In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
from statsmodels.tsa.arima_model import ARIMA

%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize']=20,12
```

In [3]:
```python
dataset= pd.read_csv('TimeSeries_Gediz_no_null_trainset.csv')
dataset['DateAndTime']=pd.to_datetime(dataset['DateAndTime'],infer_datetime_forma
indexedDataset=dataset.set_index(['DateAndTime'])
```

In [4]:
```python
from datetime import datetime
indexedDataset
```

Out[4]:

	Total
DateAndTime	
2021-05-01 00:00:00	164
2021-05-01 00:30:00	66
2021-05-01 01:00:00	9
2021-05-01 01:30:00	2
2021-05-01 02:00:00	5
...	...
2022-06-14 21:30:00	283
2022-06-14 22:00:00	184
2022-06-14 22:30:00	166
2022-06-14 23:00:00	102
2022-06-14 23:30:00	80

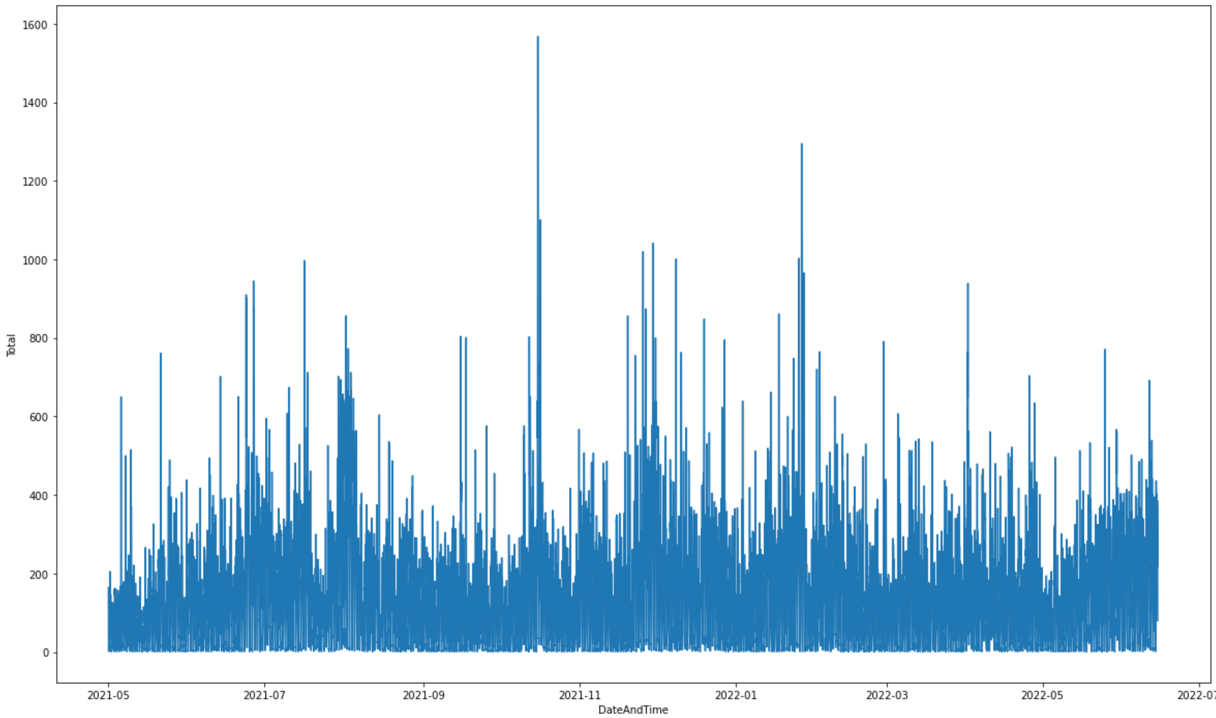
19487 rows × 1 columns

In [5]:
```python
len(indexedDataset)
```

Out[5]: 19487

In [98]:
```
# plot graph
plt.xlabel("DateAndTime")
plt.ylabel("Total")
plt.plot(indexedDataset)
```

Out[98]:
```
[<matplotlib.lines.Line2D at 0x1d281707c10>]
```



In [6]:
```
rolmean=indexedDataset.rolling(window=12).mean()
rolstd=indexedDataset.rolling(window=12).std()
```

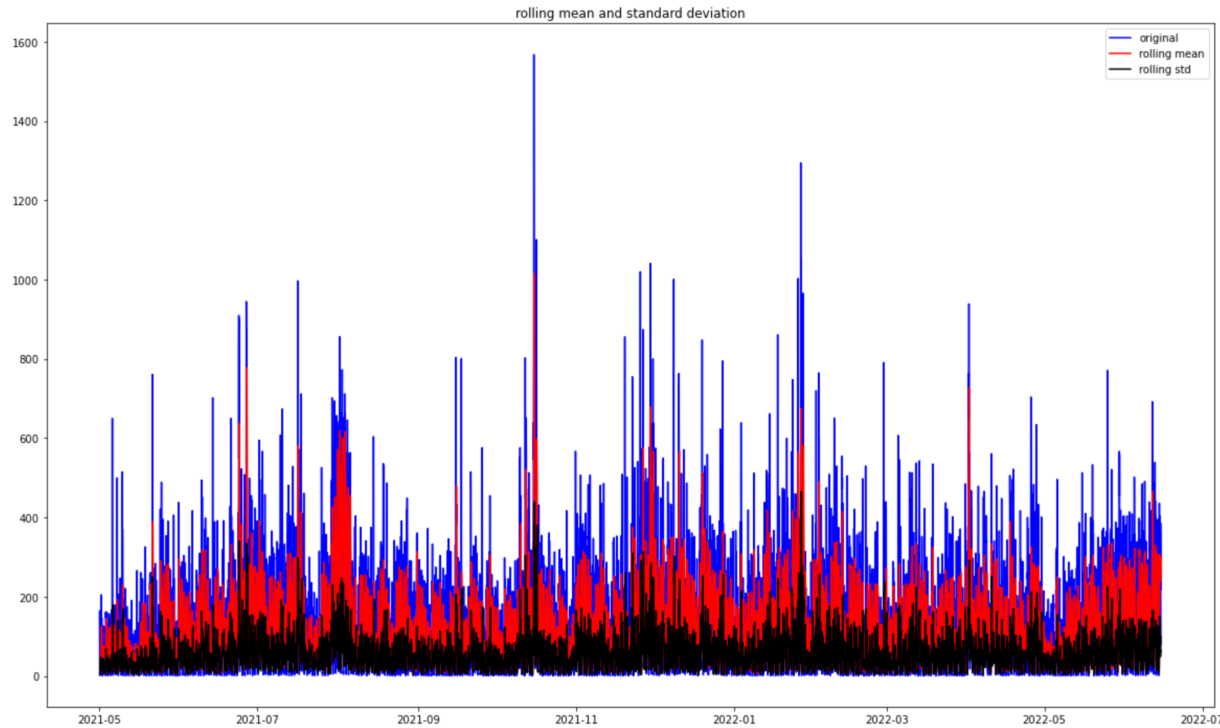
In [7]:
```
print(rolmean)
```

```
                          Total
DateAndTime
2021-05-01 00:00:00         NaN
2021-05-01 00:30:00         NaN
2021-05-01 01:00:00         NaN
2021-05-01 01:30:00         NaN
2021-05-01 02:00:00         NaN
...                         ...
2022-06-14 21:30:00  299.250000
2022-06-14 22:00:00  292.166667
2022-06-14 22:30:00  283.750000
2022-06-14 23:00:00  261.333333
2022-06-14 23:30:00  240.333333

[19487 rows x 1 columns]
```

In [8]:
```
print(rolstd)
```

```
                         Total
DateAndTime
2021-05-01 00:00:00        NaN
2021-05-01 00:30:00        NaN
2021-05-01 01:00:00        NaN
2021-05-01 01:30:00        NaN
2021-05-01 02:00:00        NaN
...                        ...
2022-06-14 21:30:00  61.430116
2022-06-14 22:00:00  69.593408
2022-06-14 22:30:00  78.456850
2022-06-14 23:00:00  88.984507
2022-06-14 23:30:00  99.862026

[19487 rows x 1 columns]
```

In [9]:
```
orig=plt.plot(indexedDataset, 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)
```



In [10]:
```
print('Dataset is Stationary according to statistics of mean and standard deviati
```

```
Dataset is Stationary according to statistics of mean and standard deviation.
```

In [11]:
```python
indexedDataset['Total']=indexedDataset['Total'].fillna(0.0)
```

In [12]:
```python
indexedDataset
```

Out[12]:

	Total
DateAndTime	
2021-05-01 00:00:00	164
2021-05-01 00:30:00	66
2021-05-01 01:00:00	9
2021-05-01 01:30:00	2
2021-05-01 02:00:00	5
...	...
2022-06-14 21:30:00	283
2022-06-14 22:00:00	184
2022-06-14 22:30:00	166
2022-06-14 23:00:00	102
2022-06-14 23:30:00	80

19487 rows × 1 columns

In [13]:
```python
# Perform Dickey-Fuller Test

from statsmodels.tsa.stattools import adfuller

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

dftest=adfuller(indexedDataset['Total'], autolag='AIC')
dfoutput=pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', 'Lags Used',

