Stock Analysis using-ARIMA model

Ananth Subramanian, Vitaly Briker, Jackson Au & Richard Farrow

October 20, 2019

1. Introduction

Performance of company stock, sales figures for a particular product, or ice melting trends in Tanana river are just a few examples of time series examples in the world of data we face today. Within a time series problem, the task is to fit data across a period of time using pertinent statistical methods. One of the most common techniques used for fitting and predicting time series is known as the Autoregression Integrated Moving Average or ARIMA for short.

Autoregression because the model uses the dependent relationship between an observation and some number of lagged observations. Integrated because differencing observations achieves time series stationary.

Moving Average, as the dependency between an observation and a residual error from a moving average model is applied and in turn is applied to lagged observations.

The parameters involved within ARIMA are referenced as p, d and q. All these parameters should be integers.

- p : Refers to the number of lag observations included in the model
- d: The number of times the observations are differenced
- q: Refers to the size of the moving average window

To perform time series predictions, one must first ensure that the data displays stationarity, or does not contain indications of trends. Given that trends often exist in the data, statisticians perform operations to remove trends or seasonality by differencing the data.

Stationarity can be validated using the Dickey-Fuller Test; the test statistic should be less than the critical values. There are several rules related to parameters p and q and the rule that stands out is to ensure that they are not greater than 1. The case study involves fitting the stock prices of “Amazon” using the ARIMAS methodology.

2. Methods

Python is used for the time series analysis and necessary libraries include among others are datareader, matplotlib, datetime, adfuller, seasonal_decompose, acf and pacf. API’s are used to read monthly “Amazon” stock price for four years in. between Jan 2014 and Dec 2018.

Step 1: Bring stationarity to the model. The test statistic is one of the outputs from Dickey Fuller Test. The rule is such that the test statistic should be less than the critical values reported. If the condition is not satisfied, the Dickey Fuller Test should be repeated by using the differenced observations with the moving average.

Step 2: In order to get better results, logarithmic values are used.

Step 3: Next steps involves plotting autocorrelation and partial autocorrelation graphs. This helps in arriving at the values of “p” (AR) and “q” (MA).

Step 4: The residual plot is generated after importing ARIMA library and passing the values of p, q and d. The outcome should be normally distributed. If not the values of p and q are adjusted. The resultant model is the one that better fits the time series.

Step 5: Grid search is also used to arrive at the best parameters.

3. Results

The data loaded using “pandas_datareader” package to import 4 years of Amazon stock data, at closing, from 1/1/2014 to 12/31/2018. The data is monthly averaged presented in Figure1.

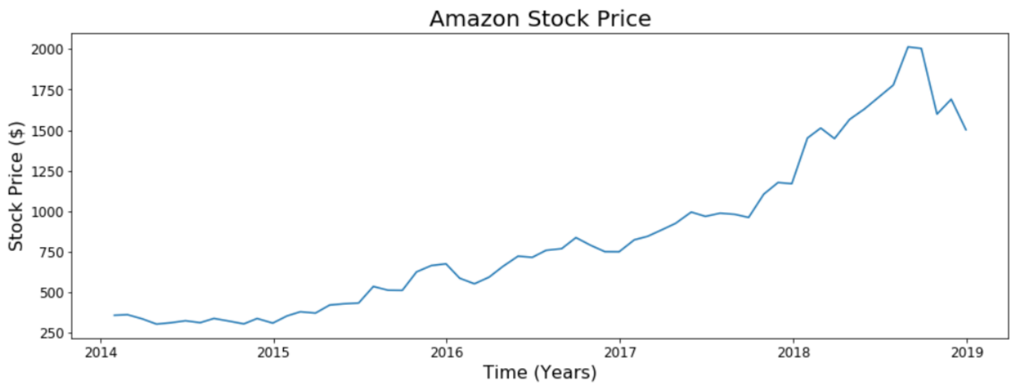
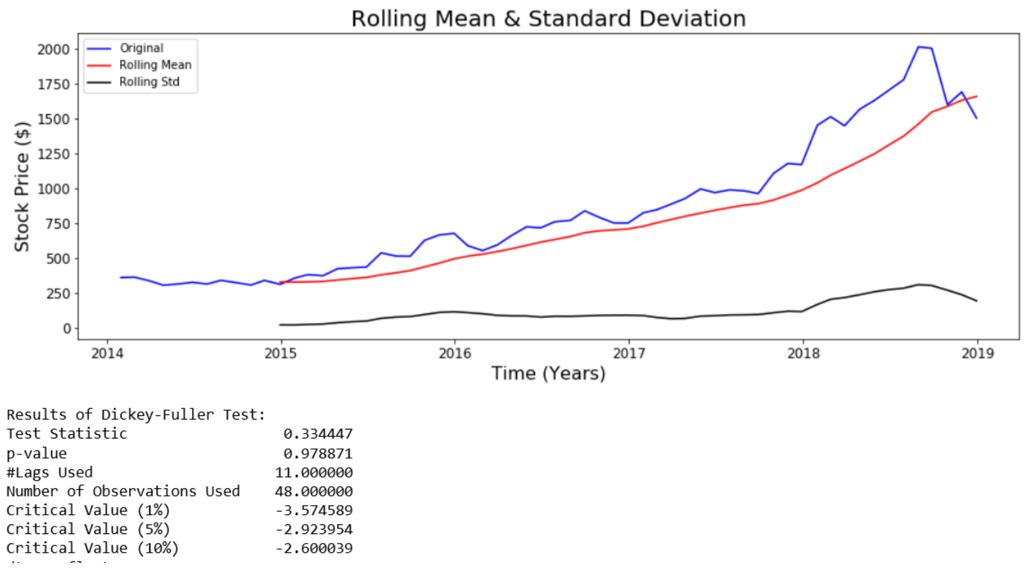


FIGURE 1 – Amazon stock price trend

The Figure 2 shows the stock trend with rolling mean and standard deviation. Visually observing the plot there is evidence against stationarity since mean and standard deviation look depending on time and violates conditions of stationarity. The result of Dickey-Fuller Test indicates that stock data is not stationary, the test statistics is greater the critical value



```
Results of Dickey-Fuller Test:
Test Statistic                  0.334447
p-value                         0.978871
#Lags Used                     11.000000
Number of Observations Used    48.000000
Critical Value (1%)            -3.574589
Critical Value (5%)            -2.923954
Critical Value (10%)           -2.600039
```

FIGURE 2 – Dickey-Fuller Test result with original data

On Figured 3 we present time series decomposition into several components that represent underlying patterns. By removing trend and seasonality from the data we expected to receive small residuals in order to fit the model.

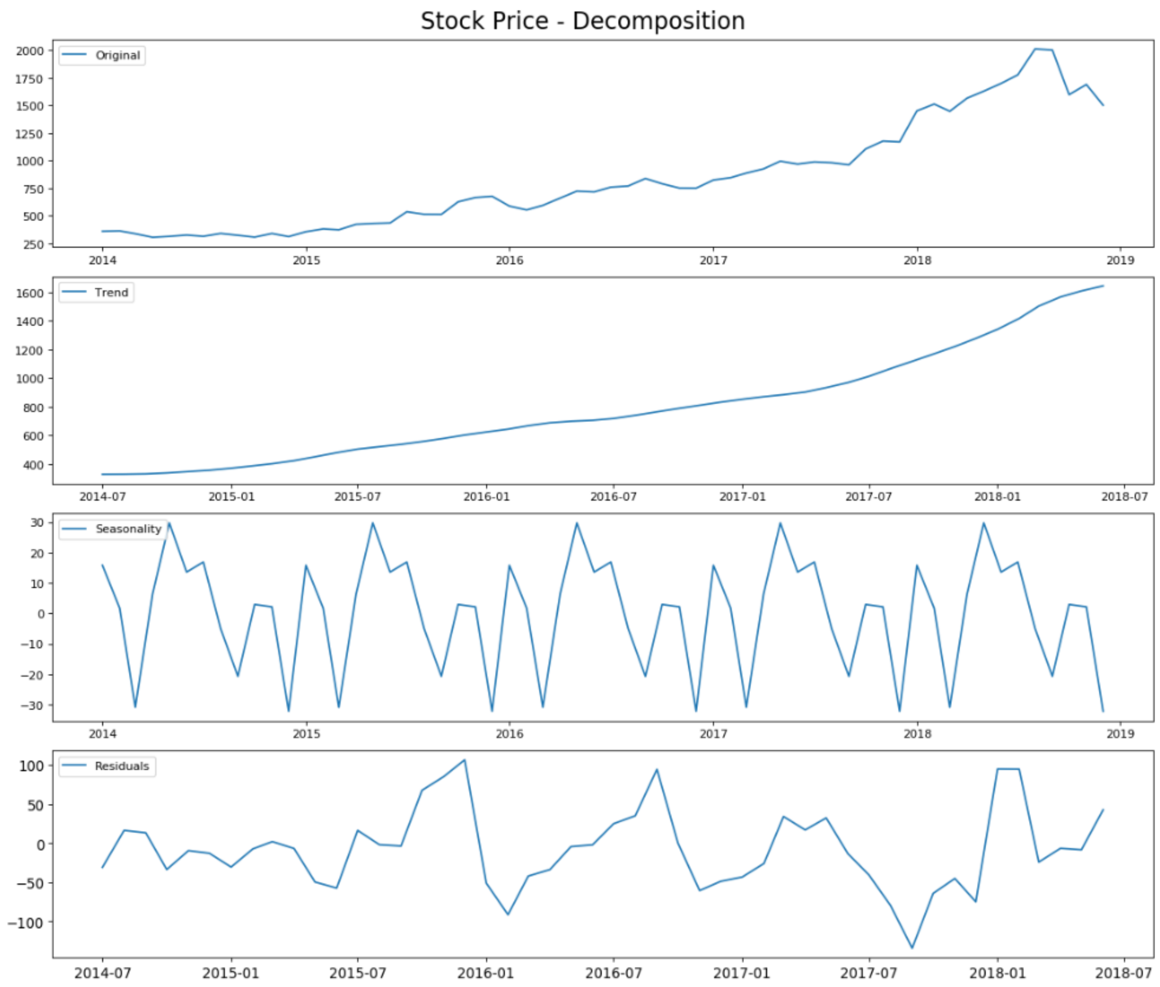
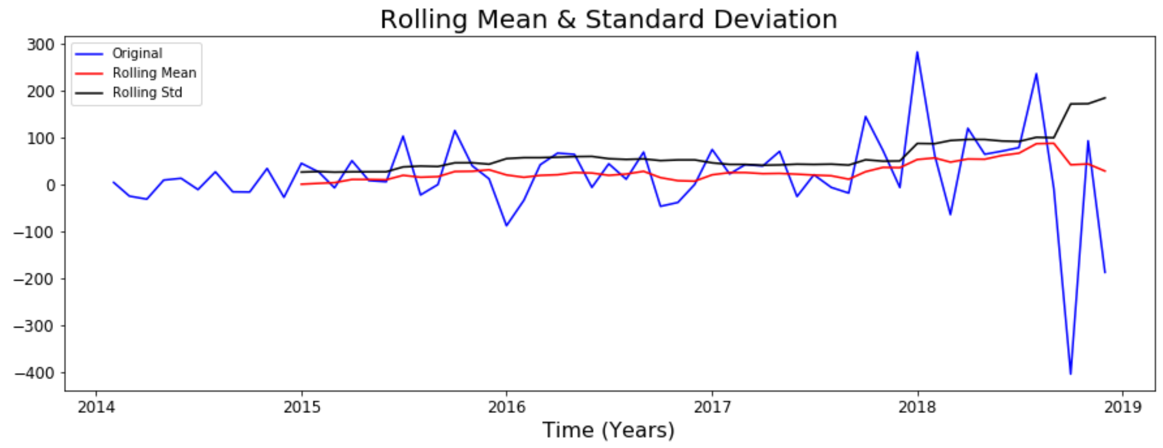


FIGURE 3 – Stock Price Decomposition Result

In attempt to correct trend and seasonality we will try to apply differencing of 1 shift $(y_t - y_{t-1})$, so d=1, and test the data for stationarity again. In Figure 4 we can see the test statistics result from Dickey-Fuller test is between 1% and 5% meaning we have enough evidence to conclude that data is stationary after the conversion.



```
Results of Dickey-Fuller Test:
Test Statistic                 -2.957058
p-value                         0.039104
#Lags Used                     11.000000
Number of Observations Used    47.000000
Critical Value (1%)            -3.577848
Critical Value (5%)            -2.925338
Critical Value (10%)           -2.600774
```

FIGURE 4 – Dickey-Fuller Test result with d=1

Now we review the ACF and PACF plot to get additional data regarding stationarity and pick p and q values for our model.

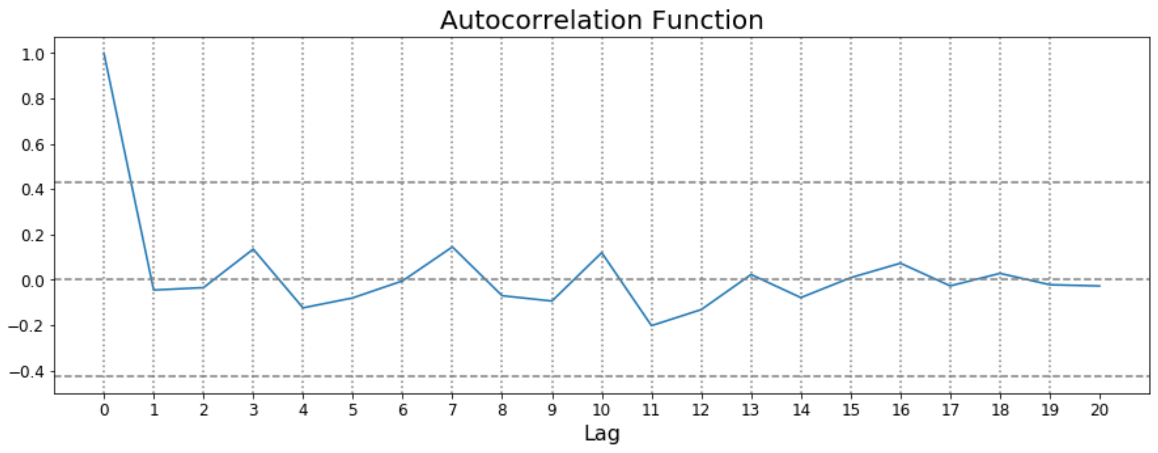


FIGURE 5 – Autocorrelation Function Result

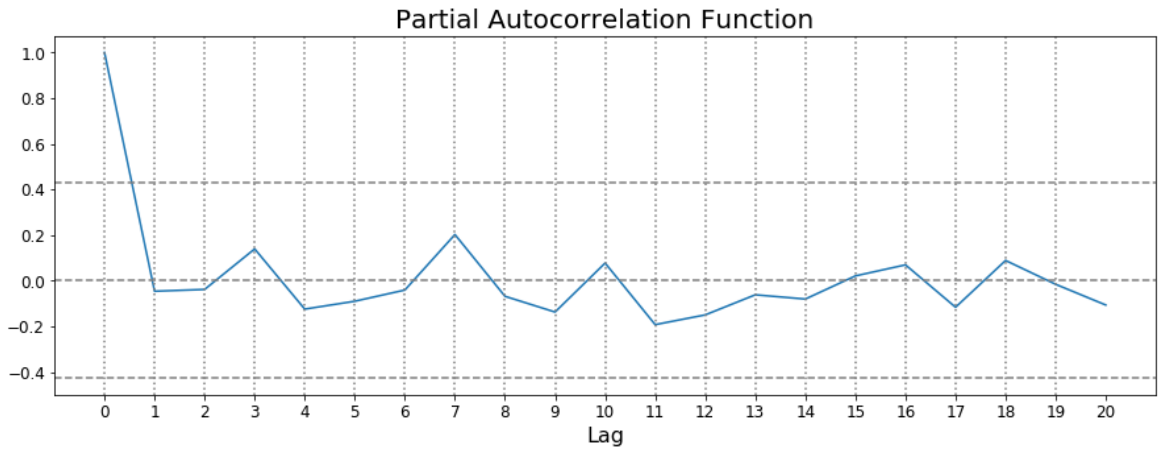
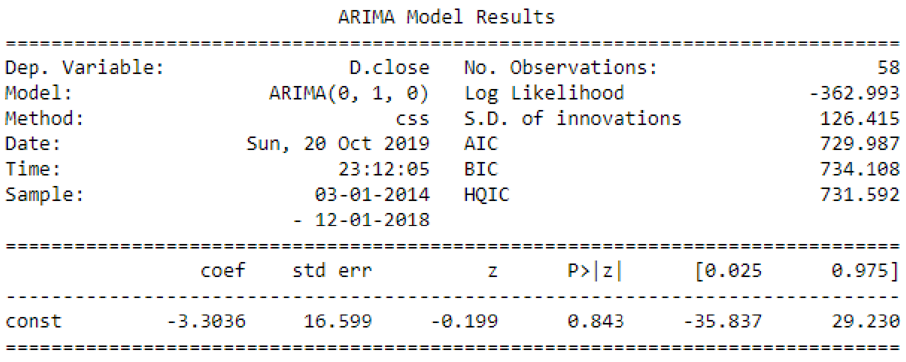


FIGURE 6 – Partial Autocorrelation Function Result

Both autocorrelation and partial autocorrelation result show correlation above the CI limit only for lag=0. It supports that current data post differencing application is stationary and we will select p=0 and q=0.

Finally we will fit the ARIMA model with our selected p=0, d=1 and q=0.



```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.close   No. Observations:                   58
Model:                 ARIMA(0, 1, 0)   Log Likelihood                -362.993
Method:                           css   S.D. of innovations            126.415
Date:                Sun, 20 Oct 2019   AIC                            729.987
Time:                        23:12:05   BIC                            734.108
Sample:                    03-01-2014   HQIC                           731.592
                         - 12-01-2018
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.3036     16.599     -0.199      0.843     -35.837      29.230
==============================================================================
```

FIGURE 7 – ARIMA Model Result Amazon stock data

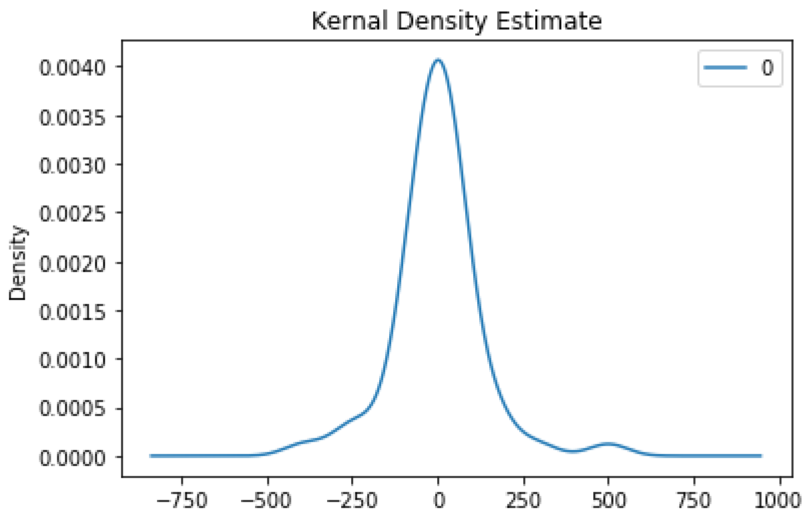


FIGURE 8 – Kernel Density plot of residuals Amazon stock data (0,1,0 model)

The Figure 8 represents the density of residual error. The plot has Gaussian form and suggests that our modeling process is correct.

In the next step we will use different approach, Brute Force search, on stationary differencing data to find the p and q for ARIMA model. The result in Figure 9 shows the model result of ARIMA (3,0,1) and calculated LOESS of 652.8637 is the best combination for Amazon stock data set.

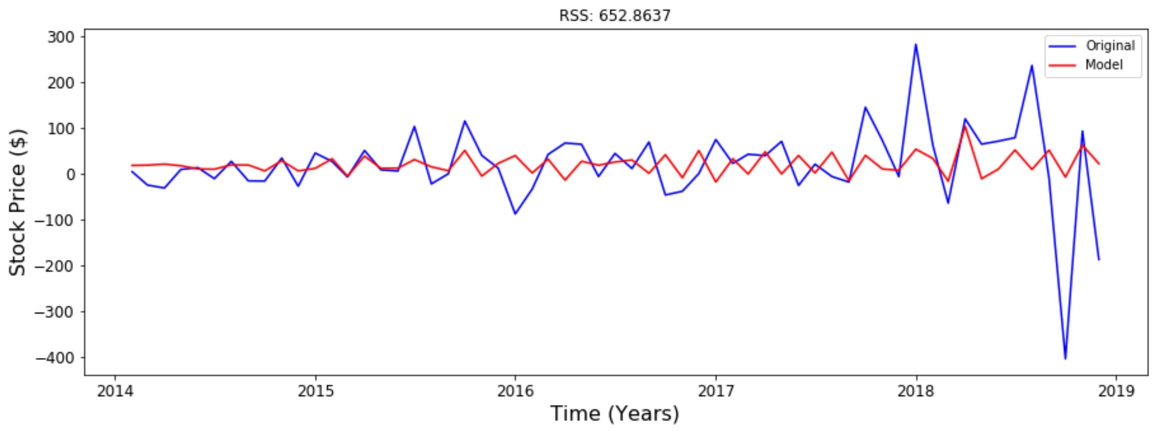


FIGURE 9 – Original vs Model result

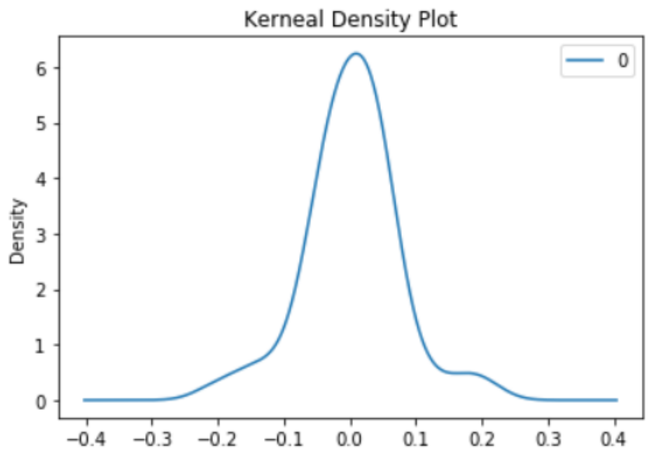
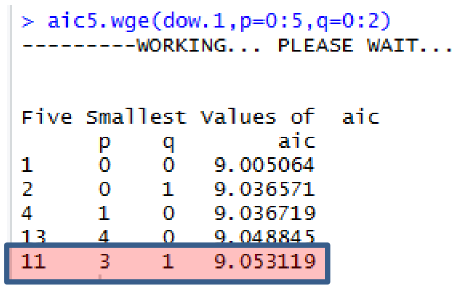


FIGURE 10 – Kernel Density plot of residuals Amazon stock data (3,0,1 model)

4. Conclusion

We examined two different approaches in our case study, using ACF and PACF functions to analyze step by step evaluating each result separately and second one using Brute Force search to find the best combination. The Kernel Density Plot for both 0,1,0 model and 3,0,1 model match and are closer to normal distribution. In addition, we run the analysis using tswge package in R and one of the 5 best recommended combinations is matching our result using ARIMA package in R.



```
> aic5.wge(dow.1,p=0:5,q=0:2)
---------WORKING... PLEASE WAIT...


Five Smallest Values of  aic
      p    q        aic
1     0    0    9.005064
2     0    1    9.036571
4     1    0    9.036719
13    4    0    9.048845
11    3    1    9.053119
```

FIGURE 10 – Recommended p & q values using R tswge package

5. Appendix – Code

```
%matplotlib inline

import os
import pandas as pd
import pandas_datareader as pdr
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot as plt
import datetime
from datetime import datetime as dt
import pandas_datareader.data as web
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
import numpy as np
from statsmodels.tsa.arima_model import ARIMA

pd.core.common.is_list_like = pd.api.types.is_list_like

#from pandas_datareader import data as web
start = datetime.datetime(2014, 1, 1)
end = datetime.datetime(2018, 12, 31)

api_key = 'ASQ3S4V70MVE74NR'
price = web.DataReader("AMZN", "av-monthly",
start=start,end=end,api_key=api_key)

price_new = pd.DataFrame(price['close'])
price_new.index = pd.to_datetime(price_new.index)

print(price_new.head())

#convert to equal frequencies, we have monthly data so each
data point is 1st #day of the month
price_new.index = pd.to_datetime(price_new.index)
price_new.index = price_new.index -
pd.to_timedelta(price_new.index.day - 1, unit='d')

#Function for checking stationarity Dickey-Fuller test and
#calculating the #rolling mean and standard deviation

def test_stationarity(timeseries,sw=1,mymeanstd=12):

    #Determing rolling statistics
    rolmean = timeseries.rolling(mymeanstd).mean()
    rolstd = timeseries.rolling(mymeanstd).std()

    #Plot rolling statistics:
    fig, ax = plt.subplots(1,1,figsize=(15,5))
      orig = plt.plot(timeseries,
      color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling
     Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling
      Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation',
fontsize=20)
```

```
    plt.xlabel('Time (Years)', fontsize=16)
    if sw==1:
        plt.ylabel('Stock Price ($)', fontsize=16)
    plt.tick_params(labelsize=12)
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    timeseries=timeseries.iloc[:,0]
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test
Statistic','p-value','#Lags Used','Number of Observations
Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

# test stationarity
test_stationarity(price_new,1,12)

#decomposition of the dataset
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(price_new,freq=12)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
fig, ax = plt.subplots(4,1,figsize=(14,12),dpi=80,
facecolor='w', edgecolor='k')

plt.subplot(411)
plt.plot(price_new, label='Original')
plt.legend(loc='upper left')

plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='upper left')

plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='upper left')

plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='upper left')

fig.suptitle('Stock Price - Decomposition', fontsize=20)
fig.tight_layout()
fig.subplots_adjust(top=0.95)

plt.tick_params(labelsize=12)
plt.show

#take difference to make data stationer
diff_price = price_new-price_new.shift()
test_stationarity(diff_price.dropna(),0,12)

#ACF calculation and plotting
#from statsmodels.tsa.stattools import acf, pacf
```

```
acf_stock_price = acf(diff_price.dropna(), nlags=20)
fig, ax = plt.subplots(1,1,figsize=(15,5))
plt.title("Autocorrelation Function", fontsize=20)
plt.xlabel('Lag', fontsize=16)
plt.tick_params(labelsize=12)
ax.plot(acf_stock_price)
plt.xticks(np.arange(21))
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-
1.96/np.sqrt(len(acf_stock_price)),linestyle='--
',color='gray')
plt.axhline(y=1.96/np.sqrt(len(acf_stock_price)),linestyle='
--',color='gray')

for i in range(0,20):
    plt.axvline(x=i,linestyle=':',color='gray')

#PACF calculation and plotting
pacf_stock_price = pacf(diff_price.dropna(), nlags=20)
#Plot PACF:
fig, ax = plt.subplots(1,1,figsize=(15,5))
plt.title("Partial Autocorrelation Function", fontsize=20)
plt.xlabel('Lag', fontsize=16)
plt.tick_params(labelsize=12)
plt.plot(pacf_stock_price)
plt.xticks(np.arange(21))
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-
1.96/np.sqrt(len(acf_stock_price)),linestyle='--
',color='gray')
plt.axhline(y=1.96/np.sqrt(len(acf_stock_price)),linestyle='
--',color='gray')
for i in range(0,20):
    plt.axvline(x=i,linestyle=':',color='gray')

#Calculated ARIMA model :
model = ARIMA(diff_price.dropna(), order=(0,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
plt.show()
residuals.plot(kind='kde', title='Kernal Density Estimate')
plt.show()
print(residuals.describe())

#Brute Force approach
diff_price = diff_price.astype(float)
loss_best = 1E16
best_ints = [-1,-1,-1]

for p in range(4):
    for d in range(2):
        for q in range(2):
            model = ARIMA(diff_price.dropna(), order=(p, d,
                  q))
            try:
                results_ARIMA = model.fit(disp=-1)
            except ValueError:
                pass
            except:
                pass

            fig, ax = plt.subplots(1,1,figsize=(15,5))


ax.plot(diff_price,color='blue',label='Original')
            ax.plot(results_ARIMA.fittedvalues,
color='red',label='Model')

            plt.legend(loc='best')
            plt.xlabel('Time (Years)', fontsize=16)
            plt.ylabel('Stock Price ($)', fontsize=16)
            plt.tick_params(labelsize=12)
            x=pd.DataFrame(results_ARIMA.fittedvalues)
            x=x.join(diff_price)
            x['out']=(x.iloc[:,0]-x.iloc[:,1])**2
            loss=np.sqrt(x['out'].sum())
            plt.title('RSS: %.4f'% loss)
            print("Evaludated p,d,q", p,d,q)
            if loss < loss_best:
                print("LOSS=" ,loss)
                loss_best = loss
                best_ints=[p,d,q]
            plt.show()

print("_______________________________________________")

#print the best combination
print(loss_best)
print(best_ints)
```