In [36]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

QUERY 1-

2.1 Load the week2.csv file into a dataframe. What is the type of the Date column? Make sure it is of type datetime64. Convert the Date column to the index of the dataframe. Plot the closing price of each of the days for the entire time frame to get an idea of what the general outlook of the stock is. Look out for drastic changes in this stock, you have the exact date when these took place, try to fetch the news for this day of this stock This would be helpful if we are to train our model to take NLP inputs.

In [37]:
```python
data=pd.read_csv('week2.csv')
```

In [38]:
```python
data.tail()
```

Out[38]:

	Symbol	Series	Date	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price	To Trad Quan
489	BAJFINANCE	EQ	07-May-2019	3034.30	3052.9	3069.80	3007.6	3023.0	3017.05	3041.32	970;
490	BAJFINANCE	EQ	08-May-2019	3017.05	3012.0	3017.00	2900.0	2910.0	2921.30	2969.30	1155(
491	BAJFINANCE	EQ	09-May-2019	2921.30	2900.0	2991.80	2885.0	2969.0	2971.35	2951.93	1745;
492	BAJFINANCE	EQ	10-May-2019	2971.35	2970.1	2996.00	2900.0	2922.0	2922.85	2929.29	1630(
493	BAJFINANCE	EQ	13-May-2019	2922.85	2929.9	2957.95	2906.0	2935.0	2931.85	2932.66	1356;

In [39]:
```python
plt.style.use('fivethirtyeight')
```

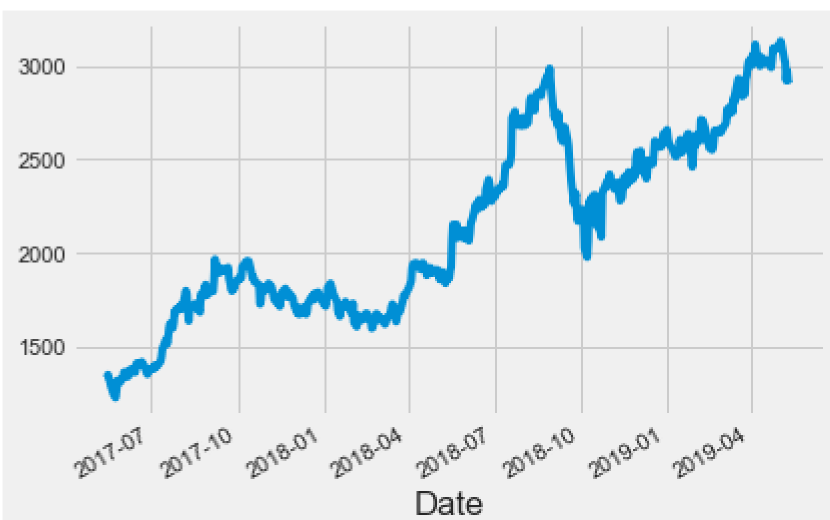
In [40]:
```
data.dtypes
```

Out[40]:
```
Symbol                       object
Series                       object
Date                         object
Prev Close                  float64
Open Price                  float64
High Price                  float64
Low Price                   float64
Last Price                  float64
Close Price                 float64
Average Price               float64
Total Traded Quantity         int64
Turnover                    float64
No. of Trades                 int64
Deliverable Qty               int64
% Dly Qt to Traded Qty      float64
Date_new                     object
month                        object
Day_Perc_Change             float64
Trend                        object
dtype: object
```

In [41]:
```
data['Date'] = pd.to_datetime(data['Date_new'])
data.drop('Date_new',inplace=True,axis=1)
data.index = data.Date
```

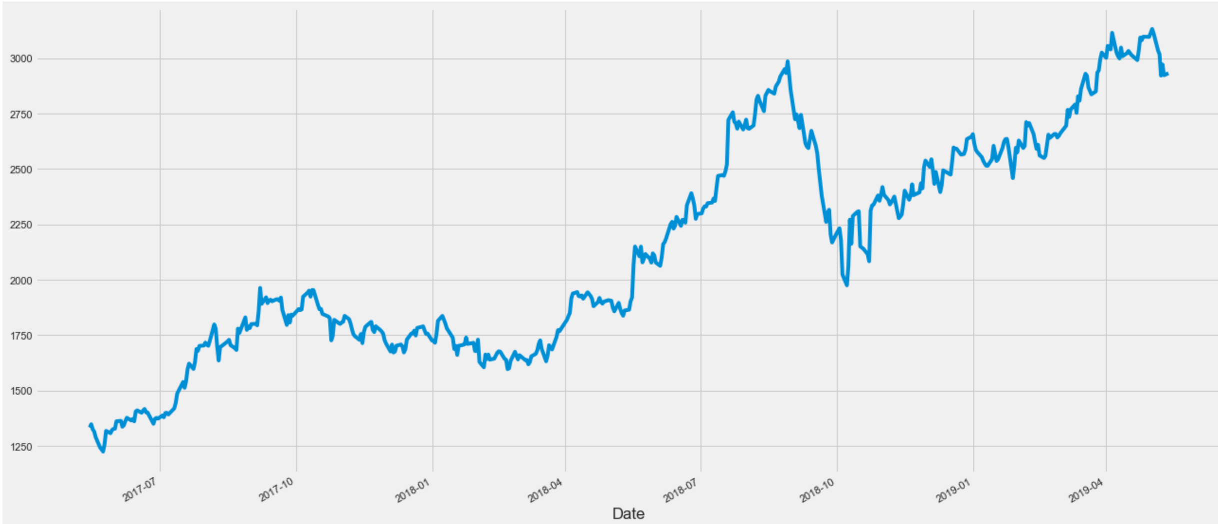
In [42]:
```
data['Close Price'].plot()
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa21dbaef0>



```
In [43]: data['Close Price'].plot(figsize=(20,10))
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa21e56c88>

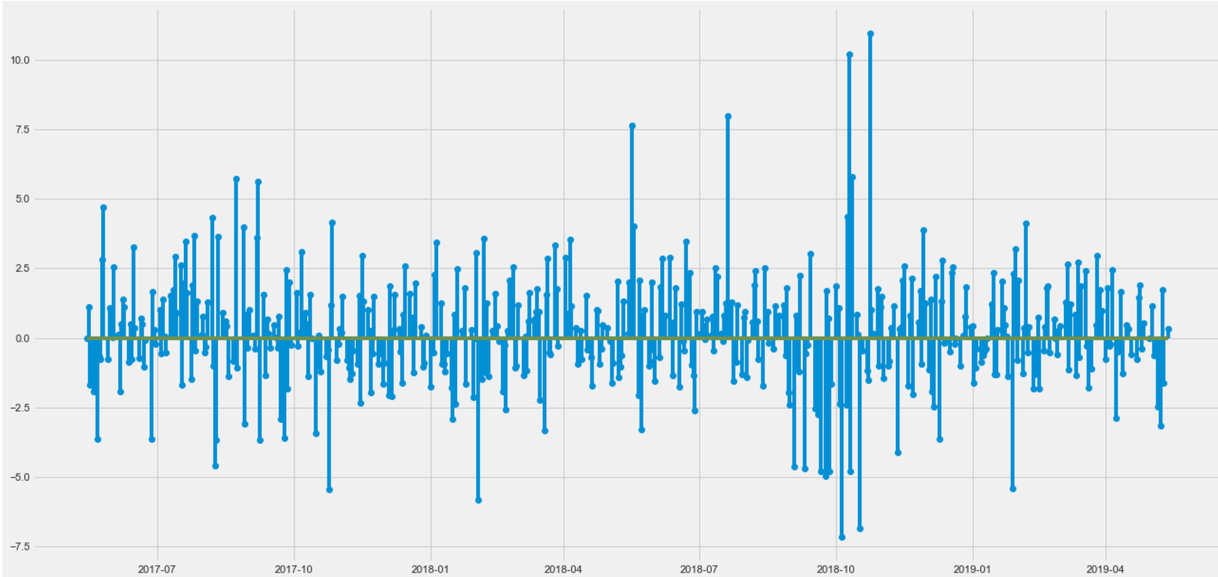


QUERY 2-

2.2 A stem plot is a discrete series plot, ideal for plotting daywise data. It can be plotted using the plt.stem() function. Display a stem plot of the daily change in of the stock price in percentage. This column was calculated in module 1 and should be already available in week2.csv. Observe whenever there's a large change.

In [44]:
```
fig=plt.figure(figsize=(20,10))
plt.stem(data.Date, data['Day_Perc_Change'])
```

Out[44]: <StemContainer object of 3 artists>

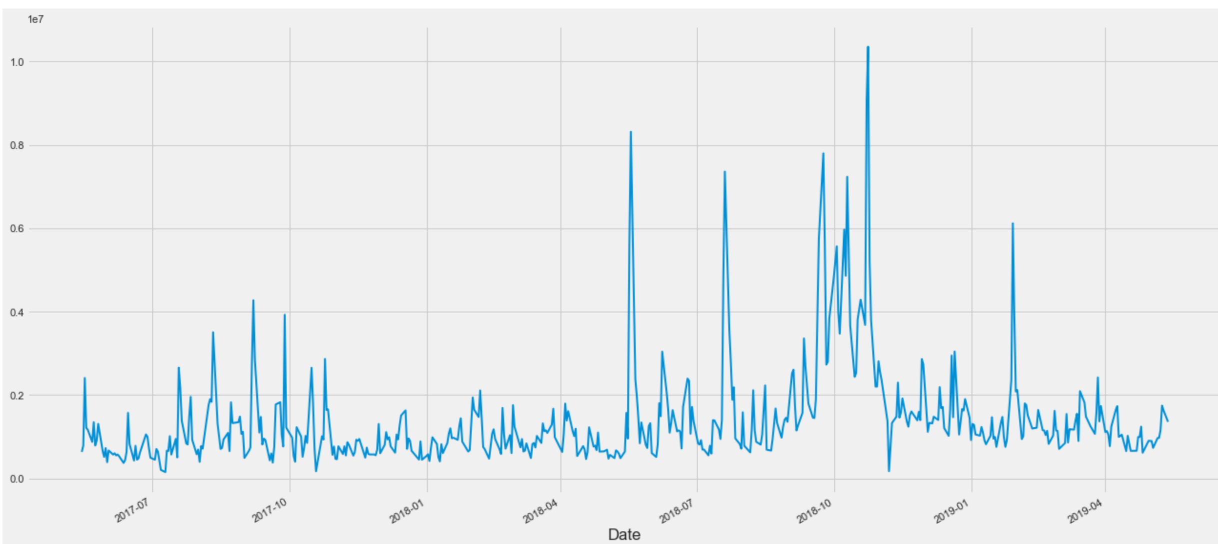


QUERY 3-

2.3 Plot the daily volumes as well and compare the percentage stem plot to it. Document your analysis of the relationship between volume and daily percentage change.

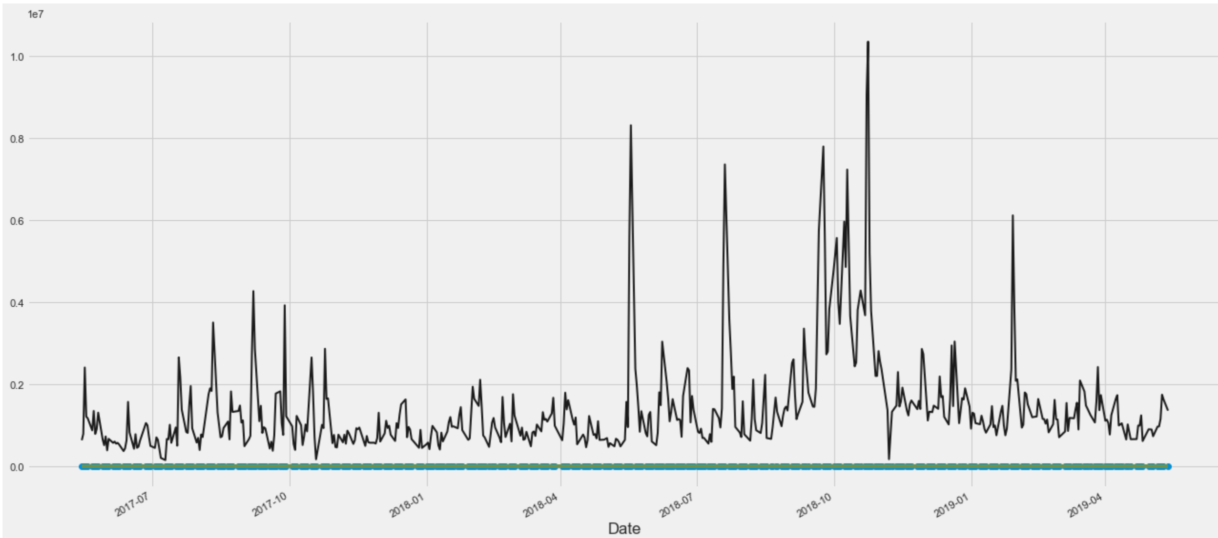
In [45]:
```
data['Total Traded Quantity'].plot(figsize=(20,10),lw=2)
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa24c9a860>



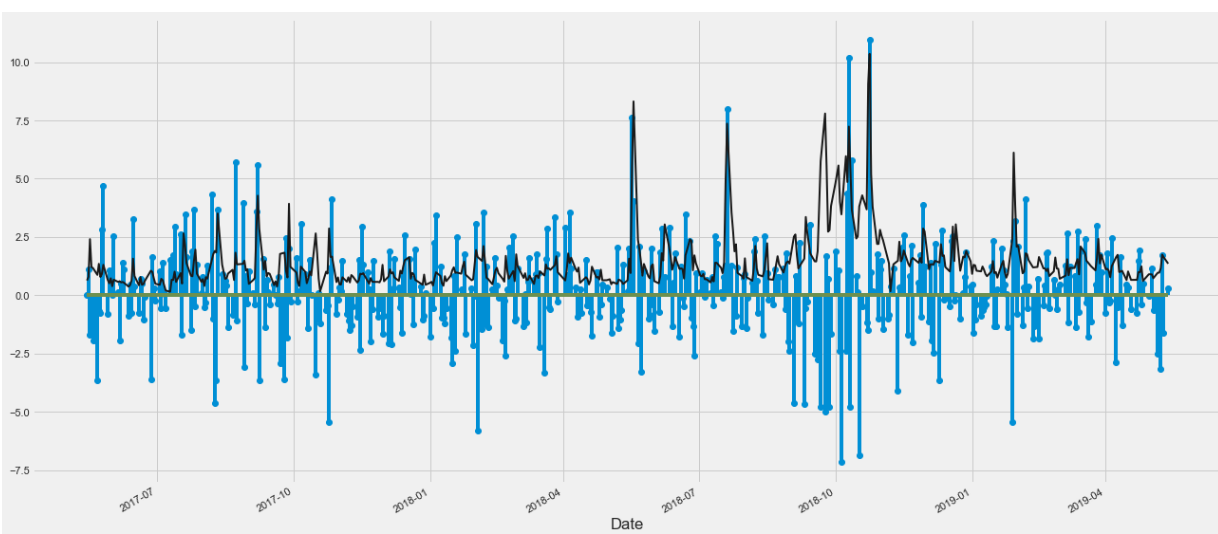
```
In [46]: fig=plt.figure(figsize=(20,10))
         plt.stem(data.Date, data['Day_Perc_Change'])
         (data['Total Traded Quantity']).plot(figsize=(20,10),lw=2, c='k')
```

```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa23ca5128>
```




```
In [47]: fig=plt.figure(figsize=(20,10))
         plt.stem(data.Date, data['Day_Perc_Change'])
         (data['Total Traded Quantity']/1000000).plot(figsize=(20,10),lw=2, c='k')
```

```
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa233808d0>
```

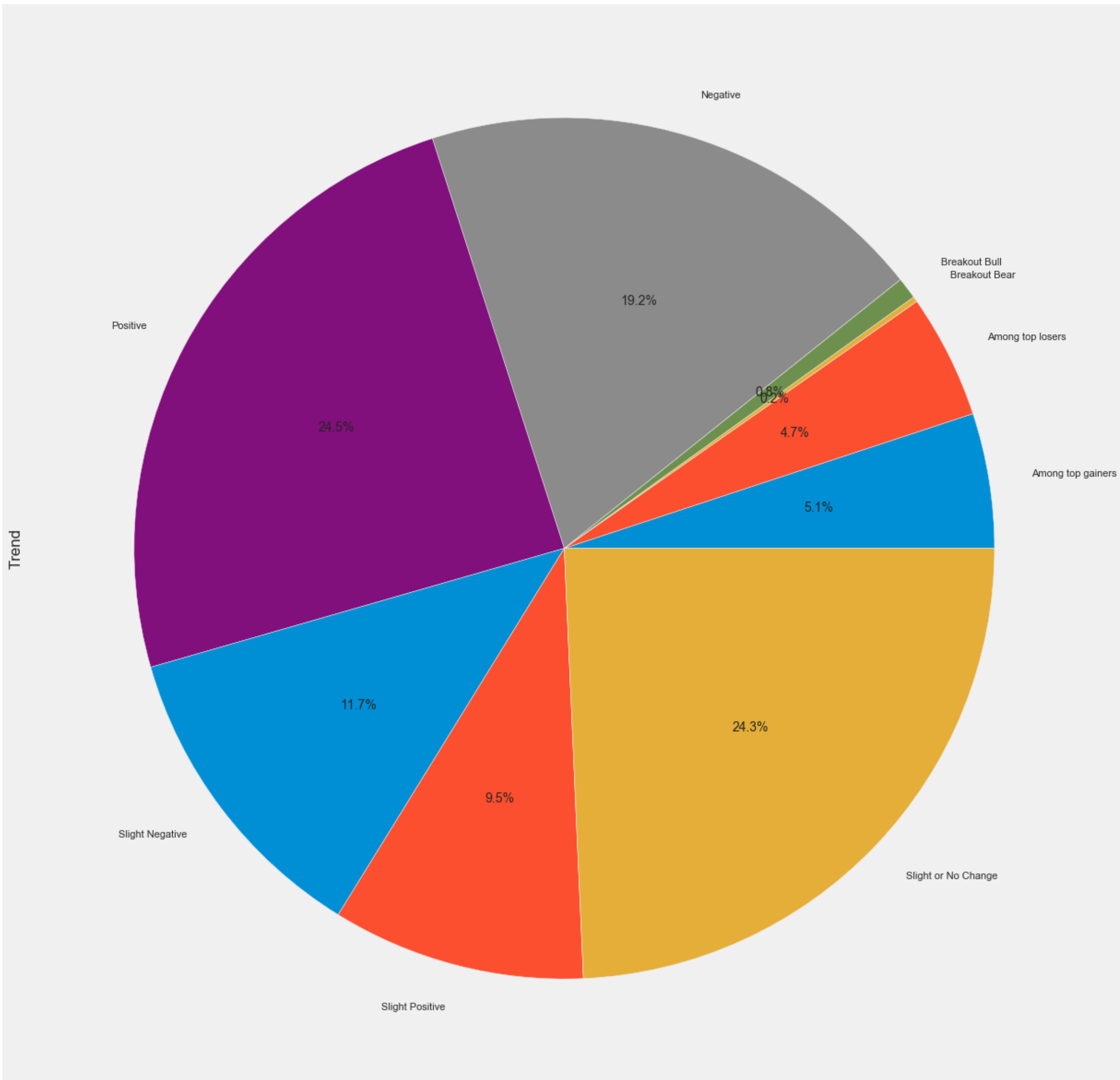


QUERY 4-

2.4 We had created a Trend column in module 1. We want to see how often each Trend type occurs. This can be seen as a pie chart, with each sector representing the percentage of days each trend occurs. Plot a pie chart for all the 'Trend' to know about relative frequency of each trend. You can use the groupby function with the trend column to group all days with the same trend into a single group before plotting the pie chart. From the grouped data, create a BAR plot of average & median values of the 'Total Traded Quantity' by Trend type.

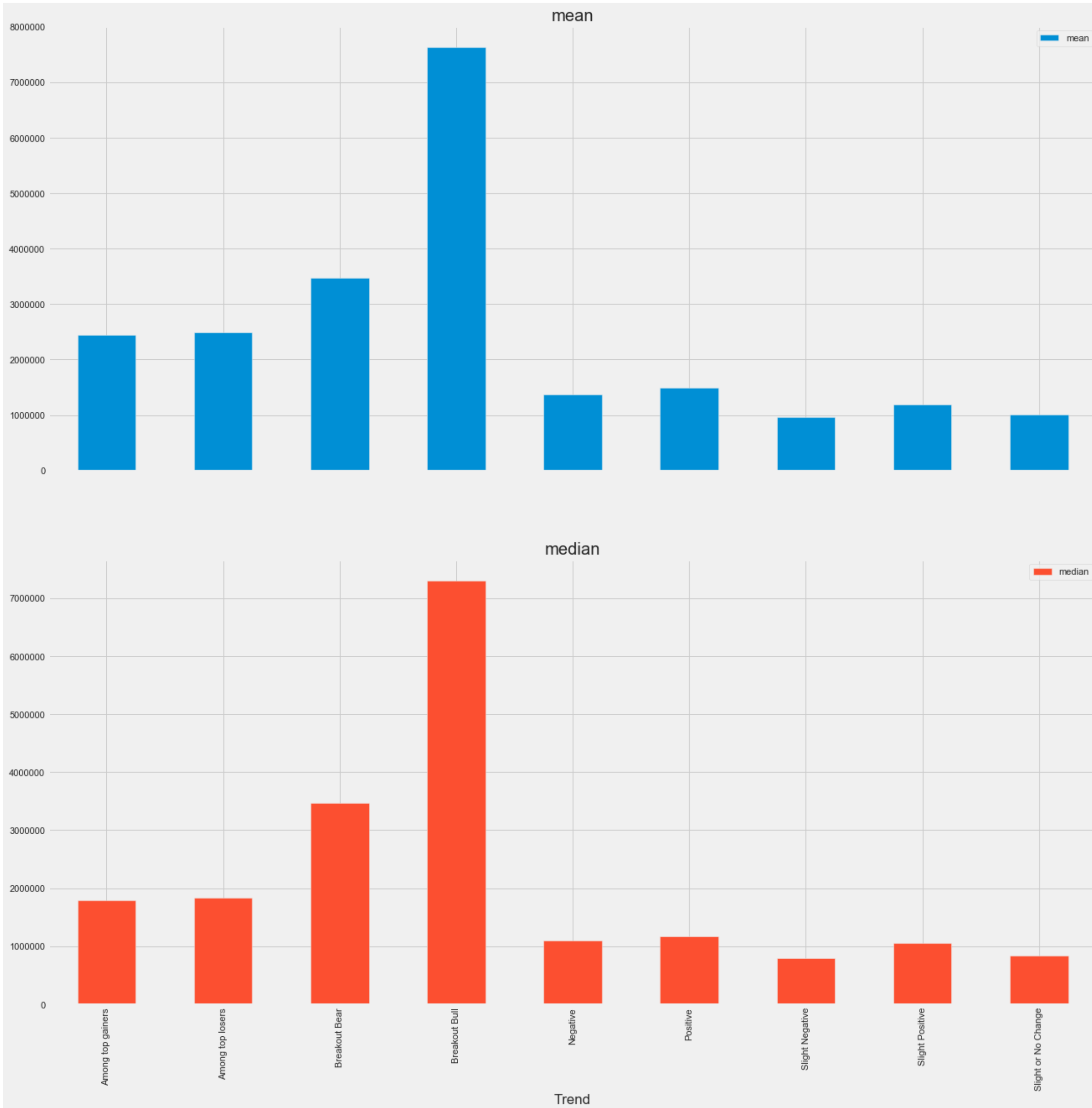
In [48]:
```
pie_data=data.groupby('Trend').Trend.count()
pie_data.plot.pie(subplots=True,figsize=(20,20), autopct='%1.1f%%')
```

Out[48]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001FA233A7240>],
      dtype=object)
```



In [49]:
```python
bar_data=data.groupby('Trend')['Total Traded Quantity'].agg(['mean','median'])
bar_data.plot.bar(subplots=True,figsize=(20,20))
```

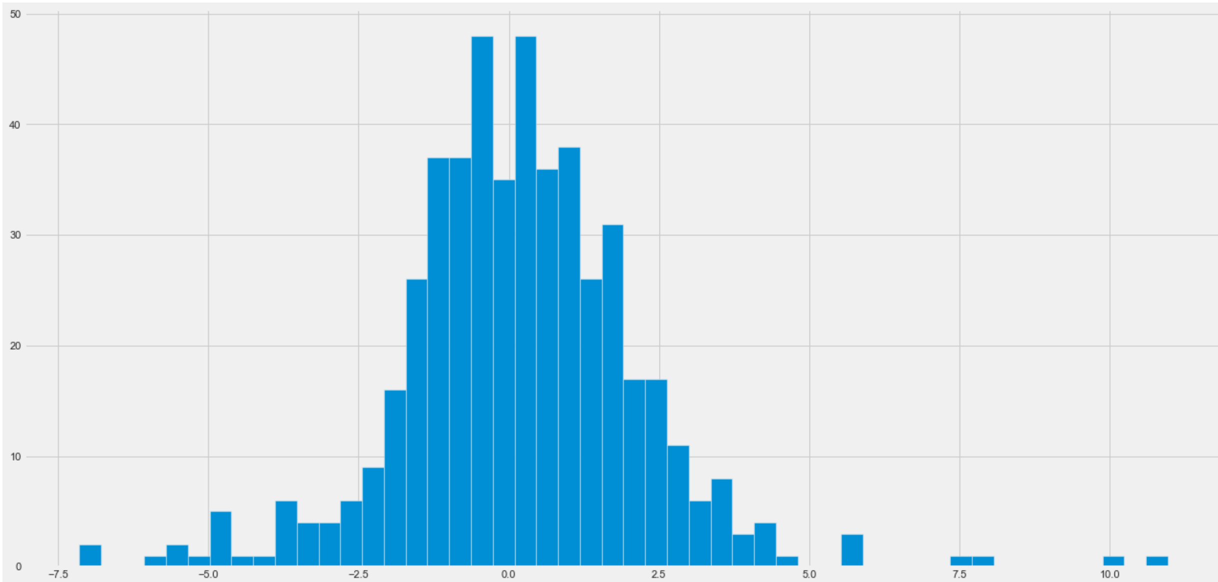
Out[49]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001FA232AE240>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001FA21C8AEF0>],
      dtype=object)
```



QUERY 5-

2.5 Plot the daily return (percentage) distribution as a histogram. Histogram analysis is one of the most fundamental methods of exploratory data analysis. In this case, it'd return a frequency plot of various values of percentage changes .

In [50]:
```
data.Day_Perc_Change.hist(bins=50, figsize=(20,10))
plt.show()
print(data.Day_Perc_Change.describe())
```




```
count    494.000000
mean       0.179926
std        2.020442
min       -7.151871
25%       -0.943483
50%        0.138536
75%        1.259951
max       10.965512
Name: Day_Perc_Change, dtype: float64
```

QUERY 6-

2.6 We next want to analyse how the behaviour of different stocks are correlated. The correlation is performed on the percentage change of the stock price instead of the stock price. Load any 5 stocks of your choice into 5 dataframes. Retain only rows for which 'Series' column has value 'EQ'. Create a single dataframe which contains the 'Closing Price' of each stock. This dataframe should hence have five columns. Rename each column to the name of the stock that is contained in the column. Create a new dataframe which is a percentage change of the values in the previous dataframe. Drop Nan's from this dataframe. Using seaborn, analyse the correlation between the percentage changes in the five stocks. This is extremely useful for a fund manager to design a diversified portfolio. To know more, check out these resources on correlation and diversification.

In [51]:
```
data1=pd.read_csv('CIPLA.csv')
data2=pd.read_csv('SUZLON.csv')
data3=pd.read_csv('MARUTI.csv')
data4=pd.read_csv('BAJFINANCE.csv')
data5=pd.read_csv('JUBLFOOD.csv')
datai=pd.read_csv('Nifty50.csv')
```

In [52]:
```
data1=data1[data1.Series=='EQ']
data1.reset_index(inplace=True, drop=True)
data2=data2[data2.Series=='EQ']
data2.reset_index(inplace=True, drop=True)
data3=data3[data3.Series=='EQ']
data3.reset_index(inplace=True, drop=True)
data4=data4[data4.Series=='EQ']
data4.reset_index(inplace=True, drop=True)
data5=data5[data5.Series=='EQ']
data5.reset_index(inplace=True, drop=True)
```

In [53]:
```
data1=data1[['Close Price']]
data1.columns=['CIPLA']
data2=data2[['Close Price']]
data2.columns=['SUZLON']
data3=data3[['Close Price']]
data3.columns=['MARUTI']
data4=data4[['Close Price']]
data4.columns=['BAJFINANCE']
data5=data5[['Close Price']]
data5.columns=['JUBLFOOD']
datai=datai[['Close']]
datai.columns=['Nifty']
```

In [54]:
```
compare=pd.concat([data1,data2,data3,data4,data5,datai],axis=1)
```

In [55]:
```
compare
```

Out[55]:

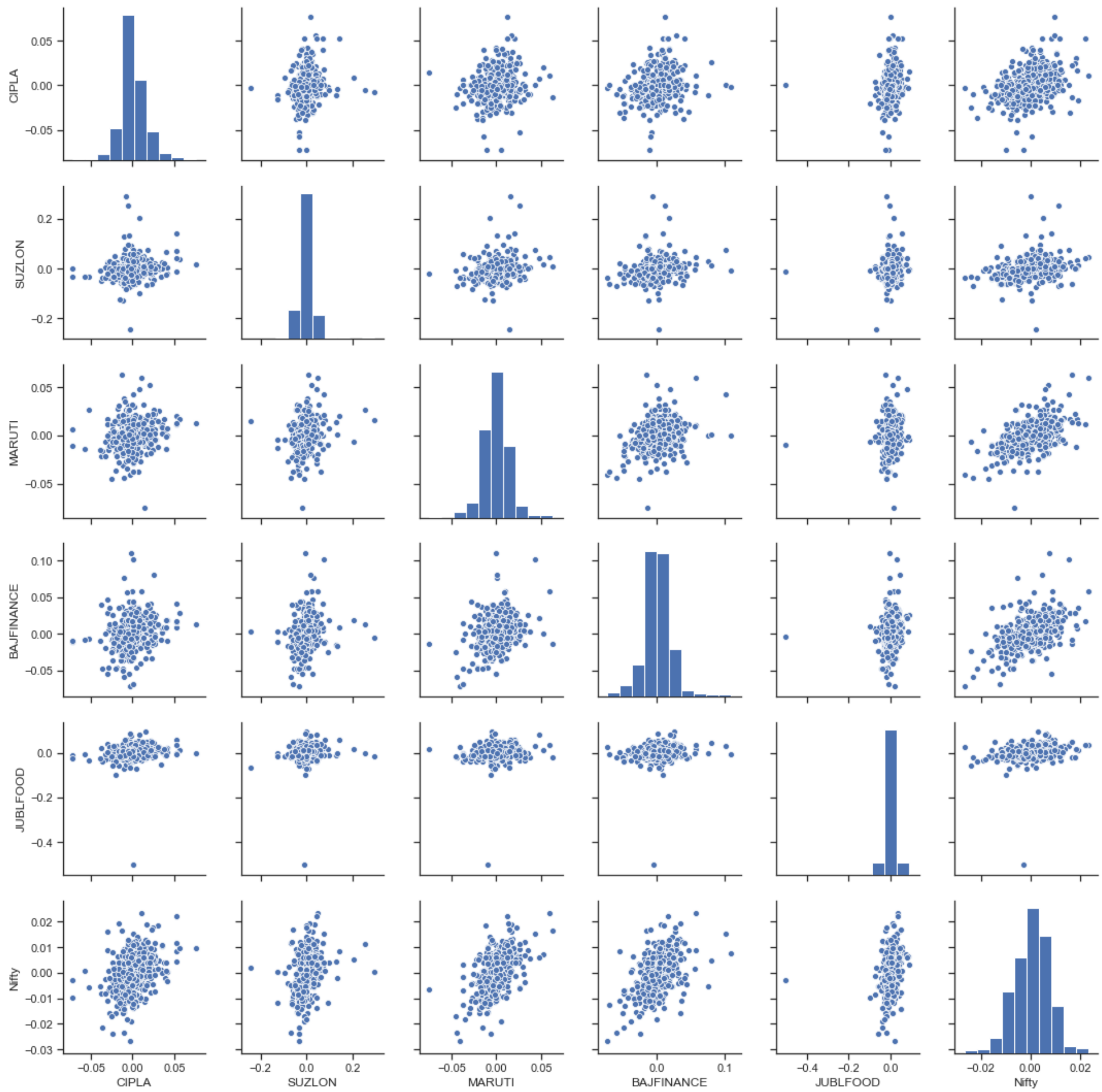
	CIPLA	SUZLON	MARUTI	BAJFINANCE	JUBLFOOD	Nifty
0	569.00	19.60	6823.90	1332.95	1025.45	9445.40
1	565.60	19.70	6953.95	1347.75	1050.65	9512.25
2	562.35	19.90	6958.20	1324.80	1049.05	9525.75
3	560.10	20.00	6831.05	1314.55	1019.35	9429.45
4	564.95	20.60	6790.55	1289.15	1018.10	9427.90
5	563.10	20.40	6701.70	1242.15	1030.30	9438.25
6	533.20	19.75	6878.85	1233.75	993.15	9386.15
7	519.65	18.85	6869.65	1224.35	976.45	9360.55
8	504.00	19.35	6985.70	1258.85	1014.95	9509.75
9	488.90	19.80	7064.80	1317.80	992.70	9595.10
10	505.35	19.15	7134.45	1307.45	940.35	9604.90
11	510.50	19.15	7147.50	1321.55	909.15	9624.55
12	516.35	19.30	7211.00	1326.95	914.90	9621.25
13	515.55	19.50	7146.60	1327.35	912.25	9616.10
14	530.05	19.75	7114.70	1361.25	930.85	9653.50
15	534.80	19.85	7125.70	1363.10	936.05	9675.10
16	534.65	19.35	7112.10	1336.65	952.55	9637.15
17	540.20	19.35	7205.70	1343.15	955.20	9663.90
18	550.15	19.15	7249.50	1361.70	963.80	9647.25
19	551.10	19.20	7464.85	1376.85	969.10	9668.25
20	553.35	19.00	7373.65	1364.95	959.55	9616.40
21	551.55	18.95	7348.95	1371.50	970.05	9606.90
22	539.90	18.75	7351.05	1361.00	962.60	9618.15
23	549.45	18.95	7312.30	1405.25	959.25	9578.05
24	536.75	18.75	7263.90	1410.30	938.20	9588.05
25	539.75	18.55	7249.25	1399.85	924.65	9657.55
26	547.85	18.60	7207.25	1409.60	924.00	9653.50
27	541.00	18.70	7268.20	1416.65	926.20	9633.60
28	540.15	18.70	7316.10	1401.80	913.25	9630.00
29	541.15	18.40	7219.15	1400.75	922.95	9574.95
...	...	...	...	...	...	...
464	525.50	6.65	6518.00	2945.25	1446.60	11445.05
465	525.60	6.50	6596.25	2995.85	1459.40	11570.00