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

print(dfoutput)
```

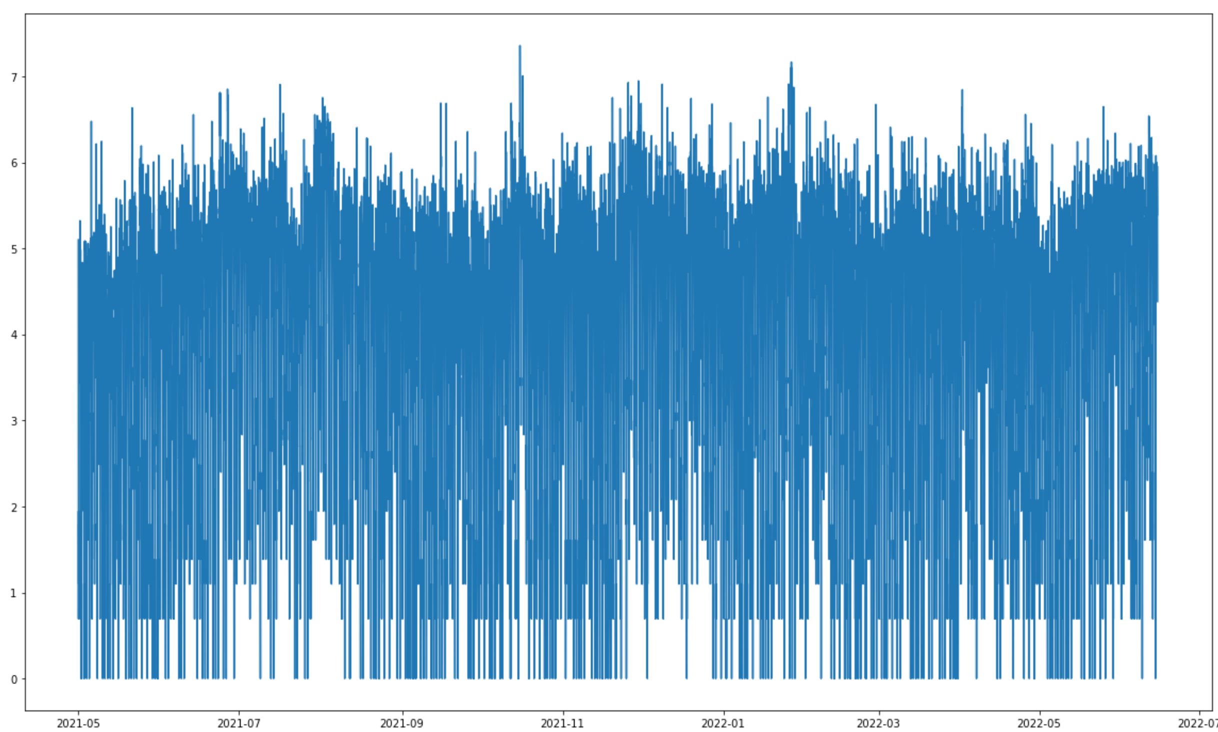
```
Results of Dickey-Fuller Test :
Test Statistic                -8.020799e+00
p-value                        2.078948e-12
Lags Used                      4.500000e+01
Number of Observations Used    1.944100e+04
Critical value (1%)           -3.430686e+00
Critical value (5%)           -2.861689e+00
Critical value (10%)          -2.566849e+00
dtype: float64
```

In [15]:
```
#Estimating Trend (scale has changed)
indexedDataset_logscale=np.log(indexedDataset)
plt.plot(indexedDataset_logscale)

indexedDataset_logscale
```

Out[15]:

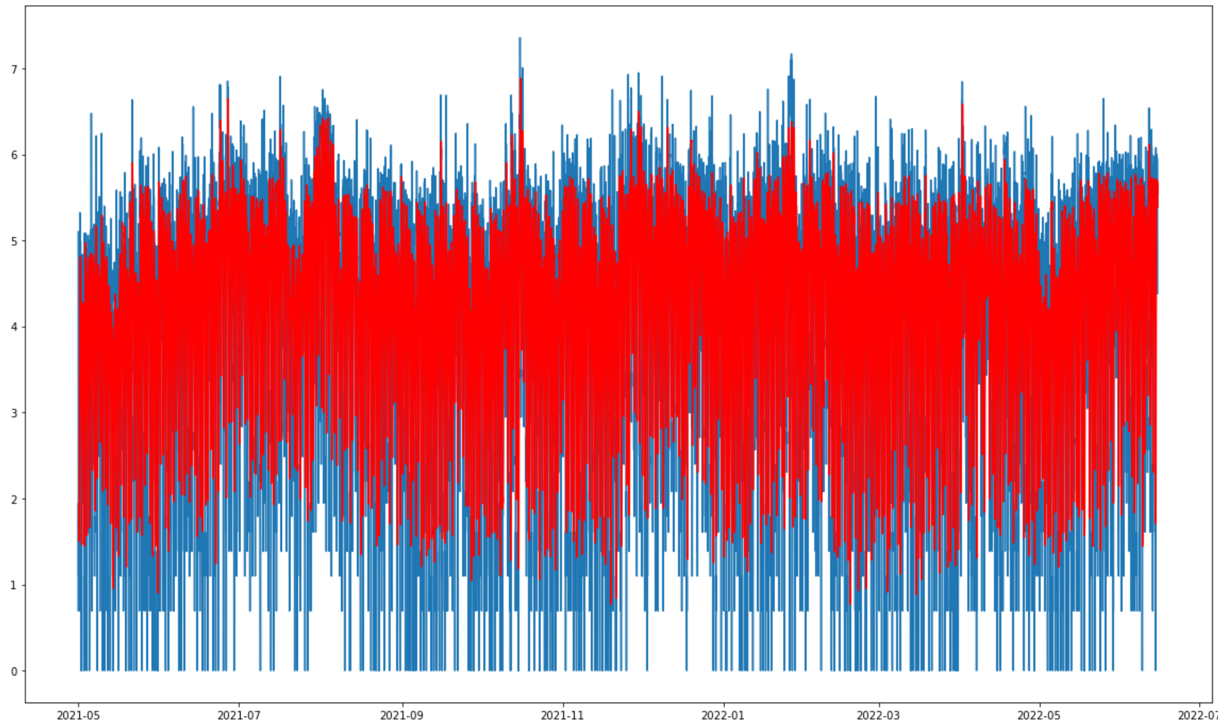
	Total
DateAndTime	
2021-05-01 00:00:00	5.099866
2021-05-01 00:30:00	4.189655
2021-05-01 01:00:00	2.197225
2021-05-01 01:30:00	0.693147
2021-05-01 02:00:00	1.609438
...	...
2022-06-14 21:30:00	5.645447
2022-06-14 22:00:00	5.214936
2022-06-14 22:30:00	5.111988
2022-06-14 23:00:00	4.624973
2022-06-14 23:30:00	4.382027

19487 rows × 1 columns



In [16]:
```
#Taking MA (Moving Average Model)

movingAverage=indexedDataset_logscale.rolling(window=12).mean()
movingSTD=indexedDataset_logscale.rolling(window=12).std()
plt.plot(indexedDataset_logscale)
plt.plot(movingAverage, color='red')
```

Out[16]: [<matplotlib.lines.Line2D at 0x213b90bb7f0>]



In [17]:
```
datasetLogScaleMinusMovingAverage=indexedDataset_logscale - movingAverage
```

In [18]:
```
movingAverage
```

Out[18]:

	Total
DateAndTime	
2021-05-01 00:00:00	Nan
2021-05-01 00:30:00	Nan
2021-05-01 01:00:00	Nan
2021-05-01 01:30:00	Nan
2021-05-01 02:00:00	Nan
...	...
2022-06-14 21:30:00	5.681851
2022-06-14 22:00:00	5.650203
2022-06-14 22:30:00	5.610598
2022-06-14 23:00:00	5.502995
2022-06-14 23:30:00	5.384403

19487 rows × 1 columns

In [19]:

```
#Remove Nan Values

datasetLogScaleMinusMovingAverage = datasetLogScaleMinusMovingAverage.dropna()
datasetLogScaleMinusMovingAverage.head(50)
```

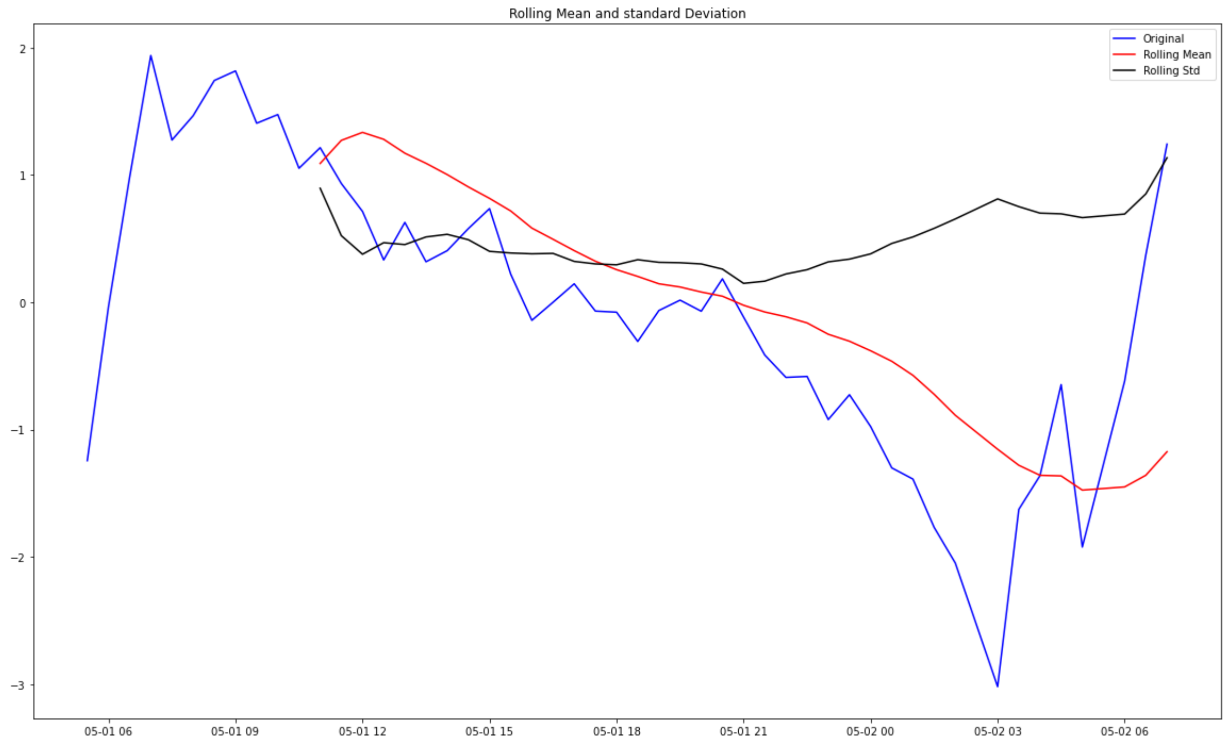
Out[19]:

	Total
DateAndTime	
2021-05-01 05:30:00	-1.242751
2021-05-01 06:00:00	-0.035592
2021-05-01 06:30:00	0.981939
2021-05-01 07:00:00	1.939204
2021-05-01 07:30:00	1.275074
2021-05-01 08:00:00	1.465973
2021-05-01 08:30:00	1.742746
2021-05-01 09:00:00	1.817974
2021-05-01 09:30:00	1.407211
2021-05-01 10:00:00	1.474996
2021-05-01 10:30:00	1.053285
2021-05-01 11:00:00	1.215154
2021-05-01 11:30:00	0.933801
2021-05-01 12:00:00	0.714616
2021-05-01 12:30:00	0.333379
2021-05-01 13:00:00	0.628495
2021-05-01 13:30:00	0.319151
2021-05-01 14:00:00	0.406339
2021-05-01 14:30:00	0.579644
2021-05-01 15:00:00	0.736739
2021-05-01 15:30:00	0.222046
2021-05-01 16:00:00	-0.140846
2021-05-01 16:30:00	0.001303
2021-05-01 17:00:00	0.146088
2021-05-01 17:30:00	-0.068272
2021-05-01 18:00:00	-0.076969
2021-05-01 18:30:00	-0.306747
2021-05-01 19:00:00	-0.063684
2021-05-01 19:30:00	0.017809
2021-05-01 20:00:00	-0.069340

	Total
DateAndTime	
2021-05-01 20:30:00	0.185176
2021-05-01 21:00:00	-0.115288
2021-05-01 21:30:00	-0.413083
2021-05-01 22:00:00	-0.589069
2021-05-01 22:30:00	-0.581440
2021-05-01 23:00:00	-0.920316
2021-05-01 23:30:00	-0.724938
2021-05-02 00:00:00	-0.974855
2021-05-02 00:30:00	-1.299527
2021-05-02 01:00:00	-1.387588
2021-05-02 01:30:00	-1.766426
2021-05-02 02:00:00	-2.046884
2021-05-02 03:00:00	-3.017877
2021-05-02 03:30:00	-1.625402
2021-05-02 04:00:00	-1.360564
2021-05-02 04:30:00	-0.645488
2021-05-02 05:00:00	-1.920981
2021-05-02 06:00:00	-0.617641
2021-05-02 06:30:00	0.372794
2021-05-02 07:00:00	1.243077

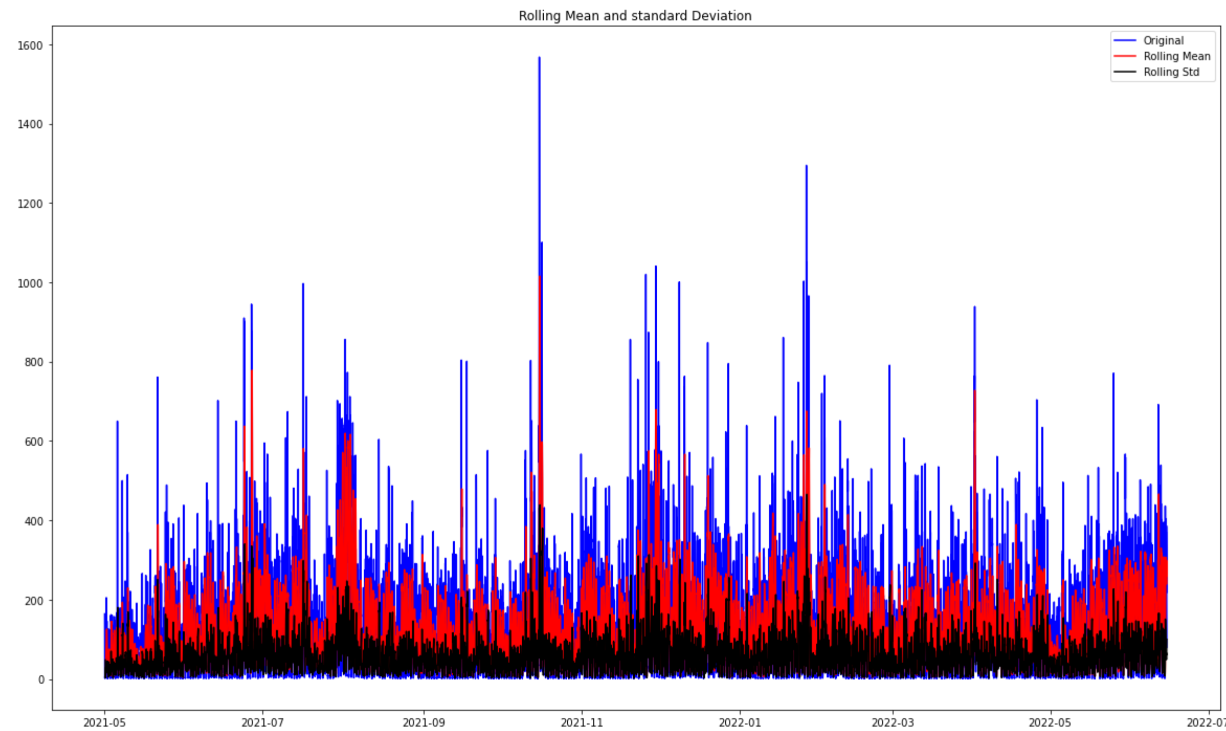
In [20]:
```
#DCF Test (Augmented Dickey-Fuller (ADF) Test)
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    movingAverage=timeseries.rolling(window=12).mean()
    movingSTD=timeseries.rolling(window=12).std()

    #plot rolling statistics
    orig = plt.plot(timeseries, color='blue', label='Original')
    mean =plt.plot(movingAverage, color='red', label='Rolling Mean')
    std = plt.plot(movingSTD, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and standard Deviation')
    plt.show(block=False)

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

    movingAverage
    movingSTD
    dftest
    dfoutput
    datasetLogScaleMinusMovingAverage
```

In [115]:
```
test_stationarity(datasetLogScaleMinusMovingAverage.head(50))
```




```
Results of Dickey-fuller Test
Test Statistic                 -2.417712
p-value                         0.136800
Lags Used                       4.000000
Number of Observations Used    45.000000
Critical value (1%)            -3.584829
Critical value (5%)            -2.928299
Critical value (10%)           -2.602344
dtype: float64
```

In [21]:
```
test_stationarity(indexedDataset)
```




```
Results of Dickey-fuller Test
Test Statistic                -8.020799e+00
p-value                        2.078948e-12
Lags Used                      4.500000e+01
Number of Observations Used    1.944100e+04
Critical value (1%)           -3.430686e+00
Critical value (5%)           -2.861689e+00
Critical value (10%)          -2.566849e+00
dtype: float64
```

In [22]:
```
datasetLogScaleMinusMovingAverage["Total"]
```

Out[22]:
```
DateAndTime
2021-05-01 05:30:00   -1.242751
2021-05-01 06:00:00   -0.035592
2021-05-01 06:30:00    0.981939
2021-05-01 07:00:00    1.939204
2021-05-01 07:30:00    1.275074
                          ...
2022-06-14 21:30:00   -0.036404
2022-06-14 22:00:00   -0.435267
2022-06-14 22:30:00   -0.498610
2022-06-14 23:00:00   -0.878022
2022-06-14 23:30:00   -1.002376
Name: Total, Length: 19476, dtype: float64
```

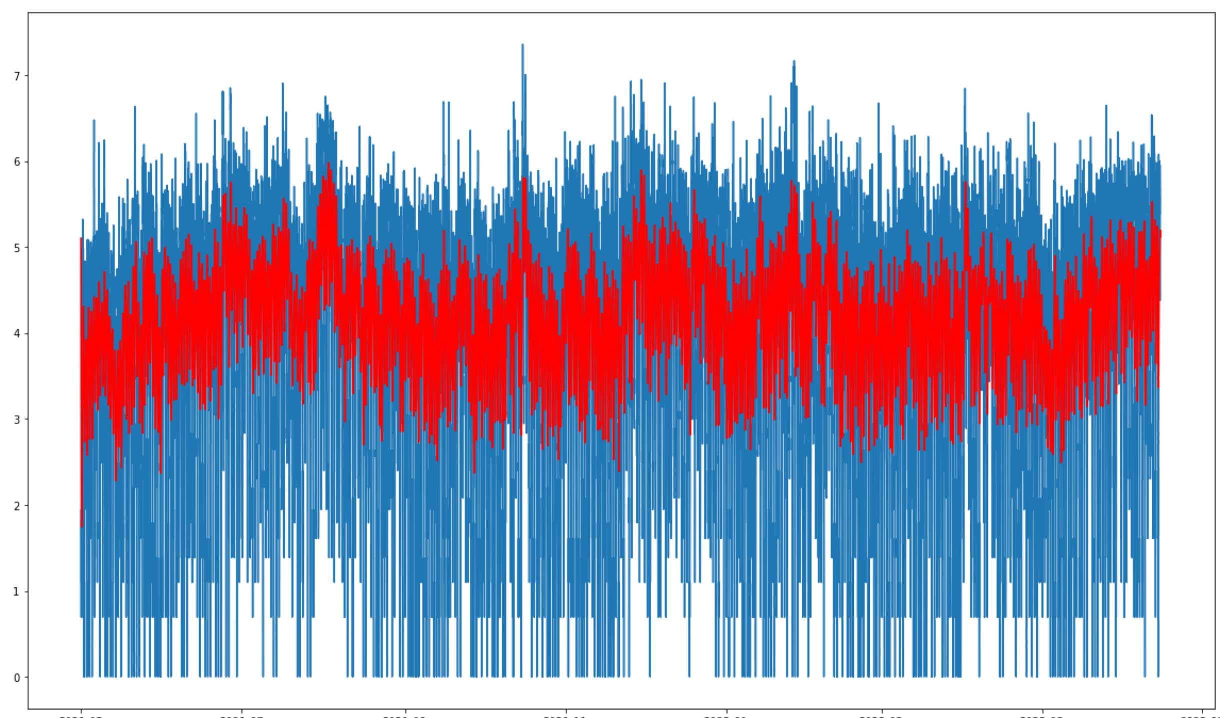
In [23]:
```
datasetLogScaleMinusMovingAverage=datasetLogScaleMinusMovingAverage.dropna()
```

In [27]:
```
exponentialDecayWeightedAverage=indexedDataset_logscale.ewm(halflife=12, min_peri
plt.plot(indexedDataset_logscale)
plt.plot(exponentialDecayWeightedAverage, color='red')

