```python
#
# Purdue University Global
#
# IN402 - Modeling and Predictive Analysis
#
# Unit 2 Assignment / Module 2 Competency Assessment
#
# Jupyter Notebook Code
#

# [1] ****************************************************************************************************************

# Data import and wrangling using multiple tools:

import sys

# [2] ****************************************************************************************************************

# For ignoring warning

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")

# [3] ****************************************************************************************************************

import xlrd

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from matplotlib.pylab import rcParams

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.arima_model import ARIMA

# [4] ****************************************************************************************************************

# Load data

xls = pd.ExcelFile("/home/codio/workspace/data/IN402/NATURALGAS.xls")

# In ts, a TimeSeries is the type of index.
# To convert df to ts, make Date column an index

df = xls.parse(0, skiprows=10, index_col=0, na_values=['NA'])

# [5] ****************************************************************************************************************

# Plot the graph

plt.figure(figsize=(10,6))

plt.grid(True)

plt.xlabel('Month')

plt.ylabel('Natural Gas Consumption, Billion Cubic Feet')

plt.plot(df['NATURALGAS'])

plt.title('Natural Gas Consumption, Monthly')

plt.show()

# [6] ****************************************************************************************************************

# Check if the series are stationary

# Determining rolling statistics

rolmean = df.rolling(window = 12).mean()

rolstd = df.rolling(window = 12).std()

rolmean.head(20)

# [7] ****************************************************************************************************************

# plot rolling statistics:

orig = plt.plot(df, color = 'blue', label = 'Original')

mean = plt.plot(rolmean, color = 'red', label = 'Rolling Mean')

std = plt.plot(rolstd, color = 'black', label = 'Rollng Std')

plt.legend(loc = 'best')

plt.title('Rolling Mean & Standard Deviation')

plt.show(block=False)

# [8] ****************************************************************************************************************

# Another option - Dickey-Fuller test
# The Dickey-Fuller test can be used to determine
# the presence of unit root in the series (help us
# understand if the series is stationary)

# Performing Dickey-Fuller test:

from statsmodels.tsa.stattools import adfuller

# [9] ****************************************************************************************************************

dftest = adfuller(df['NATURALGAS'], autolag='AIC') # Akake Information Criterion

dfoutput = pd.Series(dftest[0:4], index=['Test Statistics','p-value','#Lags Used','Number of Observations Used'])

for key, value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value

# [10] ***************************************************************************************************************

print("Results of Dickey-Fuller Test: ")

print(dfoutput)

# Example Output:
# Results of Dickey-Fuller Test:
# Test Statistics                  1.084349
# p-value                          0.995081
# Lags Used                       15.000000
# Number of Observations Used    223.000000
# Critical Value (1%)             -3.460019
# Critical Value (5%)             -2.874590
# Critical Value (10%)            -2.573725
# dtype: float64
#
# Results of Dickey-Fuller test:
#   The p-value is too high;
#   a critical value should be more than the Test Statistics
#   So, we cannot reject the 0 hypothesis and say that the data is non-stationary.

# [11] ***************************************************************************************************************

# Write the code to transform your data using the log of the time series and
# re-calculate the Dickey-Fuller test again on a transformed time series.

# To confirm, let's find the log, estimate the trend and then recalculate
# the moving average again (sometimes instead of a log, you have to take a
# square or cube roots, depends on your time series data)

# Estimating trend

# We have taken the log of the dataset

df_logScale = np.log(df)

plt.plot(df_logScale)

# [12] ***************************************************************************************************************

# Trend remains the same, although the values on y-axis have changed.

# Now, let's calculate moving average

movingAverage = df_logScale.rolling(window=12).mean()

movingSTD = df_logScale.rolling(window=12).std()

plt.plot(df_logScale)

plt.plot(movingAverage, color = 'red')

# [13] ***************************************************************************************************************

# Next, we will determine the difference between the moving average
# and the actual gas consumption

dfScaleMinueMovAvg = df_logScale - movingAverage

dfScaleMinueMovAvg.head(15)

# [14] ***************************************************************************************************************

# Remove NaN values

dfScaleMinueMovAvg.dropna(inplace=True)

dfScaleMinueMovAvg.head(15)

# [15] ***************************************************************************************************************

# Test for stationarity

def test_stationarity(timeseries):

    #Determing rolling statistics

    rolmean = timeseries.rolling(12).mean()

    rolstd = timeseries.rolling(12).std()

    #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 and Standard Deviation')

    plt.show(block=False)

    # Determine Dickey-Fuller:

    print("Results of Dickey-Fuller test")

    adft = adfuller(timeseries,autolag='AIC')

    # output for dft will give us the result without defining what the values are.
    # hence we manually write what values it explains using a for loop

    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags used','Number of observations used'])

    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values

    print(output)

test_stationarity(dfScaleMinueMovAvg)

# Example Output:
# Results of Dickey-Fuller test
# Test Statistics                 -5.533858
# p-value                          0.000002
# No. of lags used                14.000000
# Number of observations used    213.000000
# critical value (1%)             -3.461429
# critical value (5%)             -2.875207
# critical value (10%)            -2.574054
# dtype: float64
#
# Now we can visually notice that there is no trend, and the p-value is much less.
# Also, critical value and Test Statistics values are almost equal, which means your data is now stationary.

# [16] ***************************************************************************************************************

# Next, we'll calculate the weighted average of time series to see the trend that is present inside the time series.

exponentialDecayWeightedAverage = df_logScale.ewm(halflife=12, min_periods=0, adjust=True).mean()

plt.plot(df_logScale)

plt.plot(exponentialDecayWeightedAverage, color = 'red')

df_logScaleMinusMovingExponentialDecayAverage = df_logScale - exponentialDecayWeightedAverage

test_stationarity(df_logScaleMinusMovingExponentialDecayAverage)

# Example Output:
# Results of Dickey-Fuller test
# Test Statistics                 -2.184411
# p-value                          0.211952
# No. of lags used                15.000000
# Number of observations used    223.000000
# critical value (1%)             -3.460019
# critical value (5%)             -2.874590
# critical value (10%)            -2.573725
# dtype: float64
#
# The standard deviation is quite flat and not moving and there is no trend. The rolling mean is a bit better
# than the previous one. The p-value is less than 0.5, so that again confirms that the ts is stationary.

# [17] ***************************************************************************************************************

# Now let's shift the values into time series so that we can use it in forecasting.

df_LogDiffShifting = df_logScale - df_logScale.shift()

plt.plot(df_LogDiffShifting)

# [18] ***************************************************************************************************************

# drop the NA values

df_LogDiffShifting.dropna(inplace=True)

test_stationarity(df_LogDiffShifting)

# Example Output:
# Results of Dickey-Fuller test
# Test Statistics                 -5.430466
# p-value                          0.000003
# No. of lags used                14.000000
# Number of observations used    223.000000
# critical value (1%)             -3.460019
# critical value (5%)             -2.874590
# critical value (10%)            -2.573725
# dtype: float64
#
# The output is flat, no trend, so the null hypothesis is rejected and
# the time series is stationary. Now, let's see the components of Time Series.

# [19] ***************************************************************************************************************

# Based on what you have read in the reading material, decide which decomposition
# method (additive or multiplicative) is applicable and use it to calculate the
# trend-cycle and seasonal indices. Identify any outliers, or unusual features.

from statsmodels.tsa.seasonal import seasonal_decompose

# [20] ***************************************************************************************************************

# Let's separate trend and seasonality from the time series

decomposition = seasonal_decompose(df_logScale)

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

# [21] ***************************************************************************************************************

# Visualize components

plt.subplot(411)

plt.plot(df_logScale, 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')

plt.tight_layout

# The trend is going upward and quite linear in nature. Seasonality is
# present on a high scale. Errors (first plot from the bottom) are irregular in
# nature and by looking at them it is impossible to predict what's going to happen next.

# [22] ***************************************************************************************************************

# Using moving average smoothing method remove fluctuations from a transformed
# time series data. Use a 3-months moving average.

decomposedLogdata = residual

decomposedLogdata.dropna(inplace=True)

test_stationarity(decomposedLogdata)

# Example Output:
# Results of Dickey-Fuller test
# Test Statistics               -7.968812e+00
# p-value                        2.817301e-12
# No. of lags used               1.400000e+01
# Number of observations used    2.120000e+02
# critical value (1%)           -3.461578e+00
# critical value (5%)           -2.875272e+00
# critical value (10%)          -2.574089e+00
# dtype: float64
#
# This is not stationary, that's why we have to have the moving average parameter
# in place so that it's smooth and set out to predict what will happen.

# [23] ***************************************************************************************************************

# Now you know the value of P (autoregressive lags) and Q (value of moving average),
# so we need to plot PACF plot to calculate the values of P and ACF plot to
# calculate the value of Q.

# ACF and PACF plots:

from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(df_LogDiffShifting, nlags=20)

lag_pacf = pacf(df_LogDiffShifting, nlags=20, method='ols')

# [24] ***************************************************************************************************************

# ACF

plt.subplot(121)

plt.plot(lag_acf)

plt.axhline(y=0,linestyle='--', color='gray')

plt.axhline(y=-1.96/np.sqrt(len(df_LogDiffShifting)), linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(df_LogDiffShifting)), linestyle='--',color='gray')

plt.title("Autocorrelation Function")

# [25] ***************************************************************************************************************

# PACF

plt.subplot(122)

plt.plot(lag_pacf)

plt.axhline(y=0,linestyle='--', color='gray')

plt.axhline(y=-1.96/np.sqrt(len(df_LogDiffShifting)), linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(df_LogDiffShifting)), linestyle='--',color='gray')

plt.title("PartialAutocorrelation Function")

# ********************************************************************************************************************
```