Time Series Forecasting_Predicting Sales using ARIMA

In [1]:

```
#https://www.youtube.com/watch?v=MVsKaYzEggY&list=PLmPJQXJiMoUVr07-VnwDiki89Dqyu
#Changing working directory
import os
print(os.getcwd())
import warnings
warnings.filterwarnings('ignore')
```

```
/Users/pragatigupta/Documents/AI And ML/2023/Time Series/Time Series forcasting
sales
```

In [3]:

```
from IPython.display import Image
Image(filename='/Users/pragatigupta/Documents/AI And ML/2023/Time Series/Time Se
```

Out[3]:

Problem: Forecasting Sales

Step1:

Gathered (Extract) Data from all the sources

In [368…

```
# Data and package Import
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARMA
TempData = pd.read_csv('/Users/pragatigupta/Documents/AI And ML/2023/Time Series
TempData.head(5)
```

Out [368...

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume
0	2000-01-11	HCLTECH	EQ	580.00	1550.0	1725.00	1492.00	1560.00	1554.45	1582.72	1192200
1	2000-01-12	HCLTECH	EQ	1554.45	1560.0	1678.85	1560.00	1678.85	1678.85	1657.05	344850
2	2000-01-13	HCLTECH	EQ	1678.85	1790.0	1813.20	1781.00	1813.20	1813.20	1804.69	53000
3	2000-01-14	HCLTECH	EQ	1813.20	1958.3	1958.30	1835.00	1958.30	1958.30	1939.90	270950
4	2000-01-17	HCLTECH	EQ	1958.30	2115.0	2115.00	1801.65	1801.65	1801.65	1990.55	428800

Step2:Data Preprocessing:

Clean and preprocess the historical sales data, handling missing values and outliers if necessary. Explore the data visually to identify any apparent trends, seasonality, or irregular patterns.

In [369...

```
#Data Cleaning
Sales_Data = TempData.dropna()
```

In [370...

```
Sales_Data=Sales_Data.drop_duplicates(subset=['Date'])
```

In [371...

```
Sales_Data.index = pd.to_datetime(Sales_Data.Date)
```

In [372...

```
print(Sales_Data.index.min())
print(Sales_Data.index.max())
```

```
2011-06-01 00:00:00
2020-11-27 00:00:00
```

In [373...

```
Sales_Data = Sales_Data["Prev Close"]['2013-01-01':'2013-12-2'] # select the col
#or other method is to drop unwanted columns
#Sales_Data = Sales_Data.drop(['col1','col2'], axis=1)
Sales_Data.describe()
```

Out[373...

```
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 [374…

```
Sales_Data.head(10)
```

Out[374…

```
Date
2013-01-01    618.70
2013-01-02    622.15
2013-01-03    625.25
2013-01-04    625.95
2013-01-07    634.05
2013-01-08    627.90
2013-01-09    634.70
2013-01-10    635.85
2013-01-11    641.60
2013-01-14    644.85
Name: Prev Close, dtype: float64
```

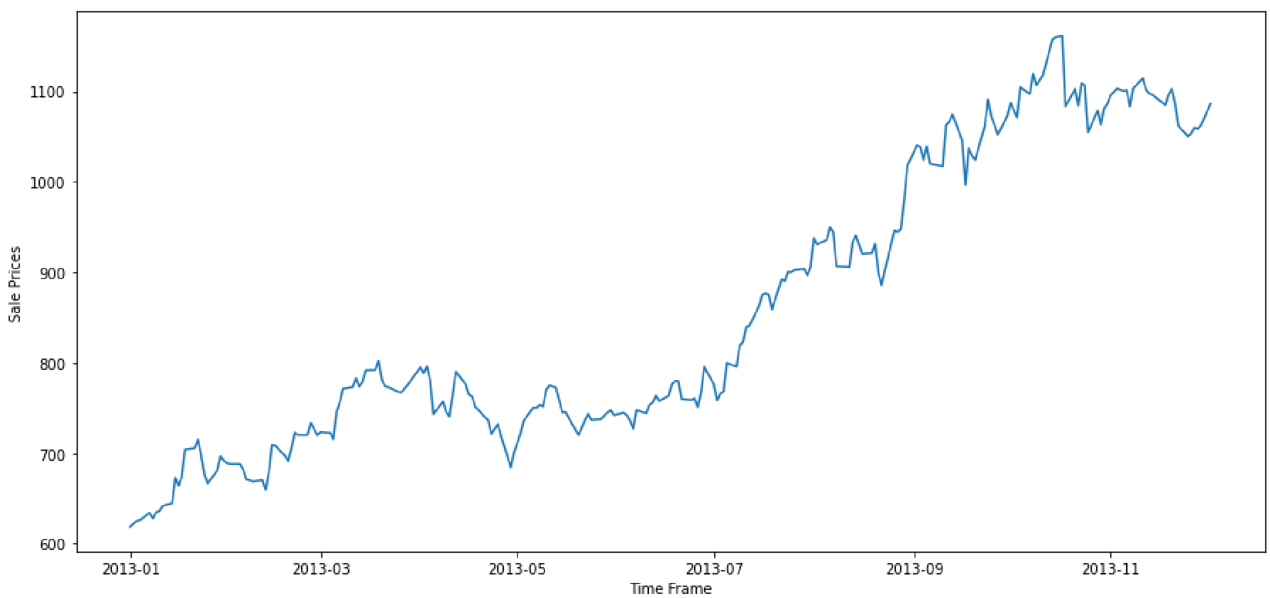
Step 3:Data Exploration

In [375…

```
#Data Exploration
plt.figure(figsize=(15,7))
fig = plt.figure(1)
ax1 = fig.add_subplot(111)
ax1.set_xlabel('Time Frame')
ax1.set_ylabel('Sale Prices')
ax1.plot(Sales_Data)
```

Out[375…

```
[<matplotlib.lines.Line2D at 0x7ff51aea45b0>]
```



Step4: Feature Engineering:

Create additional features based on external factors like marketing spend, seasonal indicators, and economic variables. Lag features: Include lagged sales data to capture autocorrelation.

In [376…

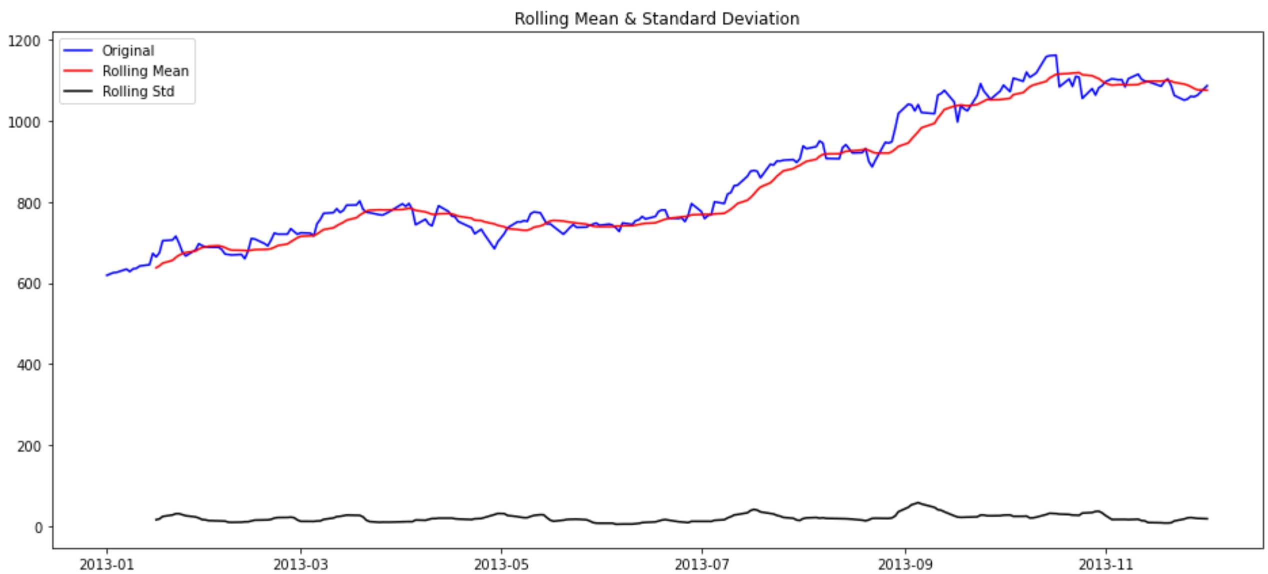
```
# Checking stationarity
```

In [377…

```
# Method 1 -  Rolling Statistics
# Method 2 -  Duckey fuller
```

In [378…

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

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

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



Making Series Stationary

In [379…

```
#Lets try transformation
plt.figure(figsize=(16,7))
fig = plt.figure(1)
#log , square root transformation ect to make the series stationary
import numpy as np
ts_log = np.log(Sales_Data)
plt.plot(ts_log)
```

Out[379…

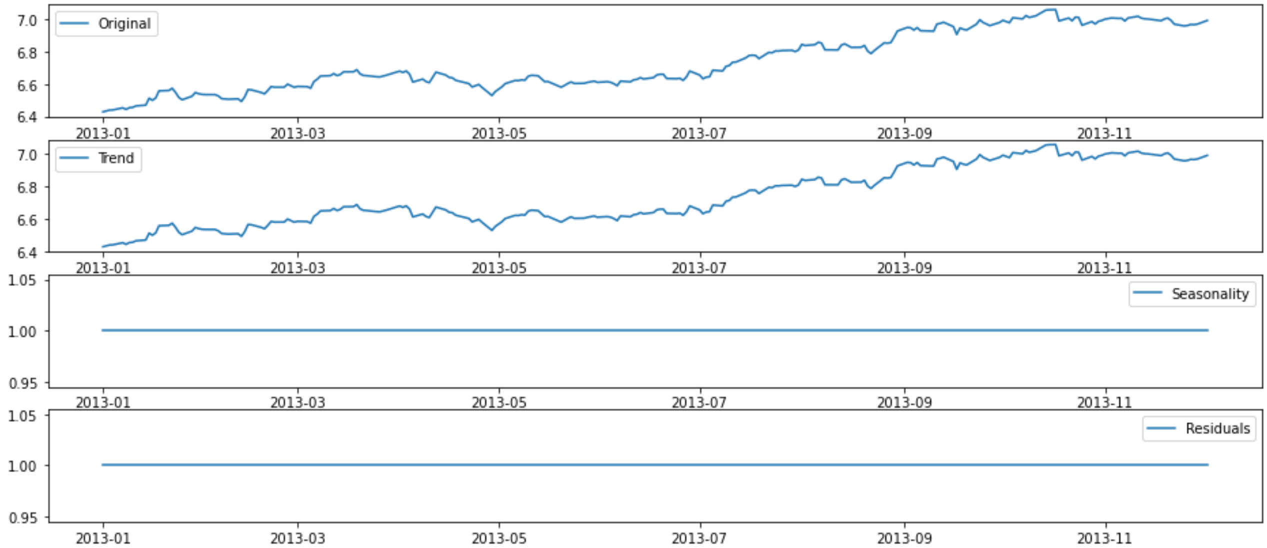
```
[<matplotlib.lines.Line2D at 0x7ff53504ad60>]
```

In [380…

```
#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')
```

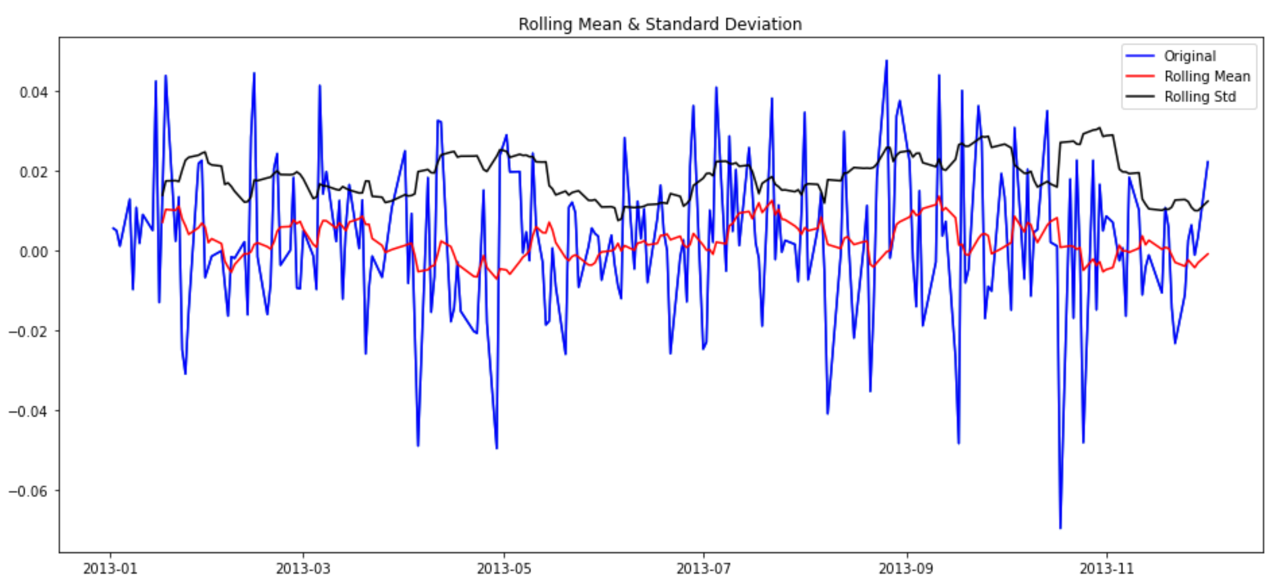
Out[380…

```
<matplotlib.legend.Legend at 0x7ff57252d0a0>
```



In [381…

```
#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)
```



In [382…

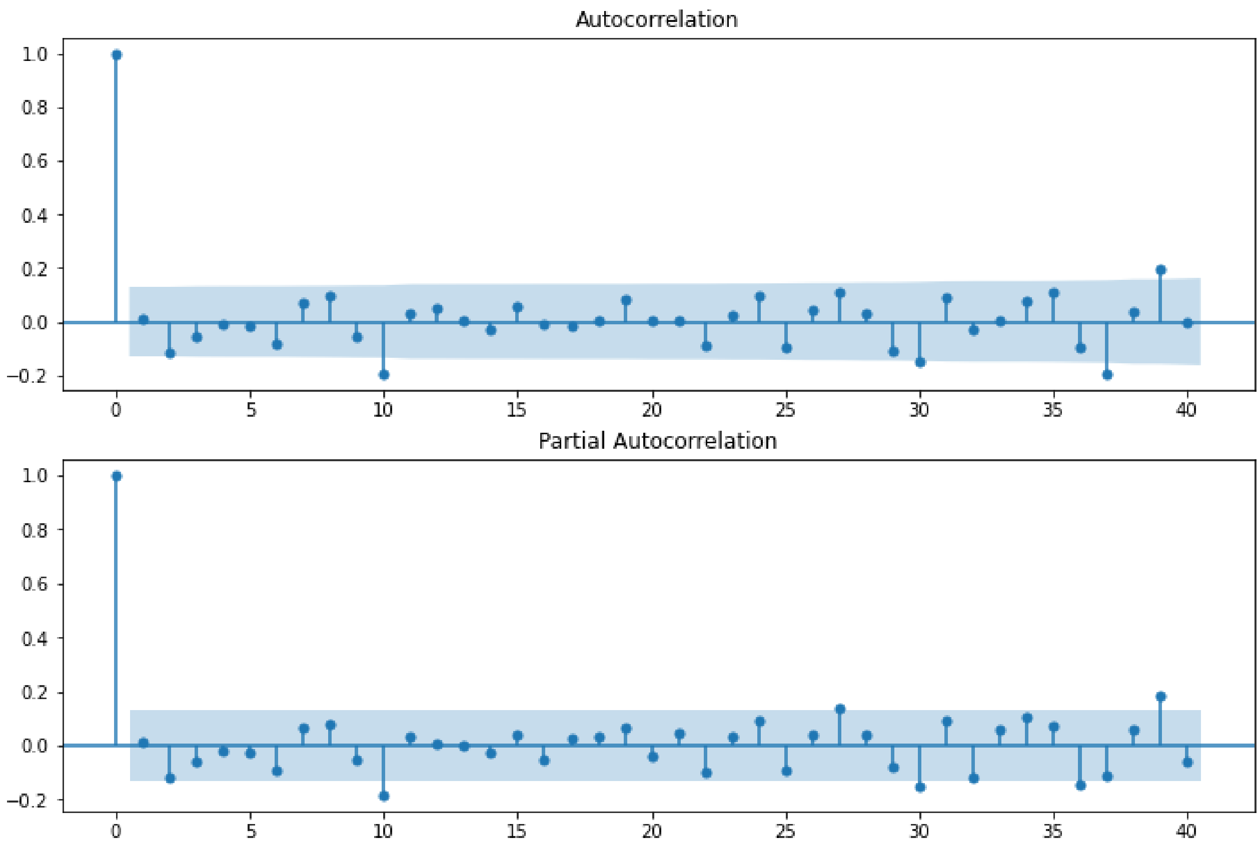
```
Sales_Data.sort_index(inplace= True)
```

In [389…

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

In [392…

```
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)
#Chart: helps to get correct AR and MA values
# highlighted part = confidence interval
# first line which touch or cross the highlighted part is the correct value
# from the fig AR=2, MA=2
```



In [393…

```
from statsmodels.tsa.arima_model import ARIMA
```

In [394…

```
type(ts_log_diff)
```

Out[394…

```
pandas.core.series.Series
```

In [395…

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

In [396…

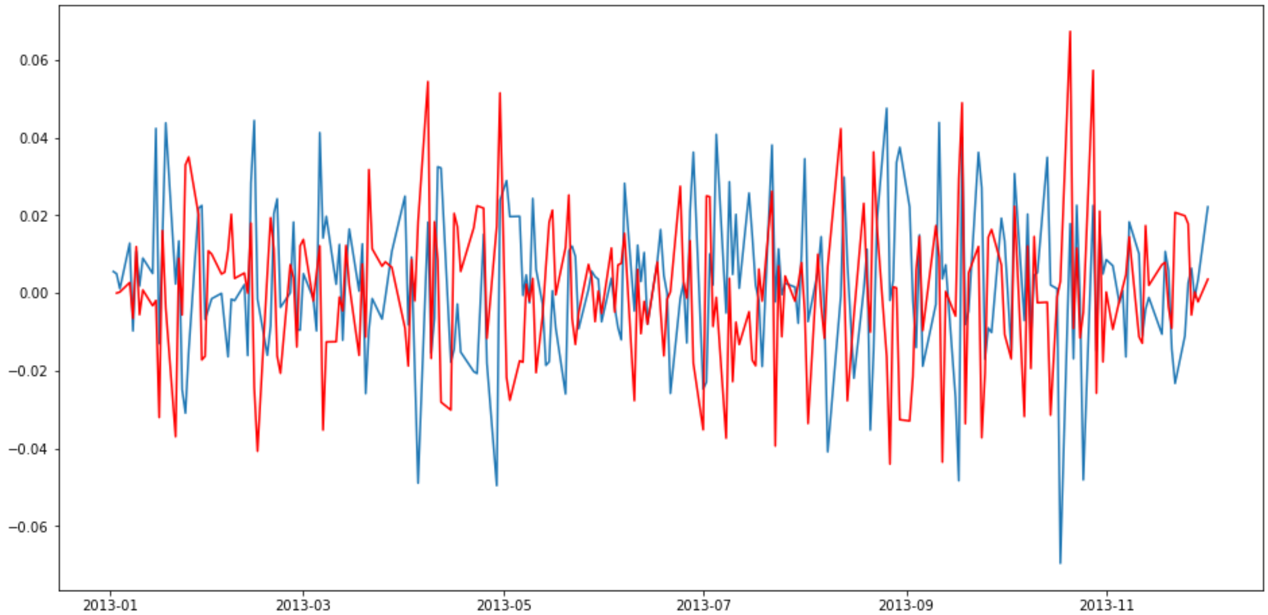
```
plt.figure(figsize=(16,8))
#ts_log_diff.dropna(inplace=True)
model = ARIMA(ts_log_diff, order=(2,1,2))
results_ARIMA = model.fit()
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
```

```
/Users/pragatigupta/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/ba
se/tsa_model.py:581: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
/Users/pragatigupta/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/ba
se/tsa_model.py:581: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
 This problem is unconstrained.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f= -2.55431D+00    |proj g|=  2.26589D+00

At iterate    5    f= -2.55431D+00    |proj g|=  4.08178D-01

At iterate   10    f= -2.55432D+00    |proj g|=  6.35795D+00

At iterate   15    f= -2.55449D+00    |proj g|=  5.80723D+01

At iterate   20    f= -2.55492D+00    |proj g|=  3.62180D-02

At iterate   25    f= -2.55492D+00    |proj g|=  5.25044D-01

At iterate   30    f= -2.55493D+00    |proj g|=  7.10730D+00

At iterate   35    f= -2.55514D+00    |proj g|=  2.78026D+01

At iterate   40    f= -2.55518D+00    |proj g|=  2.52323D-03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     41     59      1     0     0   2.523D-03  -2.555D+00
  F =  -2.5551832218989681
```

```
CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
/Users/pragatigupta/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/m
odel.py:547: HessianInversionWarning: Inverting hessian failed, no bse or cov_pa
rams available
  warnings.warn('Inverting hessian failed, no bse or cov_params '
```

Out[396…

```
[<matplotlib.lines.Line2D at 0x7ff532ea6070>]
```



Taking results back to original scale

In [397…

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

```
Date
2013-01-03   -0.000005
2013-01-04    0.000273
2013-01-07    0.002712
2013-01-08   -0.006513
2013-01-09    0.011963
dtype: float64
```

In [398…

```
ARIMA_diff_predictions_cumsum = ARIMA_diff_predictions.cumsum()
print(ARIMA_diff_predictions_cumsum.head())
# reverse the log : cumulatevive sum to diffrence the time series
```

```
Date
2013-01-03   -0.000005
2013-01-04    0.000268
2013-01-07    0.002980
2013-01-08   -0.003533
2013-01-09    0.008430
dtype: float64
```

In [399…

```
ARIMA_log_prediction = pd.Series(ts_log.iloc[0], index=ts_log.index)
ARIMA_log_prediction = ARIMA_log_prediction.add(ARIMA_diff_predictions_cumsum,fi
```

```
ARIMA_log_prediction.head()
# reverse the  diffrese
```

Out[399…

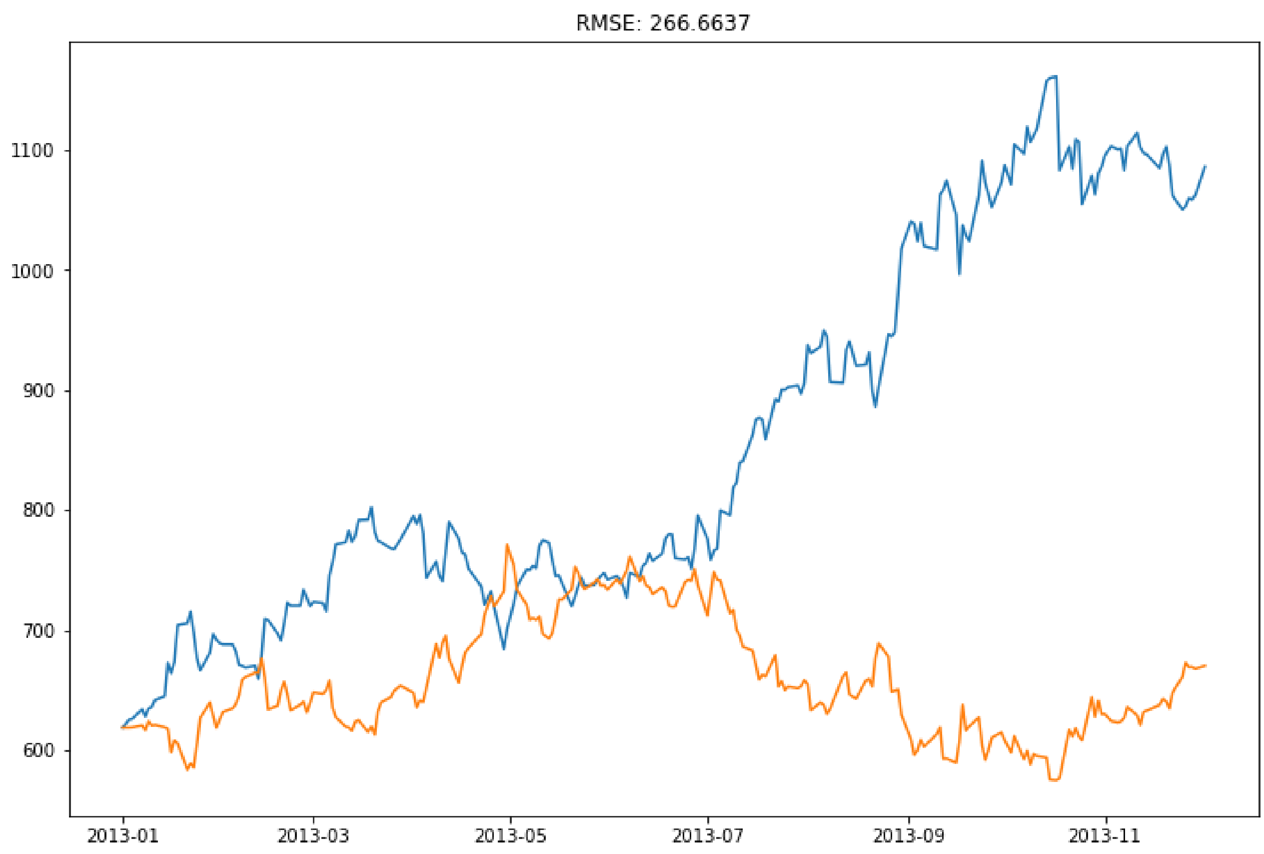
```
Date
2013-01-01    6.427621
2013-01-02    6.427621
2013-01-03    6.427615
2013-01-04    6.427888
2013-01-07    6.430600
dtype: float64
```

In [400…

```
plt.figure(figsize=(12,8))
predictions_ARIMA = np.exp(ARIMA_log_prediction)
plt.plot(Sales_Data)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-Sales_Data)**2)/len(Sales
# reverse the log: expontital
```

Out[400…

```
Text(0.5, 1.0, 'RMSE: 266.6637')
```



In [401…

```
results_ARIMA.predict(10,20)
```

Out[401…

```
Date
2013-01-16   -0.032051
2013-01-17    0.016081
2013-01-18   -0.003840
2013-01-21   -0.037005
2013-01-22    0.008956
2013-01-23   -0.005627
2013-01-24    0.033022
2013-01-25    0.035053
2013-01-28    0.020183
```

```
2013-01-29   -0.017208
2013-01-30   -0.016286
dtype: float64
```

In [402…

```
#Auto ARIMA: pip install pmdarima on command window
```

In [403…

```
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
```

In [404…

```
arimamodel(ts_log)
```

```
Performing stepwise search to minimize aic
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-1159.043, Time=0.31 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1167.424, Time=0.02 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=0.04 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=0.08 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.518 seconds
```

Out[404…

```
ARIMA(order=(0, 1, 0), scoring_args={}, suppress_warnings=True)
```

In [411…

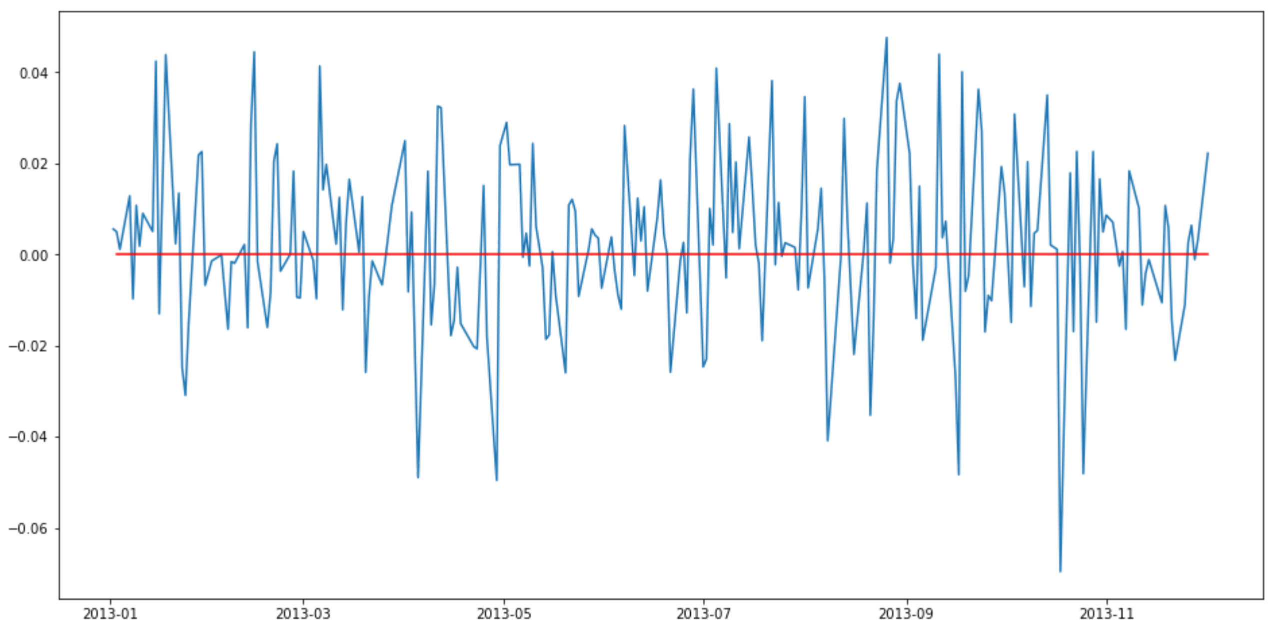
```
plt.figure(figsize=(16,8))
#ts_log_diff.dropna(inplace=True)
model = ARIMA(ts_log_diff, order=(0,1,0))
results_ARIMA = model.fit()
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            1     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f= -2.21608D+00    |proj g|=  7.19425D-06

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
```

```
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    1     1      14      1     0     0   6.839D-06  -2.216D+00
  F =  -2.2160806223129810

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
/Users/pragatigupta/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:581: ValueWarning: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
/Users/pragatigupta/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.py:581: ValueWarning: A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
 This problem is unconstrained.

 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
```

Out[411…

```
[<matplotlib.lines.Line2D at 0x7ff57151da90>]
```



In [412…

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

```
Date
2013-01-03    0.000073
2013-01-04    0.000073
2013-01-07    0.000073
2013-01-08    0.000073
2013-01-09    0.000073
dtype: float64
```

In [413…

```
ARIMA_diff_predictions_cumsum = ARIMA_diff_predictions.cumsum()
print(ARIMA_diff_predictions_cumsum.head())
# reverse the log : cumulatevive sum to diffrence the time series
```

```
Date
2013-01-03    0.000073
2013-01-04    0.000146
2013-01-07    0.000219
2013-01-08    0.000292
2013-01-09    0.000365
dtype: float64
```

In [414…

```
ARIMA_log_prediction = pd.Series(ts_log.iloc[0], index=ts_log.index)
ARIMA_log_prediction = ARIMA_log_prediction.add(ARIMA_diff_predictions_cumsum,fi
ARIMA_log_prediction.head()
# reverse the  diffrese
```

Out[414…

```
Date
2013-01-01    6.427621
2013-01-02    6.427621
2013-01-03    6.427694
2013-01-04    6.427767
2013-01-07    6.427840
dtype: float64
```

In [415…

```
plt.figure(figsize=(12,8))
predictions_ARIMA = np.exp(ARIMA_log_prediction)
plt.plot(Sales_Data)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-Sales_Data)**2)/len(Sales
# reverse the log: expontital
```

Out[415…

```
Text(0.5, 1.0, 'RMSE: 275.6730')
```


RMSE: 275.6730
1100
1000
900
800
700
600
2013-01
2013-03
2013-05
2013-07
2013-09
2013-11

In [ ]:

In [ ]: