Target of Stock price close prediction with ARIMA model(Time series analysis)

In [1]:

```
#Changing working directory
import os
print(os.getcwd())
import warnings
warnings.filterwarnings('ignore')
```

```
/Users/myyntiimac
```

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARMA
df = pd.read_csv('/Users/myyntiimac/Desktop/HCLTECH.csv')
df.head(30)
```

Out[2]:

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%De
0	2000-01-11	HCLTECH	EQ	580.00	1550.00	1725.00	1492.00	1560.00	1554.45	1582.72	1192200	1.886915e+14	NaN	NaN	
1	2000-01-12	HCLTECH	EQ	1554.45	1560.00	1678.85	1560.00	1678.85	1678.85	1657.05	344850	5.714349e+13	NaN	NaN	
2	2000-01-13	HCLTECH	EQ	1678.85	1790.00	1813.20	1781.00	1813.20	1813.20	1804.69	53000	9.564880e+12	NaN	NaN	
3	2000-01-14	HCLTECH	EQ	1813.20	1958.30	1958.30	1835.00	1958.30	1958.30	1939.90	270950	5.256169e+13	NaN	NaN	
4	2000-01-17	HCLTECH	EQ	1958.30	2115.00	2115.00	1801.65	1801.65	1801.65	1990.55	428800	8.535473e+13	NaN	NaN	
5	2000-01-18	HCLTECH	EQ	1801.65	1730.55	1815.00	1657.55	1775.00	1774.50	1716.39	359900	6.177280e+13	NaN	NaN	
6	2000-01-19	HCLTECH	EQ	1774.50	1815.00	1889.00	1760.00	1842.80	1851.15	1842.81	316050	5.824204e+13	NaN	NaN	
7	2000-01-20	HCLTECH	EQ	1851.15	1865.00	1865.00	1750.00	1753.50	1757.85	1801.37	204700	3.687409e+13	NaN	NaN	
8	2000-01-21	HCLTECH	EQ	1757.85	1761.00	1815.00	1705.00	1786.00	1781.35	1774.01	282150	5.005360e+13	NaN	NaN	
9	2000-01-24	HCLTECH	EQ	1781.35	1834.90	1923.90	1795.00	1923.90	1923.90	1875.34	328650	6.163317e+13	NaN	NaN	
10	2000-01-25	HCLTECH	EQ	1923.90	1990.00	2077.85	1990.00	2077.85	2077.85	2065.71	313500	6.476010e+13	NaN	NaN	
11	2000-01-27	HCLTECH	EQ	2077.85	2239.90	2239.90	1965.00	2078.00	2080.60	2101.92	352400	7.407150e+13	NaN	NaN	
12	2000-01-28	HCLTECH	EQ	2080.60	2100.00	2245.00	2020.20	2190.00	2196.95	2173.00	360800	7.840197e+13	NaN	NaN	
13	2000-01-31	HCLTECH	EQ	2196.95	2111.00	2200.00	2021.20	2021.20	2030.40	2083.45	217650	4.534620e+13	NaN	NaN	
14	2000-02-01	HCLTECH	EQ	2030.40	1981.20	2180.00	1981.20	2160.00	2162.25	2110.66	261000	5.508834e+13	NaN	NaN	
15	2000-02-02	HCLTECH	EQ	2162.25	2260.00	2260.00	2010.00	2035.00	2024.25	2085.91	326550	6.811542e+13	NaN	NaN	
16	2000-02-03	HCLTECH	EQ	2024.25	2075.00	2075.00	1915.20	1965.00	1956.50	1989.66	175800	3.497821e+13	NaN	NaN	
17	2000-02-04	HCLTECH	EQ	1956.50	1993.00	2064.00	1965.00	1977.00	1986.60	2008.25	176500	3.544560e+13	NaN	NaN	
18	2000-02-07	HCLTECH	EQ	1986.60	1985.00	2119.95	1960.05	2036.00	2062.95	2048.51	277850	5.691791e+13	NaN	NaN	
19	2000-02-08	HCLTECH	EQ	2062.95	2049.40	2060.00	1970.00	2014.00	1999.20	2017.09	199100	4.016027e+13	NaN	NaN	
20	2000-02-09	HCLTECH	EQ	1999.20	2100.00	2100.00	1975.00	1991.00	1999.00	2011.50	180500	3.630763e+13	NaN	NaN	
21	2000-02-10	HCLTECH	EQ	1999.00	1976.65	2014.80	1965.00	1990.00	1996.75	1992.36	76850	1.531132e+13	NaN	NaN	
22	2000-02-11	HCLTECH	EQ	1996.75	1998.00	2156.50	1992.00	2156.50	2156.50	2110.61	166150	3.506781e+13	NaN	NaN	
23	2000-02-14	HCLTECH	EQ	2156.50	2199.00	2329.05	2199.00	2329.05	2324.70	2293.27	265150	6.080618e+13	NaN	NaN	
24	2000-02-15	HCLTECH	EQ	2324.70	2350.00	2510.70	2210.00	2500.00	2508.40	2425.21	224500	5.444586e+13	NaN	NaN	
25	2000-02-16	HCLTECH	EQ	2508.40	2560.00	2560.00	2307.75	2418.00	2423.10	2411.48	260850	6.290343e+13	NaN	NaN	
26	2000-02-17	HCLTECH	EQ	2423.10	2445.00	2616.95	2380.00	2616.95	2614.70	2526.71	276550	6.987609e+13	NaN	NaN	
27	2000-02-18	HCLTECH	EQ	2614.70	2800.00	2823.90	2760.50	2771.00	2775.95	2807.40	173400	4.868032e+13	NaN	NaN	
28	2000-02-21	HCLTECH	EQ	2775.95	2823.00	2998.00	2725.00	2805.00	2819.15	2883.95	151800	4.377830e+13	NaN	NaN	
29	2000-02-22	HCLTECH	EQ	2819.15	2800.00	2875.00	2652.00	2710.00	2700.30	2772.79	88250	2.446984e+13	NaN	NaN	

```
#from this we consider our main column is previous close , on what target this stock price is close for particuler day
```

In [5]:

```
#Cleaning the data, first delete NA column
df1 = df.dropna()

#Make data column as index
df1.index = pd.to_datetime(df1.Date)

df1 = df1["Prev Close"]["2013-01-01":"2013-12-02"]
df1.describe()

```

Out[5]:

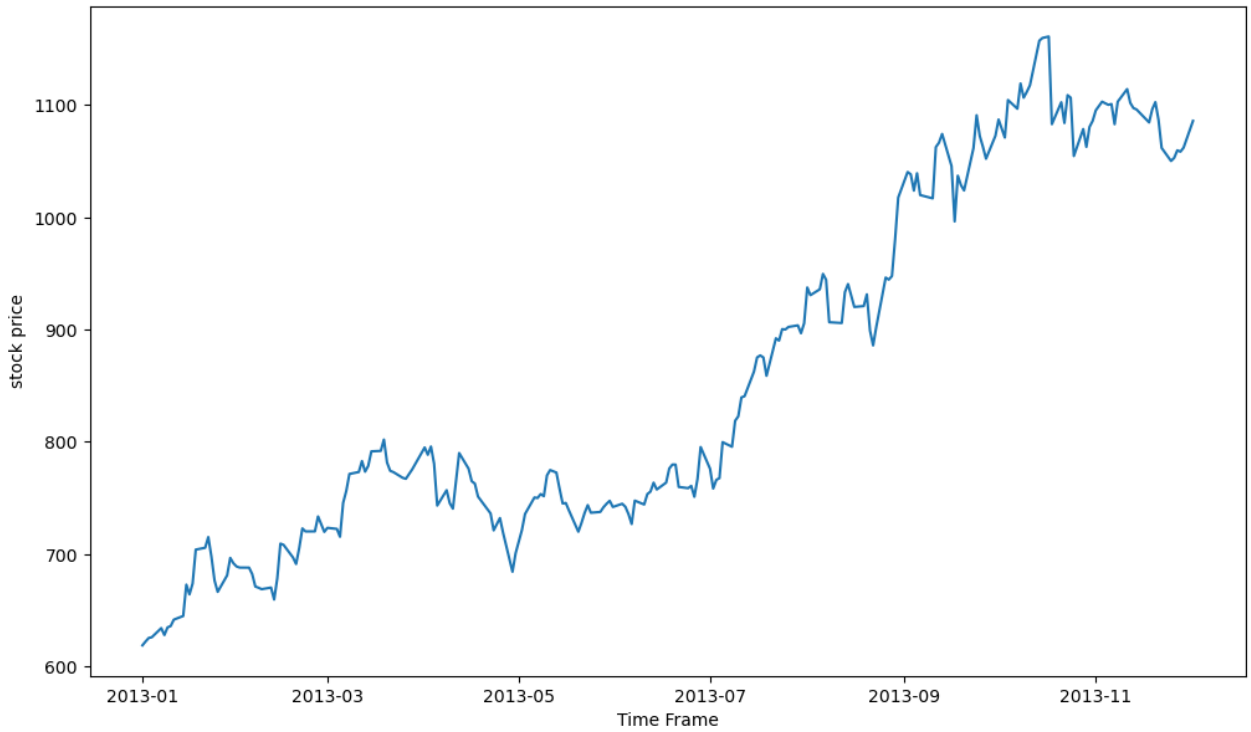
```
count     230.000000
mean      852.953478
std       156.484472
min       618.700000
25%       736.350000
50%       777.450000
75%      1023.962500
max      1161.150000
Name: Prev Close, dtype: float64
```

In [6]:

```
#Data exploration
#Data Exploration
plt.figure(figsize=(12,7))
fig = plt.figure(1)
ax1 = fig.add_subplot(111)
ax1.set_xlabel('Time Frame')
ax1.set_ylabel('stock price')
ax1.plot(df1)

```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x13fc09790>]
```



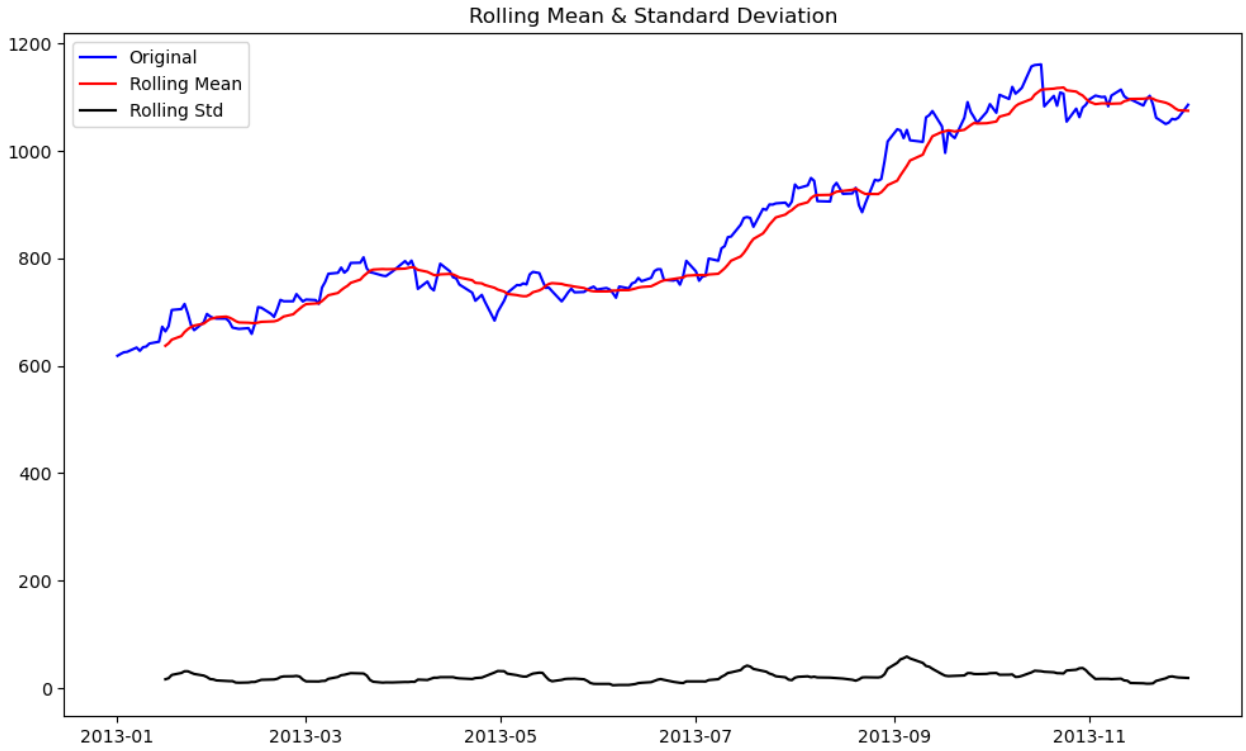

```
insight:the trend is upward but seasonality is ambigouus
```

In [ ]:

```
#check the stationary
#Method 1: Rolling statistics(Rolling mean, Rolling  std.dev)
#Method 2:Duckey fuller

```

In [8]:

```
#determining rolling statistics
rolLmean = df1.rolling(12).mean()
rolLstd =df1.rolling(12).std()

plt.figure(figsize=(12,7))
fig = plt.figure(1)

#Plot rolling statistics:
orig = plt.plot(df1, color='blue',label='Original')
mean = plt.plot(rolLmean, color='red', label='Rolling Mean')
std = plt.plot(rolLstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)

```



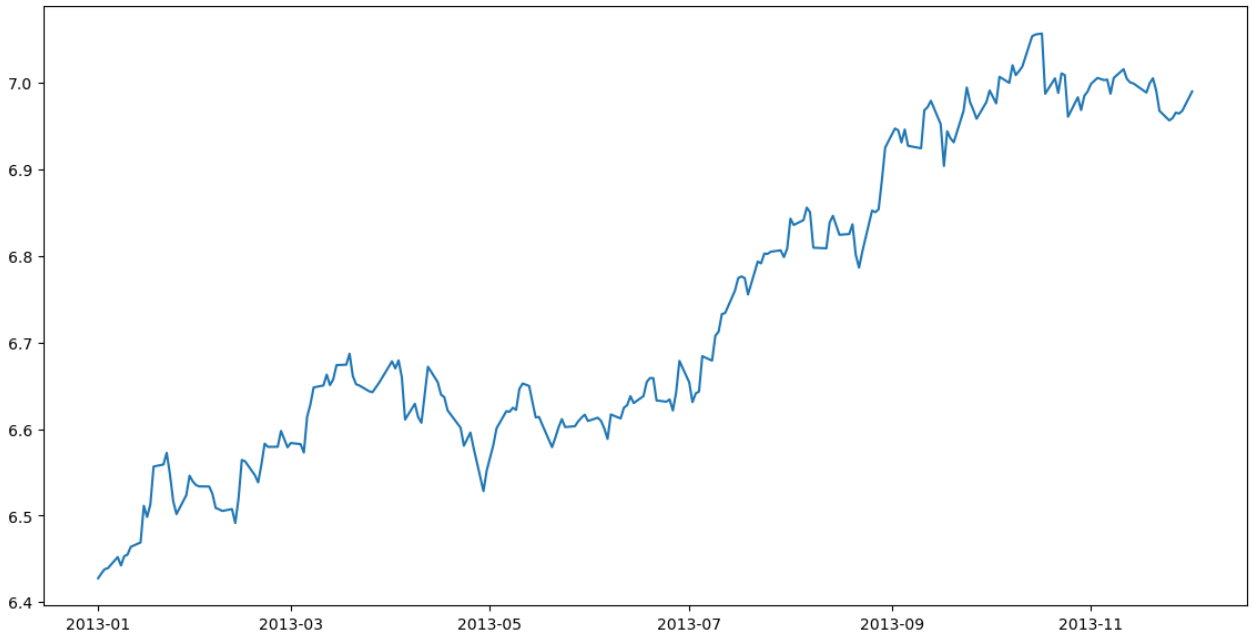

```
Insight:std .dev is constant over time but mean is not constant, to be stationery of time series both need to
be constant, that means this series is not stationery
```

In [9]:

```
#Making stationery with log transformaion
#Lets try transformation
plt.figure(figsize=(14,7))
fig = plt.figure(1)

import numpy as np
ts_log = np.log(df1)
plt.plot(ts_log)
```

Out[9]:

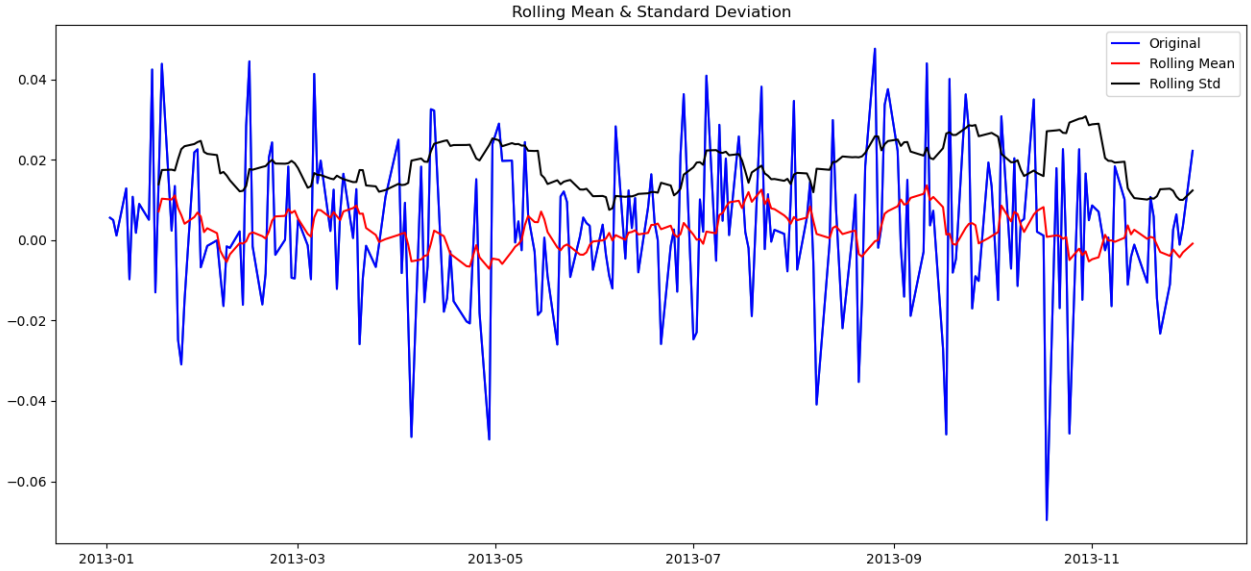
```
[<matplotlib.lines.Line2D at 0x1425f06d0>]
```




```
Insight:its not become statinery after log transfomation , Lets try differencing
```

In [19]:

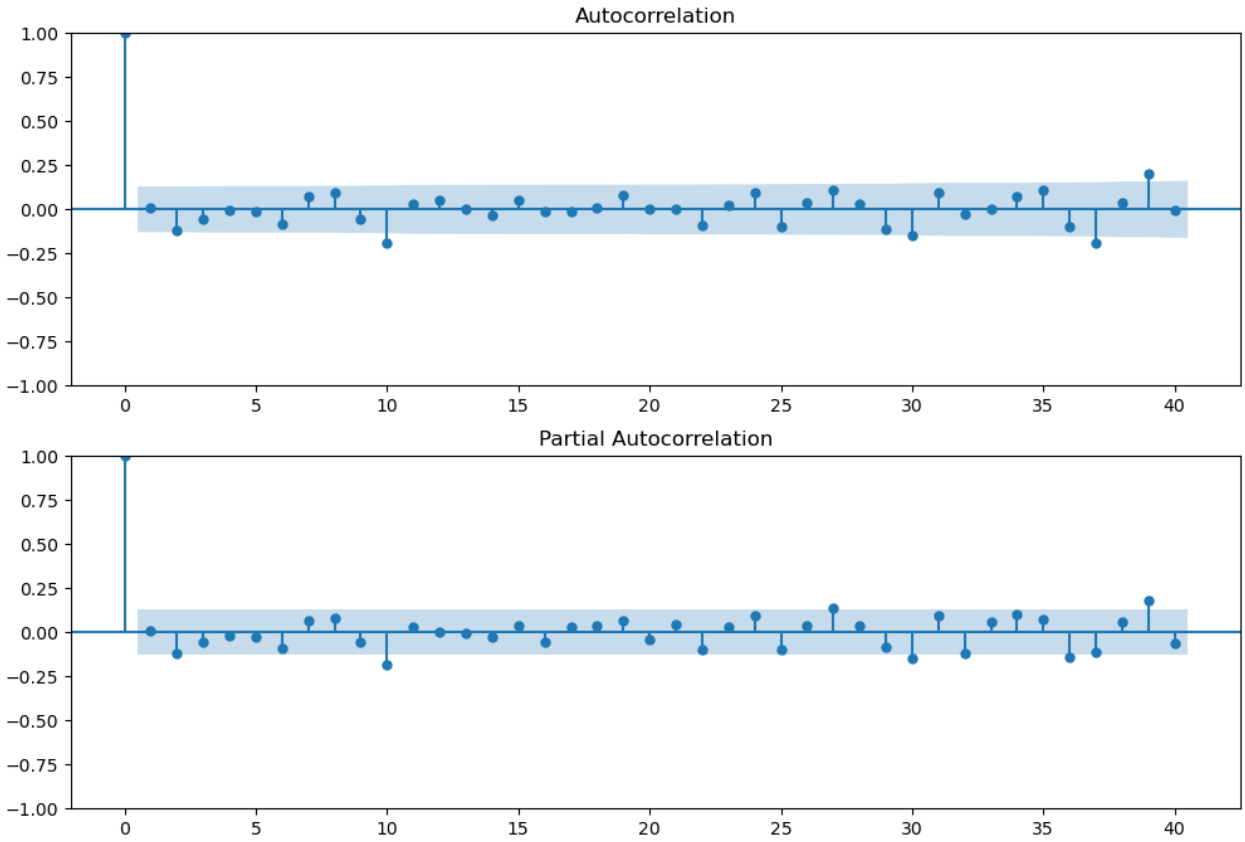
```
#Lets try differencing
plt.figure(figsize=(16,7))
fig = plt.figure(1)
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)

#Determing rolling statistics
rolLmean = ts_log_diff.rolling(12).mean()
rolLstd = ts_log_diff.rolling(12).std()



#Plot rolling statistics:
orig = plt.plot(ts_log_diff, color='blue',label='Original')
mean = plt.plot(rolLmean, color='red', label='Rolling Mean')
std = plt.plot(rolLstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```




```
Insight: this looks stationery,but there is possibility to create nan values as because there is no lag time
ithe begiining, so we need to drop during the plot
```

In [21]:

```
#Lets calculate ACF and PACF which will determinge AR component order and MA complonent order
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts_log_diff.dropna(),lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(ts_log_diff.dropna(),lags=40,ax=ax2)
```




```
Insight: when you run ARIMA model , you dont know , what order of ACF AND PACF good for your model, all the
highlited part i here is called confidence interval, the first line cross this chart , its your order, that
means 2 is order for ACF and " is oder for PACF (we dont knowis best fit or not). so PDQ is 202, where
dfference is 0

```

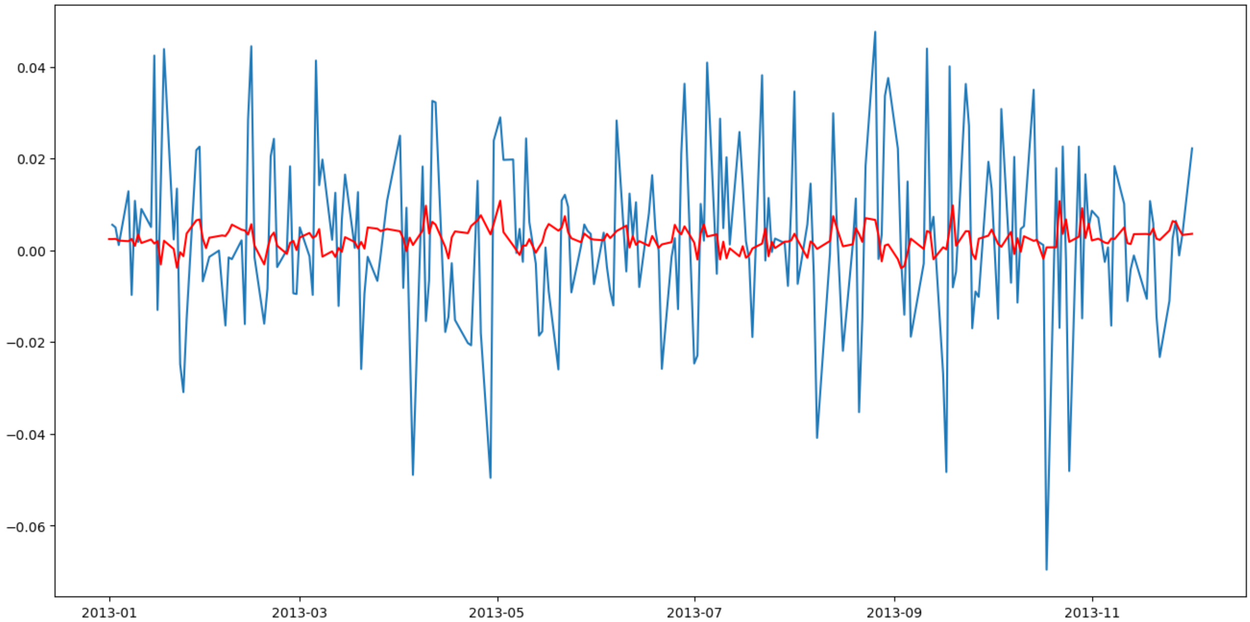
In [25]:

