In [78]:

```
%matplotlib inline
import pandas_datareader.data as web
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [9]:

```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
gdp = web.DataReader("GDPC1", "fred", sdt, edt)
gdp.head()
```

Out[9]:

	GDPC1
DATE	
2000-01-01	12359.1
2000-04-01	12592.5
2000-07-01	12607.7
2000-10-01	12679.3
2001-01-01	12643.3

In [ ]:

```
```

In [29]:

```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
cnp = web.DataReader("CNP16OV", "fred", sdt, edt)
cnp.head()
```

Out[29]:

	CNP16OV
DATE	
2000-01-01	211410
2000-02-01	211576
2000-03-01	211772
2000-04-01	212018
2000-05-01	212242

In [20]:
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
pcec = web.DataReader("PCEC", "fred", sdt, edt)
pcec.head()
```

Out[20]:

	PCEC
DATE	
2000-01-01	6642.7
2000-04-01	6737.3
2000-07-01	6845.1
2000-10-01	6944.4
2001-01-01	7020.4

In [99]:
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
gdpdef = web.DataReader("GDPDEF", "fred", sdt, edt)
gdpdef.reset_index()
gdpdef.head()
```

Out[99]:

	GDPDEF
DATE	
2000-01-01	81.163
2000-04-01	81.623
2000-07-01	82.152
2000-10-01	82.593
2001-01-01	83.112

In [22]:
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
fpi = web.DataReader("FPI", "fred", sdt, edt)
fpi.head()
```

Out[22]:

	FPI
DATE	
2000-01-01	1929.8
2000-04-01	1981.3
2000-07-01	1998.5
2000-10-01	2007.2
2001-01-01	1998.2

In [23]:
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
comp = web.DataReader("COMPNFB", "fred", sdt, edt)
comp.head()
```

Out[23]:

DATE	COMPNFB
2000-01-01	73.167
2000-04-01	73.324
2000-07-01	74.755
2000-10-01	75.170
2001-01-01	76.904

In [24]:
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
prs = web.DataReader("PRS85006023", "fred", sdt, edt)
prs.head()
```

Out[24]:

DATE	PRS85006023
2000-01-01	104.738
2000-04-01	104.601
2000-07-01	104.473
2000-10-01	104.112
2001-01-01	103.646

In [25]:
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
ce = web.DataReader("CE16OV", "fred", sdt, edt)
ce.head()
```

Out[25]:

DATE	CE16OV
2000-01-01	136559
2000-02-01	136598
2000-03-01	136701
2000-04-01	137270
2000-05-01	136630

In [26]:

```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
fed = web.DataReader("FEDFUNDS", "fred", sdt, edt)
fed.head()
```

Out[26]:

	FEDFUNDS
DATE	
2000-01-01	5.45
2000-02-01	5.73
2000-03-01	5.85
2000-04-01	6.02
2000-05-01	6.27

In [27]:

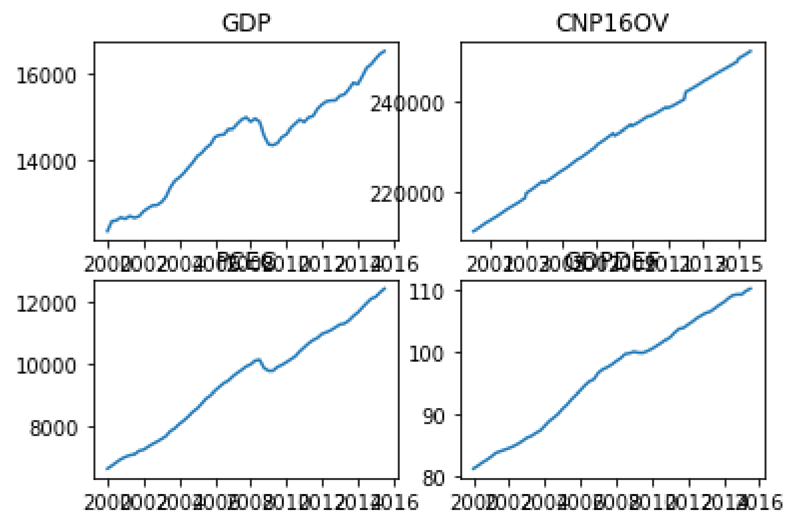
```
sdt = dt.datetime(2000, 1, 1)
edt = dt.datetime(2015, 9, 1)
baa = web.DataReader("BAA", "fred", sdt, edt)
baa.head()
```

Out[27]:

	BAA
DATE	
2000-01-01	8.33
2000-02-01	8.29
2000-03-01	8.37
2000-04-01	8.40
2000-05-01	8.90

In [77]:

```
plt.figure(1)
plt.subplot(221)
plt.plot(gdp)
plt.title('GDP')
plt.subplot(222)
plt.plot(cnp)
plt.title('CNP16OV')
plt.subplot(223)
plt.plot(pcec)
plt.title('PCEC')
plt.subplot(224)
plt.plot(gdpdef)
plt.title('GDPDEF')
plt.show()
```

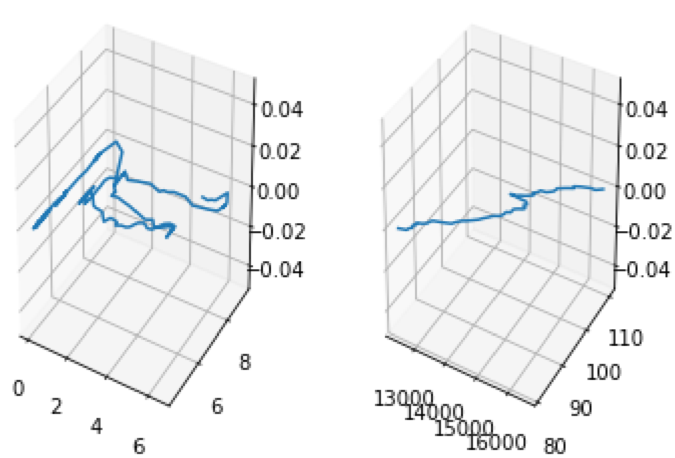


In [96]:

```
plt.figure(2)
fig = plt.figure()
ax = fig.add_subplot(1,2,1,projection='3d')
ax.plot(fed,baa)
ax1=fig.add_subplot(1,2,2,projection='3d')
ax1.plot(gdp,gdpdef)
```

Out[96]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x118ef4a90>]

<matplotlib.figure.Figure at 0x1189909b0>



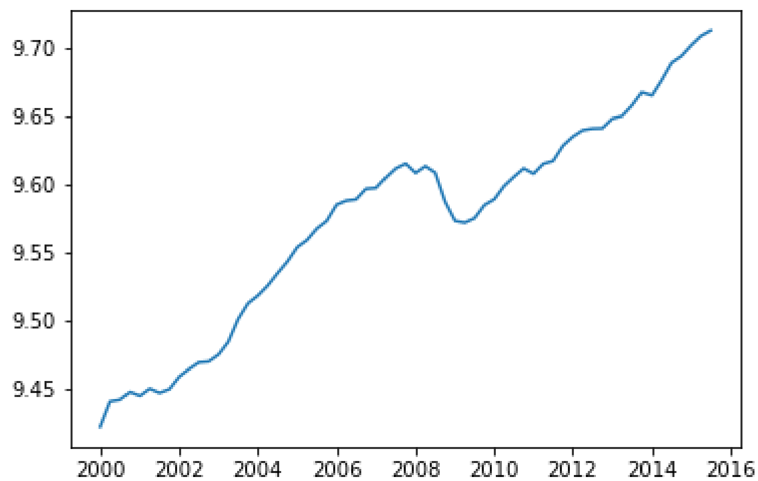
In [109]:
```
ts= gdp
ts_log = np.log(ts)
ts_log.head()
```

Out[109]:

	GDPC1
DATE	
2000-01-01	9.422148
2000-04-01	9.440857
2000-07-01	9.442063
2000-10-01	9.447726
2001-01-01	9.444883

In [110]:
```
plt.plot(ts_log)
```

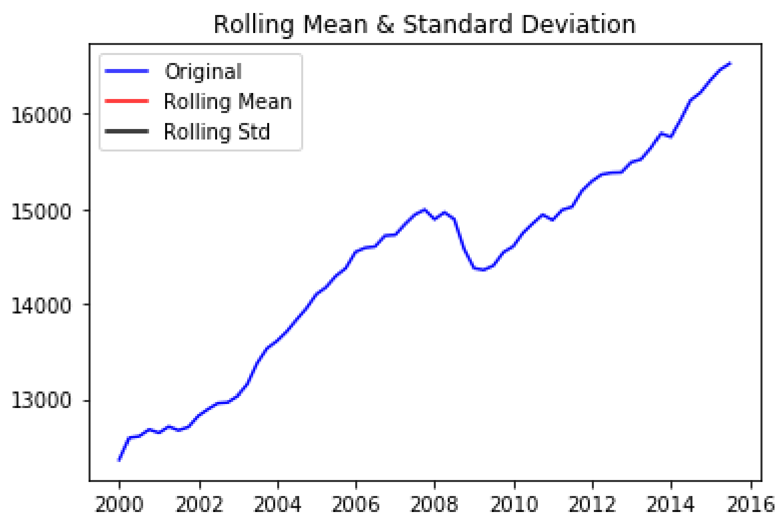
Out[110]: [<matplotlib.lines.Line2D at 0x11b1a8940>]



In [113]:
```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = pd.rolling_mean(timeseries, window=432)
    rolstd = pd.rolling_std(timeseries, window=432)

    #Plot rolling statistics:
    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')
    plt.show(block=False)
```

In [114]:
```
test_stationarity(ts)
```

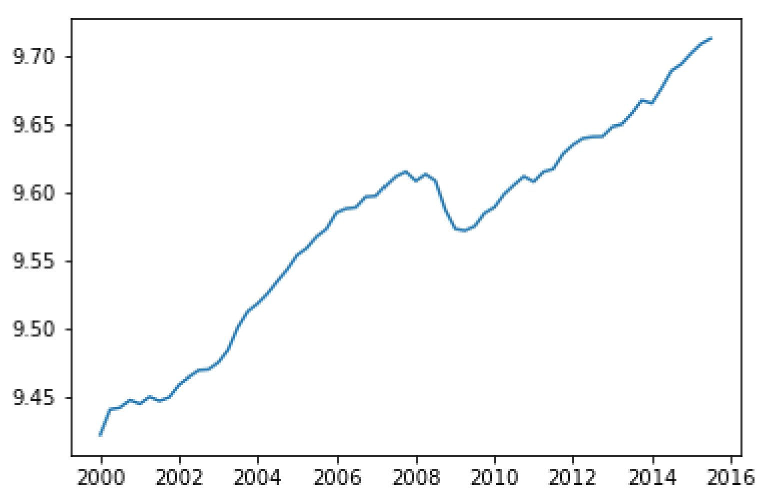
```
/Users/monilshah/anaconda3/lib/python3.5/site-packages/ipykernel_launche
r.py:5: FutureWarning: pd.rolling_mean is deprecated for DataFrame and wi
ll be removed in a future version, replace with
        DataFrame.rolling(window=432,center=False).mean()
  """
/Users/monilshah/anaconda3/lib/python3.5/site-packages/ipykernel_launche
r.py:6: FutureWarning: pd.rolling_std is deprecated for DataFrame and wil
l be removed in a future version, replace with
        DataFrame.rolling(window=432,center=False).std()
```



In [115]:
```
moving_avg = pd.rolling_mean(ts_log,432)
plt.plot(ts_log)
plt.plot(moving_avg, color='red')
```

```
/Users/monilshah/anaconda3/lib/python3.5/site-packages/ipykernel_launche
r.py:1: FutureWarning: pd.rolling_mean is deprecated for DataFrame and wi
ll be removed in a future version, replace with
        DataFrame.rolling(window=432,center=False).mean()
  """Entry point for launching an IPython kernel.
```

Out[115]: [<matplotlib.lines.Line2D at 0x11b4e0be0>]

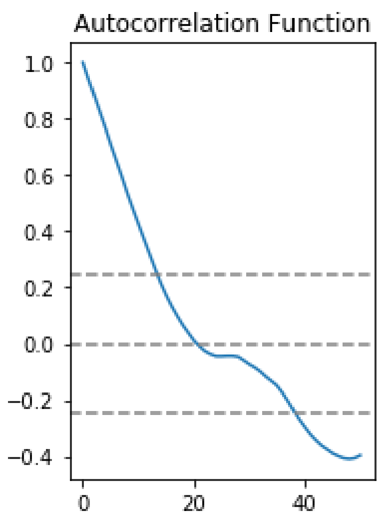


In [118]:
```
from statsmodels.tsa.stattools import acf, pacf
```

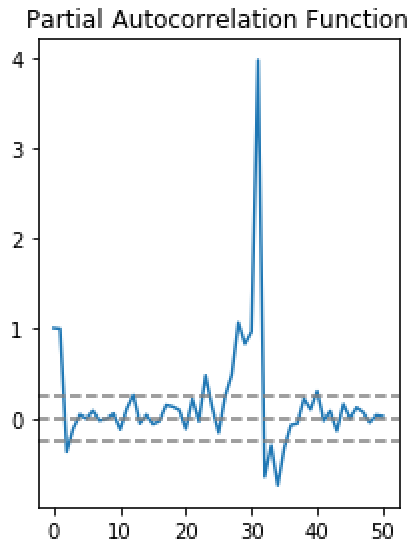
In [120]:
```
lag_acf = acf(ts_log, nlags=50)
lag_pacf = pacf(ts_log, nlags=50, method='ols')
```

In [122]:
```
#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
```

Out[122]: <matplotlib.text.Text at 0x11b40c898>

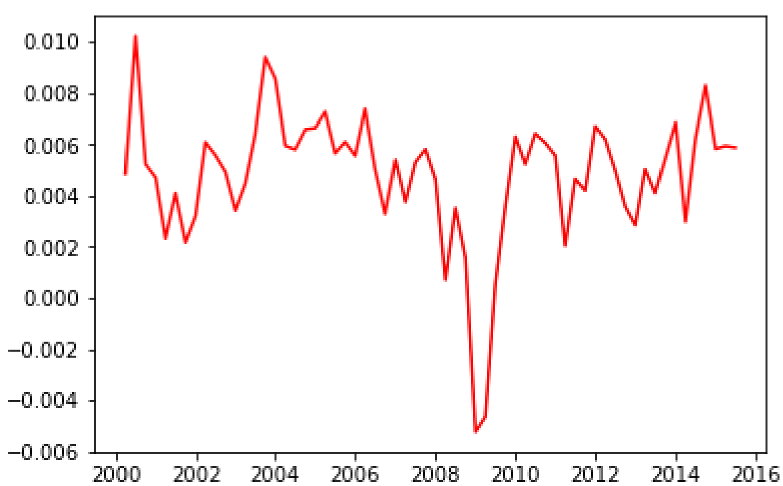


In [123]:
```
#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```



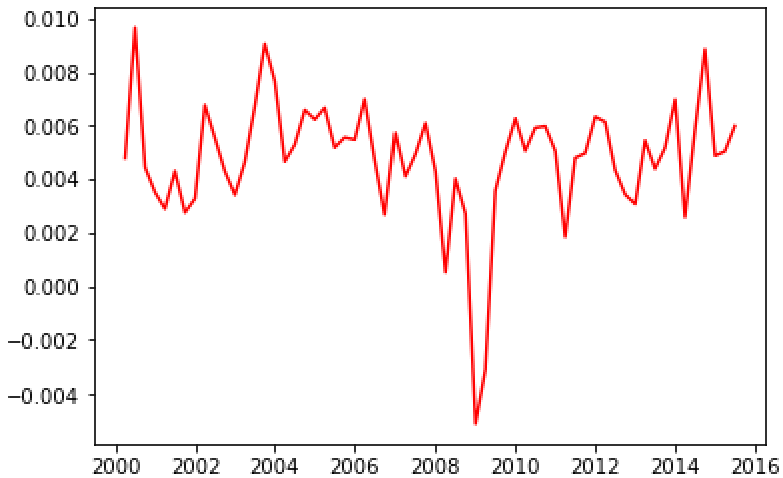
In [127]:
```
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(ts_log, order=(2, 1, 0),freq='B')
results_AR = model.fit(disp=-1)

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

Out[127]: [<matplotlib.lines.Line2D at 0x11d75b4e0>]



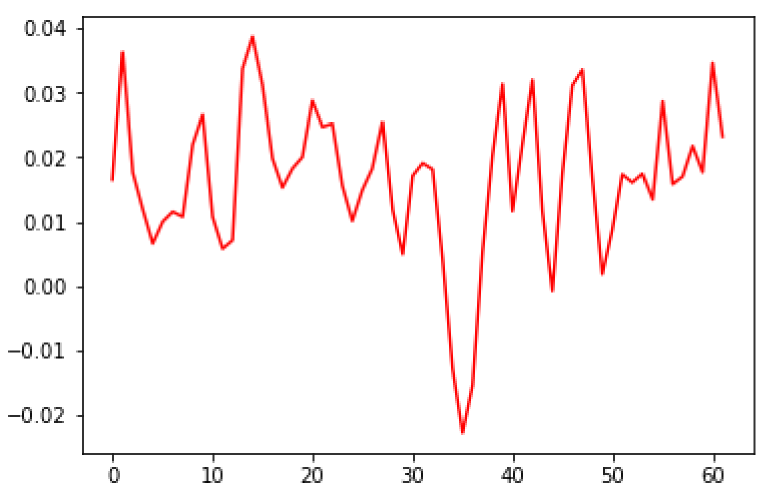
In [128]:
```
model = ARIMA(ts_log, order=(0, 1, 2), freq = 'B')
results_MA = model.fit(disp=-1)
plt.plot(results_MA.fittedvalues, color='red')
```

Out[128]: [<matplotlib.lines.Line2D at 0x11d6d9518>]



In [184]:
```
import time
start1=time.clock()
model = ARIMA(ts_log, order=(10, 1, 2), freq ='B')
results_ARIMA = model.fit(disp=-1)
plt.plot(results_ARIMA.fittedvalues, color='red')
print(time.clock()-start1)
```

```
6.527475000000001

/Users/monilshah/anaconda3/lib/python3.5/site-packages/statsmodels/base/m
odel.py:496: ConvergenceWarning: Maximum Likelihood optimization failed t
o converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```



In [179]:

```
print(results_ARIMA.summary())
# plot residual errors
```

```
                             ARIMA Model Results

==============================================================================
=====
Dep. Variable:                    D.y   No. Observations:
   62
Model:                 ARIMA(10, 1, 2)   Log Likelihood                    15
8.736
Method:                       css-mle   S.D. of innovations
0.018
Date:                Sat, 19 Aug 2017   AIC                              -28
9.473
Time:                        15:19:57   BIC                              -25
9.693
Sample:                             1   HQIC                             -27
7.781


==============================================================================
=====
                coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const         0.0165      0.004      4.057      0.000       0.009
0.024
ar.L1.D.y     1.0437      0.138      7.541      0.000       0.772
1.315
ar.L2.D.y    -0.9060      0.188     -4.815      0.000      -1.275      -
0.537
ar.L3.D.y     0.1359      0.224      0.608      0.546      -0.302
0.574
ar.L4.D.y     0.1710      0.230      0.745      0.460      -0.279
0.621
ar.L5.D.y    -0.1493      0.238     -0.627      0.533      -0.616
0.317
ar.L6.D.y     0.1062      0.242      0.440      0.662      -0.367
0.580
ar.L7.D.y    -0.0876      0.241     -0.363      0.718      -0.561
0.386
ar.L8.D.y    -0.0753      0.245     -0.307      0.760      -0.556
0.406
ar.L9.D.y     0.2908      0.208      1.398      0.168      -0.117
0.698
ar.L10.D.y   -0.2385      0.141     -1.688      0.098      -0.515
0.038
ma.L1.D.y    -0.7328      0.071    -10.288      0.000      -0.872      -
0.593
ma.L2.D.y     1.0000      0.077     13.023      0.000       0.850
1.150
                                    Roots

==============================================================================
```

```
=====
                  Real          Imaginary           Modulus         Freq
uency
-----------------------------------------------------------------------------
-----
AR.1            -1.1505           -0.4071j            1.2204           -
0.4459
AR.2            -1.1505           +0.4071j            1.2204
0.4459
AR.3            -0.4537           -1.0807j            1.1721           -
0.3133
AR.4            -0.4537           +1.0807j            1.1721
0.3133
AR.5             0.2806           -0.9817j            1.0210           -
0.2057
AR.6             0.2806           +0.9817j            1.0210
0.2057
AR.7             0.7454           -0.8502j            1.1307           -
0.1354
AR.8             0.7454           +0.8502j            1.1307
0.1354
AR.9             1.1879           -0.3562j            1.2402           -
0.0464
AR.10            1.1879           +0.3562j            1.2402
0.0464
MA.1             0.3664           -0.9304j            1.0000           -
0.1903
MA.2             0.3664           +0.9304j            1.0000
0.1903
-----------------------------------------------------------------------------
-----
```

Lowest AIC for model: -289.473

#LSTM

In [153]:

```
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [155]:

```
np.random.seed(7)
```

In [157]:

```
ts_log=ts_log.astype('float32')
```

In [158]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
ts_log = scaler.fit_transform(ts_log)
```

In [175]:
```
train_size = int(len(ts_log) * 0.67)
test_size = len(ts_log) - train_size
train, test = ts_log[0:train_size,:], ts_log[train_size:len(ts_log),:]
print(len(train), len(test))
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

```
42 21
```

In [176]:
```
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

In [185]:
```

start = time.clock()
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
print(time.clock() - start)
```

```
Epoch 1/100
0s - loss: 0.1790
Epoch 2/100
0s - loss: 0.1391
Epoch 3/100
0s - loss: 0.1080
Epoch 4/100
0s - loss: 0.0827
Epoch 5/100
0s - loss: 0.0626
Epoch 6/100
0s - loss: 0.0492
Epoch 7/100
0s - loss: 0.0397
Epoch 8/100
0s - loss: 0.0343
Epoch 9/100
0s - loss: 0.0306
Epoch 10/100
0s - loss: 0.0286
```

In [ ]:
```
```

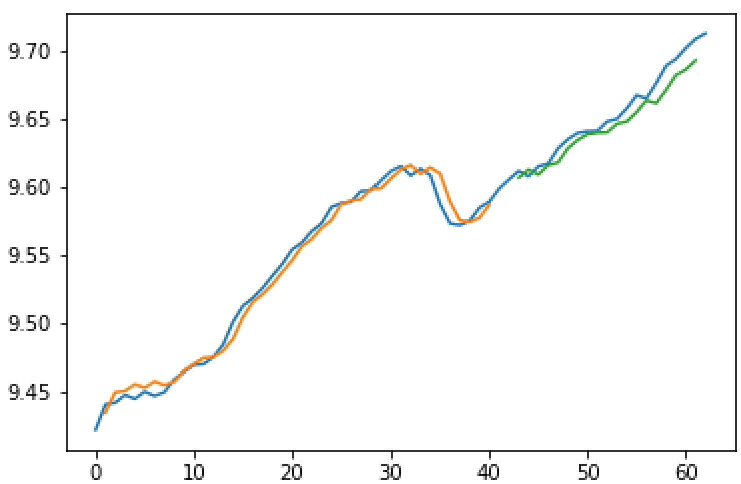
In [168]:

```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

```
Train Score: 0.01 RMSE
Test Score: 0.01 RMSE
```

In [171]:

```
# shift train predictions for plotting
trainPredictPlot = np.empty_like(ts_log)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(ts_log)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(ts_log)-1, :] = testPr
# plot baseline and predictions
plt.plot(scaler.inverse_transform(ts_log))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



As per the timing results, one of the things to take from here is that, ARMA model is faster compared to the LSTM . Also both the models require tuning tje parametes but LSTM requires more preprocessing in the case of tuning the paramenters. As I have not compared the performance of both the model I cannot compare them on the basis of above example. As per my understanding of both the models , I can answer that ARMA is a good model for small dataset and mostly static timeseries. While LSTM works good for larger dataset though it is slower a bit.

In [ ]: