Data Mining and Neural Networks Computational Task 3

Importing Libraries

In [1]:
```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
from scipy.stats import zscore
from sklearn.preprocessing import MinMaxScaler
import datetime
from matplotlib.pylab import gca, figure, plot, subplot, title, xlabel, ylabel
, xlim,show
from matplotlib.lines import Line2D
import segment
import fit
import os
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
from datetime import datetime
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
from statsmodels.tsa.seasonal import seasonal_decompose
from math import sqrt
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
```

Loading Datasets for 2018, 2019 and 2020 AAPL data set which is the time series dataset of APPLE inc.

In [2]:
```
# Apple Dataset for 2018, 2019 and 2020
dataset1=  pd.read_csv('AAPL_2018.csv')
dataset2=  pd.read_csv('AAPL_2019.csv')
dataset3=  pd.read_csv('AAPL_2020.csv')
```

In [3]:
```
dataset1.head(5)
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1/2/2018	170.160004	172.300003	169.259995	172.259995	166.353714	25555900
1	1/3/2018	172.529999	174.550003	171.960007	172.229996	166.324722	29517900
2	1/4/2018	172.539993	173.470001	172.080002	173.029999	167.097290	22434600
3	1/5/2018	173.440002	175.369995	173.050003	175.000000	168.999741	23660000
4	1/8/2018	174.350006	175.610001	173.929993	174.350006	168.372040	20567800

In [4]:
```
dataset2.head(5)
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1/2/2019	154.889999	158.850006	154.229996	157.919998	154.794983	37039700
1	1/3/2019	143.979996	145.720001	142.000000	142.190002	139.376251	91312200
2	1/4/2019	144.529999	148.550003	143.800003	148.259995	145.326126	58607100
3	1/7/2019	148.699997	148.830002	145.899994	147.929993	145.002686	54777800
4	1/8/2019	149.559998	151.820007	148.520004	150.750000	147.766861	41025300

In [5]:
```
dataset3.head(5)
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1/2/2020	74.059998	75.150002	73.797501	75.087502	74.573036	135480400
1	1/3/2020	74.287498	75.144997	74.125000	74.357498	73.848030	146322800
2	1/6/2020	73.447502	74.989998	73.187500	74.949997	74.436470	118387200
3	1/7/2020	74.959999	75.224998	74.370003	74.597504	74.086395	108872000
4	1/8/2020	74.290001	76.110001	74.290001	75.797501	75.278160	132079200

1. Data evaluation and elementary preprocessing

Analyzing Datasets Checking Missed Values On Overall Data

In [6]:
```
dataset1.isnull().sum()
```

Out[6]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [7]:
```
dataset2.isnull().sum()
```

Out[7]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [8]:
```
dataset3.isnull().sum()
```

Out[8]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Analyzing Missed Values for Weekends but before that we need to add column Day Name in Data

In [9]:
```
import datetime
date=dataset1['Date']
data_copy1 = dataset1
data_copy1['Day Name'] = data_copy1['Date']
day_name= ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']
for i in range(len(date)):
    day = datetime.datetime.strptime(date[i], '%m/%d/%Y').weekday()
    data_copy1['Day Name'][i]= day_name[day]
data_copy1['Day Name']
```

Out[9]:
```
0        Tuesday
1      Wednesday
2       Thursday
3         Friday
4         Monday
          ...
246       Monday
247    Wednesday
248     Thursday
249       Friday
250       Monday
Name: Day Name, Length: 251, dtype: object
```

In [10]:
```
data_copy1.drop('Day Name', inplace=True, axis=1)
idx = pd.date_range('1/1/2018','12/31/2018')

data_copy1.index = pd.DatetimeIndex(data_copy1['Date'])

data_copy1 = data_copy1.reindex(idx, fill_value=np.nan,index=False)
data_copy1['Date']=data_copy1.index
```

For dataset 1 we see missing value count = 365 - 251 = 114 out of which 104 are Weekend days because in a year we have 52.1429 weeks multiplied by 2 gives us 104 days and ten extra days so we will use preprocessing on the data to fill the values for now we have added the missing values as 0 which means null values

In [11]:
```
date=dataset2['Date']
data_copy2 = dataset2
data_copy2['Day Name'] = data_copy2['Date']
day_name= ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']
for i in range(len(date)):
    day = datetime.datetime.strptime(date[i], '%m/%d/%Y').weekday()
    data_copy2['Day Name'][i]= day_name[day]
data_copy2['Day Name']
```

Out[11]:
```
0      Wednesday
1       Thursday
2         Friday
3         Monday
4        Tuesday
         ...
216      Tuesday
217     Thursday
218       Friday
219       Monday
220      Tuesday
Name: Day Name, Length: 221, dtype: object
```

In [12]:
```
data_copy2.drop('Day Name', inplace=True, axis=1)
idx = pd.date_range('1/1/2019','12/31/2019')

data_copy2.index = pd.DatetimeIndex(data_copy2['Date'])

data_copy2 = data_copy2.reindex(idx, fill_value=np.nan,index=False)
data_copy2['Date']=data_copy2.index
```

For 2019 dataset we see missing value count = 365 - 221 = 144 out of which 104 are Weekend days because in a year we have 52.1429 weeks multiplied by 2 gives us 104 days and 40 extra days so we will use preprocessing on the data to fill the values for now we have added the missing values as 0 which means null values

In [13]:
```
date=dataset3['Date']
data_copy3 = dataset3
data_copy3['Day Name'] = data_copy3['Date']
day_name= ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']
for i in range(len(date)):
    day = datetime.datetime.strptime(date[i], '%m/%d/%Y').weekday()
    data_copy3['Day Name'][i]= day_name[day]
data_copy3['Day Name']
```

Out[13]:
```
0       Thursday
1         Friday
2         Monday
3        Tuesday
4      Wednesday
          ...
170     Thursday
171       Friday
172      Tuesday
173    Wednesday
174     Thursday
Name: Day Name, Length: 175, dtype: object
```

In [14]:
```
data_copy3.drop('Day Name', inplace=True, axis=1)
idx = pd.date_range('1/1/2020','12/31/2020')

data_copy3.index = pd.DatetimeIndex(data_copy3['Date'])

data_copy3 = data_copy3.reindex(idx, fill_value=np.nan,index=False)
data_copy3['Date']=data_copy3.index
```

For 2020 dataset we see missing value count = 365 - 175 = 190 out of which 104 are Weekend days because in a year we have 52.1429 weeks multiplied by 2 gives us 104 days and 86 extra days so we will use preprocessing on the data to fill the values for now we have added the missing values as 0 which means null values.

In Time Series Datasets traditionally, researchers omit days where data includes crisis e.g. wars, conflicts and other impacts on the stocks and for that they drop the days including the values because it may affect the statistical signifcance of the datasets. According to me this the reason of the missing data and on Weekends this could be a normal practice as mostly on weekends there is no work going on so those dataset points are outside of Trading Hours.

Pre Processing Techniques

- Fill values with closest in time values for that if the last working day is Friday we will take the value of Friday and add it to the weekends
- Normalise to the z-score so each row has mean=0 and sd=1, it allows us to understand the probability of a score occuring with normal distribution of the data
- It helps compare values from two different normal distributions.
- Missing Values according to Average of High and Low columns of the dataset

Filling Values with Closest In Time using Nearest Interpolation Method of Python

In [15]:
```
a=data_copy1
b=data_copy2
c=data_copy3
a.interpolate(method='nearest', inplace=True,fill_value='extrapolate',  limit_
direction='both')
a
```

Out[15]:

	Date	Open	High	Low	Close	Adj Close	Volume
2018-01-01	2018-01-01	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0
2018-01-02	2018-01-02	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0
2018-01-03	2018-01-03	172.529999	174.550003	171.960007	172.229996	166.324722	29517900.0
2018-01-04	2018-01-04	172.539993	173.470001	172.080002	173.029999	167.097290	22434600.0
2018-01-05	2018-01-05	173.440002	175.369995	173.050003	175.000000	168.999741	23660000.0
...	...	...	...	...	...	...	...
2018-12-27	2018-12-27	155.839996	156.770004	150.070007	156.149994	153.059998	53117100.0
2018-12-28	2018-12-28	157.500000	158.520004	154.550003	156.229996	153.138428	42291400.0
2018-12-29	2018-12-29	157.500000	158.520004	154.550003	156.229996	153.138428	42291400.0
2018-12-30	2018-12-30	158.529999	159.360001	156.479996	157.740005	154.618546	35003500.0
2018-12-31	2018-12-31	158.529999	159.360001	156.479996	157.740005	154.618546	35003500.0

365 rows × 7 columns

Creating the variable Average, between the low and the high price to plot the Average Stock Price of Apple

In [16]:
```
df1 = a
df1['Average'] = (a['High'] + a['Low'])/2
df1 = df1[['Average']]
```

In [17]:
```
df1.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

In [18]:
```
b.interpolate(method='nearest',fill_value='extrapolate' ,inplace=True,limit_direction='both')
b
```

Out[18]:

	Date	Open	High	Low	Close	Adj Close	Volume
2019-01-01	2019-01-01	154.889999	158.850006	154.229996	157.919998	154.794983	37039700.0
2019-01-02	2019-01-02	154.889999	158.850006	154.229996	157.919998	154.794983	37039700.0
2019-01-03	2019-01-03	143.979996	145.720001	142.000000	142.190002	139.376251	91312200.0
2019-01-04	2019-01-04	144.529999	148.550003	143.800003	148.259995	145.326126	58607100.0
2019-01-05	2019-01-05	144.529999	148.550003	143.800003	148.259995	145.326126	58607100.0
...	...	...	...	...	...	...	...
2019-12-27	2019-12-27	72.779999	73.492500	72.029999	72.449997	71.953598	146266000.0
2019-12-28	2019-12-28	72.779999	73.492500	72.029999	72.449997	71.953598	146266000.0
2019-12-29	2019-12-29	72.364998	73.172501	71.305000	72.879997	72.380653	144114400.0
2019-12-30	2019-12-30	72.364998	73.172501	71.305000	72.879997	72.380653	144114400.0
2019-12-31	2019-12-31	72.482498	73.419998	72.379997	73.412498	72.909500	100805600.0

365 rows × 7 columns

In [19]:
```
df2 = b
df2['Average'] = (b['High'] + b['Low'])/2
df2 = df2[['Average']]
```

In [20]:
```
df2.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

In [21]:
```
c.interpolate(method='nearest',inplace=True,fill_value='extrapolate'    ,limit_
direction='both')
c
```

Out[21]:

	Date	Open	High	Low	Close	Adj Close	Volume
2020-01-01	2020-01-01	74.059998	75.150002	73.797501	75.087502	74.573036	135480400.0
2020-01-02	2020-01-02	74.059998	75.150002	73.797501	75.087502	74.573036	135480400.0
2020-01-03	2020-01-03	74.287498	75.144997	74.125000	74.357498	73.848030	146322800.0
2020-01-04	2020-01-04	74.287498	75.144997	74.125000	74.357498	73.848030	146322800.0
2020-01-05	2020-01-05	73.447502	74.989998	73.187500	74.949997	74.436470	118387200.0
...	...	...	...	...	...	...	...
2020-12-27	2020-12-27	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0
2020-12-28	2020-12-28	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0
2020-12-29	2020-12-29	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0
2020-12-30	2020-12-30	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0
2020-12-31	2020-12-31	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0

366 rows × 7 columns

In [22]:
```
df3 = c
df3['Average'] = (c['High'] + c['Low'])/2
df3 = df3[['Average']]
```

In [23]:
```
df3.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

Filling Values with Z-Score Normalization using built in library zcore of Python and Plotting Average

In [24]:
```
d=data_copy1
e=data_copy2
f=data_copy3
df4=d
zetascore_table=zscore(df4.iloc[:,1:6],axis=1)
```

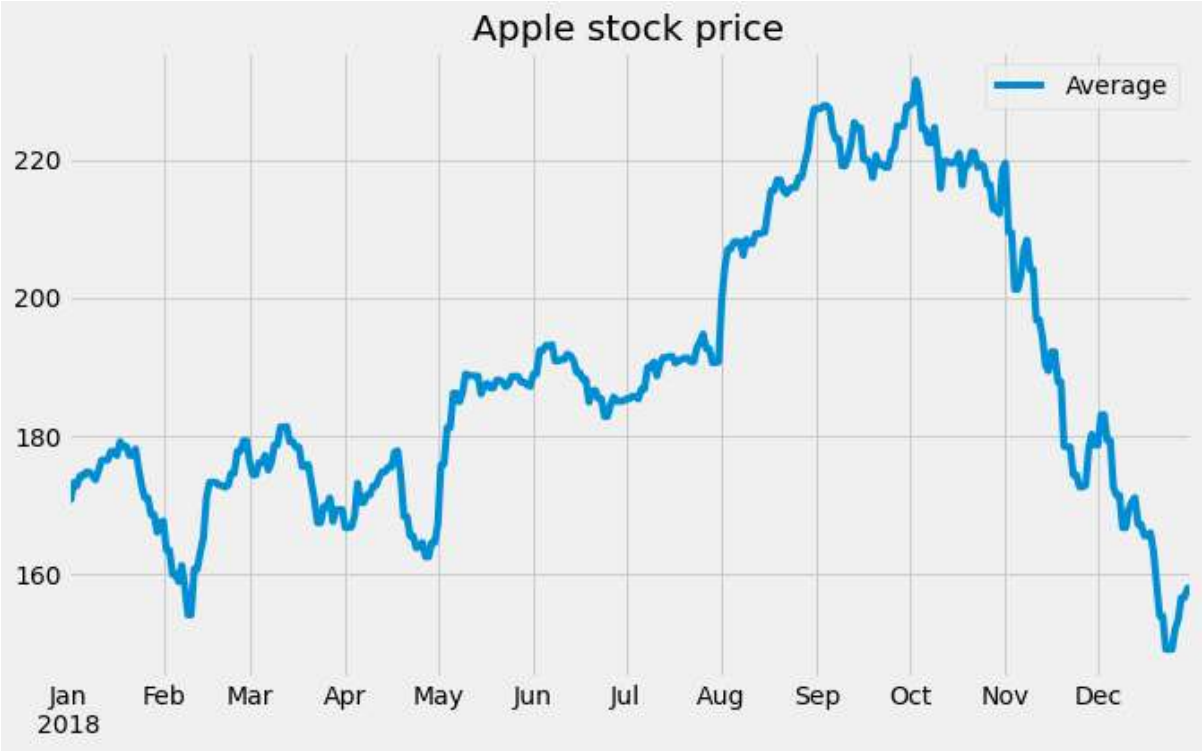
In [25]:
```
df4['Average'] = (df4['High'] + df4['Low'])/2
df4 = df4[['Average']]
df4.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

In [26]:
```
df5=e
zetascore_table=zscore(df5.iloc[:,1:6],axis=1)
df5['Average'] = (df5['High'] + df5['Low'])/2
df5 = df5[['Average']]
df5.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

In [27]:
```
df6=f
zetascore_table=zscore(df6.iloc[:,1:6],axis=1)
df6['Average'] = (df6['High'] + df6['Low'])/2
df6 = df6[['Average']]
df6.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

Filling Values with Mean Values and Imputing Null Values

In [28]:
```
g=data_copy1
h=data_copy2
i=data_copy3
df7=g
df7.fillna(df7.iloc[:,1:6].mean())
df7['Average'] = (df7['High'] + df7['Low'])/2
df7 = df7[['Average']]
df7.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```



In [29]:
```
df8=h
df8.fillna(df8.iloc[:,1:6].mean())
df8['Average'] = (df8['High'] + df8['Low'])/2
df8 = df8[['Average']]
df8.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

In [30]:
```
df9=i
df9.fillna(df9.iloc[:,1:6].mean())
df9['Average'] = (df9['High'] + df9['Low'])/2
df9 = df9[['Average']]
df9.plot(legend=True)
plt.title('Apple stock price')
plt.show() # Plotting Average Stock Price after Interpolation
```

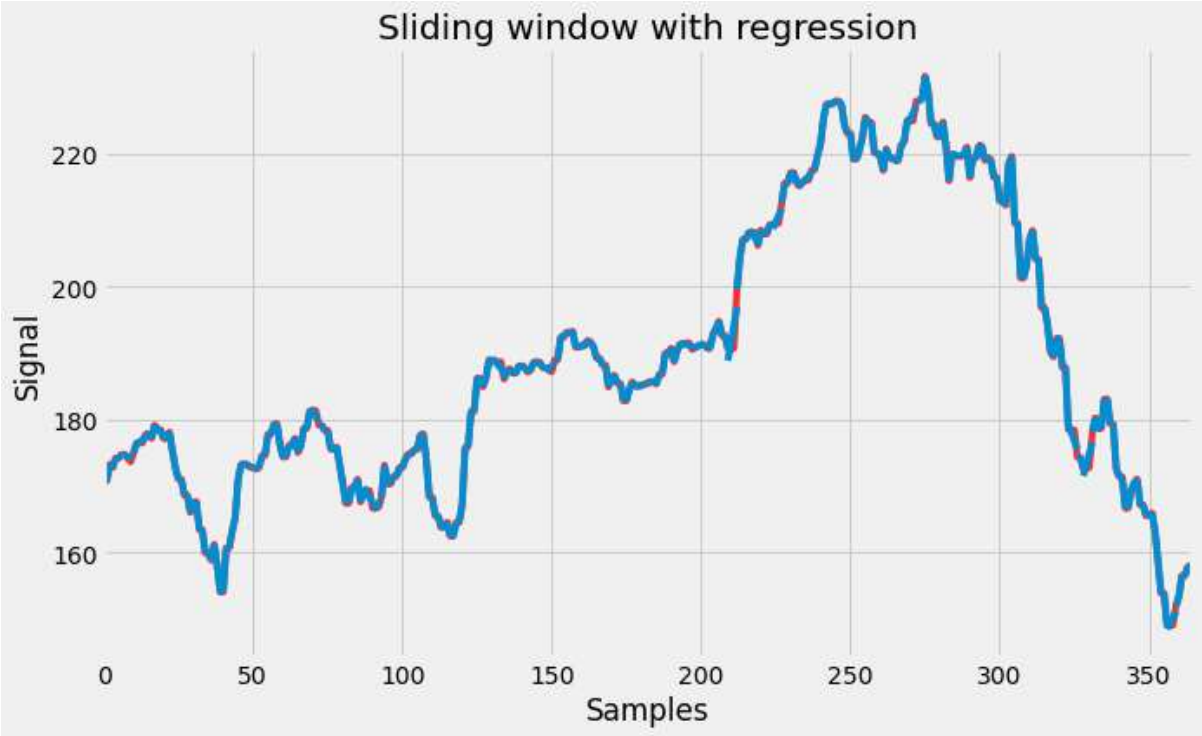
2. Segmentation

Bottom-up piecewise linear segmentation for the transformed and normalised log-return time series. For this we used three python files. Segment that has the implementation of bottom-up piecewise, fit that regression mse sum square error and interpolate methods and wrappers include the least square line fit function returns the parameters and error for a least squares line fit of one segment of a sequence

In [31]:
```
from matplotlib.pylab import gca, figure, plot, subplot, title, xlabel, ylabel
, xlim,show
from matplotlib.lines import Line2D
import segment
import fit

def draw_plot(data,plot_title):
    plot(range(len(data)),data,alpha=0.8,color='red')
    title(plot_title)
    xlabel("Samples")
    ylabel("Signal")
    xlim((0,len(data)-1))

def draw_segments(segments):
    ax = gca()
    for segment in segments:
        line = Line2D((segment[0],segment[2]),(segment[1],segment[3]))
        ax.add_line(line)
```

Mean square errors tolerance levels: 1%

In [32]:
```
data = data_copy1['Average']

max_error = 0.01

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error)
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

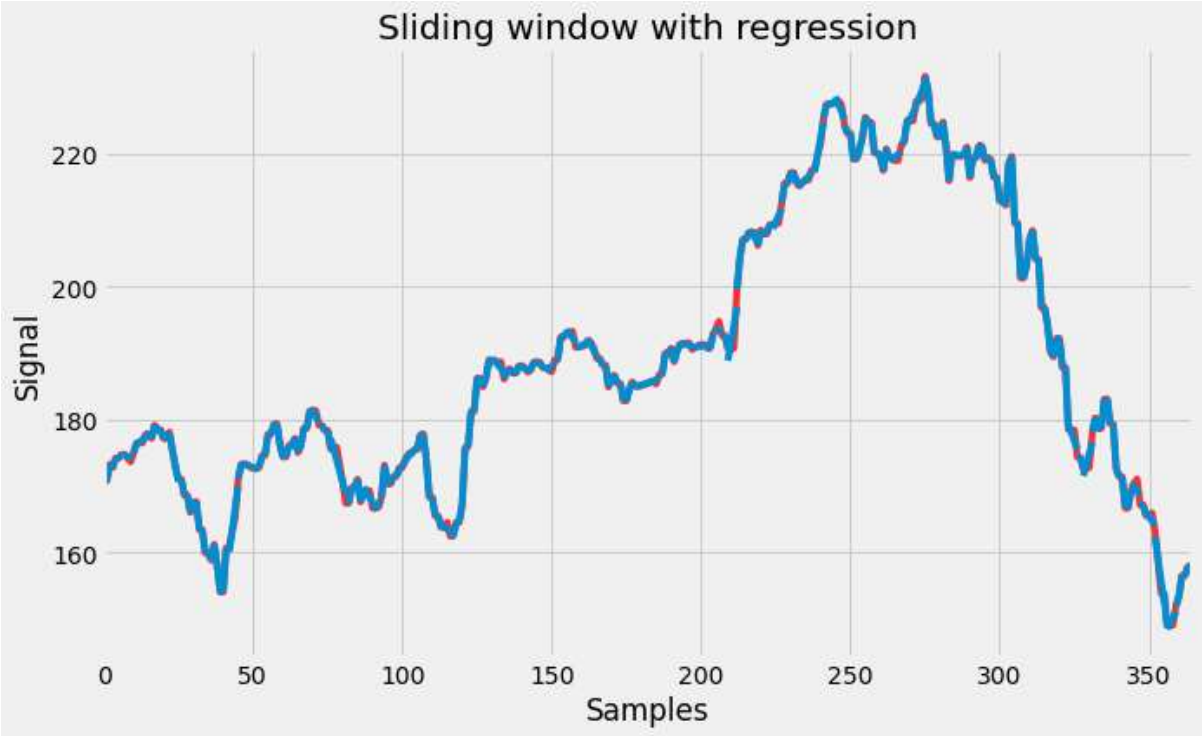


In [33]:
```
data = data_copy2['Average']

max_error = 0.01

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

In [34]:
```
data = data_copy3['Average']

max_error = 0.01

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

Mean square errors tolerance levels: 5%

In [35]:
```
data = data_copy1['Average']

max_error = 0.05

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error)
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

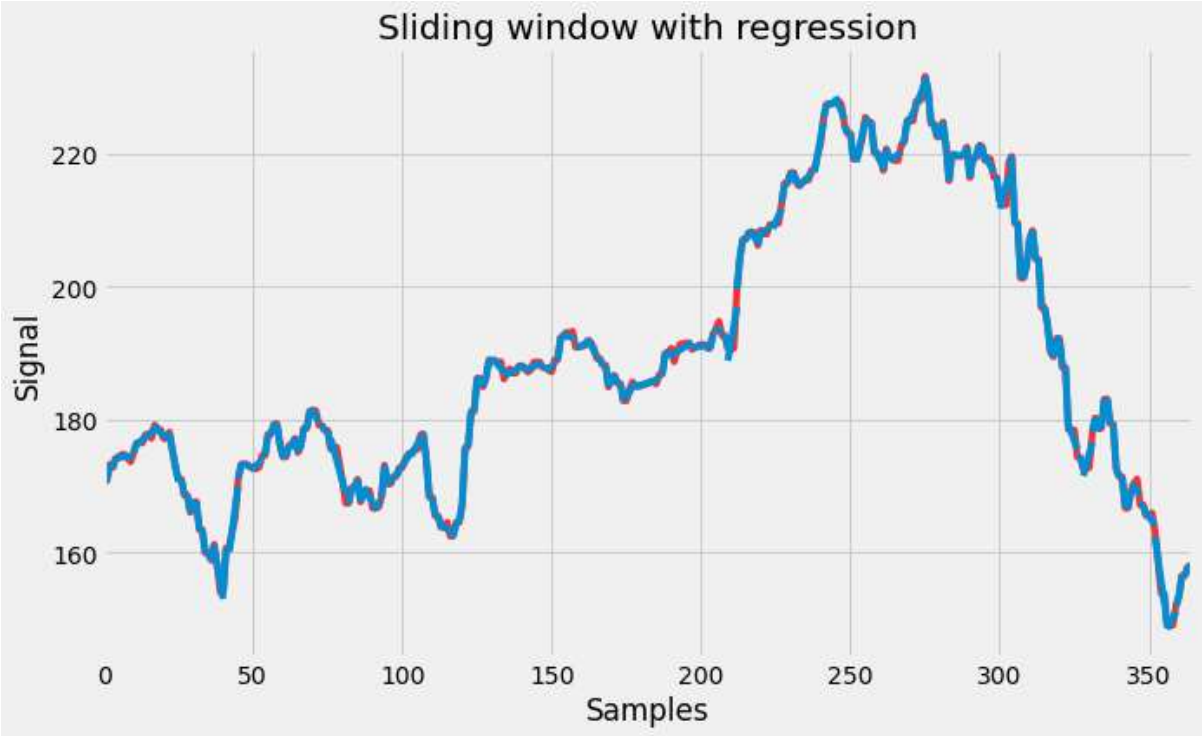


In [36]:
```
data = data_copy2['Average']

max_error = 0.05

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

In [37]:
```
data = data_copy3['Average']

max_error = 0.05

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

Mean square errors tolerance levels: 10%

In [38]:
```
data = data_copy1['Average']

max_error = 0.1

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```



In [39]:
```
data = data_copy2['Average']

max_error = 0.1

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

In [40]:
```
data = data_copy3['Average']

max_error = 0.1

#sliding window with regression
figure()
segments = segment.bottomupsegment(data, fit.regression, fit.sumsquared_error,
max_error )
draw_plot(data,"Sliding window with regression")
draw_segments(segments)
show()
```

The segments are similar for the mean squared error tolerance level for each dataset but are different from each other

3. Prediction

For Prediction

We transformed and normalised time series as a target $g(t)$ and other 2 as supporting data where we used 2018 2019 data as trainset and 2020 as test set and evaluated the error of the “next-day forecast”

In [41]:
```
trainset= pd.concat([a, b], axis=0)
```

In [42]:
```
trainset
```

Out[42]:

	Date	Open	High	Low	Close	Adj Close	Volume	Avera
2018-01-01	2018-01-01	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0	170.7799
2018-01-02	2018-01-02	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0	170.7799
2018-01-03	2018-01-03	172.529999	174.550003	171.960007	172.229996	166.324722	29517900.0	173.2550
2018-01-04	2018-01-04	172.539993	173.470001	172.080002	173.029999	167.097290	22434600.0	172.7750
2018-01-05	2018-01-05	173.440002	175.369995	173.050003	175.000000	168.999741	23660000.0	174.2099
...	...	...	...	...	...	...	...	
2019-12-27	2019-12-27	72.779999	73.492500	72.029999	72.449997	71.953598	146266000.0	72.7612
2019-12-28	2019-12-28	72.779999	73.492500	72.029999	72.449997	71.953598	146266000.0	72.7612
2019-12-29	2019-12-29	72.364998	73.172501	71.305000	72.879997	72.380653	144114400.0	72.2387
2019-12-30	2019-12-30	72.364998	73.172501	71.305000	72.879997	72.380653	144114400.0	72.2387
2019-12-31	2019-12-31	72.482498	73.419998	72.379997	73.412498	72.909500	100805600.0	72.8999

730 rows × 8 columns

In [43]:
```
testset=c
```

In [44]:
```
testset
```

Out[44]:

	Date	Open	High	Low	Close	Adj Close	Volume	Averag
2020-01-01	2020-01-01	74.059998	75.150002	73.797501	75.087502	74.573036	135480400.0	74.47375
2020-01-02	2020-01-02	74.059998	75.150002	73.797501	75.087502	74.573036	135480400.0	74.47375
2020-01-03	2020-01-03	74.287498	75.144997	74.125000	74.357498	73.848030	146322800.0	74.63499
2020-01-04	2020-01-04	74.287498	75.144997	74.125000	74.357498	73.848030	146322800.0	74.63499
2020-01-05	2020-01-05	73.447502	74.989998	73.187500	74.949997	74.436470	118387200.0	74.08874
...	...	...	...	...	...	...	...	
2020-12-27	2020-12-27	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.28500
2020-12-28	2020-12-28	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.28500
2020-12-29	2020-12-29	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.28500
2020-12-30	2020-12-30	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.28500
2020-12-31	2020-12-31	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.28500

366 rows × 8 columns

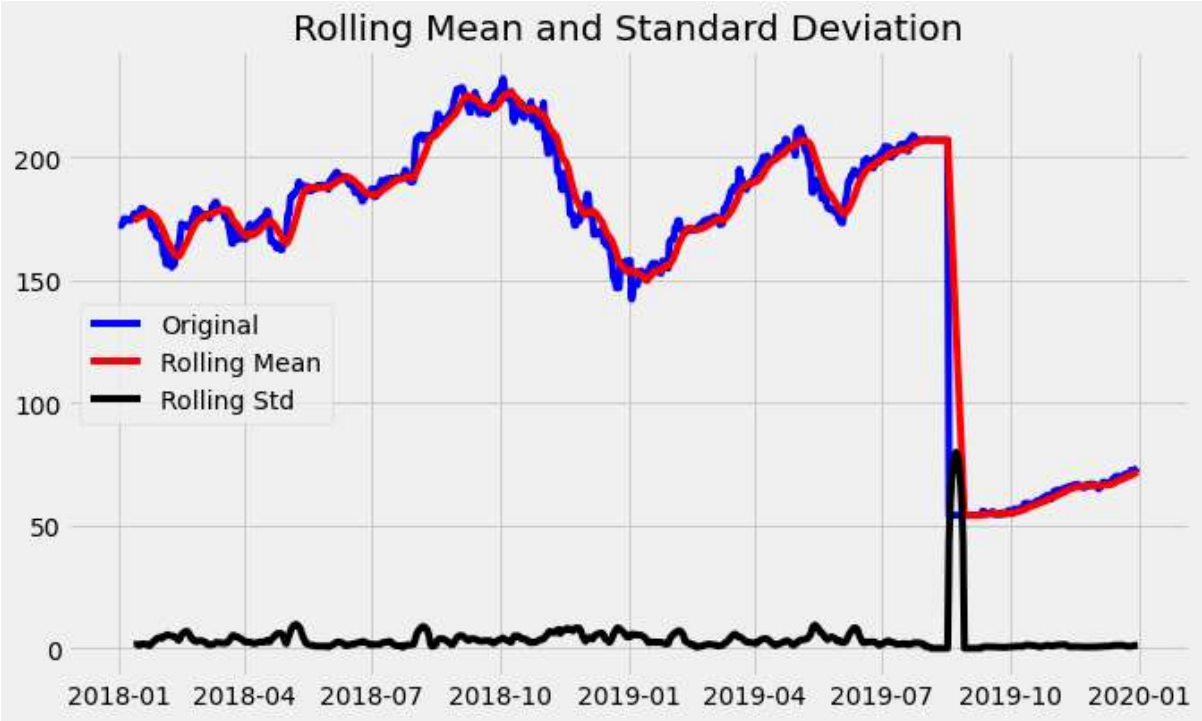
We had to build ARIMA model for predicting forecast series

1st step to make data stationary Steps to make our data stationary.

We use Dickey Fuller test to check the stationarity of the series.

First step Dickey-Fuller test.

In [45]:
```
def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = timeseries.rolling(12).mean()
 rolstd = timeseries.rolling(12).std()
 #Plot rolling statistics:
 plt.plot(timeseries, color='blue',label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
 plt.plot(rolstd, color='black', label = 'Rolling Std')
 plt.legend(loc='best')
 plt.title('Rolling Mean and Standard Deviation')
 plt.show(block=False)

 print("Results of dickey fuller test")
 adft = adfuller(timeseries,autolag='AIC')
 # output for dft will give us without defining what the values are.
 #hence we manually write what values does 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(trainset['Close'])
```

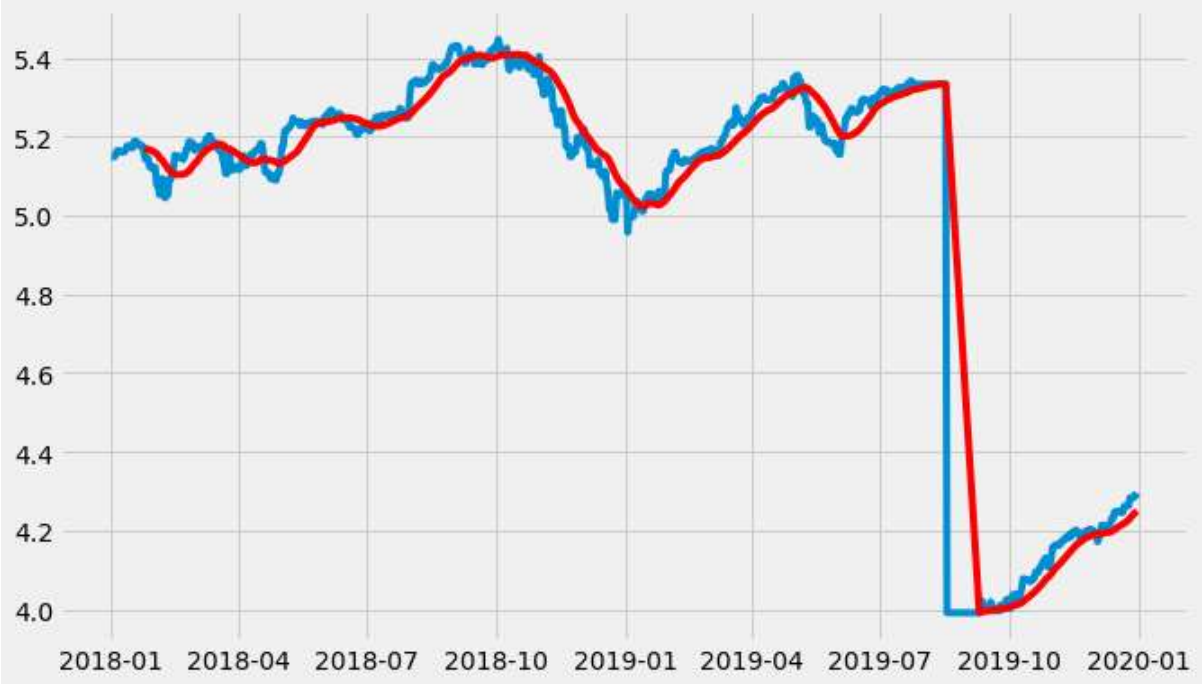



```
Results of dickey fuller test
Test Statistics                  -1.114148
p-value                           0.709397
No. of lags used                  0.000000
Number of observations used     729.000000
critical value (1%)              -3.439352
critical value (5%)              -2.865513
critical value (10%)             -2.568886
dtype: float64
```

The statistics shows that the time series is non-stationary as Test Statistic > Critical value, the p-value is greater than 5%, and we can see an increasing trend in the data. So, firstly we will try to make the data stationary. For doing so, we need to remove the trend and seasonality from the data.

Removing trend and seasonality to make data Stationary

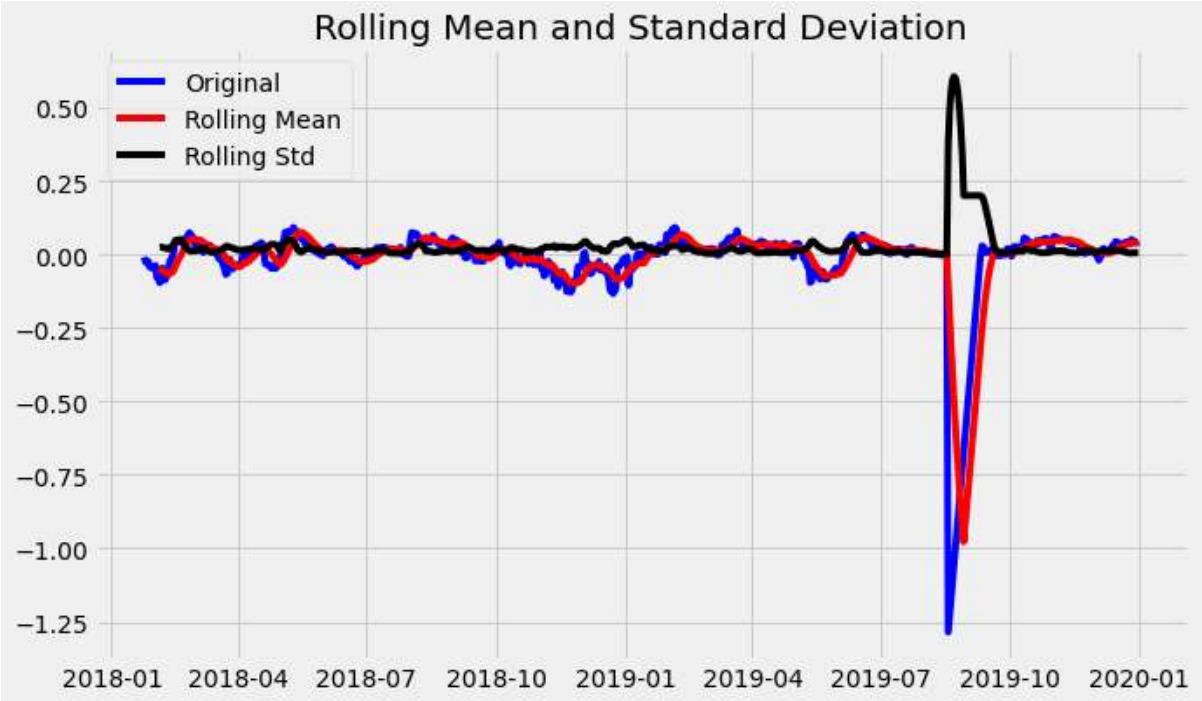
In [46]:
```
train_log = np.log(trainset['Close'])
test_log = np.log(testset['Close'])
moving_avg = train_log.rolling(24).mean()
plt.plot(train_log)
plt.plot(moving_avg, color = 'red')
plt.show()
```



In [47]:
```
train_log_moving_avg_diff = train_log - moving_avg
```

Since we took the average of values, rolling mean is not defined for the first few values. So let's drop those null values.

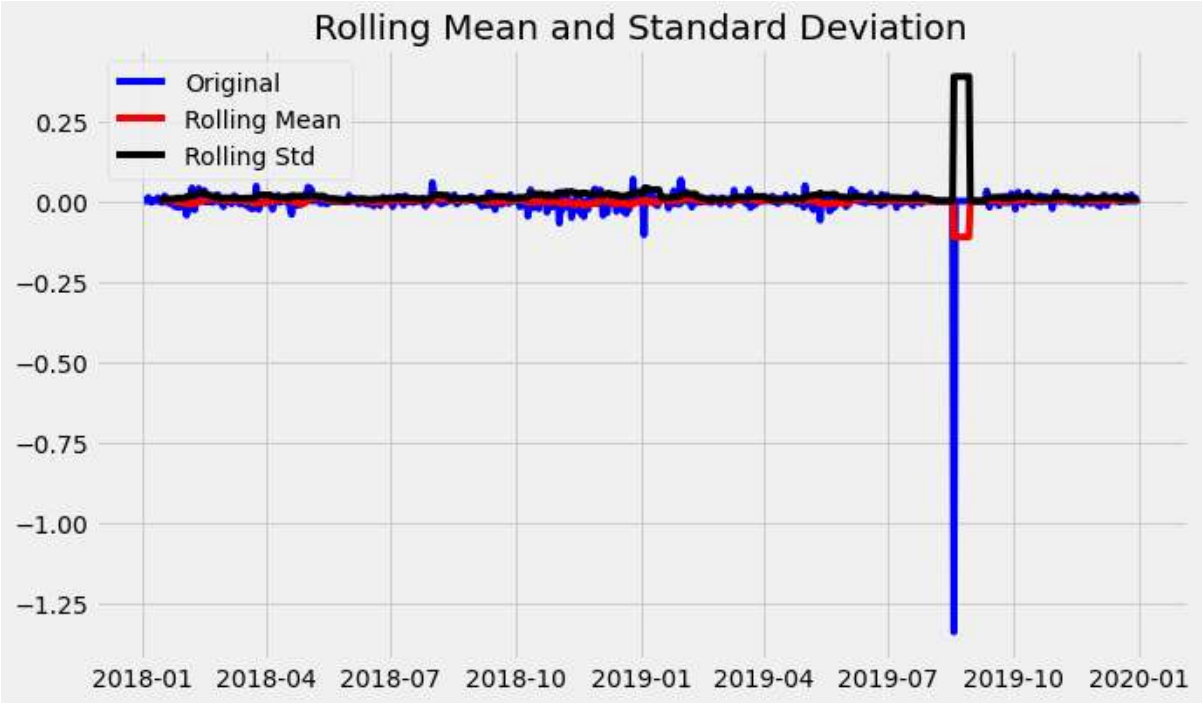
In [48]:
```
train_log_moving_avg_diff.dropna(inplace = True),
test_stationarity(train_log_moving_avg_diff)
```




```
Results of dickey fuller test
Test Statistics                  -4.808346
p-value                           0.000052
No. of lags used                  0.000000
Number of observations used     706.000000
critical value (1%)              -3.439646
critical value (5%)              -2.865643
critical value (10%)             -2.568955
dtype: float64
```

We can see that the Test Statistic is less than the Critical Value and the p-value is less than 5%. So, we can be confident that the trend is almost removed. Let's now stabilize the mean of the time series which is also a requirement for a stationary time series. Differencing can help to make the series stable and eliminate the trend.

In [49]:
```
train_log_diff = train_log - train_log.shift(1)
test_stationarity(train_log_diff.dropna())
```

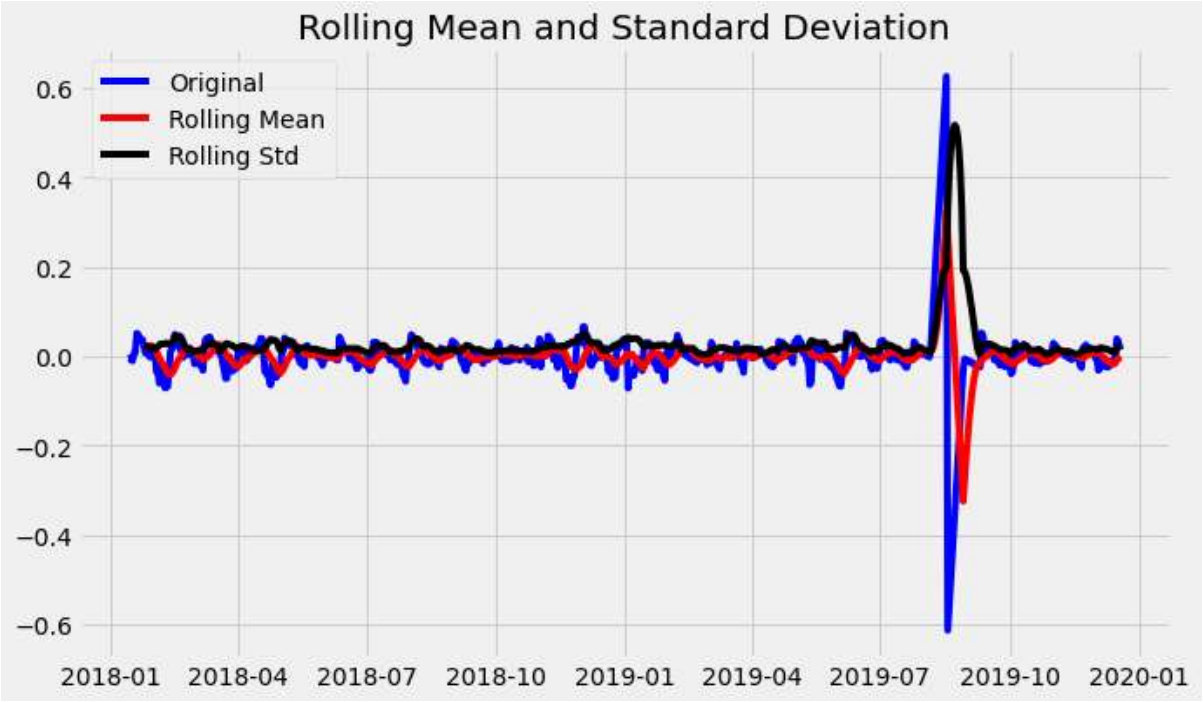



```
Results of dickey fuller test
Test Statistics                -26.968676
p-value                          0.000000
No. of lags used                 0.000000
Number of observations used    728.000000
critical value (1%)             -3.439364
critical value (5%)             -2.865518
critical value (10%)            -2.568888
dtype: float64
```

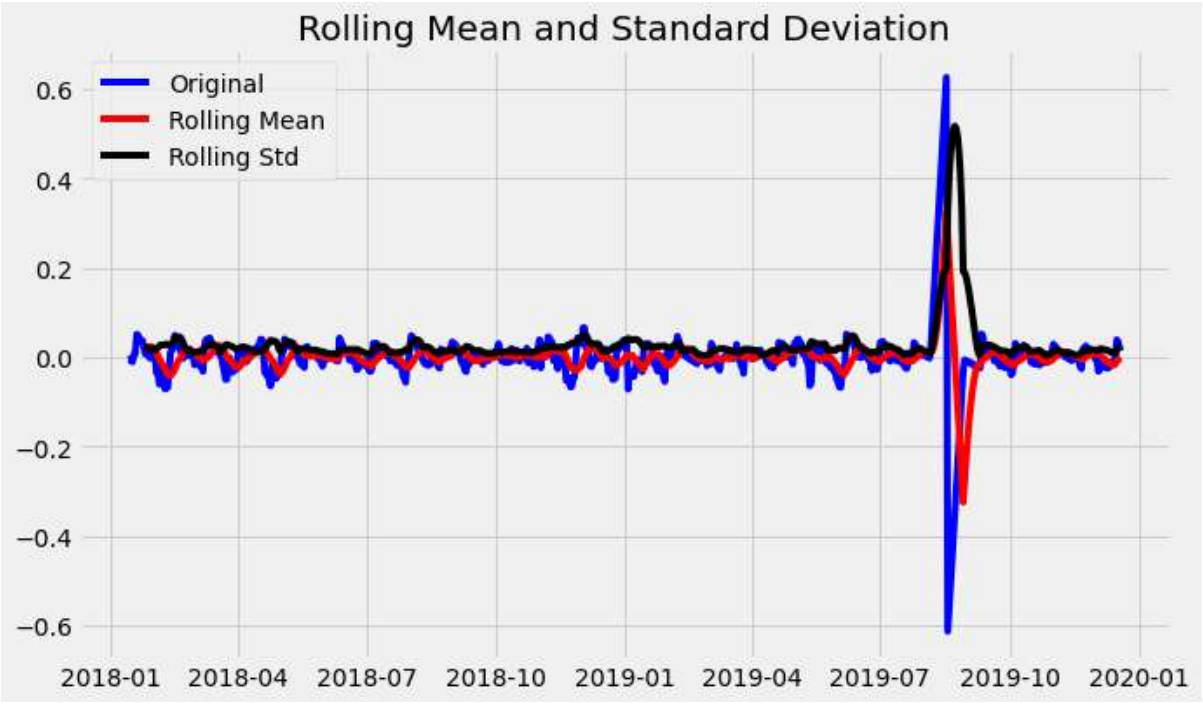
Now we will decompose the time series into trend and seasonality and will get the residual which is the random variation in the series. Removing Seasonality By seasonality, we mean periodic fluctuations. A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, or day of the week). We will use seasonal decompose to decompose the time series into trend, seasonality and residuals. We can see the trend, residuals and the seasonality clearly in the above graph. Seasonality shows a constant trend in counter. We use the code below to check the stationarity of residuals.

In [50]:
```
decomposition = seasonal_decompose(pd.DataFrame(train_log).Close.values, freq
= 24)

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

In [51]:
```
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
```




```
Results of dickey fuller test
Test Statistics                -1.056232e+01
p-value                         7.671554e-19
No. of lags used                7.000000e+00
Number of observations used     6.980000e+02
critical value (1%)            -3.439753e+00
critical value (5%)            -2.865690e+00
critical value (10%)           -2.568980e+00
dtype: float64
```

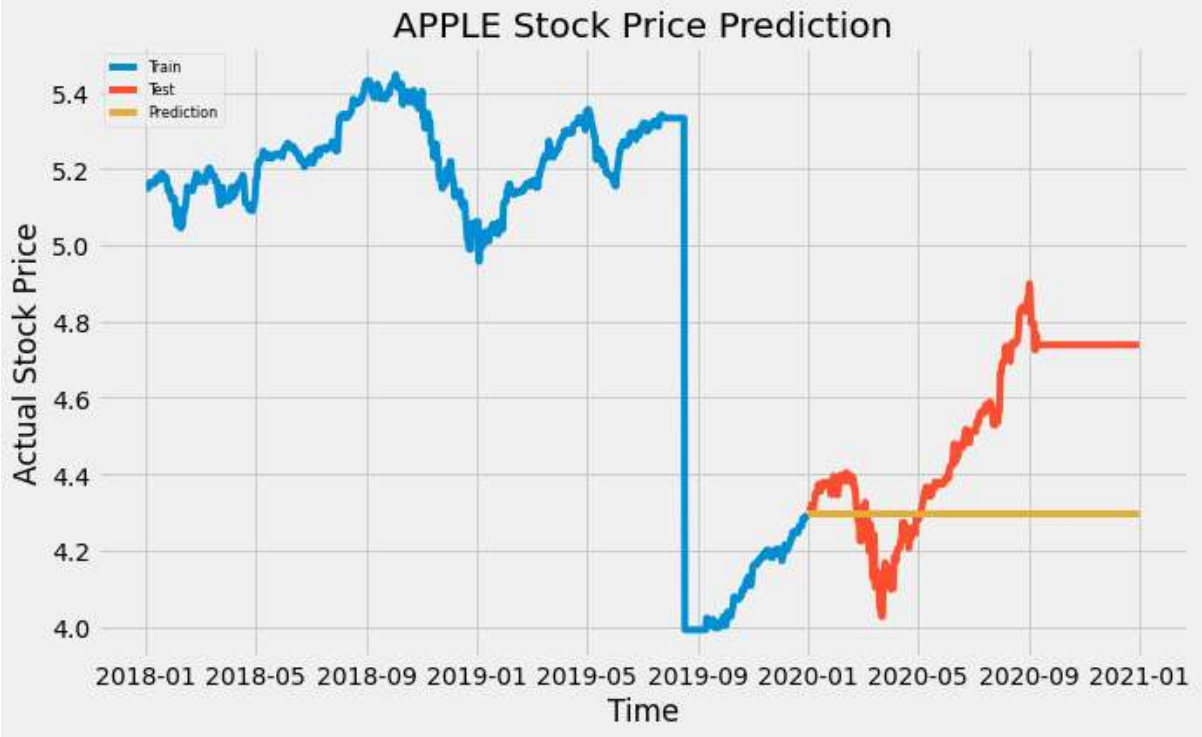


```
Results of dickey fuller test
Test Statistics               -1.056232e+01
p-value                        7.671554e-19
No. of lags used               7.000000e+00
Number of observations used    6.980000e+02
critical value (1%)           -3.439753e+00
critical value (5%)           -2.865690e+00
critical value (10%)          -2.568980e+00
dtype: float64
```

Fit Auto ARIMA: Fit the model on the univariate series. Predict values on validation set: Make predictions on the test set. Calculate RMSE: Check the performance of the model using the predicted values against the actual values.

In [52]:
```
model = auto_arima(train_log, trace=True, error_action='ignore', suppress_warn
ings=True)
model.fit(train_log)
forecast = model.predict(n_periods=len(testset))
forecast = pd.DataFrame(forecast,index = test_log.index,columns=['Prediction'
])
#plot the predictions for validation set
plt.plot(train_log, label='Train')
plt.plot(test_log, label='Test')
plt.plot(forecast, label='Prediction')
plt.title('APPLE Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-2243.103, Time=0.37 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-2251.102, Time=0.14 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-2249.103, Time=0.30 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-2249.103, Time=0.42 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-2252.727, Time=0.17 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-2247.103, Time=0.34 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 1.801 seconds
```



In [53]:
```
rms = sqrt(mean_squared_error(test_log,forecast))
print("RMSE: ", rms)
```

```
RMSE:  0.3119861247938011
```

A RMSE of 0 indicates a perfect fit with no errors. The smaller the RMSE score the better the model. A RMSE score of 0.3119861247938011 indicates a good model and we have plotted it as well.

Now completing the remaining parts similarly so the subpart number 1 is we used 2018 dataset as train set and 2019 as test set

In [54]:
```
trainset= a
```

In [55]:
```
trainset
```

Out[55]:

	Date	Open	High	Low	Close	Adj Close	Volume	Averag
2018-01-01	2018-01-01	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0	170.77999
2018-01-02	2018-01-02	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0	170.77999
2018-01-03	2018-01-03	172.529999	174.550003	171.960007	172.229996	166.324722	29517900.0	173.25500
2018-01-04	2018-01-04	172.539993	173.470001	172.080002	173.029999	167.097290	22434600.0	172.77500
2018-01-05	2018-01-05	173.440002	175.369995	173.050003	175.000000	168.999741	23660000.0	174.20999
...	...	...	...	...	...	...	...	...
2018-12-27	2018-12-27	155.839996	156.770004	150.070007	156.149994	153.059998	53117100.0	153.42000
2018-12-28	2018-12-28	157.500000	158.520004	154.550003	156.229996	153.138428	42291400.0	156.53500
2018-12-29	2018-12-29	157.500000	158.520004	154.550003	156.229996	153.138428	42291400.0	156.53500
2018-12-30	2018-12-30	158.529999	159.360001	156.479996	157.740005	154.618546	35003500.0	157.91999
2018-12-31	2018-12-31	158.529999	159.360001	156.479996	157.740005	154.618546	35003500.0	157.91999

365 rows × 8 columns

In [56]:
```
testset=b
```

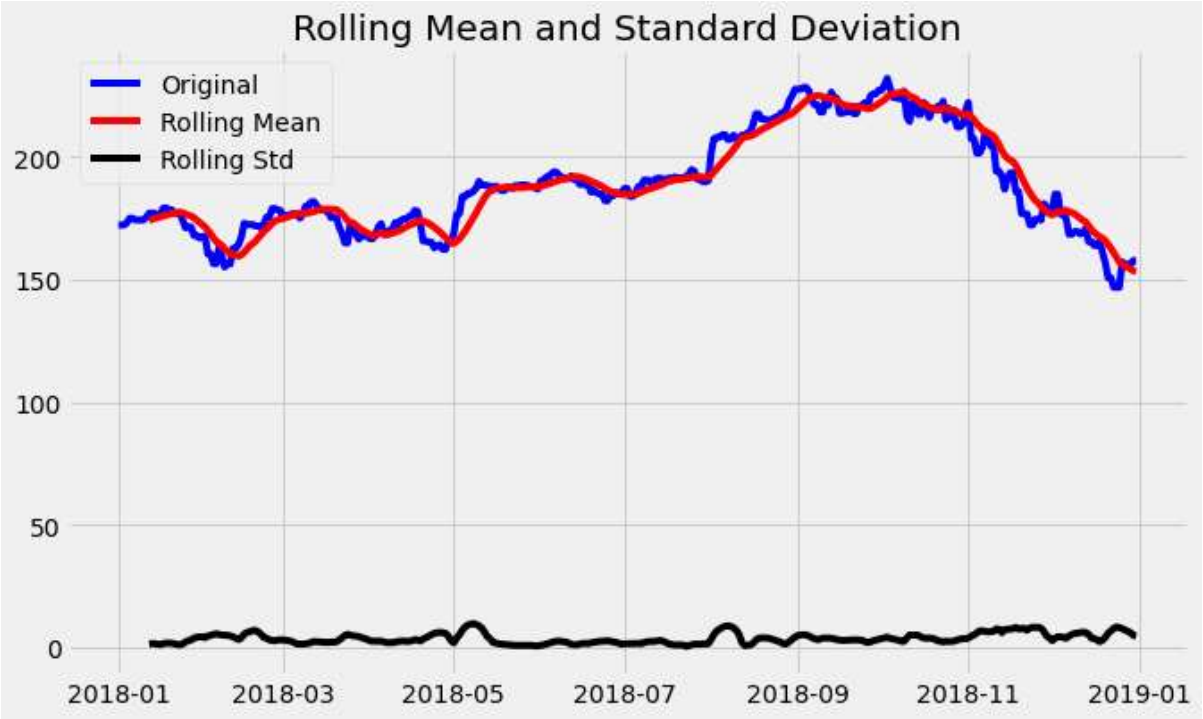
In [57]:
```
testset
```

Out[57]:

	Date	Open	High	Low	Close	Adj Close	Volume	Avera
2019-01-01	2019-01-01	154.889999	158.850006	154.229996	157.919998	154.794983	37039700.0	156.5400
2019-01-02	2019-01-02	154.889999	158.850006	154.229996	157.919998	154.794983	37039700.0	156.5400
2019-01-03	2019-01-03	143.979996	145.720001	142.000000	142.190002	139.376251	91312200.0	143.8600
2019-01-04	2019-01-04	144.529999	148.550003	143.800003	148.259995	145.326126	58607100.0	146.1750
2019-01-05	2019-01-05	144.529999	148.550003	143.800003	148.259995	145.326126	58607100.0	146.1750
...	...	...	...	...	...	...	...	...
2019-12-27	2019-12-27	72.779999	73.492500	72.029999	72.449997	71.953598	146266000.0	72.7612
2019-12-28	2019-12-28	72.779999	73.492500	72.029999	72.449997	71.953598	146266000.0	72.7612
2019-12-29	2019-12-29	72.364998	73.172501	71.305000	72.879997	72.380653	144114400.0	72.2387
2019-12-30	2019-12-30	72.364998	73.172501	71.305000	72.879997	72.380653	144114400.0	72.2387
2019-12-31	2019-12-31	72.482498	73.419998	72.379997	73.412498	72.909500	100805600.0	72.8999

365 rows × 8 columns

In [58]:
```
def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = timeseries.rolling(12).mean()
 rolstd = timeseries.rolling(12).std()
 #Plot rolling statistics:
 plt.plot(timeseries, color='blue',label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
 plt.plot(rolstd, color='black', label = 'Rolling Std')
 plt.legend(loc='best')
 plt.title('Rolling Mean and Standard Deviation')
 plt.show(block=False)

 print("Results of dickey fuller test")
 adft = adfuller(timeseries,autolag='AIC')
 # output for dft will give us without defining what the values are.
 #hence we manually write what values does 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(trainset['Close'])
```

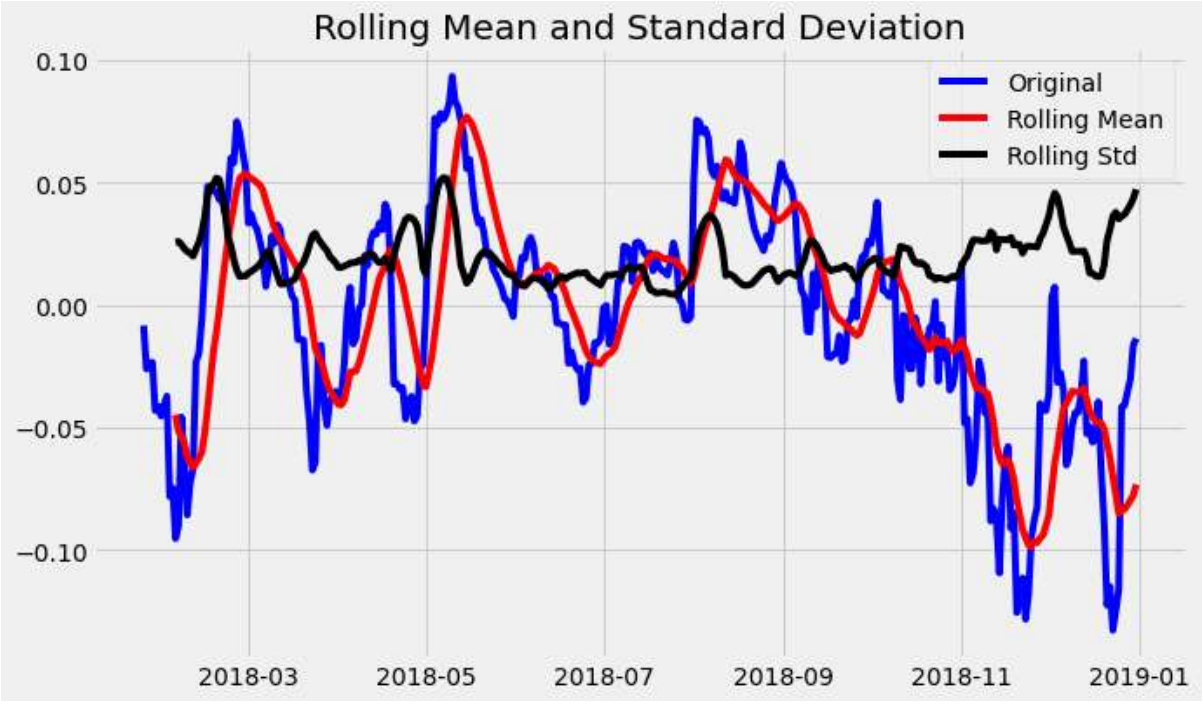



```
Results of dickey fuller test
Test Statistics                  -0.985586
p-value                           0.758485
No. of lags used                  0.000000
Number of observations used     364.000000
critical value (1%)              -3.448443
critical value (5%)              -2.869513
critical value (10%)             -2.571018
dtype: float64
```

In [59]:
```
train_log = np.log(trainset['Close'])
test_log = np.log(testset['Close'])
moving_avg = train_log.rolling(24).mean()
plt.plot(train_log)
plt.plot(moving_avg, color = 'red')
plt.show()
```

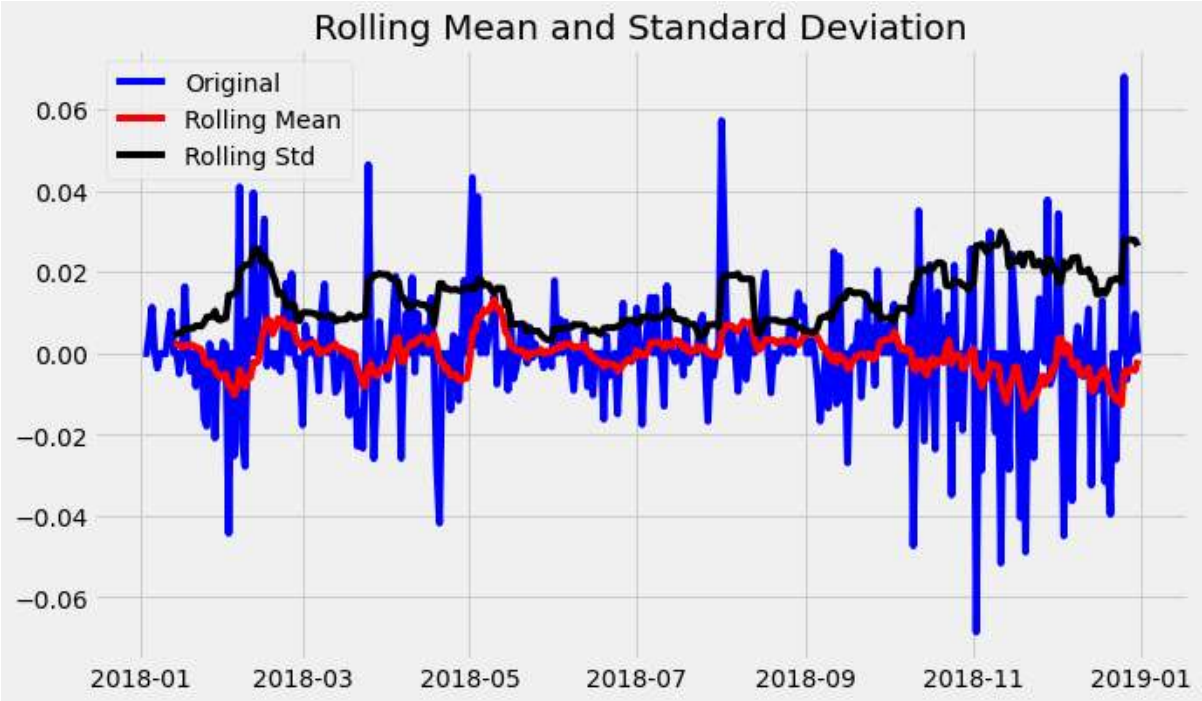
In [60]:
```
train_log_moving_avg_diff = train_log - moving_avg
```

In [61]:
```
train_log_moving_avg_diff.dropna(inplace = True),
test_stationarity(train_log_moving_avg_diff)
```




```
Results of dickey fuller test
Test Statistics                  -3.317439
p-value                           0.014117
No. of lags used                  1.000000
Number of observations used     340.000000
critical value (1%)              -3.449730
critical value (5%)              -2.870079
critical value (10%)             -2.571319
dtype: float64
```

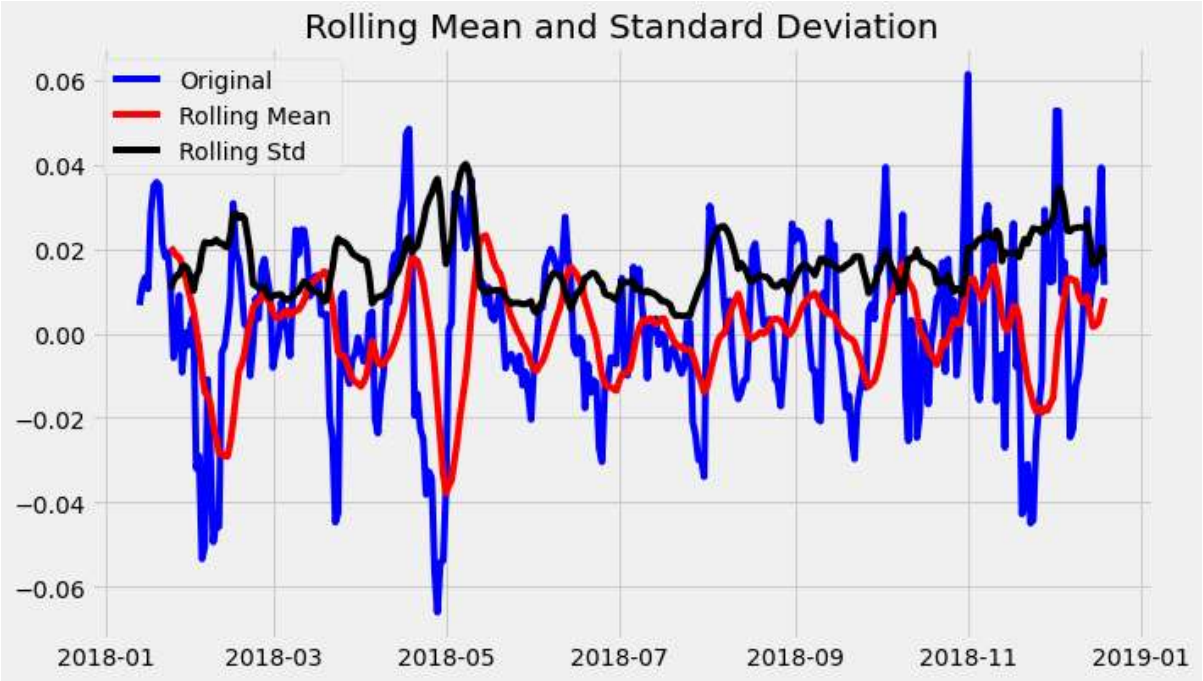
In [62]:
```
train_log_diff = train_log - train_log.shift(1)
test_stationarity(train_log_diff.dropna())
```



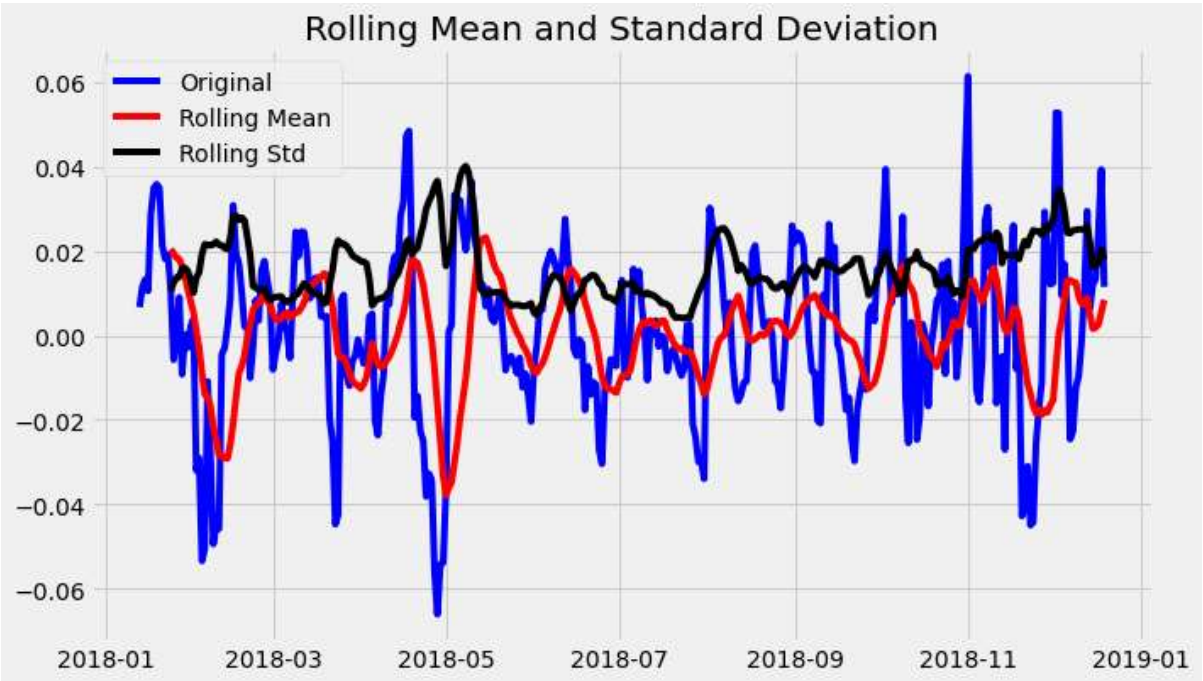

```
Results of dickey fuller test
Test Statistics                -1.825638e+01
p-value                         2.336541e-30
No. of lags used                0.000000e+00
Number of observations used     3.630000e+02
critical value (1%)            -3.448494e+00
critical value (5%)            -2.869535e+00
critical value (10%)           -2.571029e+00
dtype: float64
```

In [63]:
```
decomposition = seasonal_decompose(pd.DataFrame(train_log).Close.values, freq
= 24)

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

In [64]:
```
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
```



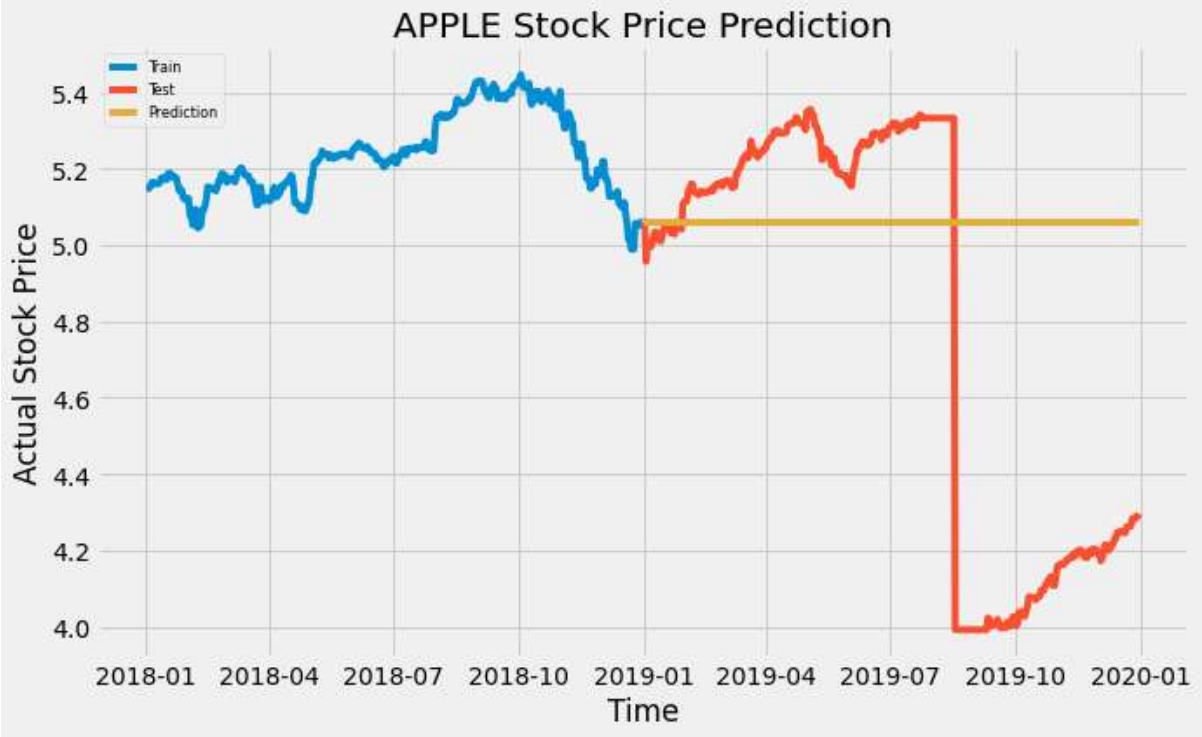

```
Results of dickey fuller test
Test Statistics                -6.561091e+00
p-value                         8.377780e-09
No. of lags used                1.600000e+01
Number of observations used     3.240000e+02
critical value (1%)            -3.450695e+00
critical value (5%)            -2.870502e+00
critical value (10%)           -2.571545e+00
dtype: float64
```



```
Results of dickey fuller test
Test Statistics                -6.561091e+00
p-value                         8.377780e-09
No. of lags used                1.600000e+01
Number of observations used     3.240000e+02
critical value (1%)            -3.450695e+00
critical value (5%)            -2.870502e+00
critical value (10%)           -2.571545e+00
dtype: float64
```

In [65]:
```
model = auto_arima(train_log, trace=True, error_action='ignore', suppress_warn
ings=True)
model.fit(train_log)
forecast = model.predict(n_periods=len(testset))
forecast = pd.DataFrame(forecast,index = test_log.index,columns=['Prediction'
])
#plot the predictions for validation set
plt.plot(train_log, label='Train')
plt.plot(test_log, label='Test')
plt.plot(forecast, label='Prediction')
plt.title('APPLE Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-2013.245, Time=0.31 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-2020.632, Time=0.08 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-2019.210, Time=0.26 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-2019.215, Time=0.23 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-2022.537, Time=0.16 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-2017.222, Time=0.20 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 1.250 seconds
```



In [66]:
```
rms = sqrt(mean_squared_error(test_log,forecast))
print("RMSE: ", rms)
```

```
RMSE:  0.6021089303722202
```

A RMSE score of 0.60 indicates not a good model. We can use other methods for time series forecasting and compare your results using accuracy metrics to pick the best model.

Subtask two 2018 as train set and 2020 as test set this is the 2nd part of prediction

In [67]:
```
trainset
```

Out[67]:

	Date	Open	High	Low	Close	Adj Close	Volume	Averag
2018-01-01	2018-01-01	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0	170.77999
2018-01-02	2018-01-02	170.160004	172.300003	169.259995	172.259995	166.353714	25555900.0	170.77999
2018-01-03	2018-01-03	172.529999	174.550003	171.960007	172.229996	166.324722	29517900.0	173.25500
2018-01-04	2018-01-04	172.539993	173.470001	172.080002	173.029999	167.097290	22434600.0	172.77500
2018-01-05	2018-01-05	173.440002	175.369995	173.050003	175.000000	168.999741	23660000.0	174.20999
...	...	...	...	...	...	...	...	.
2018-12-27	2018-12-27	155.839996	156.770004	150.070007	156.149994	153.059998	53117100.0	153.42000
2018-12-28	2018-12-28	157.500000	158.520004	154.550003	156.229996	153.138428	42291400.0	156.53500
2018-12-29	2018-12-29	157.500000	158.520004	154.550003	156.229996	153.138428	42291400.0	156.53500
2018-12-30	2018-12-30	158.529999	159.360001	156.479996	157.740005	154.618546	35003500.0	157.91999
2018-12-31	2018-12-31	158.529999	159.360001	156.479996	157.740005	154.618546	35003500.0	157.91999

365 rows × 8 columns

In [68]:
```
testset=c
```

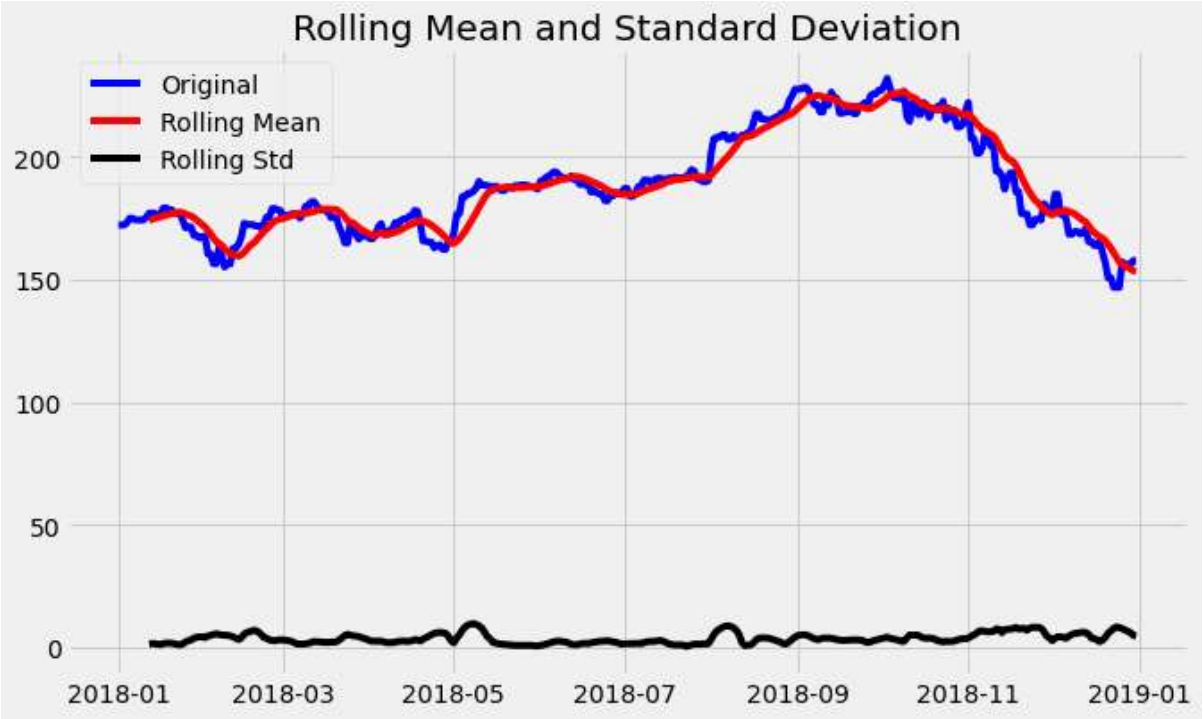
In [69]:
```
testset
```

Out[69]:

	Date	Open	High	Low	Close	Adj Close	Volume	Averag
2020-01-01	2020-01-01	74.059998	75.150002	73.797501	75.087502	74.573036	135480400.0	74.47375
2020-01-02	2020-01-02	74.059998	75.150002	73.797501	75.087502	74.573036	135480400.0	74.47375
2020-01-03	2020-01-03	74.287498	75.144997	74.125000	74.357498	73.848030	146322800.0	74.63499
2020-01-04	2020-01-04	74.287498	75.144997	74.125000	74.357498	73.848030	146322800.0	74.63499
2020-01-05	2020-01-05	73.447502	74.989998	73.187500	74.949997	74.436470	118387200.0	74.08874
...	...	...	...	...	...	...	...	...
2020-12-27	2020-12-27	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.2850(
2020-12-28	2020-12-28	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.2850(
2020-12-29	2020-12-29	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.2850(
2020-12-30	2020-12-30	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.2850(
2020-12-31	2020-12-31	120.360001	120.419998	114.150002	114.274597	114.274597	118115972.0	117.2850(

366 rows × 8 columns

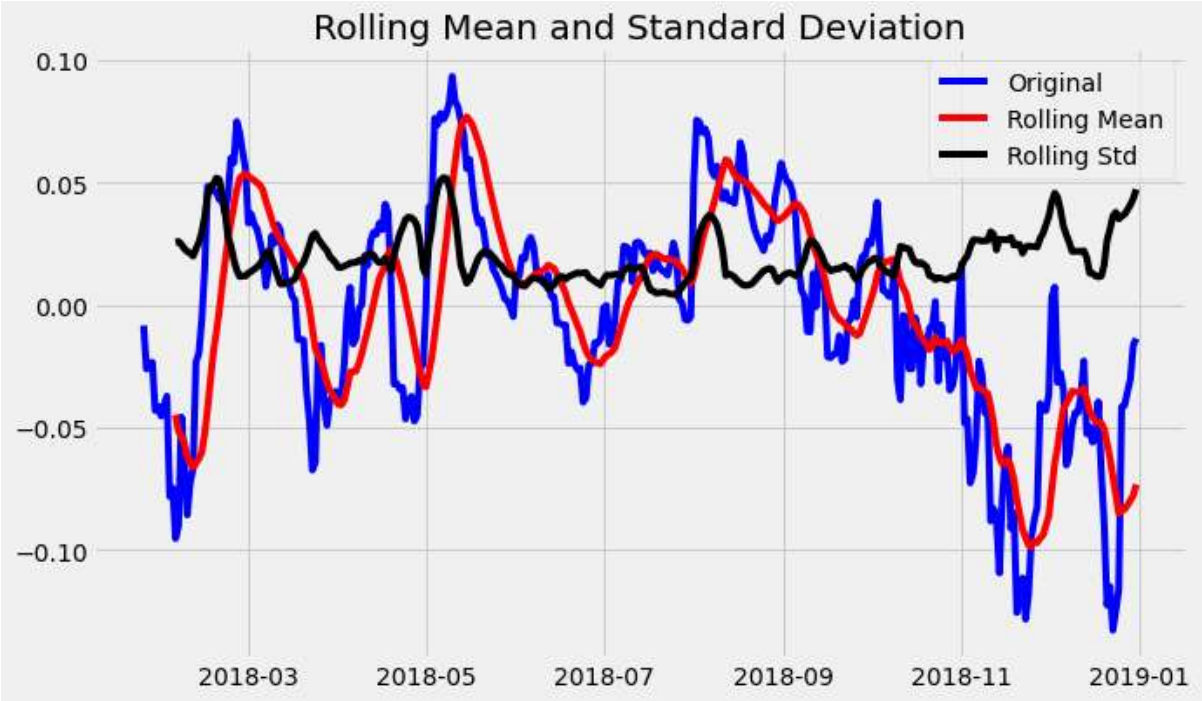
In [70]:
```
def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = timeseries.rolling(12).mean()
 rolstd = timeseries.rolling(12).std()
 #Plot rolling statistics:
 plt.plot(timeseries, color='blue',label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
 plt.plot(rolstd, color='black', label = 'Rolling Std')
 plt.legend(loc='best')
 plt.title('Rolling Mean and Standard Deviation')
 plt.show(block=False)

 print("Results of dickey fuller test")
 adft = adfuller(timeseries,autolag='AIC')
 # output for dft will give us without defining what the values are.
 #hence we manually write what values does 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(trainset['Close'])
```




```
Results of dickey fuller test
Test Statistics                  -0.985586
p-value                           0.758485
No. of lags used                  0.000000
Number of observations used     364.000000
critical value (1%)              -3.448443
critical value (5%)              -2.869513
critical value (10%)             -2.571018
dtype: float64
```

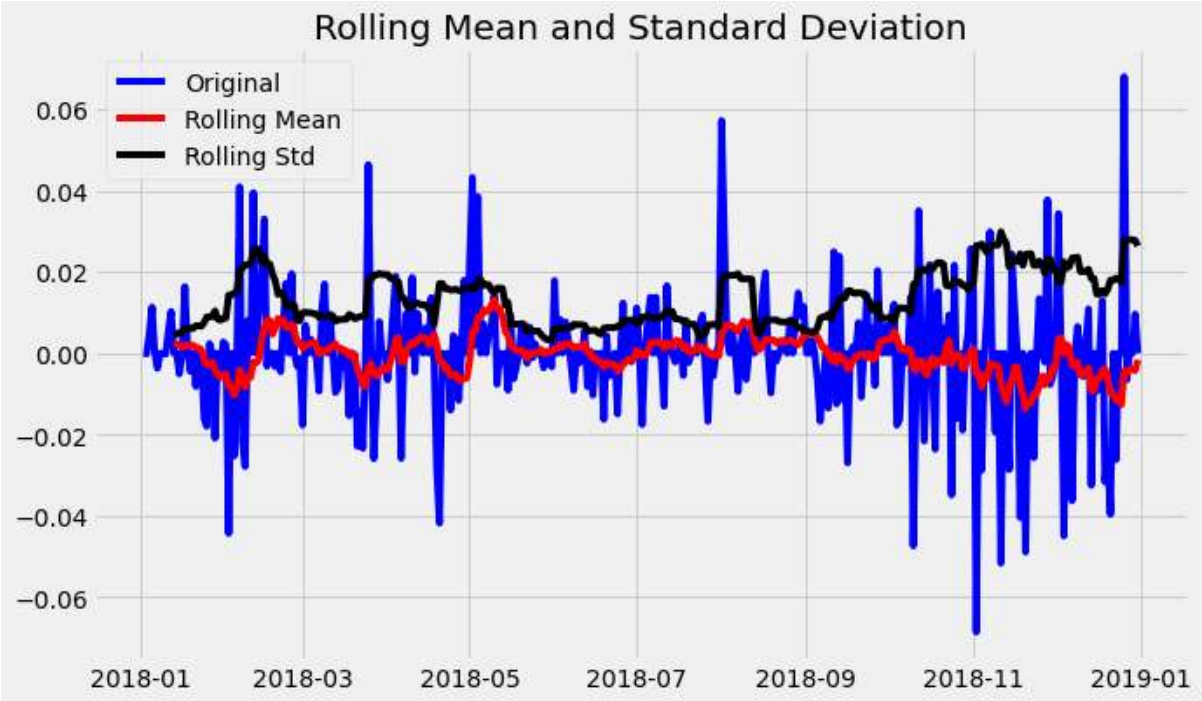
In [71]:
```
train_log = np.log(trainset['Close'])
test_log = np.log(testset['Close'])
moving_avg = train_log.rolling(24).mean()
plt.plot(train_log)
plt.plot(moving_avg, color = 'red')
plt.show()
```

In [72]:
```
train_log_moving_avg_diff.dropna(inplace = True),
test_stationarity(train_log_moving_avg_diff)
```




```
Results of dickey fuller test
Test Statistics                  -3.317439
p-value                           0.014117
No. of lags used                  1.000000
Number of observations used     340.000000
critical value (1%)              -3.449730
critical value (5%)              -2.870079
critical value (10%)             -2.571319
dtype: float64
```

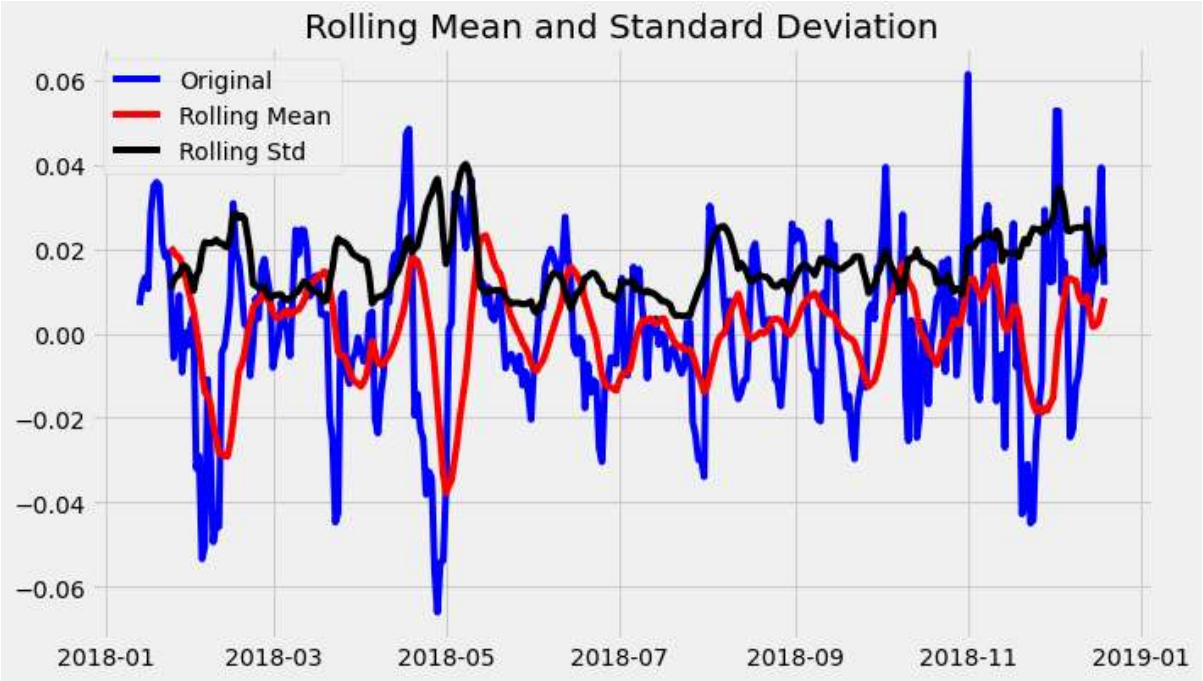
In [73]:
```
train_log_diff = train_log - train_log.shift(1)
test_stationarity(train_log_diff.dropna())
```



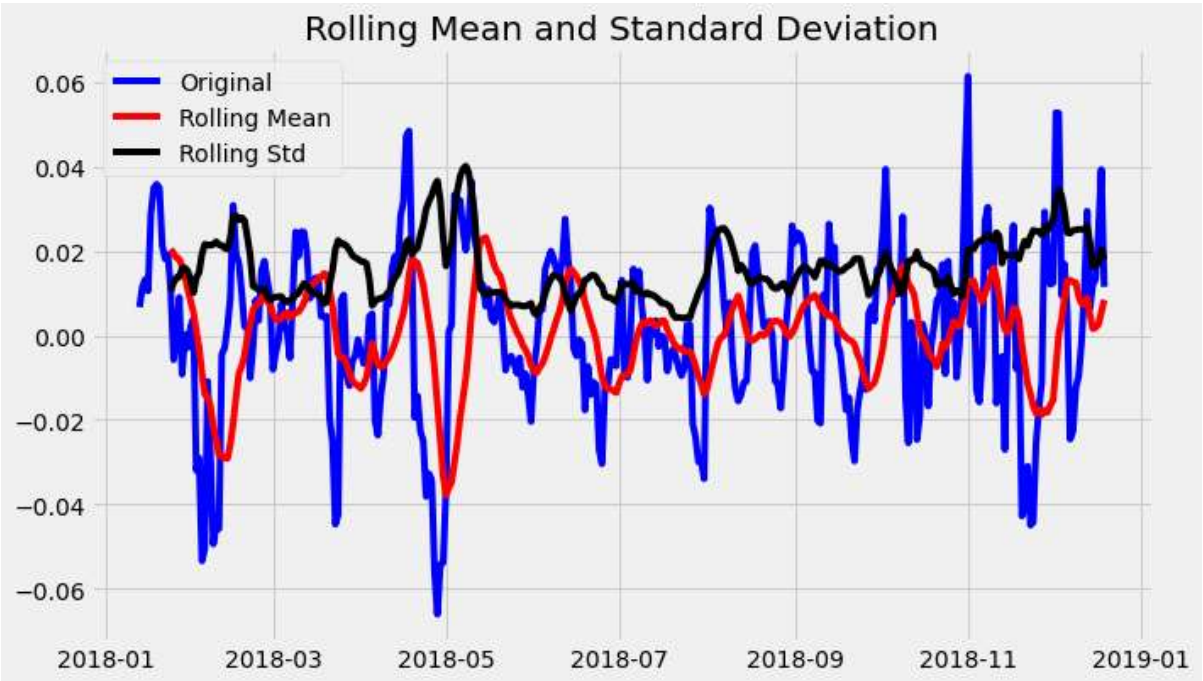

```
Results of dickey fuller test
Test Statistics                -1.825638e+01
p-value                         2.336541e-30
No. of lags used                0.000000e+00
Number of observations used     3.630000e+02
critical value (1%)            -3.448494e+00
critical value (5%)            -2.869535e+00
critical value (10%)           -2.571029e+00
dtype: float64
```

In [74]:
```
decomposition = seasonal_decompose(pd.DataFrame(train_log).Close.values, freq
= 24)

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

In [75]:
```
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
```



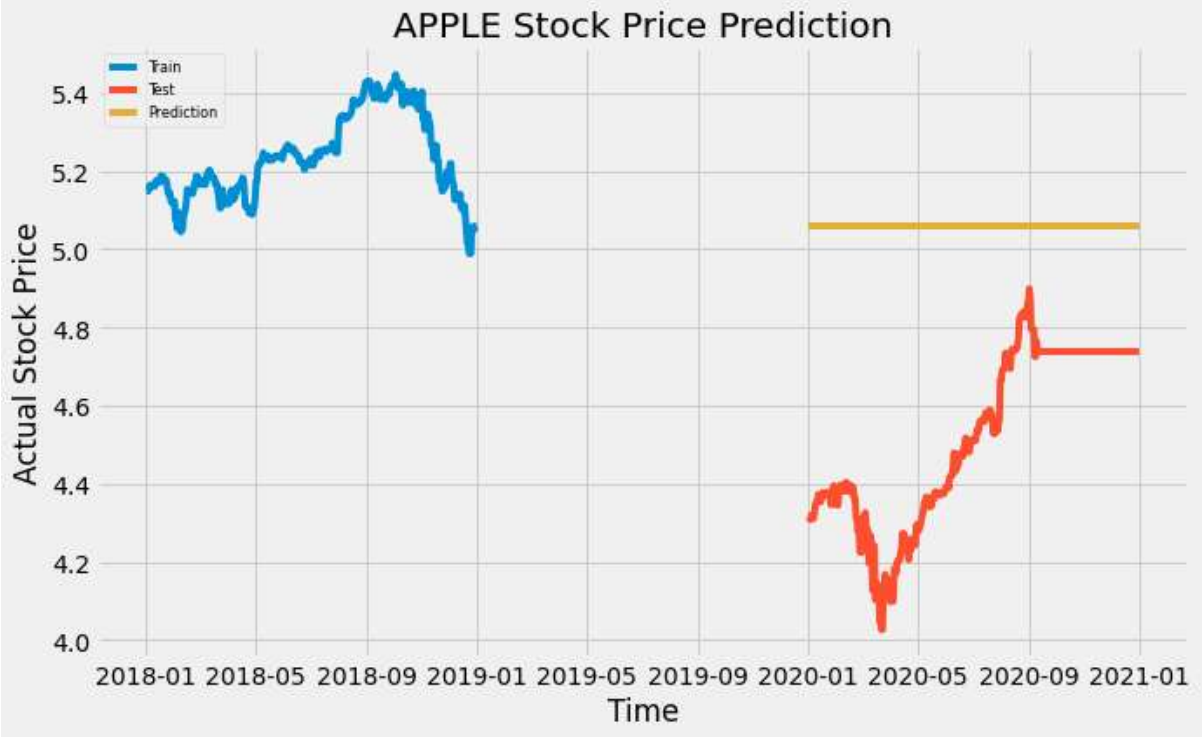

```
Results of dickey fuller test
Test Statistics                -6.561091e+00
p-value                         8.377780e-09
No. of lags used                1.600000e+01
Number of observations used     3.240000e+02
critical value (1%)            -3.450695e+00
critical value (5%)            -2.870502e+00
critical value (10%)           -2.571545e+00
dtype: float64
```



```
Results of dickey fuller test
Test Statistics                -6.561091e+00
p-value                         8.377780e-09
No. of lags used                1.600000e+01
Number of observations used     3.240000e+02
critical value (1%)            -3.450695e+00
critical value (5%)            -2.870502e+00
critical value (10%)           -2.571545e+00
dtype: float64
```

In [76]:
```
model = auto_arima(train_log, trace=True, error_action='ignore', suppress_warn
ings=True)
model.fit(train_log)
forecast = model.predict(n_periods=len(testset))
forecast = pd.DataFrame(forecast,index = test_log.index,columns=['Prediction'
])
#plot the predictions for validation set
plt.plot(train_log, label='Train')
plt.plot(test_log, label='Test')
plt.plot(forecast, label='Prediction')
plt.title('APPLE Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-2013.245, Time=0.28 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-2020.632, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-2019.210, Time=0.24 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-2019.215, Time=0.15 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-2022.537, Time=0.06 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-2017.222, Time=0.18 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 1.001 seconds
```



In [77]:
```
rms = sqrt(mean_squared_error(test_log,forecast))
print("RMSE: ", rms)
```

```
RMSE:  0.5862092552866461
```

0.58 is partially a normal model and we can use other methods for time series forecasting and compare our results using accuracy metrics to pick the best model.

4. Adaptive predictors.

In the last part that is for the adaptive predictors we used the SARIMAX Model "Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors, or SARIMAX, is an extension of the ARIMA class of models. Intuitively, ARIMA models compose 2 parts: the autoregressive term (AR) and the moving-average term (MA). The former views the value at one time just as a weighted sum of past values." We used ARIMA model and for the data we got we splitted our data on delta= 5, 10 and 30 and created and predicted the adaptive predictors and we used 2020's dataset for this exercise.

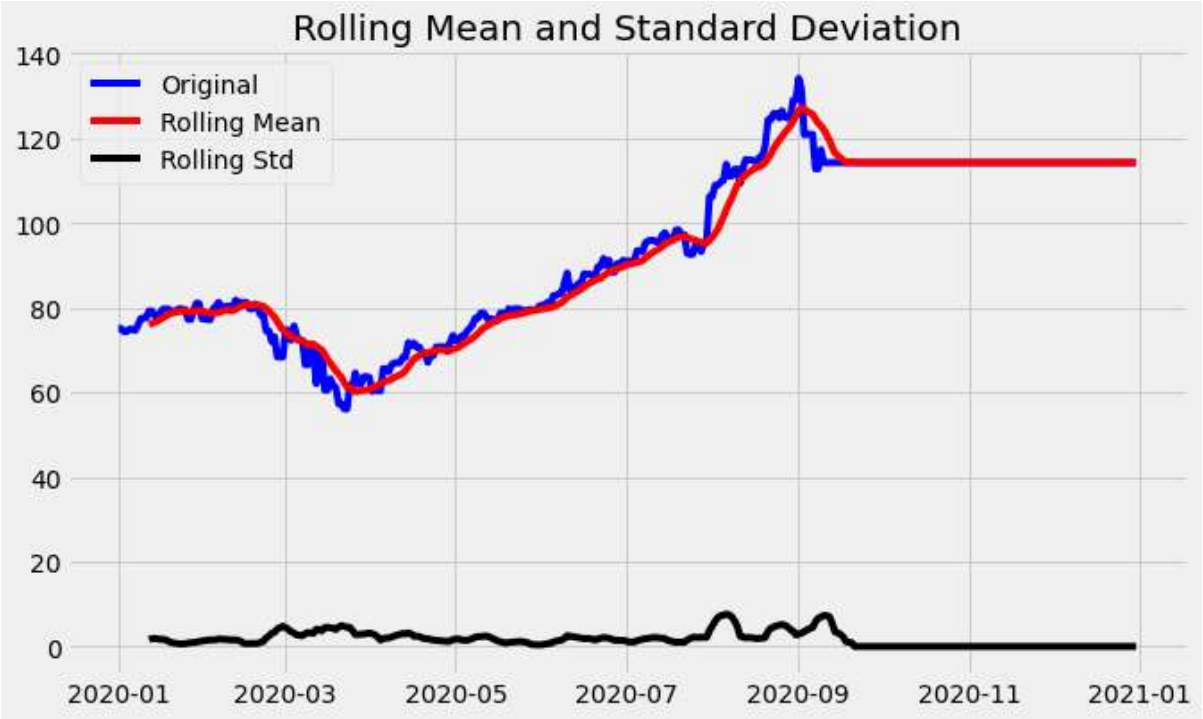
We had to build ARIMA model for predicting forecast series

1st step to make data stationary Steps to make our data stationary.

We use Dickey Fuller test to check the stationarity of the series.

First step Dickey-Fuller test.

In [78]:
```
def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = timeseries.rolling(12).mean()
 rolstd = timeseries.rolling(12).std()
 #Plot rolling statistics:
 plt.plot(timeseries, color='blue',label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
 plt.plot(rolstd, color='black', label = 'Rolling Std')
 plt.legend(loc='best')
 plt.title('Rolling Mean and Standard Deviation')
 plt.show(block=False)

 print("Results of dickey fuller test")
 adft = adfuller(timeseries,autolag='AIC')
 # output for adft will give us without defining what the values are.
 #hence we manually write what values does 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(c['Close'])
```



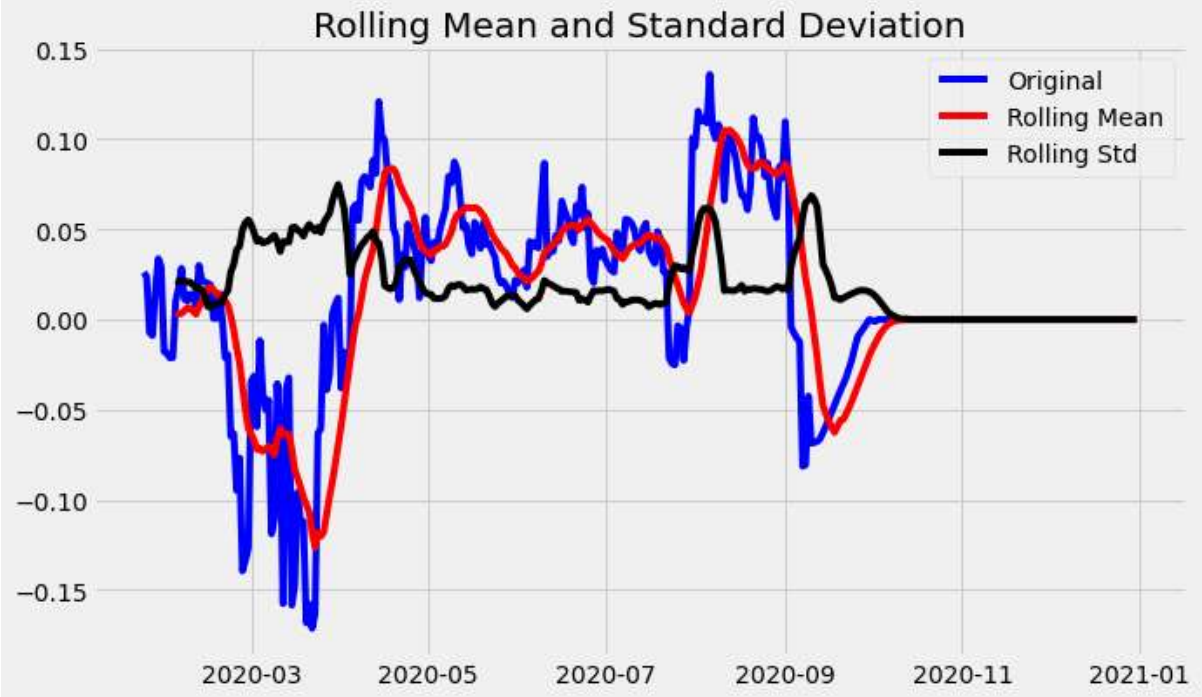

```
Results of dickey fuller test
Test Statistics                  -0.635597
p-value                           0.862734
No. of lags used                  2.000000
Number of observations used     363.000000
critical value (1%)              -3.448494
critical value (5%)              -2.869535
critical value (10%)             -2.571029
dtype: float64
```

The statistics shows that the time series is non-stationary as Test Statistic > Critical value, the p-value is greater than 5%, and we can see an increasing trend in the data. So, firstly we will try to make the data stationary. For doing so, we need to remove the trend and seasonality from the data.¶

Removing trend and seasonality to make data Stationary

In [79]:
```
train_log = np.log(c['Close'])
moving_avg = train_log.rolling(24).mean()
plt.plot(train_log)
plt.plot(moving_avg, color = 'red')
plt.show()
```

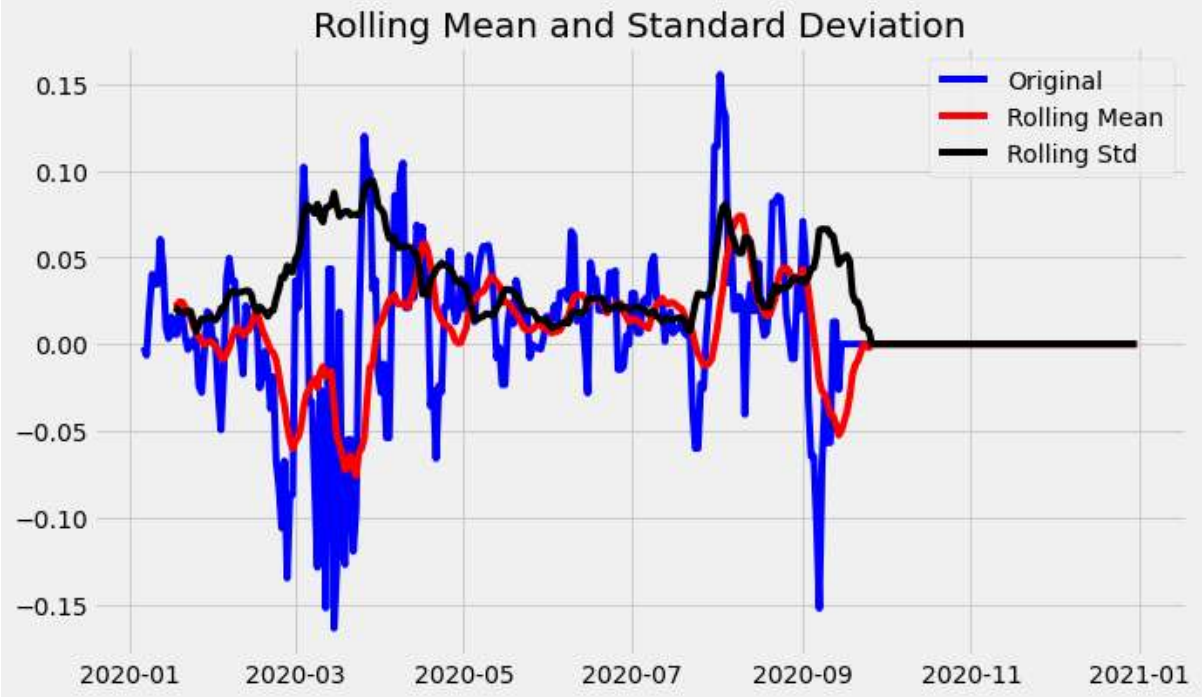
In [80]:
```
train_log_moving_avg_diff = train_log - moving_avg
train_log_moving_avg_diff.dropna(inplace = True),
test_stationarity(train_log_moving_avg_diff)
```




```
Results of dickey fuller test
Test Statistics                  -3.162073
p-value                           0.022288
No. of lags used                  4.000000
Number of observations used     338.000000
critical value (1%)              -3.449846
critical value (5%)              -2.870129
critical value (10%)             -2.571346
dtype: float64
```

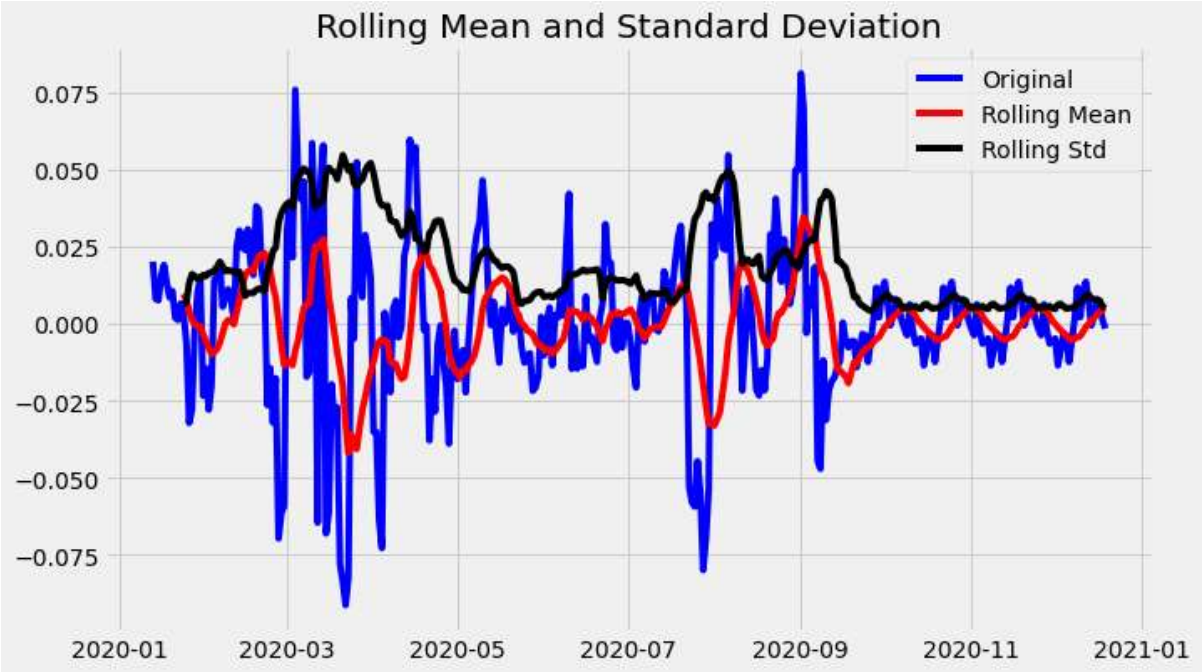
We can see that the Test Statistic is less than the Critical Value and the p-value is less than 5%. So, we can be confident that the trend is almost removed. Let's now stabilize the mean of the time series which is also a requirement for a stationary time series. Differencing can help to make the series stable and eliminate the trend.

In [81]:
```
train_log_diff = train_log - train_log.shift(5) #setting frame width delta=5 d
ays
test_stationarity(train_log_diff.dropna())
```

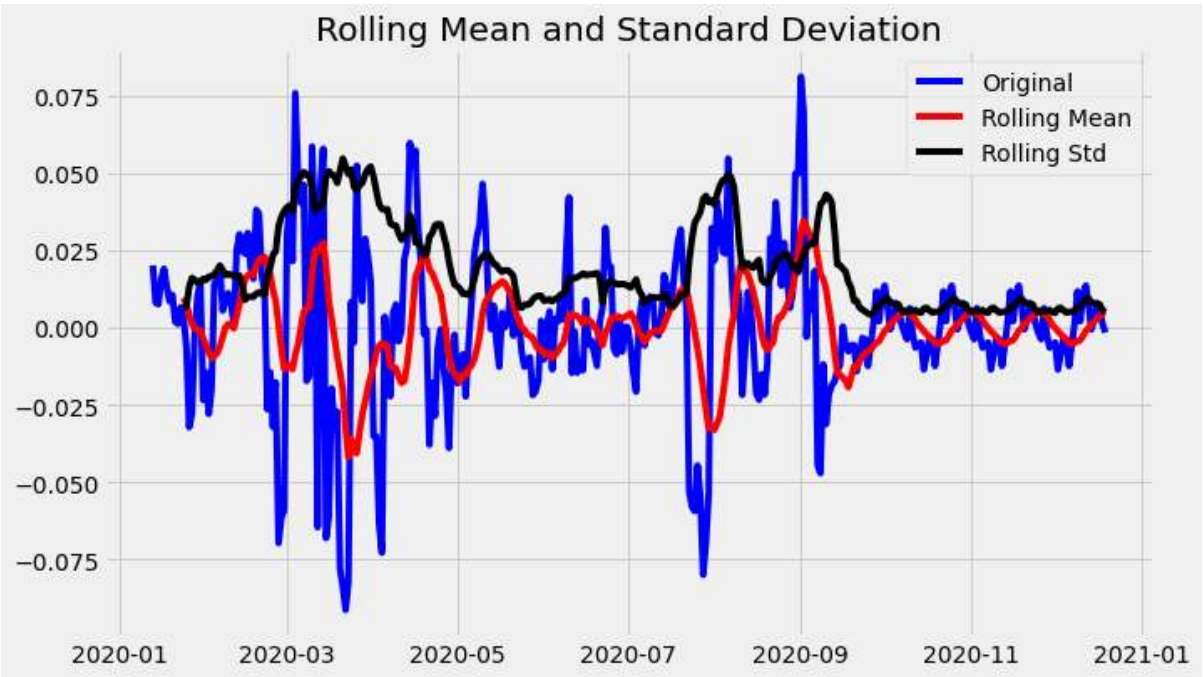



```
Results of dickey fuller test
Test Statistics                  -3.371025
p-value                           0.011987
No. of lags used                 15.000000
Number of observations used     345.000000
critical value (1%)              -3.449447
critical value (5%)              -2.869954
critical value (10%)             -2.571253
dtype: float64
```

In [82]:
```
decomposition = seasonal_decompose(pd.DataFrame(train_log).Close.values, freq
= 24)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
```



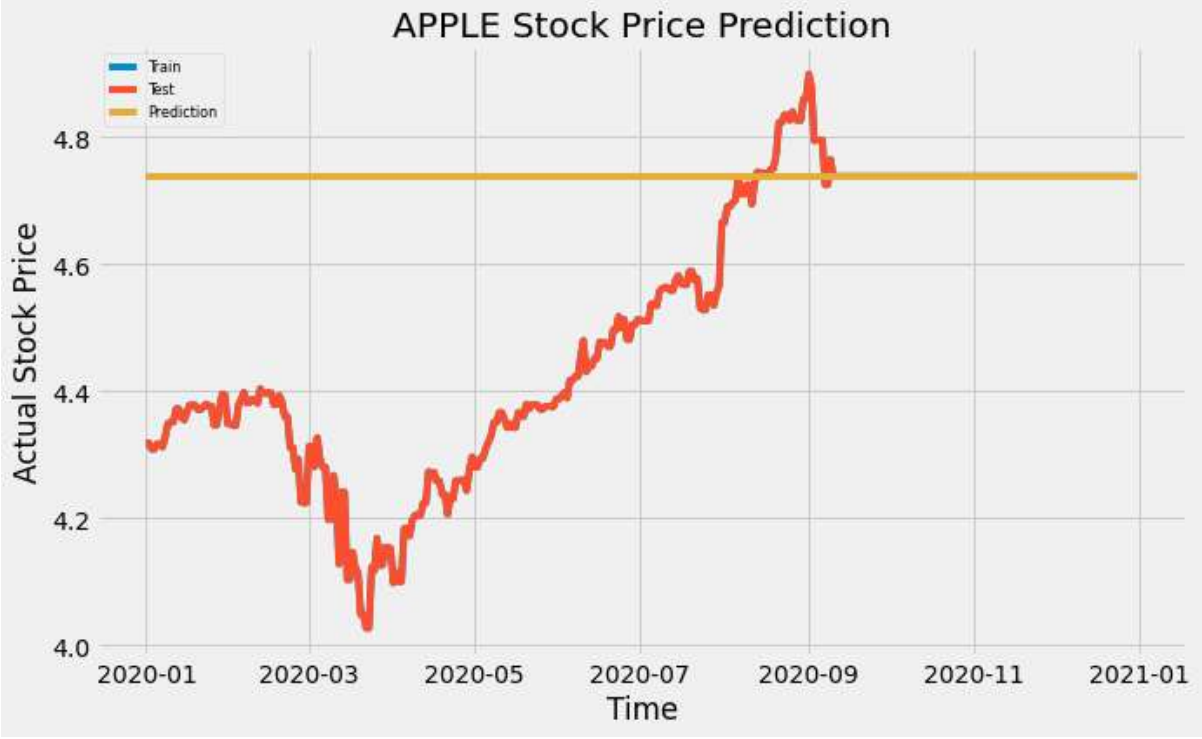

```
Results of dickey fuller test
Test Statistics                -7.957479e+00
p-value                         3.010129e-12
No. of lags used                7.000000e+00
Number of observations used     3.340000e+02
critical value (1%)            -3.450081e+00
critical value (5%)            -2.870233e+00
critical value (10%)           -2.571401e+00
dtype: float64
```



```
Results of dickey fuller test
Test Statistics                -7.957479e+00
p-value                         3.010129e-12
No. of lags used                7.000000e+00
Number of observations used     3.340000e+02
critical value (1%)            -3.450081e+00
critical value (5%)            -2.870233e+00
critical value (10%)           -2.571401e+00
dtype: float64
```

In [83]:
```
model = auto_arima(train_log, trace=True, error_action='ignore', suppress_warn
ings=True)
model.fit(train_log)
forecast = model.predict(len(test_log))
forecast = pd.DataFrame(forecast,index = test_log.index,columns=['Prediction'
])
#plot the predictions for validation set
plt.plot(train_log, label='Train')
plt.plot(test_log, label='Test')
plt.plot(forecast, label='Prediction')
plt.title('APPLE Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

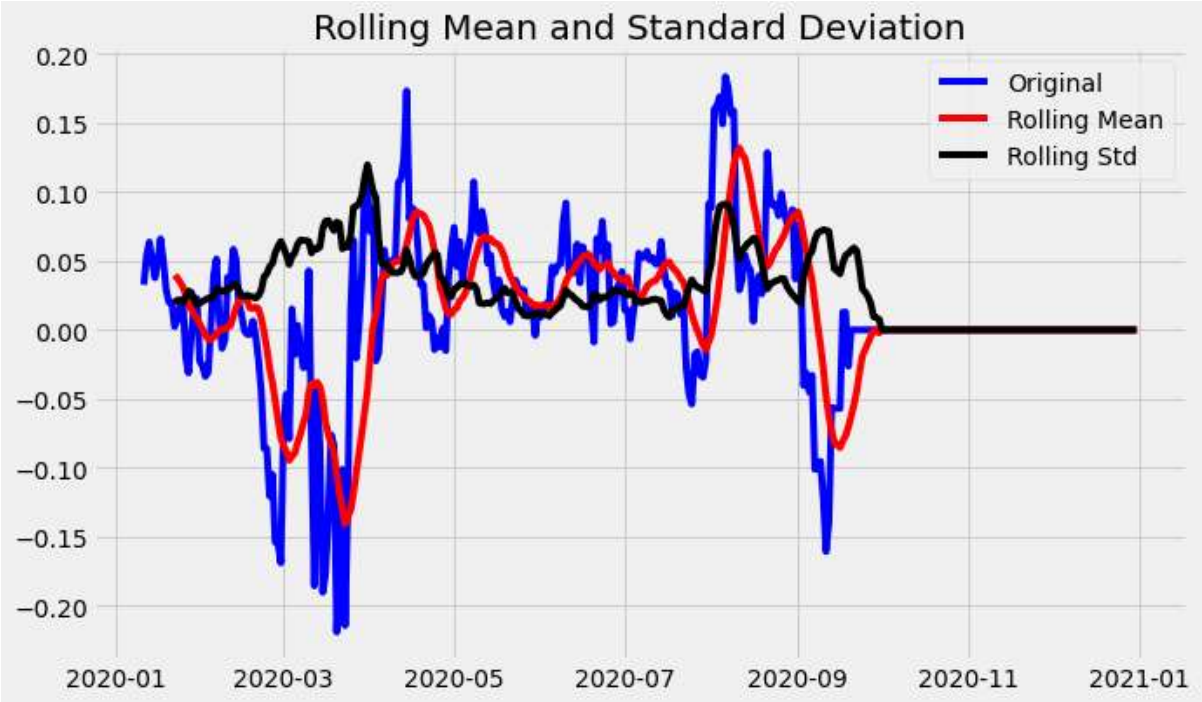
```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-1757.175, Time=0.70 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1740.181, Time=0.13 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-1743.584, Time=0.19 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-1746.406, Time=0.36 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-1741.201, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-1753.146, Time=0.97 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-1756.035, Time=0.68 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-1755.099, Time=1.18 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-1755.154, Time=0.70 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-1746.930, Time=0.63 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-1754.301, Time=0.42 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-1755.816, Time=1.14 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-1752.980, Time=1.01 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=-1757.562, Time=0.17 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-1753.381, Time=0.39 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=-1756.276, Time=0.37 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=-1755.407, Time=0.14 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=-1755.605, Time=0.32 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-1747.243, Time=0.23 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=-1754.913, Time=0.45 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=-1756.420, Time=0.57 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=-1753.607, Time=0.82 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 11.671 seconds
```



In [84]:
```
rms = sqrt(mean_squared_error(test_log,forecast))
print("RMSE: ", rms)
```

```
RMSE:  0.31176350442708917
```

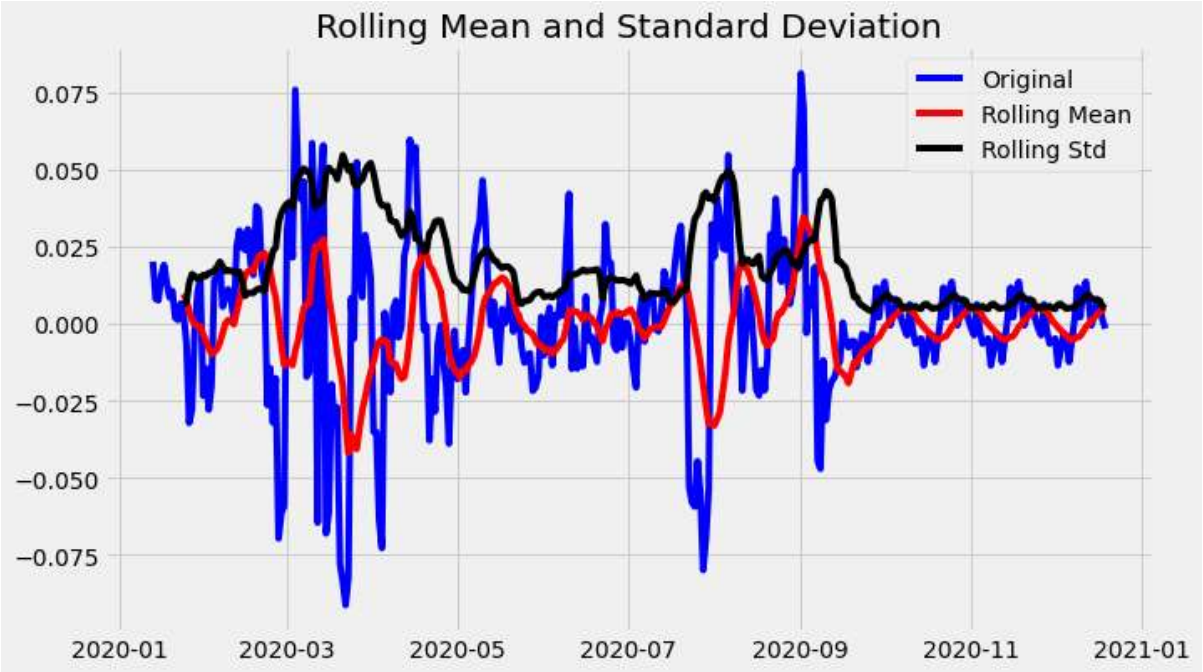
In [85]:
```
train_log_diff = train_log - train_log.shift(10) #setting frame width delta=10 days
test_stationarity(train_log_diff.dropna())
```



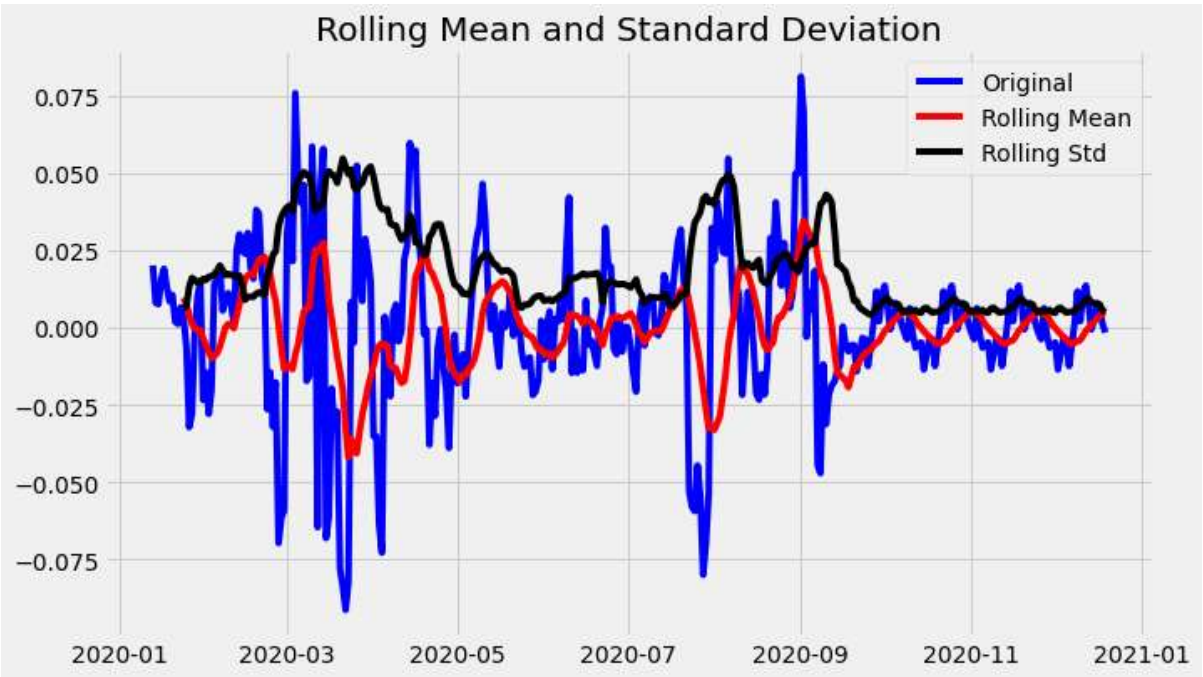

```
Results of dickey fuller test
Test Statistics                  -3.142954
p-value                           0.023533
No. of lags used                 10.000000
Number of observations used     345.000000
critical value (1%)              -3.449447
critical value (5%)              -2.869954
critical value (10%)             -2.571253
dtype: float64
```

Now we will decompose the time series into trend and seasonality and will get the residual which is the random variation in the series. Removing Seasonality By seasonality, we mean periodic fluctuations. A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, or day of the week). We will use seasonal decompose to decompose the time series into trend, seasonality and residuals. We can see the trend, residuals and the seasonality clearly in the above graph. Seasonality shows a constant trend in counter. We use the code below to check the stationarity of residuals.

In [86]:
```
decomposition = seasonal_decompose(pd.DataFrame(train_log).Close.values, freq
= 24)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
```



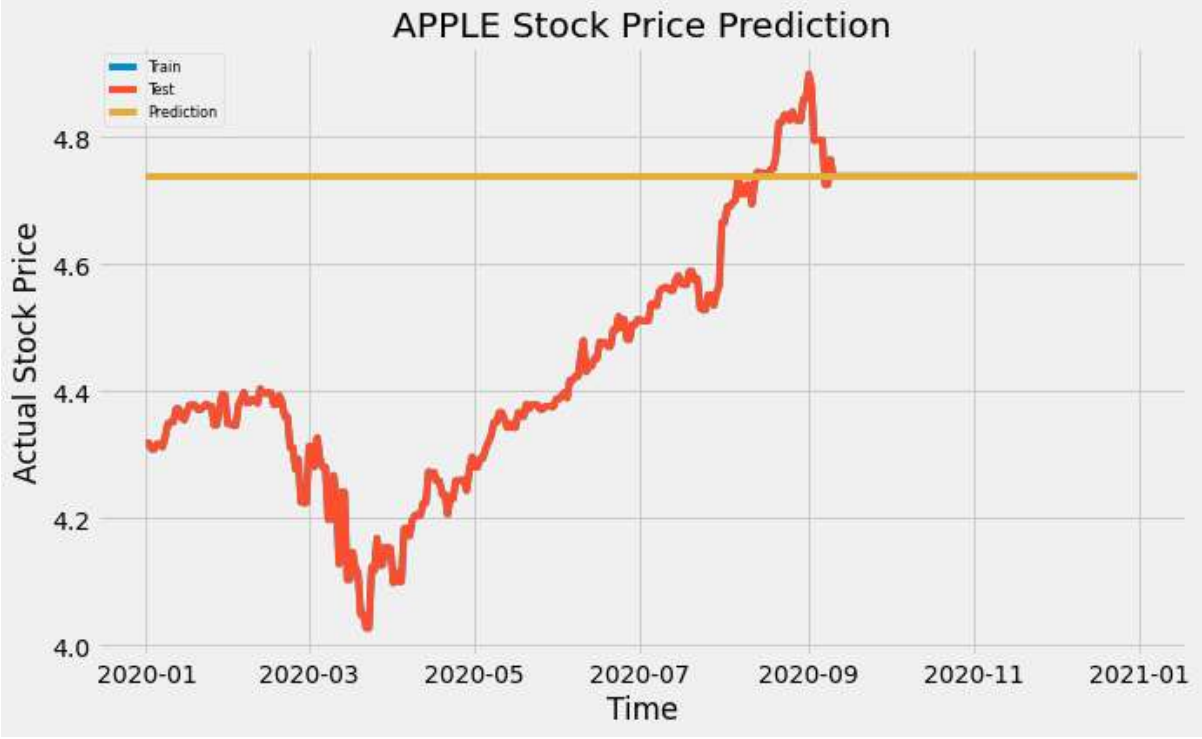

```
Results of dickey fuller test
Test Statistics                -7.957479e+00
p-value                         3.010129e-12
No. of lags used                7.000000e+00
Number of observations used     3.340000e+02
critical value (1%)            -3.450081e+00
critical value (5%)            -2.870233e+00
critical value (10%)           -2.571401e+00
dtype: float64
```



```
Results of dickey fuller test
Test Statistics                -7.957479e+00
p-value                         3.010129e-12
No. of lags used                7.000000e+00
Number of observations used     3.340000e+02
critical value (1%)            -3.450081e+00
critical value (5%)            -2.870233e+00
critical value (10%)           -2.571401e+00
dtype: float64
```

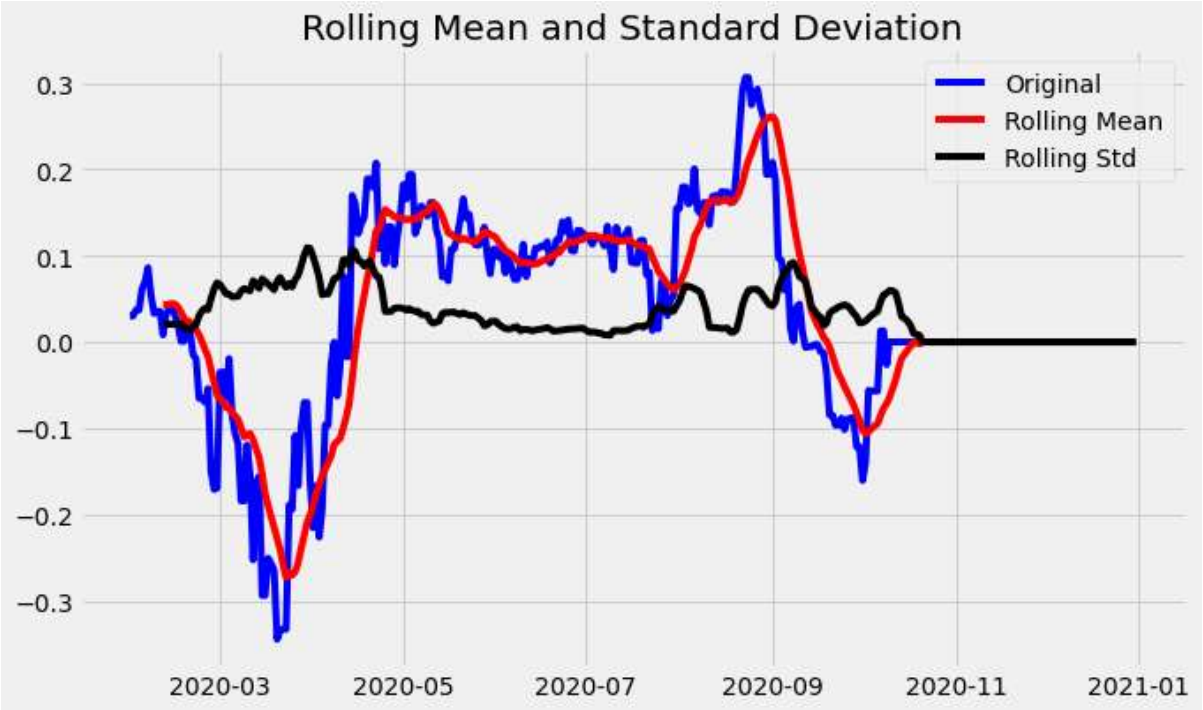
In [87]:
```
model = auto_arima(train_log, trace=True, error_action='ignore', suppress_warn
ings=True)
model.fit(train_log)
forecast = model.predict(len(test_log))
forecast = pd.DataFrame(forecast,index = test_log.index,columns=['Prediction'
])
#plot the predictions for validation set
plt.plot(train_log, label='Train')
plt.plot(test_log, label='Test')
plt.plot(forecast, label='Prediction')
plt.title('APPLE Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()

rms = sqrt(mean_squared_error(test_log,forecast))
print("RMSE: ", rms)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-1757.175, Time=0.77 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1740.181, Time=0.12 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-1743.584, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-1746.406, Time=0.38 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-1741.201, Time=0.06 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-1753.146, Time=0.77 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-1756.035, Time=0.68 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-1755.099, Time=1.18 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-1755.154, Time=0.50 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-1746.930, Time=0.53 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-1754.301, Time=0.39 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-1755.816, Time=1.28 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-1752.980, Time=0.86 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=-1757.562, Time=0.12 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-1753.381, Time=0.34 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=-1756.276, Time=0.34 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=-1755.407, Time=0.10 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=-1755.605, Time=0.27 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-1747.243, Time=0.16 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=-1754.913, Time=0.42 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=-1756.420, Time=0.63 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=-1753.607, Time=0.76 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 10.774 seconds
```

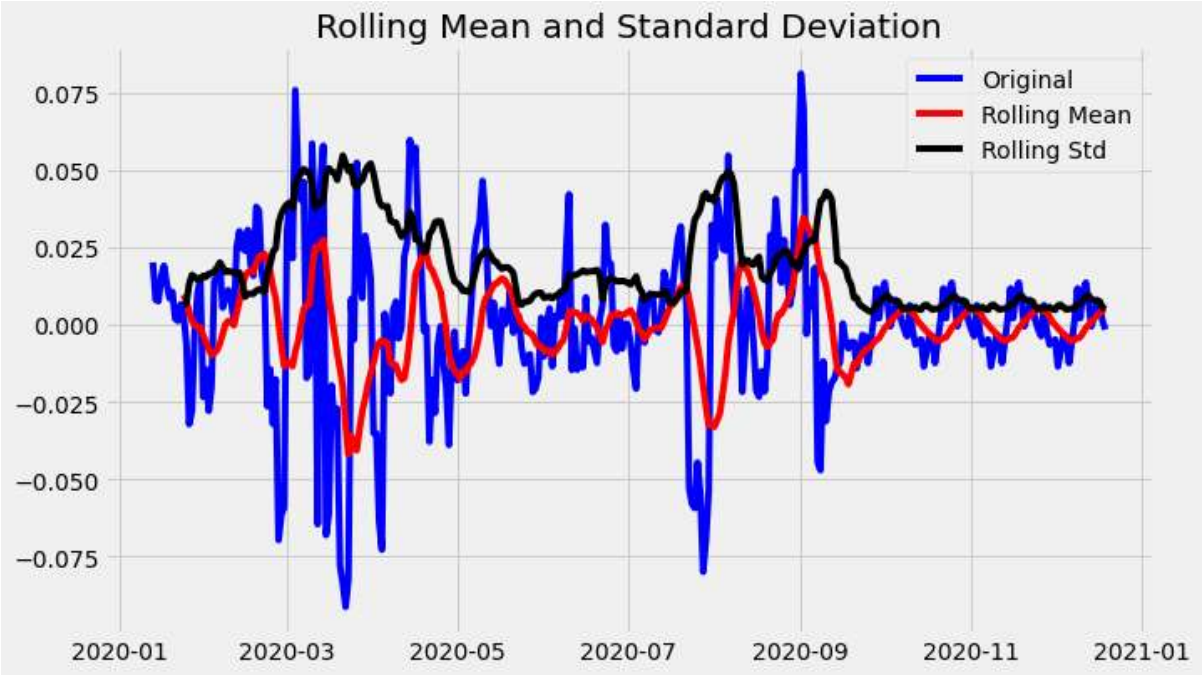



```
RMSE:  0.31176350442708917
```

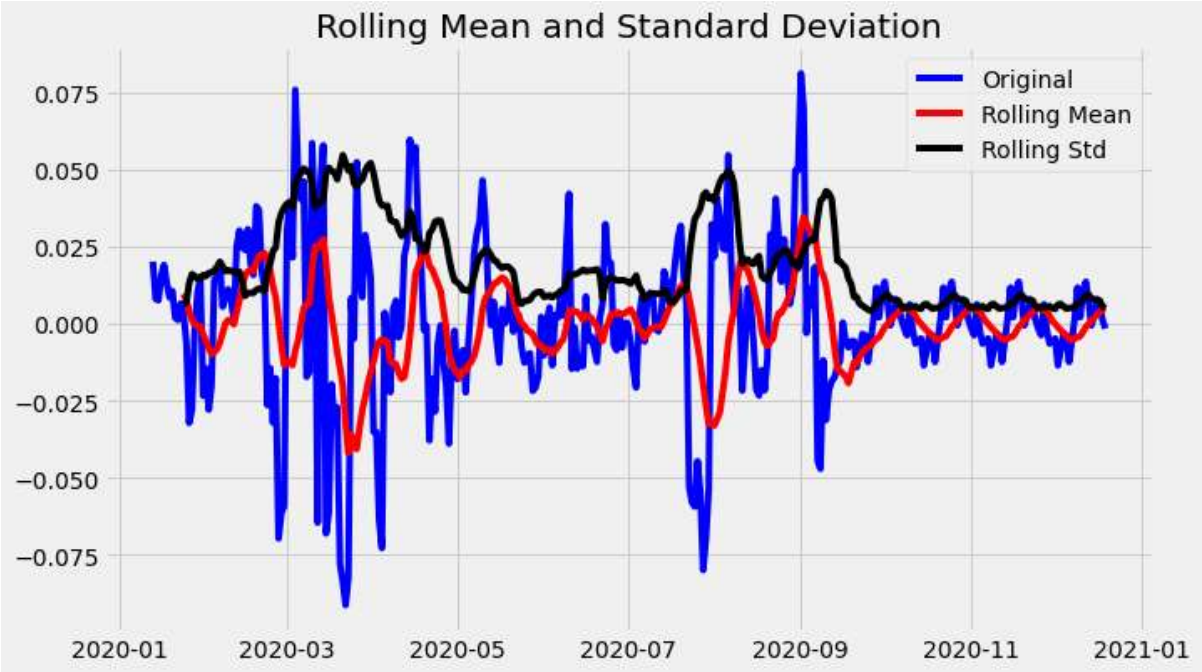
In [88]:
```
train_log_diff = train_log - train_log.shift(30)  #setting frameset delta=30 d
ays
test_stationarity(train_log_diff.dropna())

decomposition = seasonal_decompose(pd.DataFrame(train_log).Close.values, freq
= 24)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
train_log_decompose = pd.DataFrame(residual)
train_log_decompose['Date'] = train_log.index
train_log_decompose.set_index('Date', inplace = True)
train_log_decompose.dropna(inplace=True)
test_stationarity(train_log_decompose[0])
#Fit Auto ARIMA: Fit the model on the univariate series. Predict values on val
idation set: Make predictions on the test set.
#Calculate RMSE: Check the performance of the model using the predicted values
against the actual values.
model = auto_arima(train_log, trace=True, error_action='ignore', suppress_warn
ings=True)
model.fit(train_log)
forecast = model.predict(len(test_log))
forecast = pd.DataFrame(forecast,index = test_log.index,columns=['Prediction'
])
#plot the predictions for validation set
plt.plot(train_log, label='Train')
plt.plot(test_log, label='Test')
plt.plot(forecast, label='Prediction')
plt.title('APPLE Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()

rms = sqrt(mean_squared_error(test_log,forecast))
print("RMSE: ", rms)
```




```
Results of dickey fuller test
Test Statistics                  -2.177834
p-value                           0.214388
No. of lags used                  3.000000
Number of observations used     332.000000
critical value (1%)              -3.450201
critical value (5%)              -2.870285
critical value (10%)             -2.571429
dtype: float64
```



```
Results of dickey fuller test
Test Statistics                -7.957479e+00
p-value                         3.010129e-12
No. of lags used                7.000000e+00
Number of observations used     3.340000e+02
critical value (1%)            -3.450081e+00
critical value (5%)            -2.870233e+00
critical value (10%)           -2.571401e+00
dtype: float64
```



```
Results of dickey fuller test
Test Statistics                -7.957479e+00
p-value                         3.010129e-12
No. of lags used                7.000000e+00
Number of observations used     3.340000e+02
critical value (1%)            -3.450081e+00
critical value (5%)            -2.870233e+00
critical value (10%)           -2.571401e+00
dtype: float64
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-1757.175, Time=0.60 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1740.181, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-1743.584, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-1746.406, Time=0.45 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-1741.201, Time=0.09 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-1753.146, Time=0.71 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-1756.035, Time=0.58 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-1755.099, Time=1.02 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-1755.154, Time=0.51 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-1746.930, Time=0.48 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-1754.301, Time=0.34 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=-1755.816, Time=1.02 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-1752.980, Time=0.81 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=-1757.562, Time=0.13 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-1753.381, Time=0.31 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=-1756.276, Time=0.29 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=-1755.407, Time=0.14 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=-1755.605, Time=0.24 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-1747.243, Time=0.14 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=-1754.913, Time=0.16 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=-1756.420, Time=0.47 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=-1753.607, Time=0.85 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 9.509 seconds
```


```
RMSE:  0.31176350442708917
```

A RMSE of 0 indicates a perfect fit with no errors. The smaller the RMSE score the better the model. A RMSE score of 0.31176350442708917 indicates a good model and we have plotted it as well.

In [89]:
```
model=sm.tsa.statespace.SARIMAX(c['Close'],order=(1, 1, 1),seasonal_order=(1,1
,1,12))
results=model.fit()

c['forecast']=results.predict(start=90,end=200,dynamic=True)
c[['Close','forecast']].plot(figsize=(12,8))
```

Out[89]: <AxesSubplot:>

For the mean square error we have good model with RMSE score just above 0.31 and when we are trying to predict/forecast value with SARIMAX it predicts our overall trend but was not able to adjust for the seasonality correctly for 2020's year dataset. We can measure this with other models like Prophet , LSTM and also this is corely dependent on the dataset as well