466	528.90	6.15	6672.55	3025.00	1444.00	11623.90

	CIPLA	SUZLON	MARUTI	BAJFINANCE	JUBLFOOD	Nifty
467	525.65	6.20	6840.70	3001.45	1460.30	11669.15
468	522.65	6.80	6889.70	3055.20	1441.20	11713.20
469	520.45	6.50	7072.90	3046.00	1423.15	11643.95
470	521.00	6.70	7113.10	3039.45	1432.85	11598.00
471	532.10	6.70	7107.70	3114.20	1417.60	11665.95
472	525.40	6.65	7129.45	3024.10	1427.70	11604.50
473	532.00	6.70	7216.55	3008.70	1429.55	11671.95
474	546.45	6.70	7186.35	2998.35	1409.65	11584.30
475	544.85	6.70	7187.85	3047.85	1375.30	11596.70
476	554.85	7.20	7342.85	3008.80	1361.50	11643.45
477	566.30	7.35	7352.50	3022.25	1370.40	11690.35
478	559.35	7.35	7458.55	3032.50	1381.15	11787.15
479	561.55	7.10	7447.45	3014.45	1345.15	11752.80
480	561.10	6.80	7321.25	2991.25	1331.05	11594.45
481	561.70	7.20	7048.90	3035.05	1316.10	11575.95
482	558.55	7.25	7016.70	3093.05	1317.00	11726.15
483	555.60	7.15	6905.25	3081.00	1327.90	11641.80
484	567.95	7.20	6842.85	3096.85	1342.80	11754.65
485	565.00	6.85	6666.40	3095.95	1328.50	11748.15
486	565.60	6.75	6683.25	3131.75	1345.90	11724.75
487	564.50	6.75	6710.00	3111.85	1326.35	11712.25
488	563.35	6.60	6709.65	3034.30	1299.85	11598.25
489	557.95	6.35	6702.00	3017.05	1282.25	11497.90
490	558.00	5.95	6650.15	2921.30	1262.45	11359.45
491	557.75	5.65	6624.95	2971.35	1268.80	11301.80
492	555.55	6.40	6631.60	2922.85	1264.50	11278.90
493	546.70	5.60	6543.75	2931.85	1242.60	11148.20

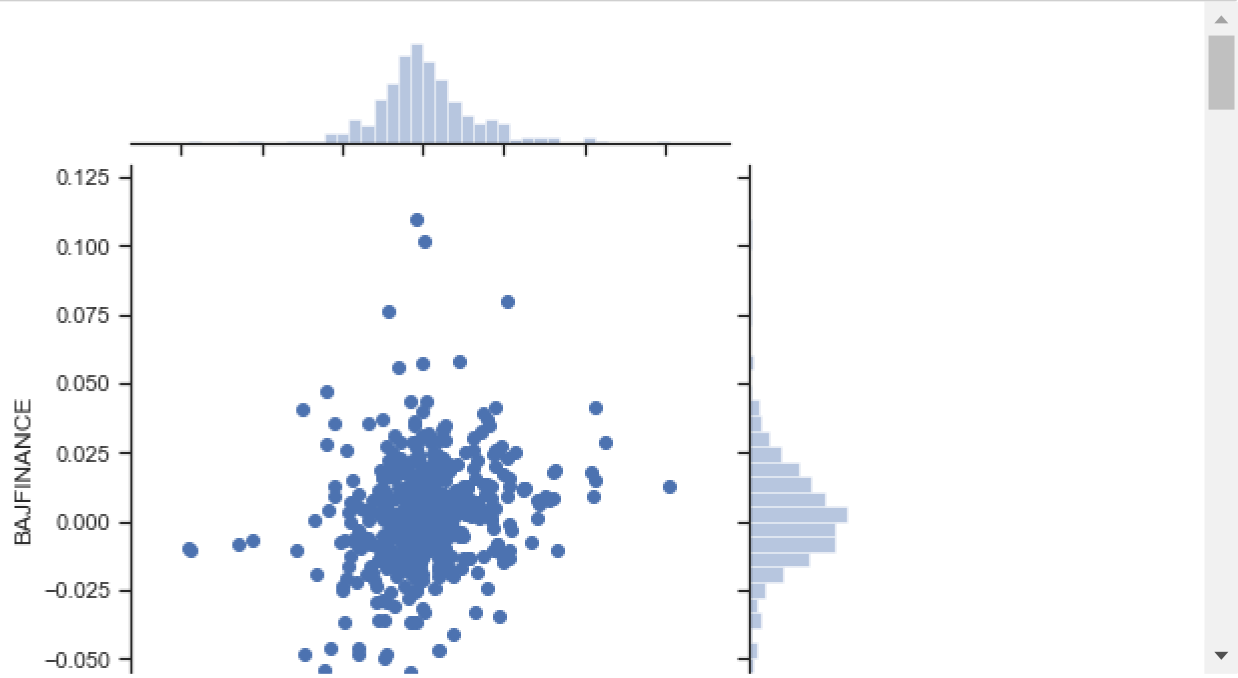
494 rows × 6 columns

In [56]:
```
compare = compare.pct_change()
import seaborn as sns; sns.set(style="ticks", color_codes=True)
compare.replace([np.inf, -np.inf], np.nan)
compare.dropna(inplace=True, how='any', axis=0)
sns.pairplot(compare)
```

Out[56]: <seaborn.axisgrid.PairGrid at 0x1fa23406470>



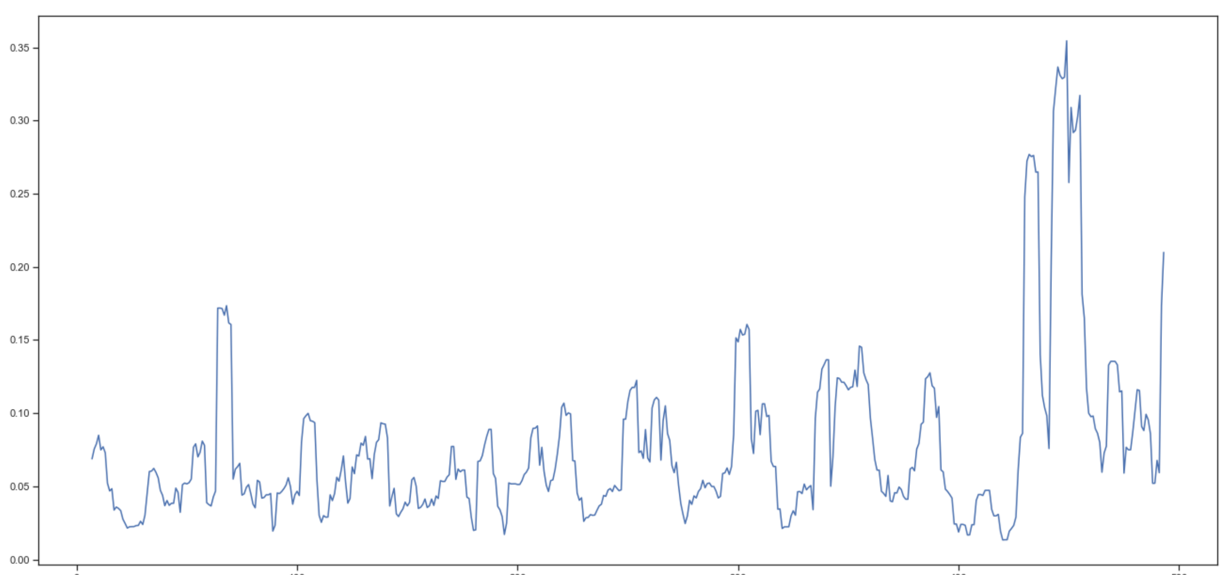
```
In [58]: for i in compare.columns:
             sns.jointplot(i,'BAJFINANCE',compare,kind='scatter')
```



QUERY 7-

2.7 Volatility is the change in variance in the returns of a stock over a specific period of time.Do give the following documentation on volatility a read. You have already calculated the percentage changes in several stock prices. Calculate the 7 day rolling average of the percentage change of any of the stock prices, then compute the standard deviation (which is the square root of the variance) and plot the values. Note: pandas provides a rolling() function for dataframes and a std() function also which you can use.

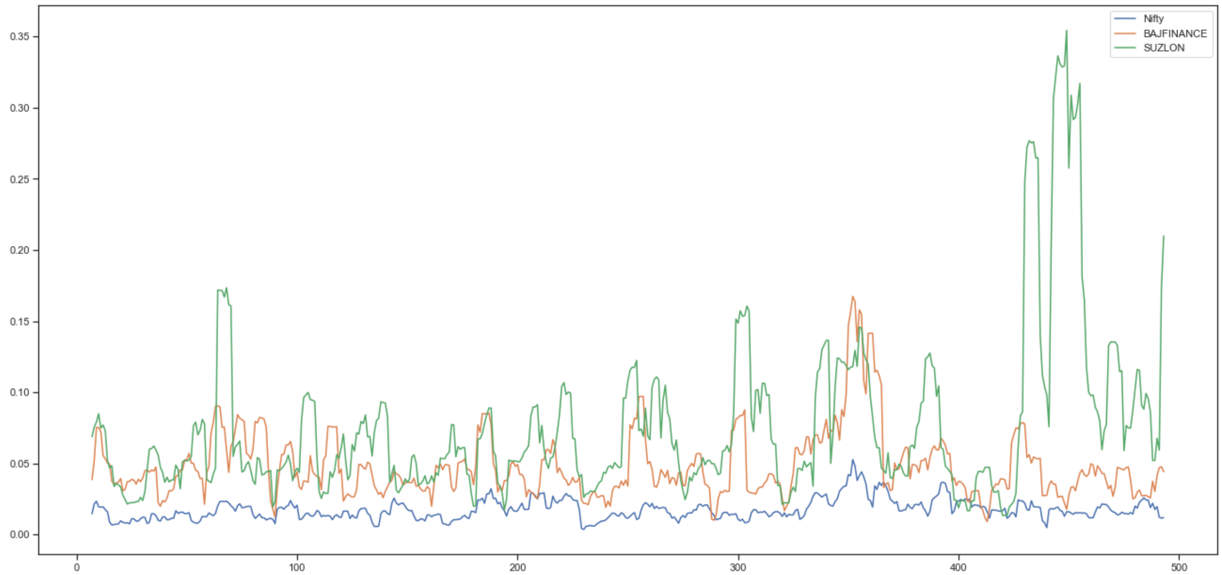
```
In [64]: bajaj=compare['BAJFINANCE'].rolling(7).std()*np.sqrt(7)
         marut.plot(figsize=(20,10))
         plt.show()
```



QUERY 8-

2.8 Calculate the volatility for the Nifty index and compare the 2. This leads us to a useful indicator known as 'Beta' (We'll be covering this in length in Module 3)

In [66]:
```
compare=compare[['Nifty','BAJFINANCE','SUZLON']]
vol=compare.rolling(7).std()*np.sqrt(7)
vol.plot(figsize=(20,10))
plt.show()
```



QUERY 9-

2.9 Trade Calls - Using Simple Moving Averages. Study about moving averages here. Plot the 21 day and 34 day Moving average with the average price and decide a Call ! Call should be buy whenever the smaller moving average (21) crosses over longer moving average (34) AND the call should be sell whenever smaller moving average crosses under longer moving average. One of the most widely used technical indicators.

In [67]:

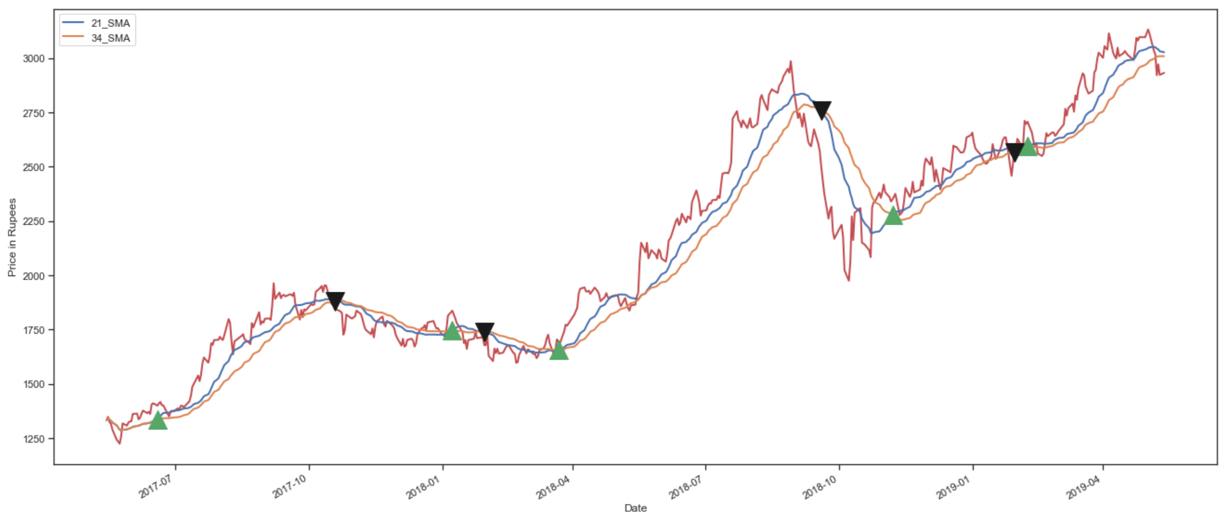
```python
signals = pd.DataFrame(index=data.index)
signals['signal']=0.0
signals['21_SMA'] = data['Close Price'].rolling(window=21,min_periods=1).mean()
signals['34_SMA'] = data['Close Price'].rolling(window=34,min_periods=1).mean()
signals['signal'][21:] = np.where(signals['21_SMA'][21:] > signals['34_SMA'][21:
signals['positions'] = signals['signal'].diff()
print(signals)
```

