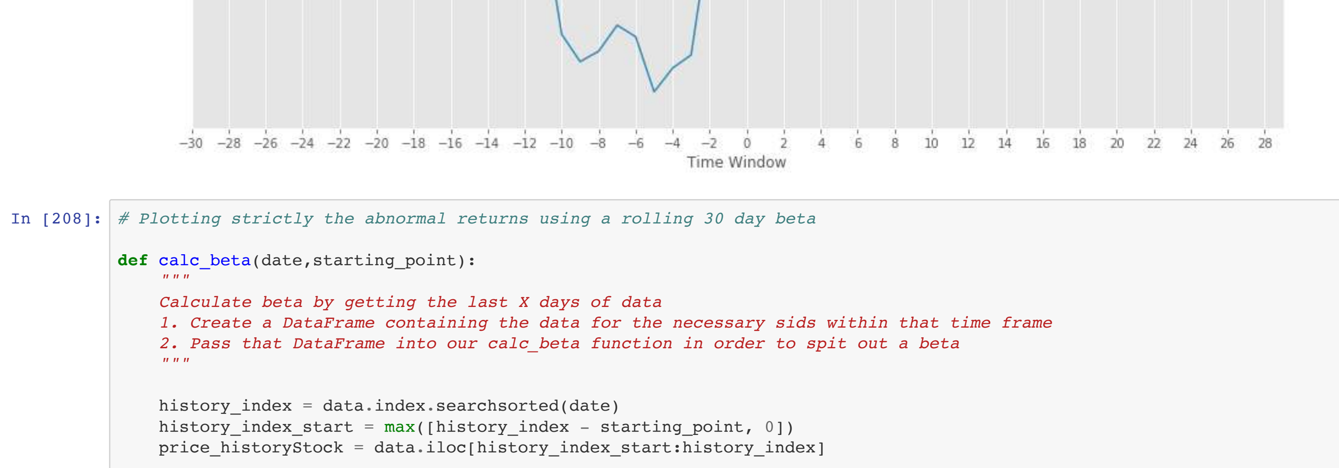
In [207]:
```python
#Comparing with the benchmark's cumulative returns

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))
        #: This 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="AFL")
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=u'major', axis=u'y')
```

In [208]:
```python
# 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 sids 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()
    returns = remove_outliers(returns, 2)

    ab_volatility[day_num] = np.std(returns)
    ab_all_returns[day_num] = np.average(returns)
```

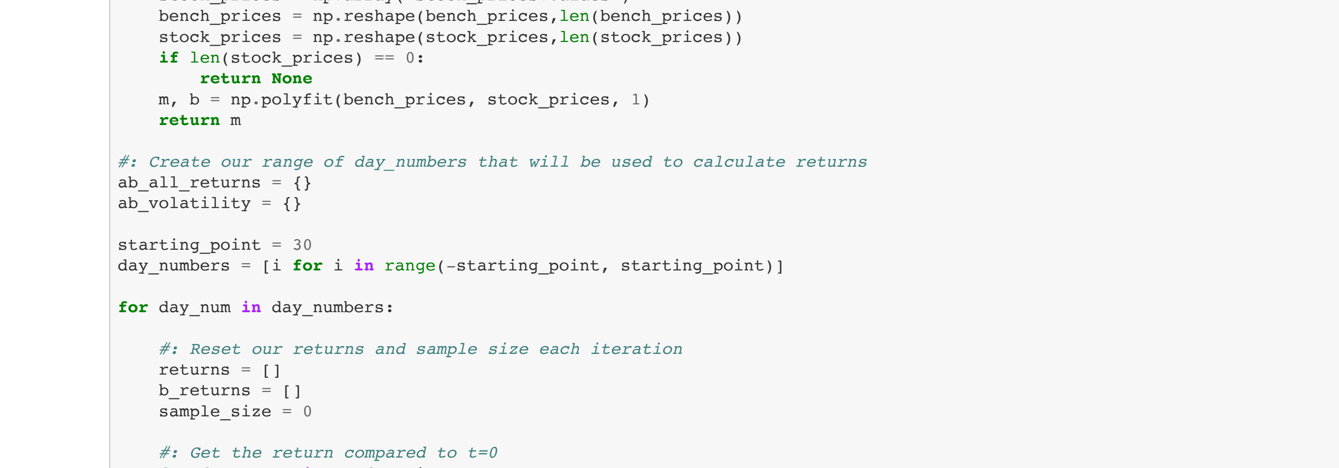
In [209]:
```python
#Plotting cumulative abnormal returns

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.axhline(y=ab_all_returns.max(), linestyle='--', color='black', alpha=.3)
pyplot.title(symbols_list[sy_in] + " Cumulative Abnormal Returns versus Cumulative Returns")
pyplot.ylabel("% Cumulative Return")
pyplot.xlabel("Time Window")
pyplot.grid(b=None, which=u'major', axis=u'y')
pyplot.legend()
```

Out[209]: <matplotlib.legend.Legend at 0x1a1fd927f0>

In [210]:
```python
#Plotting the same graph but with error bars

all_std_devs.loc[:-1] = 0
pyplot.errorbar(all_returns.index, all_returns, xerr=0, yerr=all_std_devs, label="N=%s" % N)
pyplot.grid(b=None, which=u'major', axis=u'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 [211]:
```python
# 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=u'major', axis=u'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 [ ]: