Library Used ¶

The following are the libraries required to be installed beforehand which can easily be downloaded with the help of the pip function. A brief description of the Library's name and its application is provided below

```
**Library**                               **Application**

Yahoo Finance                       To download Stock data
Pandas                              To Handel Dataframe in python
Numpy                               Numerical  Python
Matplotlib                          Ploting Graph
```

Here We will take the three companies examples TCS,Infosys and Wipro which are industry leaders providing IT service

In [1]:
```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

```

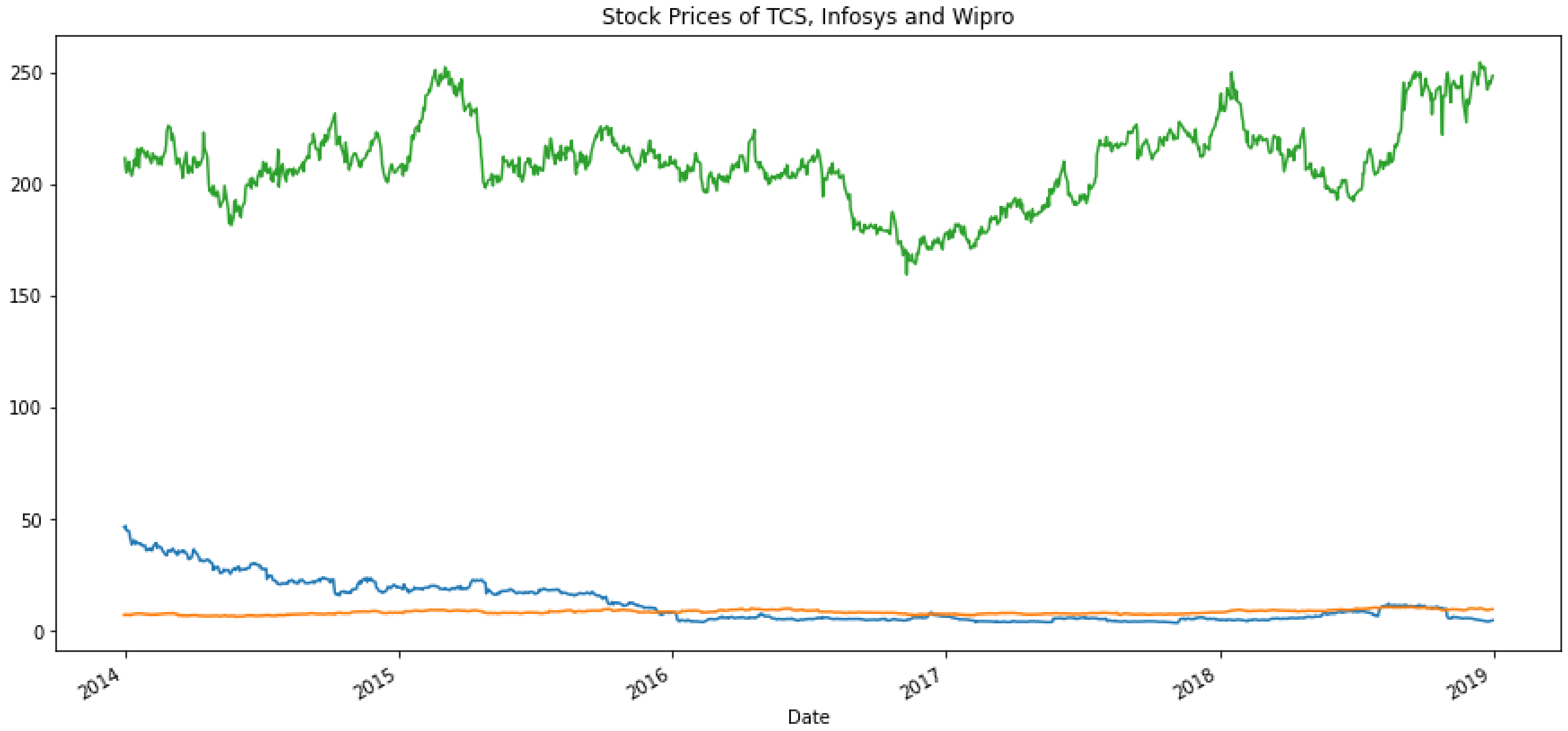
Here, we will take the Example of three companies TCS, Infosys, and Wipro which are the industry leaders in providing IT services.

In [2]:

```
start = "2014-01-01"
end = '2019-1-01'
tcs = yf.download('TCS',start,end)
infy = yf.download('INFY',start,end)
wipro = yf.download('WIPRO.NS',start,end)

tcs['Open'].plot(label = 'TCS', figsize = (15,7))
infy['Open'].plot(label = "Infosys")
wipro['Open'].plot(label = 'Wipro')
plt.title('Stock Prices of TCS, Infosys and Wipro')
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

Out[2]: Text(0.5, 1.0, 'Stock Prices of TCS, Infosys and Wipro')

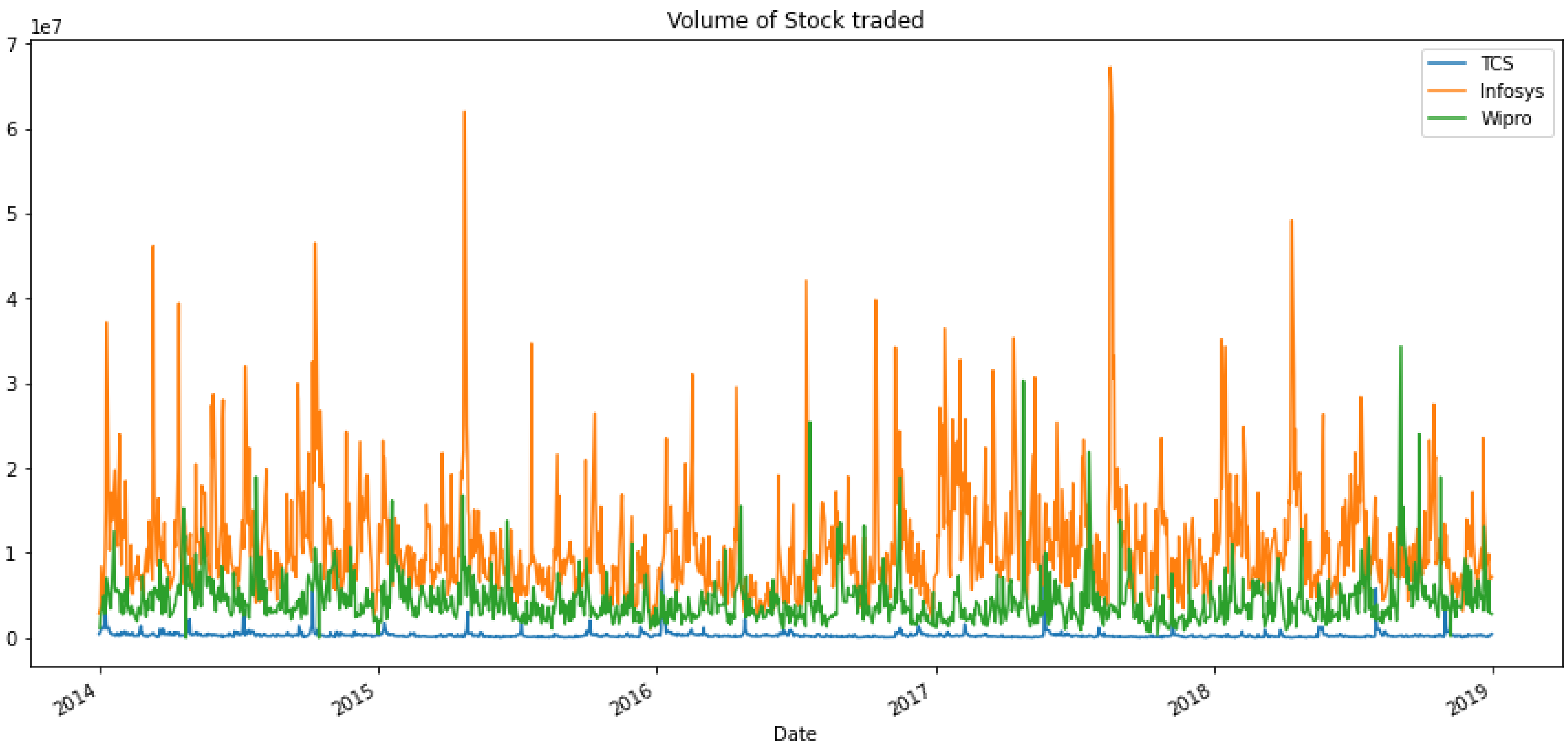


The above graph is the representation of open stock prices for these three companies via line graph by leveraging matplotlib library in python. The Graph clearly shows that the prices of Wipro is more when comparing it to other two companies but we are not interested in the absolute prices for these companies but wanted to understand how these stock fluctuate with time.

In [3]:

