In [1]:
```
#Import data and libraries

import pandas as pd
import numpy as np
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt
from datetime import datetime
from pmdarima import auto_arima
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.stattools import acf,pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
from math import sqrt
import scipy.stats as st
import warnings
warnings.filterwarnings("ignore")

#Increase size of plot
from matplotlib.pylab import rcParams
rcParams['figure.figsize']= 12,6

file = 'C:/Users/cdric/OneDrive/Desktop/School/D214/avocado20-23.csv'
data = pd.read_csv(file)
print(data.head())
print(data.tail())
```

```
               Geography Timeframe Week_Ending         Type       ASP  \
0                 Albany    Weekly   1/13/2020  Conventional  1.227609
1                 Albany    Weekly   1/13/2020       Organic  1.814664
2                Atlanta    Weekly   1/13/2020  Conventional  0.924121
3                Atlanta    Weekly   1/13/2020       Organic  1.422905
4  Baltimore/Washington    Weekly   1/13/2020  Conventional  1.337480

   4046_Units  4225_Units  4770_Units  TotalBagged_Units
0     4019.89    96152.64      205.88           16359.88
1      185.96      175.80        0.00            1399.22
2   551153.18    94886.96     2526.53          107631.15
3    11106.05    10932.16        0.00            7524.48
4    95768.26   450660.70     4451.67          276413.89
                  Geography Timeframe Week_Ending         Type       ASP  \
20763                  West    Weekly   5/21/2023       Organic  1.519799
20764    West Tex/New Mexico    Weekly   5/21/2023  Conventional  0.816465
20765    West Tex/New Mexico    Weekly   5/21/2023       Organic  1.175170
20766               Wichita    Weekly   5/21/2023  Conventional  0.901398
20767               Wichita    Weekly   5/21/2023       Organic  1.534101

       4046_Units  4225_Units  4770_Units  TotalBagged_Units
20763   107819.36    93752.04       44.84          138061.44
20764   498229.25    89374.53     2444.56          429555.58
20765     7645.39      586.25        0.00           15632.56
20766    49347.48    37732.21       85.50           34758.94
20767      921.41       10.64        0.00            1125.60
```

In [2]:
```
#Keep relevant columns
```

```
data=data.drop(columns = ['4046_Units','4225_Units','4770_Units','TotalBagged_Units'])
print(data.head())
```

```
             Geography Timeframe Week_Ending          Type       ASP
0               Albany    Weekly   1/13/2020  Conventional  1.227609
1               Albany    Weekly   1/13/2020       Organic  1.814664
2              Atlanta    Weekly   1/13/2020  Conventional  0.924121
3              Atlanta    Weekly   1/13/2020       Organic  1.422905
4  Baltimore/Washington    Weekly   1/13/2020  Conventional  1.337480
```

In [3]:
```
#Data exploration and cleaning

data['Timeframe'].unique()
```

Out[3]:
```
array(['Weekly'], dtype=object)
```

In [4]:
```
data['Type'].unique()
```

Out[4]:
```
array(['Conventional', 'Organic'], dtype=object)
```

In [5]:
```
data['Geography'].unique()
```

Out[5]:
```
array(['Albany', 'Atlanta', 'Baltimore/Washington',
       'Birmingham/Montgomery', 'Boise', 'Boston', 'Buffalo/Rochester',
       'California', 'Charlotte', 'Chicago', 'Cincinnati/Dayton',
       'Columbus', 'Dallas/Ft. Worth', 'Denver', 'Detroit',
       'Grand Rapids', 'Great Lakes', 'Harrisburg/Scranton',
       'Hartford/Springfield', 'Houston', 'Indianapolis', 'Jacksonville',
       'Las Vegas', 'Los Angeles', 'Louisville', 'Miami/Ft. Lauderdale',
       'Midsouth', 'Nashville', 'New Orleans/Mobile', 'New York',
       'Northeast', 'Northern New England', 'Orlando',
       'Peoria/Springfield', 'Philadelphia', 'Phoenix/Tucson',
       'Pittsburgh', 'Plains', 'Portland', 'Providence',
       'Raleigh/Greensboro', 'Richmond/Norfolk', 'Roanoke', 'Sacramento',
       'San Diego', 'San Francisco', 'Seattle', 'South Carolina',
       'South Central', 'Southeast', 'Spokane', 'St. Louis', 'Syracuse',
       'Tampa', 'Toledo', 'Total U.S.', 'West', 'West Tex/New Mexico',
       'Wichita'], dtype=object)
```

In [6]:
```
#Check for nulls

data.isnull().any()
```

Out[6]:
```
Geography      False
Timeframe      False
Week_Ending    False
Type           False
ASP            False
dtype: bool
```

In [7]:
```
#Drop any row that does not have 'Total U.S.' as the Geography
#https://sparkbyexamples.com/pandas/pandas-delete-rows-based-on-column-value/

data.drop(data[data['Geography'] != 'Total U.S.'].index, inplace = True)
data
```

Out[7]:

	Geography	Timeframe	Week_Ending	Type	ASP
110	Total U.S.	Weekly	1/13/2020	Conventional	1.015298
111	Total U.S.	Weekly	1/13/2020	Organic	1.498858
228	Total U.S.	Weekly	1/19/2020	Conventional	0.955296
229	Total U.S.	Weekly	1/19/2020	Organic	1.455650
346	Total U.S.	Weekly	1/26/2020	Conventional	1.067556
...	...	...	...	...	...
20525	Total U.S.	Weekly	5/7/2023	Organic	1.445973
20642	Total U.S.	Weekly	5/14/2023	Conventional	1.030217
20643	Total U.S.	Weekly	5/14/2023	Organic	1.453891
20760	Total U.S.	Weekly	5/21/2023	Conventional	1.050588
20761	Total U.S.	Weekly	5/21/2023	Organic	1.460592

352 rows × 5 columns

In [8]:

```
#Create dataframe with just Organic prices

data_org = data.loc[data['Type'] !='Conventional']
print(data_org)
```

```
         Geography Timeframe Week_Ending     Type       ASP
111     Total U.S.    Weekly   1/13/2020  Organic  1.498858
229     Total U.S.    Weekly   1/19/2020  Organic  1.455650
347     Total U.S.    Weekly   1/26/2020  Organic  1.571187
465     Total U.S.    Weekly    2/2/2020  Organic  1.494387
583     Total U.S.    Weekly    2/9/2020  Organic  1.518099
...            ...       ...         ...      ...       ...
20289   Total U.S.    Weekly   4/23/2023  Organic  1.435000
20407   Total U.S.    Weekly   4/30/2023  Organic  1.442130
20525   Total U.S.    Weekly    5/7/2023  Organic  1.445973
20643   Total U.S.    Weekly   5/14/2023  Organic  1.453891
20761   Total U.S.    Weekly   5/21/2023  Organic  1.460592

[176 rows x 5 columns]
```

In [9]:

```
#Create dataframe with just Conventional prices

data_conv = data.loc[data['Type'] !='Organic']
print(data_conv)
```

```
         Geography Timeframe Week_Ending         Type       ASP
110    Total U.S.    Weekly   1/13/2020  Conventional  1.015298
228    Total U.S.    Weekly   1/19/2020  Conventional  0.955296
346    Total U.S.    Weekly   1/26/2020  Conventional  1.067556
464    Total U.S.    Weekly    2/2/2020  Conventional  0.898828
582    Total U.S.    Weekly    2/9/2020  Conventional  1.000716
...           ...       ...         ...           ...       ...
20288  Total U.S.    Weekly   4/23/2023  Conventional  1.024829
20406  Total U.S.    Weekly   4/30/2023  Conventional  1.070490
20524  Total U.S.    Weekly    5/7/2023  Conventional  0.952580
20642  Total U.S.    Weekly   5/14/2023  Conventional  1.030217
20760  Total U.S.    Weekly   5/21/2023  Conventional  1.050588

[176 rows x 5 columns]
```

In [10]:
```
data_org['Date'] = pd.date_range(start=datetime(2020,1,12), end=datetime(2023,5,21), f
data_conv['Date'] = pd.date_range(start=datetime(2020,1,12), end=datetime(2023,5,21),
#Drop 'Day' column
data_conv.drop(['Week_Ending'], axis=1,inplace=True)
data_org.drop(['Week_Ending'], axis=1,inplace=True)
```

In [11]:
```
#Convert org data to datetime
#https://favtutor.com/blogs/datetime-from-string-python

data_org['Date']= pd.to_datetime(data_org['Date'],infer_datetime_format=True)
idata_org = data_org.set_index(['Date'])
idata_org=idata_org.dropna()
print(idata_org.head())
print(idata_org.shape)
```

```
             Geography Timeframe     Type       ASP
Date
2020-01-12  Total U.S.    Weekly  Organic  1.498858
2020-01-19  Total U.S.    Weekly  Organic  1.455650
2020-01-26  Total U.S.    Weekly  Organic  1.571187
2020-02-02  Total U.S.    Weekly  Organic  1.494387
2020-02-09  Total U.S.    Weekly  Organic  1.518099
(176, 4)
```

In [12]:
```
#Keep only date and price in org df

idata_org=idata_org.drop(columns = ['Geography','Timeframe','Type'])
print(idata_org.head())
```

```
                 ASP
Date
2020-01-12  1.498858
2020-01-19  1.455650
2020-01-26  1.571187
2020-02-02  1.494387
2020-02-09  1.518099
```

In [13]:
```
#Convert conv data to datetime
#https://favtutor.com/blogs/datetime-from-string-python

data_conv['Date']= pd.to_datetime(data_conv['Date'],infer_datetime_format=True)
idata_conv = data_conv.set_index(['Date'])
idata_conv=idata_conv.dropna()
print(idata_conv.head())
print(idata_conv.shape)
```

```
              Geography Timeframe         Type       ASP
Date
2020-01-12  Total U.S.    Weekly  Conventional  1.015298
2020-01-19  Total U.S.    Weekly  Conventional  0.955296
2020-01-26  Total U.S.    Weekly  Conventional  1.067556
2020-02-02  Total U.S.    Weekly  Conventional  0.898828
2020-02-09  Total U.S.    Weekly  Conventional  1.000716
(176, 4)
```

In [14]:

```
#Keep only date and price in conv df

idata_conv=idata_conv.drop(columns = ['Geography','Timeframe','Type'])
print(idata_conv.head())
```

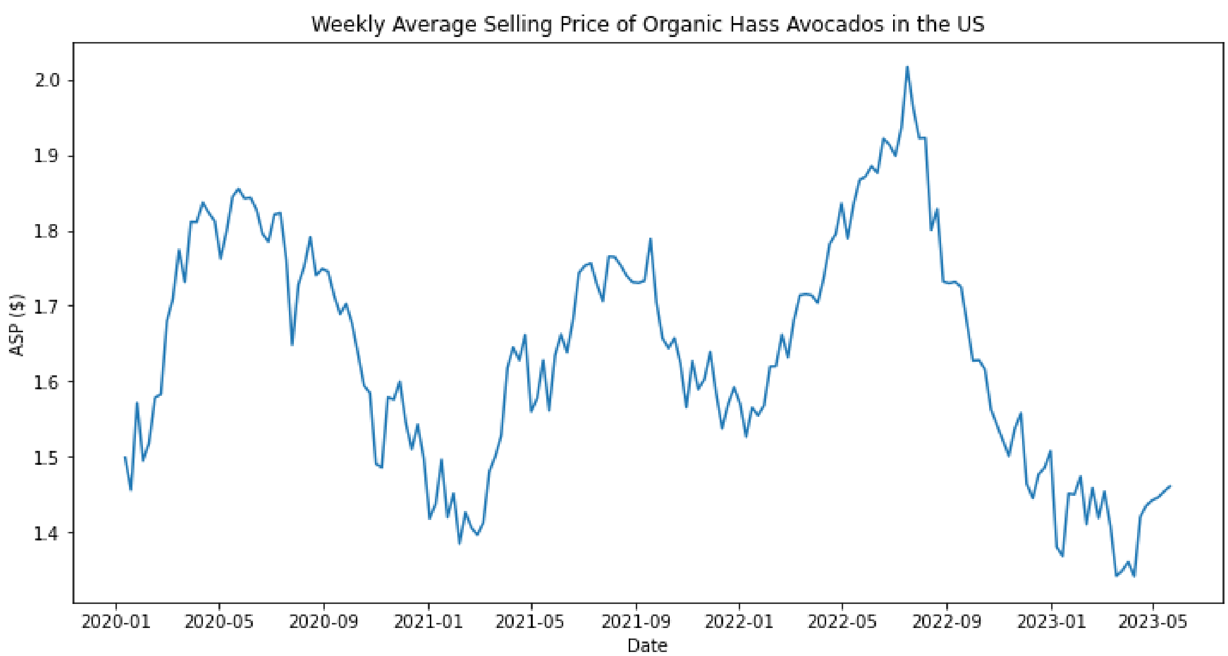
```
                 ASP
Date
2020-01-12  1.015298
2020-01-19  0.955296
2020-01-26  1.067556
2020-02-02  0.898828
2020-02-09  1.000716
```

In [15]:

```
#Plot org data

plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.title('Weekly Average Selling Price of Organic Hass Avocados in the US')
plt.plot(idata_org)
```

Out[15]:

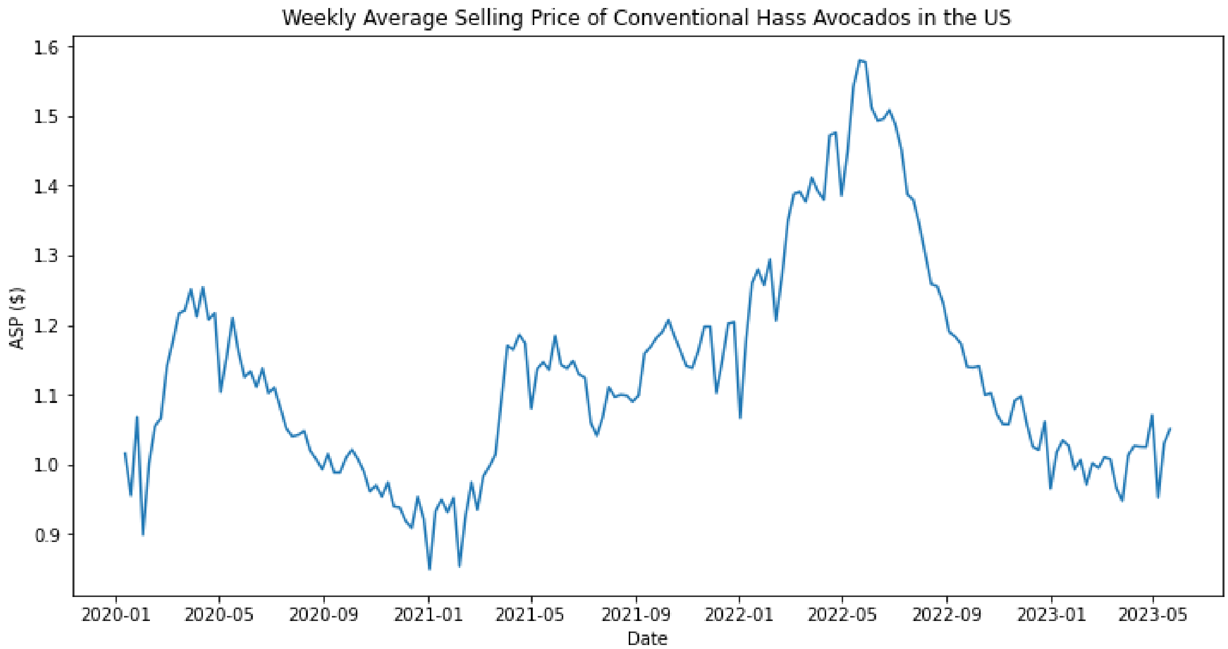
```
[<matplotlib.lines.Line2D at 0x1aea7a99fd0>]
```



In [16]:

```
#Plot conv data

plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.title('Weekly Average Selling Price of Conventional Hass Avocados in the US')
plt.plot(idata_conv)
```

Out[16]: [<matplotlib.lines.Line2D at 0x1aea9f01430>]



In [17]:

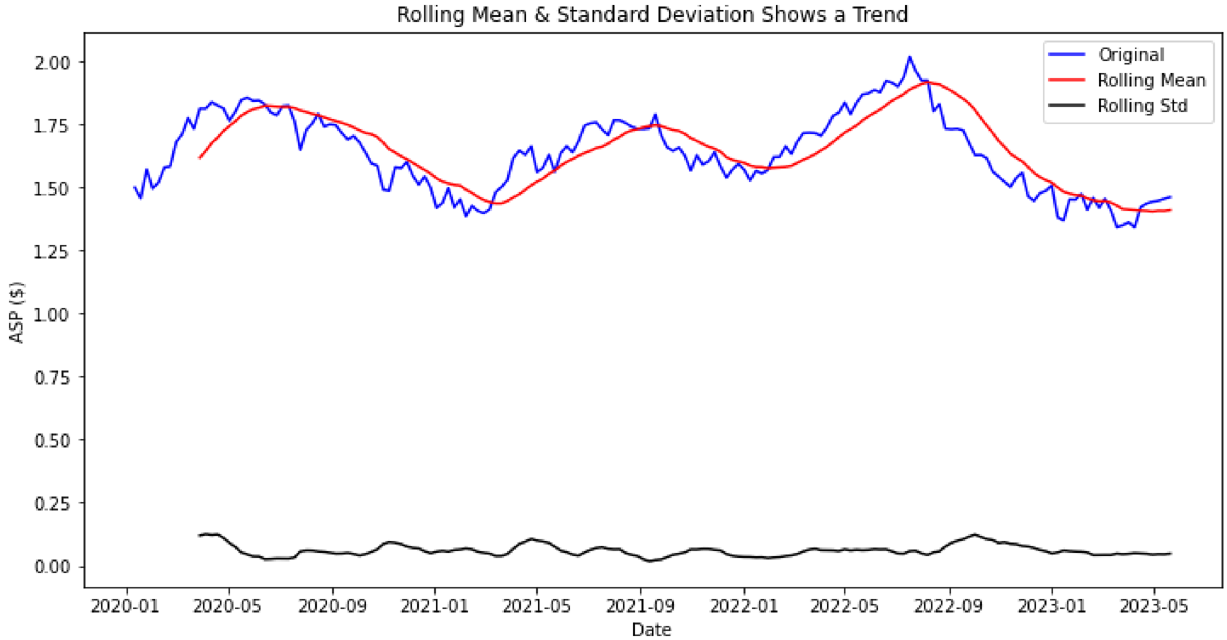
```
#Create rolling mean and std. dev

o_rolmean = idata_org.rolling(window=12).mean()
o_rolstd = idata_org.rolling(window=12).std()
print(o_rolmean, o_rolstd)
```

```
                 ASP
Date
2020-01-12       NaN
2020-01-19       NaN
2020-01-26       NaN
2020-02-02       NaN
2020-02-09       NaN
...              ...
2023-04-23  1.405907
2023-04-30  1.403278
2023-05-07  1.406257
2023-05-14  1.405877
2023-05-21  1.409376

[176 rows x 1 columns]                   ASP
Date
2020-01-12       NaN
2020-01-19       NaN
2020-01-26       NaN
2020-02-02       NaN
2020-02-09       NaN
...              ...
2023-04-23  0.047135
2023-04-30  0.043771
2023-05-07  0.045470
2023-05-14  0.045012
2023-05-21  0.047646

[176 rows x 1 columns]
```

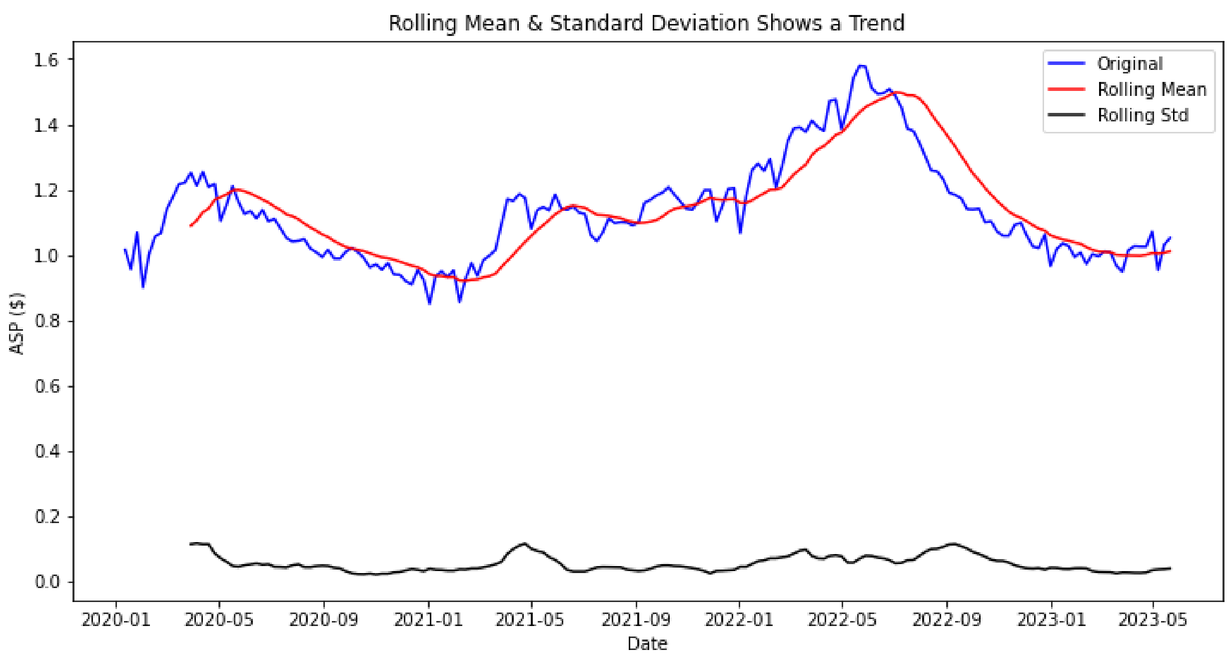
In [18]:
```
#Create rolling mean and std. dev

c_rolmean = idata_conv.rolling(window=12).mean()
c_rolstd = idata_conv.rolling(window=12).std()
print(c_rolmean, c_rolstd)
```

```
                 ASP
Date
2020-01-12       NaN
2020-01-19       NaN
2020-01-26       NaN
2020-02-02       NaN
2020-02-09       NaN
...              ...
2023-04-23  0.999585
2023-04-30  1.004930
2023-05-07  1.003407
2023-05-14  1.005787
2023-05-21  1.010430

[176 rows x 1 columns]                   ASP
Date
2020-01-12       NaN
2020-01-19       NaN
2020-01-26       NaN
2020-02-02       NaN
2020-02-09       NaN
...              ...
2023-04-23  0.025336
2023-04-30  0.032613
2023-05-07  0.034709
2023-05-14  0.035547
2023-05-21  0.037573

[176 rows x 1 columns]
```

In [19]:
```
#Plot organic rolling mean and rolling standard deviation

o_orig = plt.plot(idata_org, color='blue', label='Original')
o_mean = plt.plot(o_rolmean, color='red', label='Rolling Mean')
o_std = plt.plot(o_rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.title('Rolling Mean & Standard Deviation Shows a Trend')
plt.show(block=False)
```



In [20]:
```
#Plot conventional rolling mean and rolling standard deviation

c_orig = plt.plot(idata_conv, color='blue', label='Original')
c_mean = plt.plot(c_rolmean, color='red', label='Rolling Mean')
c_std = plt.plot(c_rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation Shows a Trend')
plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.show(block=False)
```



In [21]:
```
#Organic data Dickey-Fuller test; p-value is greater than 0.05, so data is not statior

o_datatest = adfuller(idata_org['ASP'], autolag='AIC')
o_dataoutput = pd.Series(o_datatest[0:4], index=['Test Statistic','p-value','#Lags Use

for key,value in o_datatest[4].items():
```

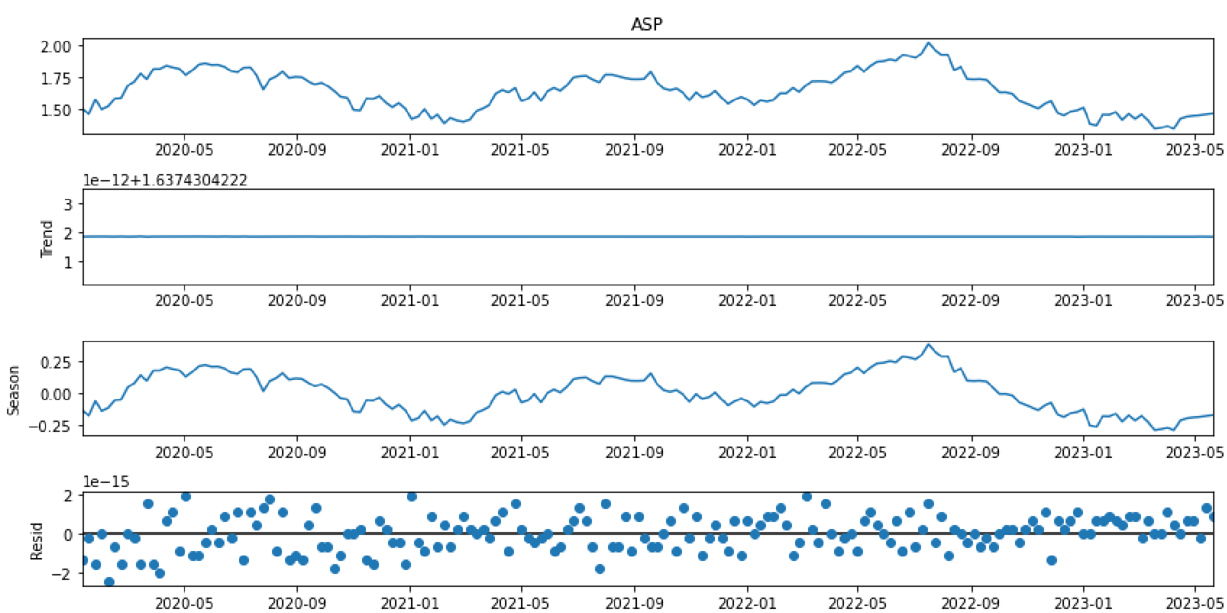
```
        o_dataoutput['Critical Value (%s) '%key] = value

print(o_dataoutput)
```

```
Test Statistic           -1.895609
p-value                   0.334130
#Lags Used                0.000000
No. of Observations     175.000000
Critical Value (1%)      -3.468280
Critical Value (5%)      -2.878202
Critical Value (10%)     -2.575653
dtype: float64
```

In [22]:
```
#Conventional data Dickey-Fuller test; p-value is greater than 0.05, so data is not st

c_datatest = adfuller(idata_conv['ASP'], autolag='AIC')
c_dataoutput = pd.Series(c_datatest[0:4], index=['Test Statistic','p-value','#Lags Use

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

print(c_dataoutput)
```

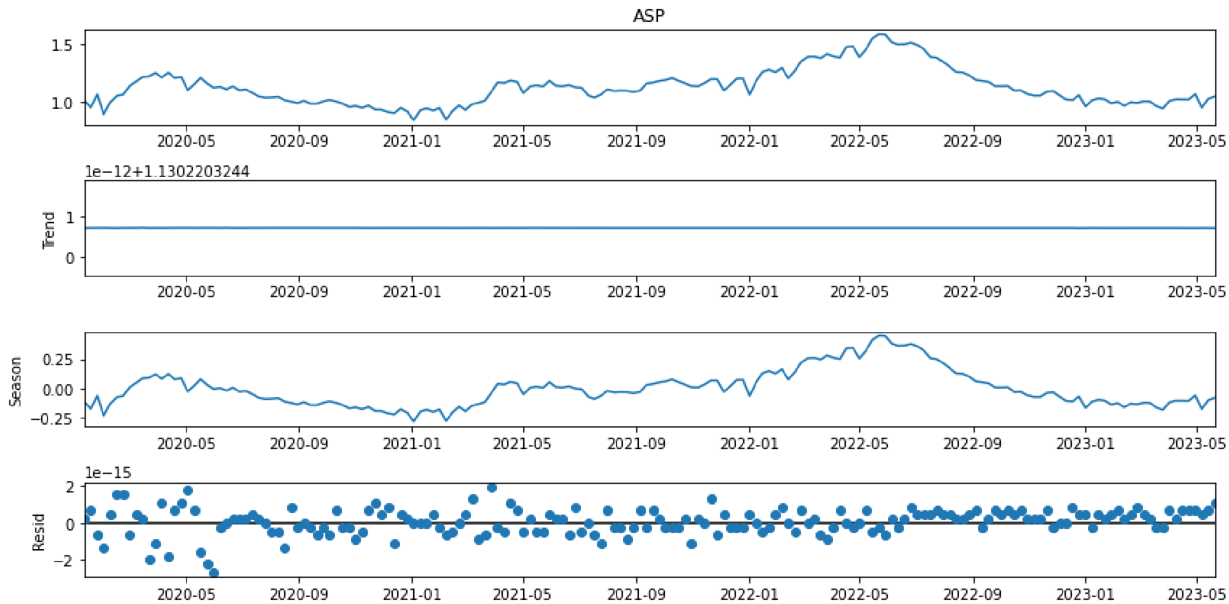
```
Test Statistic           -2.087910
p-value                   0.249404
#Lags Used                6.000000
No. of Observations     169.000000
Critical Value (1%)      -3.469648
Critical Value (5%)      -2.878799
Critical Value (10%)     -2.575971
dtype: float64
```

In [23]:
```
#Decomposing time series data (organic indexed data)

o_stl = STL(idata_org['ASP'],period = 180)
o_res=o_stl.fit()
o_fig=o_res.plot()
```

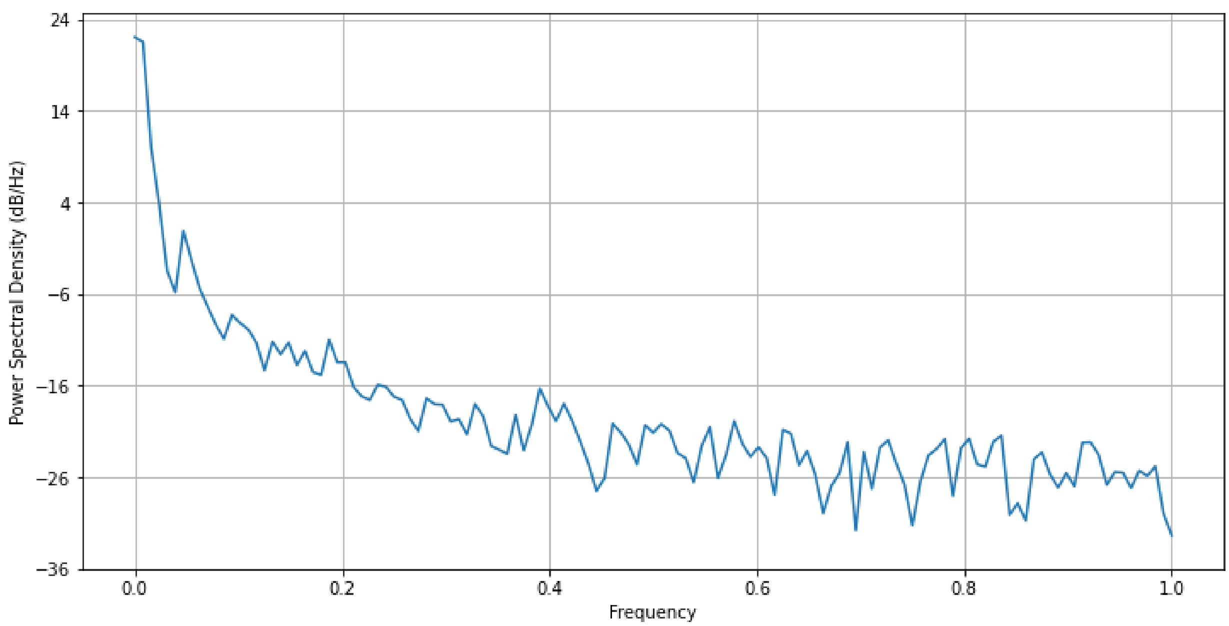


In [24]:
```
#Decomposing time series data (conventional indexed data)

c_stl = STL(idata_conv['ASP'],period = 180)
```

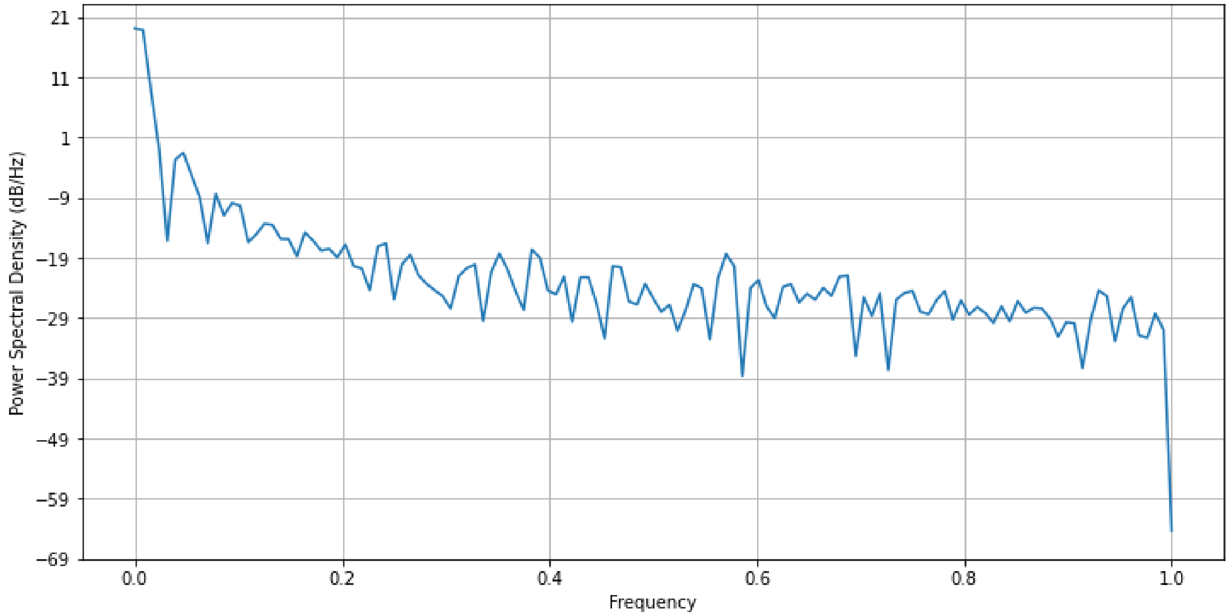
```
c_res=c_stl.fit()
c_fig=c_res.plot()
```



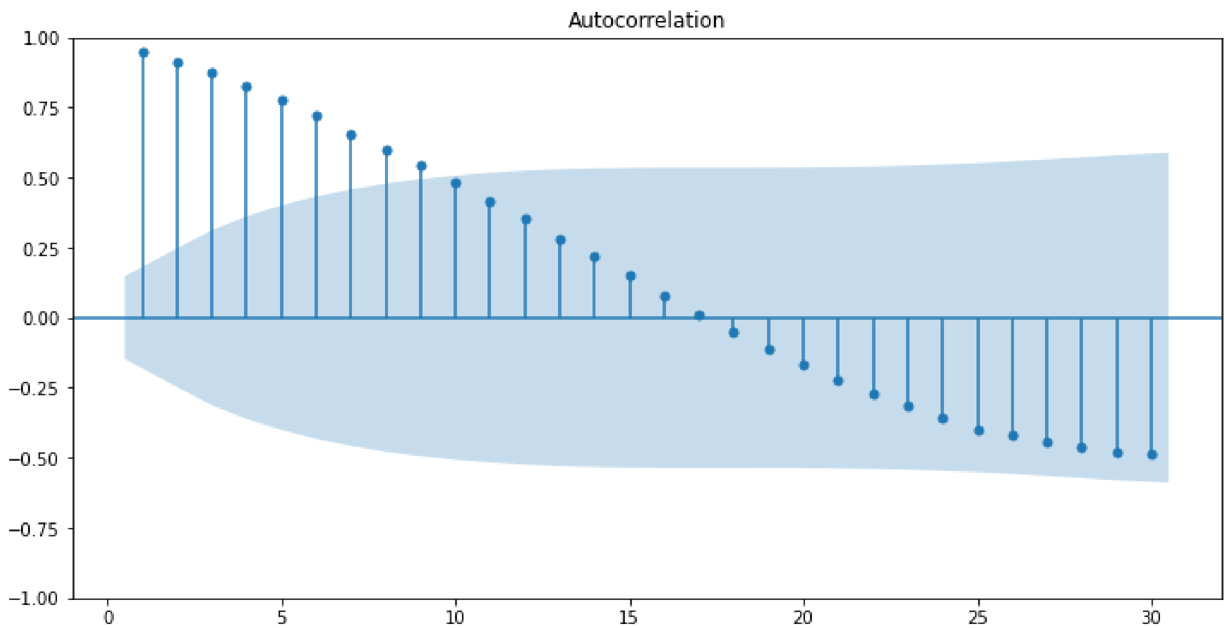
In [25]:
```
#Spectral Density (organic indexed data)

plt.psd(idata_org['ASP'])
plt.show()
```

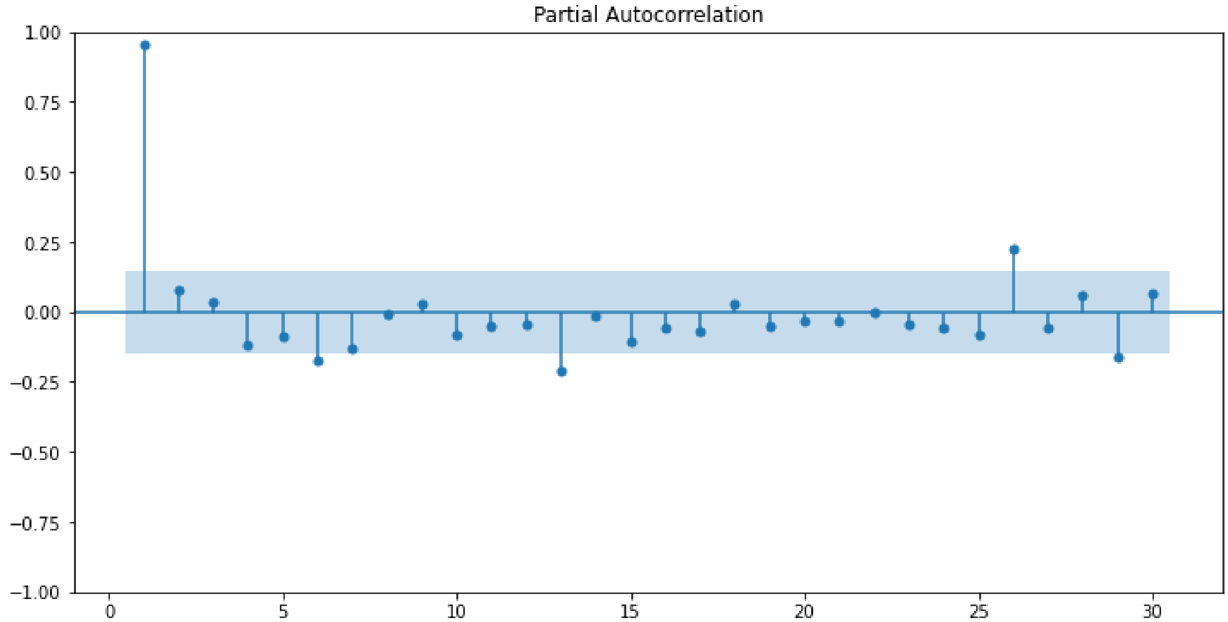


In [26]:
```
#Spectral Density (conventional indexed data)

plt.psd(idata_conv['ASP'])
plt.show()
```

