Project 05: Time Series Models

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.tools.plotting import autocorrelation_plot
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima_model import ARIMA, ARMAResults
import datetime
import sys
import seaborn as sns
import statsmodels
import statsmodels.stats.diagnostic as diag
from statsmodels.tsa.stattools import adfuller
from scipy.stats.mstats import normaltest
from matplotlib.pyplot import acorr
#plt.style.use('fivethirtyeight')
import warnings
warnings.warn('ignore')
%matplotlib inline
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWa
rning: ignore
```

In [2]:

```
df = pd.read_csv('data_stocks.csv')
df.head()
```

Out[2]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDA
0	1491226200	2363.6101	42.3300	143.6800	129.6300	82.040	1(
1	1491226260	2364.1001	42.3600	143.7000	130.3200	82.080	1(
2	1491226320	2362.6799	42.3100	143.6901	130.2250	82.030	1(
3	1491226380	2364.3101	42.3700	143.6400	130.0729	82.000	1(
4	1491226440	2364.8501	42.5378	143.6600	129.8800	82.035	1(

5 rows × 502 columns

Pick up the following stocks and generate forecasts accordingly

In [3]:

```
stock_features =['NASDAQ.AAPL','NASDAQ.ADP','NASDAQ.CBOE','NASDAQ.CSCO','NASDAQ.EBAY']
col_list = ['DATE'] + stock_features
df1 = df[col_list]
df1.head()
```

Out[3]:

	DATE	NASDAQ.AAPL	NASDAQ.ADP	NASDAQ.CBOE	NASDAQ.CSCO	NASDAQ.EBAY
0	1491226200	143.6800	102.2300	81.03	33.7400	33.3975
1	1491226260	143.7000	102.1400	81.21	33.8800	33.3950
2	1491226320	143.6901	102.2125	81.21	33.9000	33.4100
3	1491226380	143.6400	102.1400	81.13	33.8499	33.3350
4	1491226440	143.6600	102.0600	81.12	33.8400	33.4000

In [4]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41266 entries, 0 to 41265
Data columns (total 6 columns):
DATE           41266 non-null int64
NASDAQ.AAPL    41266 non-null float64
NASDAQ.ADP     41266 non-null float64
NASDAQ.CBOE    41266 non-null float64
NASDAQ.CSCO    41266 non-null float64
NASDAQ.EBAY    41266 non-null float64
dtypes: float64(5), int64(1)
memory usage: 1.9 MB
```

In [5]:

```
df1.isnull().sum()
```

Out[5]:

```
DATE           0
NASDAQ.AAPL    0
NASDAQ.ADP     0
NASDAQ.CBOE    0
NASDAQ.CSCO    0
NASDAQ.EBAY    0
dtype: int64
```

In [6]:

```
df1 =df1.copy()
df1['DATE'] =  pd.to_datetime(df1['DATE'])
```

In [7]:

```
df1.tail()
```

Out[7]:

	DATE	NASDAQ.AAPL	NASDAQ.ADP	NASDAQ.CBOE	NASDAQ.CSCO	NAS
41261	1970-01-01 00:00:01.504209360	164.11	106.565	100.89	32.185	
41262	1970-01-01 00:00:01.504209420	164.12	106.590	100.88	32.200	
41263	1970-01-01 00:00:01.504209480	164.01	106.520	100.86	32.200	
41264	1970-01-01 00:00:01.504209540	163.88	106.400	100.83	32.195	
41265	1970-01-01 00:00:01.504209600	163.98	106.470	100.89	32.225	

In [8]:

```
df1.head()
```

Out[8]:

	DATE	NASDAQ.AAPL	NASDAQ.ADP	NASDAQ.CBOE	NASDAQ.CSCO	NASDAC
0	1970-01-01 00:00:01.491226200	143.6800	102.2300	81.03	33.7400	3
1	1970-01-01 00:00:01.491226260	143.7000	102.1400	81.21	33.8800	3
2	1970-01-01 00:00:01.491226320	143.6901	102.2125	81.21	33.9000	3
3	1970-01-01 00:00:01.491226380	143.6400	102.1400	81.13	33.8499	3
4	1970-01-01 00:00:01.491226440	143.6600	102.0600	81.12	33.8400	3

In [9]:

```
df1 = df1.copy()
df1['Month'] = df1['DATE'].dt.date
```

In [10]:

```
df1.head()
```

Out[10]:

	DATE	NASDAQ.AAPL	NASDAQ.ADP	NASDAQ.CBOE	NASDAQ.CSCO	NASDAC
0	1970-01-01 00:00:01.491226200	143.6800	102.2300	81.03	33.7400	3
1	1970-01-01 00:00:01.491226260	143.7000	102.1400	81.21	33.8800	3
2	1970-01-01 00:00:01.491226320	143.6901	102.2125	81.21	33.9000	3
3	1970-01-01 00:00:01.491226380	143.6400	102.1400	81.13	33.8499	3
4	1970-01-01 00:00:01.491226440	143.6600	102.0600	81.12	33.8400	3

In [11]:

```
col_list = ['Month']+ stock_features
df2 = df1[col_list]
df2.head()
```

Out[11]:

	Month	NASDAQ.AAPL	NASDAQ.ADP	NASDAQ.CBOE	NASDAQ.CSCO	NASDAQ.EBAY
0	1970-01-01	143.6800	102.2300	81.03	33.7400	33.3975
1	1970-01-01	143.7000	102.1400	81.21	33.8800	33.3950
2	1970-01-01	143.6901	102.2125	81.21	33.9000	33.4100
3	1970-01-01	143.6400	102.1400	81.13	33.8499	33.3350
4	1970-01-01	143.6600	102.0600	81.12	33.8400	33.4000

In [12]:

```
df2.isnull().sum()
```

Out[12]:

```
Month          0
NASDAQ.AAPL    0
NASDAQ.ADP     0
NASDAQ.CBOE    0
NASDAQ.CSCO    0
NASDAQ.EBAY    0
dtype: int64
```

In [13]:

```
df2.describe().transpose()
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
NASDAQ.AAPL	41266.0	150.453566	6.236826	140.160	144.640	149.9450	155.065	164.51
NASDAQ.ADP	41266.0	103.480398	4.424244	95.870	101.300	102.4400	104.660	121.77
NASDAQ.CBOE	41266.0	89.325485	5.746178	80.000	84.140	89.3150	93.850	101.35
NASDAQ.CSCO	41266.0	32.139336	0.985571	30.365	31.455	31.7733	32.790	34.49
NASDAQ.EBAY	41266.0	34.794506	1.099296	31.890	34.065	34.7700	35.610	37.46

In [14]:

```
final = df2.copy()
final['Month']=pd.to_datetime(final['Month'])
```

Time Series Forecasting for NASDAQ.AAPL

In [15]:

```
df_AAPL = final[['Month',stock_features[0]]]
```

In [16]:

```
df_AAPL.head()
```

Out[16]:

	Month	NASDAQ.AAPL
0	1970-01-01	143.6800
1	1970-01-01	143.7000
2	1970-01-01	143.6901
3	1970-01-01	143.6400
4	1970-01-01	143.6600

In [17]:

```
df_AAPL.set_index('Month',inplace=True)
df_AAPL.head()
```

Out[17]:

	NASDAQ.AAPL
Month	
1970-01-01	143.6800
1970-01-01	143.7000
1970-01-01	143.6901
1970-01-01	143.6400
1970-01-01	143.6600

In [18]:

```
df_AAPL.index
```

Out[18]:

```
DatetimeIndex(['1970-01-01', '1970-01-01', '1970-01-01', '1970-01-01',
               '1970-01-01', '1970-01-01', '1970-01-01', '1970-01-01',
               '1970-01-01', '1970-01-01',
               ...
               '1970-01-01', '1970-01-01', '1970-01-01', '1970-01-01',
               '1970-01-01', '1970-01-01', '1970-01-01', '1970-01-01',
               '1970-01-01', '1970-01-01'],
              dtype='datetime64[ns]', name='Month', length=41266, freq=None)
```

Summary Statistics

In [19]:

```
df_AAPL.describe().transpose()
```

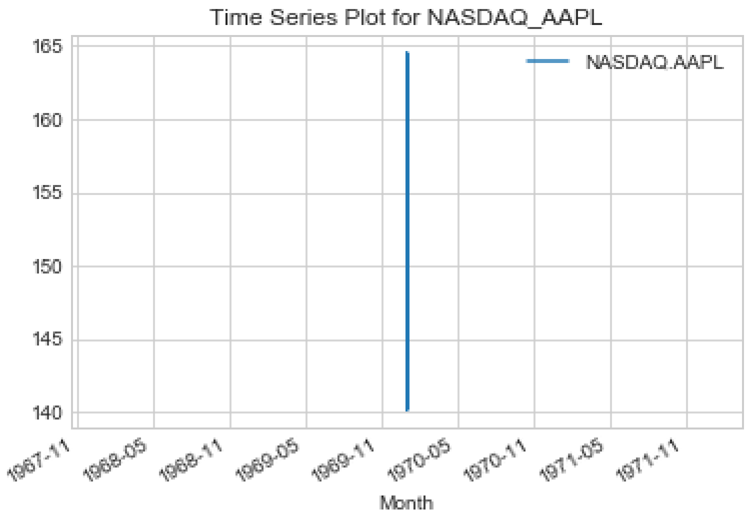
Out[19]:

	count	mean	std	min	25%	50%	75%	max
NASDAQ.AAPL	41266.0	150.453566	6.236826	140.16	144.64	149.945	155.065	164.51

Step 2 : Visualize the Data

In [20]:

```
import seaborn as sns
sns.set_style('whitegrid')
df_AAPL.plot()
plt.title('Time Series Plot for NASDAQ_AAPL')
plt.show()
```



Plotting Rolling Statistics and check for stationarity :

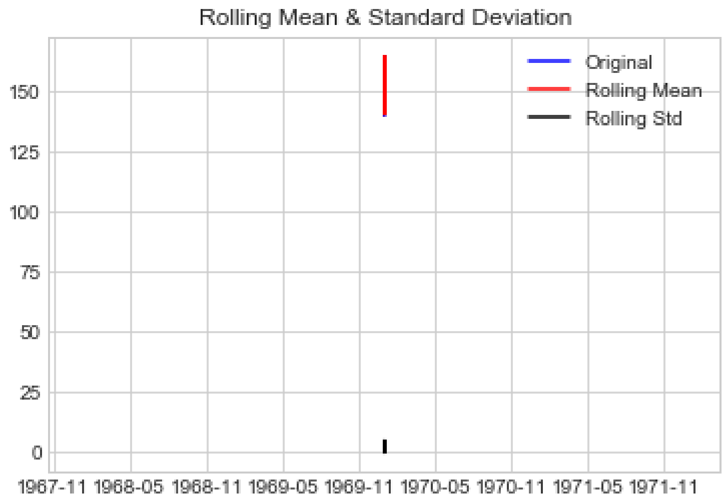
The function will plot the moving mean or moving Standard Deviation. This is still visual method

In [21]:

```
from statsmodels.tsa.stattools import adfuller
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 & Standard Deviation')
    plt.show()
    """
    Pass in a time series, returns ADF report
    """
    result = adfuller(timeseries)
    print('\nAugmented Dickey-Fuller Test:')
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']

    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    for k,v in result[4].items():
        print('Crtical {} : value {}'.format(k,v))

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis, reject the null hypothesis. Dat
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicati
```

In [22]:

```
test_stationarity(df_AAPL['NASDAQ.AAPL'])
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -0.9128532997926677
p-value : 0.7837101772613864
#Lags Used : 31
Number of Observations Used : 41234
Crtical 1% : value -3.4305085998723857
Crtical 5% : value -2.8616100975579815
Crtical 10% : value -2.5668073106689477
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

Note:

This is not stationary because : mean is increasing even though the std is small, Test stat is > critical value. • Note: the signed values are compared and the absolute values.

MAKING THE TIME SERIES STATIONARY

There are two major factors that make a time series non-stationary. They are:

• Trend: non-constant mean

• Seasonality: Variation at specific time-frames

Differencing

The first difference of a time series is the series of changes from one period to the next. We can do this easily with pandas. You can continue to take the second difference, third difference, and so on until your data is stationary.

First Difference

In [23]:

```
df_AAPL = df_AAPL.copy()
df_AAPL.loc[:,'First_Difference'] = df_AAPL['NASDAQ.AAPL'] - df_AAPL['NASDAQ.AAPL'].shift(1
```

In [24]:

```
df_AAPL.head()
```

Out[24]:

	NASDAQ.AAPL	First_Difference
Month		
1970-01-01	143.6800	NaN
1970-01-01	143.7000	0.0200
1970-01-01	143.6901	-0.0099
1970-01-01	143.6400	-0.0501
1970-01-01	143.6600	0.0200

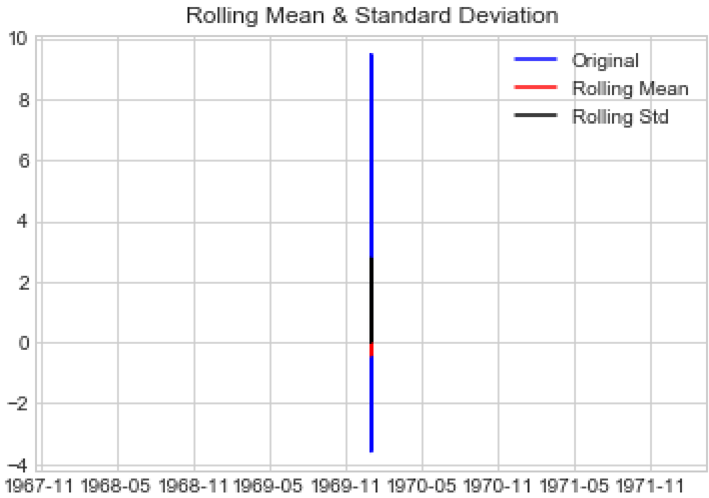
In [25]:

```
df_AAPL = df_AAPL.copy()
df_AAPL.dropna(inplace=True)
```

Test Staionarity

In [26]:

```
test_stationarity(df_AAPL['First_Difference'])
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -35.737741483401265
p-value : 0.0
#Lags Used : 30
Number of Observations Used : 41234
Crtical 1% : value -3.4305085998723857
Crtical 5% : value -2.8616100975579815
Crtical 10% : value -2.5668073106689477
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

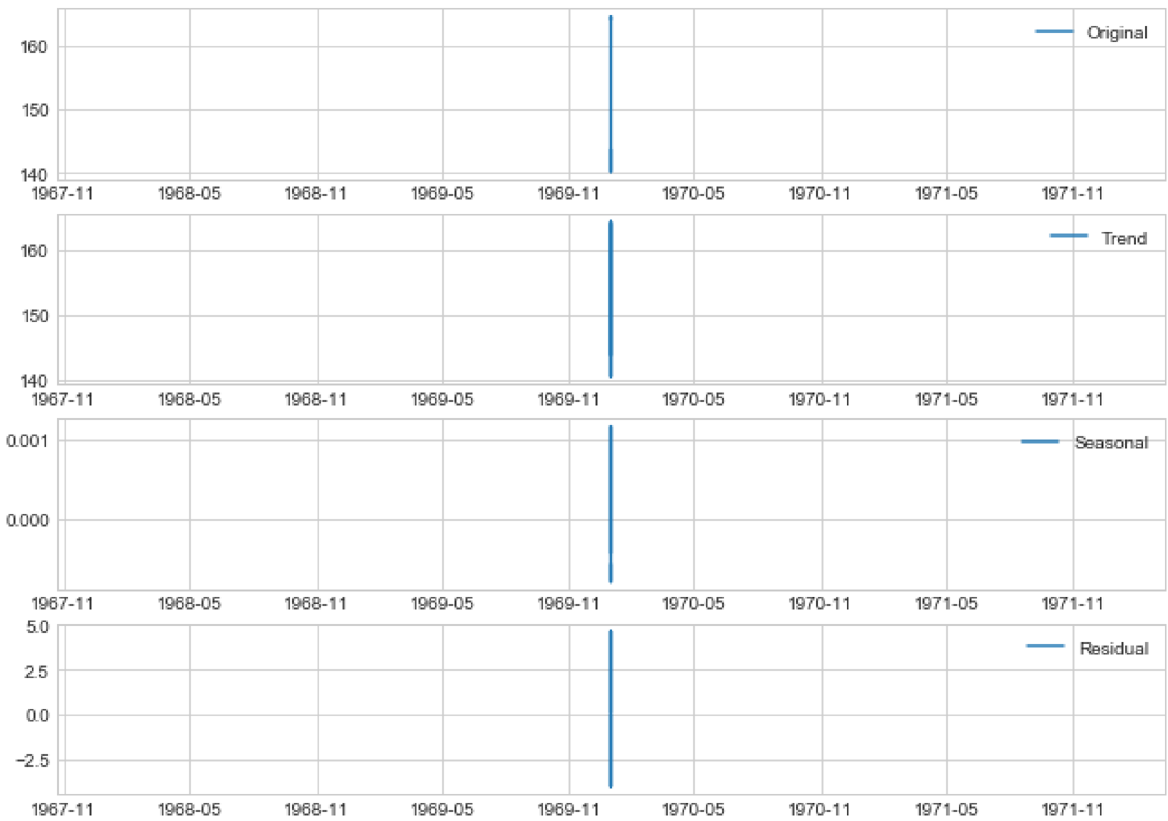
Seasonal Decomposition

In [27]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
plt.figure(figsize=(11,8))
decomposition = seasonal_decompose(df_AAPL['NASDAQ.AAPL'],freq=12)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.subplot(411)
plt.plot(df_AAPL['NASDAQ.AAPL'],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='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual,label='Residual')
plt.legend(loc='best')
```

Out[27]:

```
<matplotlib.legend.Legend at 0xc8d14a8>
```

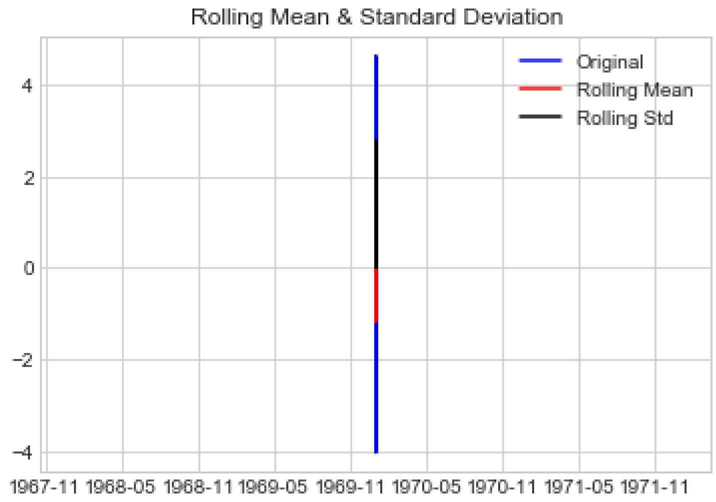


Note:

The data is seasonal as interpreted from the Seasonal plot of seasonal decomposition.

In [28]:

```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -43.04343353554248
p-value : 0.0
#Lags Used : 55
Number of Observations Used : 41197
Crtical 1% : value -3.4305087423235587
Crtical 5% : value -2.861610160516496
Crtical 10% : value -2.566807344180027
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

Note :

This is stationary because:

• test statistic is lower than critical values.

• the mean and std variations have small variations with time.

Autocorrelation and Partial Autocorrelation Plots

Autocorrelation Interpretation

The actual interpretation and how it relates to ARIMA models can get a bit complicated, but there are some basic common methods we can use for the ARIMA model. Our main priority here is to try to figure out whether we will use the AR or MA components for the ARIMA model (or both!) as well as how many lags we should use. In general you would use either AR or MA, using both is less common.

- If the autocorrelation plot shows positive autocorrelation at the first lag (lag-1), then it suggests to use the AR terms in relation to the lag
- If the autocorrelation plot shows negative autocorrelation at the first lag, then it suggests using MA terms

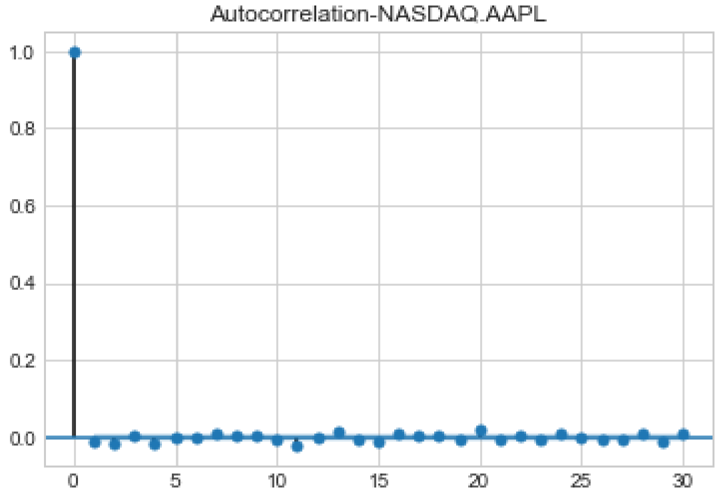
In [29]:

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [30]:

```
plt.figure(figsize=(20,8))
fig_first = plot_acf(df_AAPL["First_Difference"],lags=30,title='Autocorrelation-NASDAQ.AAPL
```

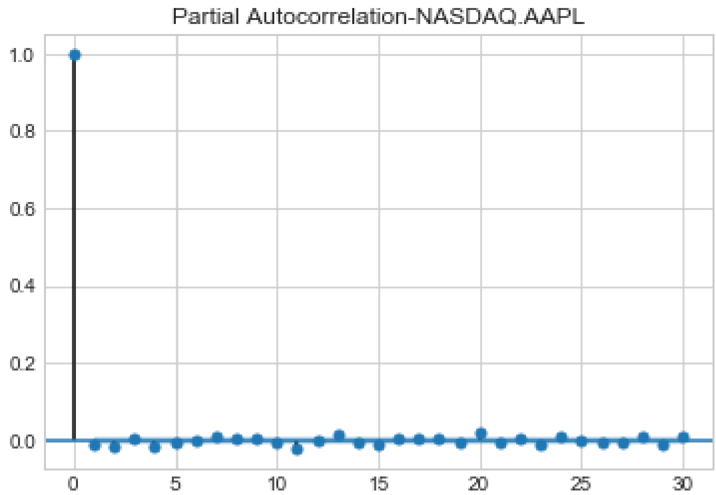
<Figure size 1440x576 with 0 Axes>



In [31]:

```
plt.figure(figsize=(20,8))
fig_pacf_first = plot_pacf(df_AAPL["First_Difference"],lags=30,title='Partial Autocorrelati
```

<Figure size 1440x576 with 0 Axes>

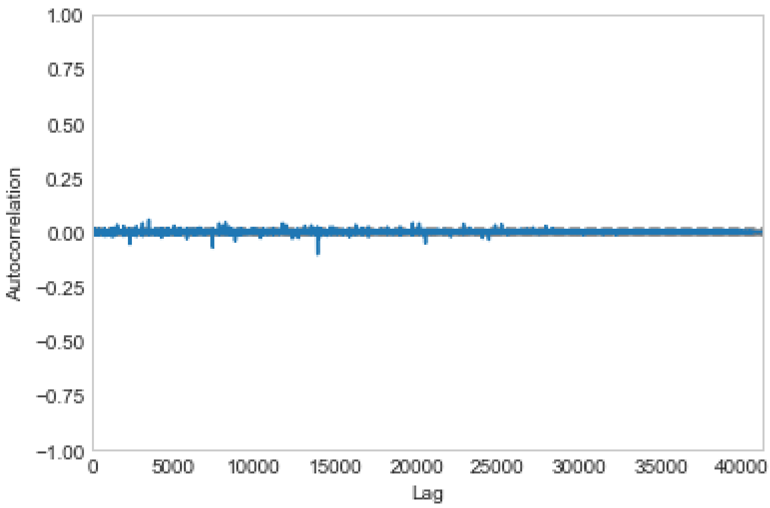


In [32]:

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df_AAPL['First_Difference'])
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xcf0df28>
```



Forecasting a Time Series

Auto Regressive Integrated Moving Average(ARIMA)—

It is like a liner regression equation where the predictors depend on parameters (p,d,q) of the ARIMA model .

Let me explain these dependent parameters:

• p : This is the number of AR (Auto-Regressive) terms . Example—if p is 3 the predictor for y(t) will be y(t-1),y(t-2),y(t-3).

• q : This is the number of MA (Moving-Average) terms . Example—if p is 3 the predictor for y(t) will be y(t-1),y(t-2),y(t-3).

• d :This is the number of differences or the number of non-seasonal differences .

Now let’s check out on how we can figure out what value of p and q to use. We use two popular plotting techniques; they are:

• Autocorrelation Function (ACF): It just measures the correlation between two consecutive (lagged version). example at lag 4, ACF will compare series at time instance t1…t2 with series at instance t1–4…t2–4

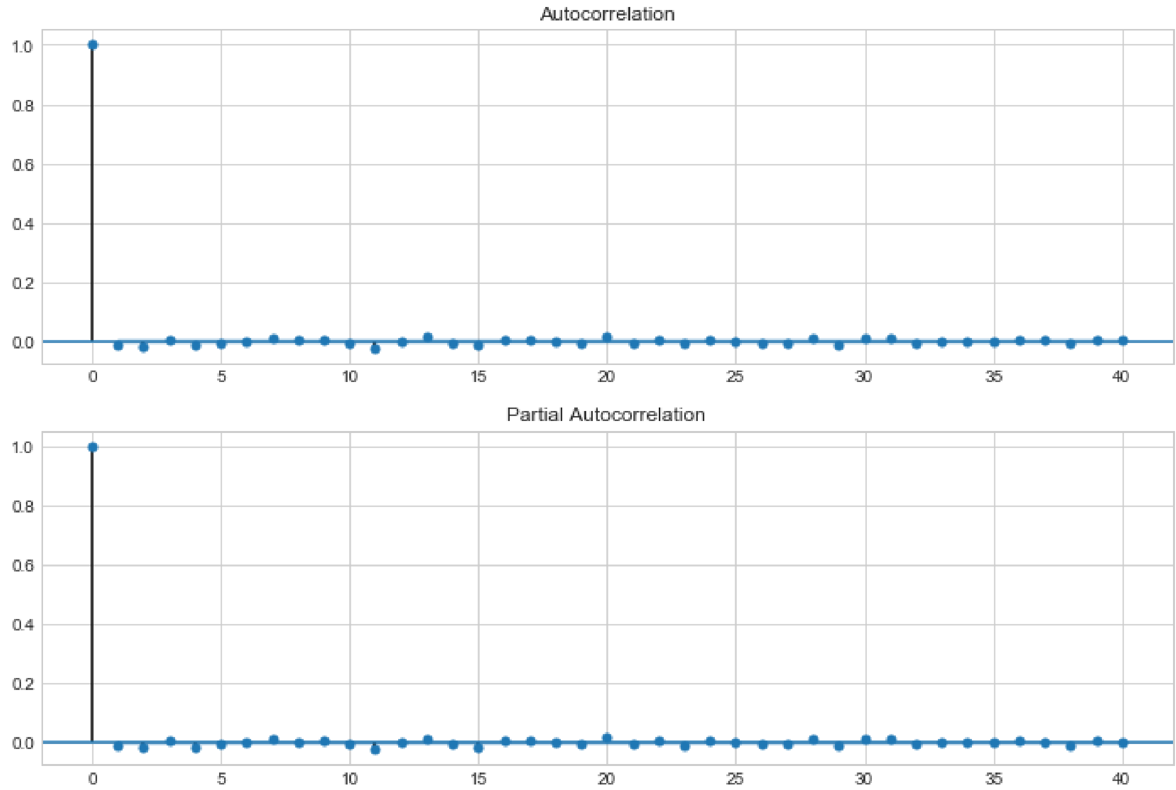
• Partial Autocorrelation Function (PACF): is used to measure the degree of association between y(t) and y(t-p).

In [33]:

```
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA, ARIMAResults
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [34]:

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_AAPL['First_Difference'].iloc[30:], lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_AAPL['First_Difference'].iloc[30:], lags=40, ax=ax2)
```

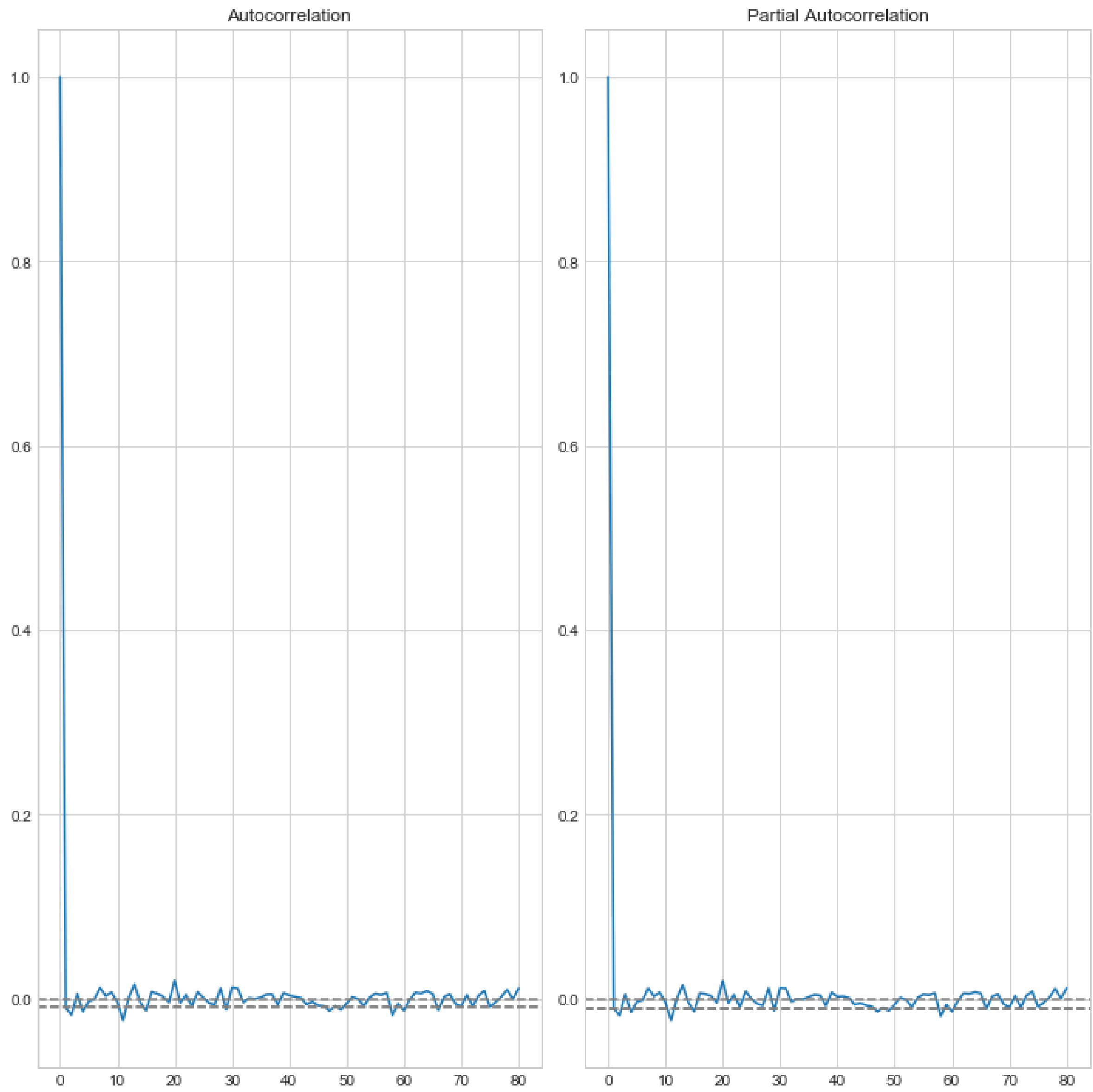


In [35]:

```
lag_acf = acf(df_AAPL['First_Difference'],nlags=80)
lag_pacf = pacf(df_AAPL['First_Difference'],nlags=80,method='ols')
```

In [36]:

```
plt.figure(figsize=(10,10))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_AAPL['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_AAPL['First_Difference'])),linestyle='--',color='gray')

plt.title('Autocorrelation')

plt.subplot(122)

plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_AAPL['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_AAPL['First_Difference'])),linestyle='--',color='gray')

plt.title('Partial Autocorrelation')

plt.tight_layout()
```



Note

The two dotted lines on either sides of 0 are the confidence intervals.

These can be used to determine the ‘p’ and ‘q’ values as:

• p: The first time where the PACF crosses the upper confidence interval, here its close to 0. hence p = 0.

• q: The first time where the ACF crosses the upper confidence interval, here its close to 0. hence p = 0.

Using the Seasonal ARIMA model

In [37]:

```
model= sm.tsa.statespace.SARIMAX(df_AAPL['NASDAQ.AAPL'],order=(0,1,0),seasonal_order=(0,1,0
results = model.fit()
print(results.summary())
```

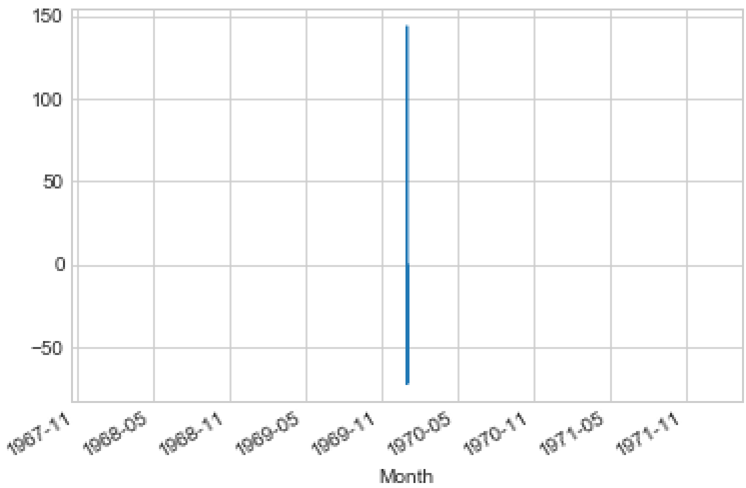
```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:225: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
                                 Statespace Model Results

============================================================================
==============
Dep. Variable:                     NASDAQ.AAPL   No. Observations:
         41265
Model:             SARIMAX(0, 1, 0)x(0, 1, 0, 12)   Log Likelihood
     24925.552
Date:                         Tue, 25 Dec 2018   AIC
    -49849.104
Time:                                 20:55:27   BIC
    -49840.477
Sample:                                      0   HQIC
    -49846.377
                                       - 41265

Covariance Type:                           opg

============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
sigma2         0.0175   4.57e-06   3828.710      0.000       0.017       0.0
17
============================================================================
=======
Ljung-Box (Q):                  10611.64   Jarque-Bera (JB):        346226
2306.74
Prob(Q):                            0.00   Prob(JB):
   0.00
Heteroskedasticity (H):             2.92   Skew:
  -2.00
Prob(H) (two-sided):                0.00   Kurtosis:
1422.26
============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```

In [38]:

```
results.resid.plot()
```

Out[38]:

<matplotlib.axes._subplots.AxesSubplot at 0xd3a7198>

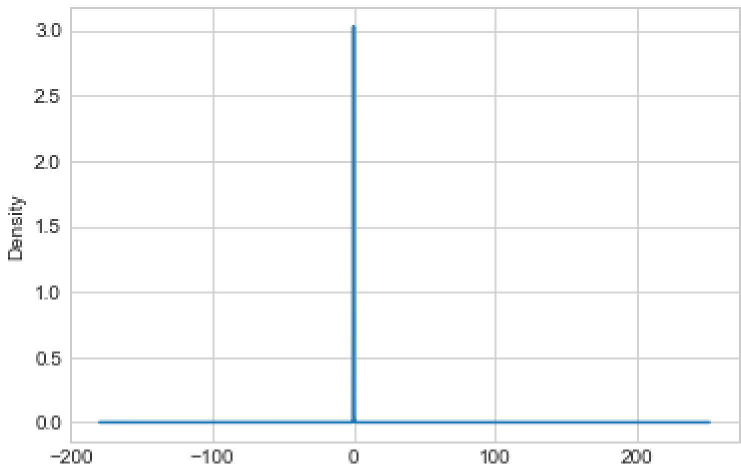


In [39]:

```
results.resid.plot(kind='kde')
```

Out[39]:

<matplotlib.axes._subplots.AxesSubplot at 0xd355ac8>



In [40]:

```
df_AAPL = df_AAPL.copy()
df_AAPL['Forecast'] = results.predict()
```

In [41]:

```
df_AAPL.head()
```

Out[41]:

	NASDAQ.AAPL	First_Difference	Forecast
Month			
1970-01-01	143.7000	0.0200	0.0000
1970-01-01	143.6901	-0.0099	143.7000
1970-01-01	143.6400	-0.0501	143.6901
1970-01-01	143.6600	0.0200	143.6400
1970-01-01	143.7800	0.1200	143.6600

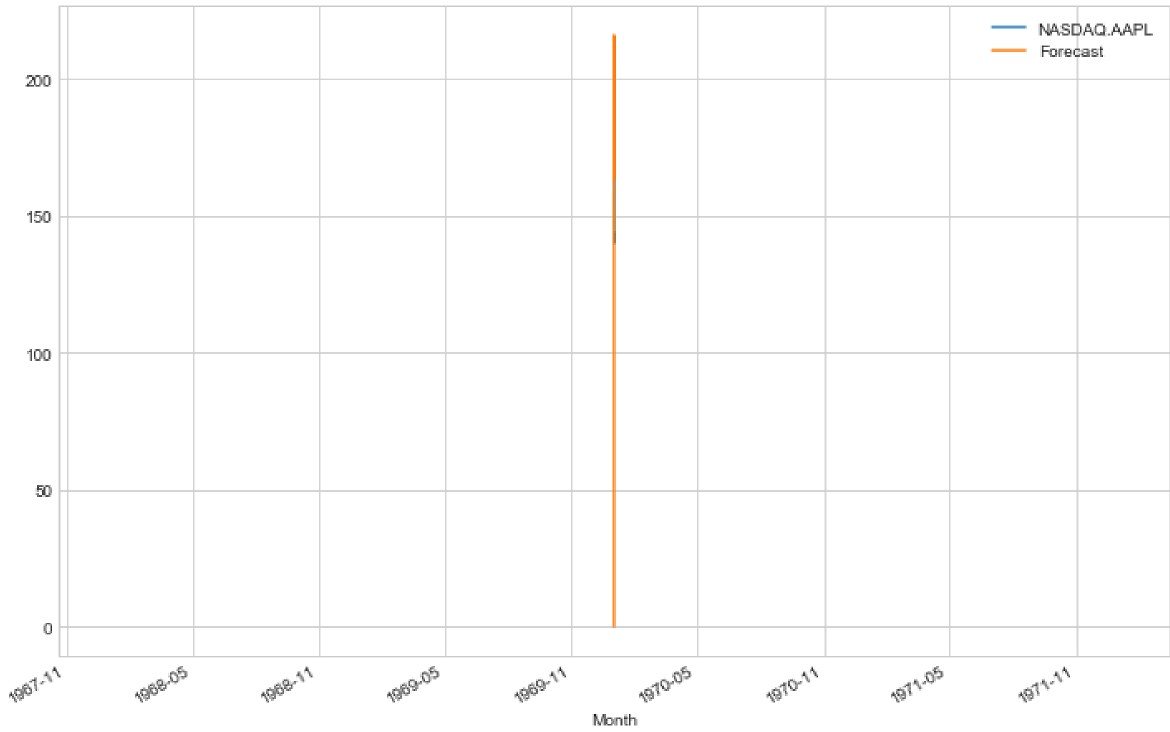
Prediction of Future Values

In [42]:

```
df_AAPL[['NASDAQ.AAPL','Forecast']].plot(figsize=(12,8))
```

Out[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd791358>
```



In [43]:

```
results.forecast(steps=10)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[43]:

```
41265    163.960
41266    163.935
41267    163.910
41268    163.810
41269    163.940
41270    163.950
41271    163.890
41272    163.860
41273    163.870
41274    163.760
dtype: float64
```

In [44]:

```
results.predict(start=41264,end=41274)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[44]:

```
41264    163.930
41265    163.960
41266    163.935
41267    163.910
41268    163.810
41269    163.940
41270    163.950
41271    163.890
41272    163.860
41273    163.870
41274    163.760
dtype: float64
```

Accuracy of the Forecast using MSE-Mean Squared Error

In [45]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Mean Squared Error NASDAQ.AAPL -', mean_squared_error(df_AAPL['NASDAQ.AAPL'],df_AAPL
print('Mean Absolute Error NASDAQ.AAPL -', mean_absolute_error(df_AAPL['NASDAQ.AAPL'],df_AA
```

```
Mean Squared Error NASDAQ.AAPL - 0.6426408211595875
Mean Absolute Error NASDAQ.AAPL - 0.07550728209100216
```

Time Series Forecasting for NASDAQ.ADP

In [46]:

```
df_ADP = final[['Month',stock_features[1]]]
```

In [47]:

```
df_ADP.head()
```

Out[47]:

	Month	NASDAQ.ADP
0	1970-01-01	102.2300
1	1970-01-01	102.1400
2	1970-01-01	102.2125
3	1970-01-01	102.1400
4	1970-01-01	102.0600

In [48]:

```
df_ADP.set_index('Month',inplace=True)
df_ADP.head()
```

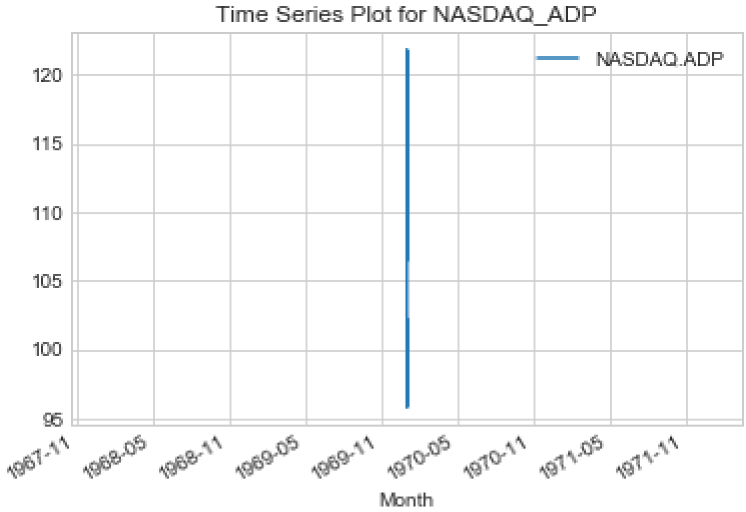
Out[48]:

	NASDAQ.ADP
Month	
1970-01-01	102.2300
1970-01-01	102.1400
1970-01-01	102.2125
1970-01-01	102.1400
1970-01-01	102.0600

Visualize Data

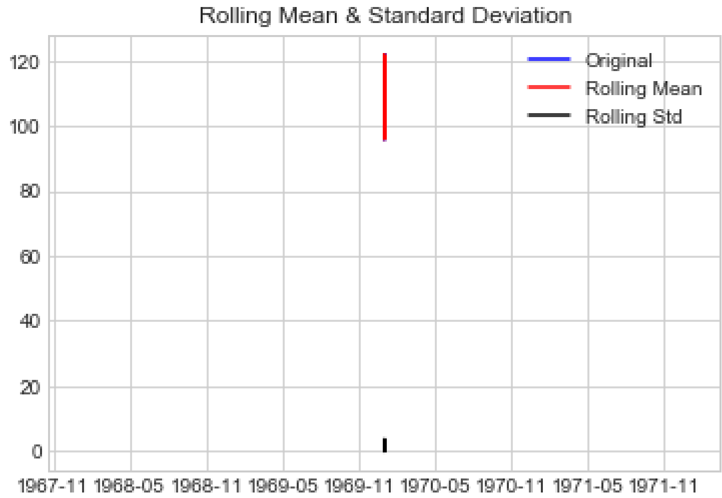
In [49]:

```
df_ADP.plot()
plt.title('Time Series Plot for NASDAQ_ADP')
plt.show()
```



In [50]:

```
test_stationarity(df_ADP['NASDAQ.ADP'])
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -1.7041735251574655
p-value : 0.4289634442066917
#Lags Used : 39
Number of Observations Used : 41226
Crtical 1% : value -3.4305086306509716
Crtical 5% : value -2.861610111161057
Crtical 10% : value -2.5668073179094897
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

MAKING THE TIME SERIES STATIONARY

Differencing

In [51]:

```
df_ADP = df_ADP.copy()
df_ADP['First_Difference'] = df_ADP['NASDAQ.ADP'] - df_ADP['NASDAQ.ADP'].shift(1)
```

In [52]:

```
df_ADP.head()
```

Out[52]:

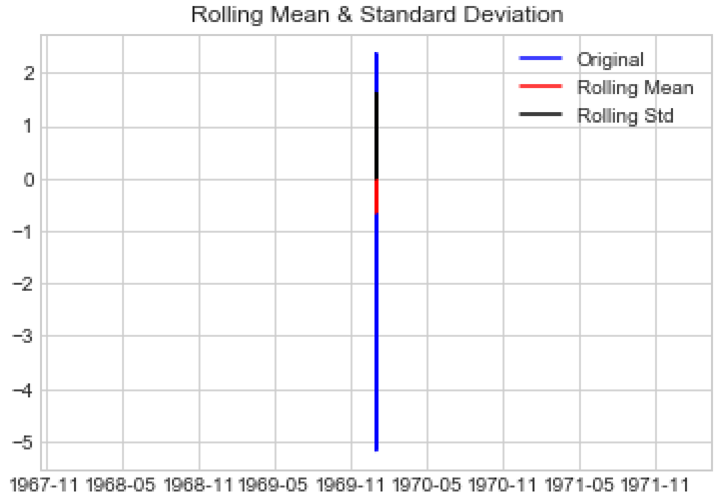
	NASDAQ.ADP	First_Difference
Month		
1970-01-01	102.2300	NaN
1970-01-01	102.1400	-0.0900
1970-01-01	102.2125	0.0725
1970-01-01	102.1400	-0.0725
1970-01-01	102.0600	-0.0800

In [53]:

```
df_ADP.dropna(inplace=True)
```

In [54]:

```
test_stationarity(df_ADP['First_Difference'])
#Now subtract the rolling mean from the original series
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -31.05566224463172
p-value : 0.0
#Lags Used : 38
Number of Observations Used : 41226
Crtical 1% : value -3.4305086306509716
Crtical 5% : value -2.861610111161057
Crtical 10% : value -2.5668073179094897
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

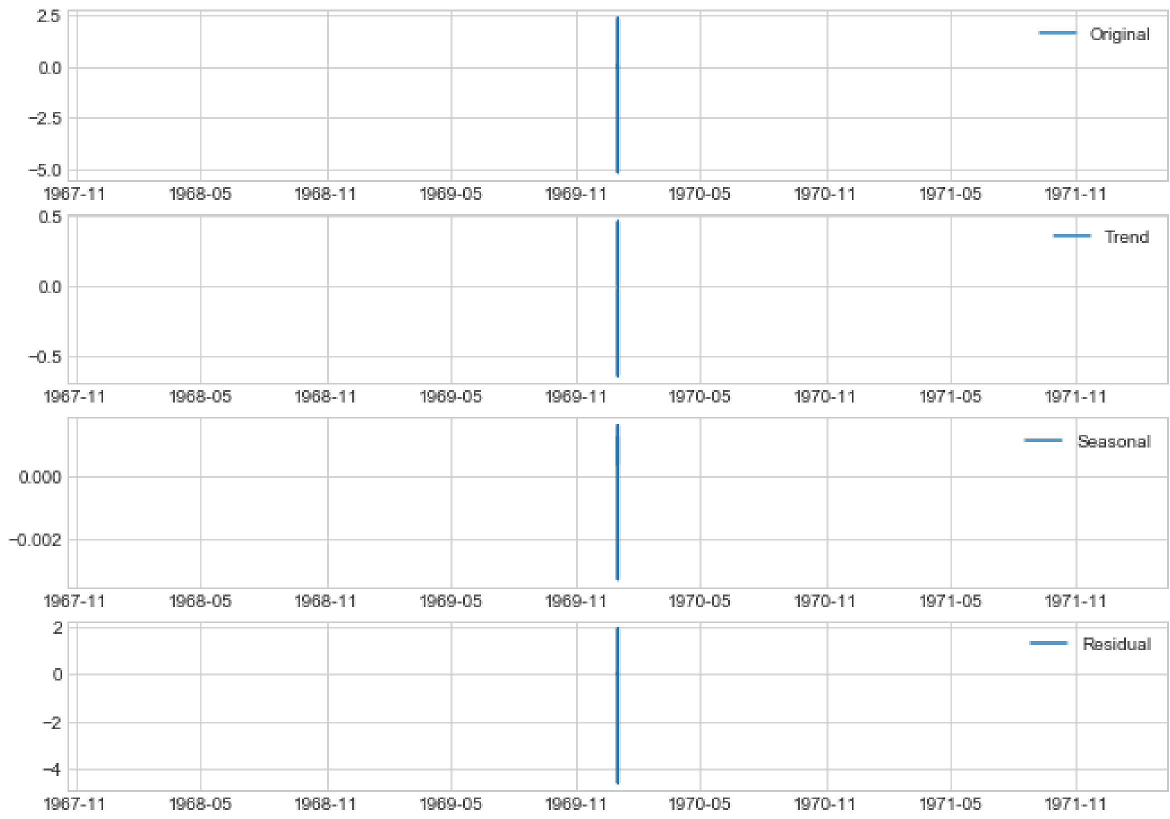
Seasonal Decomposition

In [55]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
plt.figure(figsize=(11,8))
decomposition = seasonal_decompose(df_ADP['First_Difference'],freq=12)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.subplot(411)
plt.plot(df_ADP['First_Difference'],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='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual,label='Residual')
plt.legend(loc='best')
```

Out[55]:

```
<matplotlib.legend.Legend at 0xdd3ec18>
```

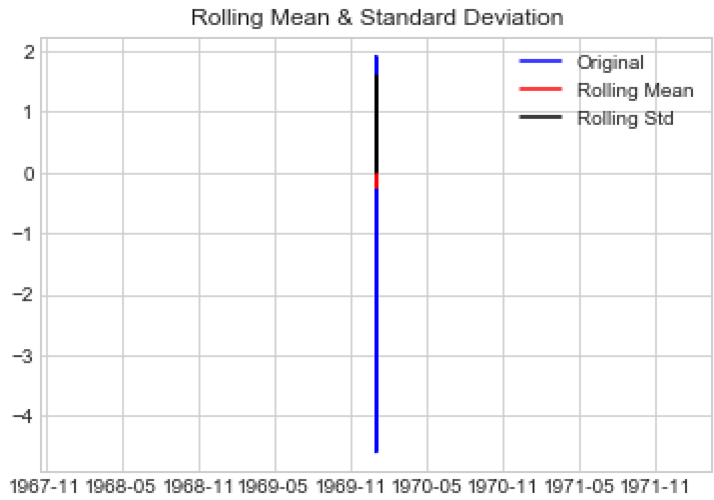


Note:

The data for NASDAQ.ADP is seasonal as interpreted from the seasonal plot of seasonal decomposition.

In [56]:

```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -57.84866544114176
p-value : 0.0
#Lags Used : 55
Number of Observations Used : 41197
Crtical 1% : value -3.4305087423235587
Crtical 5% : value -2.861610160516496
Crtical 10% : value -2.566807344180027
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

Note :

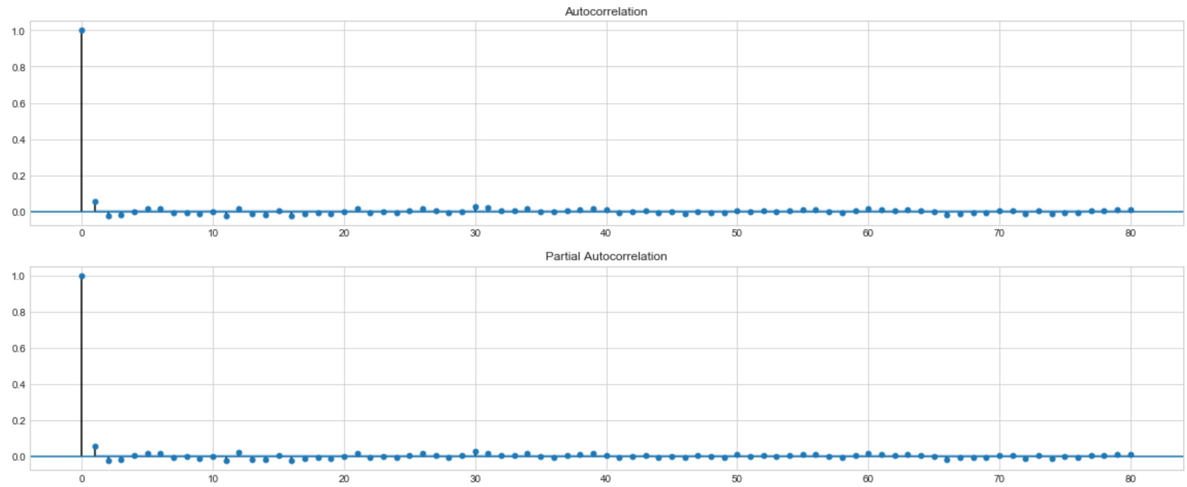
This is stationary because:

- test statistic is lower than 1% critical values.
- the mean and std variations have small variations with time

Autocorrelation and Partial Corelation plot

In [57]:

```
fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_ADP['First_Difference'].iloc[38:], lags=80, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_ADP['First_Difference'].iloc[38:], lags=80, ax=ax2)
```



In [58]:

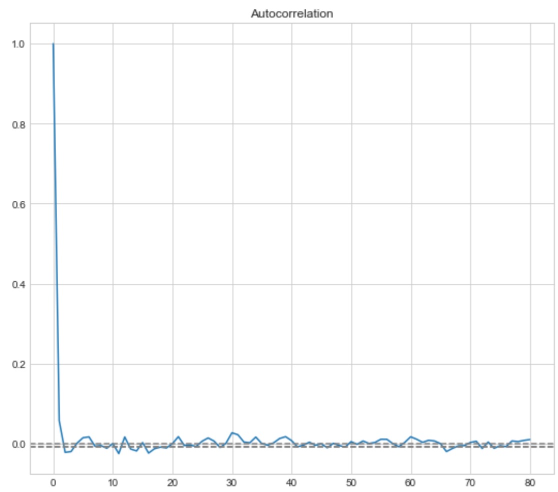
```
lag_acf = acf(df_ADP['First_Difference'],nlags=80)
lag_pacf = pacf(df_ADP['First_Difference'],nlags=80,method='ols')
```

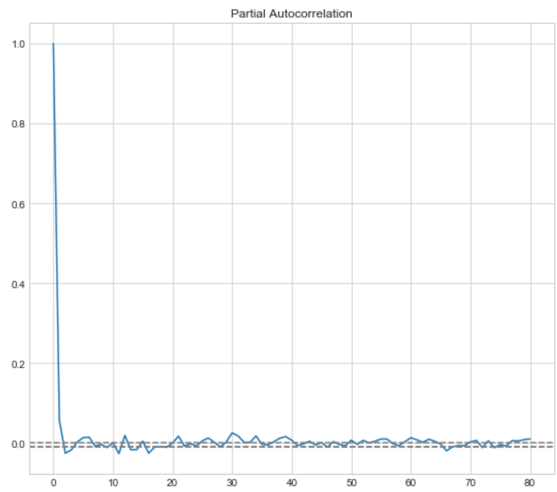
In [59]:

```
plt.figure(figsize=(20,8))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_ADP['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_ADP['First_Difference'])),linestyle='--',color='gray')

plt.title('Autocorrelation')

plt.subplot(122)

plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_ADP['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_ADP['First_Difference'])),linestyle='--',color='gray')

plt.title('Partial Autocorrelation')
```

Out[59]:

```
Text(0.5,1,'Partial Autocorrelation')
```





Note

The two dotted lines on either sides of 0 are the confidence intervals.

These can be used to determine the ‘p’ and ‘q’ values as:

• p: The first time where the PACF crosses the upper confidence interval, here its close to 0. hence p = 0.

• q: The first time where the ACF crosses the upper confidence interval, here its close to 0. hence p = 0.

In [60]:

```
model= sm.tsa.statespace.SARIMAX(df_ADP['NASDAQ.ADP'],order=(0,1,0),seasonal_order=(0,1,0,1
results = model.fit()
print(results.summary())
```

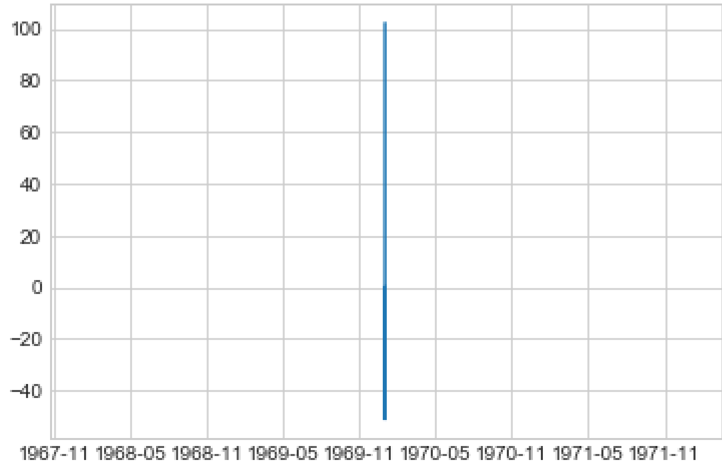
```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:225: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

                                 Statespace Model Results

============================================================================
==============
Dep. Variable:                       NASDAQ.ADP   No. Observations:
         41265
Model:             SARIMAX(0, 1, 0)x(0, 1, 0, 12)   Log Likelihood
     34733.013
Date:                          Tue, 25 Dec 2018   AIC
    -69464.026
Time:                                  20:56:44   BIC
    -69455.399
Sample:                                       0   HQIC
    -69461.299
                                        - 41265

Covariance Type:                            opg

============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
sigma2         0.0109   5.34e-06   2036.710      0.000       0.011       0.0
11
============================================================================
=======
Ljung-Box (Q):                    10628.96   Jarque-Bera (JB):         27526
6211.71
Prob(Q):                              0.00   Prob(JB):
   0.00
Heteroskedasticity (H):               2.20   Skew:
  -1.59
Prob(H) (two-sided):                  0.00   Kurtosis:
403.17
============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```

In [61]:

```
plt.plot(results.resid)
```

Out[61]:

[<matplotlib.lines.Line2D at 0xe018eb8>]

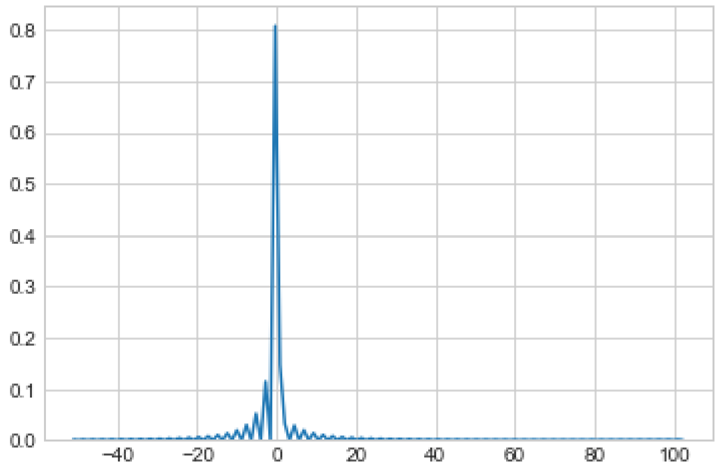


In [62]:

```
import seaborn as sns
sns.set_style('whitegrid')
sns.kdeplot(results.resid)
```

Out[62]:

<matplotlib.axes._subplots.AxesSubplot at 0xe256ac8>



In [63]:

```
df_ADP['Forecast'] = results.predict()
```

In [64]:

```
df_ADP[['NASDAQ.ADP','Forecast']].tail()
```

Out[64]:

	NASDAQ.ADP	Forecast
Month		
1970-01-01	106.565	106.705
1970-01-01	106.590	106.525
1970-01-01	106.520	106.510
1970-01-01	106.400	106.480
1970-01-01	106.470	106.430

In [65]:

```
results.forecast(steps=10)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[65]:

```
41265    106.470
41266    106.470
41267    106.440
41268    106.380
41269    106.440
41270    106.420
41271    106.450
41272    106.385
41273    106.410
41274    106.340
dtype: float64
```

In [66]:

```
results.predict(start=41264,end=41275)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[66]:

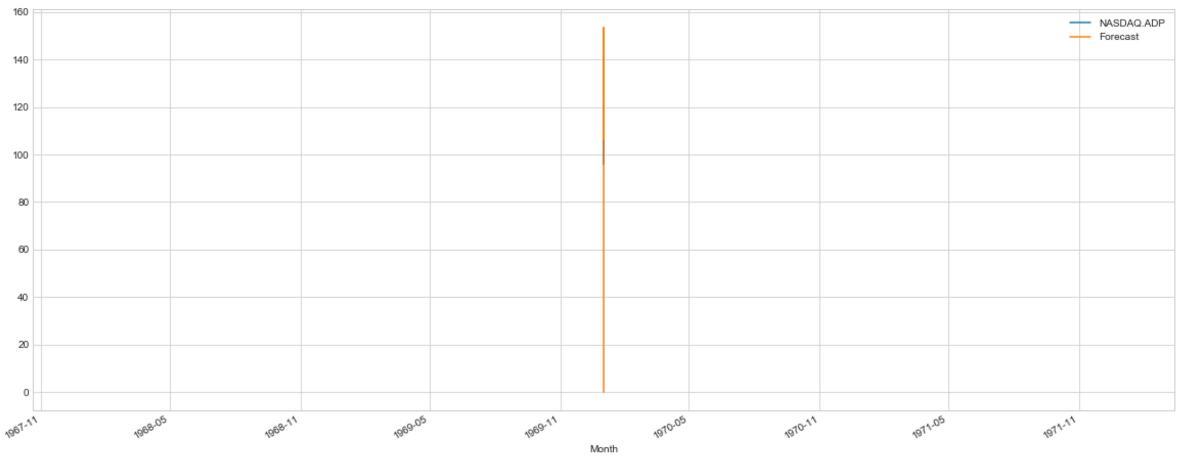
```
41264    106.430
41265    106.470
41266    106.470
41267    106.440
41268    106.380
41269    106.440
41270    106.420
41271    106.450
41272    106.385
41273    106.410
41274    106.340
41275    106.220
dtype: float64
```

In [67]:

```
df_ADP[['NASDAQ.ADP','Forecast']].plot(figsize=(20,8))
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe2c35c0>
```



In [68]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Mean Squared Error NASDAQ.AAPL -', mean_squared_error(df_ADP['NASDAQ.ADP'],df_ADP['F
print('Mean Absolute Error NASDAQ.AAPL -', mean_absolute_error(df_ADP['NASDAQ.ADP'],df_ADP[
```

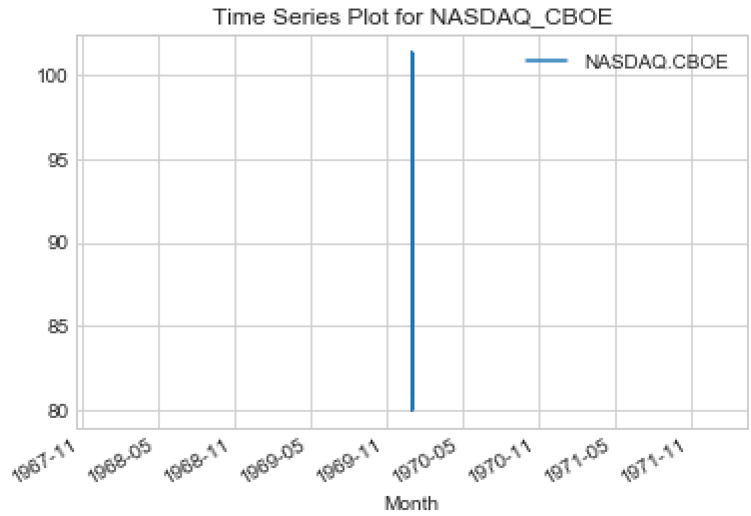
```
Mean Squared Error NASDAQ.AAPL - 0.32679381129889773
Mean Absolute Error NASDAQ.AAPL - 0.05339673819156222
```

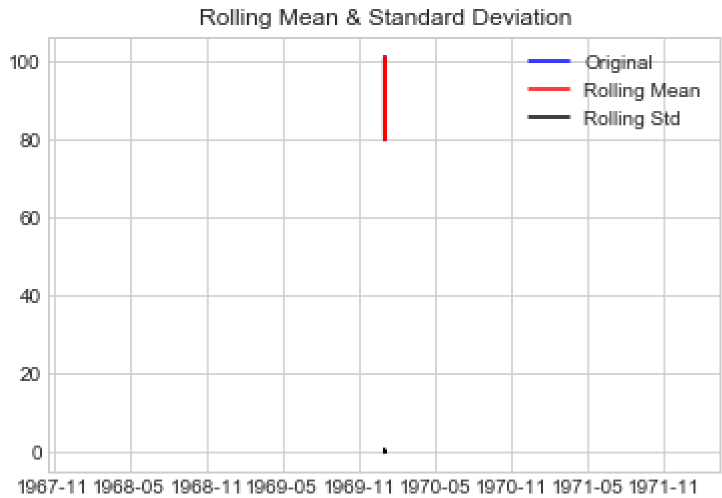
Times Series Forecasting for 'NASDAQ.CBOE'

In [69]:

```
df_CBOE= final[['Month',stock_features[2]]]
print(df_CBOE.head())
df_CBOE.set_index('Month',inplace=True)
print(df_CBOE.head())

df_CBOE.plot()
plt.title('Time Series Plot for NASDAQ_CBOE')
plt.show()
#test Stationarity
test_stationarity(df_CBOE['NASDAQ.CBOE'])
```

```
       Month  NASDAQ.CBOE
0 1970-01-01        81.03
1 1970-01-01        81.21
2 1970-01-01        81.21
3 1970-01-01        81.13
4 1970-01-01        81.12
            NASDAQ.CBOE
Month
1970-01-01        81.03
1970-01-01        81.21
1970-01-01        81.21
1970-01-01        81.13
1970-01-01        81.12
```





```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 0.1663393028261253
p-value : 0.970309203051006
#Lags Used : 27
Number of Observations Used : 41238
Crtical 1% : value -3.430508584487571
Crtical 5% : value -2.8616100907584228
Crtical 10% : value -2.5668073070497304
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

MAKING THE TIME SERIES STATIONARY

Differencing

In [70]:

```
df_CBOE = df_CBOE.copy()
```

In [71]:

```
df_CBOE.head()
```

Out[71]:

	NASDAQ.CBOE
Month	
1970-01-01	81.03
1970-01-01	81.21
1970-01-01	81.21
1970-01-01	81.13
1970-01-01	81.12

In [72]:

```
df_CBOE['First_Difference'] = df_CBOE['NASDAQ.CBOE'] - df_CBOE['NASDAQ.CBOE'].shift(1)
df_CBOE.head()
```

Out[72]:

	NASDAQ.CBOE	First_Difference
Month		
1970-01-01	81.03	NaN
1970-01-01	81.21	0.18
1970-01-01	81.21	0.00
1970-01-01	81.13	-0.08
1970-01-01	81.12	-0.01

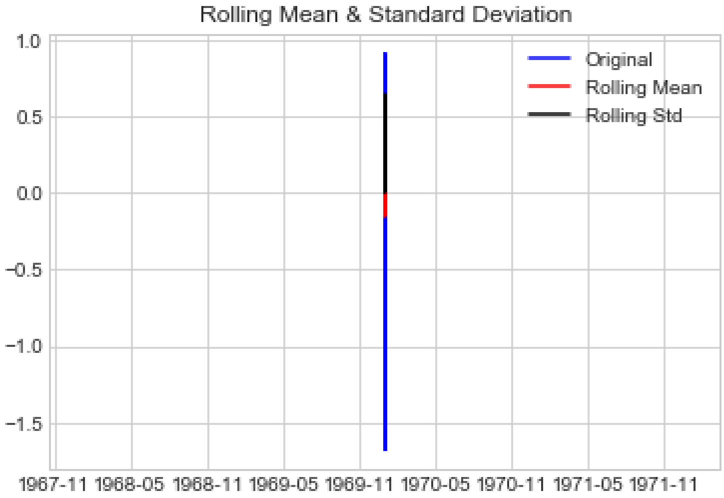
In [73]:

```
df_CBOE.dropna(inplace=True)
```

Test Seasonality

In [74]:

```
test_stationarity(df_CBOE['First_Difference'])
```



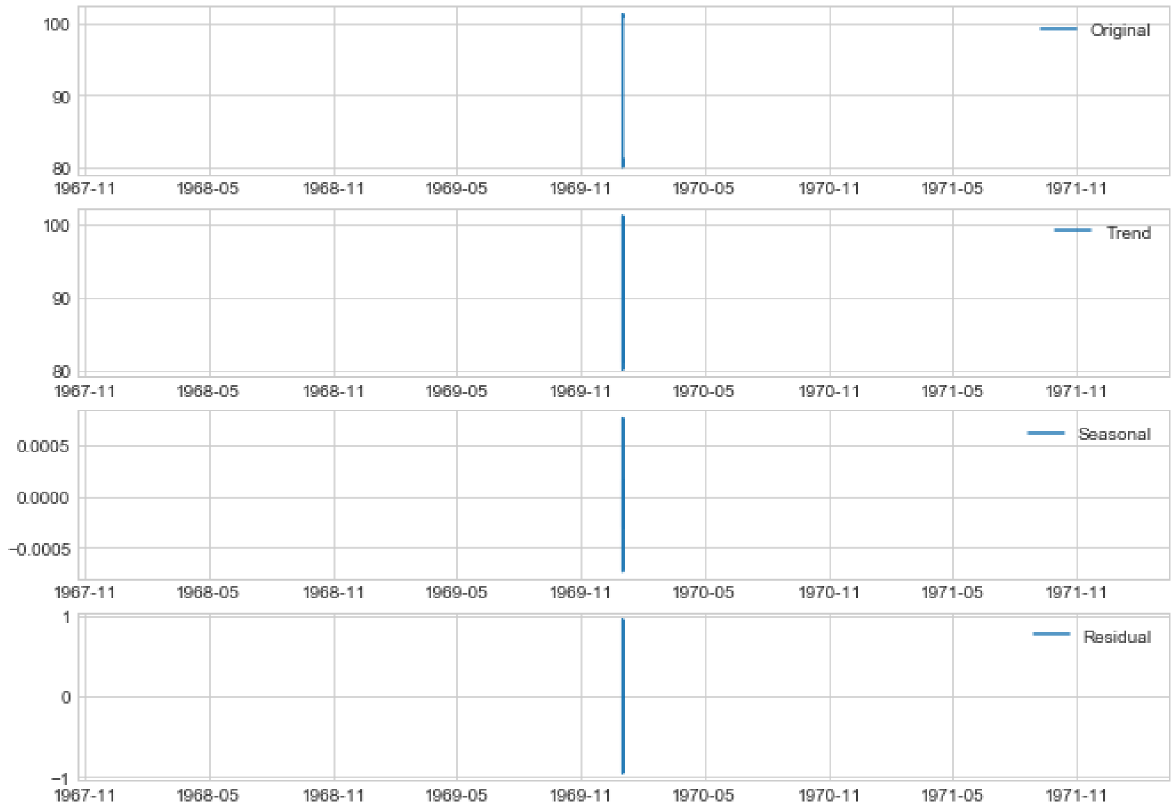

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -41.6420936454317
p-value : 0.0
#Lags Used : 26
Number of Observations Used : 41238
Crtical 1% : value -3.430508584487571
Crtical 5% : value -2.8616100907584228
Crtical 10% : value -2.5668073070497304
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

In [75]:

```
#Seasonal Decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
plt.figure(figsize=(11,8))
decomposition = seasonal_decompose(df_CBOE['NASDAQ.CBOE'],freq=12)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.subplot(411)
plt.plot(df_CBOE['NASDAQ.CBOE'],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='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual,label='Residual')
plt.legend(loc='best')
```

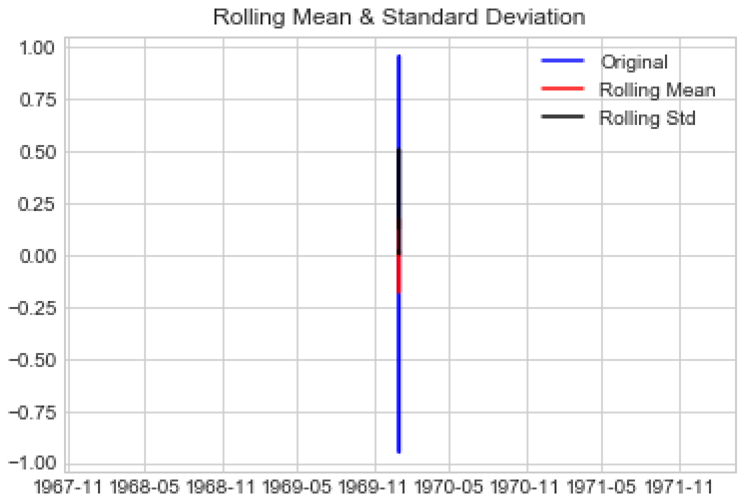
Out[75]:

```
<matplotlib.legend.Legend at 0xd1f7c50>
```



In [76]:

```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -46.21672053215849
p-value : 0.0
#Lags Used : 55
Number of Observations Used : 41197
Crtical 1% : value -3.4305087423235587
Crtical 5% : value -2.861610160516496
Crtical 10% : value -2.566807344180027
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

Note :

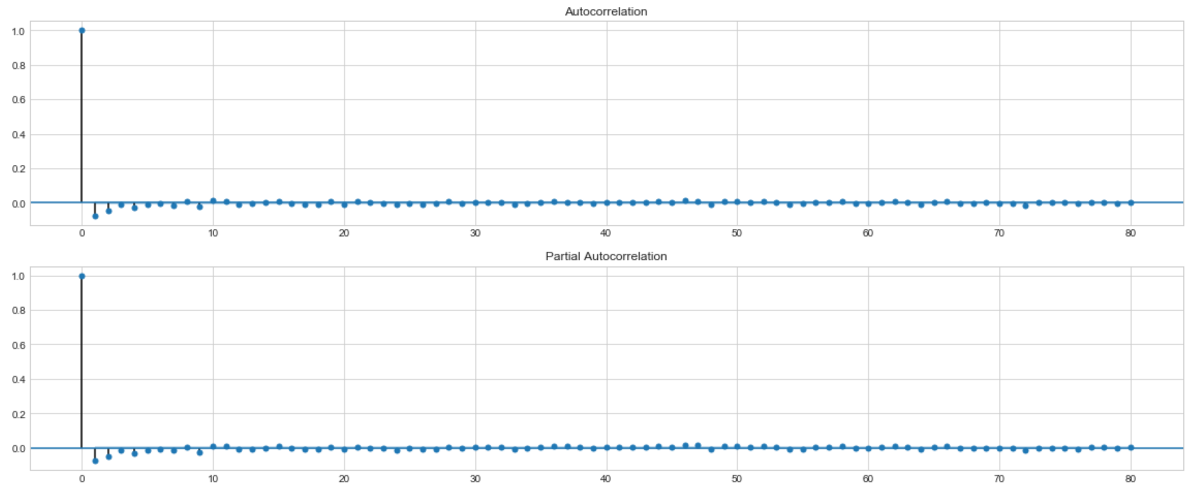
This is stationary because:

- test statistic is lower than 1% critical values.
- the mean and std variations have small variations with time

Autocorrelation and Partial Corelation plot

In [77]:

```
fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_CBOE['First_Difference'].iloc[26:], lags=80, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_CBOE['First_Difference'].iloc[26:], lags=80, ax=ax2)
```

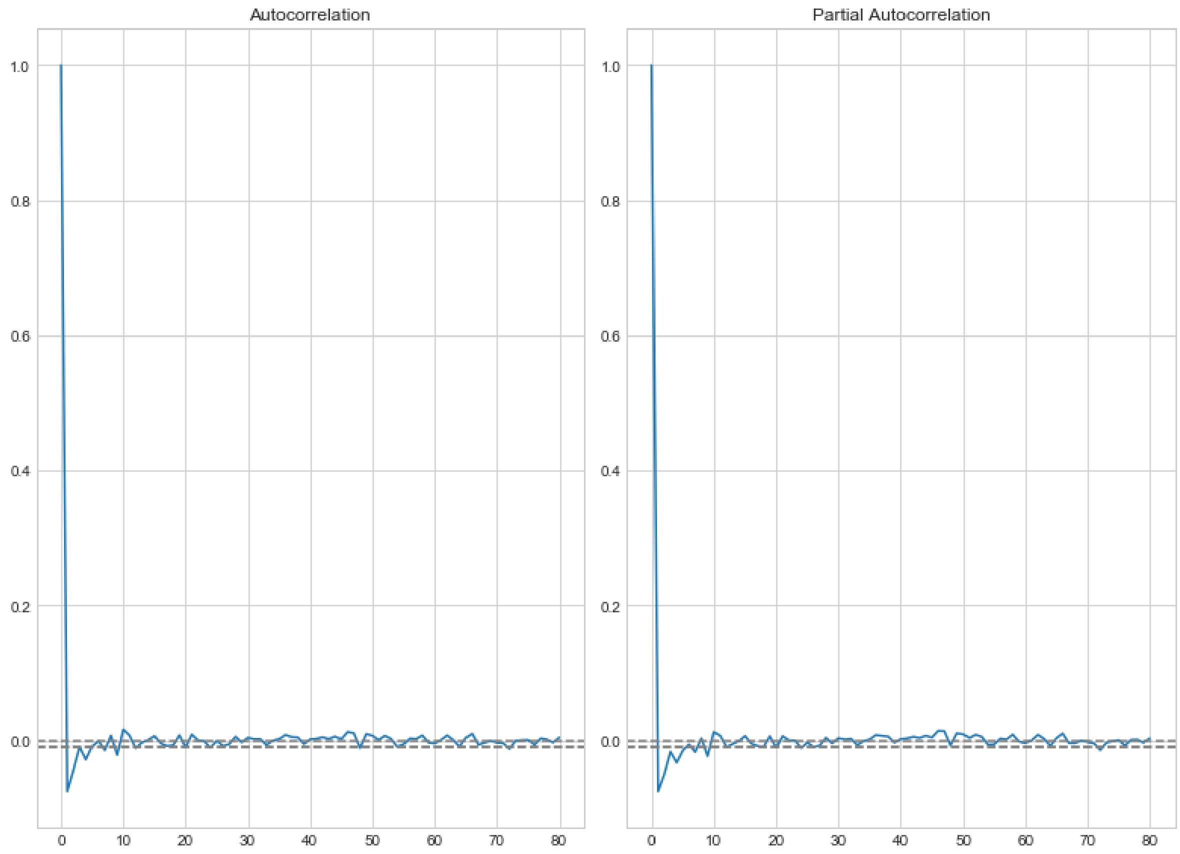


In [78]:

```
lag_acf = acf(df_CBOE['First_Difference'],nlags=80)
lag_pacf = pacf(df_CBOE['First_Difference'],nlags=80,method='ols')
```

In [79]:

```
plt.figure(figsize=(11,8))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CBOE['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CBOE['First_Difference'])),linestyle='--',color='gray')

plt.title('Autocorrelation')

plt.subplot(122)

plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CBOE['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CBOE['First_Difference'])),linestyle='--',color='gray')

plt.title('Partial Autocorrelation')

plt.tight_layout()
```



Note

The two dotted lines on either sides of 0 are the confidence intervals.

These can be used to determine the ‘p’ and ‘q’ values as:

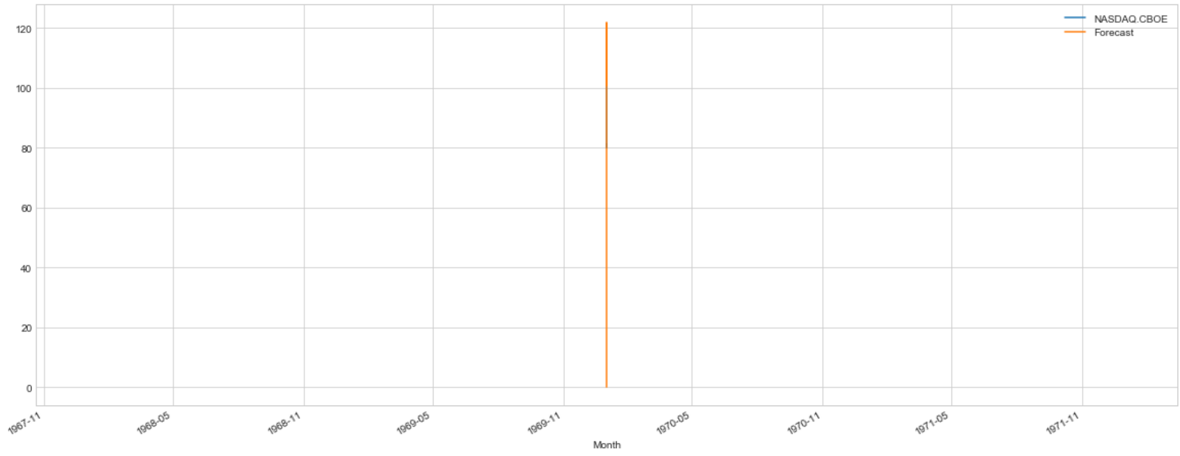
- p: The first time where the PACF crosses the upper confidence interval, here its close to 0. hence p = 0.
- q: The first time where the ACF crosses the upper confidence interval, here its close to 0. hence p = 0.

In [80]:

```
# fit model
model= sm.tsa.statespace.SARIMAX(df_CBOE['NASDAQ.CBOE'],order=(0,1,0),seasonal_order=(0,1,0
results = model.fit()
print(results.summary())
print(results.forecast())
df_CBOE['Forecast'] = results.predict()
df_CBOE[['NASDAQ.CBOE','Forecast']].plot(figsize=(20,8))
plt.show()
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:225: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

                                 Statespace Model Results

============================================================================
==============
Dep. Variable:                        NASDAQ.CBOE   No. Observations:
         41265
Model:             SARIMAX(0, 1, 0)x(0, 1, 0, 12)   Log Likelihood
     53414.092
Date:                            Tue, 25 Dec 2018   AIC
   -106826.184
Time:                                    20:57:40   BIC
   -106817.556
Sample:                                         0   HQIC
   -106823.457
                                          - 41265

Covariance Type:                              opg

============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
sigma2         0.0044   5.33e-06    824.276      0.000       0.004       0.0
04
============================================================================
=======
Ljung-Box (Q):                   11084.06   Jarque-Bera (JB):           701
1759.87
Prob(Q):                             0.00   Prob(JB):
   0.00
Heteroskedasticity (H):              0.94   Skew:
  -0.46
Prob(H) (two-sided):                 0.00   Kurtosis:
  66.86
============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).

C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
```

```
l be given with an integer index beginning at `start`.
  ValueWarning)

41265    100.84
dtype: float64
```



In [81]:

```
results.forecast(steps=10)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[81]:

```
41265    100.8400
41266    100.8900
41267    100.9100
41268    100.8700
41269    100.8800
41270    100.8700
41271    100.8799
41272    100.8800
41273    100.8700
41274    100.8500
dtype: float64
```

In [82]:

```
results.predict(start=41264,end=41273)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[82]:

```
41264    100.8200
41265    100.8400
41266    100.8900
41267    100.9100
41268    100.8700
41269    100.8800
41270    100.8700
41271    100.8799
41272    100.8800
41273    100.8700
dtype: float64
```

In [83]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Mean Squared Error NASDAQ.CBOE -', mean_squared_error(df_CBOE['NASDAQ.CBOE'],df_CBOE
print('Mean Absolute Error NASDAQ.CBOE -', mean_absolute_error(df_CBOE['NASDAQ.CBOE'],df_CB
```

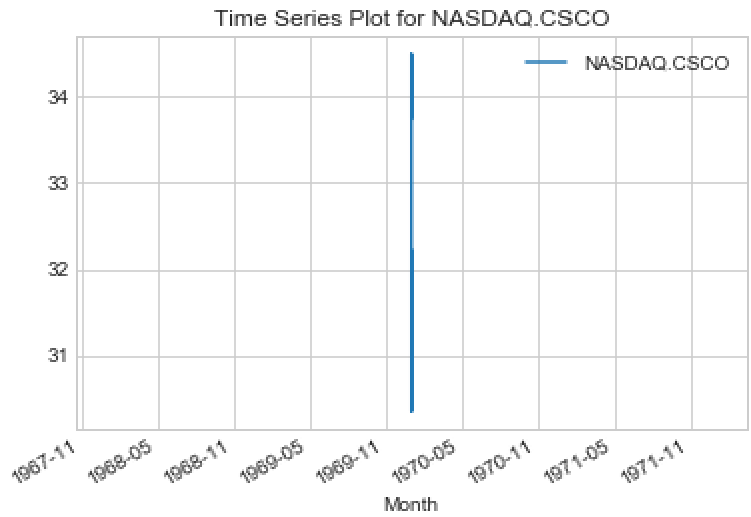
```
Mean Squared Error NASDAQ.CBOE - 0.2039940019832608
Mean Absolute Error NASDAQ.CBOE - 0.04356630559048824
```

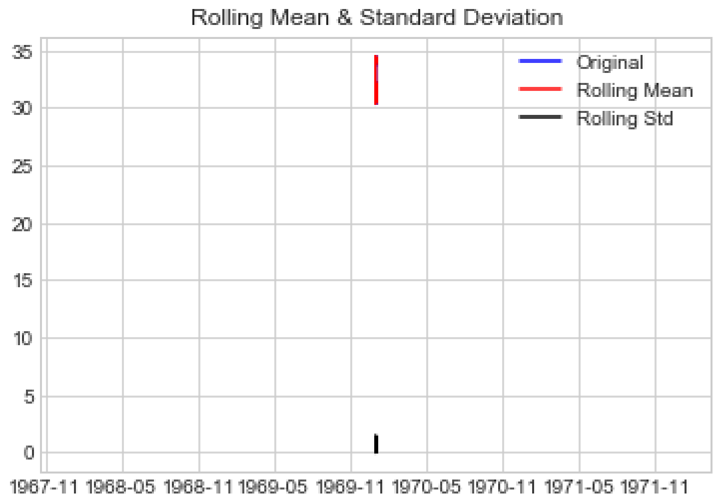
Time Series ForeCasting for 'NASDAQ.CSCO'

In [84]:

```
df_CSCO = final[['Month',stock_features[3]]]
print(df_CSCO.head())
df_CSCO.set_index('Month',inplace=True)
print(df_CSCO.head())
df_CSCO.plot()
plt.title("Time Series Plot for NASDAQ.CSCO")
plt.show()
#Test Staionarity
test_stationarity(df_CSCO['NASDAQ.CSCO'])
```

```
        Month  NASDAQ.CSCO
0 1970-01-01      33.7400
1 1970-01-01      33.8800
2 1970-01-01      33.9000
3 1970-01-01      33.8499
4 1970-01-01      33.8400
            NASDAQ.CSCO
Month
1970-01-01      33.7400
1970-01-01      33.8800
1970-01-01      33.9000
1970-01-01      33.8499
1970-01-01      33.8400
```





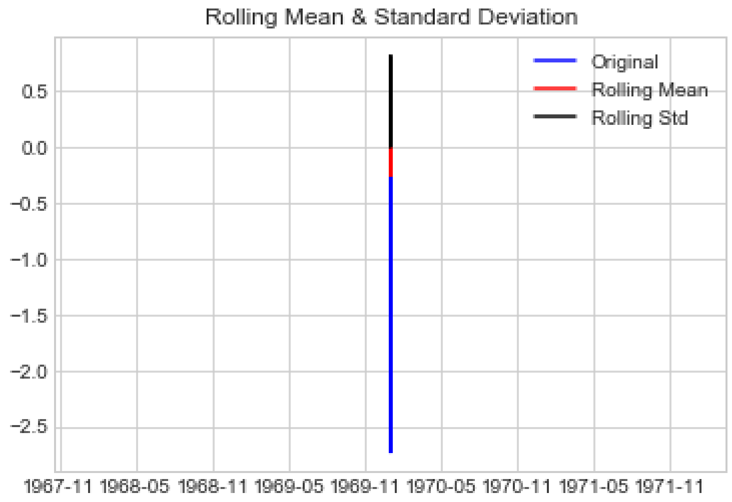
```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -2.395554610889476
p-value : 0.1429950199516406
#Lags Used : 47
Number of Observations Used : 41218
Crtical 1% : value -3.430508661441506
Crtical 5% : value -2.8616101247694137
Crtical 10% : value -2.566807325152842
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

MAKING TIME SERIES STATIONARY

Differencing

In [85]:

```
df_CSCO = df_CSCO.copy()
df_CSCO['First_Difference'] = df_CSCO['NASDAQ.CSCO'] - df_CSCO['NASDAQ.CSCO'].shift(1)
df_CSCO.dropna(inplace=True)
test_stationarity(df_CSCO['First_Difference'])
```



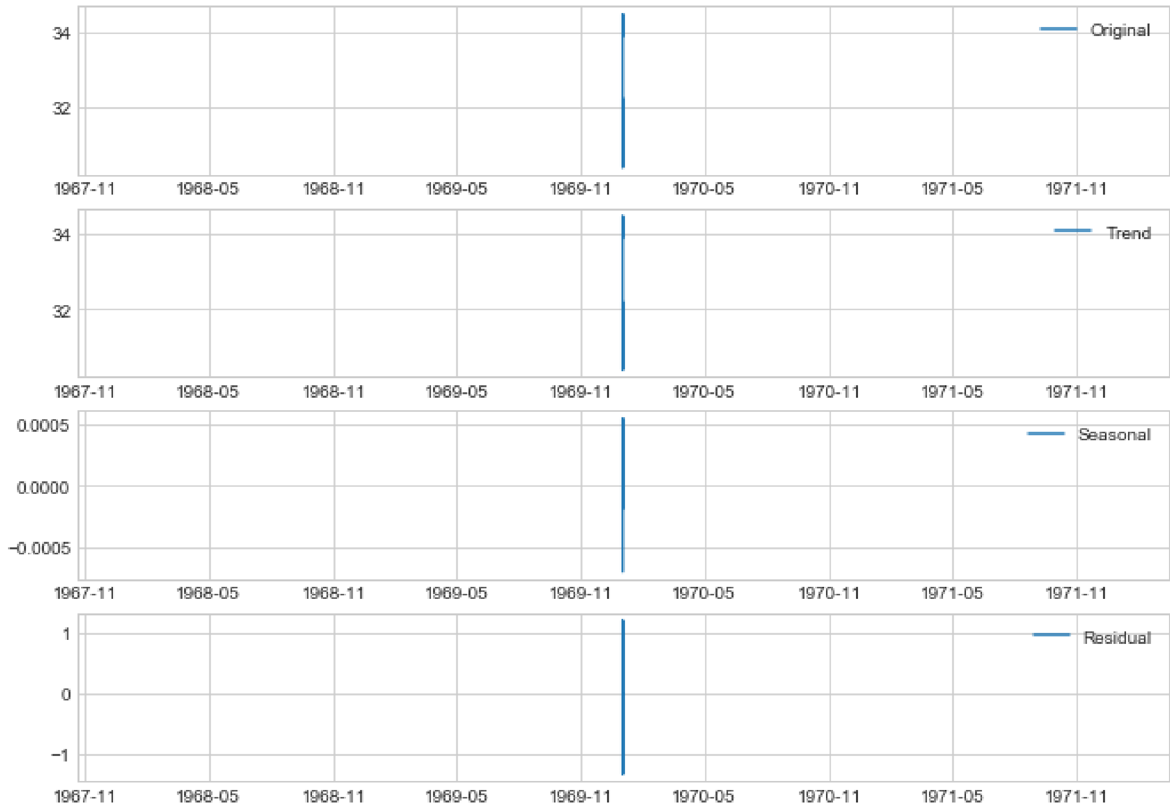

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -30.356682532566758
p-value : 0.0
#Lags Used : 46
Number of Observations Used : 41218
Crtical 1% : value -3.430508661441506
Crtical 5% : value -2.8616101247694137
Crtical 10% : value -2.566807325152842
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

In [86]:

```
#Seasonal Decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
plt.figure(figsize=(11,8))
decomposition = seasonal_decompose(df_CSCO['NASDAQ.CSCO'],freq=12)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.subplot(411)
plt.plot(df_CSCO['NASDAQ.CSCO'],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='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual,label='Residual')
plt.legend(loc='best')
```

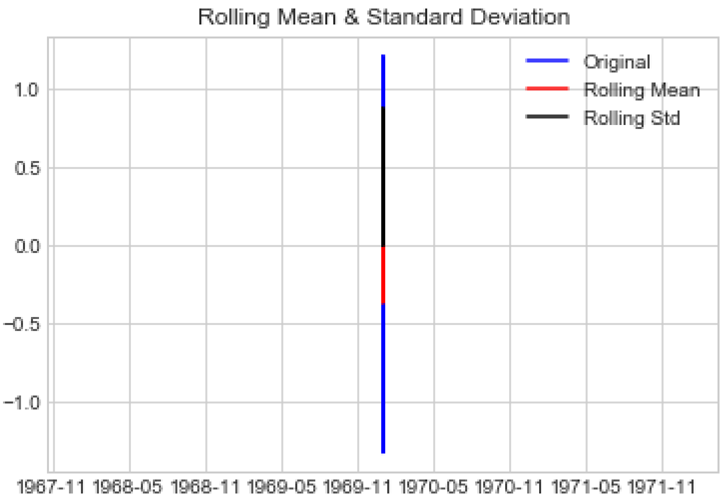
Out[86]:

```
<matplotlib.legend.Legend at 0x2a345fd0>
```



In [87]:

```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -43.94517780543416
p-value : 0.0
#Lags Used : 55
Number of Observations Used : 41197
Crtical 1% : value -3.4305087423235587
Crtical 5% : value -2.861610160516496
Crtical 10% : value -2.566807344180027
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

Note :

This is stationary because:

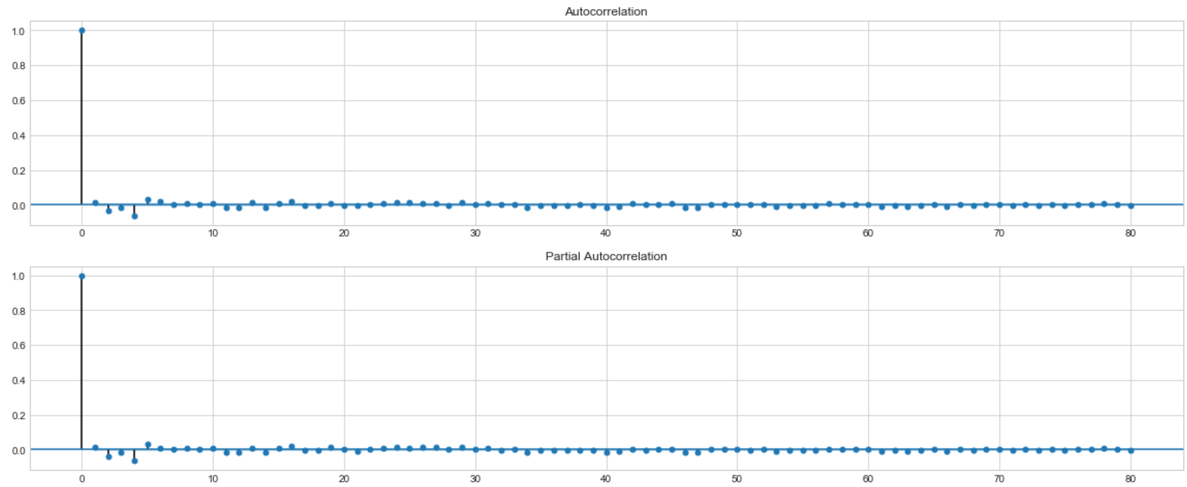
• test statistic is lower than critical values.

• the mean and std variations have small variations with time

Auto Corealtion and Partial Autocorelation Plots

In [88]:

```
fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_CSCO['First_Difference'].iloc[46:], lags=80, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_CSCO['First_Difference'].iloc[46:], lags=80, ax=ax2)
```

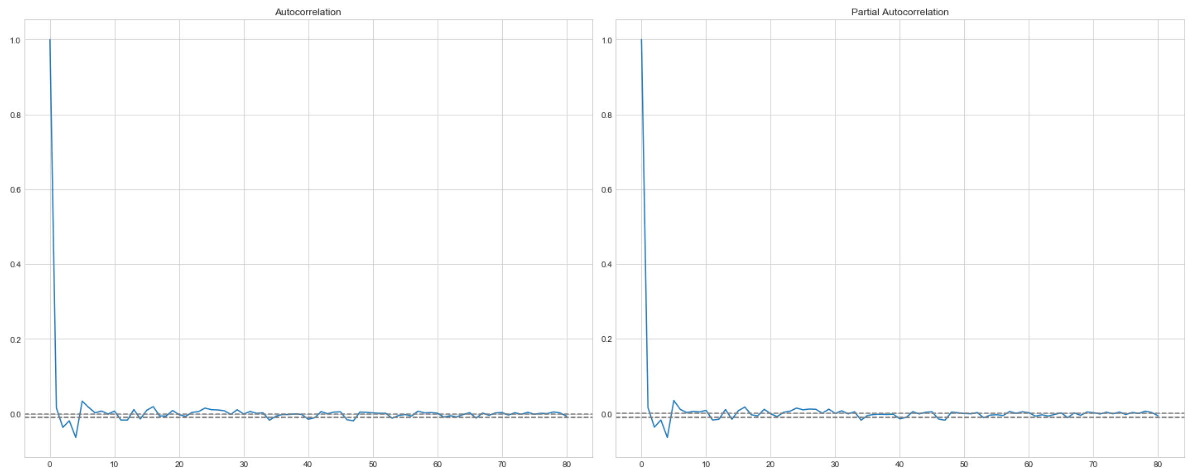


In [89]:

```
lag_acf = acf(df_CSCO['First_Difference'],nlags=80)
lag_pacf = pacf(df_CSCO['First_Difference'],nlags=80,method='ols')
```

In [90]:

```
plt.figure(figsize=(20,8))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CSCO['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CSCO['First_Difference'])),linestyle='--',color='gray')

plt.title('Autocorrelation')

plt.subplot(122)

plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CSCO['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_CSCO['First_Difference'])),linestyle='--',color='gray')

plt.title('Partial Autocorrelation')

plt.tight_layout()
```



Note

The two dotted lines on either sides of 0 are the confidence intervals.

These can be used to determine the ‘p’ and ‘q’ values as:

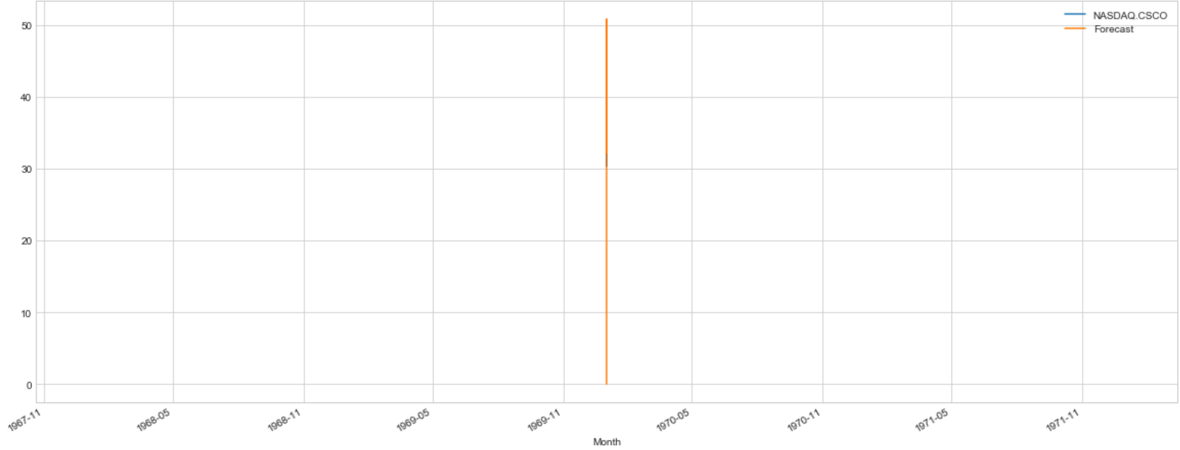
• p: The first time where the PACF crosses the upper confidence interval, here its close to 0. hence p = 0.

• q: The first time where the ACF crosses the upper confidence interval, here its close to 0. hence p = 0.

In [91]:

```
# fit model
model= sm.tsa.statespace.SARIMAX(df_CSCO['NASDAQ.CSCO'],order=(0,1,0),seasonal_order=(0,1,0
results = model.fit()
print(results.summary())
df_CSCO['Forecast'] = results.predict()
df_CSCO[['NASDAQ.CSCO','Forecast']].plot(figsize=(20,8))
plt.show()
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:225: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

                                 Statespace Model Results                     
                                                                              
=============================================================================
==============
Dep. Variable:                     NASDAQ.CSCO   No. Observations:           
         41265
Model:             SARIMAX(0, 1, 0)x(0, 1, 0, 12)   Log Likelihood              
     85502.595
Date:                         Tue, 25 Dec 2018   AIC                         
   -171003.190
Time:                                 20:58:34   BIC                         
   -170994.563
Sample:                                      0   HQIC                        
   -171000.463
                                       - 41265                                
                                                                              
Covariance Type:                           opg                                
                                                                              
=============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
-----------------------------------------------------------------------------
--
sigma2         0.0009   1.54e-07   6012.819      0.000       0.001       0.0
01
=============================================================================
=======
Ljung-Box (Q):                    11736.64   Jarque-Bera (JB):       2107338
2447.00
Prob(Q):                              0.00   Prob(JB):                      
   0.00
Heteroskedasticity (H):               0.30   Skew:                          
   2.67
Prob(H) (two-sided):                  0.00   Kurtosis:                      
3504.46
=============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```



In [92]:

```
df_CSCO.head()
```

Out[92]:

	NASDAQ.CSCO	First_Difference	Forecast
Month			
1970-01-01	33.8800	0.1400	0.0000
1970-01-01	33.9000	0.0200	33.8800
1970-01-01	33.8499	-0.0501	33.9000
1970-01-01	33.8400	-0.0099	33.8499
1970-01-01	33.8800	0.0400	33.8400

In [93]:

```
results.forecast(steps=10)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[93]:

```
41265    32.225
41266    32.190
41267    32.170
41268    32.150
41269    32.180
41270    32.170
41271    32.150
41272    32.165
41273    32.180
41274    32.180
dtype: float64
```

In [94]:

```
results.predict(start=41264,end=41275)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[94]:

```
41264    32.195
41265    32.225
41266    32.190
41267    32.170
41268    32.150
41269    32.180
41270    32.170
41271    32.150
41272    32.165
41273    32.180
41274    32.180
41275    32.175
dtype: float64
```

In [95]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Mean Squared Error NASDAQ.CSCO -', mean_squared_error(df_CSCO['NASDAQ.CSCO'],df_CSCO
print('Mean Absolute Error NASDAQ.CSCO -', mean_absolute_error(df_CSCO['NASDAQ.CSCO'],df_CS
```

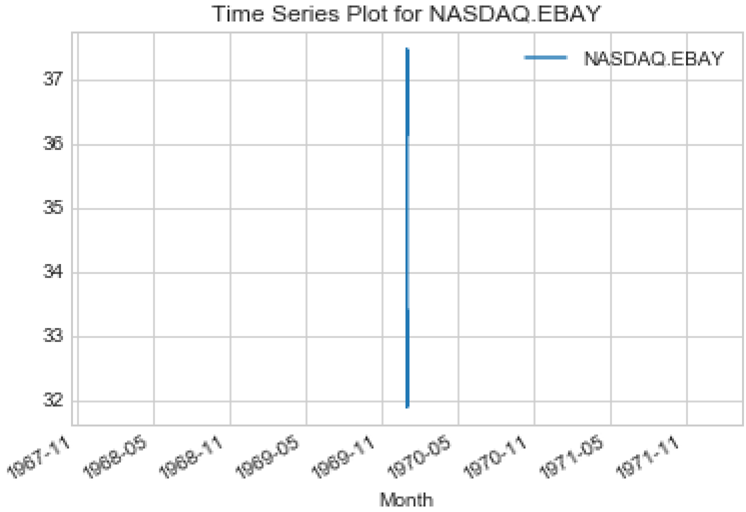
```
Mean Squared Error NASDAQ.CSCO - 0.03569378449696608
Mean Absolute Error NASDAQ.CSCO - 0.015775407730929
```

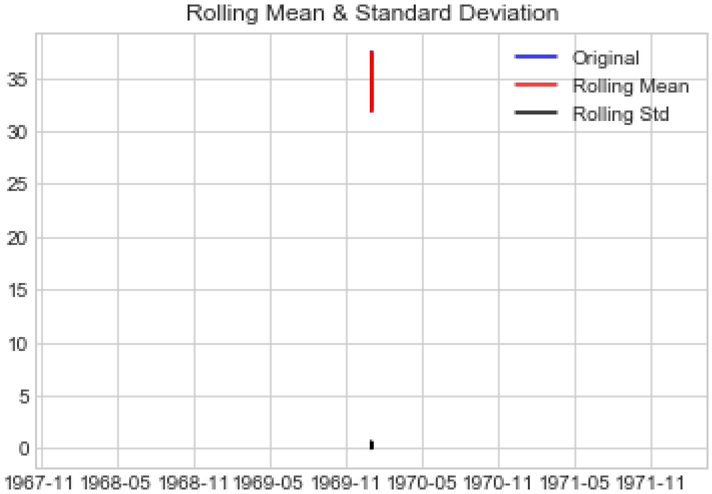
Time Series Forecasting for NASDAQ.EBAY

In [96]:

```
df_EBAY = final[['Month',stock_features[4]]]
print(df_EBAY.head())
df_EBAY.set_index('Month',inplace=True)
print(df_EBAY.head())
df_EBAY.plot()
plt.title("Time Series Plot for NASDAQ.EBAY")
plt.show()
#Test Staionarity
test_stationarity(df_EBAY['NASDAQ.EBAY'])
```

```
        Month  NASDAQ.EBAY
0 1970-01-01      33.3975
1 1970-01-01      33.3950
2 1970-01-01      33.4100
3 1970-01-01      33.3350
4 1970-01-01      33.4000
            NASDAQ.EBAY
Month
1970-01-01      33.3975
1970-01-01      33.3950
1970-01-01      33.4100
1970-01-01      33.3350
1970-01-01      33.4000
```






```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -1.8757616359413931
```

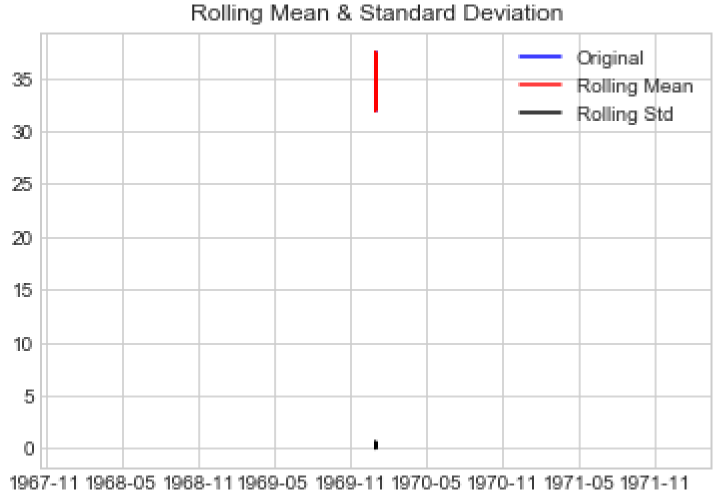
```
p-value : 0.3435480878024858
#Lags Used : 47
Number of Observations Used : 41218
Crtical 1% : value -3.430508661441506
Crtical 5% : value -2.8616101247694137
Crtical 10% : value -2.566807325152842
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

MAKING TIME SERIES STATIONARY

Differencing

In [97]:

```
df_EBAY = df_EBAY.copy()
df_EBAY['First_Difference'] = df_EBAY['NASDAQ.EBAY'] - df_EBAY['NASDAQ.EBAY'].shift(1)
df_EBAY.dropna(inplace=True)
#test Stationarity
test_stationarity(df_EBAY['NASDAQ.EBAY'])
```



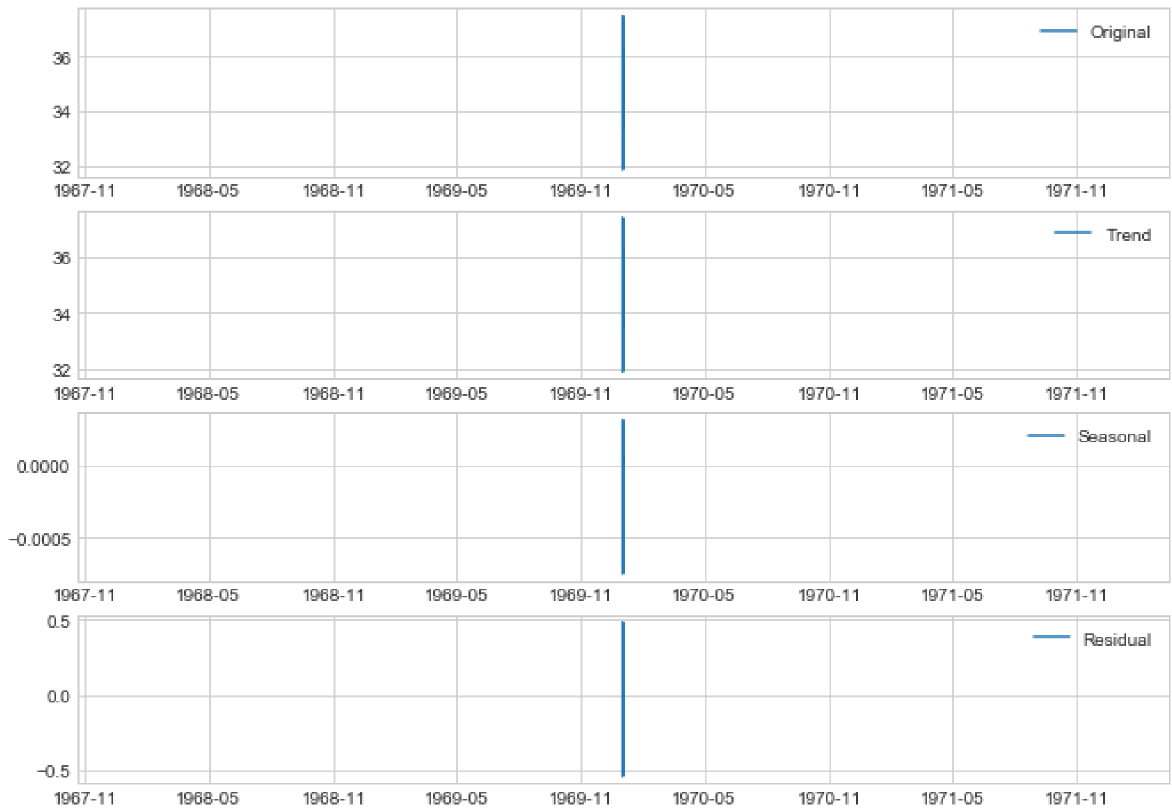

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -1.863913310658429
p-value : 0.3492231149987333
#Lags Used : 47
Number of Observations Used : 41217
Crtical 1% : value -3.4305086652911636
Crtical 5% : value -2.8616101264708296
Crtical 10% : value -2.5668073260584587
weak evidence against null hypothesis, time series has a unit root, indicati
ng it is non-stationary
```

In [98]:

```
#Seasonal Decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
plt.figure(figsize=(11,8))
decomposition = seasonal_decompose(df_EBAY['NASDAQ.EBAY'],freq=12)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.subplot(411)
plt.plot(df_EBAY['NASDAQ.EBAY'],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='Seasonal')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual,label='Residual')
plt.legend(loc='best')
```

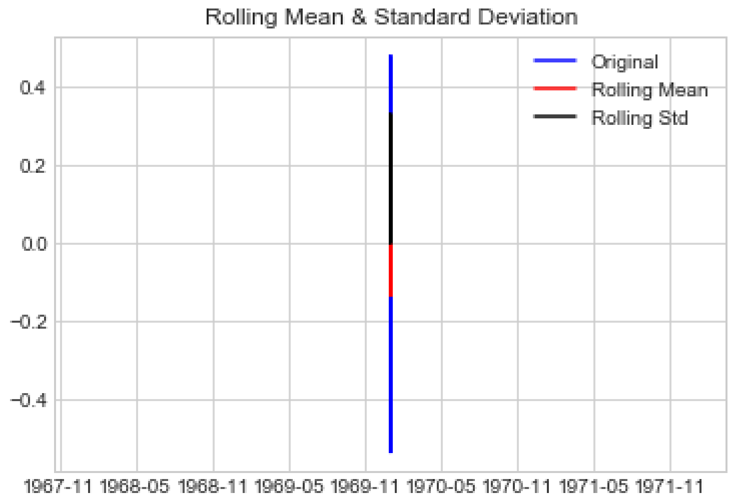
Out[98]:

```
<matplotlib.legend.Legend at 0x1a30a390>
```



In [99]:

```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```




```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -44.880491758920314
p-value : 0.0
#Lags Used : 55
Number of Observations Used : 41197
Crtical 1% : value -3.4305087423235587
Crtical 5% : value -2.861610160516496
Crtical 10% : value -2.566807344180027
strong evidence against the null hypothesis, reject the null hypothesis. Dat
a has no unit root and is stationary
```

Note :

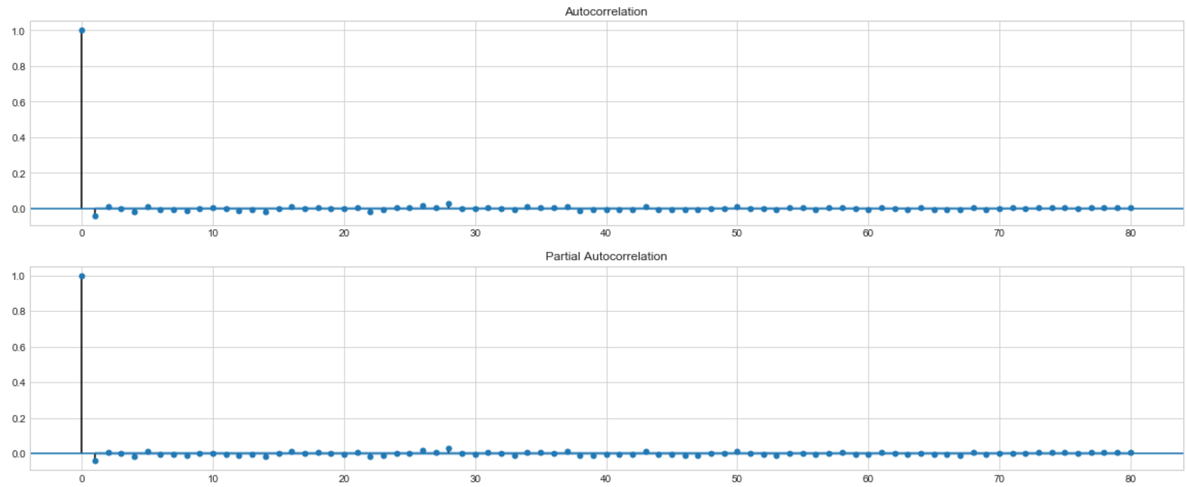
This is stationary because:

- test statistic is lower than critical values.
- the mean and std variations have small variations with time

Autocorealtion plot and Partial Autocorelation plots

In [100]:

```
fig = plt.figure(figsize=(20,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_EBAY['First_Difference'].iloc[47:], lags=80, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_EBAY['First_Difference'].iloc[47:], lags=80, ax=ax2)
```

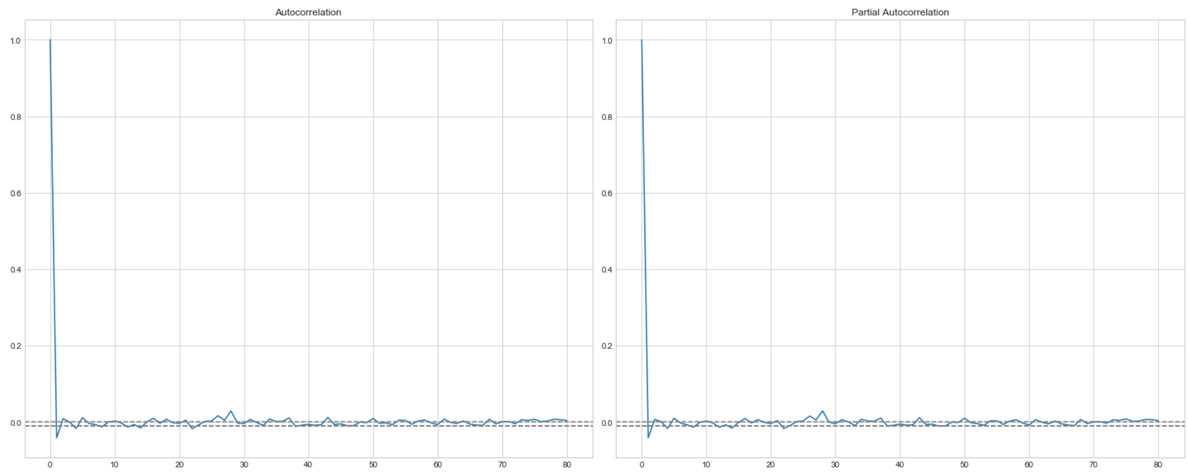


In [101]:

```


lag_acf = acf(df_EBAY['First_Difference'],nlags=80)
lag_pacf = pacf(df_EBAY['First_Difference'],nlags=80,method='ols')
```

In [102]:

```
plt.figure(figsize=(20,8))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_EBAY['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_EBAY['First_Difference'])),linestyle='--',color='gray')

plt.title('Autocorrelation')

plt.subplot(122)

plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_EBAY['First_Difference'])),linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df_EBAY['First_Difference'])),linestyle='--',color='gray')

plt.title('Partial Autocorrelation')

plt.tight_layout()
```



Note

The two dotted lines on either sides of 0 are the confidence intervals.

These can be used to determine the ‘p’ and ‘q’ values as:

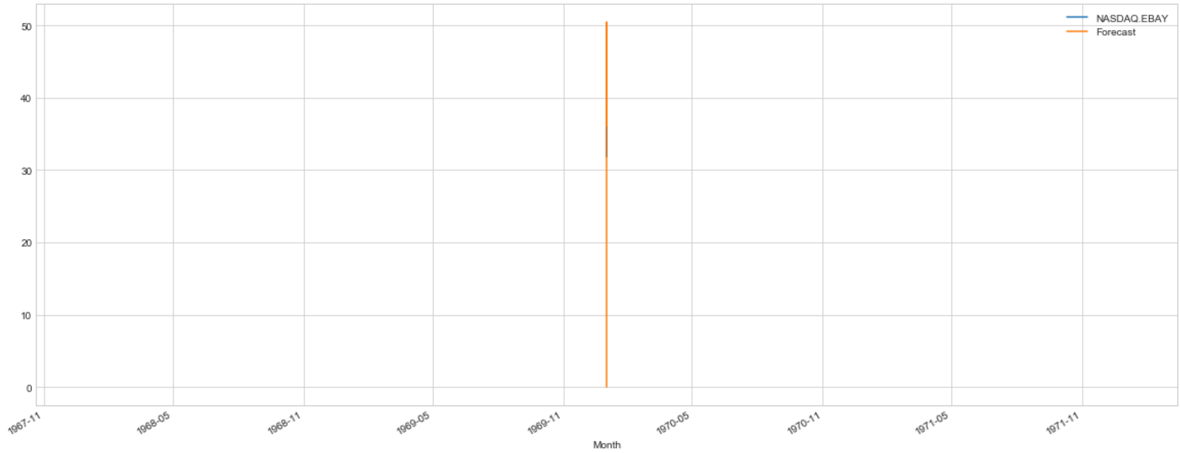
• p: The first time where the PACF crosses the upper confidence interval, here its close to 0. hence p = 0.

• q: The first time where the ACF crosses the upper confidence interval, here its close to 0. hence p = 0.

In [103]:

```
# fit model
model= sm.tsa.statespace.SARIMAX(df_EBAY['NASDAQ.EBAY'],order=(0,1,0),seasonal_order=(0,1,0
results = model.fit()
print(results.summary())
df_EBAY['Forecast'] = results.predict()
df_EBAY[['NASDAQ.EBAY','Forecast']].plot(figsize=(20,8))
plt.show()
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:225: ValueWarning: A date index has been provided, but it has no associate
d frequency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

                           Statespace Model Results

============================================================================
==============
Dep. Variable:                    NASDAQ.EBAY   No. Observations:
      41265
Model:             SARIMAX(0, 1, 0)x(0, 1, 0, 12)   Log Likelihood
    82104.712
Date:                        Tue, 25 Dec 2018   AIC
   -164207.424
Time:                                20:59:23   BIC
   -164198.797
Sample:                                     0   HQIC
   -164204.697
                                      - 41265

Covariance Type:                          opg

============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
----------------------------------------------------------------------------
--
sigma2         0.0011   9.43e-07   1158.843      0.000       0.001       0.0
01
============================================================================
=======
Ljung-Box (Q):                  10939.63   Jarque-Bera (JB):          2822
3015.29
Prob(Q):                            0.00   Prob(JB):
   0.00
Heteroskedasticity (H):             1.21   Skew:
   0.35
Prob(H) (two-sided):                0.00   Kurtosis:
131.14
============================================================================
=======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (compl
ex-step).
```



In [104]:

```
df_EBAY.head()
```

Out[104]:

	NASDAQ.EBAY	First_Difference	Forecast
Month			
1970-01-01	33.395	-0.0025	0.000
1970-01-01	33.410	0.0150	33.395
1970-01-01	33.335	-0.0750	33.410
1970-01-01	33.400	0.0650	33.335
1970-01-01	33.430	0.0300	33.400

In [105]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Mean Squared Error NASDAQ.EBAY -', mean_squared_error(df_EBAY['NASDAQ.EBAY'],df_EBAY
print('Mean Absolute Error NASDAQ.EBAY -', mean_absolute_error(df_EBAY['NASDAQ.EBAY'],df_EE
```

```
Mean Squared Error NASDAQ.EBAY - 0.03483567894575233
Mean Absolute Error NASDAQ.EBAY - 0.021688033609985485
```

In [106]:

```
results.forecast(steps=10)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[106]:

```
41265    36.090
41266    36.030
41267    36.030
41268    36.020
41269    36.020
41270    36.025
41271    36.020
41272    36.025
41273    36.020
41274    36.020
dtype: float64
```

In [107]:

```
 results.predict(start=41265,end=41275)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.p
y:531: ValueWarning: No supported index is available. Prediction results wil
l be given with an integer index beginning at `start`.
  ValueWarning)
```

Out[107]:

```
41265    36.090
41266    36.030
41267    36.030
41268    36.020
41269    36.020
41270    36.025
41271    36.020
41272    36.025
41273    36.020
41274    36.020
41275    36.010
dtype: float64
```

CONCLUSION : The predicted stock prices values have been stored in the Forecast Columns of the each stock entity dataframe