```
            signal       21_SMA       34_SMA  positions
Date
2017-05-15     0.0  1332.950000  1332.950000        NaN
2017-05-16     0.0  1340.350000  1340.350000        0.0
2017-05-17     0.0  1335.166667  1335.166667        0.0
2017-05-18     0.0  1330.012500  1330.012500        0.0
2017-05-19     0.0  1321.840000  1321.840000        0.0
2017-05-22     0.0  1308.558333  1308.558333        0.0
2017-05-23     0.0  1297.871429  1297.871429        0.0
2017-05-24     0.0  1288.681250  1288.681250        0.0
2017-05-25     0.0  1285.366667  1285.366667        0.0
2017-05-26     0.0  1288.610000  1288.610000        0.0
2017-05-29     0.0  1290.322727  1290.322727        0.0
2017-05-30     0.0  1292.925000  1292.925000        0.0
2017-05-31     0.0  1295.542308  1295.542308        0.0
2017-06-01     0.0  1297.814286  1297.814286        0.0
2017-06-02     0.0  1302.043333  1302.043333        0.0
2017-06-05     0.0  1305.859375  1305.859375        0.0
2017-06-06     0.0  1307.670588  1307.670588        0.0
2017-06-07     0.0  1309.641667  1309.641667        0.0
2017-06-08     0.0  1312.381579  1312.381579        0.0
2017-06-09     0.0  1315.605000  1315.605000        0.0
2017-06-12     0.0  1317.954762  1317.954762        0.0
2017-06-13     0.0  1319.790476  1320.388636        0.0
2017-06-14     0.0  1320.421429  1322.154348        0.0
2017-06-15     0.0  1324.252381  1325.616667        0.0
2017-06-16     0.0  1328.811905  1329.004000        0.0
2017-06-19     1.0  1334.083333  1331.728846        1.0
2017-06-20     1.0  1342.057143  1334.612963        0.0
2017-06-21     1.0  1350.766667  1337.542857        0.0
2017-06-22     1.0  1359.216667  1339.758621        0.0
2017-06-23     1.0  1365.973810  1341.791667        0.0
...            ...          ...          ...        ...
2019-03-27     1.0  2788.495238  2727.169118        0.0
2019-03-28     1.0  2804.595238  2735.539706        0.0
2019-03-29     1.0  2822.083333  2745.198529        0.0
2019-04-01     1.0  2839.200000  2753.854412        0.0
2019-04-02     1.0  2858.542857  2765.560294        0.0
2019-04-03     1.0  2876.888095  2778.036765        0.0
2019-04-04     1.0  2893.283333  2791.239706        0.0
2019-04-05     1.0  2909.830952  2806.092647        0.0
2019-04-08     1.0  2923.607143  2819.710294        0.0
2019-04-09     1.0  2935.054762  2833.217647        0.0
2019-04-10     1.0  2944.926190  2846.116176        0.0
2019-04-11     1.0  2958.964286  2859.107353        0.0
2019-04-12     1.0  2967.566667  2869.522059        0.0
2019-04-15     1.0  2977.764286  2880.752941        0.0
2019-04-16     1.0  2985.942857  2891.775000        0.0
2019-04-18     1.0  2989.985714  2902.266176        0.0
```

```
2019-04-22     1.0  2993.335714  2912.538235        0.0
2019-04-23     1.0  3001.261905  2923.892647        0.0
2019-04-24     1.0  3013.471429  2936.607353        0.0
2019-04-25     1.0  3024.480952  2947.955882        0.0
2019-04-26     1.0  3032.219048  2957.666176        0.0
2019-04-30     1.0  3039.395238  2968.288235        0.0
2019-05-02     1.0  3045.866667  2978.977941        0.0
2019-05-03     1.0  3050.002381  2988.413235        0.0
2019-05-06     1.0  3051.566667  2996.685294        0.0
2019-05-07     1.0  3049.750000  3002.241176        0.0
2019-05-08     1.0  3043.811905  3005.570588        0.0
2019-05-09     1.0  3040.569048  3008.823529        0.0
2019-05-10     1.0  3031.457143  3008.626471        0.0
2019-05-13     1.0  3027.064286  3008.948529        0.0

[494 rows x 4 columns]
```

In [68]:
```
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(111, ylabel='Price in Rupees')
data['Close Price'].plot(ax=ax1, color='r', lw=2.)
signals[['21_SMA', '34_SMA']].plot(ax=ax1, lw=2.)
ax1.plot(signals.loc[signals.positions == 1.0].index,
         signals['21_SMA'][signals.positions == 1.0],
        '^', markersize=20, color='g')
ax1.plot(signals.loc[signals.positions == -1.0].index,
         signals['21_SMA'][signals.positions == -1.0],
        'v', markersize=20, color='k')
plt.show()
```



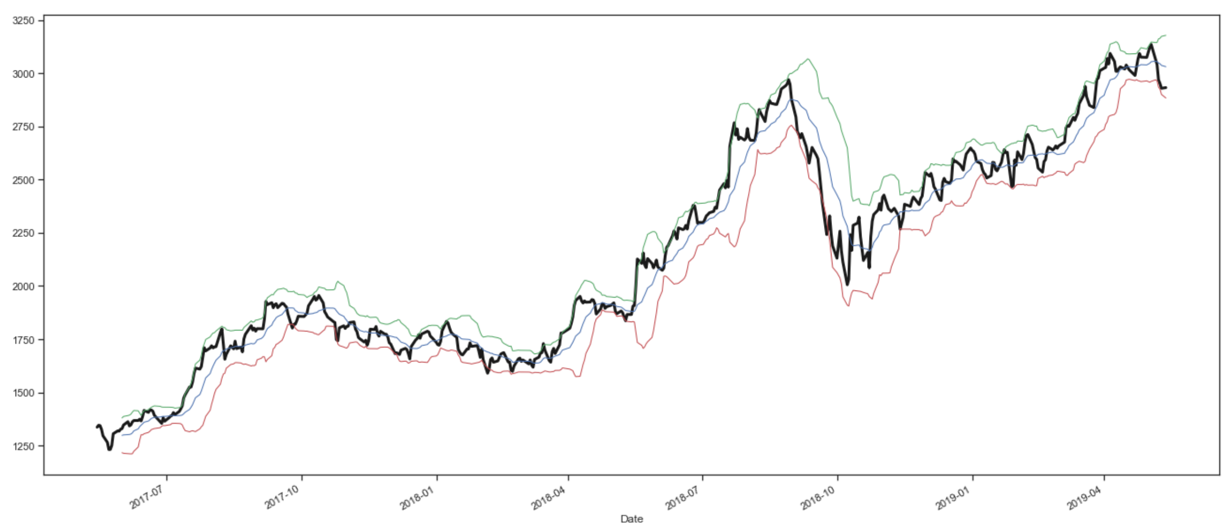
QUERY 10-

2.10 Trade Calls - Using Bollinger Bands Plot the bollinger bands for this stock - the duration of 14 days and 2 standard deviations away from the average The bollinger bands comprise the following data points- The 14 day rolling mean of the closing price (we call it the average) Upper band which is the rolling mean + 2 standard deviations away from the average. Lower band which is the rolling mean - 2 standard deviations away from the average. Average Daily stock price. Bollinger bands are extremely reliable , with a 95% accuracy at 2 standard deviations , and especially useful in sideways moving market. Observe the bands yourself , and analyse the accuracy of all the trade signals provided by the bollinger bands.

In [69]:
```
def bbands(price, length=14, numsd=2):
    ave = price.rolling(length).mean()
    sd = price.rolling(length).std()
    upband = ave + (sd*numsd)
    dnband = ave - (sd*numsd)
    return np.round(ave,3), np.round(upband,3), np.round(dnband,3)
data['ave'], data['upper'], data['lower'] = bbands(data['Close Price'])
```

In [70]:
```
data['Average Price'].plot(c='k',figsize=(20,10),lw=3)
data['ave'].plot(c='b',figsize=(20,10),lw=1)
data['upper'].plot(c='g',figsize=(20,10),lw=1)
data['lower'].plot(c='r',figsize=(20,10),lw=1)
```

Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa2c216780>
```



In [71]:
```
data.tail()
```

Out[71]:

Date	Symbol	Series	Date	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price	...
2019-05-07	BAJFINANCE	EQ	2019-05-07	3034.30	3052.9	3069.80	3007.6	3023.0	3017.05	3041.32	...
2019-05-08	BAJFINANCE	EQ	2019-05-08	3017.05	3012.0	3017.00	2900.0	2910.0	2921.30	2969.30	...
2019-05-09	BAJFINANCE	EQ	2019-05-09	2921.30	2900.0	2991.80	2885.0	2969.0	2971.35	2951.93	...
2019-05-10	BAJFINANCE	EQ	2019-05-10	2971.35	2970.1	2996.00	2900.0	2922.0	2922.85	2929.29	...
2019-05-13	BAJFINANCE	EQ	2019-05-13	2922.85	2929.9	2957.95	2906.0	2935.0	2931.85	2932.66	...

5 rows × 21 columns

Save to a new csv file.

In [72]:
```
data.drop('Date',axis=1,inplace=True)
data.to_csv('week3.csv')
```