In [1]:
```
import pandas                                as      pd
import numpy                                 as      np
import matplotlib.pyplot                     as      plt
from   IPython.display                       import  display
from   pylab                                 import  rcParams
from   datetime                              import  datetime, timedelta
from   pandas.tseries.offsets                import BDay
from statsmodels.tsa.stattools               import  adfuller
from statsmodels.tsa.stattools               import  pacf
from statsmodels.tsa.stattools               import  acf
from statsmodels.graphics.tsaplots           import  plot_pacf
from statsmodels.graphics.tsaplots           import  plot_acf
from statsmodels.graphics.gofplots           import  qqplot
from statsmodels.tsa.seasonal                import  seasonal_decompose
from   statsmodels.tsa.api                   import  ExponentialSmoothing,Holt
from statsmodels.tsa.arima.model             import  ARIMA
from sklearn.metrics                         import  mean_squared_error
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import itertools
```

In [2]:
```
plt.rcParams['figure.figsize'] = 12,8
```

In [3]:
```
df = pd.read_csv('retail_sales.csv')
df.head()
```

Out [3]:

	ds	y
0	1992-01-01	146376
1	1992-02-01	147079
2	1992-03-01	159336
3	1992-04-01	163669
4	1992-05-01	170068

In [4]:
```
df.head(),df.tail()
```

Out [4]:
```
(          ds       y
 0  1992-01-01  146376
 1  1992-02-01  147079
 2  1992-03-01  159336
 3  1992-04-01  163669
 4  1992-05-01  170068,
             ds       y
 288  2016-01-01  400928
 289  2016-02-01  413554
 290  2016-03-01  460093
 291  2016-04-01  450935
 292  2016-05-01  471421)
```

In [5]:
```
date = pd.date_range(start='1992/01/01',end='2016/05/01',freq=BDay())
date
```

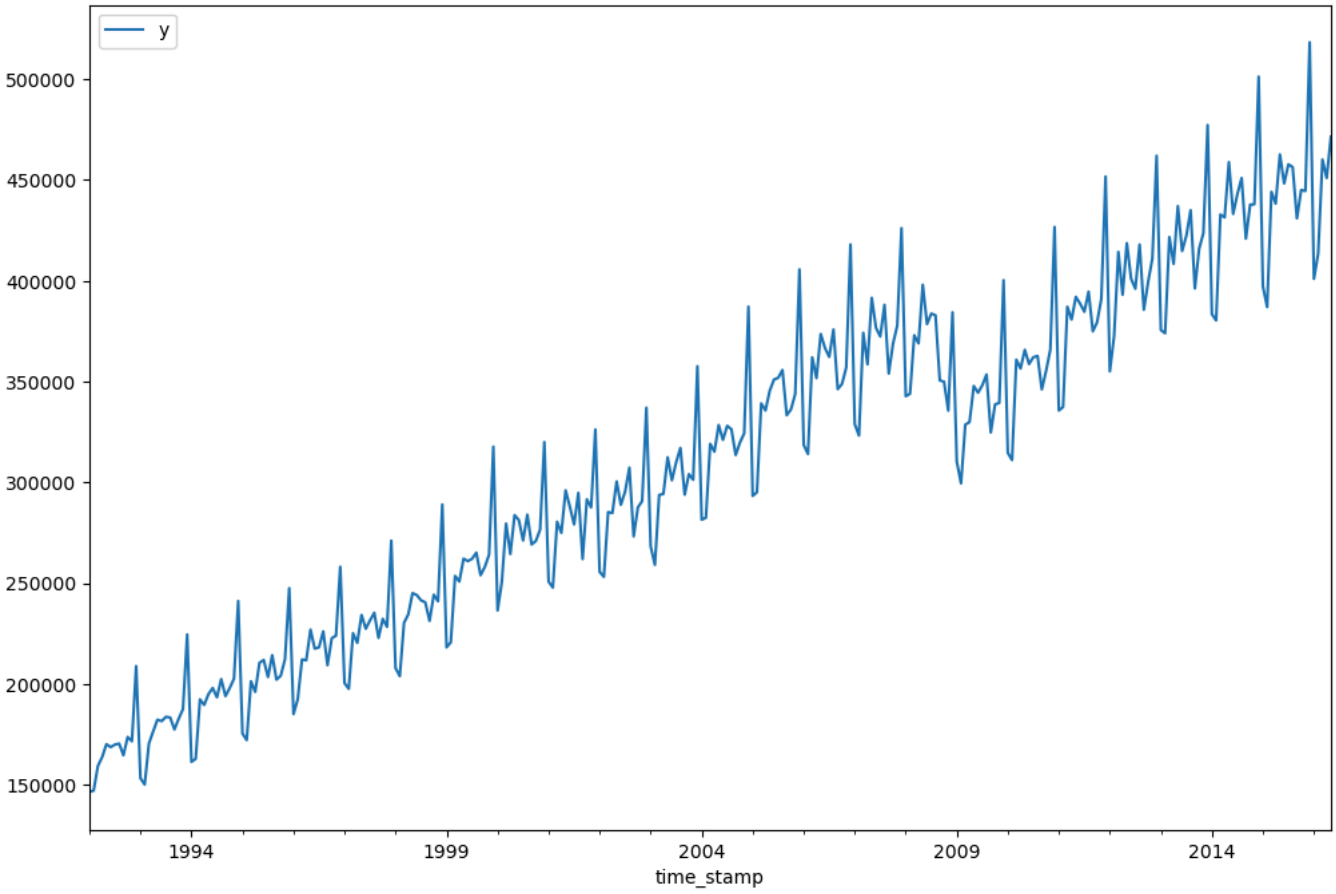
Out [5]:
```
DatetimeIndex(['1992-01-01', '1992-01-02', '1992-01-03', '1992-01-06',
               '1992-01-07', '1992-01-08', '1992-01-09', '1992-01-10',
               '1992-01-13', '1992-01-14',
               ...
               '2016-04-18', '2016-04-19', '2016-04-20', '2016-04-21',
               '2016-04-22', '2016-04-25', '2016-04-26', '2016-04-27',
               '2016-04-28', '2016-04-29'],
              dtype='datetime64[ns]', length=6348, freq='B')
```

In [6]:
```
df['time_stamp'] = pd.to_datetime(df['ds'],format='%Y-%m-%d')
df = df.set_index('time_stamp')
df = df.drop('ds',axis=1)
df.head()
```

Out [6]:

time_stamp	y
1992-01-01	146376
1992-02-01	147079
1992-03-01	159336
1992-04-01	163669
1992-05-01	170068

In [7]:
```
df.plot()
plt.show()
```



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

Out [8]:
```
y    0
dtype: int64
```

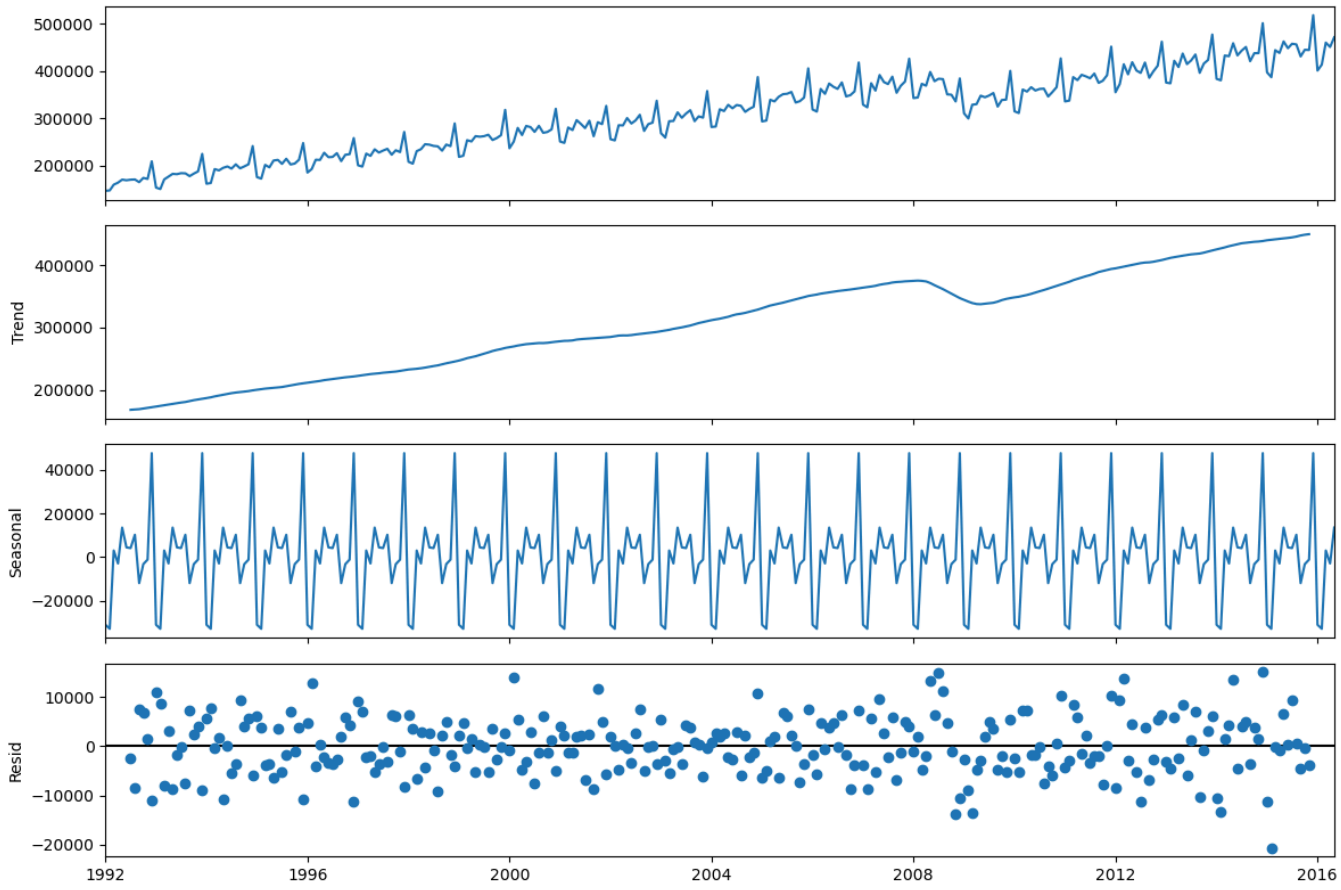
In [9]:
```
df.describe()
```

Out [9]:

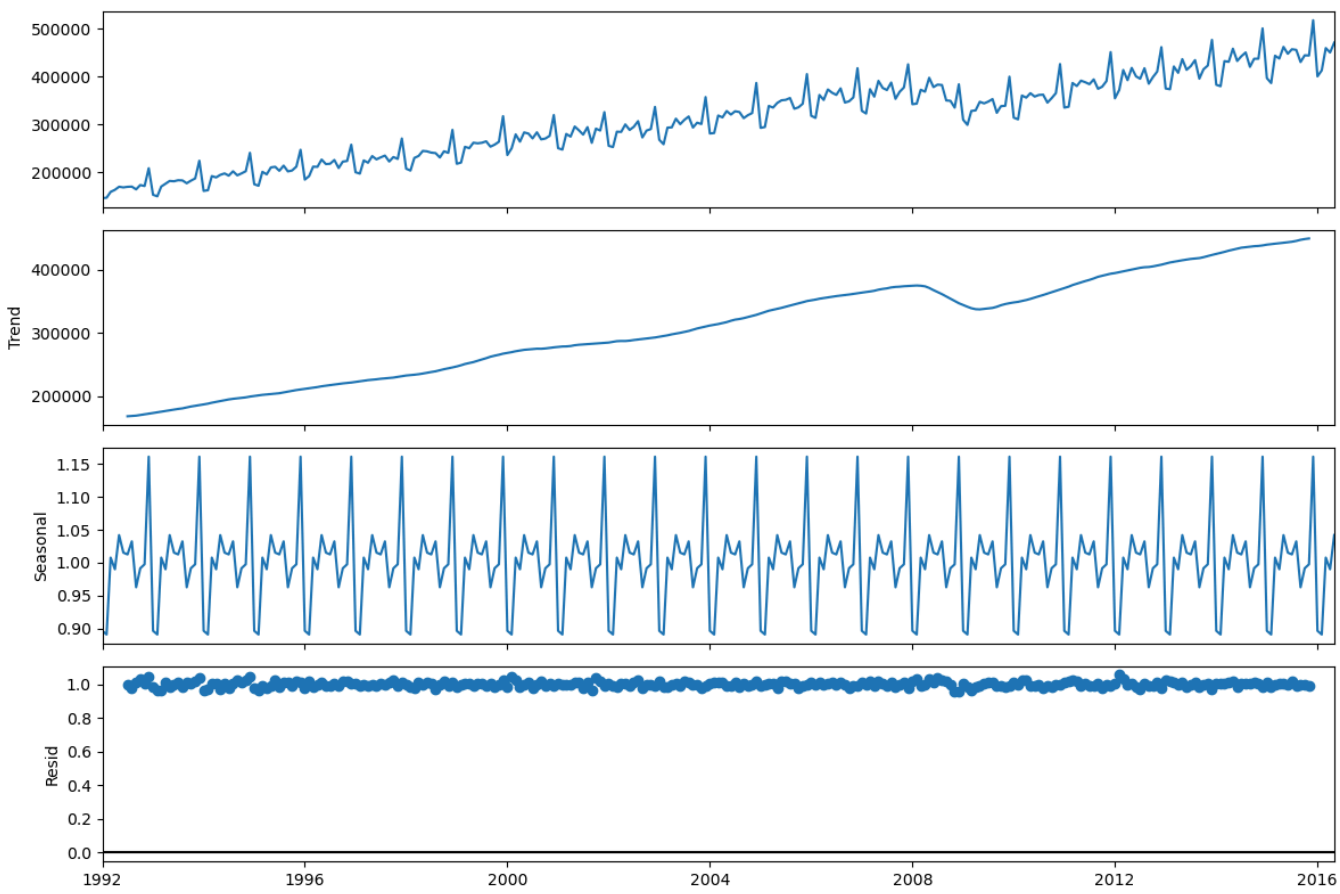
	y
count	293.000000
mean	308971.310580
std	86084.323897
min	146376.000000
25%	234503.000000
50%	314051.000000

	y
75%	375795.000000
max	518253.000000

In [10]:
```
decompose_add = seasonal_decompose(df,model='add')
decompose_add.plot()
plt.show()
```



In [11]:
```
decompose_mul = seasonal_decompose(df,model='mul')
decompose_mul.plot()
plt.show()
```



In [12]:
```
observation_df = df.values
result_df = adfuller(observation_df)

print(f"ADF test statistics: {result_df[0]}")
print(f"p_value: {result_df[1]}")
print(f"Used lags: {result_df[2]}")
print(f"Number of observations: {result_df[3]}")
for key,value in result_df[4].items():
    print(f"{key}: {value}")

print(f"IC best: {result_df[5]}")
```

```
ADF test statistics: -0.6443250505938629
p_value: 0.8606486181958646
Used lags: 15
Number of observations: 277
1%: -3.4541800885158525
5%: -2.872031361137725
10%: -2.5723603999791473
IC best: 5671.009371066914
```

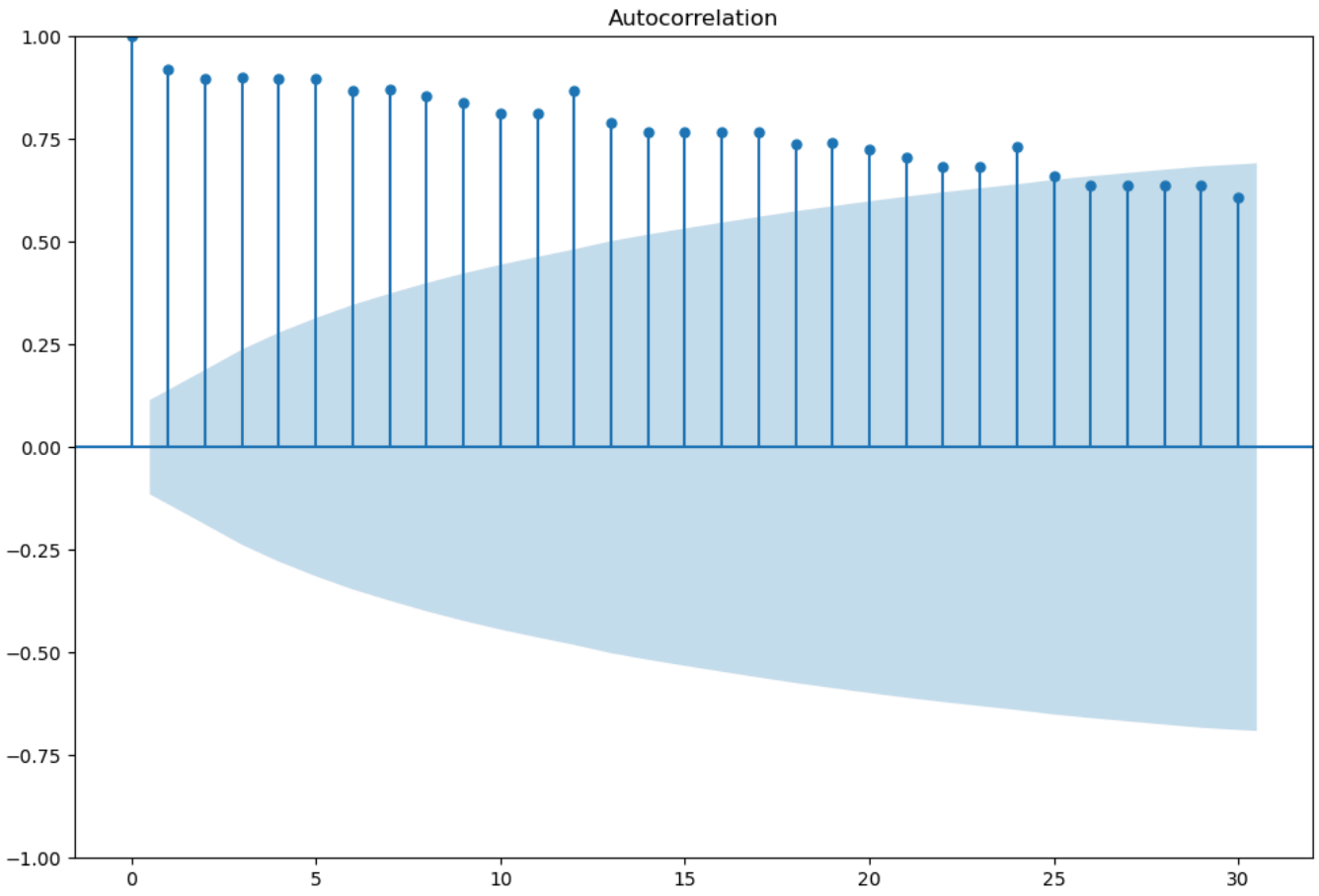
In [13]:
```
df_diff = df.diff(periods=1).dropna()
observation_diff = df_diff.values
result_diff = adfuller(observation_diff)

print(f"ADF test statistics: {result_diff[0]}")
print(f"p_value: {result_diff[1]}")
print(f"used lags: {result_diff[2]}")
print(f"Number of observations: {result_diff[3]}")
for key,values in result_diff[4].items():
    print(f"{key}: {values}")
print(f"IC Best: {result_diff[5]}")
```

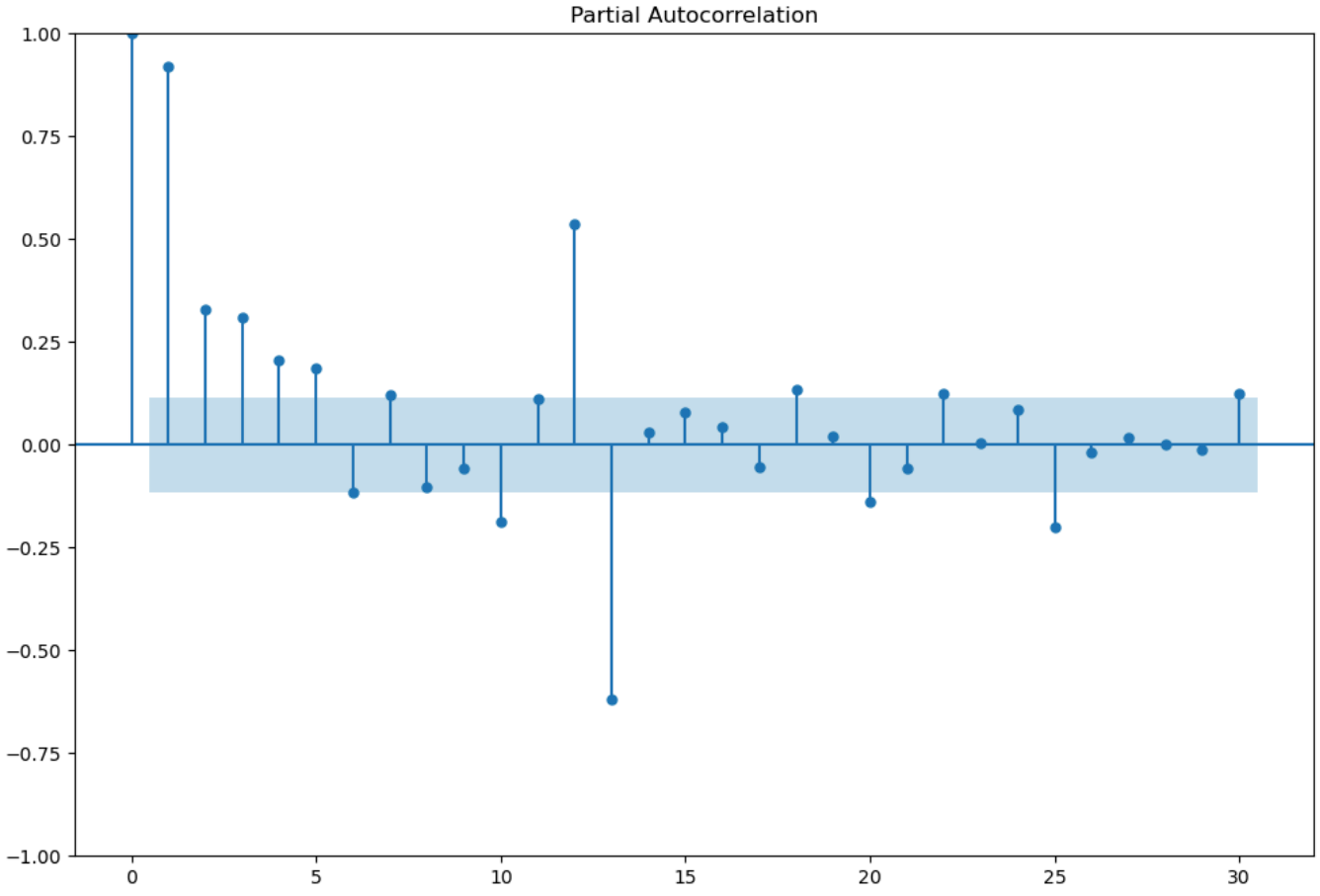
```
ADF test statistics: -3.437937154315123
p_value: 0.009731610960261502
used lags: 14
Number of observations: 277
1%: -3.4541800885158525
5%: -2.872031361137725
```

```
10%: -2.5723603999791473
IC Best: 5649.985617927769
```

In [14]:
```
plot_acf(df,lags=30)
plt.show()
```



In [15]:
```
plot_pacf(df,lags=30)
plt.show()
```



In [16]:
```
df.head(),df.tail()
```

Out [16]:
```
(                y
 time_stamp
 1992-01-01  146376
 1992-02-01  147079
 1992-03-01  159336
 1992-04-01  163669
 1992-05-01  170068,
                 y
 time_stamp
 2016-01-01  400928
 2016-02-01  413554
 2016-03-01  460093
 2016-04-01  450935
 2016-05-01  471421)
```

In [17]:
```
train_set = datetime(2013,11,1)
test_set = datetime(2016,5,1)
train = df[:train_set]
test = df[train_set+timedelta(days=1):test_set]
```

In [18]:
```
train.shape,test.shape
```

Out [18]:
```
((263, 1), (30, 1))
```

In [19]:
```
train.isnull().sum()
```

Out [19]:
```
y    0
dtype: int64
```

In [20]:
```
p = q = range(0,4)
d = range(0,2)
pdq = itertools.product(p,d,q)
arima_model = []
for i in pdq:
    model = ARIMA(train,order=i)
    result_arima = model.fit()
    arima_model.append((i,result_arima.aic))
```

In [21]:
```
frame = pd.DataFrame(arima_model,columns=['param','aic'])
```

In [22]:
```
frame.sort_values(by='aic',ascending=True).head()
```

Out [22]:

	param	aic
27	(3, 0, 3)	6000.167826
19	(2, 0, 3)	6012.776444
15	(1, 1, 3)	6014.462657
23	(2, 1, 3)	6015.041477
11	(1, 0, 3)	6015.902494

In [23]:
```
model_arima = ARIMA(train,order=(3,0,3))
result_arima_1 = model_arima.fit()
print(result_arima_1.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  263
Model:                 ARIMA(3, 0, 3)   Log Likelihood               -2992.084
Date:                Wed, 19 Feb 2025   AIC                           6000.168
Time:                        22:07:04   BIC                           6028.745
Sample:                    01-01-1992   HQIC                          6011.652
                         - 11-01-2013
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
const        2.94e+05   1.41e-09   2.08e+14      0.000    2.94e+05    2.94e+05
ar.L1          -0.6535      0.050    -12.999      0.000      -0.752      -0.555
ar.L2           0.7872      0.022     35.956      0.000       0.744       0.830
ar.L3           0.8660      0.048     17.986      0.000       0.772       0.960
ma.L1           1.0641      0.079     13.476      0.000       0.909       1.219
ma.L2          -0.4956      0.085     -5.842      0.000      -0.662      -0.329
ma.L3          -0.7968      0.072    -11.056      0.000      -0.938      -0.656
sigma2      4.862e+08   1.36e-10   3.58e+18      0.000    4.86e+08    4.86e+08
===================================================================================
Ljung-Box (L1) (Q):                   3.03   Jarque-Bera (JB):                12.21
Prob(Q):                              0.08   Prob(JB):                         0.00
Heteroskedasticity (H):               2.11   Skew:                            -0.10
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.04
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.51e+33. Standard errors may be unstable.
```

In [24]:
```
from sklearn.metrics import root_mean_squared_error
```

In [25]:
```
class Evaluate:
    def Rmse(y_true,y_pred):
        return root_mean_squared_error(y_true,y_pred)

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

In [26]:
```
arima_pred = result_arima_1.forecast(len(test))
```

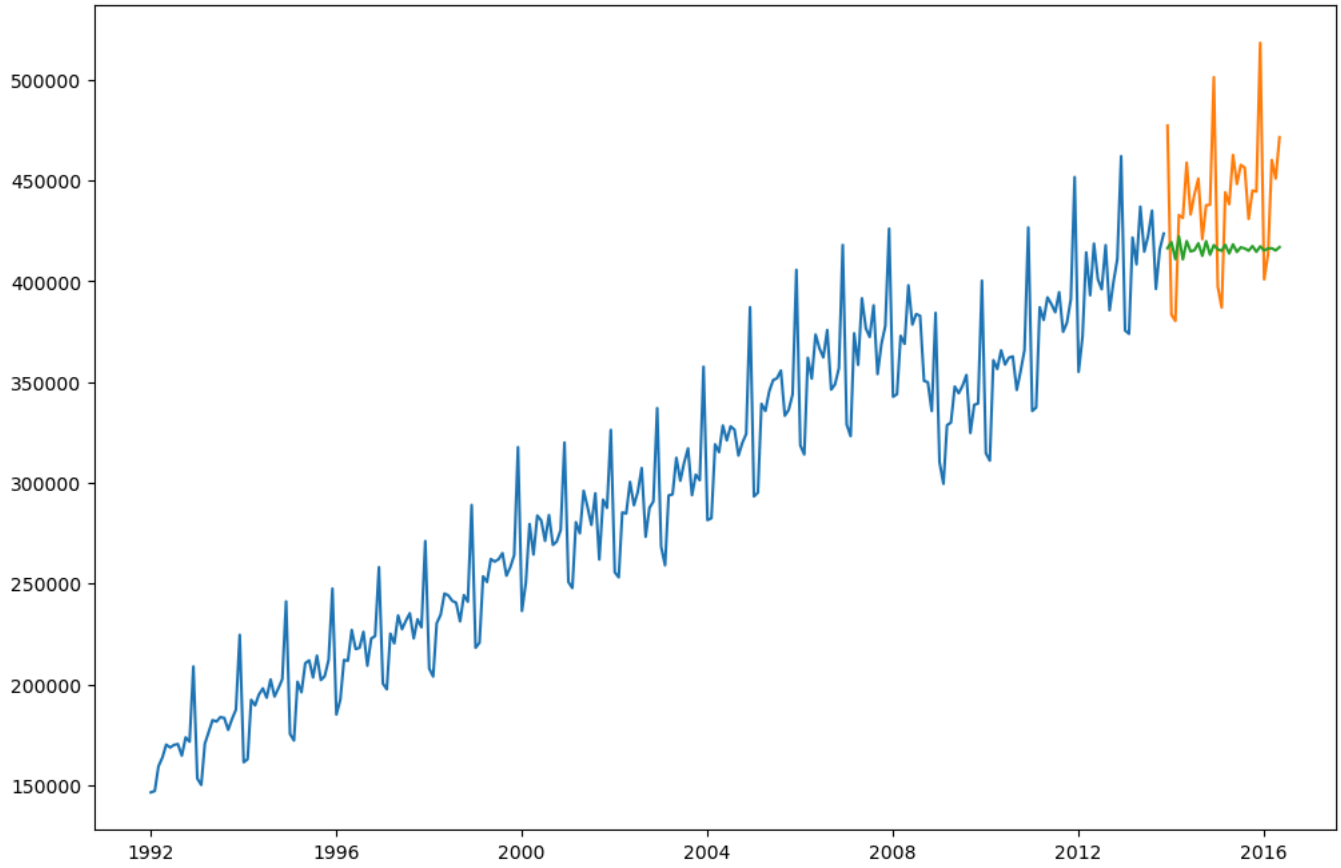
In [27]:
```
Evaluate.Rmse(test['y'],arima_pred)
```

Out [27]:
```
38866.408528879765
```

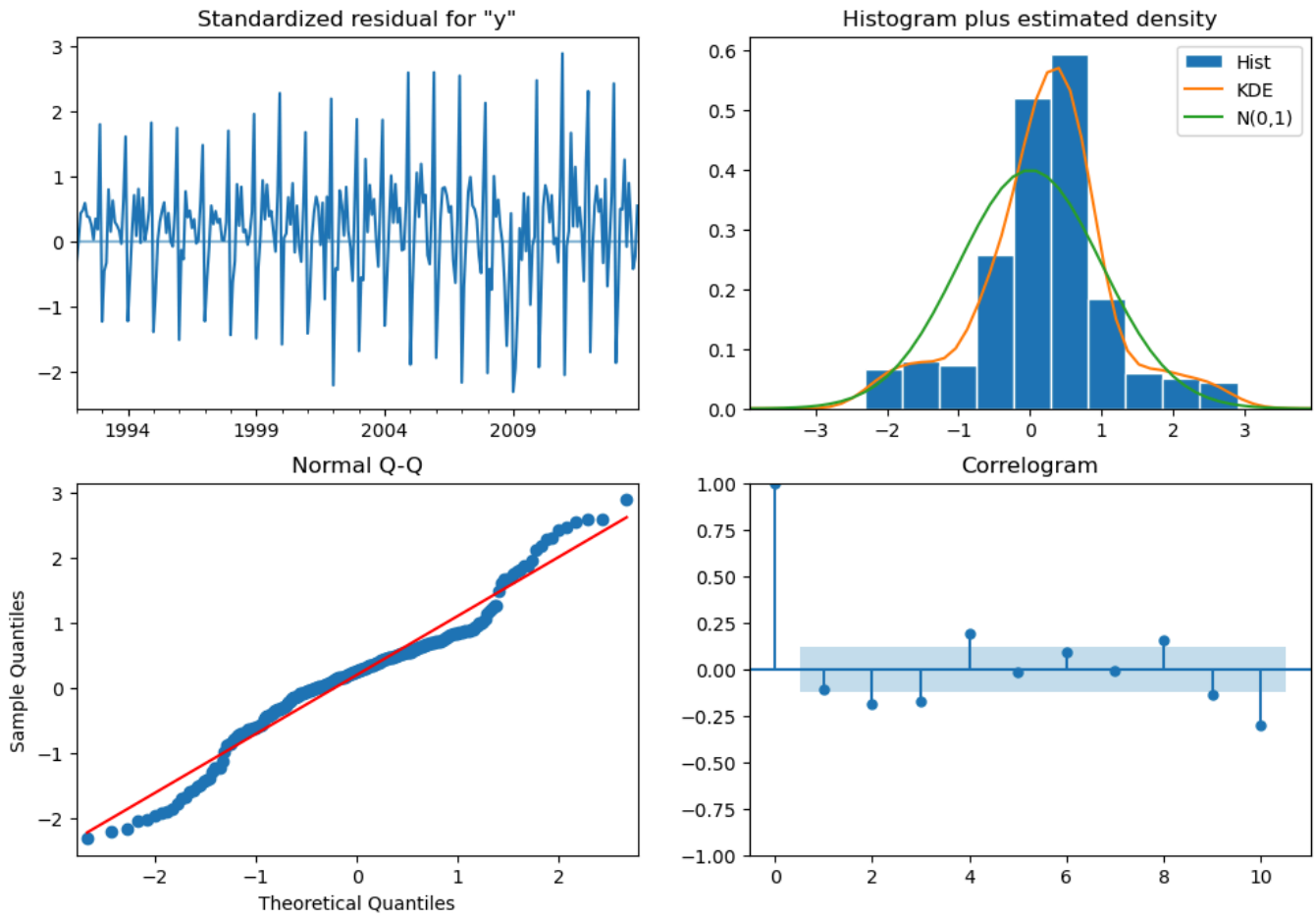
In [28]:
```
Evaluate.Mape(test['y'],arima_pred)
```

Out [28]:
```
7.309302838308487
```

In [29]:
```
plt.plot(train)
plt.plot(test)
plt.plot(arima_pred)
plt.show()
```



In [30]:
```
result_arima_1.plot_diagnostics()
plt.show()
```



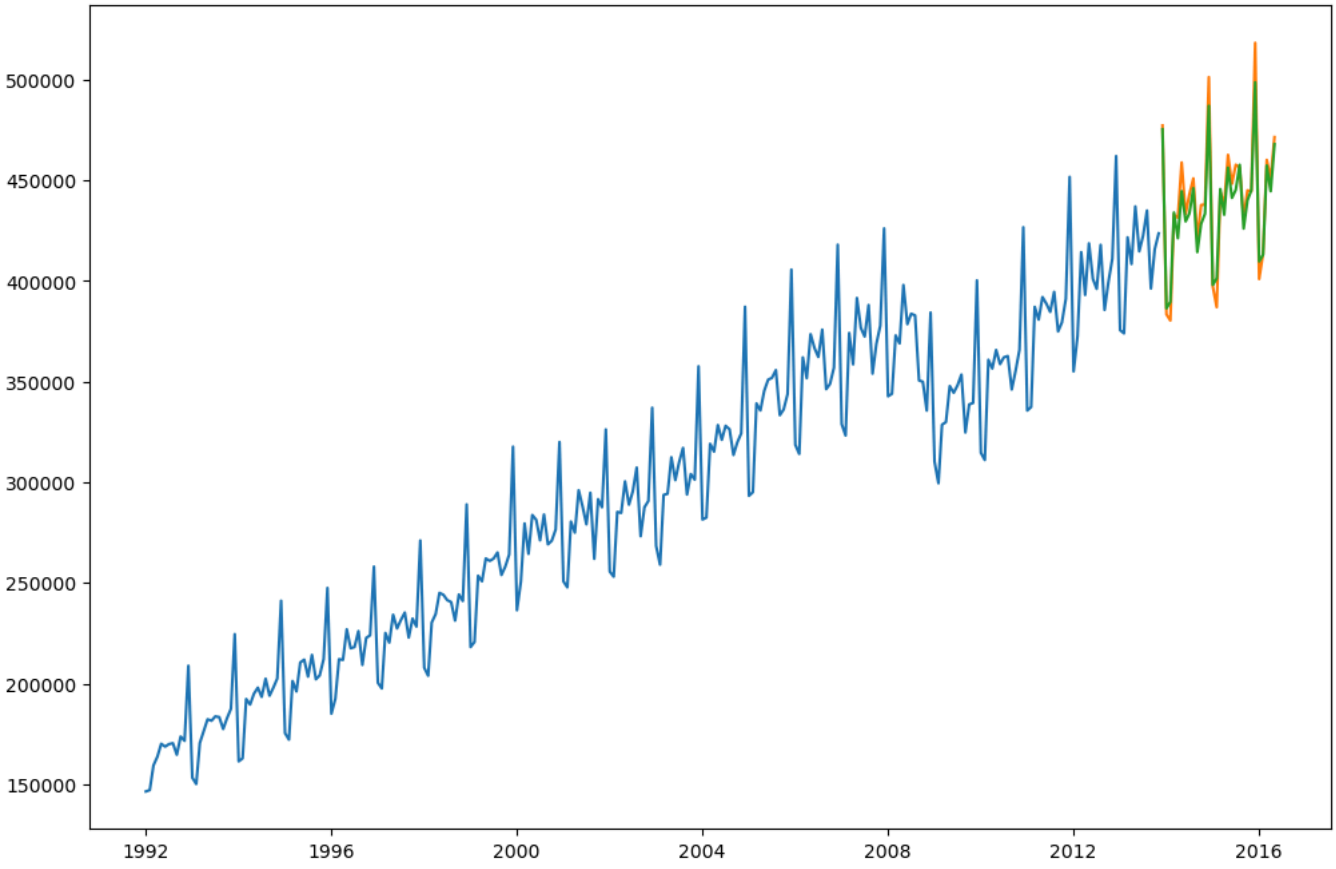
In [31]:
```
model_exp = ExponentialSmoothing(train,trend='additive',seasonal='additive',initialization
result_exp = model_exp.fit()
print(result_exp.summary())
```

```
                       ExponentialSmoothing Model Results
==============================================================================
Dep. Variable:                          y   No. Observations:                  263
```

```
Model:              ExponentialSmoothing   SSE                 10327972264.770
Optimized:                          True   AIC                        4630.810
Trend:                          Additive   BIC                        4687.964
Seasonal:                       Additive   AICC                       4633.613
Seasonal Periods:                     12   Date:                Wed, 19 Feb 2025
Box-Cox:                           False   Time:                        22:07:05
Box-Cox Coeff.:                     None
================================================================================
                          coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level            0.5000000              alpha                 True
smoothing_trend               0.0001               beta                 True
smoothing_seasonal         0.3250000              gamma                 True
initial_level             1.6654e+05                l.0                 True
initial_trend              972.51187                b.0                 True
initial_seasons.0         -24263.224                s.0                 True
initial_seasons.1         -24717.464                s.1                 True
initial_seasons.2         -1087.3281                s.2                 True
initial_seasons.3         -2654.0052                s.3                 True
initial_seasons.4          6487.0990                s.4                 True
initial_seasons.5          3994.2552                s.5                 True
initial_seasons.6          739.52604                s.6                 True
initial_seasons.7          4860.1094                s.7                 True
initial_seasons.8         -4216.2552                s.8                 True
initial_seasons.9         -193.24479                s.9                 True
initial_seasons.10         2540.8698               s.10                 True
initial_seasons.11         38509.661               s.11                 True
--------------------------------------------------------------------------------
```

In [32]:
```
pred_exp = result_exp.forecast(len(test))
```

In [33]:
```
plt.plot(train)
plt.plot(test)
plt.plot(pred_exp)
plt.show()
```



In [34]:
```
Evaluate.Rmse(test['y'],pred_exp)
```

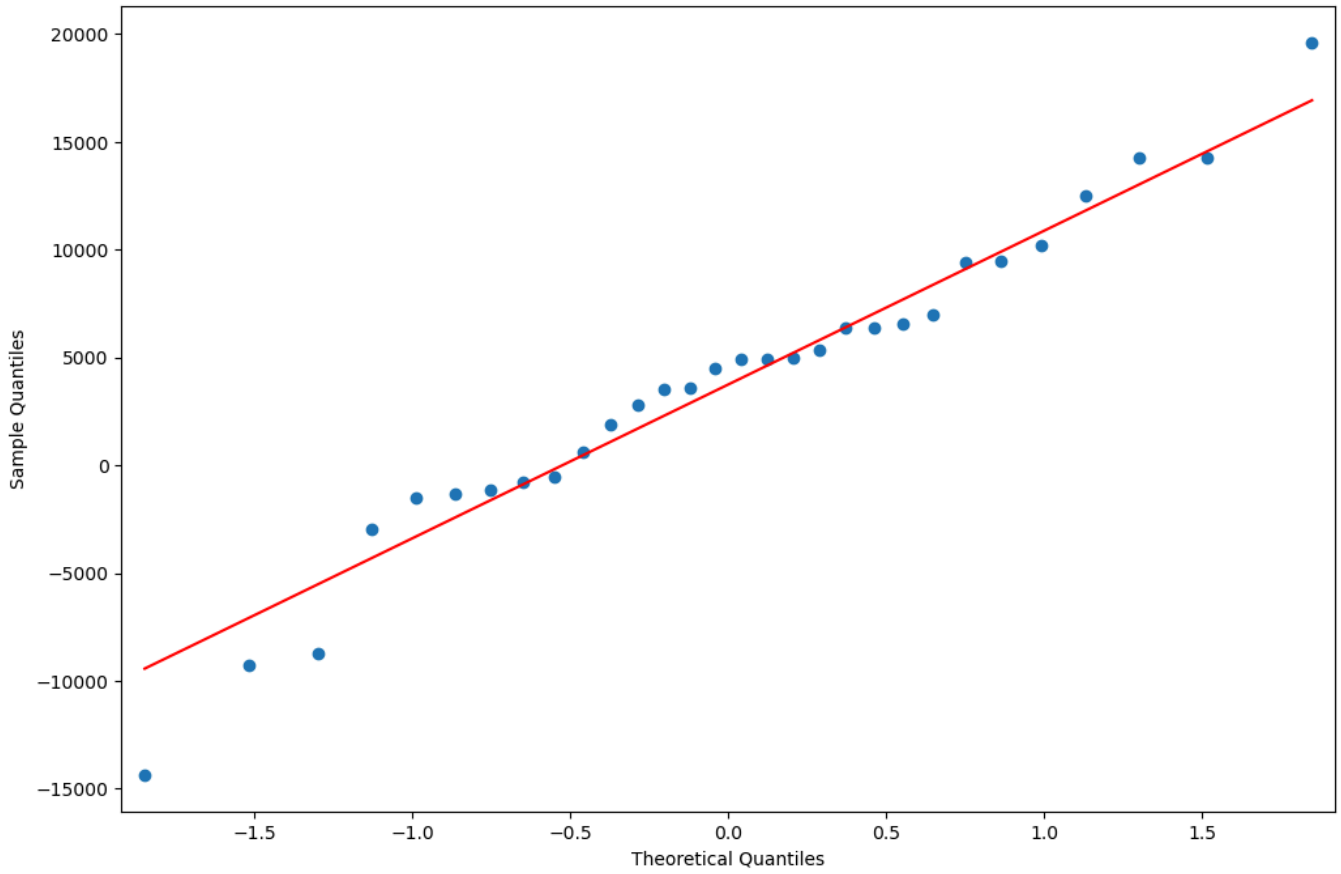
Out [34]: 8054.220053734076

In [35]:
```
Evaluate.Mape(test['y'],pred_exp)
```

Out [35]: 1.456057930754873

In [36]:
```
resuduals = test['y'].values - pred_exp.values
```

In [37]:
```
qqplot(resuduals,line='s')
plt.show()
```



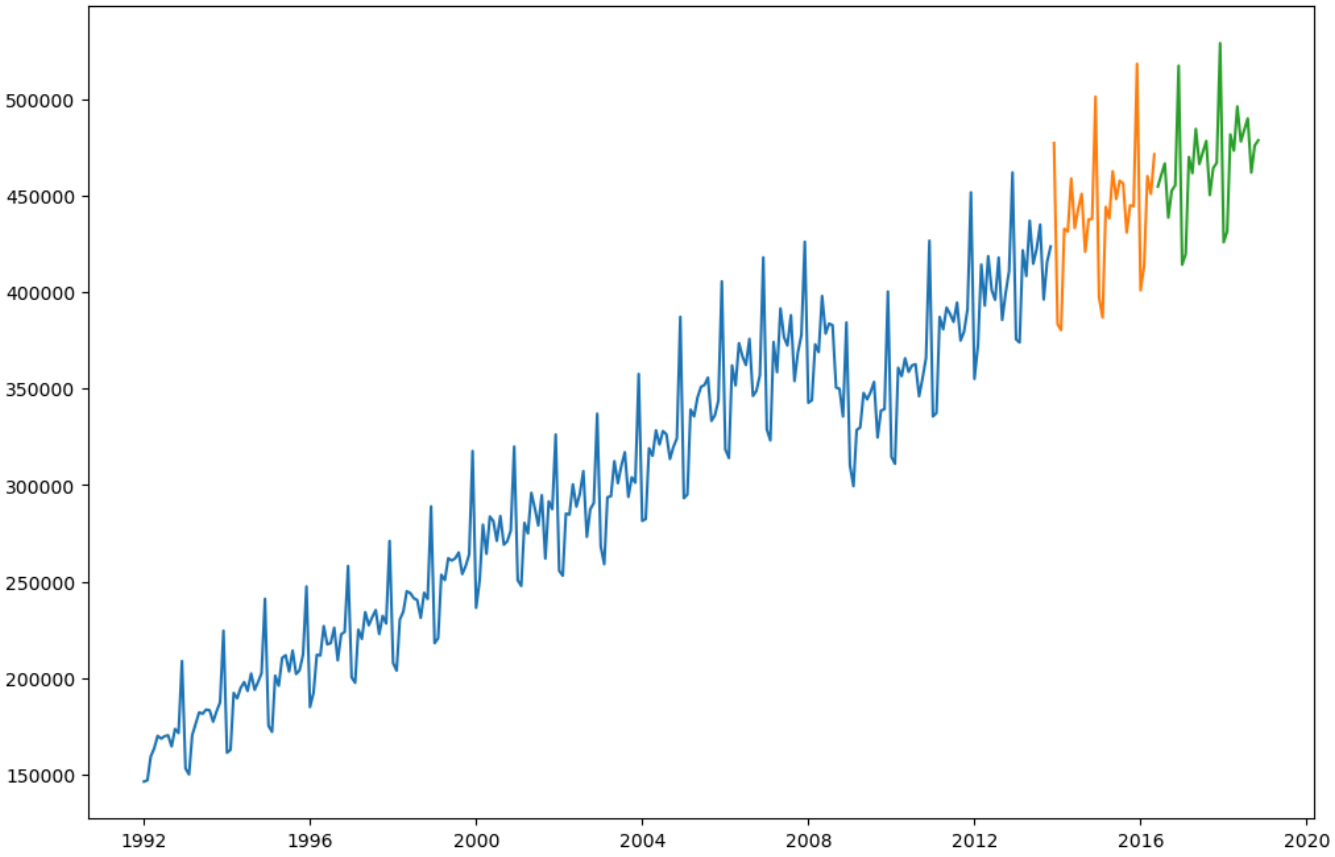
In [38]:
```
model_exp_tune = ExponentialSmoothing(train['y'],trend='add',seasonal='mul',seasonal_perio
result_exp_tune = model_exp_tune.fit()
print(result_exp_tune.summary())
```

```
                       ExponentialSmoothing Model Results
==============================================================================
Dep. Variable:                         y   No. Observations:                  263
Model:             ExponentialSmoothing   SSE                   9936302247.368
Optimized:                         True   AIC                         4620.642
Trend:                         Additive   BIC                         4677.796
Seasonal:                Multiplicative   AICC                        4623.445
Seasonal Periods:                    12   Date:                 Wed, 19 Feb 2025
Box-Cox:                          False   Time:                         22:07:06
Box-Cox Coeff.:                    None
==============================================================================
                          coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level         0.5000000                alpha                 True
smoothing_trend            0.0001                 beta                 True
smoothing_seasonal      0.2750000                gamma                 True
initial_level           1.6654e+05                  1.0                 True
initial_trend            972.51187                  b.0                 True
initial_seasons.0        0.8743406                  s.0                 True
initial_seasons.1        0.8715923                  s.1                 True
initial_seasons.2        0.9938863                  s.2                 True
initial_seasons.3        0.9869043                  s.3                 True
initial_seasons.4        1.0320753                  s.4                 True
initial_seasons.5        1.0201988                  s.5                 True
initial_seasons.6        1.0045706                  s.6                 True
initial_seasons.7        1.0248818                  s.7                 True
initial_seasons.8        0.9773991                  s.8                 True
initial_seasons.9        0.9998626                  s.9                 True
initial_seasons.10       1.0129029                 s.10                 True
initial_seasons.11       1.2013853                 s.11                 True
------------------------------------------------------------------------------
```

In [39]:
```
final_exp = ExponentialSmoothing(df['y'],trend='add',seasonal='add',seasonal_periods=12)
result_final = final_exp.fit(smoothing_level=0.5,
                             smoothing_trend = 0.0001,
                             smoothing_seasonal = 0.2750000 ,
                             optimized = False)
```

In [40]:
```
final_pred = result_final.forecast(30)
```

In [41]:
```
plt.plot(train)
plt.plot(test)
plt.plot(final_pred)
plt.show()
```



In [42]:
```
new = pd.DataFrame(final_pred,columns=['pred'])
new.head()
```

Out [42]:

	pred
2016-06-01	454690.460806
2016-07-01	460684.693340
2016-08-01	466657.798203
2016-09-01	438619.823755
2016-10-01	452555.354617

In [ ]:
```

```