ediction-using-arima-end-to-end

April 16, 2024

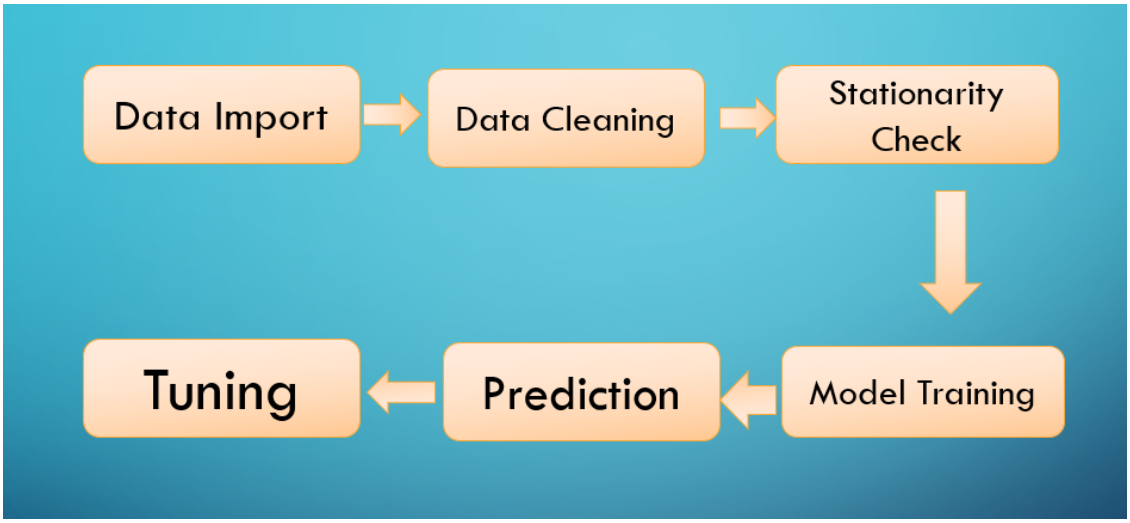
1 End to End Time Series forecasting using ARIMA

[25]:
```
#Changing working directory
import os
print(os.getcwd())
import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\amanr\Documents\Time Series
```

[2]:
```
from IPython.display import Image
Image(filename='Capture.png')
```

[2]:



[3]:
```
#Data Import and cleaning
```

[26]:
```
# Data and package Import
#Data Source - Kaggle - https://www.kaggle.com/rohanrao/
 ↪nifty50-stock-market-data
import pandas as pd
import matplotlib.pyplot as plt
```

```
from statsmodels.tsa.arima_model import ARMA
TempData = pd.read_csv('HCLTECH.csv')
TempData.head(30)
```

[26]:

	Date	Symbol	Series	Prev Close	Open	High	Low
0	2000-01-11	HCLTECH	EQ	580.00	1550.00	1725.00	1492.00
1	2000-01-12	HCLTECH	EQ	1554.45	1560.00	1678.85	1560.00
2	2000-01-13	HCLTECH	EQ	1678.85	1790.00	1813.20	1781.00
3	2000-01-14	HCLTECH	EQ	1813.20	1958.30	1958.30	1835.00
4	2000-01-17	HCLTECH	EQ	1958.30	2115.00	2115.00	1801.65
5	2000-01-18	HCLTECH	EQ	1801.65	1730.55	1815.00	1657.55
6	2000-01-19	HCLTECH	EQ	1774.50	1815.00	1889.00	1760.00
7	2000-01-20	HCLTECH	EQ	1851.15	1865.00	1865.00	1750.00
8	2000-01-21	HCLTECH	EQ	1757.85	1761.00	1815.00	1705.00
9	2000-01-24	HCLTECH	EQ	1781.35	1834.90	1923.90	1795.00
10	2000-01-25	HCLTECH	EQ	1923.90	1990.00	2077.85	1990.00
11	2000-01-27	HCLTECH	EQ	2077.85	2239.90	2239.90	1965.00
12	2000-01-28	HCLTECH	EQ	2080.60	2100.00	2245.00	2020.20
13	2000-01-31	HCLTECH	EQ	2196.95	2111.00	2200.00	2021.20
14	2000-02-01	HCLTECH	EQ	2030.40	1981.20	2180.00	1981.20
15	2000-02-02	HCLTECH	EQ	2162.25	2260.00	2260.00	2010.00
16	2000-02-03	HCLTECH	EQ	2024.25	2075.00	2075.00	1915.20
17	2000-02-04	HCLTECH	EQ	1956.50	1993.00	2064.00	1965.00
18	2000-02-07	HCLTECH	EQ	1986.60	1985.00	2119.95	1960.05
19	2000-02-08	HCLTECH	EQ	2062.95	2049.40	2060.00	1970.00
20	2000-02-09	HCLTECH	EQ	1999.20	2100.00	2100.00	1975.00
21	2000-02-10	HCLTECH	EQ	1999.00	1976.65	2014.80	1965.00
22	2000-02-11	HCLTECH	EQ	1996.75	1998.00	2156.50	1992.00
23	2000-02-14	HCLTECH	EQ	2156.50	2199.00	2329.05	2199.00
24	2000-02-15	HCLTECH	EQ	2324.70	2350.00	2510.70	2210.00
25	2000-02-16	HCLTECH	EQ	2508.40	2560.00	2560.00	2307.75
26	2000-02-17	HCLTECH	EQ	2423.10	2445.00	2616.95	2380.00
27	2000-02-18	HCLTECH	EQ	2614.70	2800.00	2823.90	2760.50
28	2000-02-21	HCLTECH	EQ	2775.95	2823.00	2998.00	2725.00
29	2000-02-22	HCLTECH	EQ	2819.15	2800.00	2875.00	2652.00

	Last	Close	VWAP	Volume	Turnover	Trades
0	1560.00	1554.45	1582.72	1192200	1.886915e+14	NaN
1	1678.85	1678.85	1657.05	344850	5.714349e+13	NaN
2	1813.20	1813.20	1804.69	53000	9.564880e+12	NaN
3	1958.30	1958.30	1939.90	270950	5.256169e+13	NaN
4	1801.65	1801.65	1990.55	428800	8.535473e+13	NaN
5	1775.00	1774.50	1716.39	359900	6.177280e+13	NaN
6	1842.80	1851.15	1842.81	316050	5.824204e+13	NaN
7	1753.50	1757.85	1801.37	204700	3.687409e+13	NaN
8	1786.00	1781.35	1774.01	282150	5.005360e+13	NaN
9	1923.90	1923.90	1875.34	328650	6.163317e+13	NaN

10	2077.85	2077.85	2065.71	313500	6.476010e+13	NaN
11	2078.00	2080.60	2101.92	352400	7.407150e+13	NaN
12	2190.00	2196.95	2173.00	360800	7.840197e+13	NaN
13	2021.20	2030.40	2083.45	217650	4.534620e+13	NaN
14	2160.00	2162.25	2110.66	261000	5.508834e+13	NaN
15	2035.00	2024.25	2085.91	326550	6.811542e+13	NaN
16	1965.00	1956.50	1989.66	175800	3.497821e+13	NaN
17	1977.00	1986.60	2008.25	176500	3.544560e+13	NaN
18	2036.00	2062.95	2048.51	277850	5.691791e+13	NaN
19	2014.00	1999.20	2017.09	199100	4.016027e+13	NaN
20	1991.00	1999.00	2011.50	180500	3.630763e+13	NaN
21	1990.00	1996.75	1992.36	76850	1.531132e+13	NaN
22	2156.50	2156.50	2110.61	166150	3.506781e+13	NaN
23	2329.05	2324.70	2293.27	265150	6.080618e+13	NaN
24	2500.00	2508.40	2425.21	224500	5.444586e+13	NaN
25	2418.00	2423.10	2411.48	260850	6.290343e+13	NaN
26	2616.95	2614.70	2526.71	276550	6.987609e+13	NaN
27	2771.00	2775.95	2807.40	173400	4.868032e+13	NaN
28	2805.00	2819.15	2883.95	151800	4.377830e+13	NaN
29	2710.00	2700.30	2772.79	88250	2.446984e+13	NaN

	Deliverable Volume	%Deliverble
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	NaN
9	NaN	NaN
10	NaN	NaN
11	NaN	NaN
12	NaN	NaN
13	NaN	NaN
14	NaN	NaN
15	NaN	NaN
16	NaN	NaN
17	NaN	NaN
18	NaN	NaN
19	NaN	NaN
20	NaN	NaN
21	NaN	NaN
22	NaN	NaN
23	NaN	NaN
24	NaN	NaN

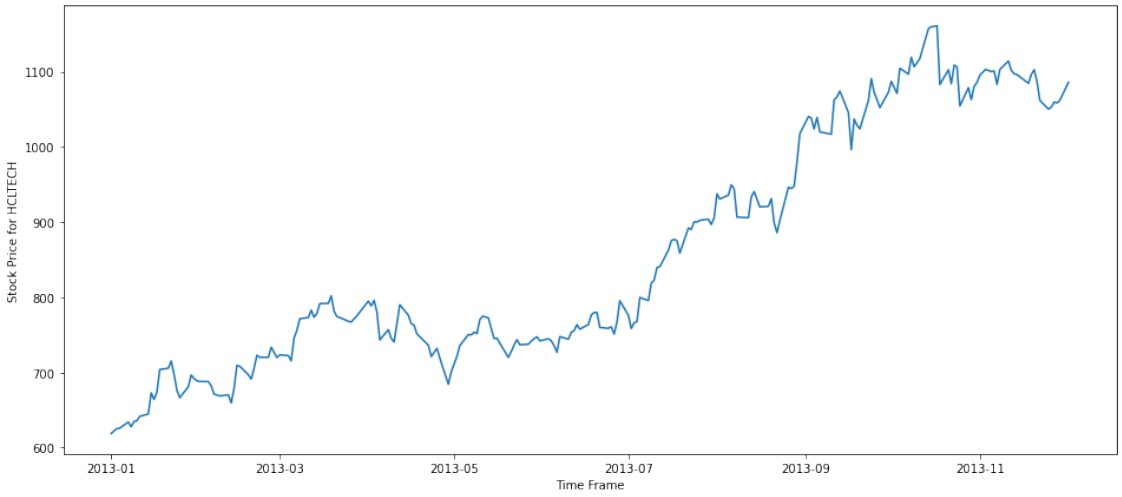
```
25                  NaN          NaN
26                  NaN          NaN
27                  NaN          NaN
28                  NaN          NaN
29                  NaN          NaN
```

[27]:
```
#Data Cleaning
HCLTechStockData = TempData.dropna()


HCLTechStockData.index = pd.to_datetime(HCLTechStockData.Date)

HCLTechStockData = HCLTechStockData["Prev Close"]['2013-01-01':'2013-12-2']
HCLTechStockData.describe()
```

[27]:
```
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
```

[28]:
```
#Data Exploration
plt.figure(figsize=(16,7))
fig = plt.figure(1)
ax1 = fig.add_subplot(111)
ax1.set_xlabel('Time Frame')
ax1.set_ylabel('Stock Price for HCLTECH')
ax1.plot(HCLTechStockData)
```

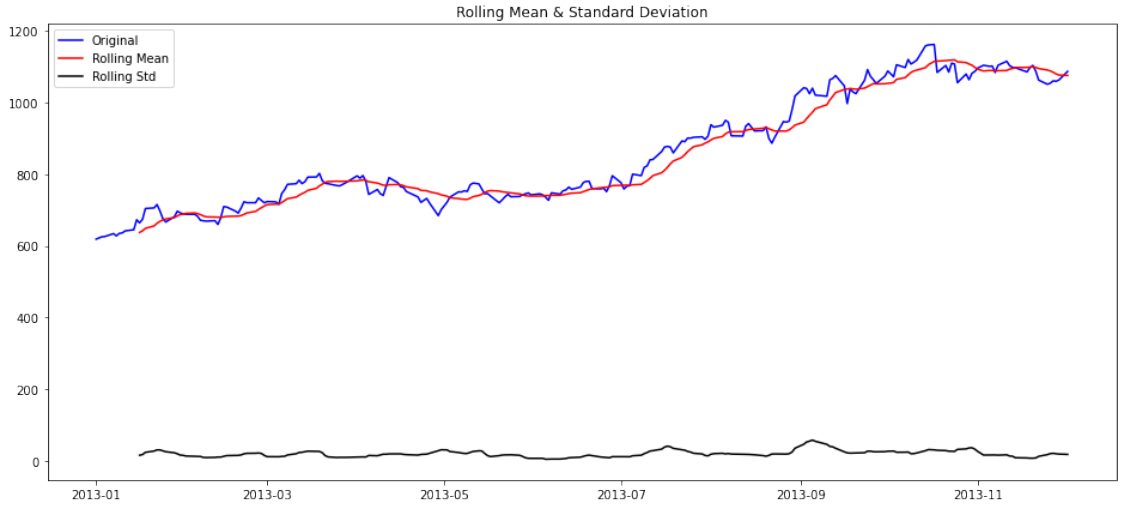
[28]:
```
[<matplotlib.lines.Line2D at 0x13d6eee79a0>]
```



[7]:
```
# Checking stationarity
```

[8]:
```
# Method 1 -  Rolling Statistics
# Method 2 -  Duckey fuller
```

[29]:
```
#Determing rolling statistics
rolLmean = HCLTechStockData.rolling(12).mean()
rolLstd = HCLTechStockData.rolling(12).std()

plt.figure(figsize=(16,7))
fig = plt.figure(1)

#Plot rolling statistics:
orig = plt.plot(HCLTechStockData, 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)
```



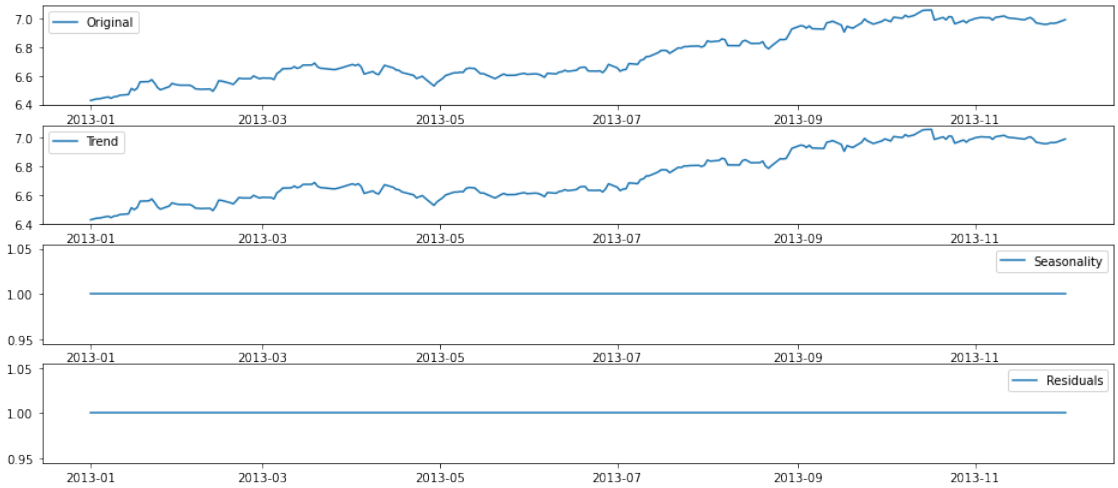
2 making Series Stationary

[30]:
```
#Lets try transformation
plt.figure(figsize=(16,7))
fig = plt.figure(1)

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

[30]: [<matplotlib.lines.Line2D at 0x13d6ee15fa0>]

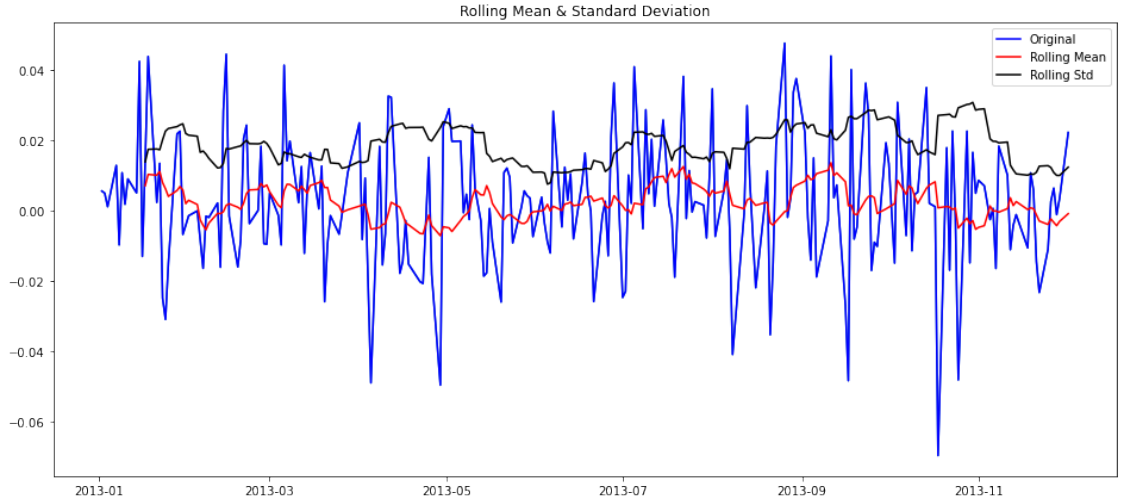
[31]:
```
#Decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_log,freq=1,model = 'multiplicative')

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(16,7))
fig = plt.figure(1)

plt.subplot(411)
plt.plot(ts_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
```

[31]:
```
<matplotlib.legend.Legend at 0x13d6f496d00>
```



[32]:
```
#Lets try differencing
plt.figure(figsize=(16,7))
fig = plt.figure(1)
ts_log_diff = ts_log - ts_log.shift()
```

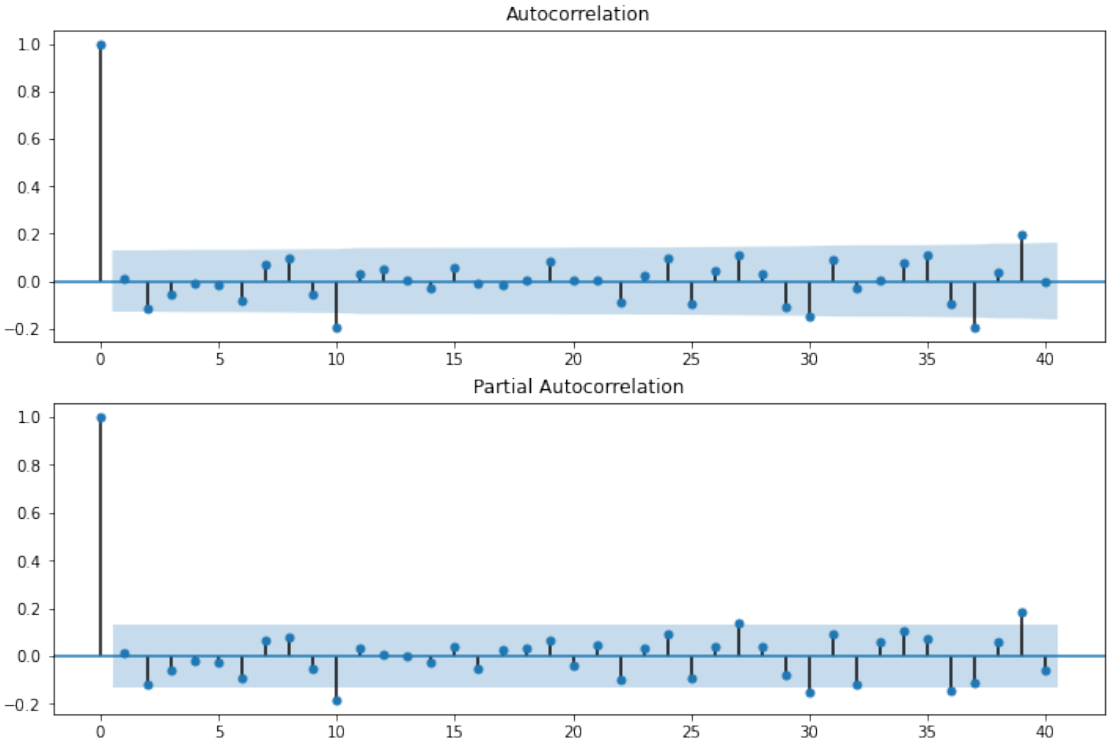
```
plt.plot(ts_log_diff)

#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)
```



[13]:
```
HCLTechStockData.sort_index(inplace= True)
```

[33]:
```
from statsmodels.tsa.stattools import acf, pacf
lag_acf = acf(ts_log_diff, nlags=20)
lag_pacf = pacf(ts_log_diff, nlags=20)
```

[34]:
```
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)
```



[16]:
```
from statsmodels.tsa.arima_model import ARIMA
```

[47]:
```
type(ts_log_diff)
```

[47]:
```
pandas.core.series.Series
```

[53]:
```
#ts_log_diff.dropna()
ts_log_diff = ts_log_diff[~ts_log_diff.isnull()]
```

[54]:
```
plt.figure(figsize=(16,8))
#ts_log_diff.dropna(inplace=True)
model = ARIMA(ts_log_diff, order=(2,1,2))
results_ARIMA = model.fit()
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-54-7d3fb1504929> in <module>
      2 #ts_log_diff.dropna(inplace=True)
      3 model = ARIMA(ts_log_diff, order=(2,1,2))
----> 4 results_ARIMA = model.fit()
      5 plt.plot(ts_log_diff)
```

```
      6 plt.plot(results_ARIMA.fittedvalues, color='red')

~\anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py in fit(self,␣
↪start_params, trend, method, transparams, solver, maxiter, full_output, disp,␣
↪callback, start_ar_lags, **kwargs)
   1195         r, order = 'F')
   1196         """
-> 1197         mlefit = super(ARIMA, self).fit(start_params, trend,

   1198                                         method, transparams, solver,
   1199                                         maxiter, full_output, disp,

~\anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py in fit(self,␣
↪start_params, trend, method, transparams, solver, maxiter, full_output, disp,␣
↪callback, start_ar_lags, **kwargs)
    994             kwargs.setdefault('m', 12)
    995             kwargs.setdefault('approx_grad', True)
--> 996         mlefit = super(ARMA, self).fit(start_params, method=solver,

    997                                        maxiter=maxiter,
    998                                        full_output=full_output,␣
↪disp=disp,

~\anaconda3\lib\site-packages\statsmodels\base\model.py in fit(self,␣
↪start_params, method, maxiter, full_output, disp, fargs, callback, retall,␣
↪skip_hessian, **kwargs)
    516         warn_convergence = kwargs.pop('warn_convergence', True)
    517         optimizer = Optimizer()
--> 518         xopt, retvals, optim_settings = optimizer._fit(f, score,␣
↪start_params,

    519                                                        fargs, kwargs,
    520                                                        hessian=hess,

~\anaconda3\lib\site-packages\statsmodels\base\optimizer.py in _fit(self,␣
↪objective, gradient, start_params, fargs, kwargs, hessian, method, maxiter,␣
↪full_output, disp, callback, retall)
    213
    214         func = fit_funcs[method]
--> 215         xopt, retvals = func(objective, gradient, start_params, fargs,␣
↪kwargs,

    216                              disp=disp, maxiter=maxiter,␣
↪callback=callback,
    217                              retall=retall, full_output=full_output,

~\anaconda3\lib\site-packages\statsmodels\base\optimizer.py in _fit_lbfgs(f,␣
↪score, start_params, fargs, kwargs, disp, maxiter, callback, retall,␣
↪full_output, hess)
    435         func = f
    436
```

```
--> 437     retvals = optimize.fmin_l_bfgs_b(func, start_params, maxiter=maxite
    438                                      callback=callback, args=fargs,
    439                                      bounds=bounds, disp=disp,

~\anaconda3\lib\site-packages\scipy\optimize\lbfgsb.py in fmin_l_bfgs_b(func,
 ↪x0, fprime, args, approx_grad, bounds, m, factr, pgtol, epsilon, iprint,
 ↪maxfun, maxiter, disp, callback, maxls)
    195             'maxls': maxls}
    196
--> 197     res = _minimize_lbfgsb(fun, x0, args=args, jac=jac, bounds=bounds,
    198                            **opts)
    199     d = {'grad': res['jac'],

~\anaconda3\lib\site-packages\scipy\optimize\lbfgsb.py in _minimize_lbfgsb(fun,
 ↪x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps, maxfun, maxiter, iprint
 ↪callback, maxls, finite_diff_rel_step, **unknown_options)
    358             # until the completion of the current minimization iteratio
    359             # Overwrite f and g:
--> 360             f, g = func_and_grad(x)
    361         elif task_str.startswith(b'NEW_X'):
    362             # new iteration

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in
 ↪fun_and_grad(self, x)
    198         if not np.array_equal(x, self.x):
    199             self._update_x_impl(x)
--> 200         self._update_fun()
    201         self._update_grad()
    202         return self.f, self.g

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in
 ↪_update_fun(self)
    164     def _update_fun(self):
    165         if not self.f_updated:
--> 166             self._update_fun_impl()
    167             self.f_updated = True
    168

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in
 ↪update_fun()
     71
     72         def update_fun():
---> 73             self.f = fun_wrapped(self.x)
     74
     75         self._update_fun_impl = update_fun
```

```
~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in␣
↪fun_wrapped(x)
     68         def fun_wrapped(x):
     69             self.nfev += 1
---> 70             return fun(x, *args)
     71
     72         def update_fun():

~\anaconda3\lib\site-packages\statsmodels\base\model.py in f(params, *args)
    498
    499         def f(params, *args):
--> 500             return -self.loglike(params, *args) / nobs
    501
    502         if method == 'newton':

~\anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py in loglike(self,␣
↪params, set_sigma2)
    808         method = self.method
    809         if method in ['mle', 'css-mle']:
--> 810             return self.loglike_kalman(params, set_sigma2)
    811         elif method == 'css':
    812             return self.loglike_css(params, set_sigma2)

~\anaconda3\lib\site-packages\statsmodels\tsa\arima_model.py in␣
↪loglike_kalman(self, params, set_sigma2)
    818         Compute exact loglikelihood for ARMA(p,q) model by the Kalman␣
↪Filter.
    819         """
--> 820         return KalmanFilter.loglike(params, self, set_sigma2)
    821
    822     def loglike_css(self, params, set_sigma2=True):

~\anaconda3\lib\site-packages\statsmodels\tsa\kalmanf\kalmanfilter.py in␣
↪loglike(cls, params, arma_model, set_sigma2)
    216          paramsdtype) = cls._init_kalman_state(params, arma_model)
    217         if np.issubdtype(paramsdtype, np.float64):
--> 218             loglike, sigma2 = kalman_loglike.kalman_loglike_double(
    219                 y, k, k_ar, k_ma, k_lags, int(nobs),
    220                 Z_mat, R_mat, T_mat)

statsmodels\tsa\kalmanf\kalman_loglike.pyx in statsmodels.tsa.kalmanf.
↪kalman_loglike.kalman_loglike_double()

statsmodels\tsa\kalmanf\kalman_loglike.pyx in statsmodels.tsa.kalmanf.
↪kalman_loglike.kalman_filter_double()

<__array_function__ internals> in pinv(*args, **kwargs)
```

```
~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in pinv(a, rcond, hermitian)
   1959         return wrap(res)
   1960     a = a.conjugate()
-> 1961     u, s, vt = svd(a, full_matrices=False, hermitian=hermitian)
   1962
   1963     # discard small singular values

<__array_function__ internals> in svd(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in svd(a, full_matrices,␣
 ↪compute_uv, hermitian)
   1624
   1625         signature = 'D->DdD' if isComplexType(t) else 'd->ddd'
-> 1626         u, s, vh = gufunc(a, signature=signature, extobj=extobj)
   1627         u = u.astype(result_t, copy=False)
   1628         s = s.astype(_realType(result_t), copy=False)

~\anaconda3\lib\site-packages\numpy\linalg\linalg.py in␣
 ↪_raise_linalgerror_svd_nonconvergence(err, flag)
    104
    105 def _raise_linalgerror_svd_nonconvergence(err, flag):
--> 106     raise LinAlgError("SVD did not converge")
    107
    108 def _raise_linalgerror_lstsq(err, flag):

LinAlgError: SVD did not converge
```

```
<Figure size 1152x576 with 0 Axes>
```

3 Taking results back to original scale

[40]:
```
ARIMA_diff_predictions = pd.Series(results_ARIMA.fittedvalues, copy=True)
print(ARIMA_diff_predictions.head())
```

```
Date
2013-01-02    0.002432
2013-01-03    0.002471
2013-01-04    0.002117
2013-01-07    0.001981
2013-01-08    0.002377
dtype: float64
```

[41]:
```
ARIMA_diff_predictions_cumsum = ARIMA_diff_predictions.cumsum()
print(ARIMA_diff_predictions_cumsum.head())
```

```
Date
```

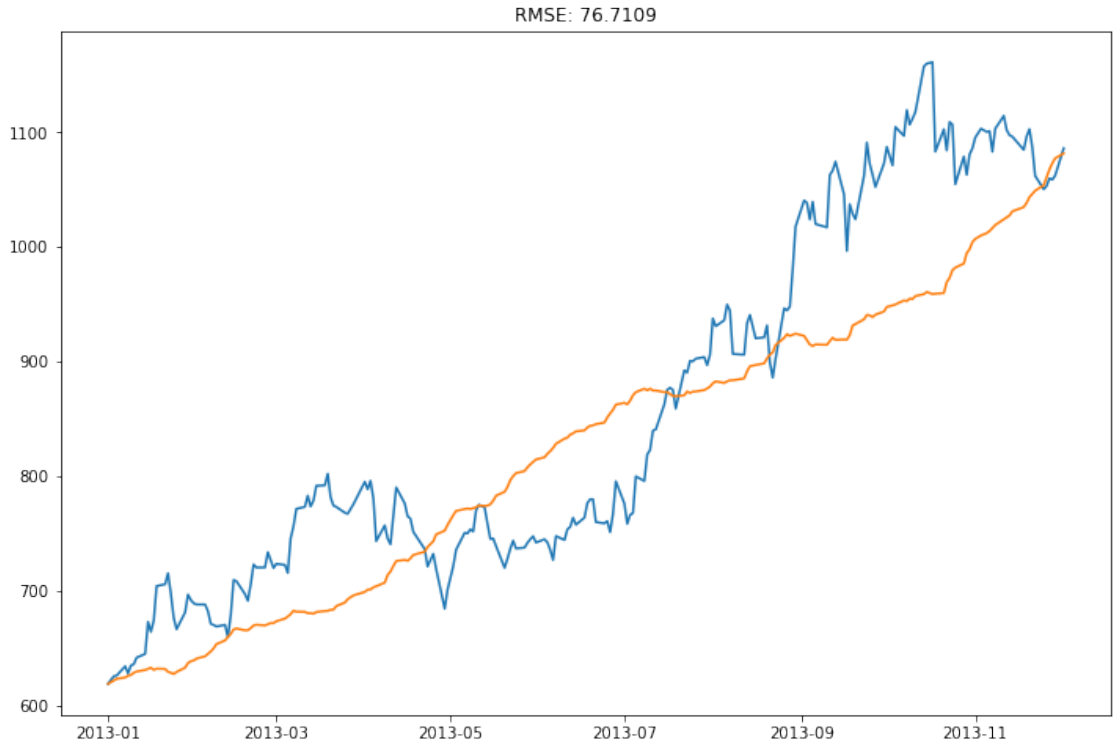
```
2013-01-02    0.002432
2013-01-03    0.004903
2013-01-04    0.007021
2013-01-07    0.009002
2013-01-08    0.011379
dtype: float64
```

[21]:
```
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()
```

[21]:
```
Date
2013-01-01    6.427621
2013-01-02    6.430053
2013-01-03    6.432524
2013-01-04    6.434641
2013-01-07    6.436622
dtype: float64
```

[42]:
```
plt.figure(figsize=(12,8))
predictions_ARIMA = np.exp(ARIMA_log_prediction)
plt.plot(HCLTechStockData)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-HCLTechStockData)**2)/
 ↪len(HCLTechStockData)))
```

[42]:
```
Text(0.5, 1.0, 'RMSE: 76.7109')
```



[43]:
```
results_ARIMA.predict(10,20)
```

[43]:
```
Date
2013-01-15    0.001398
2013-01-16    0.001704
2013-01-17   -0.002997
2013-01-18    0.001599
2013-01-21   -0.000306
2013-01-22   -0.004052
2013-01-23   -0.001140
2013-01-24   -0.001885
2013-01-25    0.002912
2013-01-28    0.005861
2013-01-29    0.006330
dtype: float64
```

[24]:
```
import pmdarima as pm
def arimamodel(timeseries):
    automodel = pm.auto_arima(timeseries,
                              start_p=3,
                              start_q=3,
                              max_p=5,
                              max_q=5,
```

```
                                test="adf",
                                seasonal=True,
                                trace=True)
    return automodel
```

[53]:
```
arimamodel(ts_log)
```

```
Performing stepwise search to minimize aic
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-1158.020, Time=0.38 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1167.424, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-1165.456, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-1165.465, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-1165.525, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-1163.485, Time=0.09 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.629 seconds
```

[53]:
```
ARIMA(order=(0, 1, 0), scoring_args={}, suppress_warnings=True)
```

[ ]:
```

```