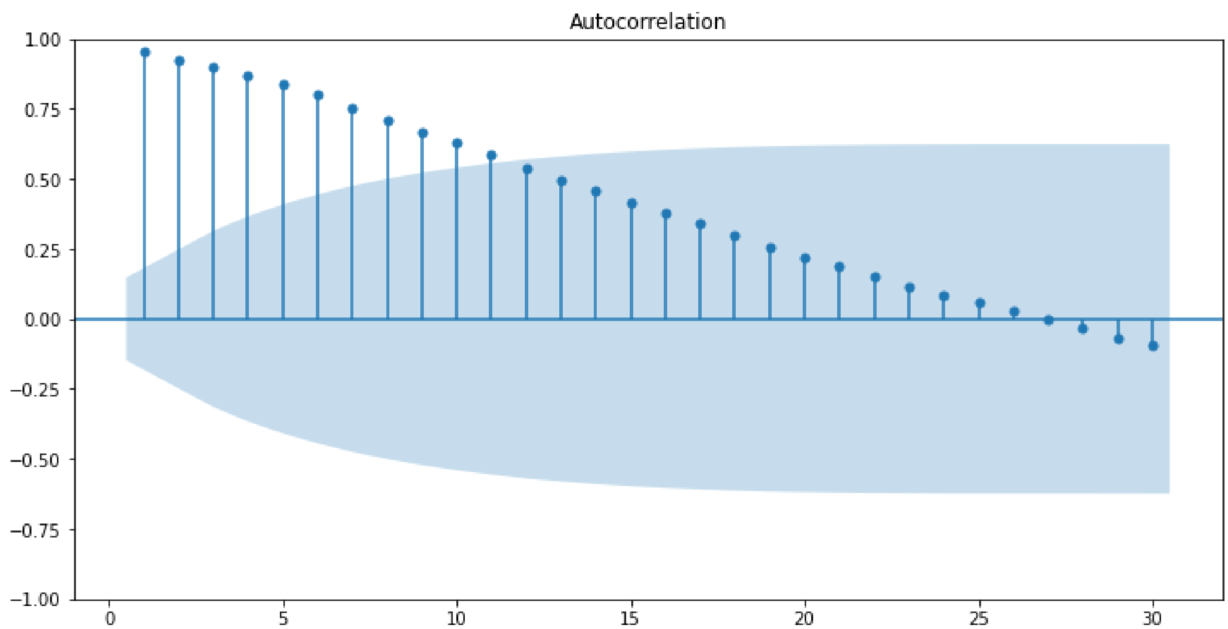


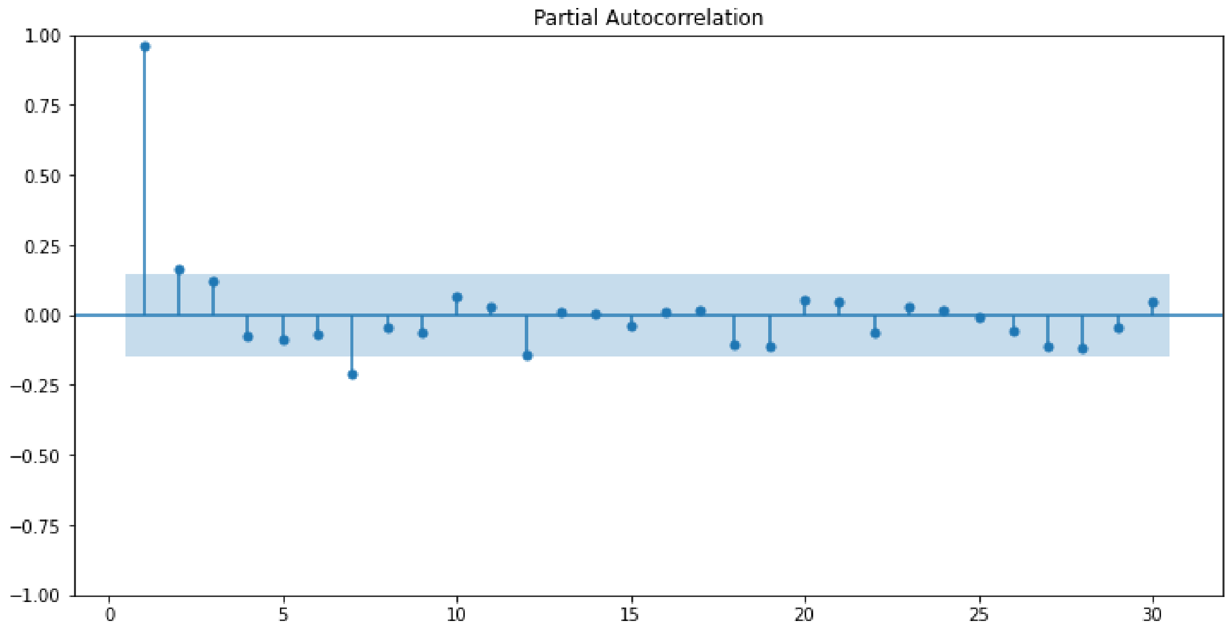
In [27]:
```
#Organic data acf and pacf

plot_acf(idata_org, lags=30, zero=False)
plot_pacf(idata_org, lags=30, zero=False)
plt.show()
```





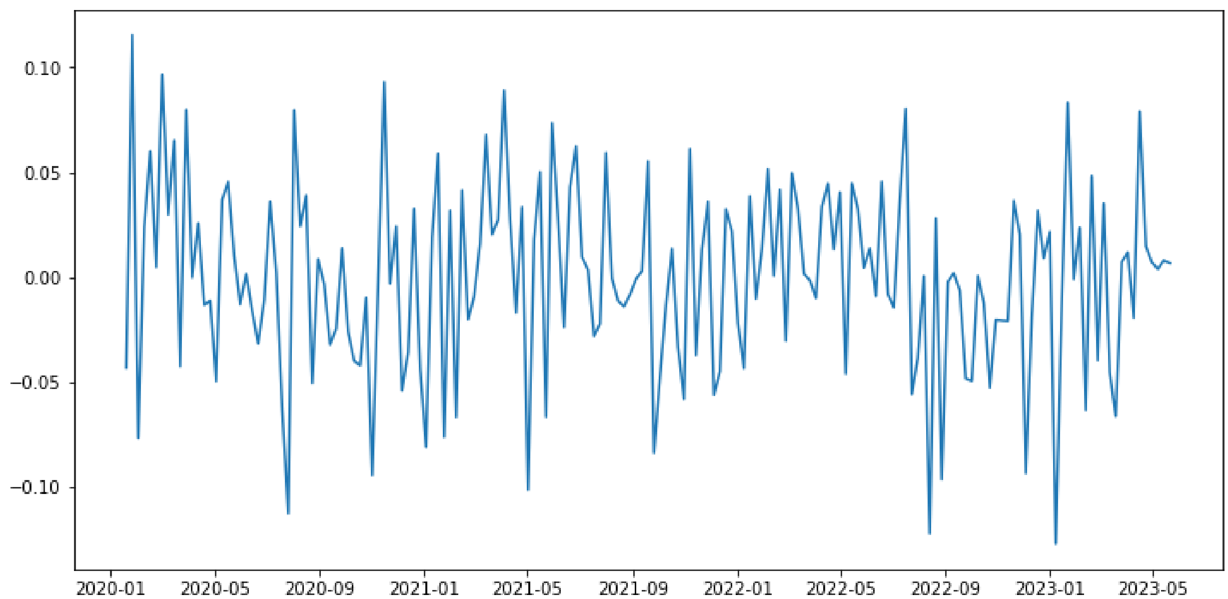
In [28]:
```
#Conventional data acf and pacf

plot_acf(idata_conv, lags=30, zero=False)
plot_pacf(idata_conv, lags=30, zero=False)
plt.show()
```





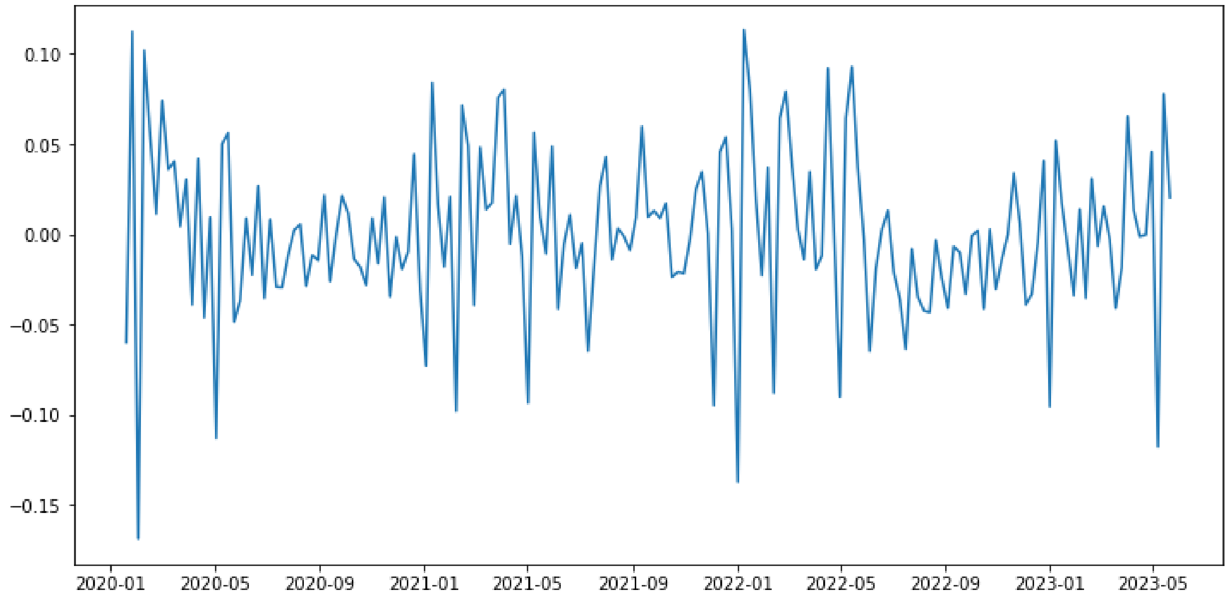
In [29]:
```
#Organic
#Differencing at a shift of 1

o_diffdata=idata_org.diff()
o_diffdata=o_diffdata.dropna()
plt.plot(o_diffdata)
```

Out[29]: [<matplotlib.lines.Line2D at 0x1aeaa4953a0>]



In [30]:
```
#Conventional
#Differencing at a shift of 1

c_diffdata=idata_conv.diff()
c_diffdata=c_diffdata.dropna()
plt.plot(c_diffdata)
```

Out[30]: [<matplotlib.lines.Line2D at 0x1aeaa731460>]



In [31]:
```
#Organic dickey-Fuller test; p-value is less than 0.05, so data stationary

od_datatest = adfuller(o_diffdata['ASP'], autolag='AIC')
od_dataoutput = pd.Series(od_datatest[0:4], index=['Test Statistic','p-value','#Lags U

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

print(od_dataoutput)
```

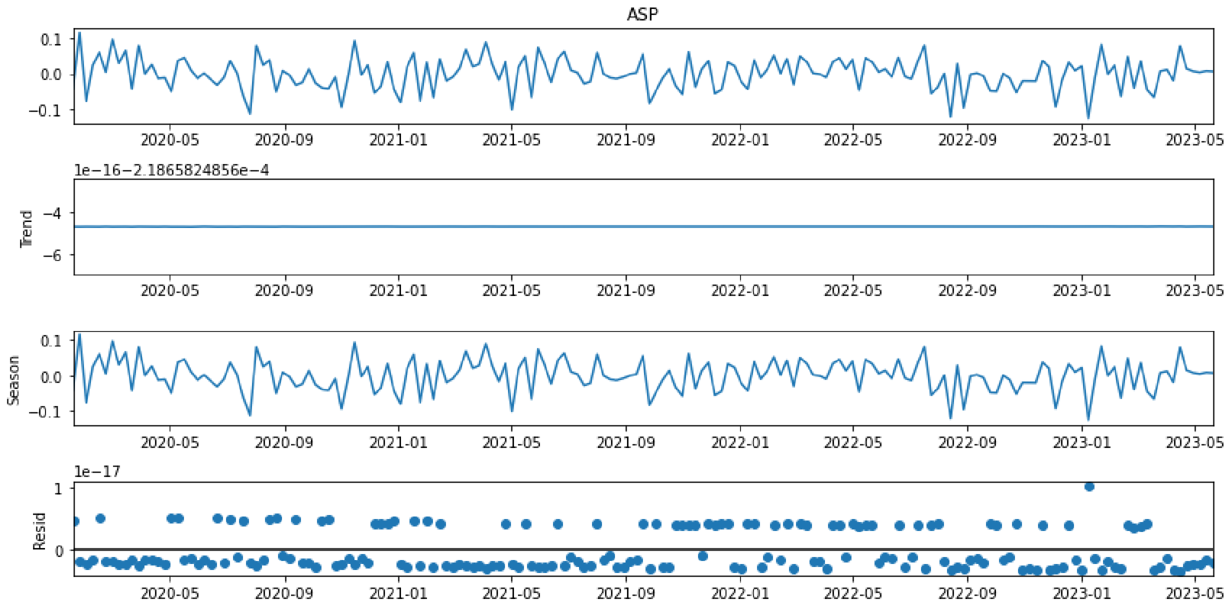
```
Test Statistic          -1.487869e+01
p-value                  1.621780e-27
#Lags Used               0.000000e+00
No. of Observations      1.740000e+02
Critical Value (1%)     -3.468502e+00
Critical Value (5%)     -2.878298e+00
Critical Value (10%)    -2.575704e+00
dtype: float64
```

In [32]:
```
#Conventional dickey-Fuller test; p-value is less than 0.05, so data stationary

cd_datatest = adfuller(c_diffdata['ASP'], autolag='AIC')
cd_dataoutput = pd.Series(cd_datatest[0:4], index=['Test Statistic','p-value','#Lags U

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

print(cd_dataoutput)
```

```
Test Statistic            -3.995663
p-value                    0.001433
#Lags Used                 5.000000
No. of Observations      169.000000
Critical Value (1%)       -3.469648
Critical Value (5%)       -2.878799
Critical Value (10%)      -2.575971
dtype: float64
```

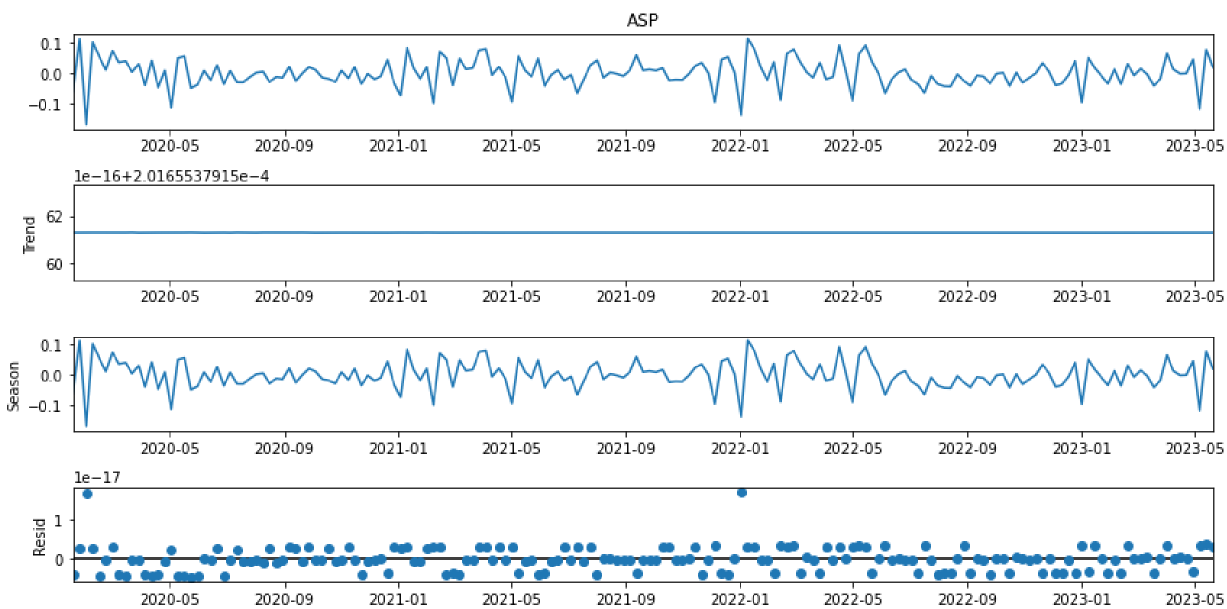
In [33]:
```
#Decomposing time series data (organic differenced data)
```

```
od_stl = STL(o_diffdata['ASP'],period = 180)
od_res=od_stl.fit()
od_fig=od_res.plot()
```



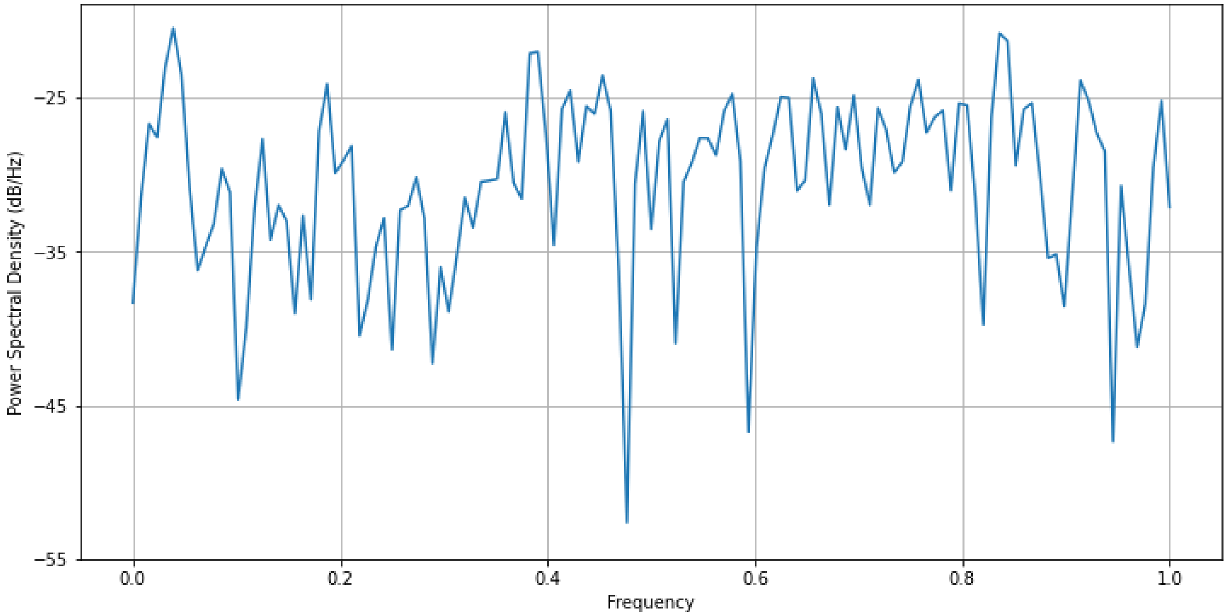
In [34]:

```
#Decomposing time series data (conventional differenced data)

cd_stl = STL(c_diffdata['ASP'],period = 180)
cd_res=cd_stl.fit()
cd_fig=cd_res.plot()
```

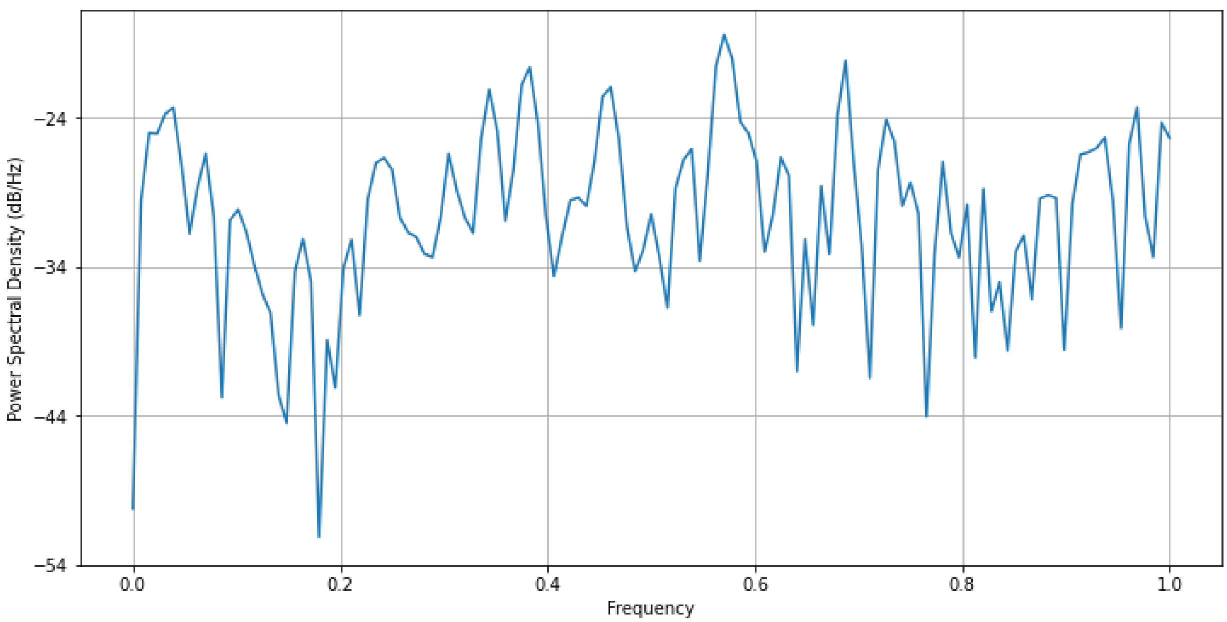


In [35]:

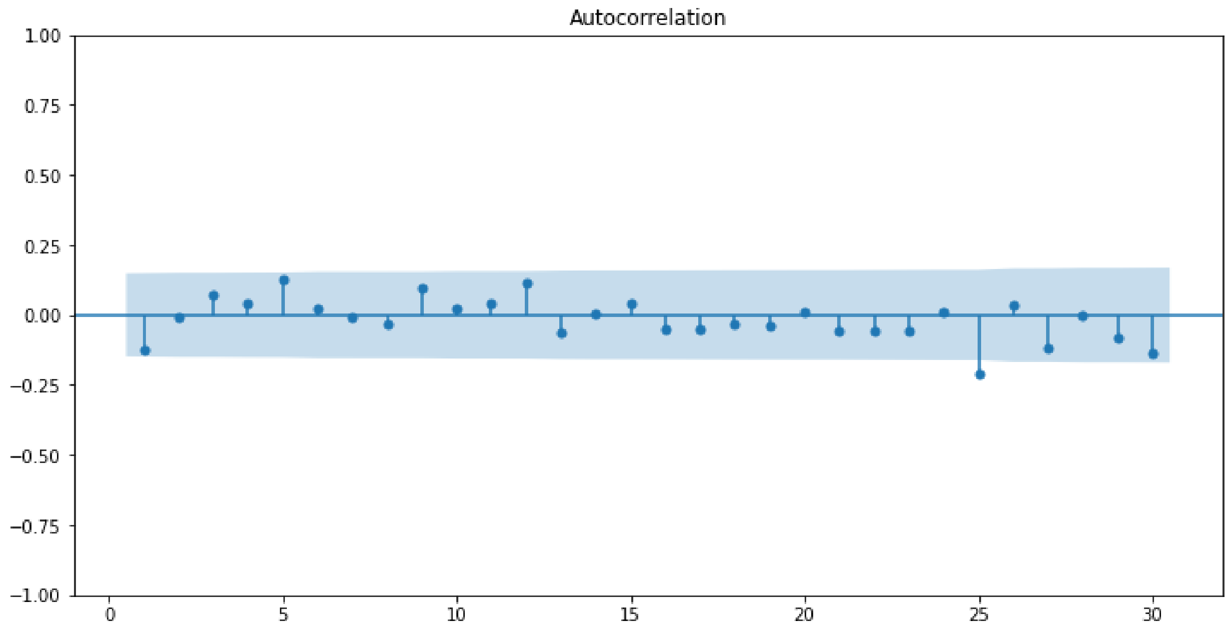
```
#Spectral Density (organic differenced data)

plt.psd(o_diffdata['ASP'])
plt.show()
```

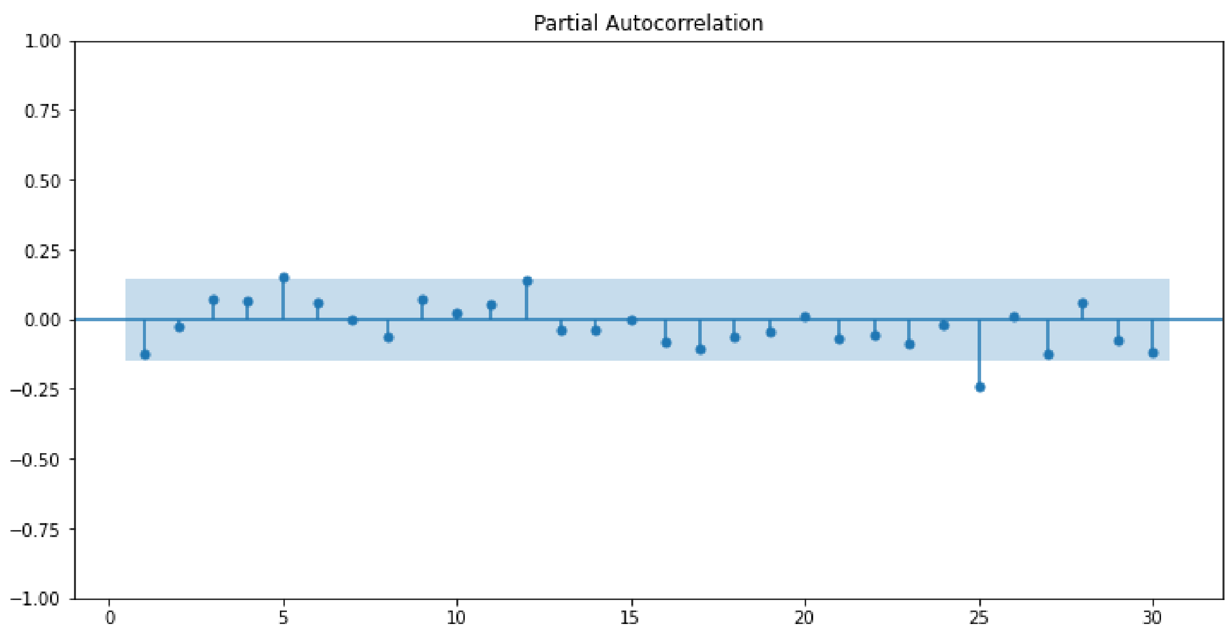


In [36]:
```
#Spectral Density (conventional differenced data)

plt.psd(c_diffdata['ASP'])
plt.show()
```



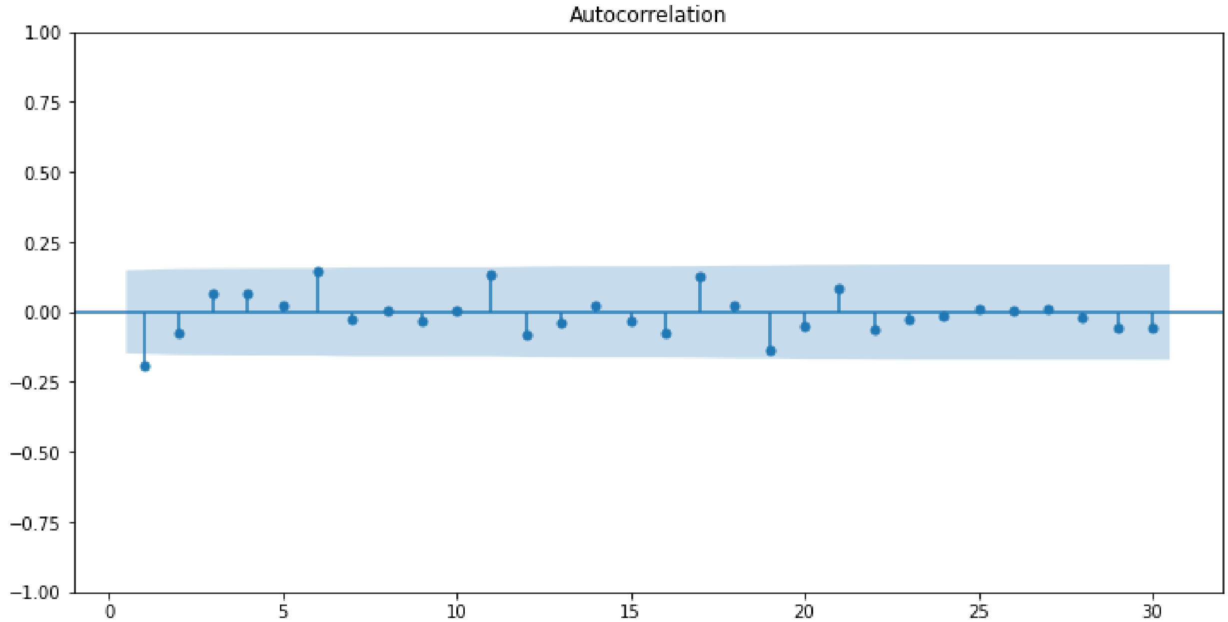
In [37]:
```
#Organic ACF and PACF

plot_acf(o_diffdata,lags=30, zero=False)
plot_pacf(o_diffdata,lags=30, zero=False)
plt.show()
```

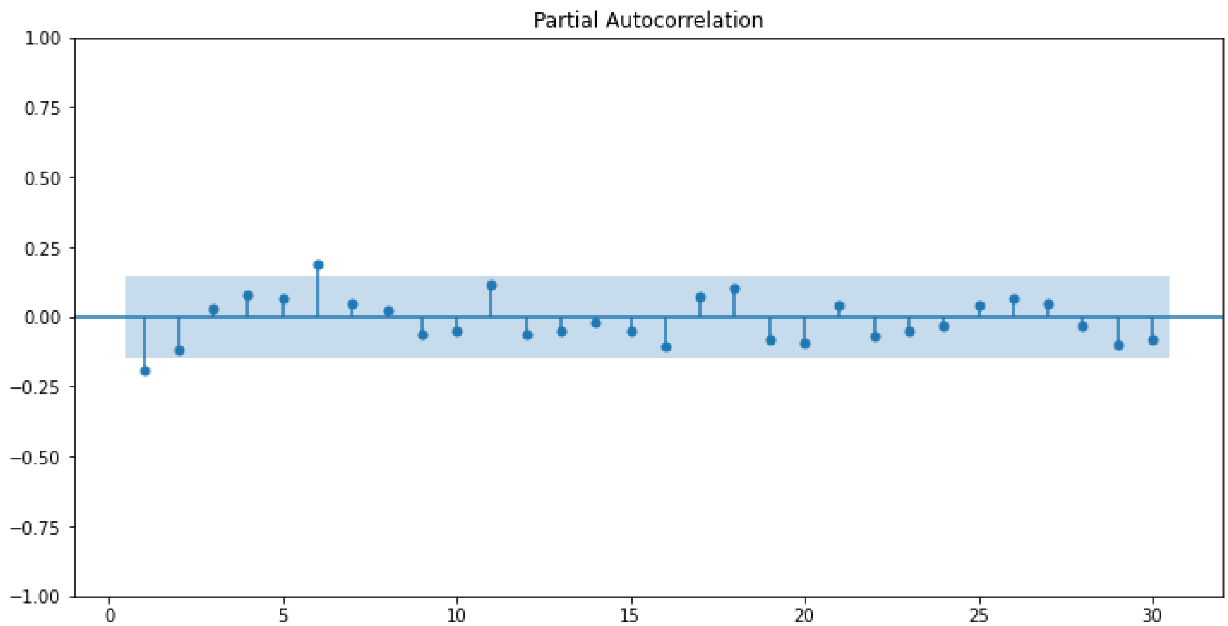




In [38]:

```
#Conventional ACF and PACF

plot_acf(c_diffdata,lags=30, zero=False)
plot_pacf(c_diffdata,lags=30, zero=False)
plt.show()
```





In [39]:
```
#Train test split 20% of organic data

o_train = idata_org.iloc[:-36]
o_test = idata_org.iloc[-36:]
print(o_train.shape, o_test.shape)
```

```
(140, 1) (36, 1)
```

In [40]:
```
#Train test split 20% of organic data

c_train = idata_conv.iloc[:-36]
c_test = idata_conv.iloc[-36:]
print(c_train.shape, c_test.shape)
```

```
(140, 1) (36, 1)
```

In [41]:
```
#Organic
#Finding p,d,q values from Python, second set of parenthesis shows no seasonality
```

```
stepwise_fit = auto_arima(o_train['ASP'], trace = True, suppress_warnings=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=-462.568, Time=0.18 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=-154.978, Time=0.03 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=-464.374, Time=0.04 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=-285.518, Time=0.04 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=545.252, Time=0.01 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=-464.124, Time=0.11 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=-463.962, Time=0.11 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-462.143, Time=0.08 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.05 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.651 seconds
```

Out[41]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	140
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	235.187
Date:	Sun, 30 Jul 2023	**AIC**	-464.374
Time:	21:02:39	**BIC**	-455.549
Sample:	01-12-2020	**HQIC**	-460.788
	- 09-11-2022		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	0.1000	0.044	2.269	0.023	0.014	0.186
ar.L1	0.9400	0.027	35.364	0.000	0.888	0.992
sigma2	0.0020	0.000	7.759	0.000	0.001	0.003

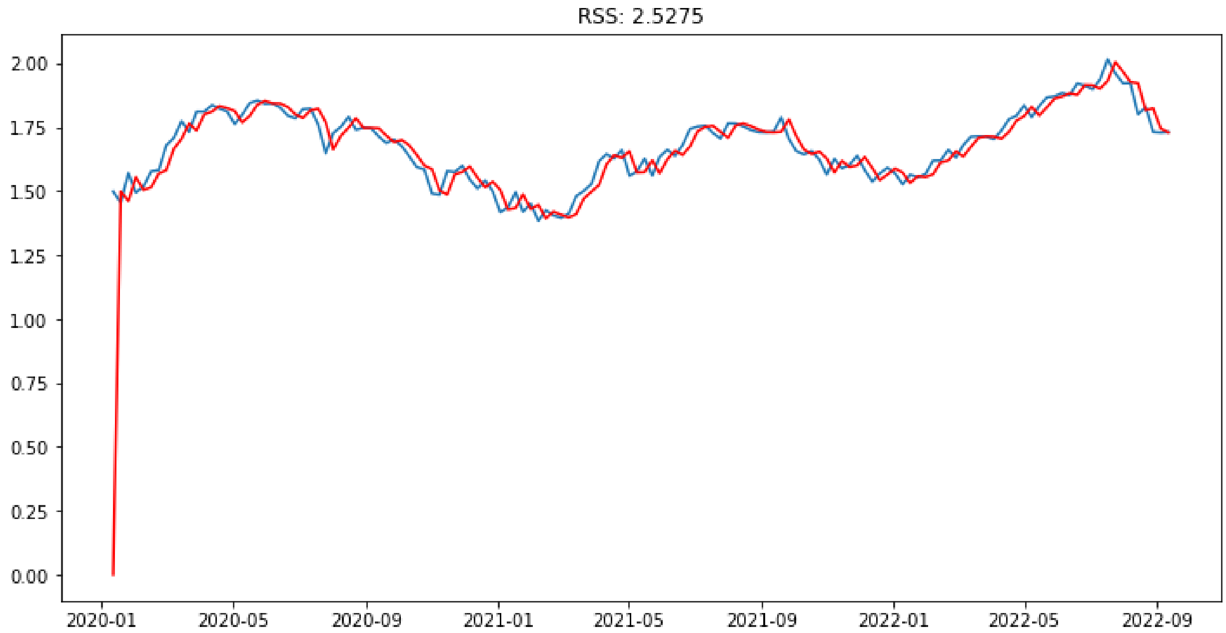
Ljung-Box (L1) (Q):	1.35	**Jarque-Bera (JB):**	2.12
Prob(Q):	0.25	**Prob(JB):**	0.35
Heteroskedasticity (H):	0.75	**Skew:**	-0.29
Prob(H) (two-sided):	0.32	**Kurtosis:**	2.82

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

In [42]:

```
#Arima based on auto arima and d of 1

o_model = ARIMA(o_train['ASP'], order = (1,1,0))
o_results_ARIMA = o_model.fit()
plt.plot(o_train)
plt.plot(o_results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((o_results_ARIMA.fittedvalues-o_train['ASP'])**2))
print('Plotting ARIMA Model')
```

Plotting ARIMA Model

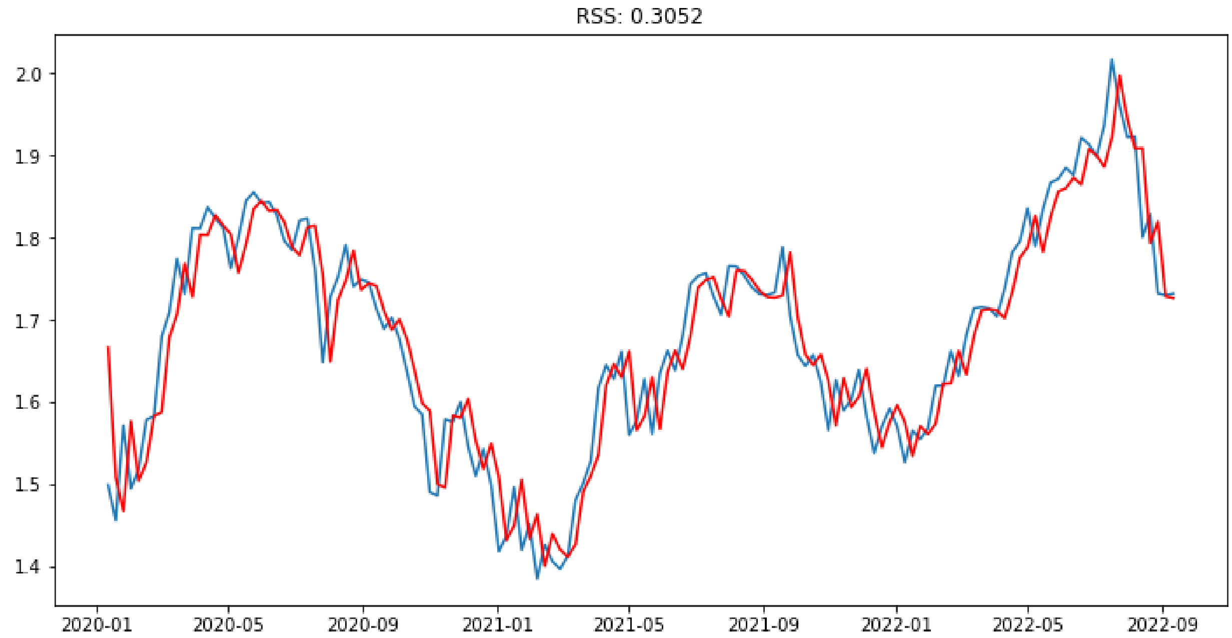


In [43]:
```
#Arima 1 summary
#p stat greater than 0, so we'll take away the d value

print(o_results_ARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    ASP   No. Observations:                  140
Model:                 ARIMA(1, 1, 0)   Log Likelihood                 233.955
Date:                Sun, 30 Jul 2023   AIC                           -463.911
Time:                        21:02:39   BIC                           -458.042
Sample:                    01-12-2020   HQIC                          -461.526
                         - 09-11-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.1362      0.088     -1.555      0.120      -0.308       0.035
sigma2         0.0020      0.000      8.189      0.000       0.002       0.003
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 1.36
Prob(Q):                              0.99   Prob(JB):                         0.51
Heteroskedasticity (H):               0.72   Skew:                            -0.24
Prob(H) (two-sided):                  0.27   Kurtosis:                         2.97
===================================================================================

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

In [44]:
```
#Arima based on auto arima

o_model2 = ARIMA(o_train['ASP'], order = (1,0,0))
o_results_ARIMA2 = o_model2.fit()
plt.plot(o_train)
plt.plot(o_results_ARIMA2.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((o_results_ARIMA2.fittedvalues-o_train['ASP'])**2))
print('Plotting ARIMA Model')
```

Plotting ARIMA Model



In [45]:
```
#Arima 2 summary
#p less than 0.05. RSS the lowest. AICs are close but this AIC is just a tad higher

print(o_results_ARIMA2.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    ASP   No. Observations:                  140
Model:                 ARIMA(1, 0, 0)   Log Likelihood                 235.204
Date:                Sun, 30 Jul 2023   AIC                           -464.408
Time:                        21:02:39   BIC                           -455.583
Sample:                    01-12-2020   HQIC                          -460.822
                         - 09-11-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.6661      0.060     27.986      0.000       1.549       1.783
ar.L1          0.9449      0.027     35.499      0.000       0.893       0.997
sigma2         0.0020      0.000      7.768      0.000       0.001       0.003
===================================================================================
Ljung-Box (L1) (Q):                   1.51   Jarque-Bera (JB):                 2.03
Prob(Q):                              0.22   Prob(JB):                         0.36
Heteroskedasticity (H):               0.75   Skew:                            -0.28
Prob(H) (two-sided):                  0.32   Kurtosis:                         2.83
===================================================================================

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

In [46]:
```
#Conventional
#Finding p,d,q values from Python, second set of parenthesis shows no seasonality

stepwise_fit = auto_arima(c_train['ASP'], trace = True, suppress_warnings=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-446.566, Time=0.23 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-448.958, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-450.649, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-451.367, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-450.868, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-449.511, Time=0.08 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-449.671, Time=0.10 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-448.068, Time=0.12 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=-453.206, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-451.342, Time=0.05 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=-451.497, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=-452.510, Time=0.02 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-449.900, Time=0.09 sec

Best model:  ARIMA(0,1,1)(0,0,0)[0]
Total fit time: 0.923 seconds
```

Out[46]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	140
Model:	SARIMAX(0, 1, 1)	**Log Likelihood**	228.603
Date:	Sun, 30 Jul 2023	**AIC**	-453.206
Time:	21:02:40	**BIC**	-447.338
Sample:	01-12-2020	**HQIC**	-450.821
	- 09-11-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.1873	0.074	-2.537	0.011	-0.332	-0.043
sigma2	0.0022	0.000	9.025	0.000	0.002	0.003

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	4.18
Prob(Q):	0.93	**Prob(JB):**	0.12
Heteroskedasticity (H):	1.38	**Skew:**	-0.31
Prob(H) (two-sided):	0.28	**Kurtosis:**	3.58

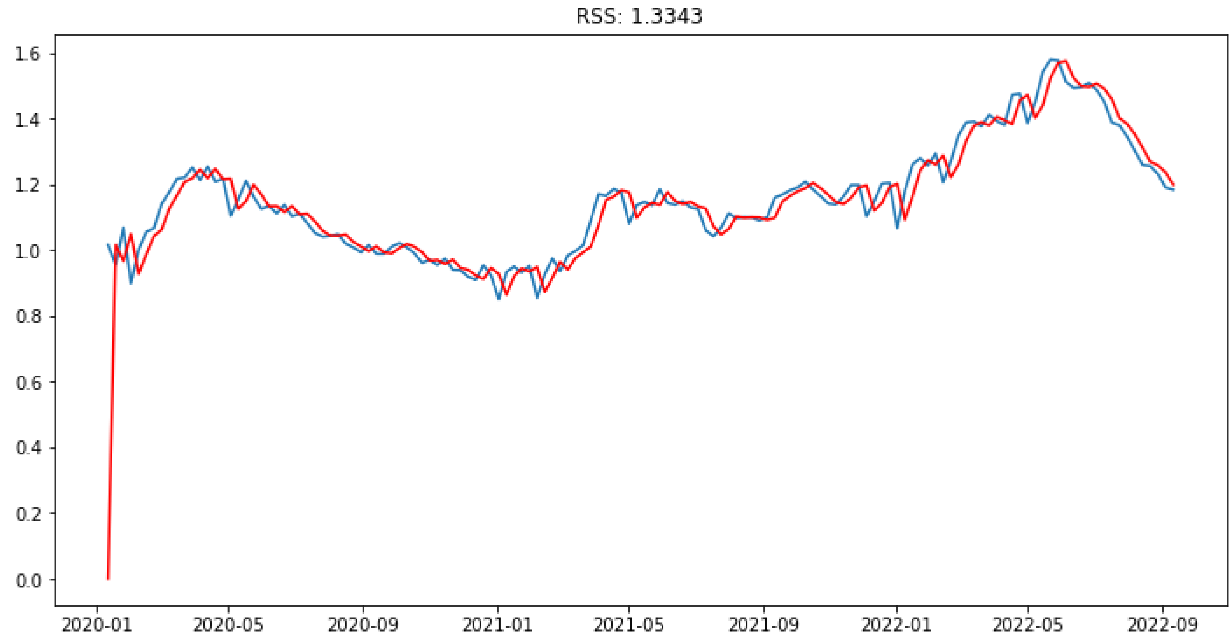
Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [47]:

```
#Arima based on auto arima

c_model = ARIMA(c_train['ASP'], order = (0,1,1))
c_results_ARIMA = c_model.fit()
plt.plot(c_train)
plt.plot(c_results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((c_results_ARIMA.fittedvalues-c_train['ASP'])**2))
print('Plotting ARIMA Model')
```

Plotting ARIMA Model



In [48]:

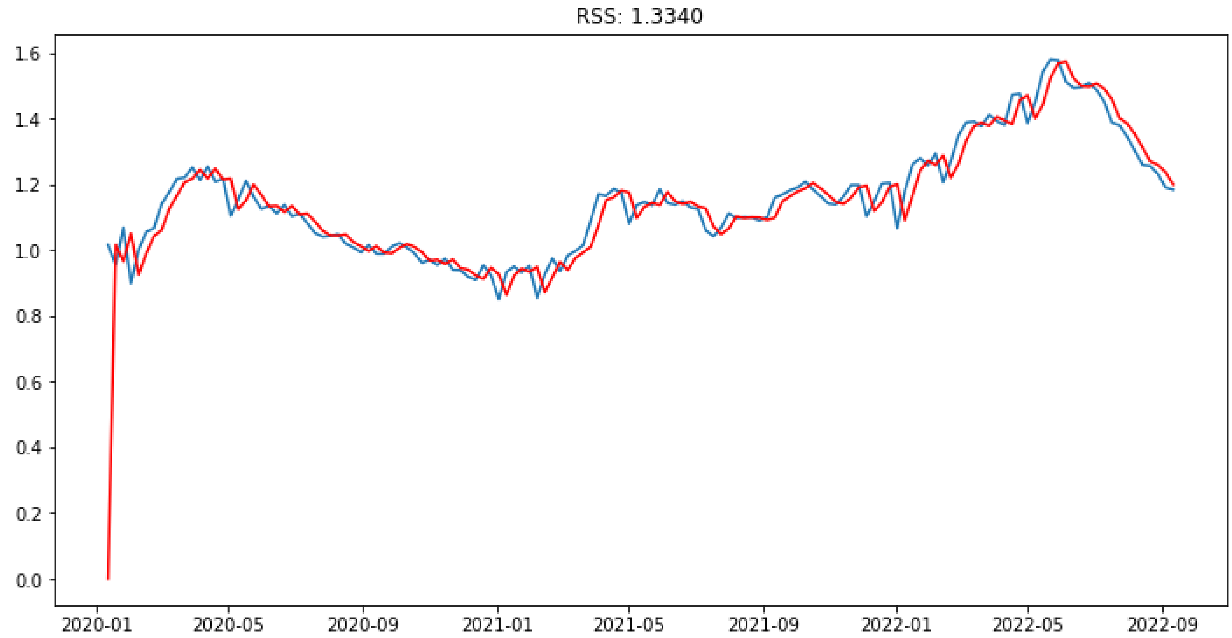
```
#Arima 1 summary
#p less than 0.05

print(c_results_ARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    ASP   No. Observations:                  140
Model:                 ARIMA(0, 1, 1)   Log Likelihood                 228.603
Date:                Sun, 30 Jul 2023   AIC                           -453.206
Time:                        21:02:41   BIC                           -447.338
Sample:                    01-12-2020   HQIC                          -450.821
                         - 09-11-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.1873      0.074     -2.537      0.011      -0.332      -0.043
sigma2         0.0022      0.000      9.025      0.000       0.002       0.003
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                 4.18
Prob(Q):                              0.93   Prob(JB):                         0.12
Heteroskedasticity (H):               1.38   Skew:                            -0.31
Prob(H) (two-sided):                  0.28   Kurtosis:                         3.58
===================================================================================

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

In [49]:

```
#Arima based on auto arima and 1 in p_value

c_model2 = ARIMA(c_train['ASP'], order = (1,1,1))
c_results_ARIMA2 = c_model2.fit()
plt.plot(c_train)
plt.plot(c_results_ARIMA2.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((c_results_ARIMA2.fittedvalues-c_train['ASP'])**2))
print('Plotting ARIMA Model')
```

Plotting ARIMA Model



In [50]:
```
#Arima 2 summary
#p values greater than 0.05

print(c_results_ARIMA2.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    ASP   No. Observations:                  140
Model:                 ARIMA(1, 1, 1)   Log Likelihood                 228.671
Date:                Sun, 30 Jul 2023   AIC                           -451.342
Time:                        21:02:41   BIC                           -442.539
Sample:                    01-12-2020   HQIC                          -447.765
                         - 09-11-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1068      0.402      0.266      0.790      -0.681       0.894
ma.L1         -0.2866      0.420     -0.683      0.495      -1.109       0.536
sigma2         0.0022      0.000      8.840      0.000       0.002       0.003
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 4.37
Prob(Q):                              0.98   Prob(JB):                         0.11
Heteroskedasticity (H):               1.35   Skew:                            -0.32
Prob(H) (two-sided):                  0.31   Kurtosis:                         3.59
===================================================================================

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

In [51]:
```
#The following cells will be for the organic data until specified otherise.
```

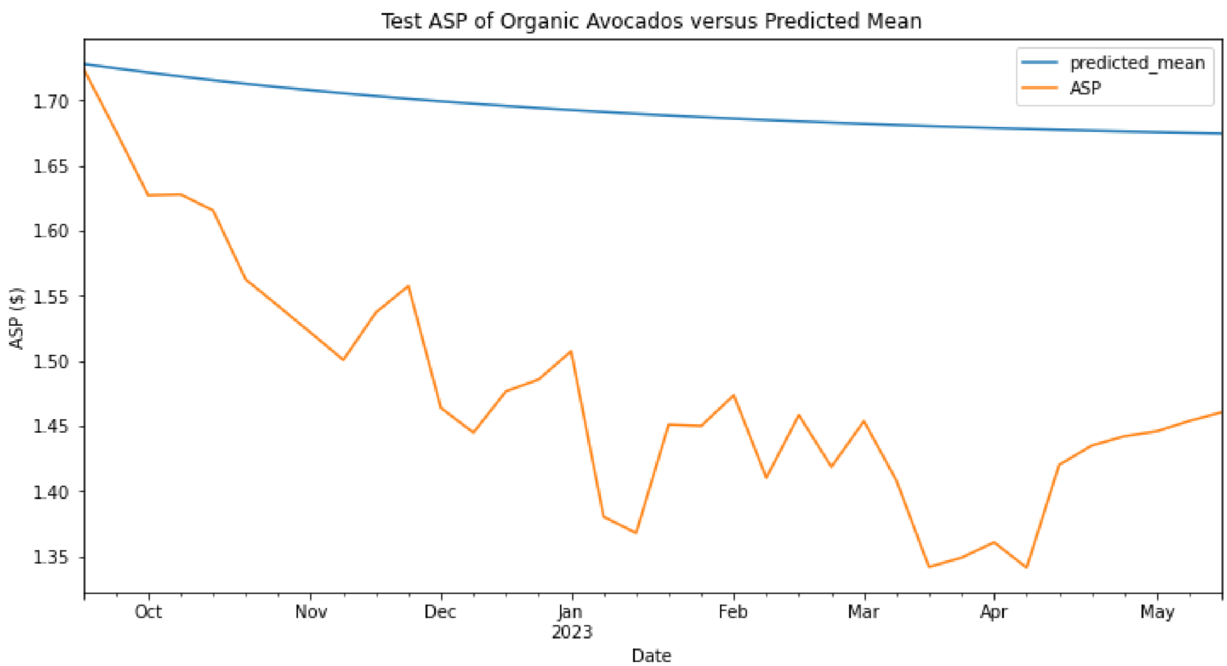
In [52]:
```
#Test data, printing predicted mean

o_start = len(o_train)
o_end=len(o_train)+len(o_test)-1
o_pred=o_results_ARIMA2.predict(start=o_start, end=o_end, type='levels')
```

```
print(o_pred.head())
o_pred.index=idata_org.index[o_start:o_end+1]
```

```
2022-09-18    1.728087
2022-09-25    1.724668
2022-10-02    1.721438
2022-10-09    1.718386
2022-10-16    1.715502
Freq: W-SUN, Name: predicted_mean, dtype: float64
```

In [53]:
```
#Plotting test data against the predicted mean

plt.title('Test ASP of Organic Avocados versus Predicted Mean')
plt.ylabel('ASP ($)')
o_pred.plot(legend=True)
o_test['ASP'].plot(legend=True)
```

Out[53]:
```
<AxesSubplot:title={'center':'Test ASP of Organic Avocados versus Predicted Mean'}, x
label='Date', ylabel='ASP ($)'>
```



In [54]:
```
#Mean of test data

o_test['ASP'].mean()
```

Out[54]:
```
1.4776738226944444
```

In [55]:
```
#RMSE of test data; >10%

o_rmse = sqrt(mean_squared_error(o_pred,o_test['ASP']))
print(o_rmse)
```

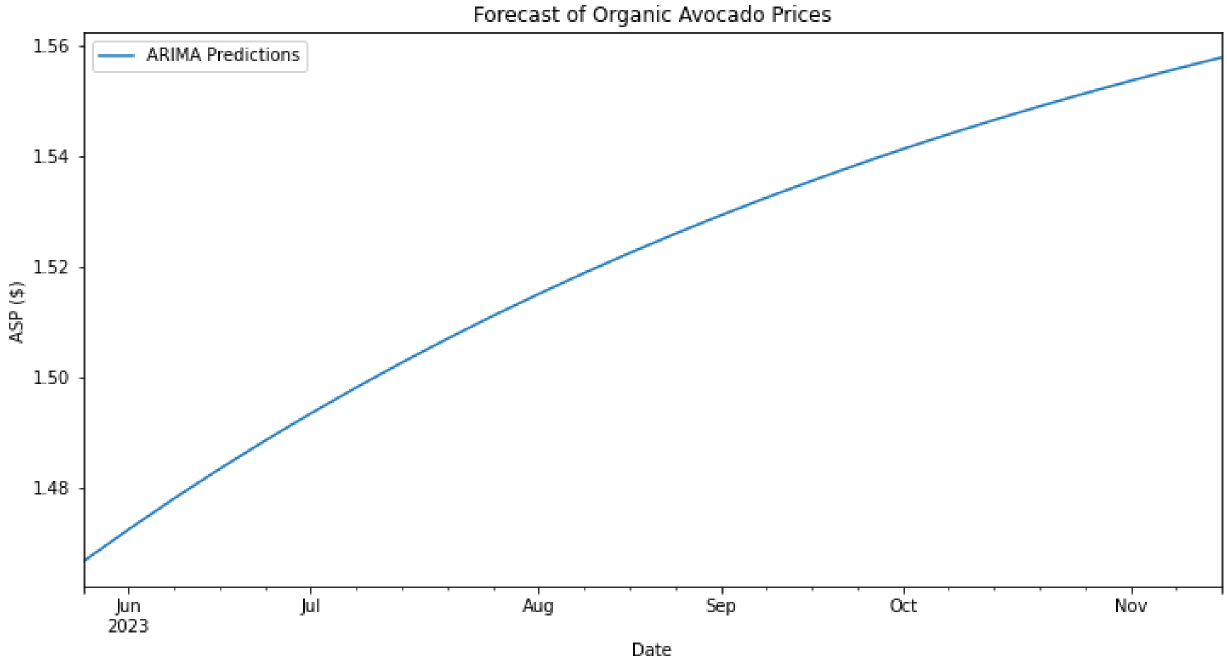
```
0.22945977570798895
```

In [56]:
```
#ARIMA on the data for forecasting

o_model2=ARIMA(idata_org, order=(1,0,0))
o_model2=o_model2.fit()
```

In [57]:
```
#Create an index of the future dates

o_index_future_dates = pd.date_range(start = '2023-05-28',end='2023-11-19', freq='W')
print(o_index_future_dates)
o_pred2=o_model2.predict(start=len(idata_org), end=len(idata_org)+25, typ='levels').re
o_pred2.index=o_index_future_dates
print(o_pred2)
plt.show()
```

```
DatetimeIndex(['2023-05-28', '2023-06-04', '2023-06-11', '2023-06-18',
               '2023-06-25', '2023-07-02', '2023-07-09', '2023-07-16',
               '2023-07-23', '2023-07-30', '2023-08-06', '2023-08-13',
               '2023-08-20', '2023-08-27', '2023-09-03', '2023-09-10',
               '2023-09-17', '2023-09-24', '2023-10-01', '2023-10-08',
               '2023-10-15', '2023-10-22', '2023-10-29', '2023-11-05',
               '2023-11-12', '2023-11-19'],
              dtype='datetime64[ns]', freq='W-SUN')
2023-05-28    1.466683
2023-06-04    1.472516
2023-06-11    1.478104
2023-06-18    1.483456
2023-06-25    1.488583
2023-07-02    1.493494
2023-07-09    1.498197
2023-07-16    1.502702
2023-07-23    1.507018
2023-07-30    1.511151
2023-08-06    1.515110
2023-08-13    1.518902
2023-08-20    1.522535
2023-08-27    1.526014
2023-09-03    1.529347
2023-09-10    1.532539
2023-09-17    1.535596
2023-09-24    1.538525
2023-10-01    1.541330
2023-10-08    1.544017
2023-10-15    1.546591
2023-10-22    1.549056
2023-10-29    1.551418
2023-11-05    1.553679
2023-11-12    1.555846
2023-11-19    1.557921
Freq: W-SUN, Name: ARIMA Predictions, dtype: float64
```

In [58]:
```
#Plotting the forecast on its own.

o_pred2.plot(legend=True)
plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.title('Forecast of Organic Avocado Prices')
plt.show()
```

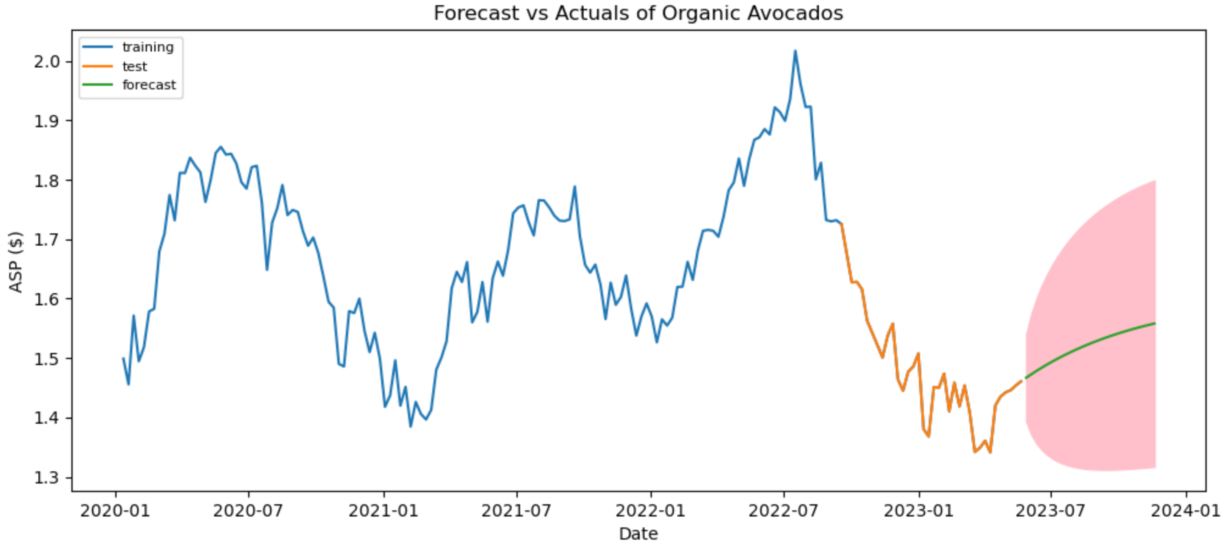


In [59]:
```
#90% confidence interval
#https://www.kaggle.com/code/jth359/confidence-interval/notebook

o_result = o_model2.get_forecast(26)
o_ci = o_result.conf_int(alpha=0.1)
print (o_ci.head())
```

```
            lower ASP  upper ASP
2023-05-28   1.393518   1.539847
2023-06-04   1.371203   1.573830
2023-06-11   1.356571   1.599637
2023-06-18   1.345963   1.620949
2023-06-25   1.337927   1.639239
```

In [60]:
```
#Lower and upper limits for visuals

o_ll = o_ci.loc[:,'lower ASP']
o_ul = o_ci.loc[:,'upper ASP']
```

In [61]:
```
#Forecast

st.t.interval(alpha=0.90, df=len(idata_org)-1,
              loc=np.mean(idata_org),
              scale=st.sem(idata_org))


#Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(idata_org, label='training')
plt.plot(o_test, label='test')
plt.plot(o_pred2, label='forecast')
plt.fill_between(o_ll.index, o_ll, o_ul, color='pink')
plt.title('Forecast vs Actuals of Organic Avocados')
plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```



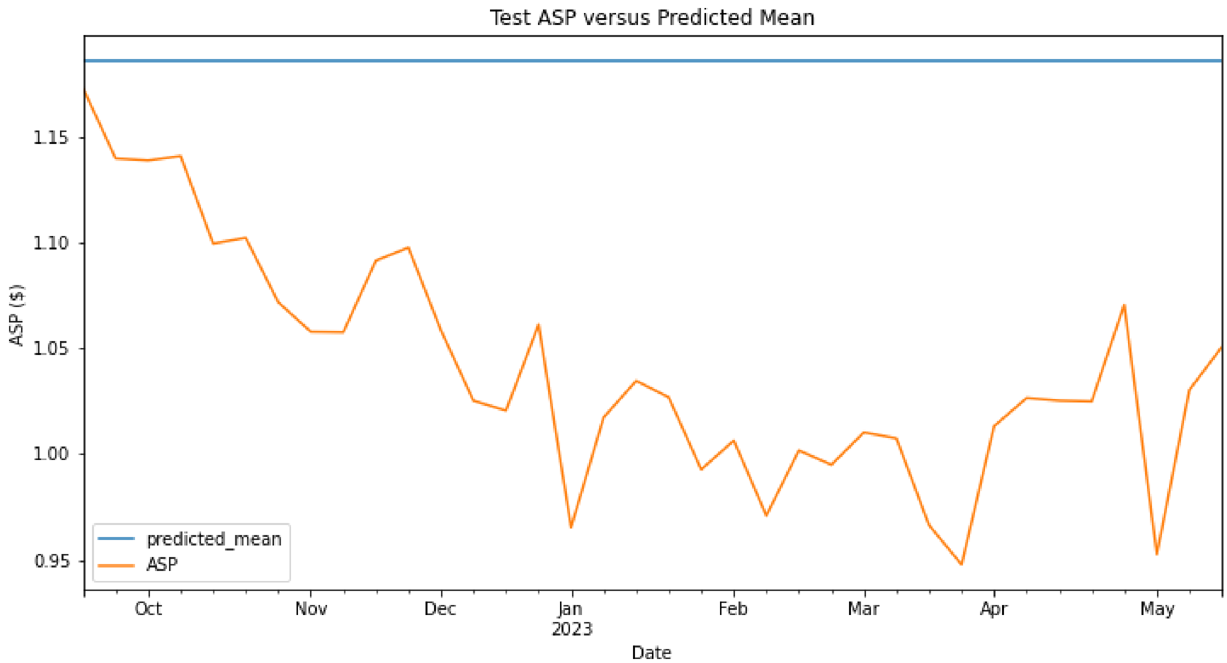
In [62]:
```
#The following cells will be for the conventional data
```

In [63]:
```
#Test data, printing predicted mean

c_start = len(c_train)
c_end=len(c_train)+len(c_test)-1
c_pred=c_results_ARIMA.predict(start=c_start, end=c_end, type='levels')
print(c_pred.head())
c_pred.index=idata_conv.index[c_start:o_end+1]
```

```
2022-09-18    1.185963
2022-09-25    1.185963
2022-10-02    1.185963
2022-10-09    1.185963
2022-10-16    1.185963
Freq: W-SUN, Name: predicted_mean, dtype: float64
```

In [64]:
```
#Plotting test data against the predicted mean

plt.title('Test ASP versus Predicted Mean')
plt.ylabel('ASP ($)')
c_pred.plot(legend=True)
c_test['ASP'].plot(legend=True)
```

Out[64]:
```
<AxesSubplot:title={'center':'Test ASP versus Predicted Mean'}, xlabel='Date', ylabel
='ASP ($)'>
```



In [65]:
```
#Mean of test data

c_test['ASP'].mean()
```

Out[65]: 1.040837422888889

In [66]:
```
#RMSE of test data; >10%

c_rmse = sqrt(mean_squared_error(c_pred,c_test['ASP']))
print(c_rmse)
```

0.155097811685126

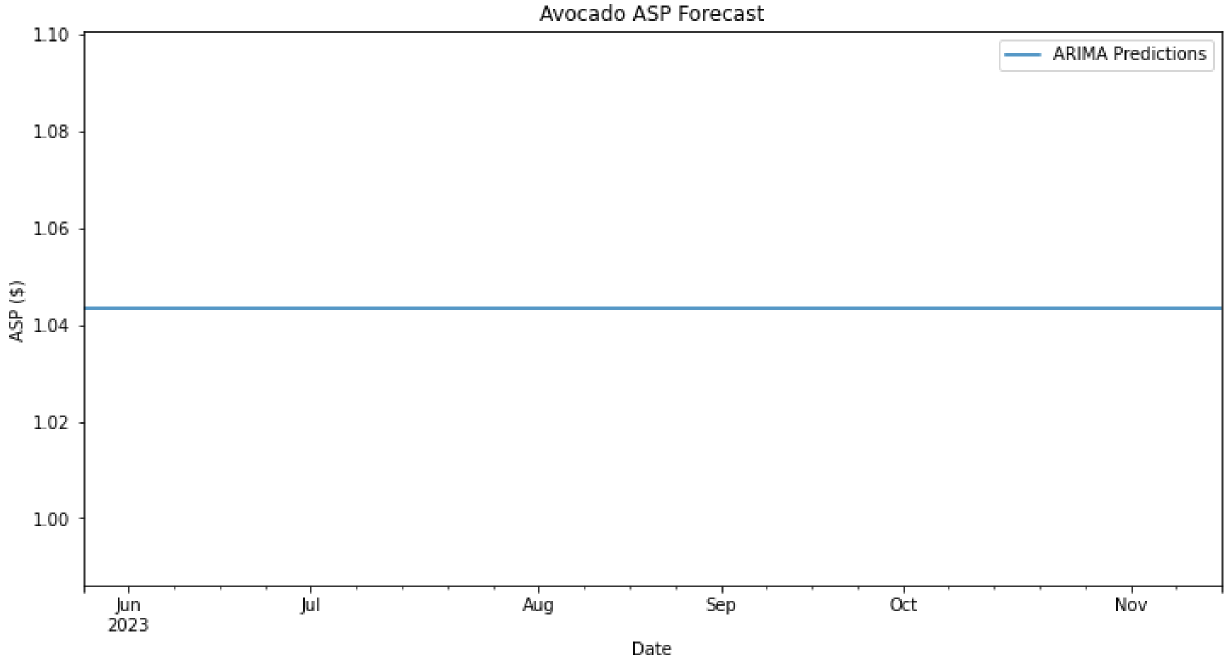
In [67]:
```
#ARIMA on the data for forecasting

c_model=ARIMA(idata_conv, order=(0,1,1))
c_model=c_model.fit()
```

In [68]:
```
#Create an index of the future dates

c_index_future_dates = pd.date_range(start = '2023-05-28',end='2023-11-19', freq='W')
print(c_index_future_dates)
c_pred2=c_model.predict(start=len(idata_conv), end=len(idata_conv)+25, typ='levels').r
c_pred2.index=c_index_future_dates
print(c_pred2)
plt.show()
```

```
DatetimeIndex(['2023-05-28', '2023-06-04', '2023-06-11', '2023-06-18',
               '2023-06-25', '2023-07-02', '2023-07-09', '2023-07-16',
               '2023-07-23', '2023-07-30', '2023-08-06', '2023-08-13',
               '2023-08-20', '2023-08-27', '2023-09-03', '2023-09-10',
               '2023-09-17', '2023-09-24', '2023-10-01', '2023-10-08',
               '2023-10-15', '2023-10-22', '2023-10-29', '2023-11-05',
               '2023-11-12', '2023-11-19'],
              dtype='datetime64[ns]', freq='W-SUN')
2023-05-28    1.043366
2023-06-04    1.043366
2023-06-11    1.043366
2023-06-18    1.043366
2023-06-25    1.043366
2023-07-02    1.043366
2023-07-09    1.043366
2023-07-16    1.043366
2023-07-23    1.043366
2023-07-30    1.043366
2023-08-06    1.043366
2023-08-13    1.043366
2023-08-20    1.043366
2023-08-27    1.043366
2023-09-03    1.043366
2023-09-10    1.043366
2023-09-17    1.043366
2023-09-24    1.043366
2023-10-01    1.043366
2023-10-08    1.043366
2023-10-15    1.043366
2023-10-22    1.043366
2023-10-29    1.043366
2023-11-05    1.043366
2023-11-12    1.043366
2023-11-19    1.043366
Freq: W-SUN, Name: ARIMA Predictions, dtype: float64
```

In [69]:

```
#Plotting the forecast on its own.

c_pred2.plot(legend=True)
plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.title('Avocado ASP Forecast')
plt.show()
```

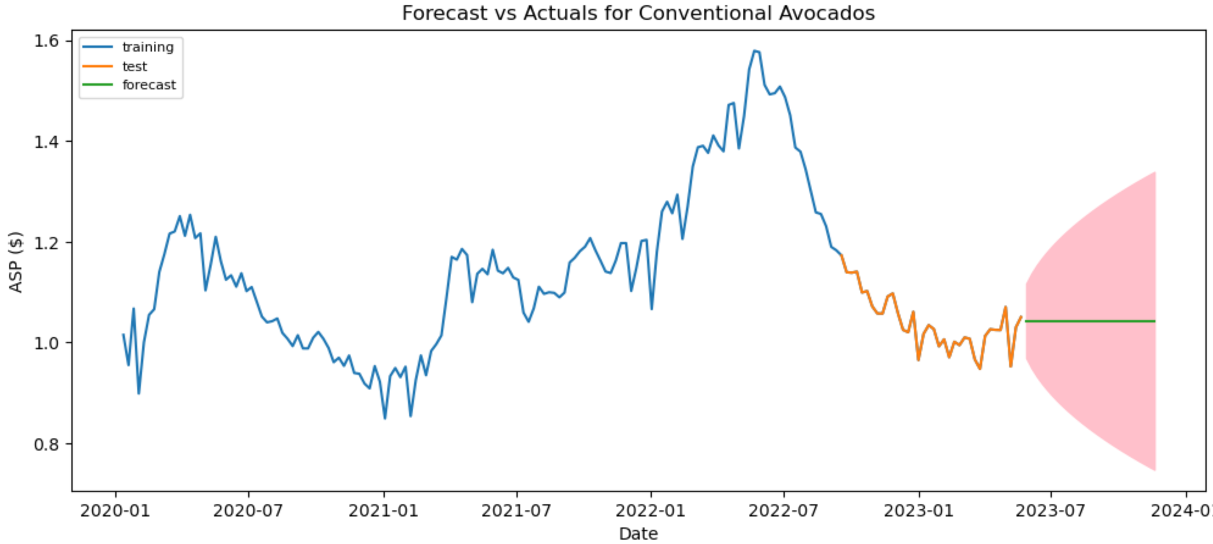


In [70]:
```
#90% confidence interval

c_result = c_model.get_forecast(26)
c_ci = c_result.conf_int(alpha=0.1)
print(c_ci.head())
```

```
            lower ASP  upper ASP
2023-05-28   0.969694   1.117038
2023-06-04   0.950083   1.136649
2023-06-11   0.933932   1.152800
2023-06-18   0.919876   1.166856
2023-06-25   0.907263   1.179468
```

In [71]:
```
#Lower and upper limits for visuals

c_ll = c_ci.loc[:,'lower ASP']
c_ul = c_ci.loc[:,'upper ASP']
```

In [72]:
```
#Forecast

st.t.interval(alpha=0.90, df=len(idata_conv)-1,
              loc=np.mean(idata_conv),
              scale=st.sem(idata_conv))


#Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(idata_conv, label='training')
plt.plot(c_test, label='test')
plt.plot(c_pred2, label='forecast')
plt.fill_between(c_ll.index, c_ll, c_ul, color='pink')
plt.title('Forecast vs Actuals for Conventional Avocados')
plt.xlabel('Date')
plt.ylabel('ASP ($)')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```


Forecast vs Actuals for Conventional Avocados
training
test
forecast
ASP ($)
1.6
1.4
1.2
1.0
0.8
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
Date

In [ ]: