Untitled5

December 29, 2022

[3]:
```
import pandas as kunfu
import numpy as dragon
import pylab as p
import matplotlib.pyplot as plot
from collections import Counter
import re

#importing packages for the prediction of time-series data
import statsmodels.api as sm
import statsmodels.tsa.api as smt
import statsmodels.formula.api as smf

from sklearn.metrics import mean_squared_error
```

[5]:
```
train = kunfu.read_csv('Desktop//coin_Bitcoin.csv')
train.head(10)
```

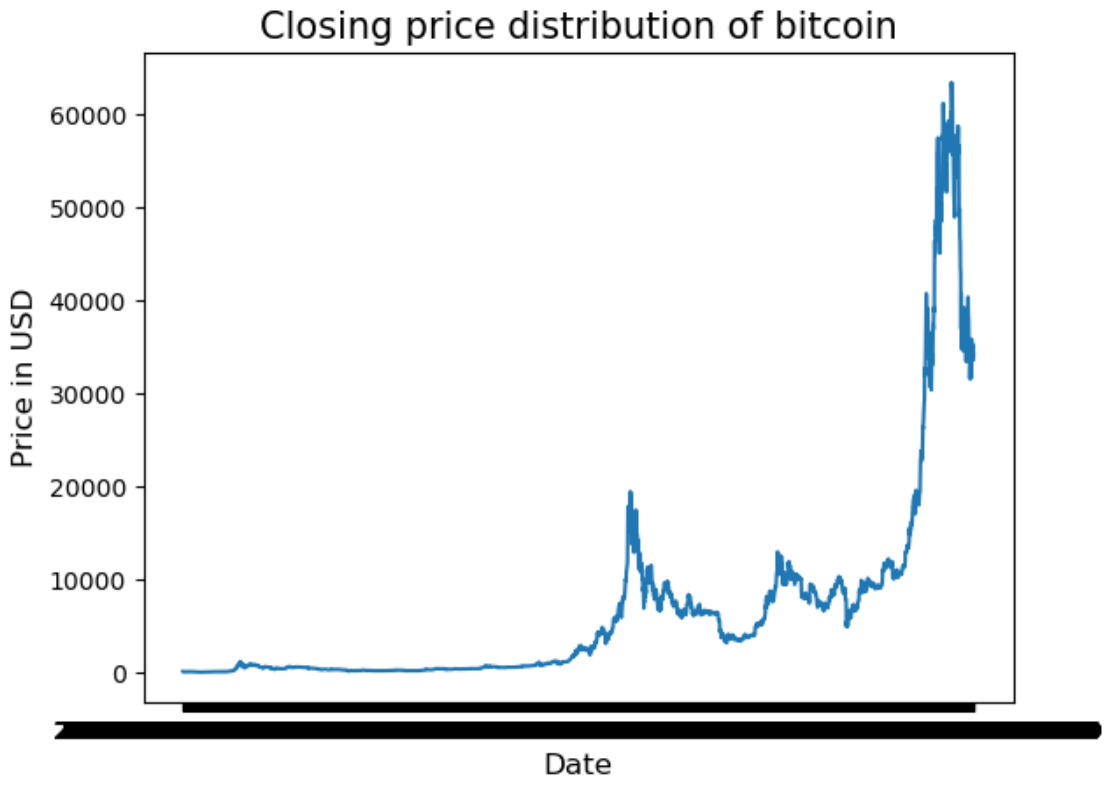
[5]:

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.0	1.603769e+09
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.0	1.542813e+09
2	3	Bitcoin	BTC	2013-05-01 23:59:59	139.889999	107.720001	139.000000	116.989998	0.0	1.298955e+09
3	4	Bitcoin	BTC	2013-05-02 23:59:59	125.599998	92.281898	116.379997	105.209999	0.0	1.168517e+09
4	5	Bitcoin	BTC	2013-05-03 23:59:59	108.127998	79.099998	106.250000	97.750000	0.0	1.085995e+09
5	6	Bitcoin	BTC	2013-05-04 23:59:59	115.000000	92.500000	98.099998	112.500000	0.0	1.250317e+09
6	7	Bitcoin	BTC	2013-05-05 23:59:59	118.800003	107.142998	112.900002	115.910004	0.0	1.288693e+09
7	8	Bitcoin	BTC	2013-05-06 23:59:59	124.663002	106.639999	115.980003	112.300003	0.0	1.249023e+09
8	9	Bitcoin	BTC	2013-05-07 23:59:59	113.444000	97.699997				
9	10	Bitcoin	BTC	2013-05-08 23:59:59	115.779999	109.599998				

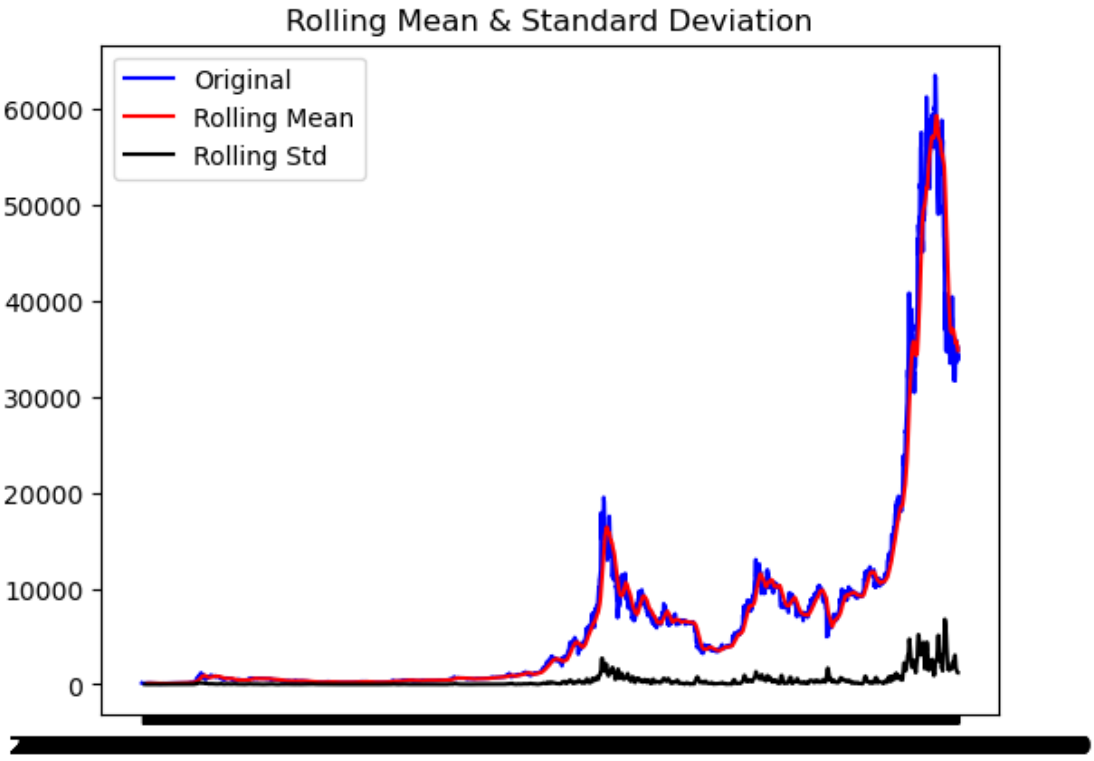
```
8  112.250000  111.500000     0.0  1.240594e+09
9  109.599998  113.566002     0.0  1.264049e+09
```

[6]:
```
data = train['Close']
Date1 = train['Date']
train1 = train[['Date','Close']]
# Setting the Date as Index
train2 = train1.set_index('Date')
train2.sort_index(inplace=True)
print (type(train2))
print (train2.head())
plot.plot(train2)
plot.xlabel('Date', fontsize=12)
plot.ylabel('Price in USD', fontsize=12)
plot.title("Closing price distribution of bitcoin", fontsize=15)
plot.show()
```

```
<class 'pandas.core.frame.DataFrame'>
                         Close
Date
2013-04-29 23:59:59  144.539993
2013-04-30 23:59:59  139.000000
2013-05-01 23:59:59  116.989998
2013-05-02 23:59:59  105.209999
2013-05-03 23:59:59   97.750000
```

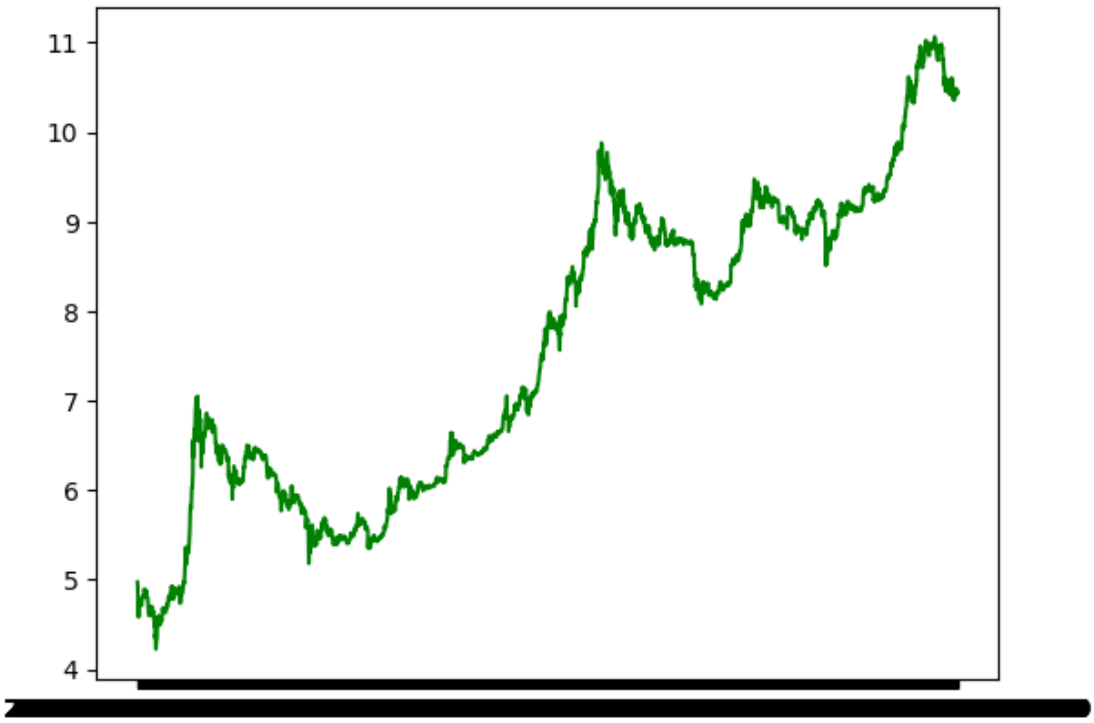


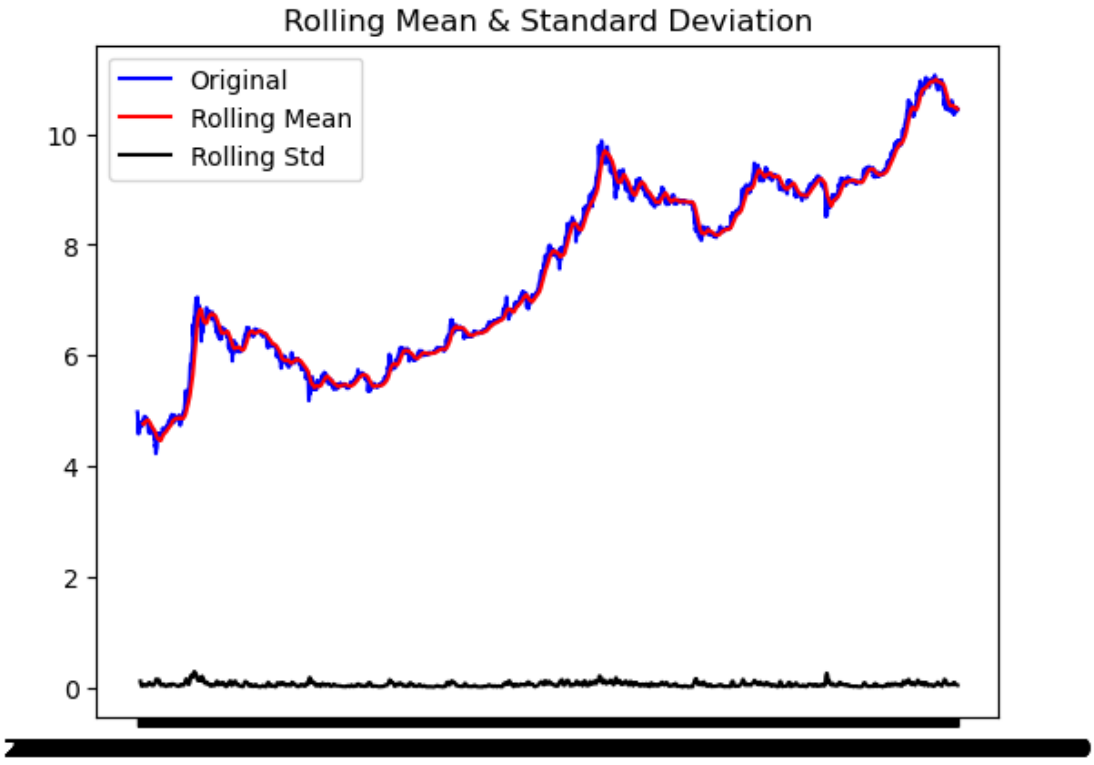
[7]:
```
from statsmodels.tsa.stattools import adfuller

def test_stationarity(x):


    #Determing rolling statistics
    rolmean = x.rolling(window=22,center=False).mean()

    rolstd = x.rolling(window=12,center=False).std()

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

    #Perform Dickey Fuller test
    result=adfuller(x)
    print('ADF Stastistic: %f'%result[0])
    print('p-value: %f'%result[1])
    pvalue=result[1]
    for key,value in result[4].items():
         if result[0]>value:
            print("The graph is non stationery")
            break
         else:
            print("The graph is stationery")
            break;
    print('Critical values:')
    for key,value in result[4].items():
        print('\t%s: %.3f ' % (key, value))

ts = train2['Close']
test_stationarity(ts)
```




```
ADF Stastistic: -0.797310
p-value: 0.819911
The graph is non stationery
Critical values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
```

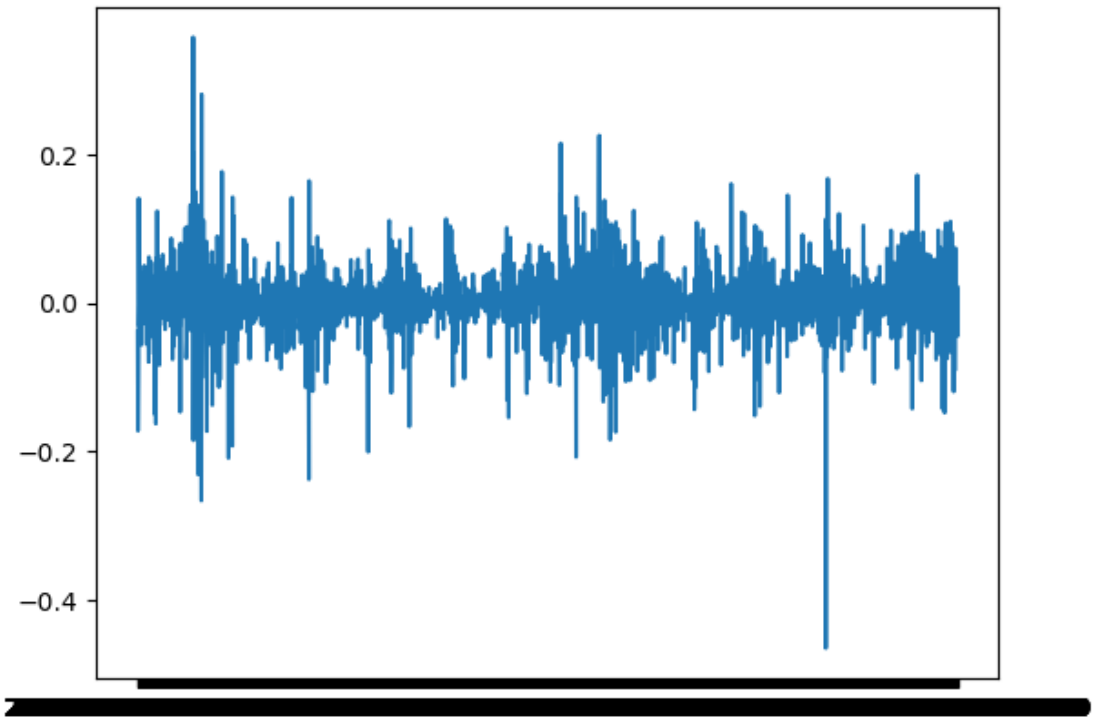
[8]:
```
ts_log = dragon.log(ts)
plot.plot(ts_log,color="green")
plot.show()

test_stationarity(ts_log)
```


11
10
9
8
7
6
5
4

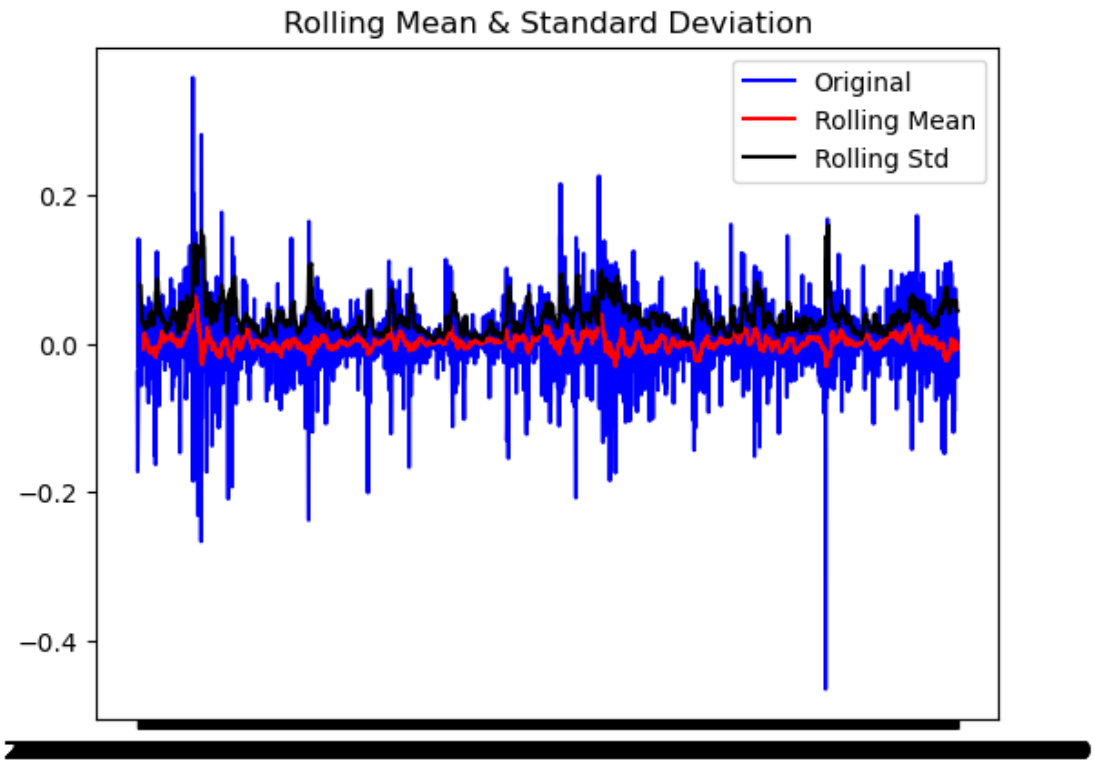

Rolling Mean & Standard Deviation
Original
Rolling Mean
Rolling Std
10
8
6
4
2
0

```
ADF Stastistic: -0.776693
p-value: 0.825873
The graph is non stationery
Critical values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
```

[9]:
```
ts_log_diff = ts_log - ts_log.shift()
plot.plot(ts_log_diff)
plot.show()
```



[10]:
```
ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)
```



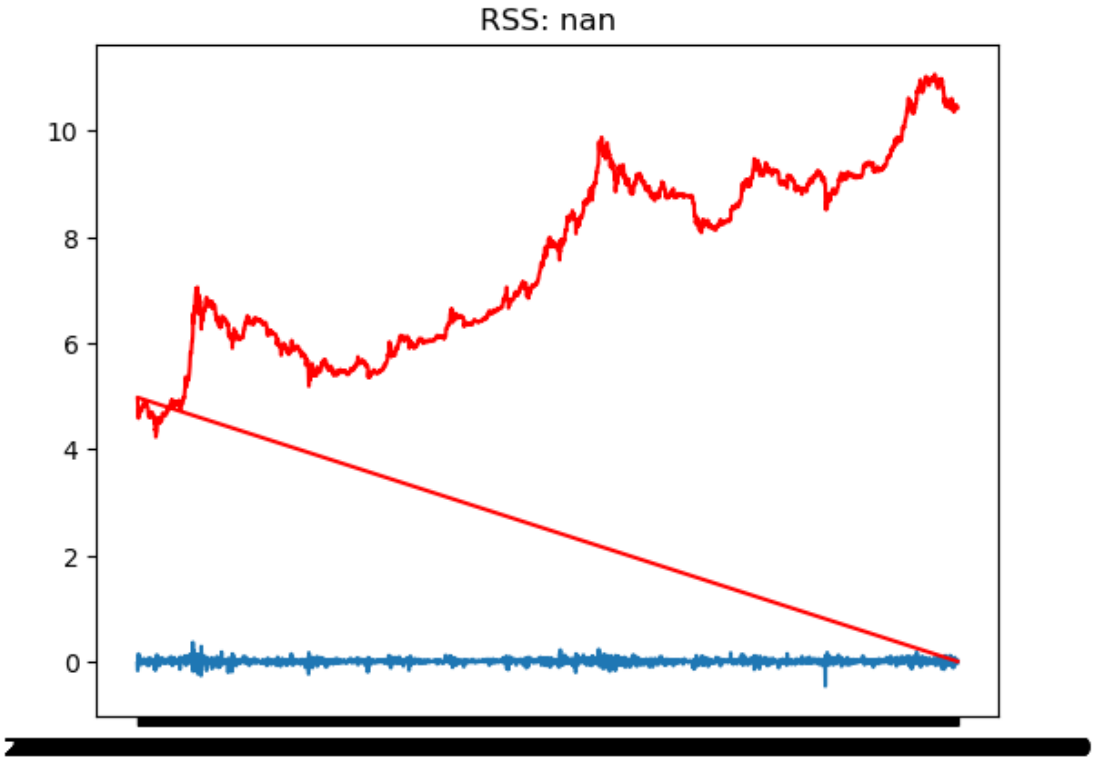

```
ADF Stastistic: -10.394251
p-value: 0.000000
The graph is stationery
Critical values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
```

[12]:
```
from statsmodels.tsa.arima_model import ARIMA
```

[17]:
```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(ts_log, order=(2,1,0))
results_ARIMA = model.fit()
plot.plot(ts_log_diff)
plot.plot(results_ARIMA.fittedvalues, color='red')
plot.title('RSS: %.7f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
plot.show()
```

```
C:\Users\sadok\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
C:\Users\sadok\anaconda3\lib\site-
```

```
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
C:\Users\sadok\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```



[20]:
```
size = int(len(ts_log)-100)
# Divide into train and test
train_arima, test_arima = ts_log[0:size], ts_log[size:len(ts_log)]
history = [x for x in train_arima]
predictions = list()
originals = list()
error_list = list()

print('Printing Predicted vs Expected Values...')
print('\n')
# We go over each value in the test set and then apply ARIMA model and calculate␣
 ↪the predicted value. We have the expected value in the test set therefore we␣
 ↪calculate the error between predicted and expected value
for t in range(len(test_arima)):
```

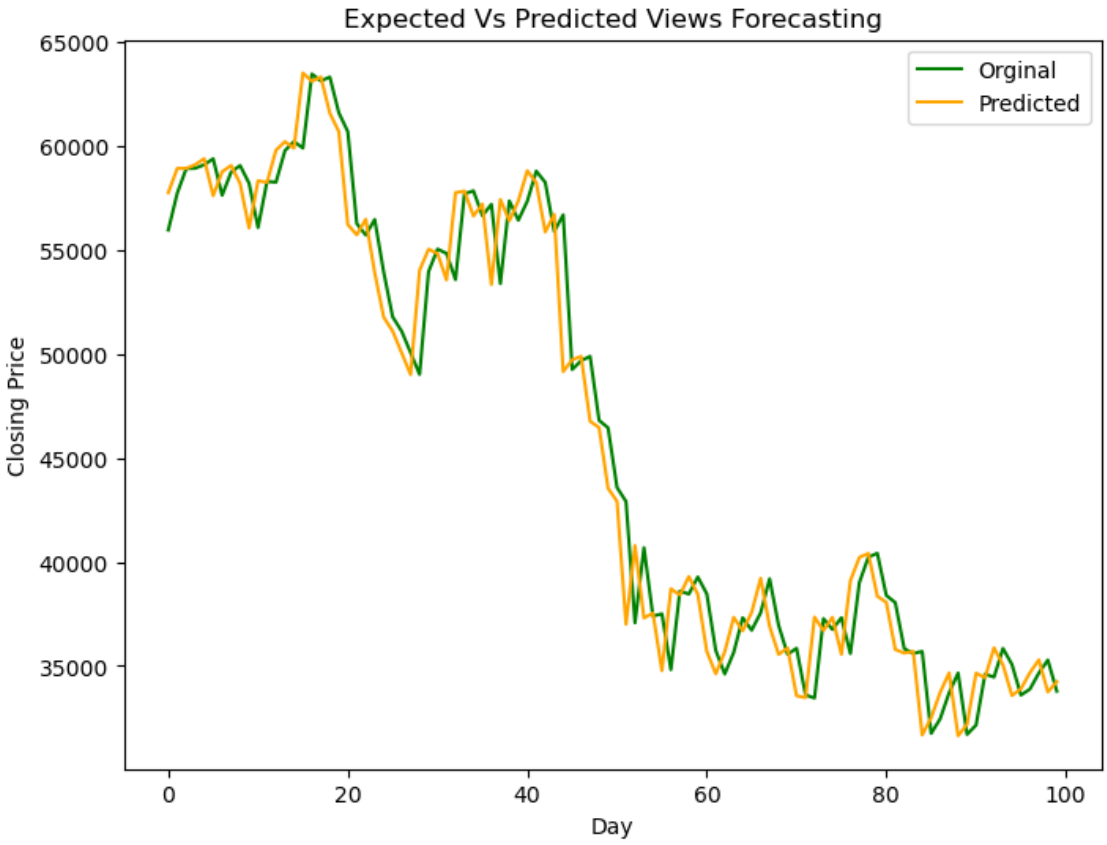
```
    model = ARIMA(history, order=(2, 1, 0))
    model_fit = model.fit()

    output = model_fit.forecast()

    pred_value = output[0]


    original_value = test_arima[t]
    history.append(original_value)

    pred_value = dragon.exp(pred_value)


    original_value = dragon.exp(original_value)

    # Calculating the error
    error = ((abs(pred_value - original_value)) / original_value) * 100
    error_list.append(error)
    print('predicted = %f,   expected = %f,   error = %f ' % (pred_value,␣
 ↪original_value, error), '%')

    predictions.append(float(pred_value))
    originals.append(float(original_value))

# After iterating over whole test set the overall mean error is calculated.
print('\n Mean Error in Predicting Test Case Articles : %f ' % (sum(error_list)/
 ↪float(len(error_list))), '%')
plot.figure(figsize=(8, 6))
test_day = [t
           for t in range(len(test_arima))]
labels={'Orginal','Predicted'}
plot.plot(test_day, predictions, color= 'green')
plot.plot(test_day, originals, color = 'orange')
plot.title('Expected Vs Predicted Views Forecasting')
plot.xlabel('Day')
plot.ylabel('Closing Price')
plot.legend(labels)
plot.show()
```

```
Printing Predicted vs Expected Values...


predicted = 55953.817479,   expected = 57750.199871,   error = 3.110608  %
predicted = 57728.413048,   expected = 58917.693045,   error = 2.018545  %
predicted = 58910.206571,   expected = 58918.832714,   error = 0.014641  %
predicted = 58922.849711,   expected = 59095.807978,   error = 0.292674  %
```

```
predicted = 59093.728935,   expected = 59384.313359,   error = 0.489329  %
predicted = 59381.526060,   expected = 57603.889174,   error = 3.085967  %
predicted = 57625.502463,   expected = 58758.556291,   error = 1.928321  %
predicted = 58738.561484,   expected = 59057.878966,   error = 0.540686  %
predicted = 59058.298279,   expected = 58192.358684,   error = 1.488064  %
predicted = 58203.559024,   expected = 56048.937813,   error = 3.844178  %
predicted = 56070.985016,   expected = 58323.953580,   error = 3.862853  %
predicted = 58288.985730,   expected = 58245.003001,   error = 0.075513  %
predicted = 58253.364468,   expected = 59793.235410,   error = 2.575326  %
predicted = 59774.025515,   expected = 60204.964914,   error = 0.715787  %
predicted = 60205.351139,   expected = 59893.451889,   error = 0.520757  %
predicted = 59898.578214,   expected = 63503.457930,   error = 5.676667  %
predicted = 63457.364541,   expected = 63109.695935,   error = 0.550896  %
predicted = 63127.206953,   expected = 63314.010845,   error = 0.295044  %
predicted = 63310.150695,   expected = 61572.789601,   error = 2.821638  %
predicted = 61594.410082,   expected = 60683.821233,   error = 1.500546  %
predicted = 60688.571331,   expected = 56216.185002,   error = 7.955692  %
predicted = 56264.107326,   expected = 55724.267098,   error = 0.968770  %
predicted = 55713.223704,   expected = 56473.032281,   error = 1.345436  %
predicted = 56462.301965,   expected = 53906.088046,   error = 4.741976  %
predicted = 53938.658160,   expected = 51762.272051,   error = 4.204580  %
predicted = 51777.525360,   expected = 51093.650710,   error = 1.338473  %
predicted = 51093.277990,   expected = 50050.868216,   error = 2.082701  %
predicted = 50060.000430,   expected = 49004.253263,   error = 2.154399  %
predicted = 49011.845871,   expected = 54021.754787,   error = 9.273873  %
predicted = 53956.106706,   expected = 55033.118013,   error = 1.957024  %
predicted = 55040.833580,   expected = 54824.704895,   error = 0.394218  %
predicted = 54830.639549,   expected = 53555.108491,   error = 2.381717  %
predicted = 53568.607408,   expected = 57750.177346,   error = 7.240792  %
predicted = 57694.905501,   expected = 57828.050632,   error = 0.230243  %
predicted = 57841.836705,   expected = 56631.077413,   error = 2.137977  %
predicted = 56645.080733,   expected = 57200.291223,   error = 0.970643  %
predicted = 57189.950002,   expected = 53333.537753,   error = 7.230745  %
predicted = 53379.356993,   expected = 57424.005948,   error = 7.043481  %
predicted = 57355.960224,   expected = 56396.514164,   error = 1.701251  %
predicted = 56425.135063,   expected = 57356.401877,   error = 1.623649  %
predicted = 57339.719042,   expected = 58803.775809,   error = 2.489733  %
predicted = 58788.585597,   expected = 58232.316142,   error = 0.955259  %
predicted = 58245.235966,   expected = 55859.797545,   error = 4.270403  %
predicted = 55887.888674,   expected = 56704.573059,   error = 1.440244  %
predicted = 56685.026648,   expected = 49150.533875,   error = 15.329422  %
predicted = 49248.533083,   expected = 49716.191603,   error = 0.940656  %
predicted = 49674.875393,   expected = 49880.533420,   error = 0.412301  %
predicted = 49880.910843,   expected = 46760.186561,   error = 6.673892  %
predicted = 46802.361598,   expected = 46456.058474,   error = 0.745442  %
predicted = 46446.793448,   expected = 43537.511389,   error = 6.682242  %
predicted = 43574.307728,   expected = 42909.400925,   error = 1.549560  %
predicted = 42903.052565,   expected = 37002.440466,   error = 15.946548  %
```

```
predicted = 37069.581340,   expected = 40782.738262,   error = 9.104727  %
predicted = 40680.040156,   expected = 37304.690671,   error = 9.048056  %
predicted = 37394.998540,   expected = 37536.631112,   error = 0.377318  %
predicted = 37501.753510,   expected = 34770.583623,   error = 7.854829  %
predicted = 34818.792151,   expected = 38705.978637,   error = 10.042858  %
predicted = 38596.915803,   expected = 38402.223851,   error = 0.506981  %
predicted = 38451.843315,   expected = 39294.197382,   error = 2.143711  %
predicted = 39273.733486,   expected = 38436.968535,   error = 2.176980  %
predicted = 38462.875820,   expected = 35697.606390,   error = 7.746372  %
predicted = 35738.406594,   expected = 34616.068003,   error = 3.242247  %
predicted = 34607.477104,   expected = 35678.129204,   error = 3.000864  %
predicted = 35646.738669,   expected = 37332.853689,   error = 4.516438  %
predicted = 37313.702642,   expected = 36684.924517,   error = 1.713996  %
predicted = 36713.598865,   expected = 37575.179576,   error = 2.292952  %
predicted = 37551.815550,   expected = 39208.765995,   error = 4.225969  %
predicted = 39188.036525,   expected = 36894.405330,   error = 6.216745  %
predicted = 36951.967826,   expected = 35551.958726,   error = 3.937924  %
predicted = 35555.118842,   expected = 35862.377727,   error = 0.856772  %
predicted = 35843.982215,   expected = 33560.707838,   error = 6.803415  %
predicted = 33604.606677,   expected = 33472.631748,   error = 0.394277  %
predicted = 33452.058220,   expected = 37345.121486,   error = 10.424556  %
predicted = 37269.025729,   expected = 36702.599375,   error = 1.543287  %
predicted = 36749.298761,   expected = 37334.399526,   error = 1.567189  %
predicted = 37316.629757,   expected = 35552.517148,   error = 4.961991  %
predicted = 35590.980747,   expected = 39097.860897,   error = 8.969494  %
predicted = 39006.516325,   expected = 40218.477859,   error = 3.013445  %
predicted = 40231.520541,   expected = 40406.268987,   error = 0.432479  %
predicted = 40413.165696,   expected = 38347.063227,   error = 5.387903  %
predicted = 38387.326782,   expected = 38053.504173,   error = 0.877245  %
predicted = 38041.346126,   expected = 35787.244782,   error = 6.298617  %
predicted = 35826.588697,   expected = 35615.869270,   error = 0.591645  %
predicted = 35598.291126,   expected = 35698.296443,   error = 0.280140  %
predicted = 35695.099724,   expected = 31676.693733,   error = 12.685686  %
predicted = 31749.685505,   expected = 32505.659824,   error = 2.325670  %
predicted = 32451.957281,   expected = 33723.028978,   error = 3.769150  %
predicted = 33706.585742,   expected = 34662.435894,   error = 2.757597  %
predicted = 34655.461411,   expected = 31637.780055,   error = 9.538221  %
predicted = 31701.944883,   expected = 32186.277671,   error = 1.504780  %
predicted = 32150.732535,   expected = 34649.644588,   error = 7.211941  %
predicted = 34603.460408,   expected = 34434.335314,   error = 0.491152  %
predicted = 34456.577979,   expected = 35867.777735,   error = 3.934450  %
predicted = 35837.059814,   expected = 35040.837249,   error = 2.272270  %
predicted = 35068.128428,   expected = 33572.117653,   error = 4.456111  %
predicted = 33594.895671,   expected = 33897.048590,   error = 0.891384  %
predicted = 33879.983367,   expected = 34668.548402,   error = 2.274583  %
predicted = 34655.425673,   expected = 35287.779766,   error = 1.791992  %
predicted = 35281.086575,   expected = 33746.002456,   error = 4.548936  %
predicted = 33780.228281,   expected = 34235.193451,   error = 1.328940  %
```

```
Mean Error in Predicting Test Case Articles : 3.521557  %
```



[ ]: