**Unit 2 Assignment**

Laurence T. Burden

Purdue Global University

IN402: Modeling and Predictive Analysis

Steward Huang

November 20, 2023

The Python code for this unit analyzes natural gas consumption data from the years 2000 to 2023 collected by the U.S. Bureau of Transportation (FRED, 2023). The data is tested multiple times using the Dickey-Fuller test to ensure the data is stationary. Multiple transformations are conducted to evaluate the data throughout.

The first step for this analysis is to import the Pandas, Numpy, Matplotlib Pyplot, and XLRD packages. The data is loaded from a spreadsheet and applies the date column as an index to convert the data frame to a time series. Exploratory analysis is conducted by graphing the initial data and finding the rolling 12-month average. The rolling mean and standard deviations are then plotted.

The Dickey-Fuller test is then set up. The Akake Information Criterion is used for the autolag parameter. The Python code outputs the test statistics, p-value, number of lags used, and the number of observations used for each Dickey-Fuller test. The critical values are added by separating the keys using a formatted string.

The first Dickey-Fuller test gives a result that prevents us from rejecting the null hypothesis. This is because the p-value is too high and the critical values are all lower than the test statistics. This leads us to believe the data is non-stationary.

Next, the log of the data is found using Numpy’s log method. The manipulated information is then plotted. The moving average of the data is found with a window of 12 months. A function is then defined to test for stationarity. This function contains the ability to plot the rolling mean and rolling standard deviation. It also contains the same setup for a Dickey-Fuller test as before.

The code then finds the difference between the moving average and the actual gas consumption. The data set is then cleaned by removing all NaN values. Another Dickey-Fuller test is then completed.

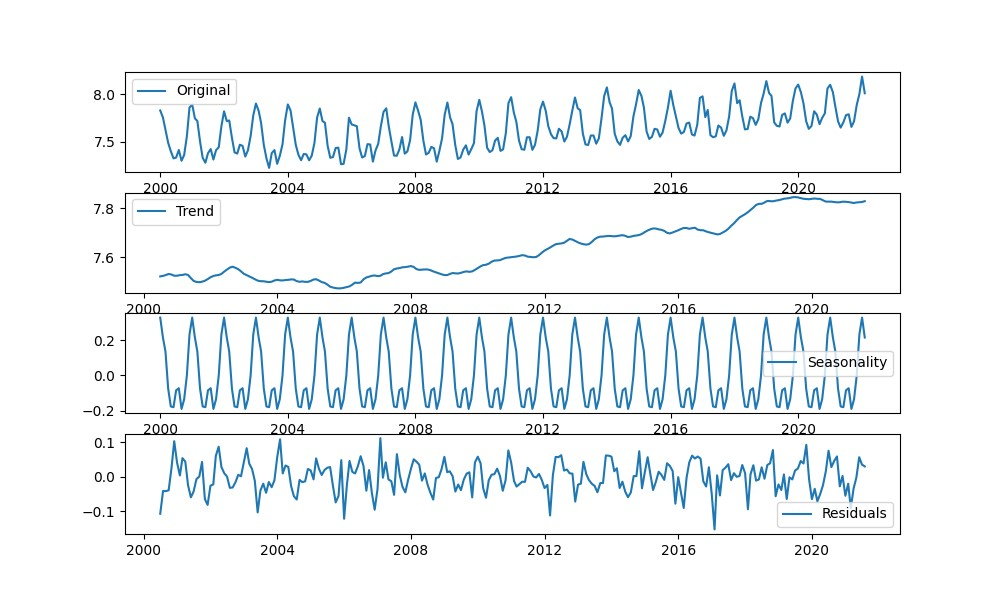
A weighted average is created next. This is accomplished with the ewm method and the mean method. The objective of this process is to see the trend that is present within the data. A plot of the information is printed to show this trend. A Dickey-Fuller test on this latest version of the data shows a stationary data set.

Next, each value in the data frame is subtracted by the next value in the series using the shift method. This is done using the default of the shift function, which moves all data down by one row. Another Dickey-Fuller test is completed. This results in a flat and trendless data set. The null hypothesis is rejected and the series is considered stationary. Next, the series will be separated into its components.

There are four components of time series data: trends, seasons, cycles, and irregular components (Anderson & Semmelroth, 2015). The code breaks the data down to three components: trend, season, and residual. The data is decomposed using an additive decomposition method. These three components are then plotted to external images.

**Figure 1**

*Plot of the Three Components Decomposed*



These charts show that the depressions of 2002 and 2008-2010 had little to no effect on the use of natural gas in this data set. This is most likely due to natural gas being a needed commodity that is needed during any economic time period. Citizens and companies will always need natural gas as long as it is used to heat our homes.

The final section of the code plots the autocorrelation and partial-autocorrelation of the data. This is done using the acf and pacf methods and the log data that has been subtracted by the shifted data frame. The number of lags used are 20 for both methods and the “ols” method is used for the partial-autocorrelation.

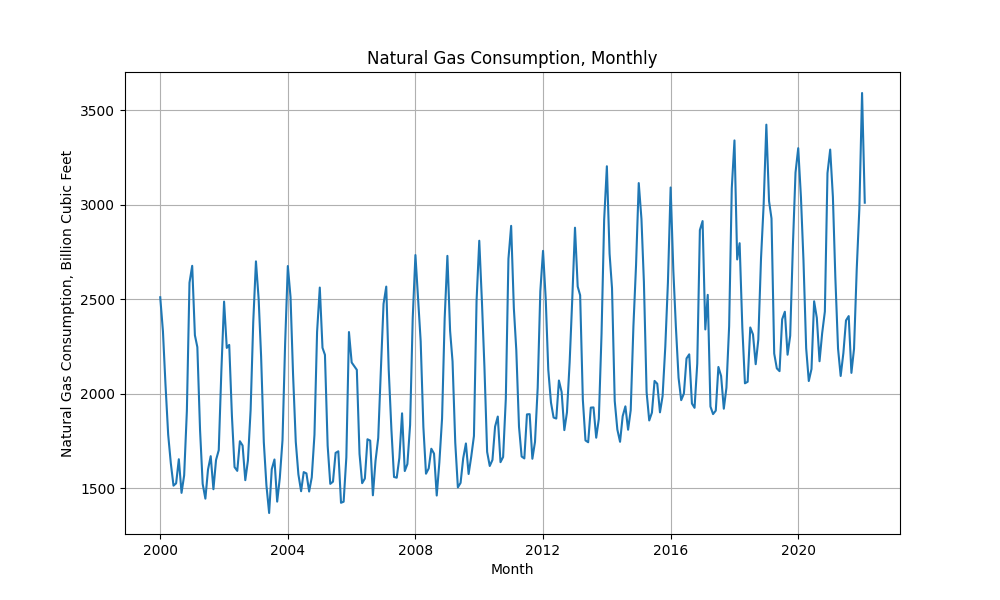
**References**

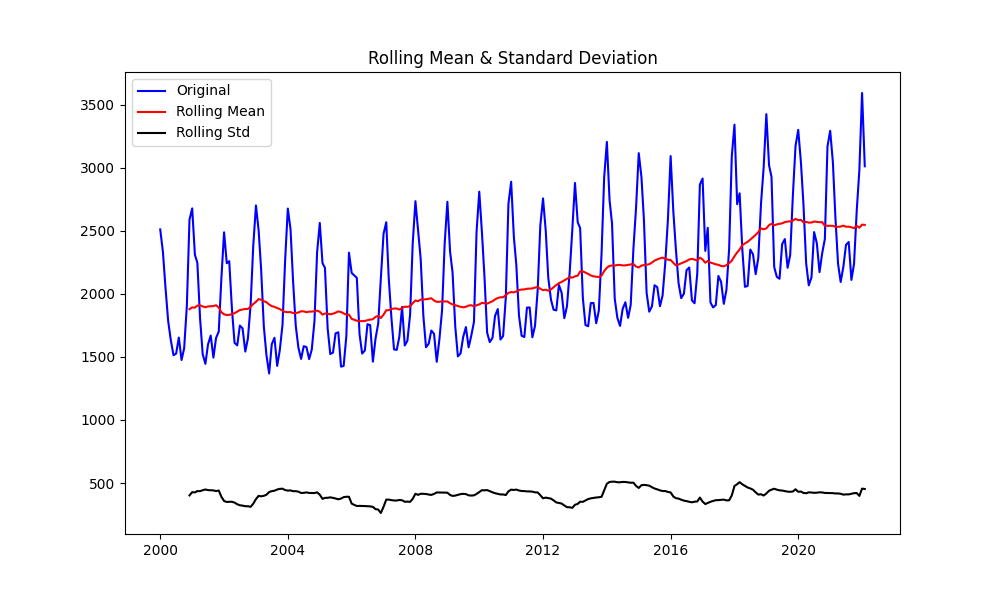
*Natural Gas Consumption*. FRED. (2023, October 12). <https://fred.stlouisfed.org/series/NATURALGAS>

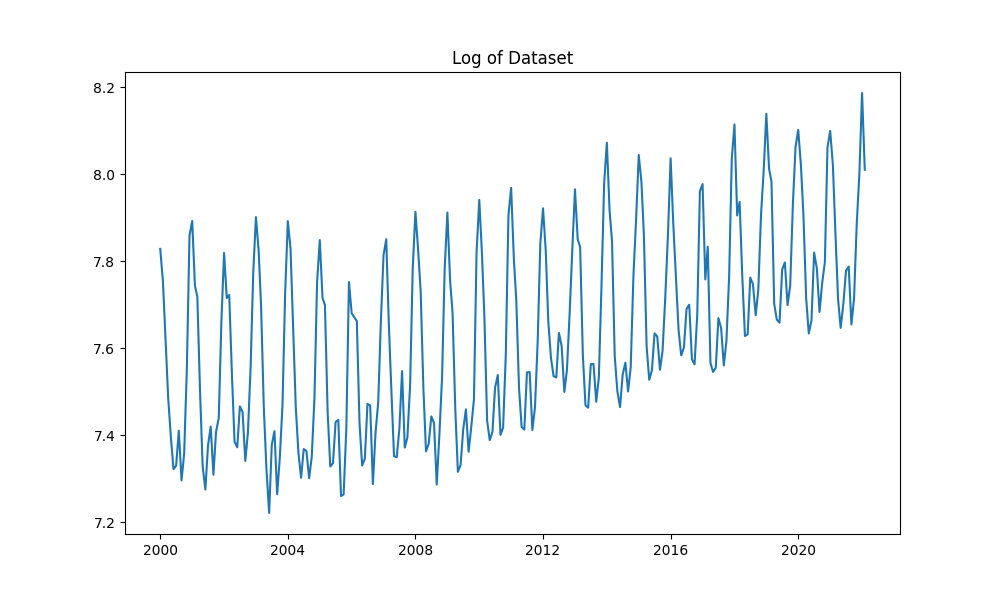
Anderson, A., & Semmelroth, D. (2015). *Statistics for big data for dummies*. Wil & Sons, Inc.

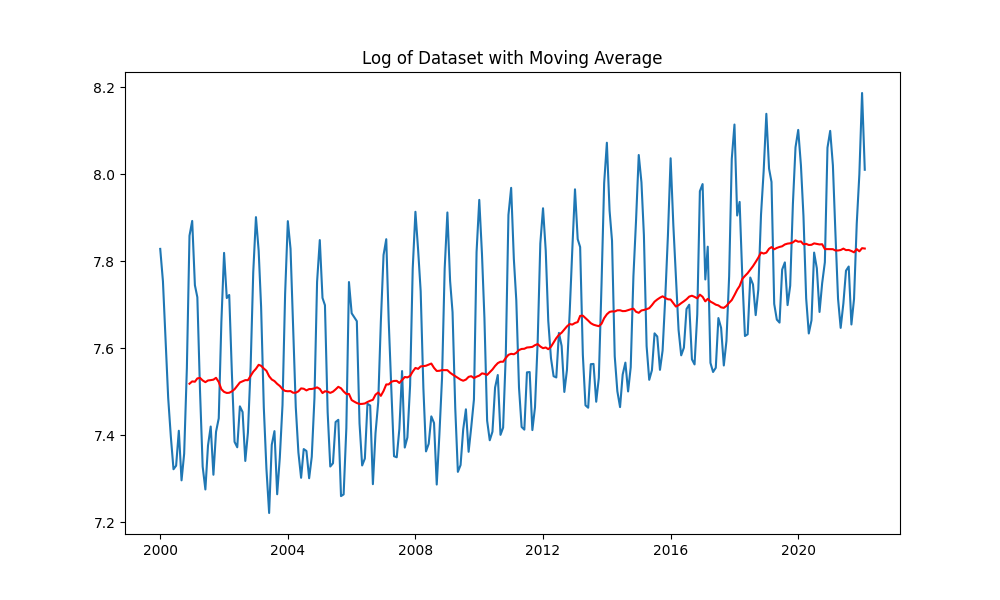
**Appendix A**

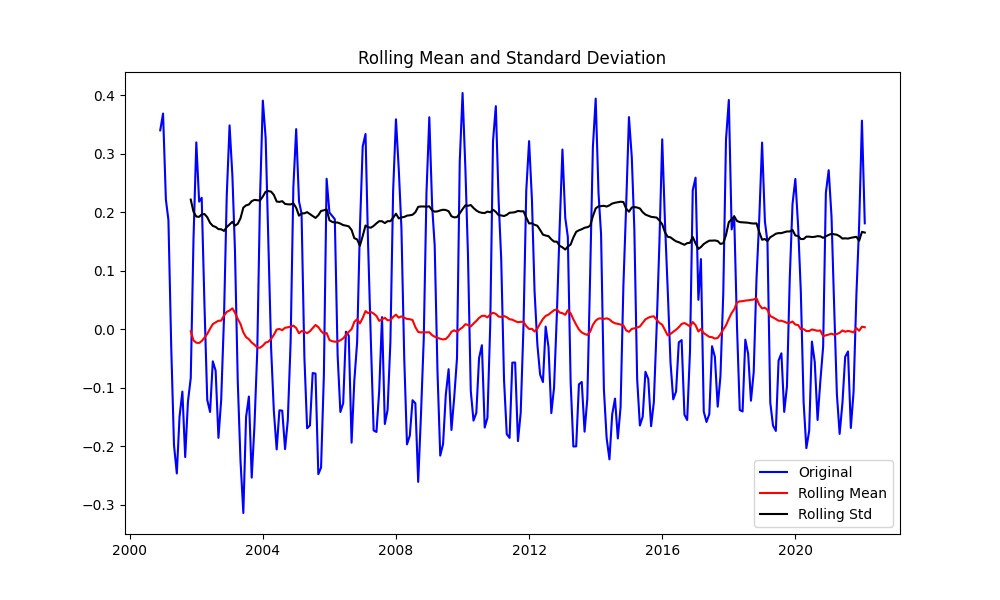
Python Plots

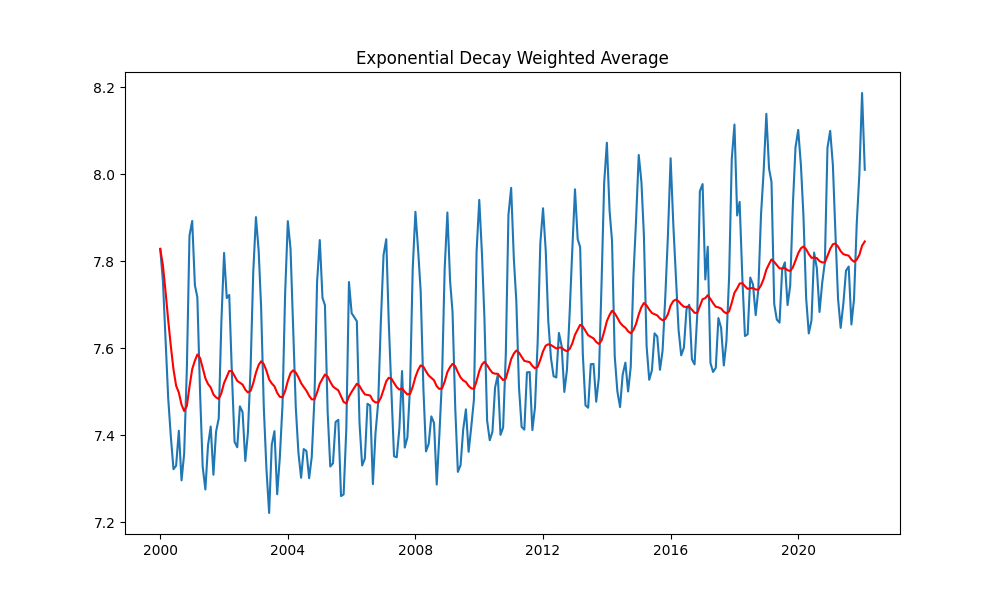


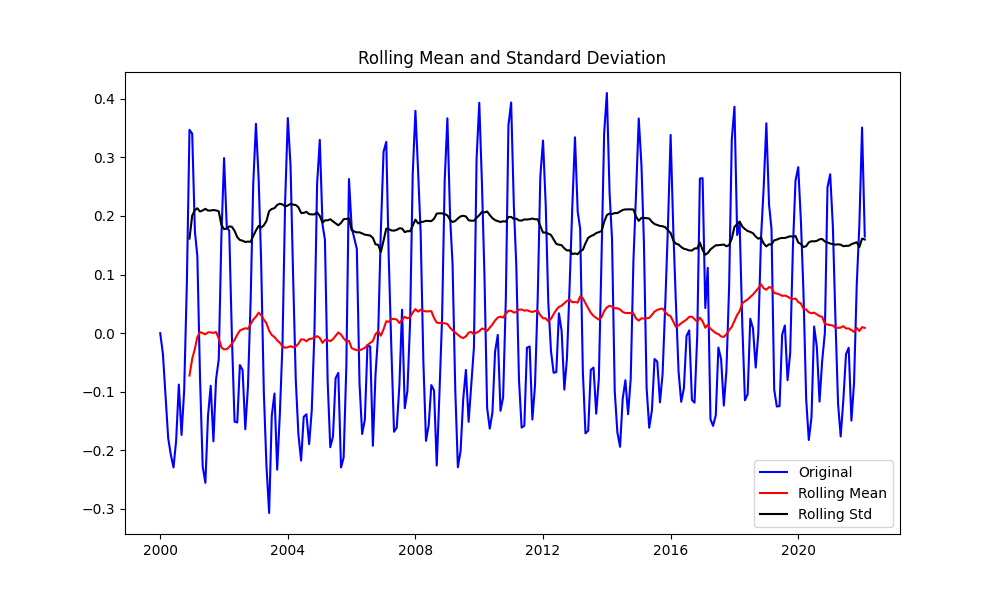


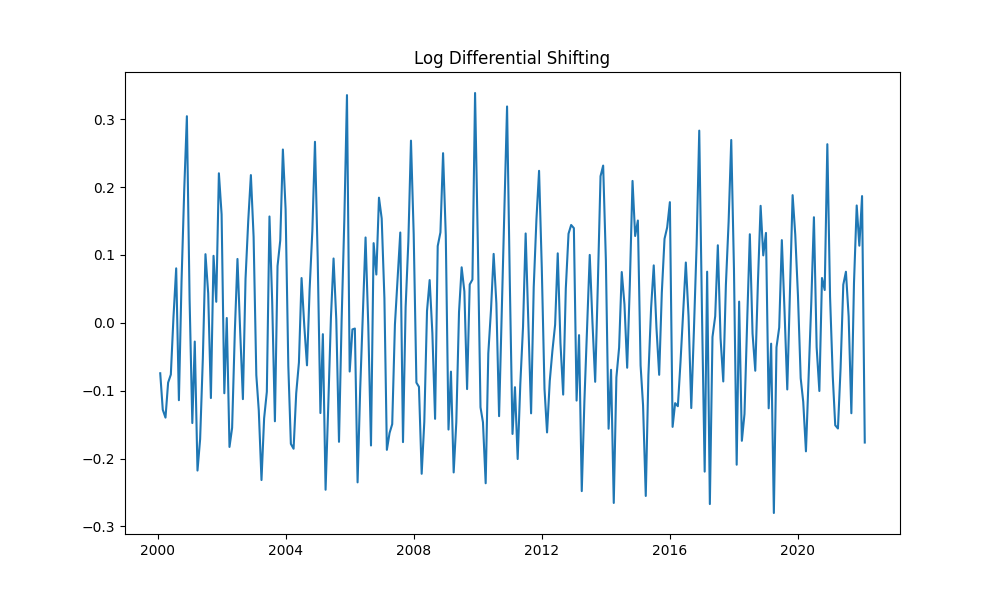


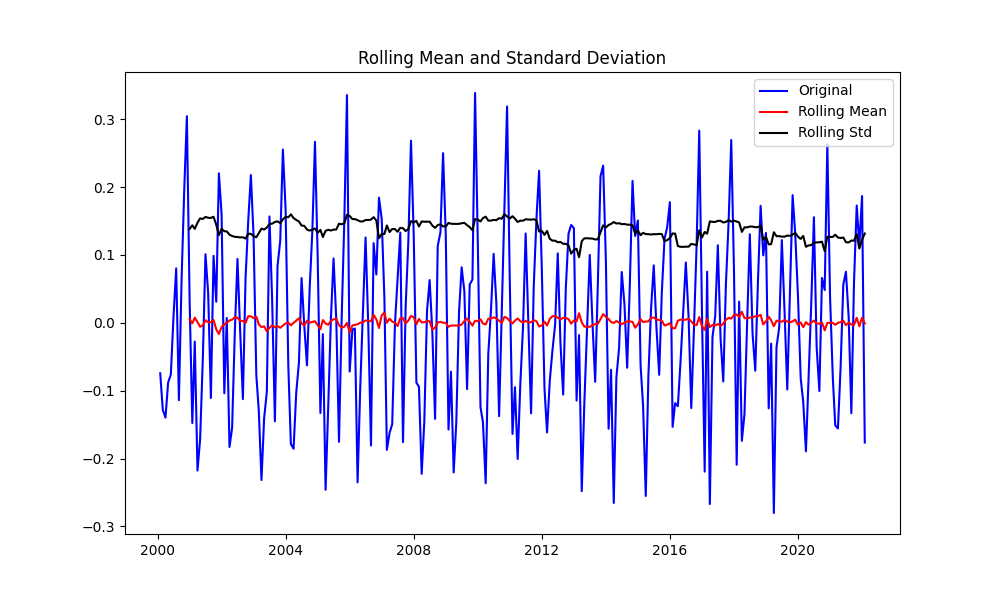


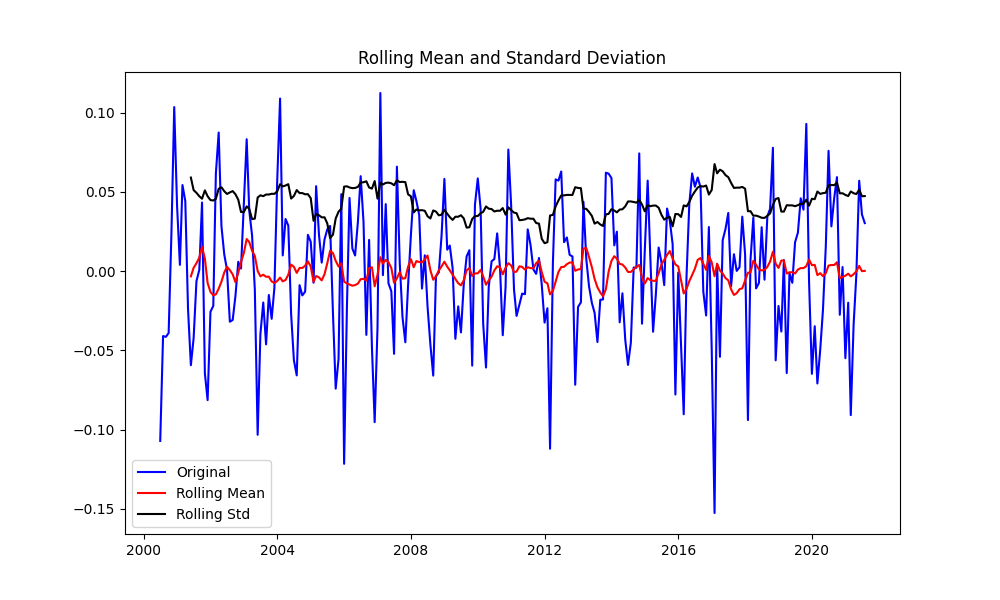


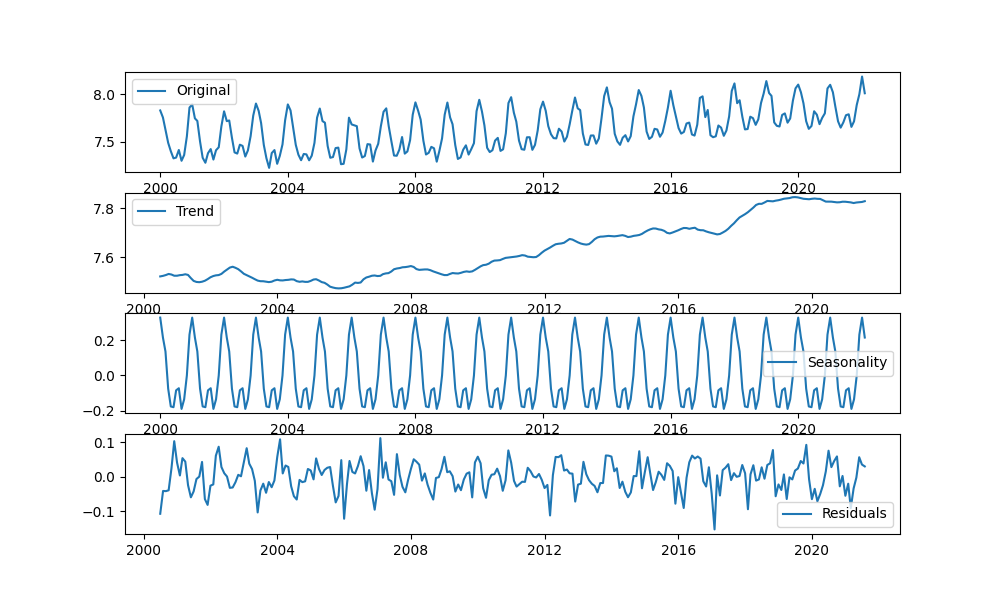


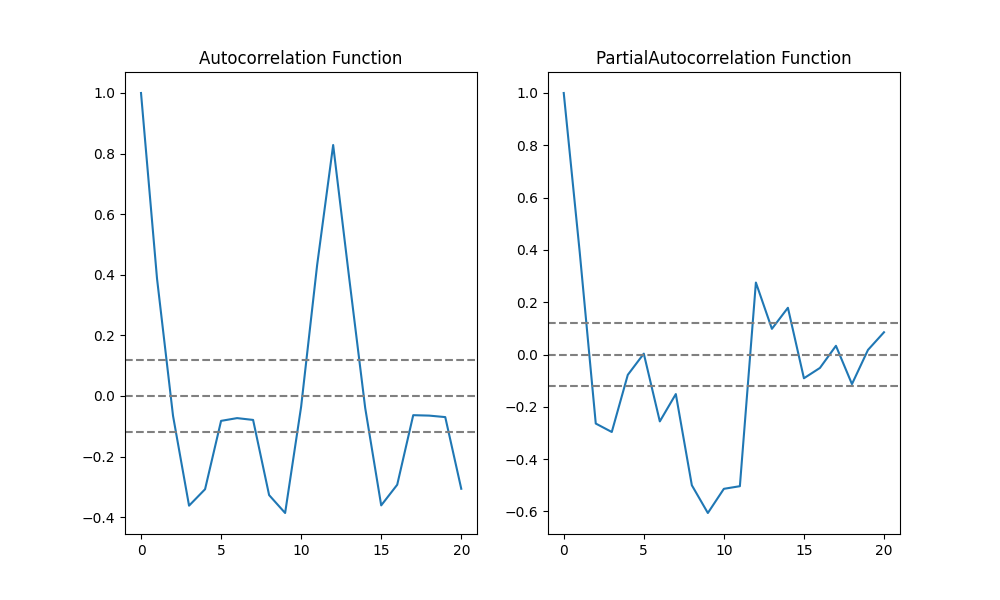












**Appendix B**

Python Code

#

# Laurence Burden for Purdue University Global

#

# Unit 2 Assignment / Module 2 Competency Assessment

#

# Imports

import sys

# Ignore warnings

if not sys.warnoptions:

import warnings

warnings.simplefilter("ignore")

import xlrd

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Output header

print('Unit 2 Assignment / Module 2 Competency Assessment Output\n')

from datetime import datetime

print(datetime.now().strftime("%m/%d/%Y %H:%M:%S"), '\n')

# 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'])

# 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()

# Check if the series are stationary

# Determining Rolling Statistics

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

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

print("ROLLING 12-MONTH MEAN")

print(rolmean.head(20))

# Plot rolling statistics

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

orig = plt.plot(df, 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()

# Another option - Dickey-Fuller test

# The Dickey-Fuller test can be used to determine

# the presence of unit root in the series (helps us

# understand if the series is stationary)

# Performing Dickey-Fuller test

from statsmodels.tsa.stattools import adfuller

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

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

print(dfoutput)

print()

# Transform data using the log of the time series and

# re-calculate the Dickey-Fuller test again on the transformed data

# Estimating trend

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

df\_logScale = np.log(df)

plt.title('Log of Dataset')

plt.plot(df\_logScale)

plt.show()

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

# Next, calculate moving average

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

movingAverage = df\_logScale.rolling(window=12).mean()

movingSTD = df\_logScale.rolling(window=12).std()

plt.title('Log of Dataset with Moving Average')

plt.plot(df\_logScale)

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

plt.show()

# Test for stationarity

def test\_stationarity(timeseries):

#Determining rolling statistics

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

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

# Plot rolling statistics

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

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()

# Determine Dickey-Fuller

print("Results of Dickey-Fuller test")

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

# Output for dft will give 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)

print()

# determine the difference between the moving average

# and the actual gas consumption

dfScaleMinueMovAvg = df\_logScale - movingAverage

print('Original Moving Average')

print(dfScaleMinueMovAvg.head(15))

print()

# Remove NaN values

dfScaleMinueMovAvg.dropna(inplace=True)

print('Removed NAs')

print(dfScaleMinueMovAvg.head(15))

print()

test\_stationarity(dfScaleMinueMovAvg)

# Calculate the weighted average of the time series to see the trend that is present

exponentialDecayWeightedAverage = df\_logScale.ewm(halflife=12, min\_periods=0, adjust=True).mean()

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

plt.plot(df\_logScale)

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

plt.title('Exponential Decay Weighted Average')

plt.show()

df\_logScaleMinusMovingExponentialDecayAverage = df\_logScale - exponentialDecayWeightedAverage

test\_stationarity(df\_logScaleMinusMovingExponentialDecayAverage)

# Shift the values into time series so that we can use it in forecasting

df\_LogDiffShifting = df\_logScale - df\_logScale.shift()

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

plt.plot(df\_LogDiffShifting)

plt.title('Log Differential Shifting')

plt.show()

# Drop the NA values

df\_LogDiffShifting.dropna(inplace=True)

test\_stationarity(df\_LogDiffShifting)

from statsmodels.tsa.seasonal import seasonal\_decompose

# Seperate trend and seasonality from the time series using additive decomposition

decomposition = seasonal\_decompose(df\_logScale)

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

# Visualize components

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

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

# Use 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)

# 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')

# ACF

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

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")

# 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")

plt.show()

**Appendix C**

Output of Python Code

Unit 2 Assignment / Module 2 Competency Assessment Output

11/20/2023 22:25:32

ROLLING 12-MONTH MEAN

NATURALGAS

observation\_date

2000-01-01 NaN

2000-02-01 NaN

2000-03-01 NaN

2000-04-01 NaN

2000-05-01 NaN

2000-06-01 NaN

2000-07-01 NaN

2000-08-01 NaN

2000-09-01 NaN

2000-10-01 NaN

2000-11-01 NaN

2000-12-01 1878.225000

2001-01-01 1892.100000

2001-02-01 1890.333333

2001-03-01 1906.666667

2001-04-01 1908.658333

2001-05-01 1899.450000

2001-06-01 1893.725000

2001-07-01 1899.766667

2001-08-01 1901.108333

Results of Dickey-Fuller Test:

Test Statistics 0.073674

p-value 0.964217

#Lags Used 14.000000

Number of Observations Used 251.000000

Critical Value (1%) -3.456674

Critical Value (5%) -2.873125

Critical Value (10%) -2.572944

dtype: float64

Original Moving Average

NATURALGAS

observation\_date

2000-01-01 NaN

2000-02-01 NaN

2000-03-01 NaN

2000-04-01 NaN

2000-05-01 NaN

2000-06-01 NaN

2000-07-01 NaN

2000-08-01 NaN

2000-09-01 NaN

2000-10-01 NaN

2000-11-01 NaN

2000-12-01 0.340008

2001-01-01 0.368661

2001-02-01 0.221757

2001-03-01 0.186537

Removed NAs

NATURALGAS

observation\_date

2000-12-01 0.340008

2001-01-01 0.368661

2001-02-01 0.221757

2001-03-01 0.186537

2001-04-01 -0.032212

2001-05-01 -0.197863

2001-06-01 -0.246585

2001-07-01 -0.149333

2001-08-01 -0.106611

2001-09-01 -0.218504

2001-10-01 -0.124006

2001-11-01 -0.083428

2001-12-01 0.153465

2002-01-01 0.319388

2002-02-01 0.218073

Results of Dickey-Fuller test

Test Statistics -5.588702

p-value 0.000001

No. of lags used 14.000000

Number of observations used 240.000000

critical value (1%) -3.457894

critical value (5%) -2.873659

critical value (10%) -2.573229

dtype: float64

Results of Dickey-Fuller test

Test Statistics -2.719708

p-value 0.070690

No. of lags used 14.000000

Number of observations used 251.000000

critical value (1%) -3.456674

critical value (5%) -2.873125

critical value (10%) -2.572944

dtype: float64

Results of Dickey-Fuller test

Test Statistics -5.564340

p-value 0.000002

No. of lags used 13.000000

Number of observations used 251.000000

critical value (1%) -3.456674

critical value (5%) -2.873125

critical value (10%) -2.572944

dtype: float64

Results of Dickey-Fuller test

Test Statistics -8.171637e+00

p-value 8.591269e-13

No. of lags used 1.400000e+01

Number of observations used 2.390000e+02

critical value (1%) -3.458011e+00

critical value (5%) -2.873710e+00

critical value (10%) -2.573256e+00

dtype: float64