```
tcs['Volume'].plot(label = 'TCS', figsize = (15,7))
infy['Volume'].plot(label = "Infosys")
wipro['Volume'].plot(label = 'Wipro')
plt.title('Volume of Stock traded')
plt.legend()
```

Out[3]: <matplotlib.legend.Legend at 0x253e7956490>

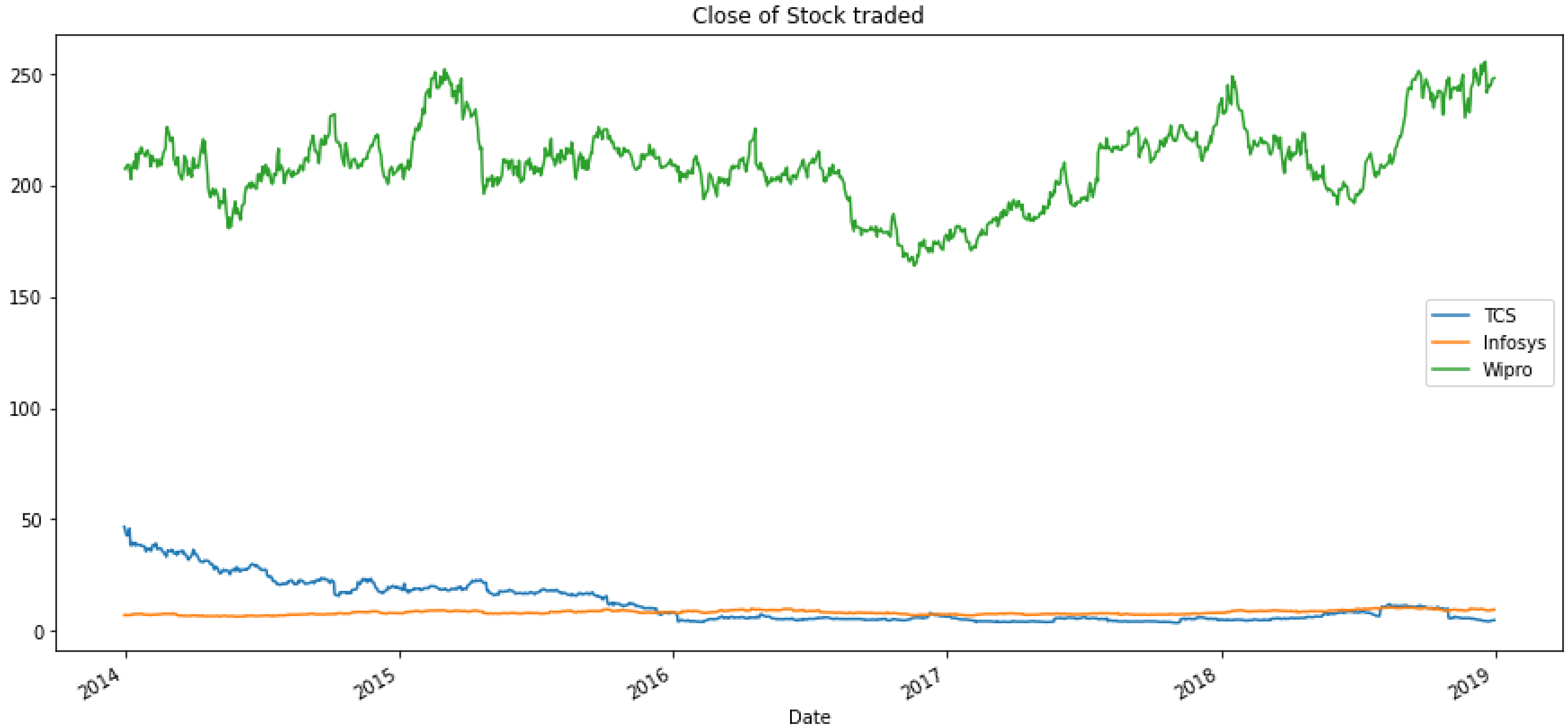


The Graph shows the volume traded by these companies which clearly shows that stocks of Infosys are traded more compared to other IT stocks.

In [4]:

```
tcs['Close'].plot(label = 'TCS', figsize = (15,7))
infy['Close'].plot(label = "Infosys")
wipro['Close'].plot(label = 'Wipro')
plt.title('Close of Stock traded')
plt.legend()
```

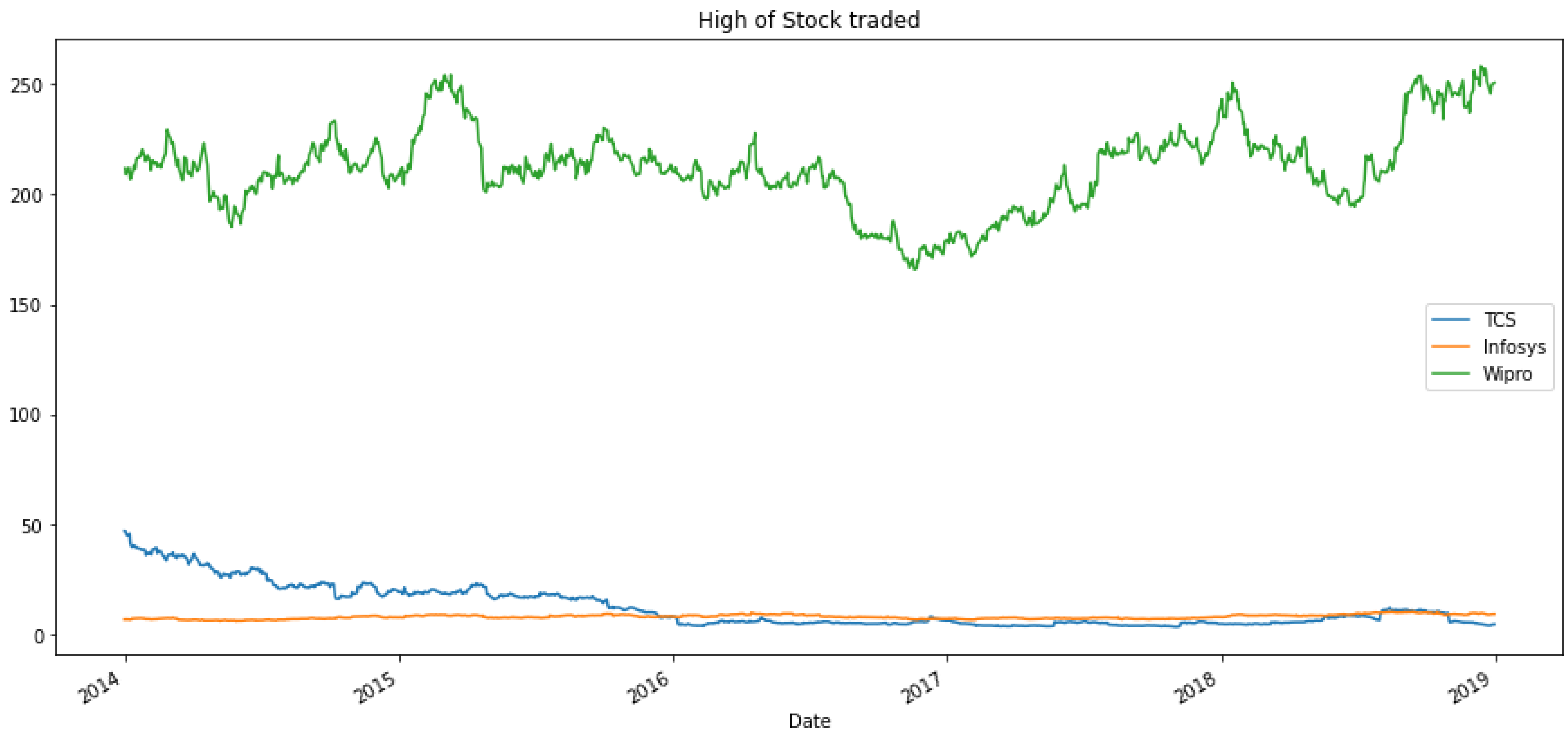
Out[4]: <matplotlib.legend.Legend at 0x253e751e250>



In [5]:

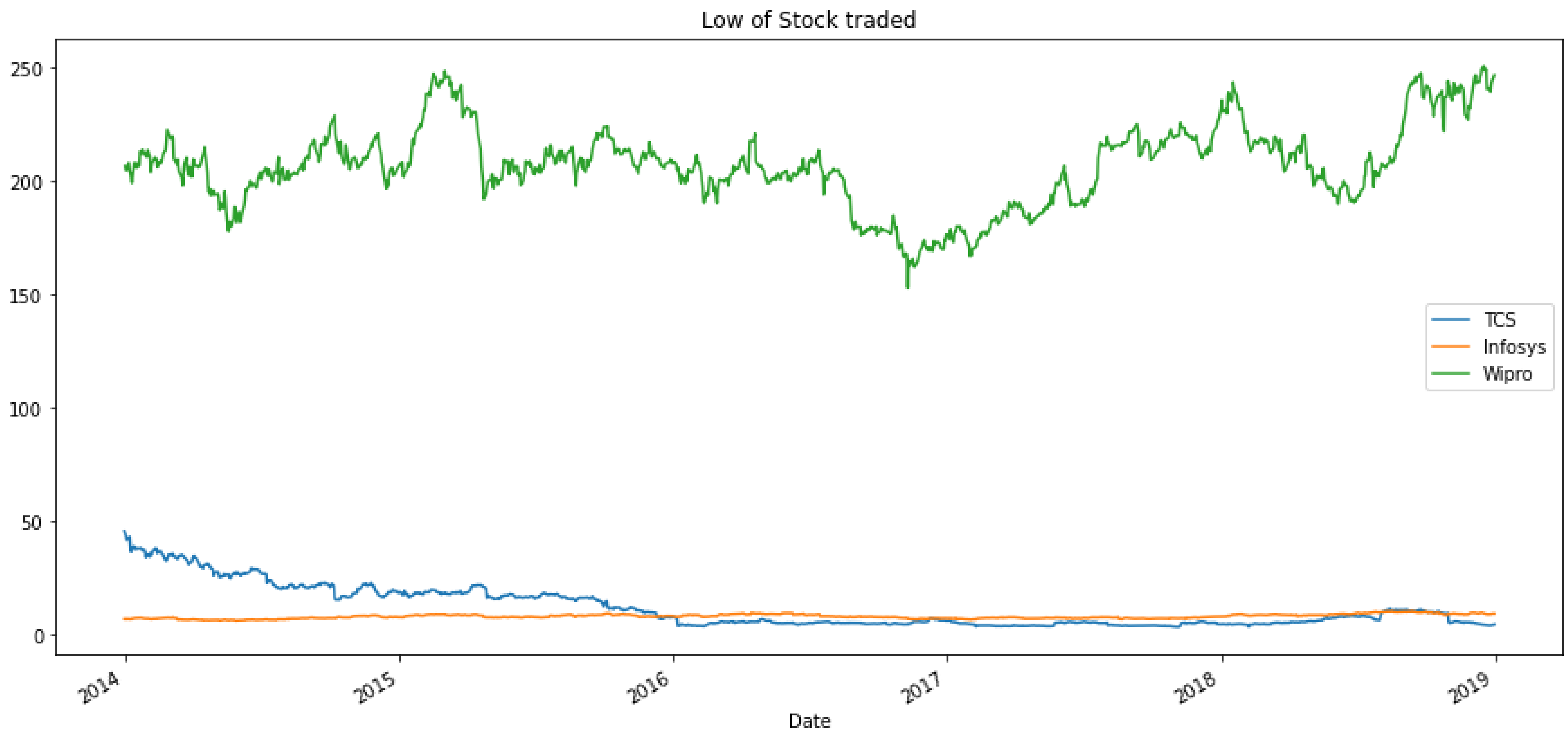
```
tcs['High'].plot(label = 'TCS', figsize = (15,7))
infy['High'].plot(label = "Infosys")
wipro['High'].plot(label = 'Wipro')
plt.title('High of Stock traded')
plt.legend()

```

Out[5]: <matplotlib.legend.Legend at 0x253e75a3490>



In [6]:
```
tcs['Low'].plot(label = 'TCS', figsize = (15,7))
infy['Low'].plot(label = "Infosys")
wipro['Low'].plot(label = 'Wipro')
plt.title('Low of Stock traded')
plt.legend()

```

Out[6]: <matplotlib.legend.Legend at 0x253e8a53880>

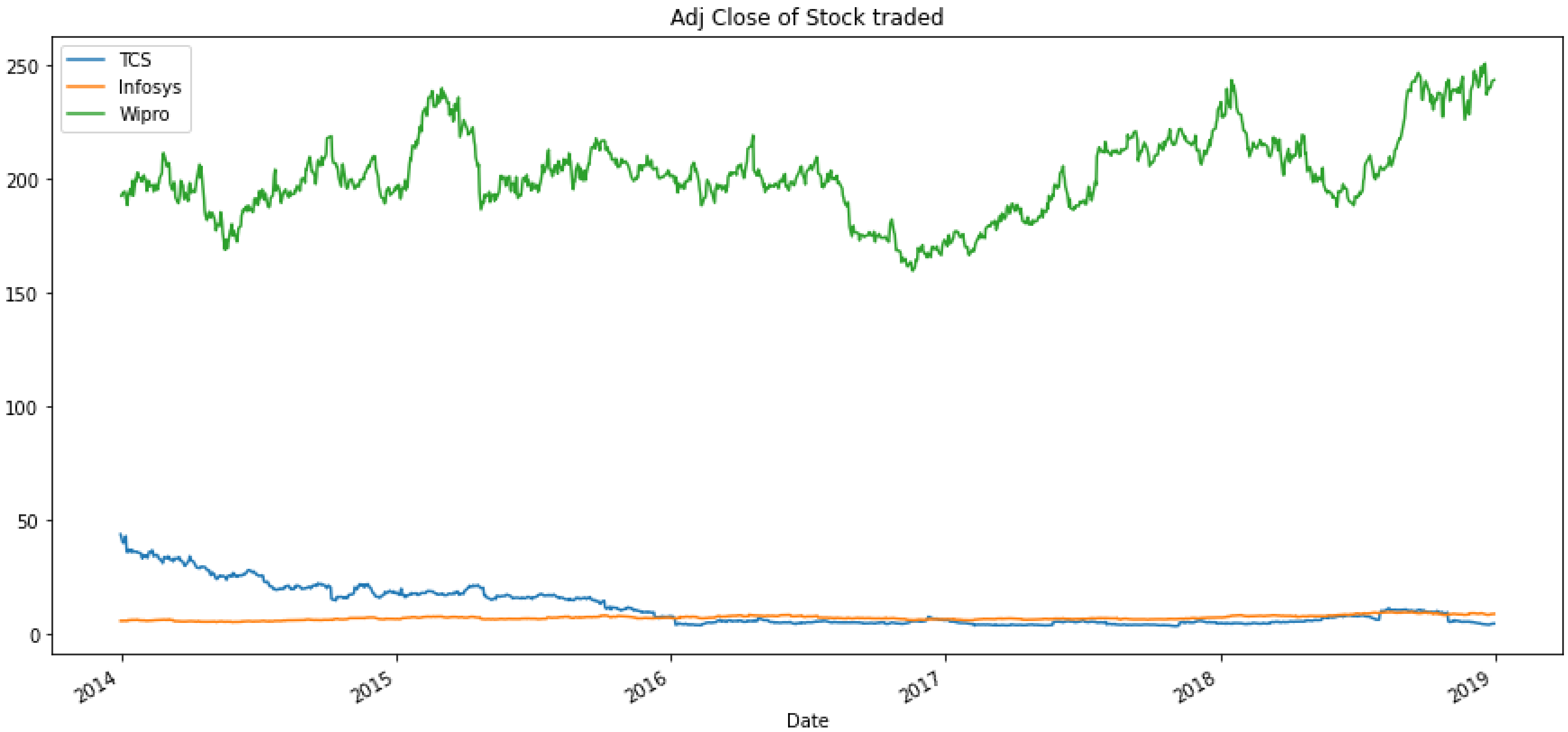


In [7]:
```
tcs.columns
```

Out[7]: Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')

In [8]:
```
tcs['Adj Close'].plot(label = 'TCS', figsize = (15,7))
infy['Adj Close'].plot(label = "Infosys")
wipro['Adj Close'].plot(label = 'Wipro')
plt.title('Adj Close of Stock traded')
plt.legend()
```

Out[8]: <matplotlib.legend.Legend at 0x253e8af3f10>



In [9]:
```
tcs.head()
```

Out[9]:

Date	Open	High	Low	Close	Adj Close	Volume
2013-12-31	46.290001	47.070000	45.599998	46.610001	43.570217	460800
2014-01-02	46.869999	46.869999	43.779999	43.779999	40.924782	798800
2014-01-03	44.790001	45.180000	41.779999	42.660000	39.877827	1247300
2014-01-06	44.669998	45.000000	42.759998	42.980000	40.176956	1101300
2014-01-07	43.840000	45.790001	43.119999	45.790001	42.803696	1677300

In [10]:
```
wipro.head()
```

Out[10]:

Date	Open	High	Low	Close	Adj Close	Volume
2014-01-01	211.500046	211.500046	206.662552	207.262558	192.681686	1181351
2014-01-02	207.262558	208.800049	204.806305	207.356308	192.768860	2441295
2014-01-03	205.125046	209.400055	204.862549	208.725052	194.041306	2953767
2014-01-06	209.962555	211.800049	208.125046	209.250046	194.529388	4948206
2014-01-07	209.250046	209.512558	204.375046	206.325058	191.810150	3286111

In [11]:

```
infy.head()
```

Out[11]:

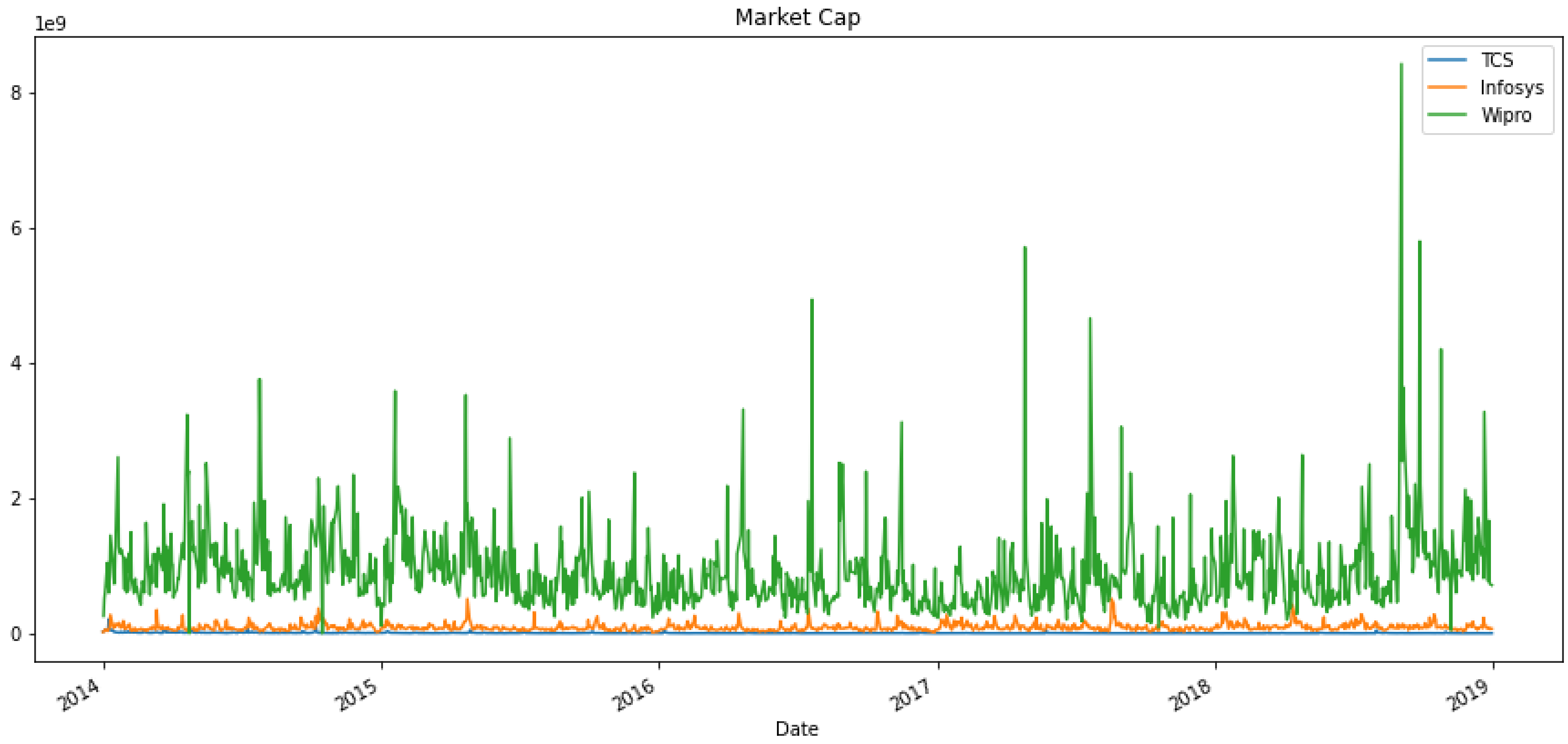
Date	Open	High	Low	Close	Adj Close	Volume
2013-12-31	7.08750	7.11250	7.06125	7.07500	5.697643	2891200
2014-01-02	7.03125	7.03125	6.92625	6.94125	5.589931	3642400
2014-01-03	7.14375	7.19750	7.11125	7.14375	5.753009	8421600
2014-01-06	7.11125	7.11375	7.02125	7.03875	5.668450	4820000
2014-01-07	6.97625	7.04875	6.95625	7.01125	5.646304	6201600

In [12]:

```python
# market Capitalisation
tcs['MarktCap'] = tcs['Open'] * tcs['Volume']
infy['MarktCap'] = infy['Open'] * infy['Volume']
wipro['MarktCap'] = wipro['Open'] * wipro['Volume']
tcs['MarktCap'].plot(label = 'TCS', figsize = (15,7))
infy['MarktCap'].plot(label = 'Infosys')
wipro['MarktCap'].plot(label = 'Wipro')
plt.title('Market Cap')
plt.legend()

```

Out[12]:

```
<matplotlib.legend.Legend at 0x253e8daeee0>
```



Only volume or stock prices do not provide a comparison between companies. In this case, we have plotted a graph for Volume * Share price to better compare the companies. As we can clearly see from the graph that Wipro seems to be traded on a higher side.

In [13]:
```
tcs
```

Out[13]:

Date	Open	High	Low	Close	Adj Close	Volume	MarktCap
2013-12-31	46.290001	47.070000	45.599998	46.610001	43.570217	460800	2.133043e+07
2014-01-02	46.869999	46.869999	43.779999	43.779999	40.924782	798800	3.743976e+07
2014-01-03	44.790001	45.180000	41.779999	42.660000	39.877827	1247300	5.586657e+07
2014-01-06	44.669998	45.000000	42.759998	42.980000	40.176956	1101300	4.919507e+07
2014-01-07	43.840000	45.790001	43.119999	45.790001	42.803696	1677300	7.353283e+07
...	...	...	...	...	...	...	...
2018-12-24	4.210000	4.370000	4.210000	4.220000	3.944782	132200	5.565620e+05
2018-12-26	4.220000	4.450000	4.210000	4.430000	4.141087	178300	7.524260e+05
2018-12-27	4.360000	4.480000	4.310000	4.450000	4.159782	297800	1.298408e+06
2018-12-28	4.460000	4.790000	4.460000	4.690000	4.384130	235600	1.050776e+06
2018-12-31	4.710000	4.860000	4.630000	4.770000	4.458913	456500	2.150115e+06

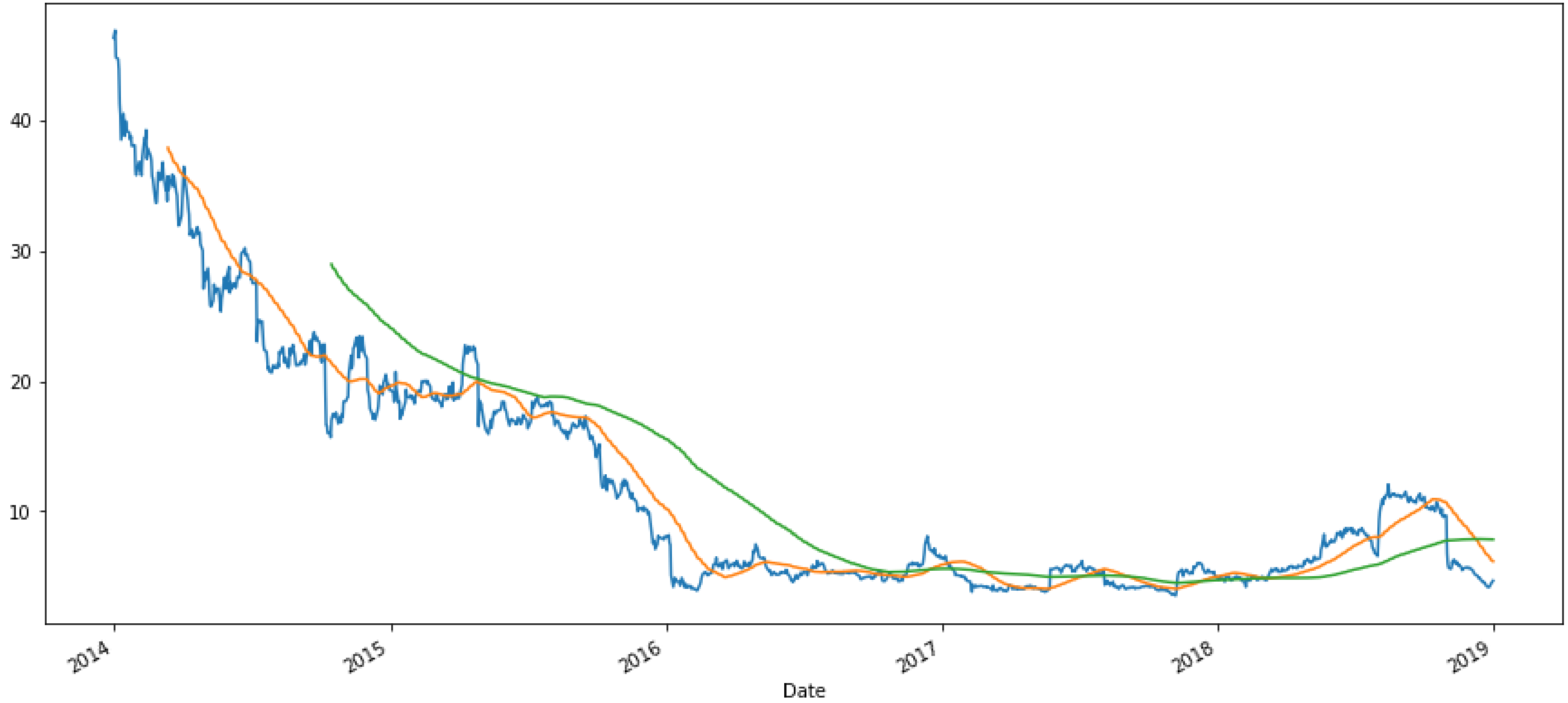
1259 rows × 7 columns

Moving Average

As we know the stock prices are highly volatile and prices change quickly with time. To observe any trend or pattern we can take the help of a 50-day 200-day average

In [14]:
```
tcs['MA50'] = tcs['Open'].rolling(50).mean()
tcs['MA200'] = tcs['Open'].rolling(200).mean()
tcs['Open'].plot(figsize=(15,7))
tcs['MA50'].plot()
tcs['MA200'].plot()
```

Out[14]: <AxesSubplot:xlabel='Date'>



Scatter Plot Matrix

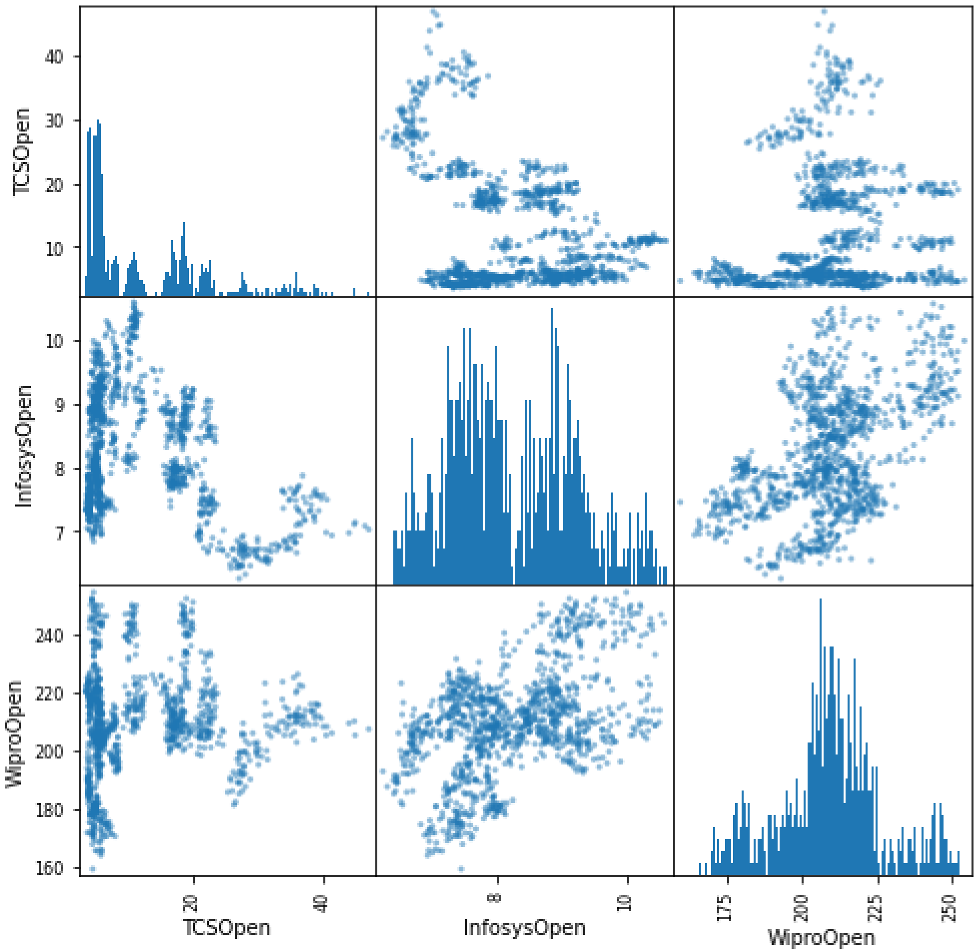
In [15]:
```
from pandas.plotting import scatter_matrix
```

In [16]:

```
data = pd.concat([tcs['Open'],infy['Open'],wipro['Open']],axis = 1)
data.columns = ['TCSOpen','InfosysOpen','WiproOpen']
scatter_matrix(data, figsize = (8,8), hist_kwds= {'bins':250})

```

Out[16]:

```
array([[<AxesSubplot:xlabel='TCSOpen', ylabel='TCSOpen'>,
        <AxesSubplot:xlabel='InfosysOpen', ylabel='TCSOpen'>,
        <AxesSubplot:xlabel='WiproOpen', ylabel='TCSOpen'>],
       [<AxesSubplot:xlabel='TCSOpen', ylabel='InfosysOpen'>,
        <AxesSubplot:xlabel='InfosysOpen', ylabel='InfosysOpen'>,
        <AxesSubplot:xlabel='WiproOpen', ylabel='InfosysOpen'>],
       [<AxesSubplot:xlabel='TCSOpen', ylabel='WiproOpen'>,
        <AxesSubplot:xlabel='InfosysOpen', ylabel='WiproOpen'>,
        <AxesSubplot:xlabel='WiproOpen', ylabel='WiproOpen'>]],
      dtype=object)
```



The above graph is the combination of histograms for each company and a subsequent scattered plot taking two companies' stocks at a time. From the graph, we can clearly figure out that Wipro stocks are loosely showing a linear correlation with Infosys.

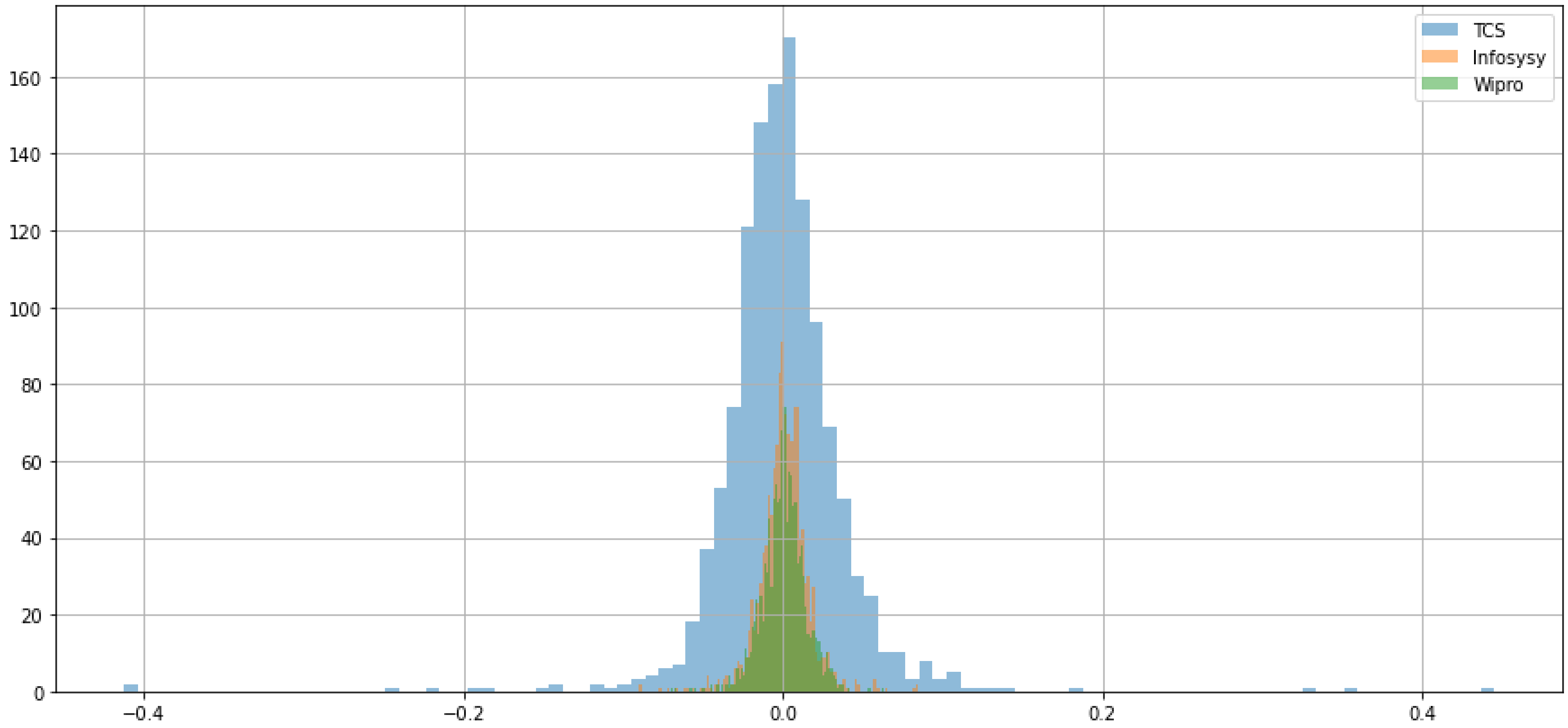
Percentage Increase in Stock Values

A percentage increase in stock value is the change in stock comparing that to the previous day. The bigger the value either positive or negative the volatile the stock is.

In [17]:

```
tcs['returns'] = (tcs['Close']/tcs['Close'].shift(1)) -1
infy['returns'] = (infy['Close']/infy['Close'].shift(1))-1
wipro['returns'] = (wipro['Close']/wipro['Close'].shift(1)) - 1
tcs['returns'].hist(bins = 100, label = 'TCS', alpha = 0.5, figsize = (15,7))
infy['returns'].hist(bins = 100, label = 'Infosysy', alpha = 0.5)
wipro['returns'].hist(bins = 100, label = 'Wipro', alpha = 0.5)
plt.legend()

```

Out[17]: <matplotlib.legend.Legend at 0x253ea02dd00>



Conclusion

The above analysis can be used to understand a stock's short-term and long-term behaviour. A decision support system can be created which stock to pick from industry for low-risk low gain or high-risk high gain depending on the risk apatite of the investor.

In [18]:
```
tcs.isna().sum()
```

Out[18]:
```
Open           0
High           0
Low            0
Close          0
Adj Close      0
Volume         0
MarktCap       0
MA50          49
MA200        199
returns        1
dtype: int64
```

In [19]:
```
wipro.isna().sum()
```

Out[19]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
MarktCap     0
returns      1
dtype: int64
```

In [20]:
```
infy.isna().sum()
```

Out[20]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
MarktCap     0
returns      1
dtype: int64
```

In [21]:
```
tcs.fillna(tcs.mean(),inplace=True)
wipro.fillna(wipro.mean(),inplace=True)
infy.fillna(infy.mean(),inplace=True)
```

In [22]:
```
tcs.isna().sum()
```

Out[22]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
MarktCap     0
MA50         0
MA200        0
returns      0
dtype: int64
```

In [23]:
```
wipro.isna().sum()
```

Out[23]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
MarktCap     0
returns      0
dtype: int64
```

In [24]:
```
infy.isna().sum()
```

Out[24]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
MarktCap     0
returns      0
dtype: int64
```

In [25]:
```
tcs['name']='tcs'
wipro['name']='wipro'
infy['name']='infy'
```

In [26]:

```
df = pd.concat([tcs,wipro,infy])
df
```

Out[26]:

Date	Open	High	Low	Close	Adj Close	Volume	MarktCap	MA50	MA200	returns	name
2013-12-31	46.290001	47.070000	45.599998	46.610001	43.570217	460800	2.133043e+07	11.759829	10.633039	-0.000882	tcs
2014-01-02	46.869999	46.869999	43.779999	43.779999	40.924782	798800	3.743976e+07	11.759829	10.633039	-0.060717	tcs
2014-01-03	44.790001	45.180000	41.779999	42.660000	39.877827	1247300	5.586657e+07	11.759829	10.633039	-0.025582	tcs
2014-01-06	44.669998	45.000000	42.759998	42.980000	40.176956	1101300	4.919507e+07	11.759829	10.633039	0.007501	tcs
2014-01-07	43.840000	45.790001	43.119999	45.790001	42.803696	1677300	7.353283e+07	11.759829	10.633039	0.065379	tcs
...	...	...	...	...	...	...	...	...	...	...	...
2018-12-24	9.190000	9.240000	9.070000	9.080000	8.327432	8590700	7.894853e+07	NaN	NaN	-0.002198	infy
2018-12-26	9.150000	9.380000	9.120000	9.380000	8.602568	9004200	8.238843e+07	NaN	NaN	0.033040	infy
2018-12-27	9.300000	9.450000	9.280000	9.450000	8.666765	9856500	9.166545e+07	NaN	NaN	0.007463	infy
2018-12-28	9.480000	9.500000	9.380000	9.430000	8.648423	6818500	6.463938e+07	NaN	NaN	-0.002116	infy
2018-12-31	9.470000	9.530000	9.390000	9.520000	8.730963	7229400	6.846242e+07	NaN	NaN	0.009544	infy

3748 rows × 11 columns

In [27]:

```
df.fillna(df.mean(),inplace=True)
df.isna().sum()
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_15232/2516141900.py:1: FutureWarning: Dropping of nuisance columns in DataF
rame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only v
alid columns before calling the reduction.
  df.fillna(df.mean(),inplace=True)
```

Out[27]:

```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
MarktCap     0
MA50         0
MA200        0
returns      0
name         0
dtype: int64
```

In [28]:

```
df
```

Out[28]:

	Open	High	Low	Close	Adj Close	Volume	MarktCap	MA50	MA200	returns	name
Date											
2013-12-31	46.290001	47.070000	45.599998	46.610001	43.570217	460800	2.133043e+07	11.759829	10.633039	-0.000882	tcs
2014-01-02	46.869999	46.869999	43.779999	43.779999	40.924782	798800	3.743976e+07	11.759829	10.633039	-0.060717	tcs
2014-01-03	44.790001	45.180000	41.779999	42.660000	39.877827	1247300	5.586657e+07	11.759829	10.633039	-0.025582	tcs
2014-01-06	44.669998	45.000000	42.759998	42.980000	40.176956	1101300	4.919507e+07	11.759829	10.633039	0.007501	tcs
2014-01-07	43.840000	45.790001	43.119999	45.790001	42.803696	1677300	7.353283e+07	11.759829	10.633039	0.065379	tcs
...	...	...	...	...	...	...	...	...	...	...	...
2018-12-24	9.190000	9.240000	9.070000	9.080000	8.327432	8590700	7.894853e+07	11.759829	10.633039	-0.002198	infy
2018-12-26	9.150000	9.380000	9.120000	9.380000	8.602568	9004200	8.238843e+07	11.759829	10.633039	0.033040	infy
2018-12-27	9.300000	9.450000	9.280000	9.450000	8.666765	9856500	9.166545e+07	11.759829	10.633039	0.007463	infy
2018-12-28	9.480000	9.500000	9.380000	9.430000	8.648423	6818500	6.463938e+07	11.759829	10.633039	-0.002116	infy
2018-12-31	9.470000	9.530000	9.390000	9.520000	8.730963	7229400	6.846242e+07	11.759829	10.633039	0.009544	infy

3748 rows × 11 columns

In [29]:
```
# df = pd.get_dummies(df, columns=['name'])
df = pd.get_dummies(df, columns=['name'])
df
```

Out[29]:

Date	Open	High	Low	Close	Adj Close	Volume	MarktCap	MA50	MA200	returns	name_infy	name_tcs	nan
2013-12-31	46.290001	47.070000	45.599998	46.610001	43.570217	460800	2.133043e+07	11.759829	10.633039	-0.000882	0	1	
2014-01-02	46.869999	46.869999	43.779999	43.779999	40.924782	798800	3.743976e+07	11.759829	10.633039	-0.060717	0	1	
2014-01-03	44.790001	45.180000	41.779999	42.660000	39.877827	1247300	5.586657e+07	11.759829	10.633039	-0.025582	0	1	
2014-01-06	44.669998	45.000000	42.759998	42.980000	40.176956	1101300	4.919507e+07	11.759829	10.633039	0.007501	0	1	
2014-01-07	43.840000	45.790001	43.119999	45.790001	42.803696	1677300	7.353283e+07	11.759829	10.633039	0.065379	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2018-12-24	9.190000	9.240000	9.070000	9.080000	8.327432	8590700	7.894853e+07	11.759829	10.633039	-0.002198	1	0	

In [30]:
```
x = df.drop(columns=['returns'])
y = df['returns']
```

In [31]:
```
y.shape
```

Out[31]: (3748,)

preprocessing

In [32]:
```
from sklearn.preprocessing import StandardScaler
```

In [33]:
```
scaler = StandardScaler()
```

In [34]:
```
x = scaler.fit_transform(x)
```

In [ ]:
```

```

In [35]:
```
from sklearn.model_selection import train_test_split
```

In [36]:
```
X_train,X_test,y_train,y_test = train_test_split(x,y, test_size=0.3)
```

In [37]:
```
X_train
```

Out[37]:
```
array([[-0.69749601, -0.69850236, -0.69615888, ...,  1.40604617,
        -0.7112142 , -0.69891553],
       [-0.70662865, -0.70801542, -0.70566503, ...,  1.40604617,
        -0.7112142 , -0.69891553],
       [ 1.43042491,  1.42007339,  1.43872532, ..., -0.7112142 ,
        -0.7112142 ,  1.43078806],
       ...,
       [ 1.31354553,  1.29590424,  1.26551996, ..., -0.7112142 ,
        -0.7112142 ,  1.43078806],
       [-0.75510602, -0.75537051, -0.75496814, ..., -0.7112142 ,
         1.40604617, -0.69891553],
       [-0.60250583, -0.59895681, -0.60372898, ..., -0.7112142 ,
         1.40604617, -0.69891553]])
```

In [38]:
```
X_test
```

Out[38]:
```
array([[-0.73078769, -0.73171925, -0.73262599, ..., -0.7112142 ,
         1.40604617, -0.69891553],
       [ 1.81650511,  1.8229334 ,  1.81051268, ..., -0.7112142 ,
        -0.7112142 ,  1.43078806],
       [ 1.18929906,  1.18553167,  1.19704342, ..., -0.7112142 ,
        -0.7112142 ,  1.43078806],
       ...,
       [-0.75499983, -0.75558075, -0.75690159, ..., -0.7112142 ,
         1.40604617, -0.69891553],
       [-0.75022112, -0.74412302, -0.74906036, ..., -0.7112142 ,
         1.40604617, -0.69891553],
       [ 1.1243882 ,  1.1053144 ,  1.09191173, ..., -0.7112142 ,
        -0.7112142 ,  1.43078806]])
```

In [39]:
```
y_train
```

Out[39]:
```
Date
2016-06-06    0.005099
2018-03-07   -0.005559
2015-05-26   -0.008613
2015-08-19    0.008889
2018-07-26   -0.001944
                ...   
2016-09-20    0.006085
2014-01-28   -0.002577
2016-02-11   -0.026355
2017-03-30   -0.006818
2015-01-27   -0.009429
Name: returns, Length: 2623, dtype: float64
```

In [40]:
```
y_test
```

Out[40]:
```
Date
2016-12-28   -0.027149
2018-09-14    0.004411
2017-05-08    0.006008
2014-11-07    0.006215
2015-03-05   -0.002143
                 ...
2014-08-07   -0.007634
2016-03-17    0.001393
2016-01-28   -0.043678
2016-06-23    0.112971
2016-09-29   -0.023775
Name: returns, Length: 1125, dtype: float64
```

Here we will evaluate and compare the following models using a regression on the train test.

1.Linear Regression

2.Random Forest Regression.

3.Support Vector Regression(SVR).

Linear Regression

In [41]:
```
from sklearn.linear_model import LinearRegression
```

In [42]:
```
lr = LinearRegression()
```

In [43]:
```
lr.fit(X_train,y_train)
```

Out[43]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
y_predict = lr.predict(X_test)
```

In [45]:
```
y_predict
```

Out[45]:
```
array([-0.0012606 ,  0.01063209,  0.00439987, ..., -0.00103895,
        0.00203682, -0.0208568 ])
```

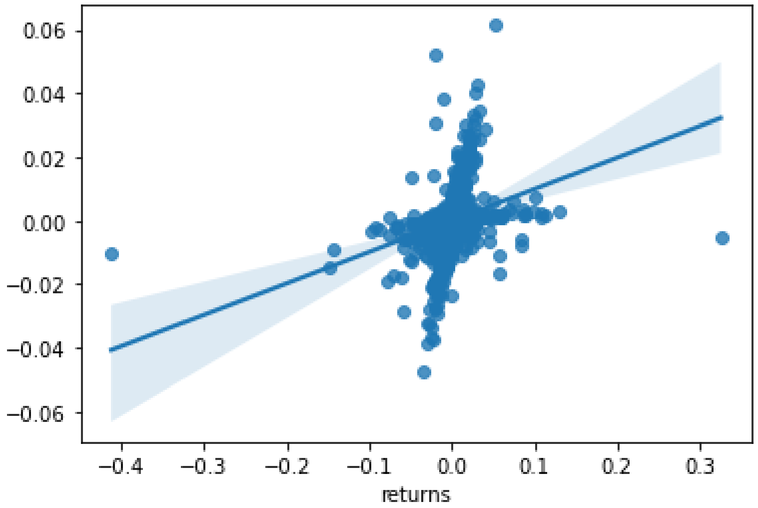
In [46]:
```
lr.coef_
```

Out[46]:
```
array([-4.30762290e-01, -1.76672817e-01, -1.25922105e-02,  6.37328863e-01,
       -1.34174368e-02, -3.09523542e-03,  3.21234587e-03, -1.25084484e-03,
        4.37032140e-04,  3.39035889e-03,  7.36893251e-04, -4.15150816e-03])
```

In [47]:
```
import seaborn as sns
sns.regplot(y_test,y_predict)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as ke
yword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[47]:
```
<AxesSubplot:xlabel='returns'>
```

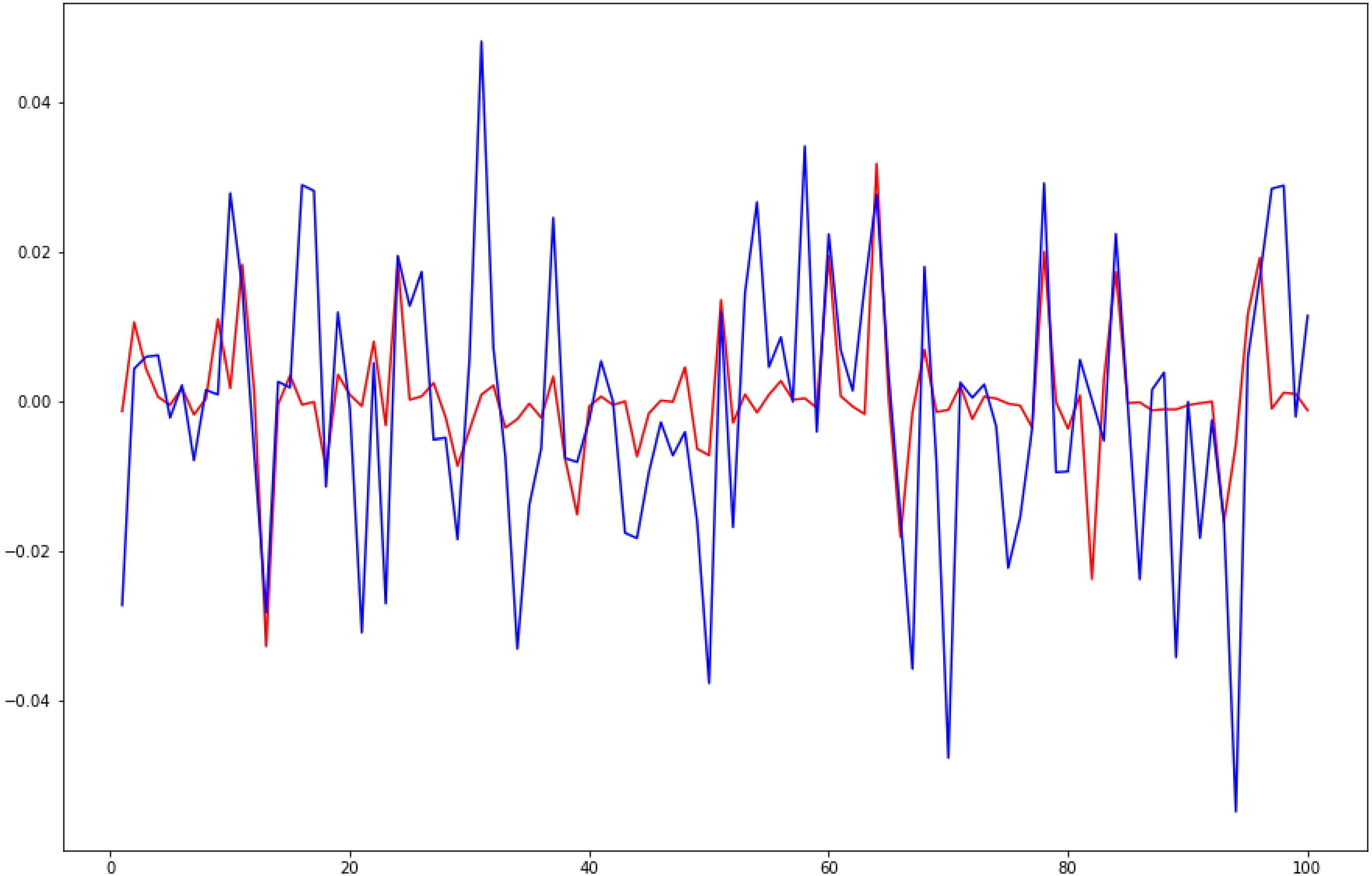


Here we see there is a positive relstionship between dependent variable and independent variable.

In [48]:
```
from sklearn.metrics import mean_absolute_error
```

In [ ]:
```

```

In [50]:
```
y_predict1=y_predict[0:100]
y_test1=y_test[0:100]
xaxis=np.linspace(1,len(y_predict1),len(y_test1))
fig, ax=plt.subplots(figsize=(15,10))
plt.plot(xaxis ,y_predict1, color='red')
plt.plot(xaxis ,y_test1 ,color='blue')
plt.show()
```



In [61]:
```
mean_absolute_error(y_test,y_predict)
```

Out[61]: 0.013535281453482779

On this graph here we see the gap(difference) of y_test an y_predict is 0.13

RandomForest

In [51]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [52]:
```
rr = RandomForestRegressor(random_state=0)
```

In [53]:
```
rr.fit(X_train,y_train)
```

Out[53]:
```
RandomForestRegressor(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [54]:
```
y_pred = rr.predict(X_test)
y_pred
```

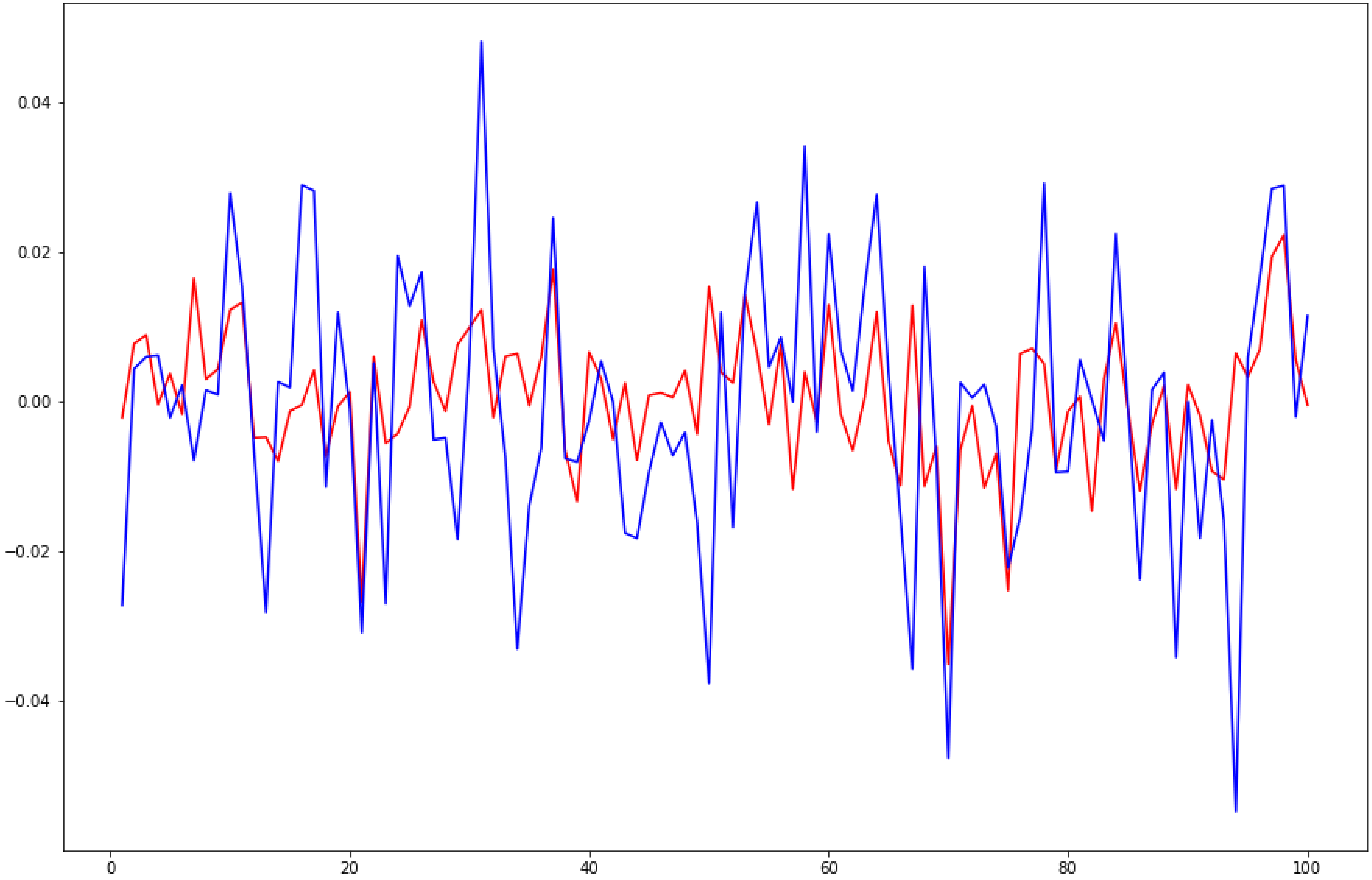
Out[54]:
```
array([-0.00212363,  0.00777762,  0.00892983, ..., -0.02891732,
        0.03720348, -0.00673748])
```

In [ ]:
```

```

In [ ]:
```

```

In [55]:

```
predrr1 = y_pred[0:100]
ytest2 = y_test[0:100]
xaxis2 = np.linspace(1,len(predrr1), len(predrr1))
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(15, 10))
plt.plot(xaxis2, predrr1, color='red')
plt.plot(xaxis2, ytest2, color='blue')
plt.show()
```



In [56]:
```
mean_absolute_error(y_test,y_pred)
```

Out[56]: 0.013068360247688865

Not quite as good as Linear regression, at least according to this regression.

Here we see the gap(difference) of y_test an y_predict is 013.

SVR

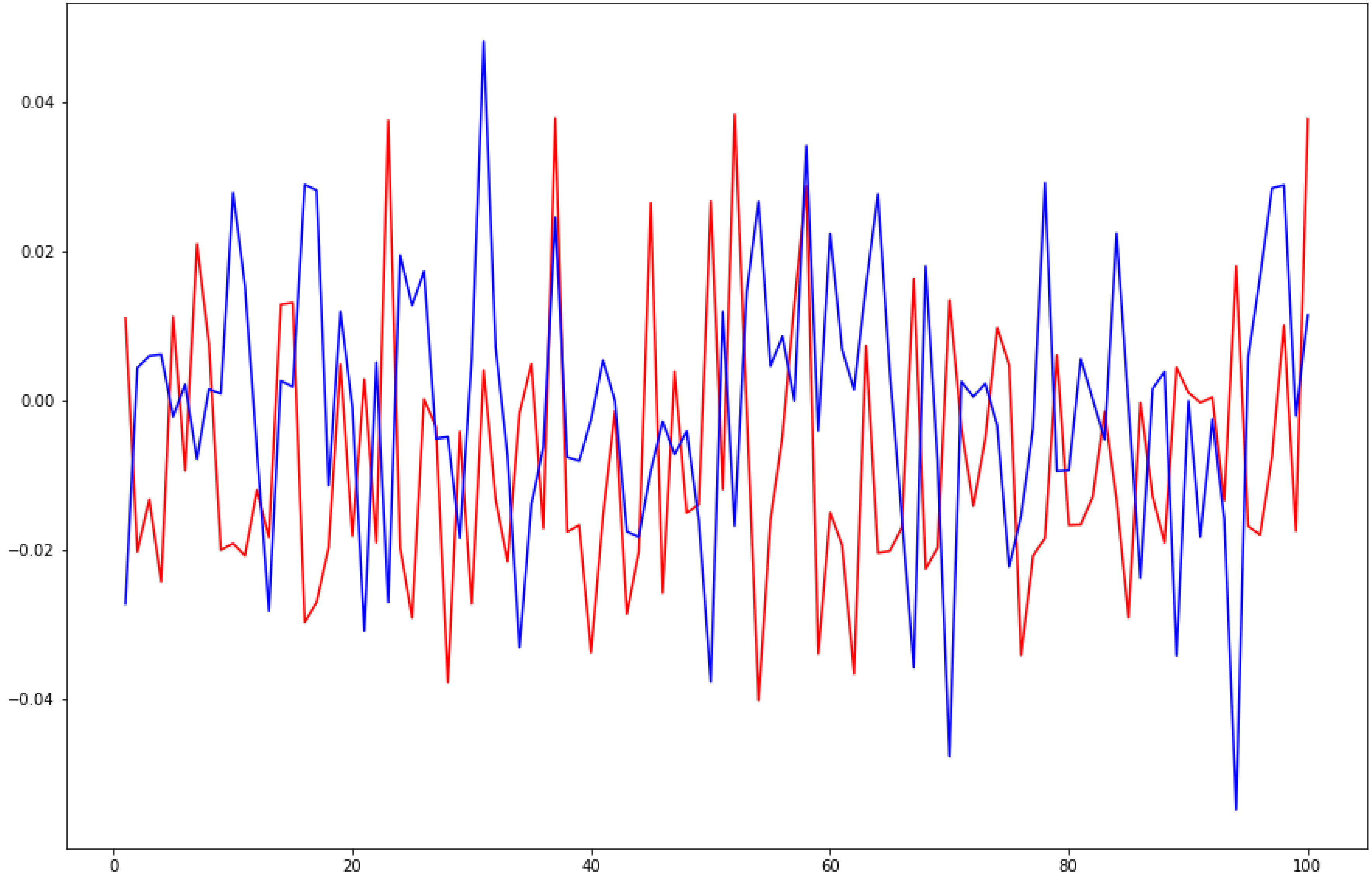
In [57]:

```
from sklearn.svm import SVR
model = SVR()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
y_pred
```

Out[57]:

```
array([ 0.01111881, -0.02017363, -0.0131483 , ..., -0.004367  ,
        0.01577078, -0.02351896])
```

In [58]:
```
predrr1 = y_pred[0:100]
ytest2 = y_test[0:100]
xaxis2 = np.linspace(1,len(predrr1), len(predrr1))
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(15, 10))
plt.plot(xaxis2, predrr1, color='red')
plt.plot(xaxis2, ytest2, color='blue')
plt.show()
```



In [60]:

```
mean_absolute_error(y_test,y_pred)
```

Out[60]: 0.024143420975696918

Here, we see Support vector regression is the best model to predict a stock/financial score data.

On this graph here we see the gap(difference) of y_test an y_predict is 024

In [ ]:

```
 
```