```
from statsmodels.tsa.arima.model import ARIMA
plt.figure(figsize=(16,8))
#ts_log_diff.dropna(inplace=True)
model = ARIMA(ts_log_diff, order=(2,0,2))
results_ARIMA = model.fit()
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
```

```
/Users/myyntiimac/anaconda3/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWa
rning: A date index has been provided, but it has no associated frequency information and so will be ig
nored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/myyntiimac/anaconda3/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWa
rning: A date index has been provided, but it has no associated frequency information and so will be ig
nored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/myyntiimac/anaconda3/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWa
rning: A date index has been provided, but it has no associated frequency information and so will be ig
nored when e.g. forecasting.
  self._init_dates(dates, freq)
```

Out[25]:

```
[<matplotlib.lines.Line2D at 0x146459350>]
```




```
insight: Red one is the fitted value that means , i predicting this model, light green one is actual value, our model fitted with data
```

In [ ]:

```
#result back to the original scale as we did two transformation(log transformation and differencing)

```

```
Create a Pandas Series named ARIMA_diff_predictions that contains the fitted values of an ARIMA model. These fitted values are obtained from the results_ARIMA object that presumably represents the result of fitting an ARIMA model to your time series data.

#These fitted values are essentially the model's predictions for the time series data after fitting the ARIMA model.

```

In [26]:

```
ARIMA_diff_predictions = pd.Series(results_ARIMA.fittedvalues, copy=True)
print(ARIMA_diff_predictions.head())
```

```
Date
2013-01-01    0.002435
2013-01-02    0.002435
2013-01-03    0.002482
2013-01-04    0.002108
2013-01-07    0.001973
dtype: float64
```

In [27]:

```
# calculating the cumulative sum of the fitted values obtained from an ARIMA model
#and storing the results in a new Pandas Series called ARIMA_diff_predictions_cumsum
#cumulative sum, you are essentially transforming the differentiated values (obtained by differencing the time s
#and assess how well the ARIMA model fits the data.
ARIMA_diff_predictions_cumsum = ARIMA_diff_predictions.cumsum()
print(ARIMA_diff_predictions_cumsum.head())

```

```
Date
2013-01-01    0.002435
2013-01-02    0.004870
2013-01-03    0.007351
2013-01-04    0.009459
2013-01-07    0.011432
dtype: float64
```

In [28]:

```
# Adding the comensent value of due to differencing to the ts_log dataframe
ARIMA_log_prediction = pd.Series(ts_log.iloc[0], index=ts_log.index)
ARIMA_log_prediction = ARIMA_log_prediction.add(ARIMA_diff_predictions_cumsum,fill_value=0)
ARIMA_log_prediction.head()
```

Out[28]:

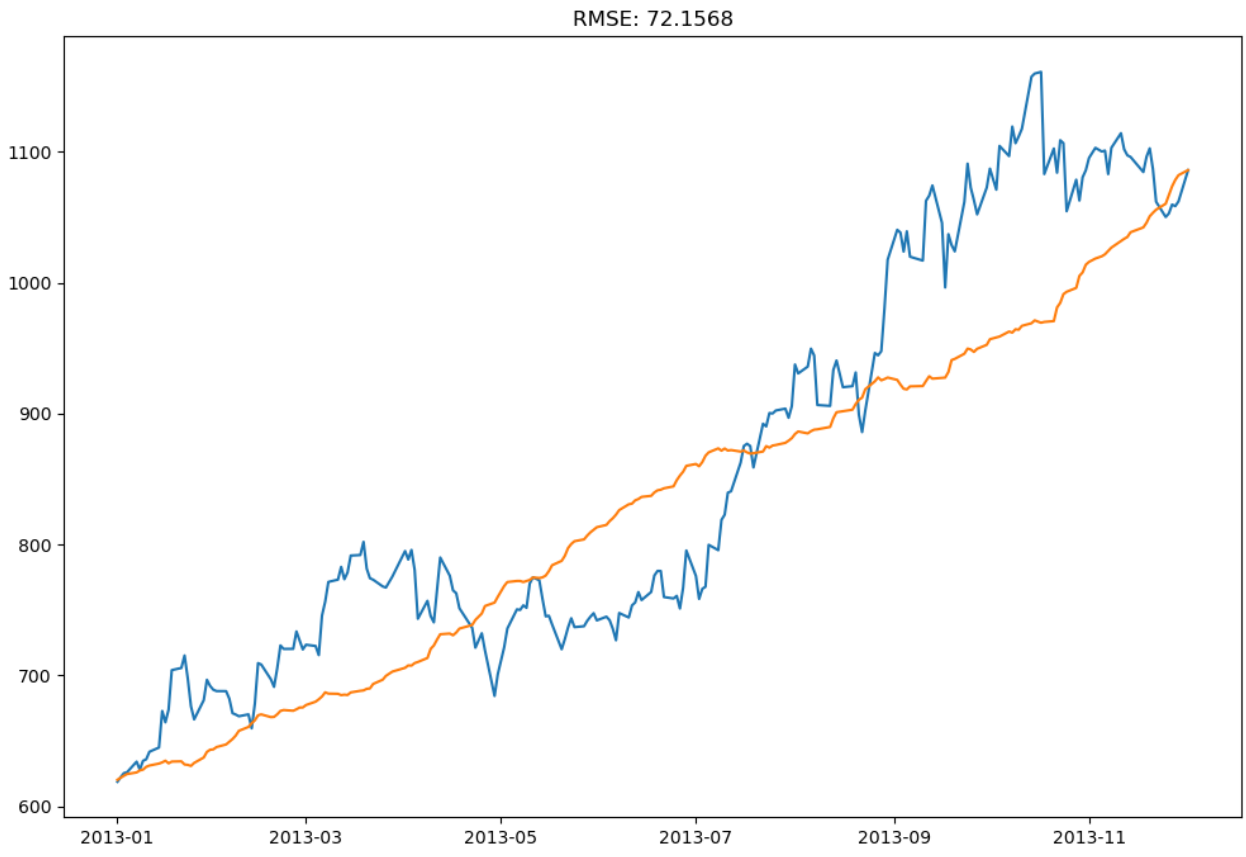
```
Date
2013-01-01    6.430055
2013-01-02    6.432490
2013-01-03    6.434972
2013-01-04    6.437080
2013-01-07    6.439053
dtype: float64
```

In [29]:

```
#now apply the log inverse function so that result back to orginal scale
plt.figure(figsize=(12,8))
predictions_ARIMA = np.exp(ARIMA_log_prediction)
plt.plot(df1)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-df1)**2)/len(df1)))
```

Out[29]:

```
Text(0.5, 1.0, 'RMSE: 72.1568')
```



```
Insight:yellow the predicted line for predicted value and blue line is for original value. we see our pedicted line capture the trend but its failed to capture the volume of up and down . how we improve it? we can improve by changing PDQ value and also try with AutoARImA model which give us best PDQ
```

In [30]:

```
#predicting index 15 to 25
results_ARIMA.predict(15,25)
```

Out[30]:

```
Date
2013-01-22   -0.003800
2013-01-23   -0.000428
2013-01-24   -0.001295
2013-01-25    0.003642
2013-01-28    0.006516
2013-01-29    0.006710
2013-01-30    0.002534
2013-01-31    0.000459
2013-02-01    0.002727
2013-02-04    0.003088
2013-02-05    0.003229
Name: predicted_mean, dtype: float64
```

In [ ]:

```

```