exponentialDecayWeightedAverage
```

Out[27]:

	Total
DateAndTime	
2021-05-01 00:00:00	5.099866
2021-05-01 00:30:00	4.631620
2021-05-01 01:00:00	3.772858
2021-05-01 01:30:00	2.934994
2021-05-01 02:00:00	2.638407
...	...
2022-06-14 21:30:00	5.196527
2022-06-14 22:00:00	5.197560
2022-06-14 22:30:00	5.192757
2022-06-14 23:00:00	5.160890
2022-06-14 23:30:00	5.117176

19487 rows × 1 columns



In [28]:
```
#Another Transformation

datasetLogScaleMinusMovingExponentialDecayAverage=indexedDataset_logscale - expor
datasetLogScaleMinusMovingExponentialDecayAverage=datasetLogScaleMinusMovingExpor
test_stationarity(datasetLogScaleMinusMovingExponentialDecayAverage.head(50))
```

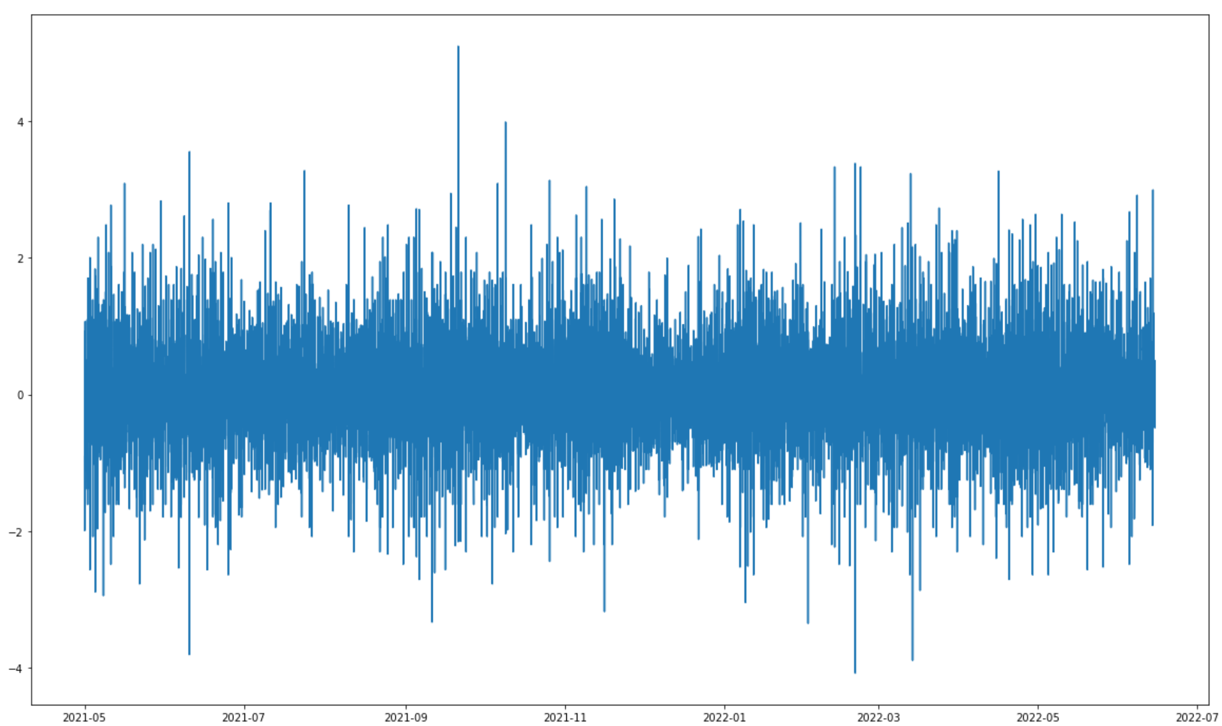

```
Results of Dickey-fuller Test
Test Statistic                 -1.534868
p-value                         0.516198
Lags Used                       4.000000
Number of Observations Used    45.000000
Critical value (1%)            -3.584829
Critical value (5%)            -2.928299
Critical value (10%)           -2.602344
dtype: float64
```

In [29]:
```
#Shift the values into timeseries to use in forecasting
datasetLogDiffShifting=indexedDataset_logscale - indexedDataset_logscale.shift()
plt.plot(datasetLogDiffShifting)

datasetLogDiffShifting
```

Out[29]:

	Total
DateAndTime	
2021-05-01 00:00:00	Nan
2021-05-01 00:30:00	-0.910212
2021-05-01 01:00:00	-1.992430
2021-05-01 01:30:00	-1.504077
2021-05-01 02:00:00	0.916291
...	...
2022-06-14 21:30:00	-0.218184
2022-06-14 22:00:00	-0.430511
2022-06-14 22:30:00	-0.102948
2022-06-14 23:00:00	-0.487015
2022-06-14 23:30:00	-0.242946

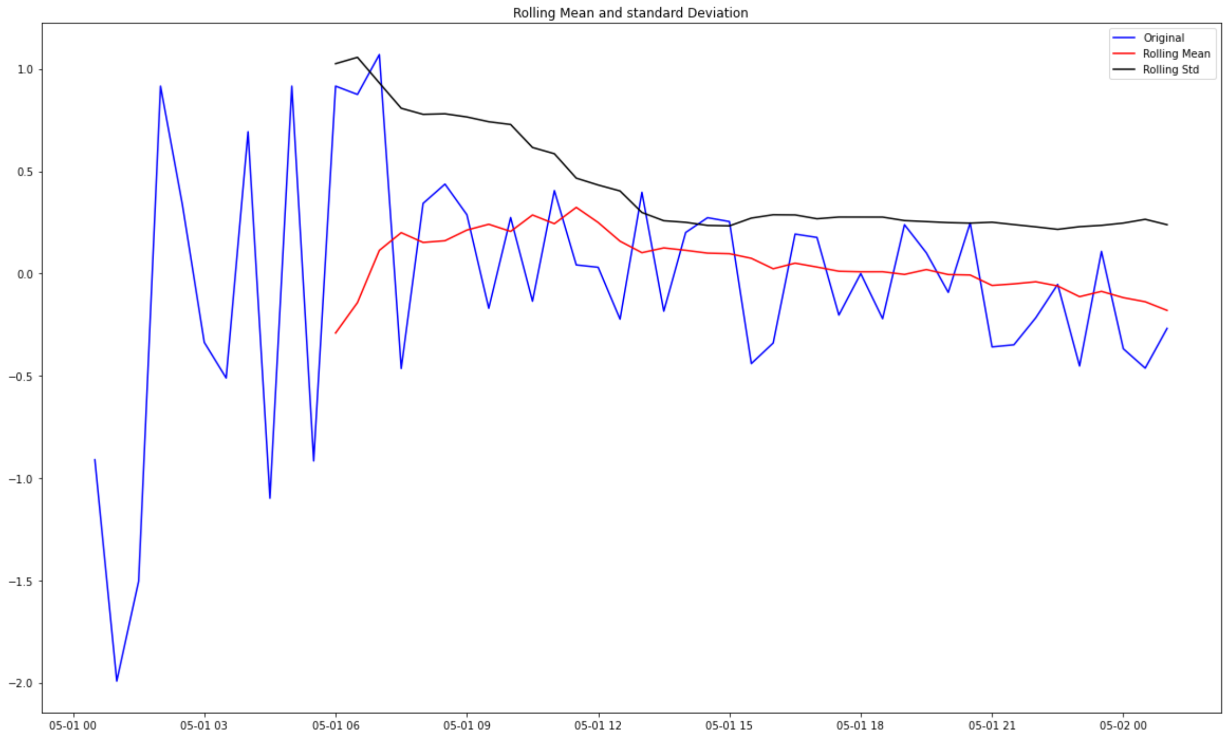
19487 rows × 1 columns



In [30]:

```
# ARIMA (Autoregressive Integrated Moving Average) model is a generalization of a
# ARMA (Autoregressive Moving Average) model is used to describe weakly stationar
# ARIMA: AR (Auto Regression) model, MA (Moving Average) model, I (Integration) m
```

In [31]:
```
#Flat output
datasetLogDiffShifting=datasetLogDiffShifting.dropna()
test_stationarity(datasetLogDiffShifting.head(50))
datasetLogDiffShifting
```




```
Results of Dickey-fuller Test
Test Statistic                 -2.208965
p-value                         0.203008
Lags Used                      10.000000
Number of Observations Used    39.000000
Critical value (1%)            -3.610400
Critical value (5%)            -2.939109
Critical value (10%)           -2.608063
dtype: float64
```

Out[31]:

	Total
DateAndTime	
2021-05-01 00:30:00	-0.910212
2021-05-01 01:00:00	-1.992430
2021-05-01 01:30:00	-1.504077
2021-05-01 02:00:00	0.916291
2021-05-01 02:30:00	0.336472
...	...
2022-06-14 21:30:00	-0.218184
2022-06-14 22:00:00	-0.430511
2022-06-14 22:30:00	-0.102948

DateAndTime	Total
2022-06-14 23:00:00	-0.487015
2022-06-14 23:30:00	-0.242946

19486 rows × 1 columns

In [32]:
```
indexedDataset
```

Out[32]:

DateAndTime	Total
2021-05-01 00:00:00	164
2021-05-01 00:30:00	66
2021-05-01 01:00:00	9
2021-05-01 01:30:00	2
2021-05-01 02:00:00	5
...	...
2022-06-14 21:30:00	283
2022-06-14 22:00:00	184
2022-06-14 22:30:00	166
2022-06-14 23:00:00	102
2022-06-14 23:30:00	80

19487 rows × 1 columns

In [33]:
```
datasetLogDiffShifting.rolling(window=12).mean()
indexedDataset_logscale=indexedDataset_logscale.dropna()
indexedDataset_logscale=indexedDataset_logscale.fillna(0)

indexedDataset_logscale
```

Out[33]:

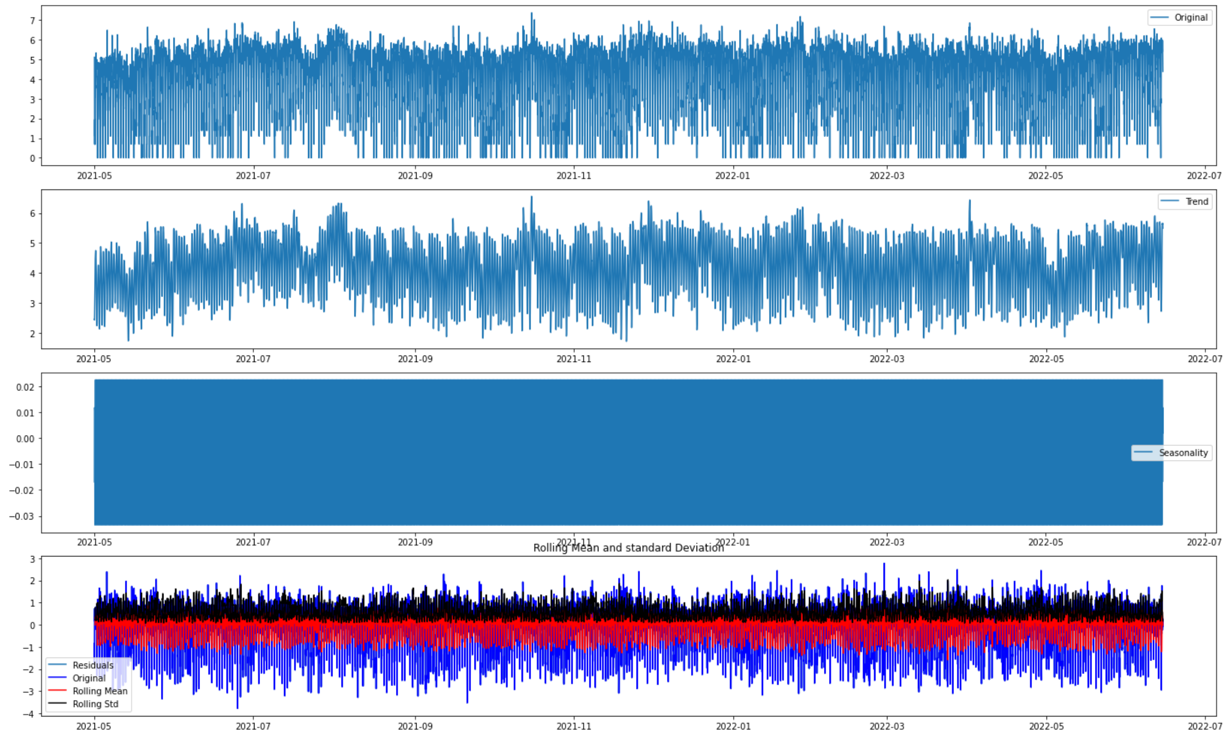
	Total
DateAndTime	
2021-05-01 00:00:00	5.099866
2021-05-01 00:30:00	4.189655
2021-05-01 01:00:00	2.197225
2021-05-01 01:30:00	0.693147
2021-05-01 02:00:00	1.609438
...	...
2022-06-14 21:30:00	5.645447
2022-06-14 22:00:00	5.214936
2022-06-14 22:30:00	5.111988
2022-06-14 23:00:00	4.624973
2022-06-14 23:30:00	4.382027

19487 rows × 1 columns

In [35]:

```
from random import randrange
from pandas import Series
from matplotlib import pyplot
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller


decomposition=seasonal_decompose(indexedDataset_logscale, period=20)


trend=decomposition.trend
seasonal=decomposition.seasonal
residual=decomposition.resid

plt.subplot(411)
plt.plot(indexedDataset_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()


decomposedLogdata=residual
decomposedLogdata.dropna(inplace=True)
decomposedLogdata

#Testing Stationarity

#Determing rolling statistics
movingAverage=decomposedLogdata.rolling(window=12).mean()
movingSTD=decomposedLogdata.rolling(window=12).std()

#plot rolling statistics
orig = plt.plot(decomposedLogdata, color='blue', label='Original')
mean =plt.plot(movingAverage, color='red', label='Rolling Mean')
std = plt.plot(movingSTD, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean and standard Deviation')
plt.show(block=False)

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

```
print(dfoutput)
```




```
Results of Dickey-fuller Test
Test Statistic                   -64.737427
p-value                            0.000000
Lags Used                         45.000000
Number of Observations Used    19421.000000
Critical value (1%)               -3.430687
Critical value (5%)               -2.861689
Critical value (10%)              -2.566849
dtype: float64
```

In [36]:
```
indexedDataset_logscale
```

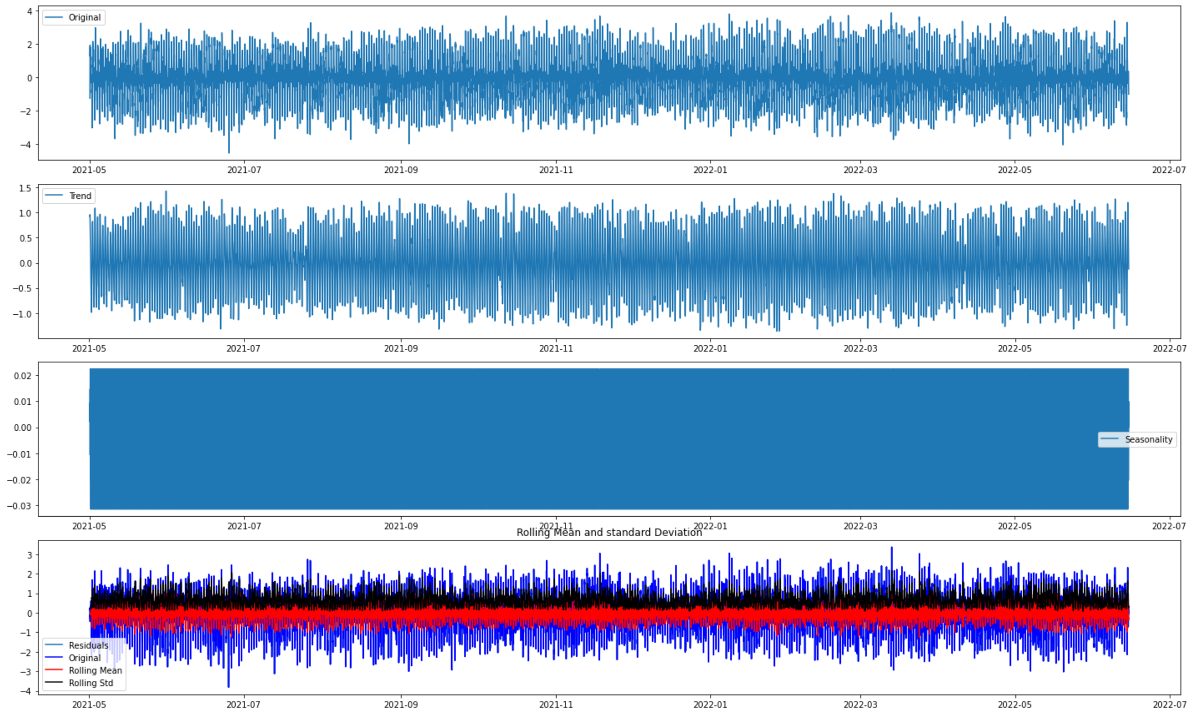
Out[36]:

	Total
DateAndTime	
2021-05-01 00:00:00	5.099866
2021-05-01 00:30:00	4.189655
2021-05-01 01:00:00	2.197225
2021-05-01 01:30:00	0.693147
2021-05-01 02:00:00	1.609438
...	...
2022-06-14 21:30:00	5.645447
2022-06-14 22:00:00	5.214936
2022-06-14 22:30:00	5.111988
2022-06-14 23:00:00	4.624973
2022-06-14 23:30:00	4.382027

19487 rows × 1 columns

In [37]:

```
decomposition=seasonal_decompose(datasetLogScaleMinusMovingAverage, period=20)

trend=decomposition.trend
seasonal=decomposition.seasonal
residual=decomposition.resid

plt.subplot(411)
plt.plot(datasetLogScaleMinusMovingAverage, 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()

decomposedLogdata=residual
decomposedLogdata.dropna(inplace=True)
decomposedLogdata

#Testing Stationarity
#Determing rolling statistics
movingAverage=decomposedLogdata.rolling(window=12).mean()
movingSTD=decomposedLogdata.rolling(window=12).std()

#plot rolling statistics
orig = plt.plot(decomposedLogdata, color='blue', label='Original')
mean =plt.plot(movingAverage, color='red', label='Rolling Mean')
std = plt.plot(movingSTD, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean and standard Deviation')
plt.show(block=False)

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

datasetLogDiffShifting
```




```
Results of Dickey-fuller Test
Test Statistic                    -67.205269
p-value                             0.000000
Lags Used                          45.000000
Number of Observations Used     19410.000000
Critical value (1%)                -3.430687
Critical value (5%)                -2.861689
Critical value (10%)               -2.566849
dtype: float64
```

Out[37]:

	Total
DateAndTime	
2021-05-01 00:30:00	-0.910212
2021-05-01 01:00:00	-1.992430
2021-05-01 01:30:00	-1.504077
2021-05-01 02:00:00	0.916291
2021-05-01 02:30:00	0.336472
...	...
2022-06-14 21:30:00	-0.218184
2022-06-14 22:00:00	-0.430511
2022-06-14 22:30:00	-0.102948
2022-06-14 23:00:00	-0.487015
2022-06-14 23:30:00	-0.242946

19486 rows × 1 columns

In [38]:
```
datasetLogDiffShifting
```

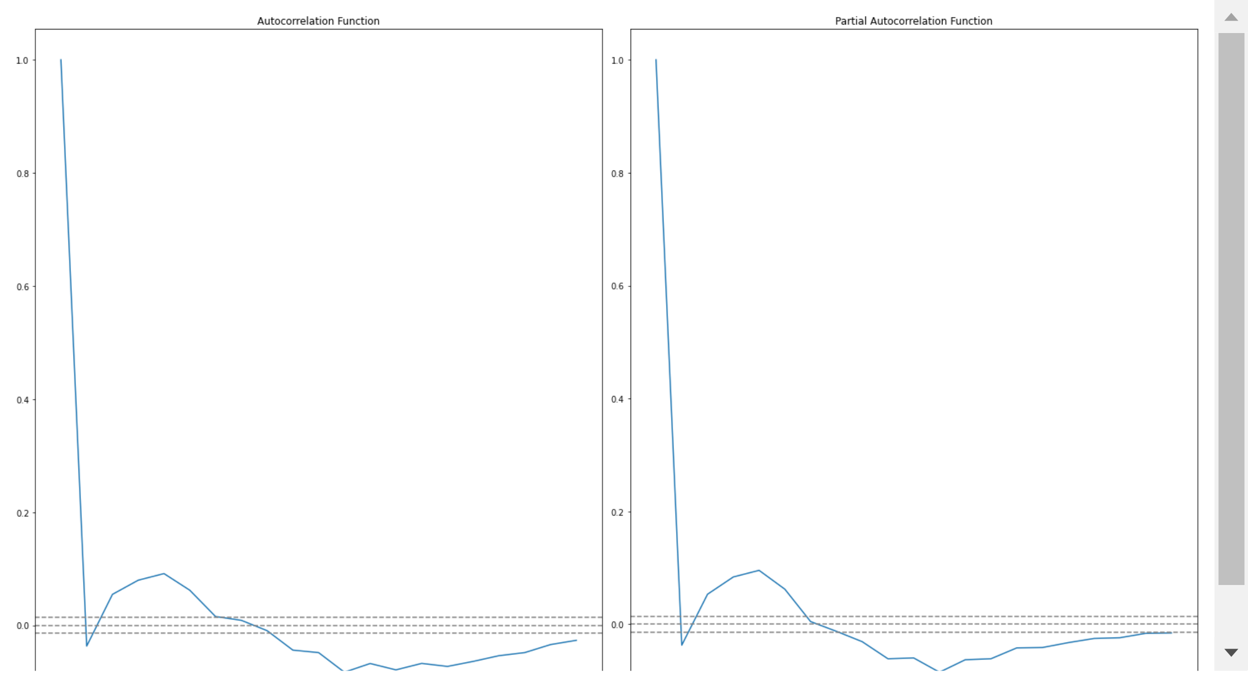
Out[38]:

	Total
DateAndTime	
2021-05-01 00:30:00	-0.910212
2021-05-01 01:00:00	-1.992430
2021-05-01 01:30:00	-1.504077
2021-05-01 02:00:00	0.916291
2021-05-01 02:30:00	0.336472
...	...
2022-06-14 21:30:00	-0.218184
2022-06-14 22:00:00	-0.430511
2022-06-14 22:30:00	-0.102948
2022-06-14 23:00:00	-0.487015
2022-06-14 23:30:00	-0.242946

19486 rows × 1 columns

In [39]:
```
#Autocorrelation Function (ACF) is a calculated value used to represent how simil
#Partial Autocorrelation Function (PACF)
```

In [40]:
```
#ACF and PACF Plot
from statsmodels.tsa.stattools import acf, pacf

lag_acf=acf(datasetLogDiffShifting, nlags=20)
lag_pacf=pacf(datasetLogDiffShifting, nlags=20, method='ols')

#Plot ACF
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gr
plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gra
plt.title('Autocorrelation Function')

#Plot PACF
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gr
plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gra
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```



In [41]:
```
lag_acf
```

Out[41]:
```
array([ 1.        , -0.0368507 ,  0.05466701,  0.07968141,  0.091395  ,
        0.06200139,  0.01546686,  0.00877073, -0.00951371, -0.04392857,
       -0.04834585, -0.08304124, -0.06759946, -0.07888219, -0.06742184,
       -0.07277546, -0.06387453, -0.05377011, -0.04831648, -0.03414877,
       -0.02658755])
```

In [42]:
```
lag_pacf
```

Out[42]:
```
array([ 1.        , -0.03685104,  0.05337265,  0.08394747,  0.09567473,
        0.0623066 ,  0.00489827, -0.01229515, -0.03061774, -0.06094376,
       -0.05948505, -0.08478612, -0.06268712, -0.06089747, -0.0416712 ,
       -0.04101061, -0.03235166, -0.02498227, -0.02370453, -0.01592014,
       -0.01525048])
```

In [43]:
```
pip install statsmodels
```

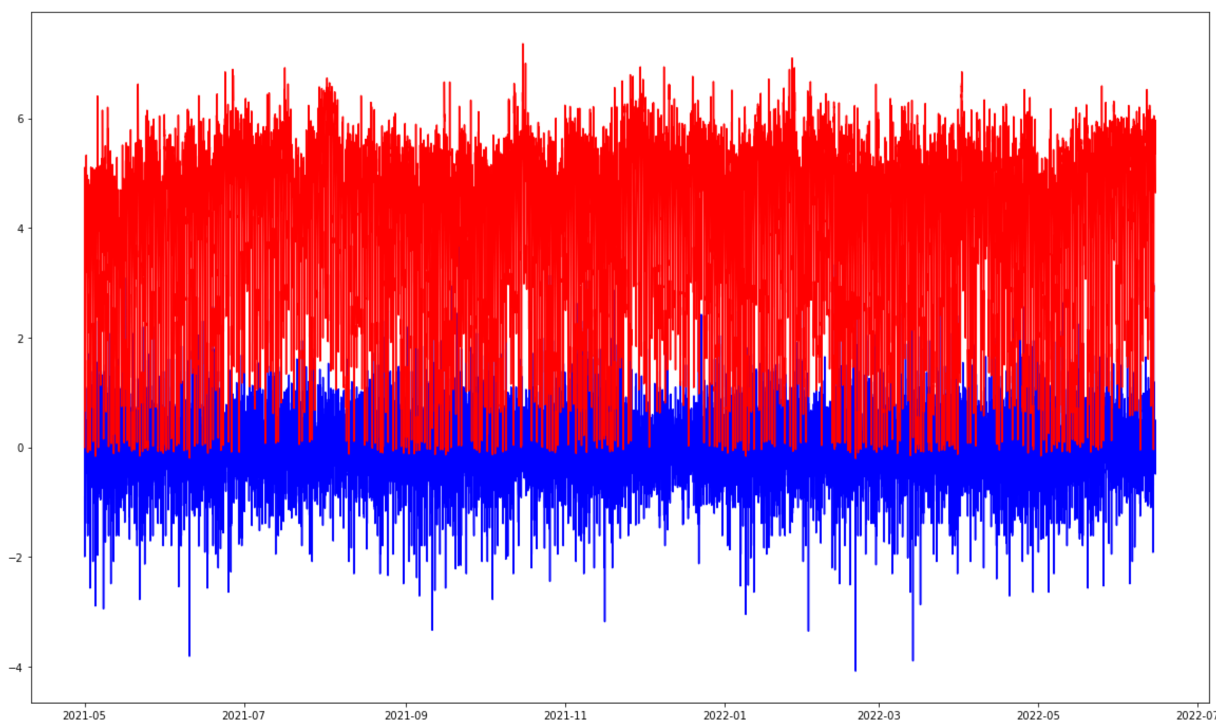
```
Requirement already satisfied: statsmodels in c:\users\bita\anaconda3\lib\site-
packages (0.13.2)Note: you may need to restart the kernel to use updated packag
es.

Requirement already satisfied: numpy>=1.17 in c:\users\bita\anaconda3\lib\site-
packages (from statsmodels) (1.21.5)
Requirement already satisfied: scipy>=1.3 in c:\users\bita\anaconda3\lib\site-p
ackages (from statsmodels) (1.7.3)
Requirement already satisfied: pandas>=0.25 in c:\users\bita\anaconda3\lib\site
-packages (from statsmodels) (1.4.2)
Requirement already satisfied: patsy>=0.5.2 in c:\users\bita\anaconda3\lib\site
-packages (from statsmodels) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\users\bita\anaconda3\lib\s
ite-packages (from statsmodels) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\bita\anacon
da3\lib\site-packages (from packaging>=21.3->statsmodels) (3.0.4)
Requirement already satisfied: pytz>=2020.1 in c:\users\bita\anaconda3\lib\site
-packages (from pandas>=0.25->statsmodels) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\bita\anaconda
3\lib\site-packages (from pandas>=0.25->statsmodels) (2.8.2)
Requirement already satisfied: six in c:\users\bita\anaconda3\lib\site-packages
(from patsy>=0.5.2->statsmodels) (1.16.0)
```

```
In [44]: from statsmodels.tsa.arima.model import ARIMA

#AR Model
cclone_indexedDataset_logscale.index = pd.DatetimeIndex(clone_indexedDataset_logs
model = ARIMA(clone_indexedDataset_logscale, order=(2, 1, 0))
results_AR = model.fit()
plt.plot(datasetLogDiffShifting, color='gray')
plt.plot(results_AR.fittedvalues, color='red')
print(results_AR.fittedvalues)
print('Plotting AR model')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [44], in <cell line: 4>()
      1 from statsmodels.tsa.arima.model import ARIMA
      3 #AR Model
----> 4 cclone_indexedDataset_logscale.index = pd.DatetimeIndex(clone_indexedDa
taset_logscale.index)
      5 model = ARIMA(clone_indexedDataset_logscale, order=(2, 1, 0))
      6 results_AR = model.fit()

NameError: name 'clone_indexedDataset_logscale' is not defined
```

In [155]:
```
#MA Model

clone_indexedDataset_logscale = indexedDataset_logscale.copy().resample('30min').
clone_indexedDataset_logscale.index = pd.DatetimeIndex(clone_indexedDataset_logsc
model = ARIMA(clone_indexedDataset_logscale, order=(0, 1, 2))
results_MA = model.fit()
plt.plot(datasetLogDiffShifting, color='blue')
plt.plot(results_MA.fittedvalues, color='red')
print(results_MA.fittedvalues)
print('Plotting MA model')
```

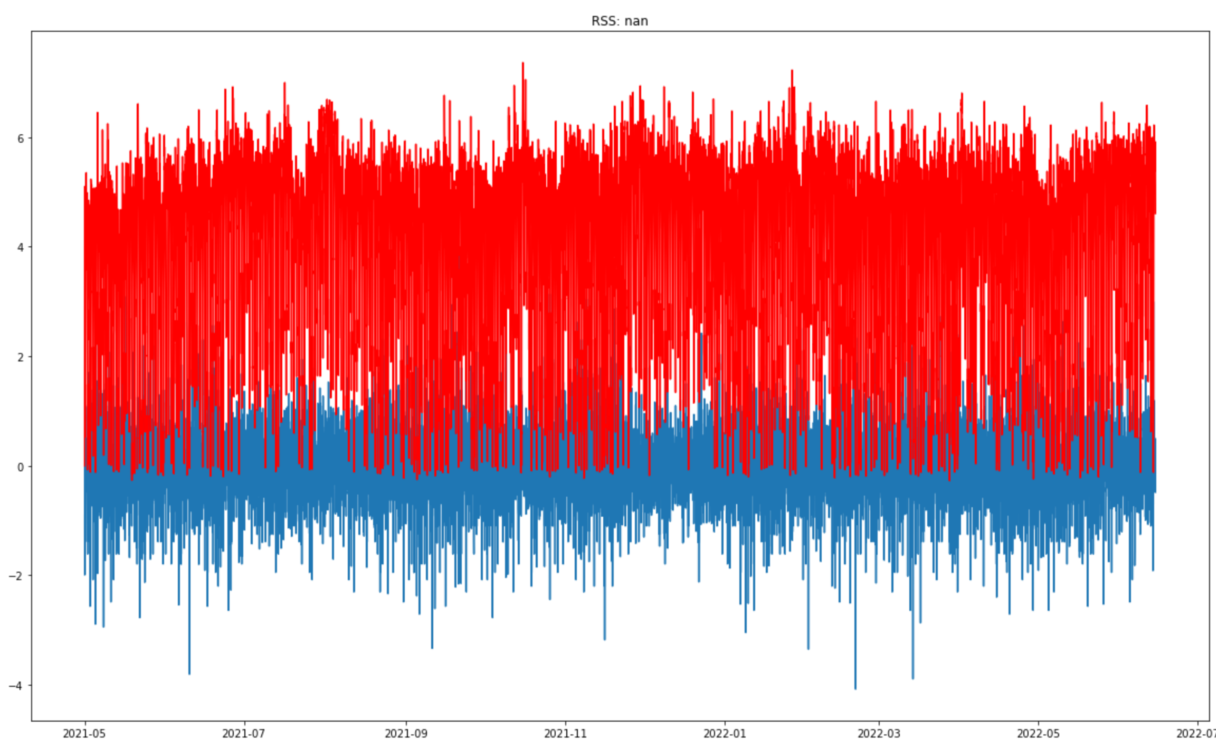
```
DateAndTime
2021-05-01 00:00:00    0.000000
2021-05-01 00:30:00    5.099866
2021-05-01 01:00:00    4.243494
2021-05-01 01:30:00    2.258168
2021-05-01 02:00:00    0.660707
                         ...
2022-06-14 21:30:00    5.869480
2022-06-14 22:00:00    5.652071
2022-06-14 22:30:00    5.226284
2022-06-14 23:00:00    5.092600
2022-06-14 23:30:00    4.644515
Freq: 30T, Length: 19680, dtype: float64
Plotting MA model
```



In [45]:
```
model=ARIMA(indexedDataset_logscale, order=(2,1,2))
results_ARIMA=model.fit()
plt.plot(datasetLogDiffShifting)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f' % sum((results_ARIMA.fittedvalues-datasetLogDiffShifting["
```

```
C:\Users\BITA\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: A date index has been provided, but it has no associated frequ
ency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\BITA\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: A date index has been provided, but it has no associated frequ
ency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\BITA\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: A date index has been provided, but it has no associated frequ
ency information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

Out[45]:
```
Text(0.5, 1.0, 'RSS: nan')
```



In [47]:
```
predictions_ARIMA_diff=pd.Series(results_ARIMA.fittedvalues, copy=True)
print(predictions_ARIMA_diff.head())
```

```
DateAndTime
2021-05-01 00:00:00    0.000000
2021-05-01 00:30:00    5.099866
2021-05-01 01:00:00    4.193625
2021-05-01 01:30:00    2.160300
2021-05-01 02:00:00    0.532114
dtype: float64
```

In [48]:
```
#Convert to cumulative sum
predictions_ARIMA_diff_cumsum=predictions_ARIMA_diff.cumsum()
print(predictions_ARIMA_diff_cumsum.head())
```

```
DateAndTime
2021-05-01 00:00:00     0.000000
2021-05-01 00:30:00     5.099866
2021-05-01 01:00:00     9.293492
2021-05-01 01:30:00    11.453792
2021-05-01 02:00:00    11.985906
dtype: float64
```

In [49]:
```
predictions_ARIMA_log=pd.Series(indexedDataset_logscale['Total'], indexedDataset_
predictions_ARIMA_log=predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum, fi
predictions_ARIMA_log.head()
```

Out[49]:
```
DateAndTime
2021-05-01 00:00:00     5.099866
2021-05-01 00:30:00     9.289521
2021-05-01 01:00:00    11.490716
2021-05-01 01:30:00    12.146939
2021-05-01 02:00:00    13.595344
dtype: float64
```

In [105]:
```
predictions_ARIMA_log
```

Out[105]:
```
DateAndTime
2021-05-01 00:00:00        5.099866
2021-05-01 00:30:00        9.289521
2021-05-01 01:00:00       11.490716
2021-05-01 01:30:00       12.146939
2021-05-01 02:00:00       13.595344
                             ...
2022-06-14 21:30:00    81747.994553
2022-06-14 22:00:00    81753.243180
2022-06-14 22:30:00    81758.395071
2022-06-14 23:00:00    81763.024469
2022-06-14 23:30:00    81767.389419
Length: 19487, dtype: float64
```

In [50]:

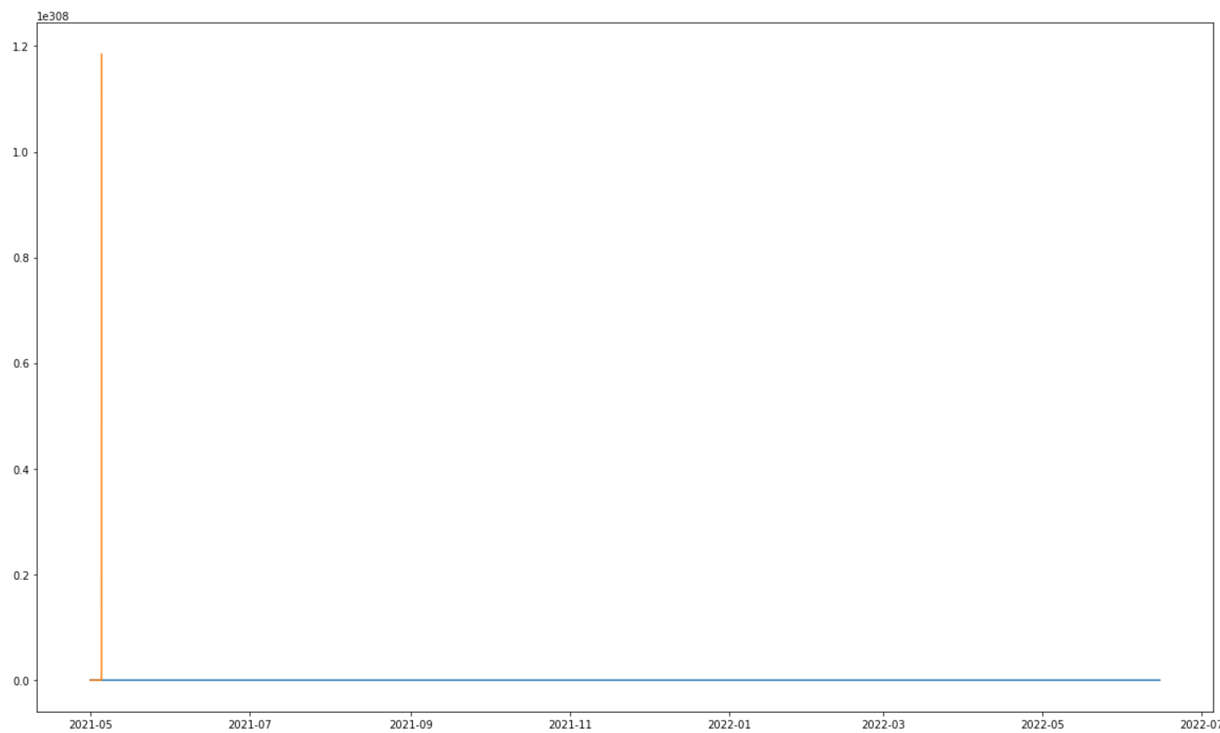
```
predictions_ARIMA=np.exp(predictions_ARIMA_log)
plt.plot(indexedDataset)
plt.plot(predictions_ARIMA)
```

```
C:\Users\BITA\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: Runtime
Warning: overflow encountered in exp
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[50]:

```
[<matplotlib.lines.Line2D at 0x213baed98e0>]
```

```
C:\Users\BITA\anaconda3\lib\site-packages\matplotlib\ticker.py:2072: RuntimeWar
ning: overflow encountered in multiply
  steps = self._extended_steps * scale
```



In [51]:
```
predictions_ARIMA
```

Out[51]:
```
DateAndTime
2021-05-01 00:00:00    1.640000e+02
2021-05-01 00:30:00    1.082400e+04
2021-05-01 01:00:00    9.780356e+04
2021-05-01 01:30:00    1.885162e+05
2021-05-01 02:00:00    8.023854e+05
                           ...
2022-06-14 21:30:00             inf
2022-06-14 22:00:00             inf
2022-06-14 22:30:00             inf
2022-06-14 23:00:00             inf
2022-06-14 23:30:00             inf
Length: 19487, dtype: float64
```

In [61]:
```
indexedDataset_logscale
```

Out[61]:

	Total
DateAndTime	
2021-05-01 00:00:00	5.099866
2021-05-01 00:30:00	4.189655
2021-05-01 01:00:00	2.197225
2021-05-01 01:30:00	0.693147
2021-05-01 02:00:00	1.609438
...	...
2022-06-14 21:30:00	5.645447
2022-06-14 22:00:00	5.214936
2022-06-14 22:30:00	5.111988
2022-06-14 23:00:00	4.624973
2022-06-14 23:30:00	4.382027

19487 rows × 1 columns

In [75]:
```
x=results_ARIMA.forecast(steps=96)
```

```
C:\Users\BITA\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:83
4: ValueWarning: No supported index is available. Prediction results will be gi
ven with an integer index beginning at `start`.
  return get_prediction_index(
```

In [80]:
```
x
```

Out[80]:
```
19487    4.319302
19488    4.215320
19489    4.098493
19490    3.991250
19491    3.908125
           ...
19578    3.952334
19579    3.952334
19580    3.952334
19581    3.952334
19582    3.952334
Name: predicted_mean, Length: 96, dtype: float64
```

In [83]:
```
results_ARIMA
```

Out[83]:
```
<statsmodels.tsa.arima.model.ARIMAResultsWrapper at 0x213b6be0190>
```

In [98]:
```
td = pd.read_csv('TimeSeries_Gediz_no_null_testset_LN.csv',
        header=0,
        usecols=["Total"])

td
```

Out[98]:

	Total
0	4.158883
1	3.610918
2	3.044522
3	2.944439
4	2.639057
...	...
91	5.303305
92	5.153292
93	5.680173
94	4.663439
95	4.248495

96 rows × 1 columns

In [97]:
```
td['Total']
```

Out[97]:
```
0     4.158883
1     3.610918
2     3.044522
3     2.944439
4     2.639057
        ...
91    5.303305
92    5.153292
93    5.680173
94    4.663439
95    4.248495
Name: Total, Length: 96, dtype: float64
```

In [68]:
```
#Measuring Time Series Forecasting Performance
#Evaluation Metrics to Measure Performance:
#R-Squared
#Mean Absolute Error
#Mean Absolute Percentage Error
#Mean Squared Error
#Root Mean Squared Error
#Normalized Root Mean Squared Error
#Weighted Absolute Percentage Error
#Weighted Mean Absolute Percentage Error
```

In [99]:
```
x
```

Out[99]:
```
19487    4.319302
19488    4.215320
19489    4.098493
19490    3.991250
19491    3.908125
           ...
19578    3.952334
19579    3.952334
19580    3.952334
19581    3.952334
19582    3.952334
Name: predicted_mean, Length: 96, dtype: float64
```

In [100]:
```
# RMSE (Root Mean Square Error)
from sklearn.metrics import mean_squared_error

y_actual = td['Total']

y_predicted = x

RMSE = mean_squared_error(y_actual, y_predicted, squared=False)

print("Root Mean Square Error (RMSE):\n")
print(RMSE)
```

```
Root Mean Square Error (RMSE):

1.6507808512623035
```

In [101]:
```
# Mean Absolute Percentage Error (MAPE)
import numpy as np

def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true))*100
```

In [102]:
```
MAPE=mean_absolute_percentage_error(y_actual, y_predicted)
```

In [104]:
```
print("Mean Absolute Percentage Error (MAPE):\n")
print(MAPE)
```

```
Mean Absolute Percentage Error (MAPE):

44.69302551428012
```

In [96]:
```python
df = pd.DataFrame(x)

# show the dataframe
print(df)

# iterating over and calling
# tolist() method for
# each column
for i in list(df):

    # show the list of values
    print(df[i].tolist())
```

```
       predicted_mean
19487        4.319302
19488        4.215320
19489        4.098493
19490        3.991250
19491        3.908125
...               ...
19578        3.952334
19579        3.952334
19580        3.952334
19581        3.952334
19582        3.952334

[96 rows x 1 columns]
[4.3193018257111975, 4.215320340571137, 4.098493289824193, 3.991250134427406,
3.9081254346501395, 3.8556573882417324, 3.833601267464753, 3.836890122203912 2,
3.857815712285781, 3.8880164989660897, 3.9200075938213588, 3.948136329271842,
3.968972453111979, 3.981230336148755, 3.9853682678832856, 3.9830207591707834,
3.9764024646445293, 3.9677876388111843, 3.95912745587539, 3.951827693246658, 3.
946677111945268, 3.943895366813619, 3.9432588823430432, 3.9442623632323714, 3.9
462798679003135, 3.948699661208688, 3.9510185836198635, 3.952892225601334, 3.95
41453557411597, 3.9547521838712862, 3.954798178311759, 3.9544348036673953, 3.95
38364646174773, 3.953165956114501, 3.9525515675478475, 3.952076222389161 7, 3.95
17769909447393, 3.951652116285359, 3.951672295089676, 3.9517931937044866, 3.951
9668364124647, 3.95215035364463, 3.95231143087463, 3.95243051634773, 3.95250035
0643569, 3.952523652360084, 3.9525098544596005, 3.952471685123538, 3.9524221871
05892, 3.9523725307917745, 3.9523307478892242, 3.952301328651262 5, 3.9522855028
07481, 3.952281965350834, 3.9522878044316694, 3.9522994248646017, 3.95231331995
1366, 3.9523266104435844, 3.9523373299173326, 3.95234448249074, 3.9523479280875
673, 3.9523481625508574, 3.9523460577587395, 3.9523426148727747, 3.952338766687
1677, 3.952335246961208, 3.952332528778143, 3.9523308222925024, 3.9523301154057
56, 3.9523302386679684, 3.952330937104197, 3.952331935451341, 3.952332988179126
7, 3.9523339105597644, 3.952334591157122, 3.952334988989752, 3.95233512016538 2
7, 3.9523350391189482, 3.952334819000498, 3.952334534608258, 3.952334249890997
2, 3.95233401073555, 3.9523338427024877, 3.9523337526725264, 3.952333733031091
3, 3.95233376699023, 3.9523338339273586, 3.9523339137147393, 3.95233398988545
6, 3.95233405121244 6, 3.9523340920363474, 3.952334111598655, 3.952334112775496
5, 3.95233410058547 6, 3.9523340807755876, 3.9523340586905262]
```

In [ ]: