Part I: Introduction

This project aims to study a random region/city electricity consumption from 1985 to 2018 and fit it into a time series to analyze and forecast future consumption, as well as identify key characteristics of this unknown location through the behavior reflected in the data.

In general, electricity consumption is expected to increase over time, but does it stop increasing at some point and what would this indicate about the random place or region that we're analyzing.

The data have been selected since it is expected to include all 3 components of a time series, namely trend, seasonality, and cycles, hence, allowing us to study those behaviors and apply the knowledge gained through this course.

Package and Data Importation

First, we import the required packages and data file that will be used throughout this project

In [1]:
```python
import pandas as pd
import numpy as np
import itertools

#Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt
plt.style.use('seaborn-white')
%matplotlib inline

#statistics libraries
import statsmodels.api as sm
import scipy
from scipy.stats import anderson
from statsmodels.tools.eval_measures import rmse
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot, plot_acf, plot_pacf, quarte
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing, SimpleExpSmoothing
from statsmodels.stats.diagnostic import acorr_ljungbox as ljung
#from nimbusml.timeseries import SsaForecaster
from statsmodels.tsa.statespace.tools import diff as diff
import pmdarima as pm
from pmdarima import ARIMA, auto_arima
from scipy import signal
from scipy.stats import shapiro
from scipy.stats import boxcox
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore")
np.random.seed(123)

from statsmodels.tsa.seasonal import seasonal_decompose

data = pd.read_csv('Electric_Production.csv',parse_dates=True, index_col="DATE")
data.head()
data.index.freq='MS'
data.head()
```

Out[1]:

	Value
DATE	
1985-01-01	72.5052
1985-02-01	70.6720
1985-03-01	62.4502
1985-04-01	57.4714
1985-05-01	55.3151

We first inspect the data for any missing values

In [2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 397 entries, 1985-01-01 to 2018-01-01
Freq: MS
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Value   397 non-null    float64
dtypes: float64(1)
memory usage: 6.2 KB
```

The data seems to be complete, next we will evaluate first order properties.

Part II: Results

1. Evaluate the first order properties of your data.

We start by looking at the data to ensure that it's proper for our use case and adjust if required.

In [3]:
```
pd.set_option("display.max_rows", None)
data
```

Out[3]:

DATE	Value
1985-01-01	72.5052
1985-02-01	70.6720
1985-03-01	62.4502
1985-04-01	57.4714
1985-05-01	55.3151
1985-06-01	58.0904
1985-07-01	62.6202
1985-08-01	63.2485
1985-09-01	60.5846
1985-10-01	56.3154

The data is monthly averages from Jan 1985 to Jan 2018 (inclusive) which should have 12*(2018-1985)+1 months

In [4]:
```
# we compute the above and compare it to the number of rows
12*(2018-1985)+1
```

Out[4]:
```
397
```

Since the number of rows is equavalent to the expected value, we conclude that the data includes all expected months without discontinuity.

Looking at all the values, nothing stands out to be out of the ordinary, we will further inspect the behavior through looking at a plot of the data and evaluate the other first order properties (trend, seasonality and cycles)

Since were interested in analyzing perfect periods, we will drop Jan 2018 to keep the data from 1985 to end of 2017 with complete months

In [5]:
```
data.drop(axis=0, index='2018-01-01',inplace=True)
```

In [6]:
```
data.tail()
```

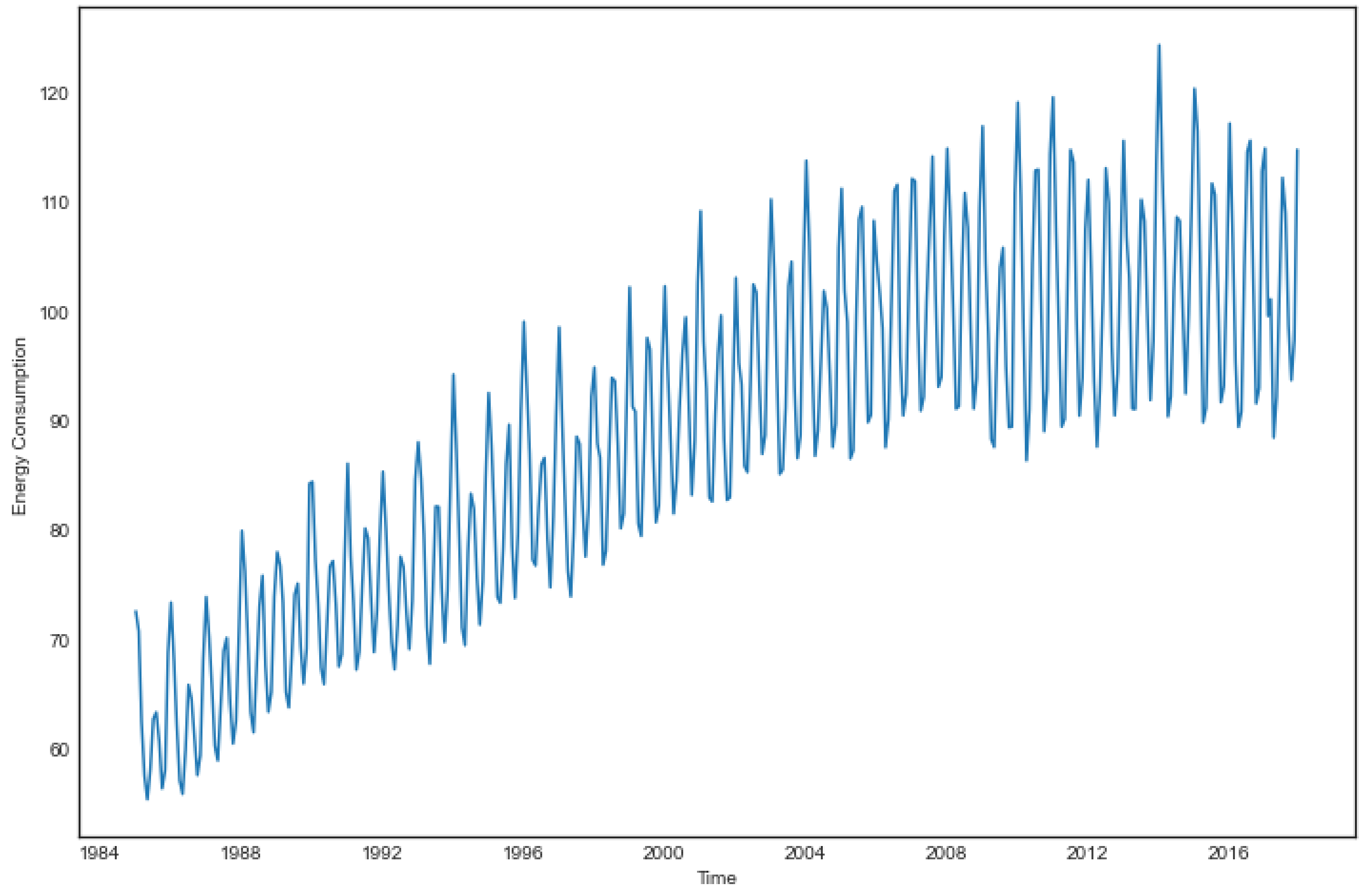
Out[6]:

DATE	Value
2017-08-01	108.9312
2017-09-01	98.6154
2017-10-01	93.6137
2017-11-01	97.3359
2017-12-01	114.7212

The data point is confirmed to be dropped

In [55]:
```
# Now we will plot the whole data for further inspection
plt.rcParams['figure.figsize'] = (12, 8);
x = data.index
y = data.Value
plt.plot(x,y,label = 'Energy Consumption from Jan 1985 to Jan 2018')
plt.xlabel('Time')
plt.ylabel('Energy Consumption')

```

Out[55]:
```
Text(0, 0.5, 'Energy Consumption')
```



Looking at the above plot, we can clearly identify a positive deterministic trend over time, which is expected since energy demand is increasing, then it plateaus around 2008 which might indicate that the data is taken from a developing region at the time that may have reached steady state by 2008.

We can also detect seasonality as evident from the fluctuations within a year, these fluctuations seem to grow over time, which might be modeled through a multiplicative seasonality model.

With that, we conclude that the data is non-stationary and requires decomposition to be modeled, having said that, the data seem to be well behaved, presenting itself to be modeled easily through the tools that we have at our disposal.

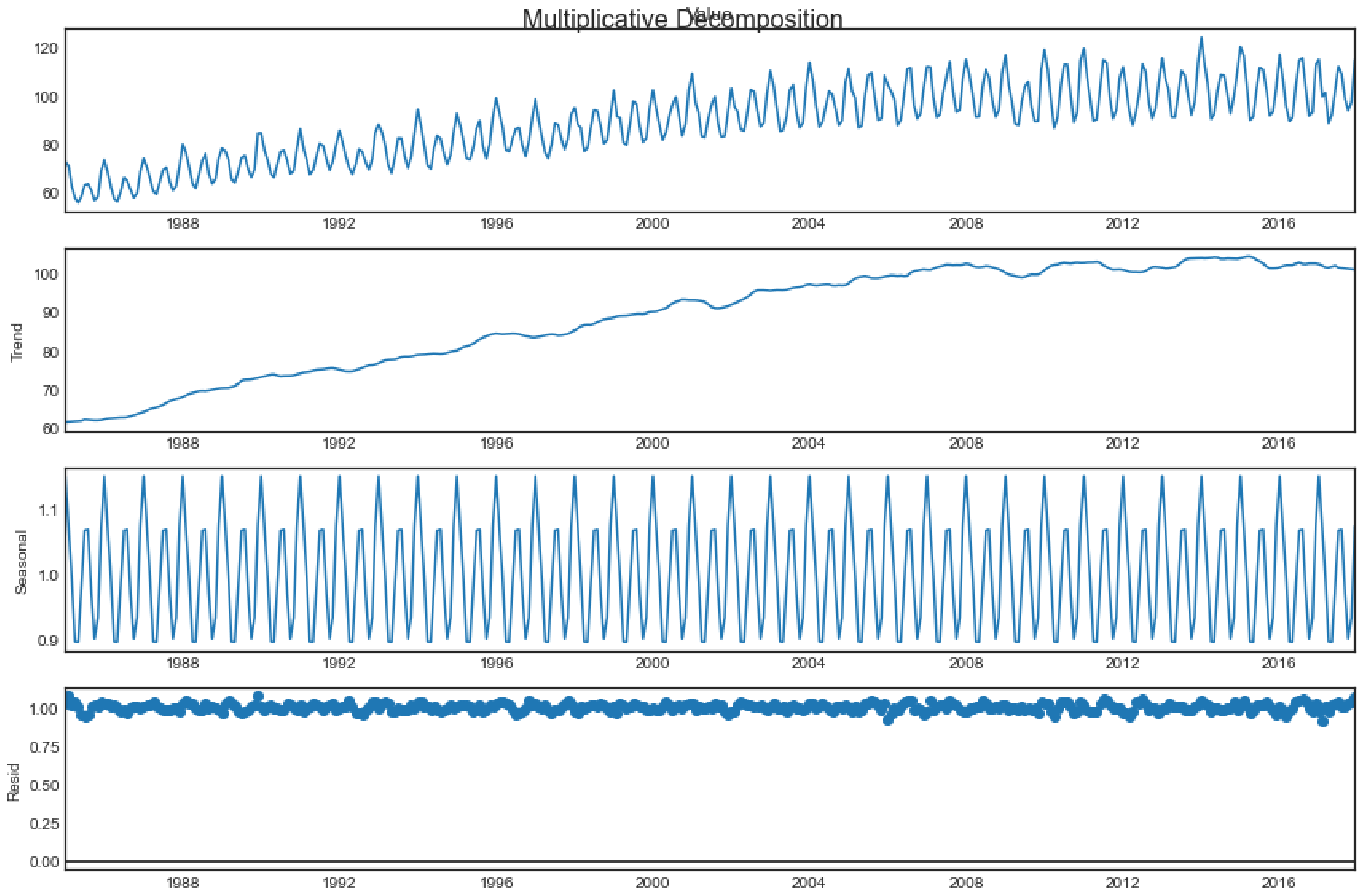
2. Evaluate the second order properties of your data.

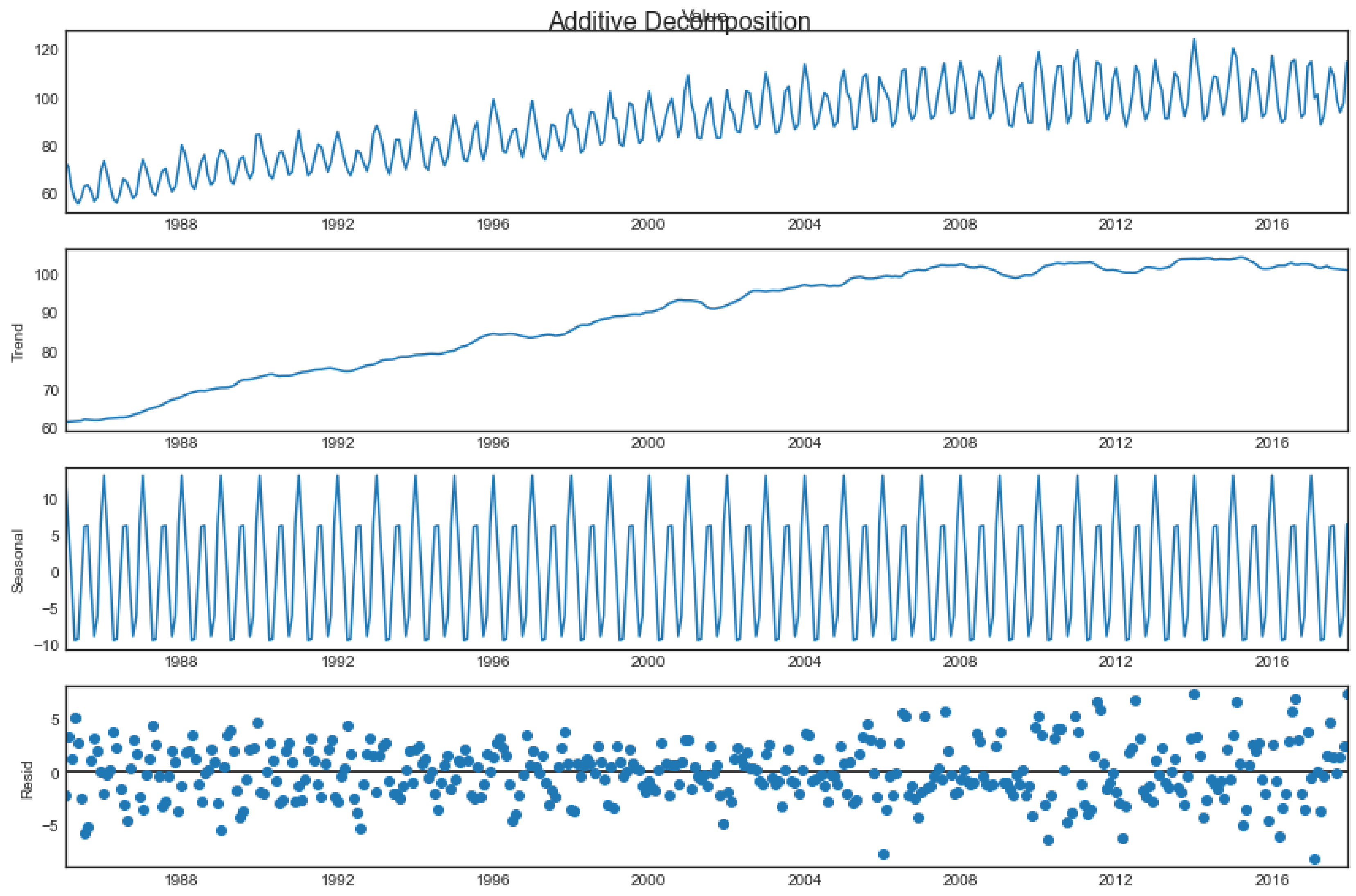
Now we will move into evaluating the second order properties of the series.

Starting with series decomposition, we will conduct a multiplicative and additive composition to gain more insight on which would be a better fit going forward.

In [8]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
#Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt
plt.style.use('seaborn-white')
%matplotlib inline

# Multiplicative Decomposition
decomposeM = seasonal_decompose(data["Value"],model='multiplicative', extrapolate_trend='freq')
plt.rcParams['figure.figsize'] = (12, 8);
#decomposeM.plot();
decomposeM.plot().suptitle('Multiplicative Decomposition', fontsize=16)


# Additive Decomposition
decomposeA = seasonal_decompose(data["Value"],model='additive', extrapolate_trend='freq')
plt.rcParams['figure.figsize'] = (12, 8);
decomposeA.plot().suptitle('Additive Decomposition', fontsize=16)
```

Out[8]:
```
Text(0.5, 0.98, 'Additive Decomposition')
```


Multiplicative Decomposition
Value
Trend
Seasonal
Resid
1988
1992
1996
2000
2004
2008
2012
2016

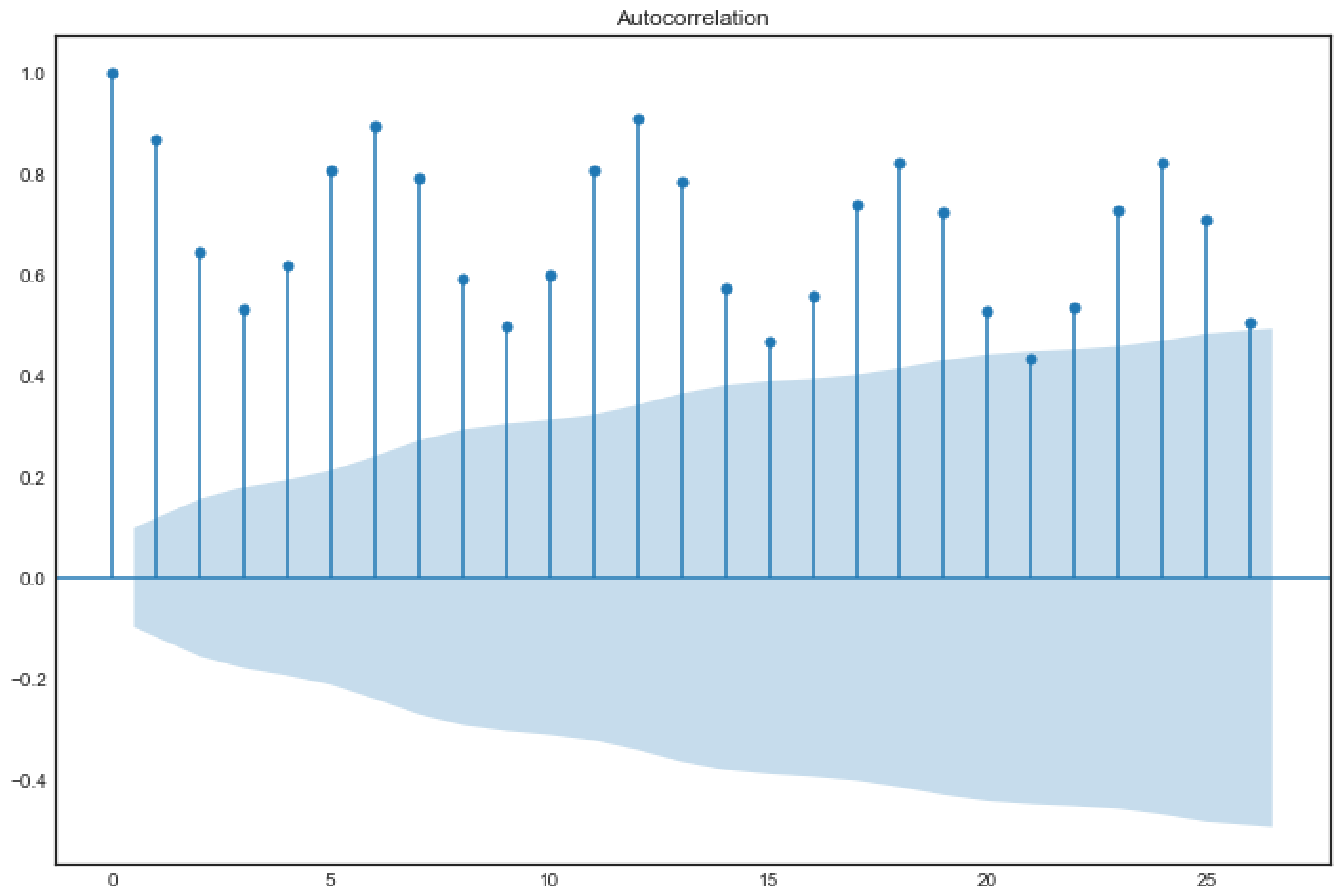


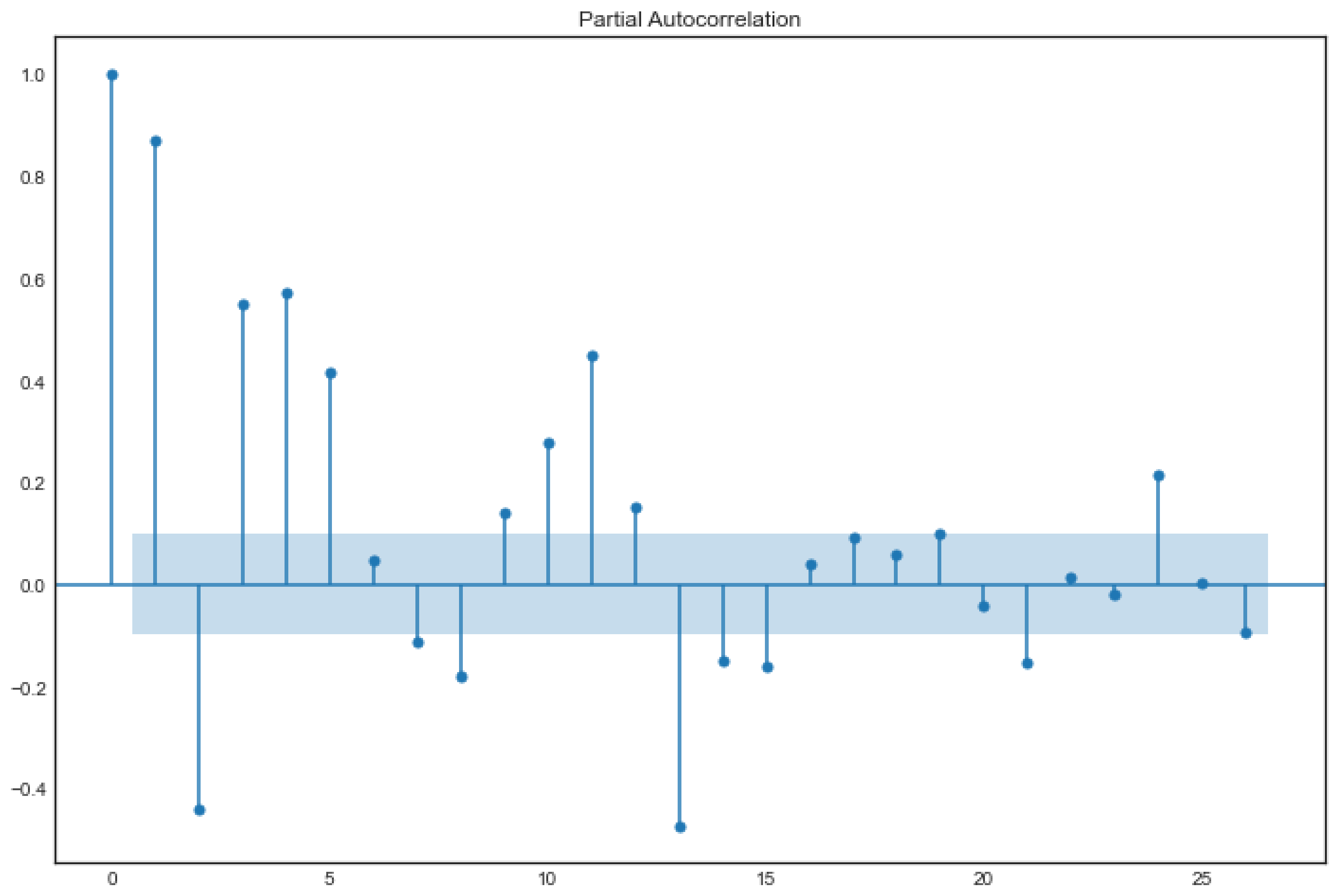
Looking at the above, it seems that our intuition of modeling multiplicative seasonality was correct, as it leaves us with well behaved residuals that can be hopefully can be easily modeled. Hence, we will multiplicatively decompose seasonality going forward.

Although we have already determined that the data is non-stationary, we can also check the ACF and PACF plots of the series or conduct an ADF test to confirm.

In [9]:
```python
plot_acf(data);
plot_pacf(data);
from statsmodels.tsa.stattools import adfuller
adf = adfuller(data["Value"])[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else ", Series is Non-Stationary")
```

```
p value:0.09680812430963193 , Series is Non-Stationary
```





Based on the above plots and test, we can clearly see that the data is not stationary

Now, we'll confirm if there are any correlations in our residuals prior to plotting and inspecting the ACF and PACF

In [10]:
```
from statsmodels.stats.diagnostic import acorr_ljungbox as ljung
ljung_p = np.mean(ljung(x=decomposeM.resid.dropna())[1]).round(3)
print("Ljung Box, p value:", ljung_p, ", Residuals are uncorrelated" if ljung_p>0.05 else ", Res
```

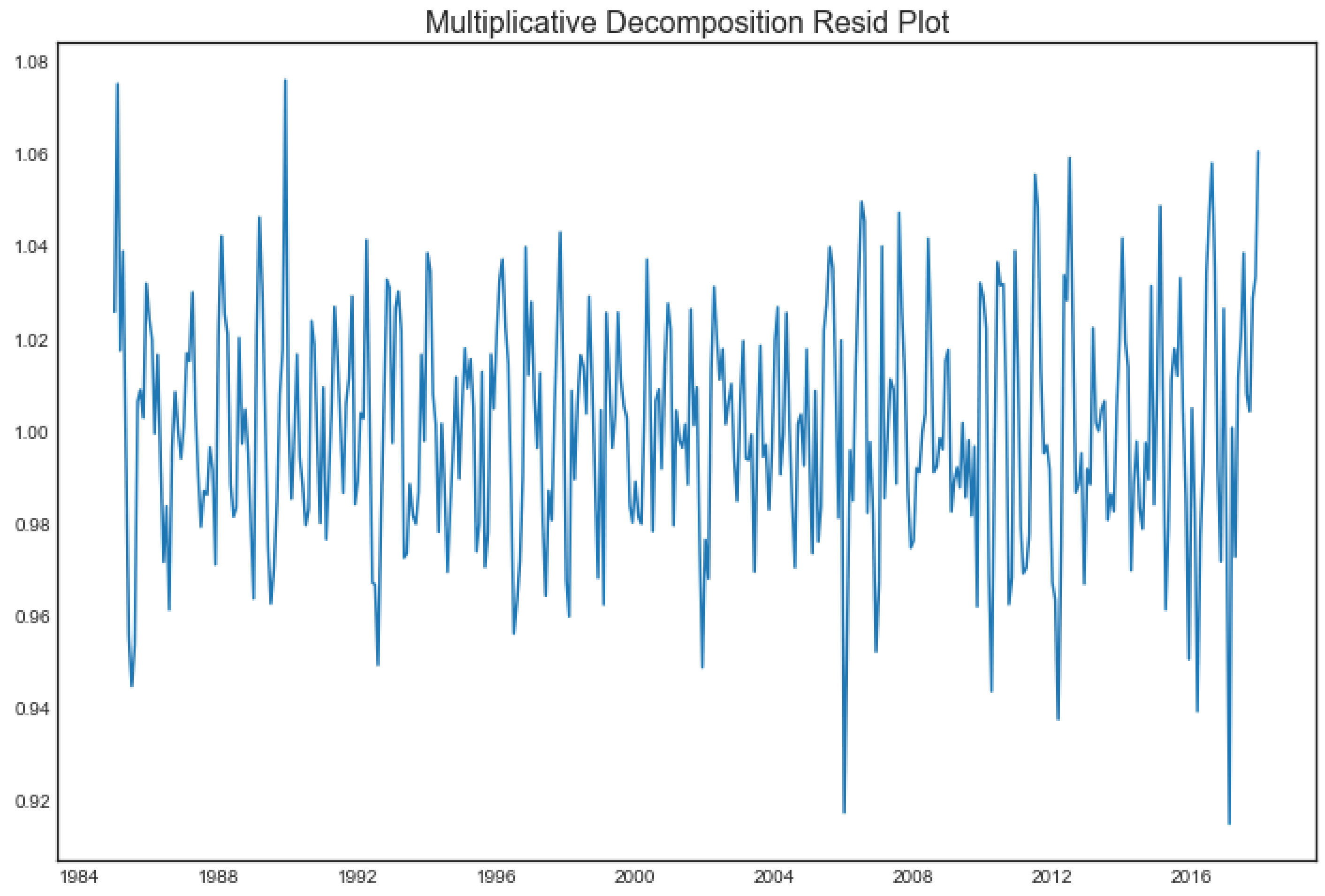
```
Ljung Box, p value: 0.0 , Residuals are correlated
```

With that confirmed, we will now detrend and adjust for seasonality to plot the ACF and PACF. If the resulting residuals are non-stationary, we will transform them to achieve stationarity.

We will now check the resid plot for stationarity after decomposition, we will compare the additive and multiplicative resids and select whichever more stationary, then apply a transformation if needed.

In [11]:
```
plt.plot(decomposeM.resid)
plt.title('Multiplicative Decomposition Resid Plot', fontsize=16)
```

Out[11]:
```
Text(0.5, 1.0, 'Multiplicative Decomposition Resid Plot')
```

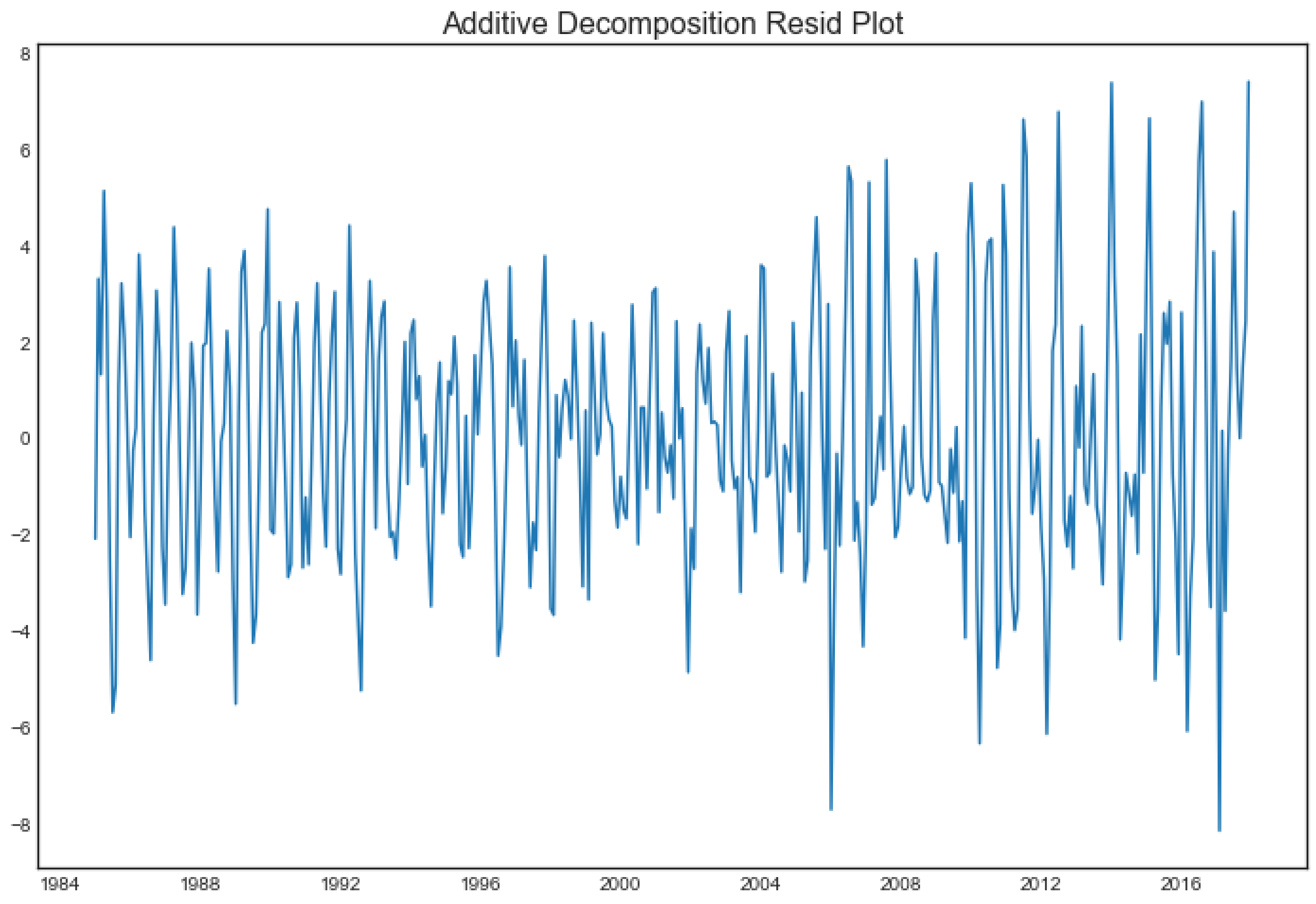


In [12]:
```
# Testing for stationarity
adf = adfuller(decomposeM.resid)[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else ", Series is Non-Stationary")
```

```
p value:9.633380306280694e-17 , Series is Stationary
```

In [13]:
```
plt.plot(decomposeA.resid)
plt.title('Additive Decomposition Resid Plot', fontsize=16)
```

Out[13]:
```
Text(0.5, 1.0, 'Additive Decomposition Resid Plot')
```

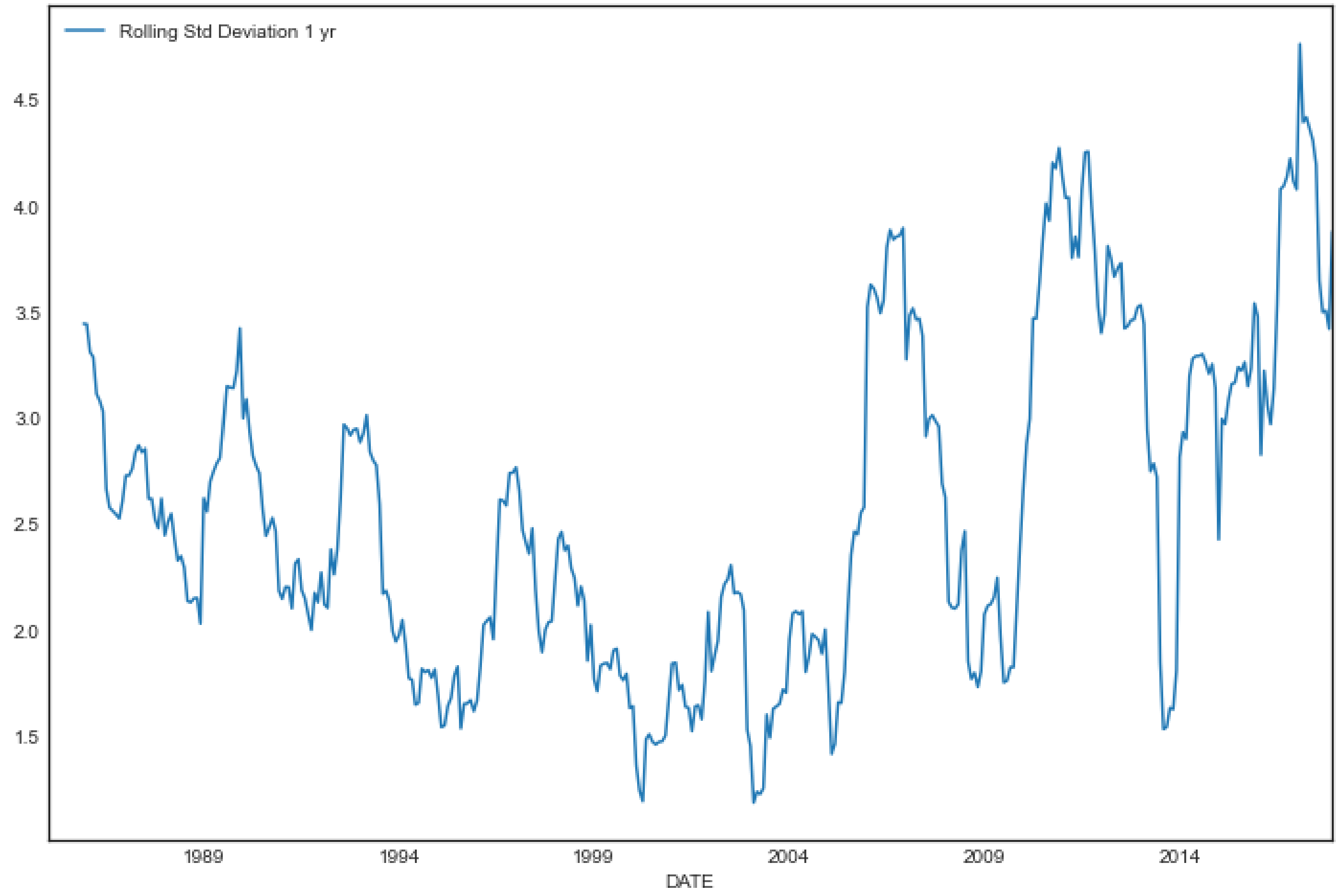


In [14]:
```python
# Testing for stationarity
adf = adfuller(decomposeA.resid)[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else ", Series is Non-Stationary")
```

```
p value:2.3006326282451945e-19 , Series is Stationary
```

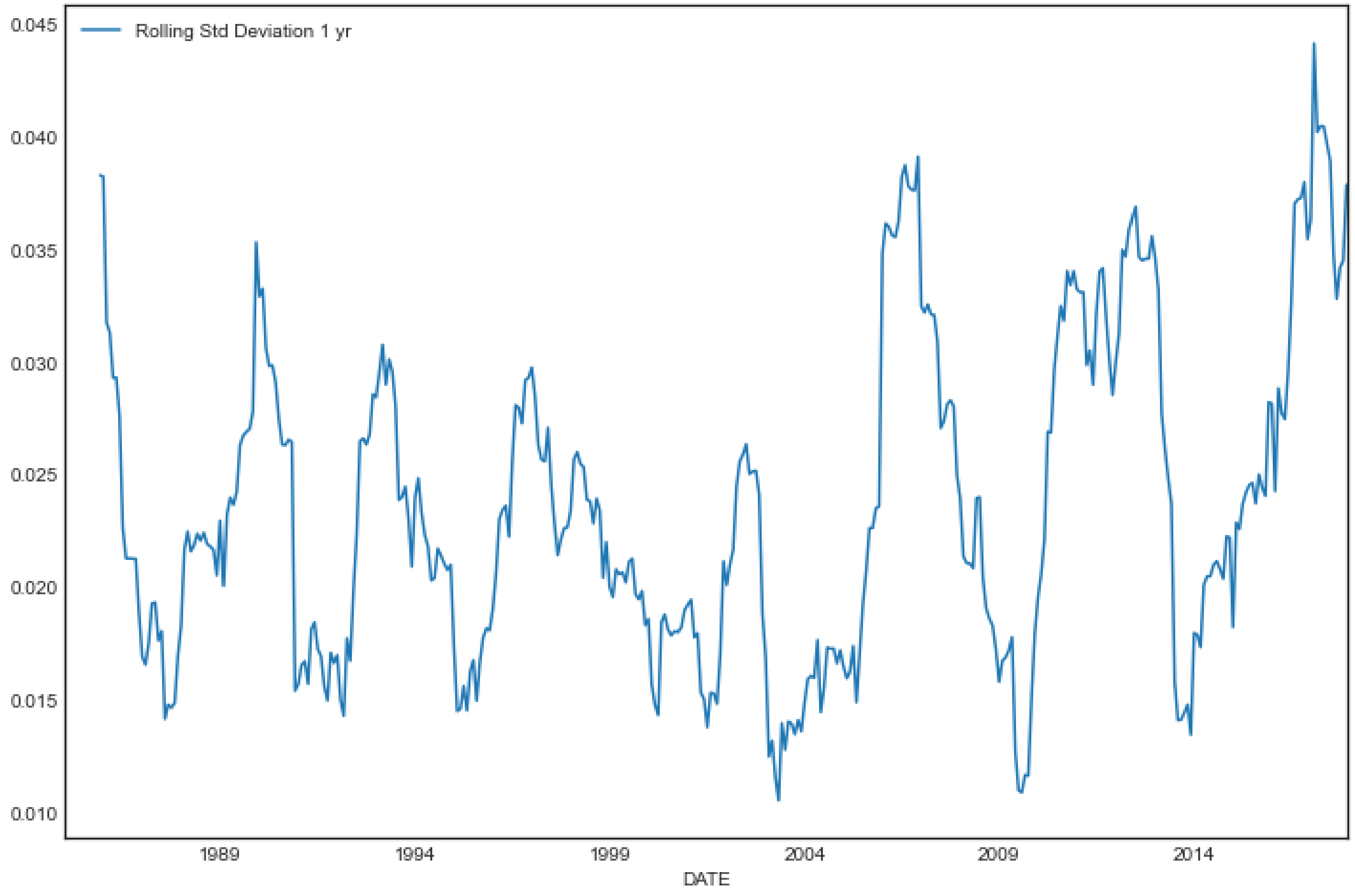
In [15]:
```python
decomposeA.resid.rolling(12).std().plot(legend=True, label="Rolling Std Deviation 1 yr");
print("S.D is:", decomposeA.resid.std())
```

```
S.D is: 2.622882234228826
```



In [16]:
```
decomposeM.resid.rolling(12).std().plot(legend=True, label="Rolling Std Deviation 1 yr");
print("S.D is:", decomposeM.resid.std())
```

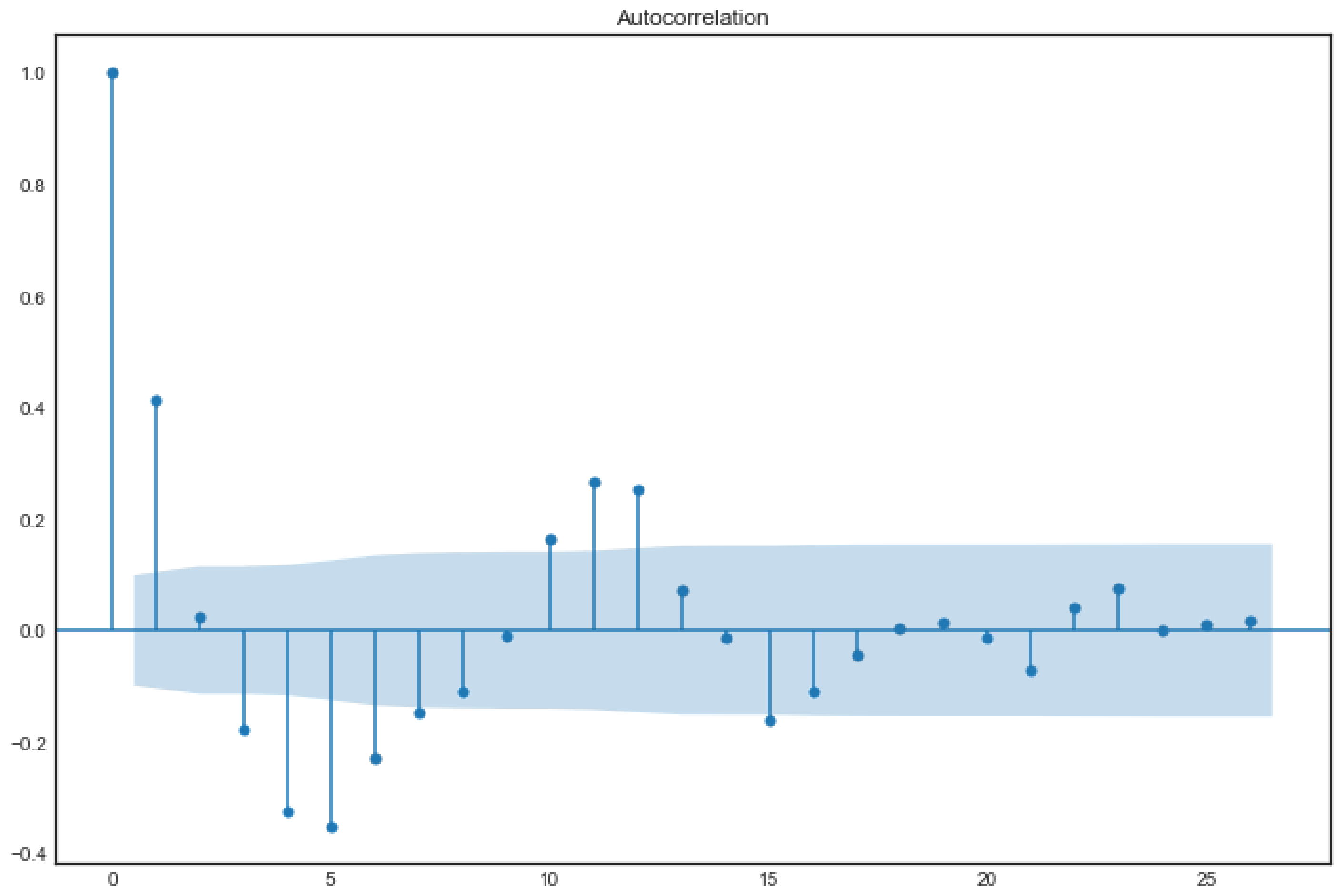
```
S.D is: 0.024378587436137850
```

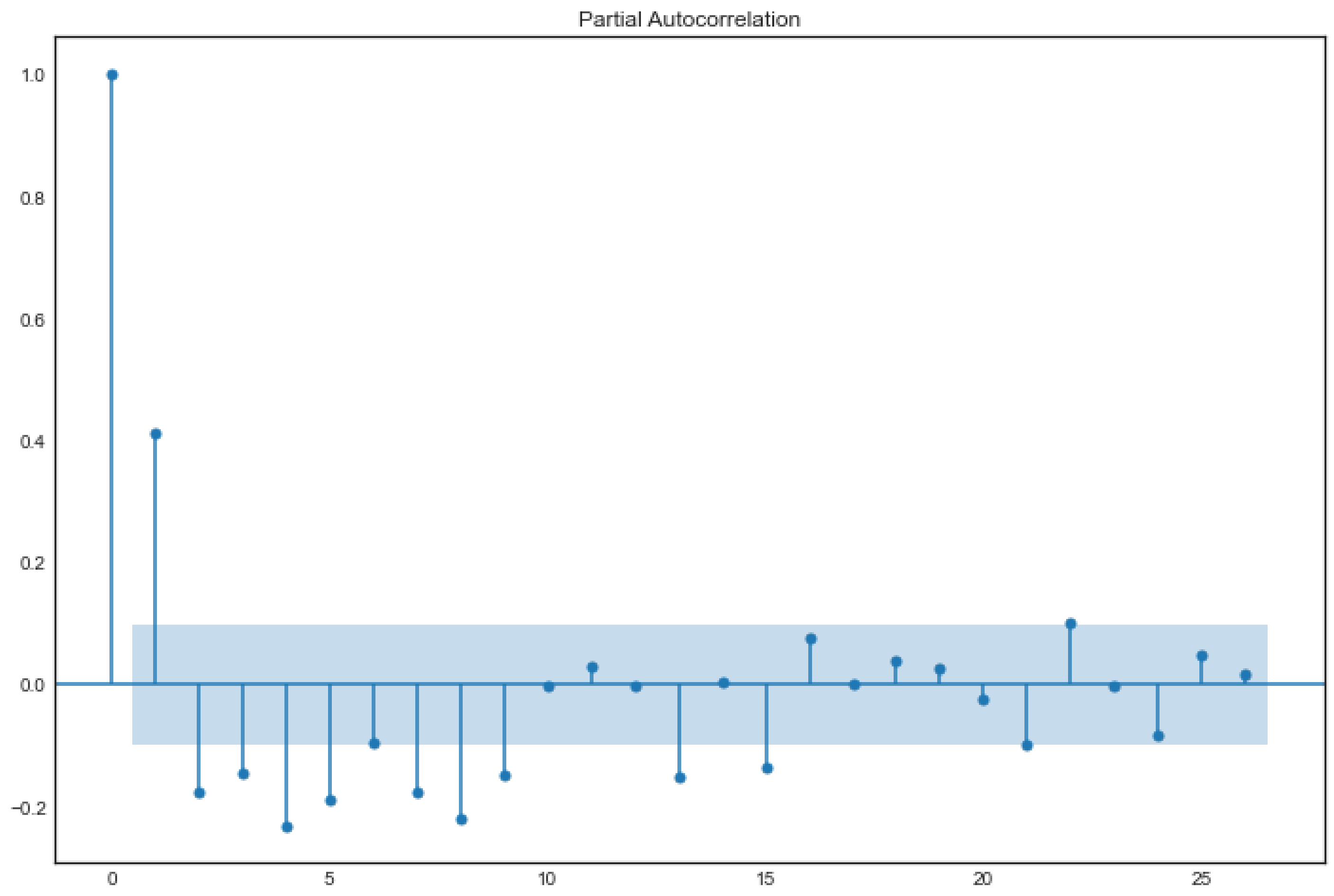


Both of which are stationary, we will just go with the multiplicative resids since they demonstrate less variation in general.

Next we will plot the ACF and PACF

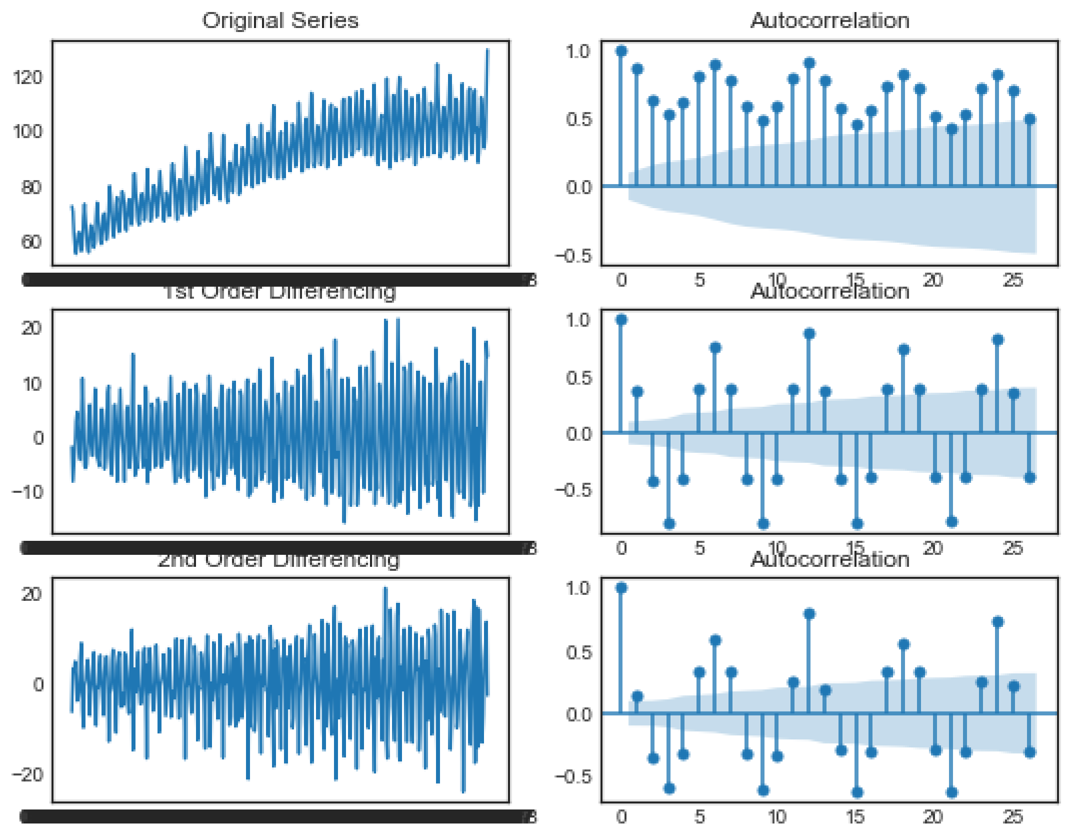
In [17]:
```
plot_acf(decomposeM.resid);
plot_pacf(decomposeM.resid);
```





Looking at the ACF and PACF, we cannot rule out AR or MA and both can still be a good fit, however, it is very evident that at least an AR component is present since ACF is decaying to zero and PACF has at least one spike at 1 with potential higher order AR.

In [18]:
```python
import numpy as np, pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt
plt.rcParams.update({'figure.figsize':(9,7)})

# Import data
df = pd.read_csv('Electric_Production.csv', names=['value'], header=0)

# Original Series
fig, axes = plt.subplots(3, 2)
axes[0, 0].plot(df.value); axes[0, 0].set_title('Original Series')
plot_acf(df.value, ax=axes[0, 1])

# 1st Differencing
axes[1, 0].plot(df.value.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(df.value.diff().dropna(), ax=axes[1, 1])

# 2nd Differencing
axes[2, 0].plot(df.value.diff().diff()); axes[2, 0].set_title('2nd Order Differencing')
plot_acf(df.value.diff().diff().dropna(), ax=axes[2, 1])

plt.show()

```

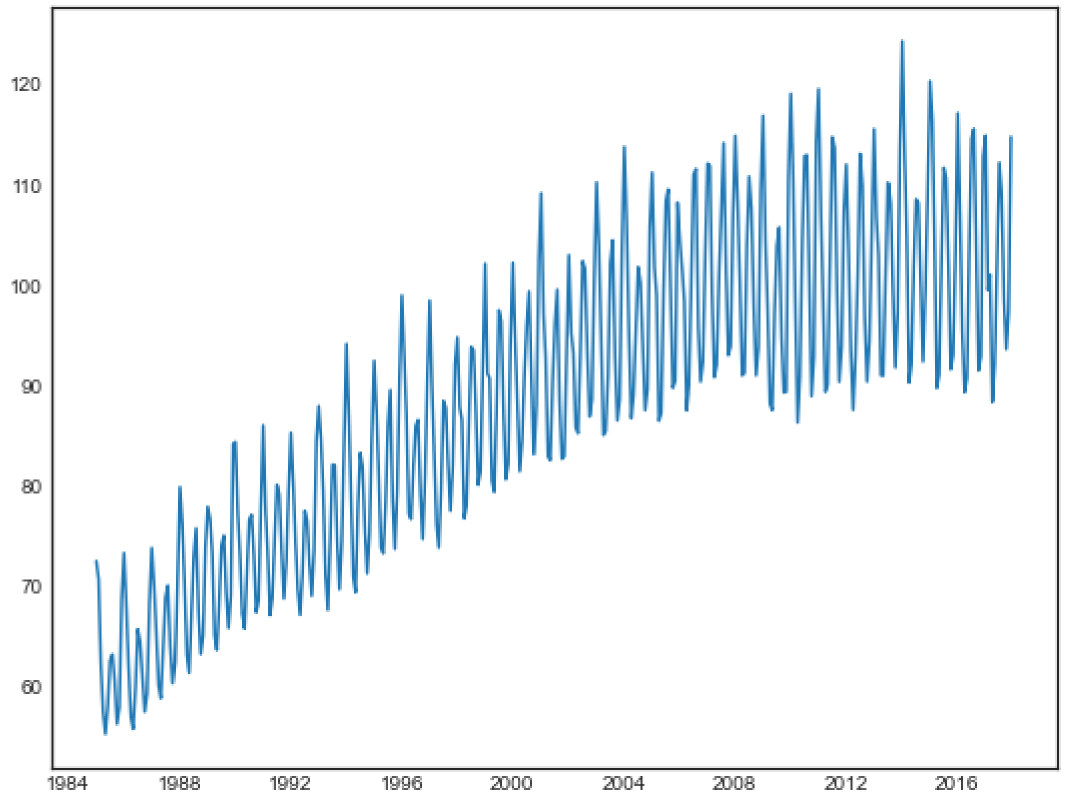


3. As a baseline model, fit an ARIMA model to each series and comment on the fit.

Next, we will use Auto-ARIMA to create a reference fit for the data

In [19]:
```
plt.plot(data)
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x2ab9e643438>]
```



In [20]:
```
smodel = pm.auto_arima(data, start_p=1, start_q=1,
                        test='adf',
                        max_p=5, max_q=5, m=12,
                        start_P=0, seasonal=True,
                        d=1, D=None, trace=True,
                        error_action='ignore',
                        suppress_warnings=True,
                        stepwise=True)

smodel.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,1)[12] intercept   : AIC=2389.183, Time=0.28 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=2740.582, Time=0.01 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=inf, Time=0.21 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=2388.128, Time=0.14 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=2738.658, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[12] intercept   : AIC=2618.005, Time=0.04 sec
 ARIMA(0,1,1)(1,0,1)[12] intercept   : AIC=1909.803, Time=0.49 sec
 ARIMA(0,1,1)(1,0,0)[12] intercept   : AIC=2028.039, Time=0.21 sec
 ARIMA(0,1,1)(2,0,1)[12] intercept   : AIC=1909.952, Time=1.67 sec
 ARIMA(0,1,1)(1,0,2)[12] intercept   : AIC=1908.490, Time=1.53 sec
 ARIMA(0,1,1)(0,0,2)[12] intercept   : AIC=2303.224, Time=0.65 sec
 ARIMA(0,1,1)(2,0,2)[12] intercept   : AIC=1910.304, Time=2.15 sec
 ARIMA(0,1,0)(1,0,2)[12] intercept   : AIC=1937.276, Time=0.94 sec
 ARIMA(1,1,1)(1,0,2)[12] intercept   : AIC=1850.680, Time=2.31 sec
 ARIMA(1,1,1)(0,0,2)[12] intercept   : AIC=2219.923, Time=0.85 sec
 ARIMA(1,1,1)(1,0,1)[12] intercept   : AIC=1849.712, Time=0.87 sec
 ARIMA(1,1,1)(1,0,0)[12] intercept   : AIC=1961.294, Time=0.75 sec
 ARIMA(1,1,1)(2,0,1)[12] intercept   : AIC=1855.483, Time=2.23 sec
 ARIMA(1,1,1)(0,0,0)[12] intercept   : AIC=2619.215, Time=0.06 sec
 ARIMA(1,1,1)(2,0,0)[12] intercept   : AIC=1925.911, Time=1.30 sec
 ARIMA(1,1,1)(2,0,2)[12] intercept   : AIC=1938.673, Time=2.40 sec
 ARIMA(1,1,0)(1,0,1)[12] intercept   : AIC=1926.498, Time=0.37 sec
 ARIMA(2,1,1)(1,0,1)[12] intercept   : AIC=1852.579, Time=0.91 sec
 ARIMA(1,1,2)(1,0,1)[12] intercept   : AIC=1851.873, Time=0.81 sec
 ARIMA(0,1,0)(1,0,1)[12] intercept   : AIC=1937.515, Time=0.28 sec
 ARIMA(0,1,2)(1,0,1)[12] intercept   : AIC=1853.175, Time=0.72 sec
 ARIMA(2,1,0)(1,0,1)[12] intercept   : AIC=1900.610, Time=0.53 sec
 ARIMA(2,1,2)(1,0,1)[12] intercept   : AIC=inf, Time=0.88 sec
 ARIMA(1,1,1)(1,0,1)[12]             : AIC=1847.645, Time=0.44 sec
 ARIMA(1,1,1)(0,0,1)[12]             : AIC=2387.231, Time=0.16 sec
 ARIMA(1,1,1)(1,0,0)[12]             : AIC=1962.753, Time=0.18 sec
 ARIMA(1,1,1)(2,0,1)[12]             : AIC=1848.088, Time=1.03 sec
 ARIMA(1,1,1)(1,0,2)[12]             : AIC=1846.845, Time=1.08 sec
 ARIMA(1,1,1)(0,0,2)[12]             : AIC=2226.431, Time=0.57 sec
 ARIMA(1,1,1)(2,0,2)[12]             : AIC=1937.222, Time=2.05 sec
 ARIMA(0,1,1)(1,0,2)[12]             : AIC=1906.497, Time=0.89 sec
 ARIMA(1,1,0)(1,0,2)[12]             : AIC=1924.427, Time=0.66 sec
 ARIMA(2,1,1)(1,0,2)[12]             : AIC=1845.425, Time=1.16 sec
 ARIMA(2,1,1)(0,0,2)[12]             : AIC=2096.757, Time=0.48 sec
 ARIMA(2,1,1)(1,0,1)[12]             : AIC=1846.350, Time=0.48 sec
 ARIMA(2,1,1)(2,0,2)[12]             : AIC=1842.678, Time=2.25 sec
 ARIMA(2,1,1)(2,0,1)[12]             : AIC=1846.718, Time=1.48 sec
 ARIMA(2,1,0)(2,0,2)[12]             : AIC=1895.146, Time=1.87 sec
 ARIMA(3,1,1)(2,0,2)[12]             : AIC=1843.955, Time=2.57 sec
 ARIMA(2,1,2)(2,0,2)[12]             : AIC=1896.346, Time=2.71 sec
 ARIMA(1,1,0)(2,0,2)[12]             : AIC=1922.682, Time=1.89 sec
 ARIMA(1,1,2)(2,0,2)[12]             : AIC=1841.907, Time=2.01 sec
 ARIMA(1,1,2)(1,0,2)[12]             : AIC=1844.440, Time=1.37 sec
 ARIMA(1,1,2)(2,0,1)[12]             : AIC=1845.857, Time=1.33 sec
 ARIMA(1,1,2)(1,0,1)[12]             : AIC=1845.683, Time=0.30 sec
 ARIMA(0,1,2)(2,0,2)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,1,3)(2,0,2)[12]             : AIC=inf, Time=2.74 sec
 ARIMA(0,1,1)(2,0,2)[12]             : AIC=1905.153, Time=1.75 sec
 ARIMA(0,1,3)(2,0,2)[12]             : AIC=1843.595, Time=2.64 sec
 ARIMA(2,1,3)(2,0,2)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,1,2)(2,0,2)[12] intercept   : AIC=2090.120, Time=1.77 sec

Best model:  ARIMA(1,1,2)(2,0,2)[12]
Total fit time: 62.927 seconds
```

Out[20]:

SARIMAX Results

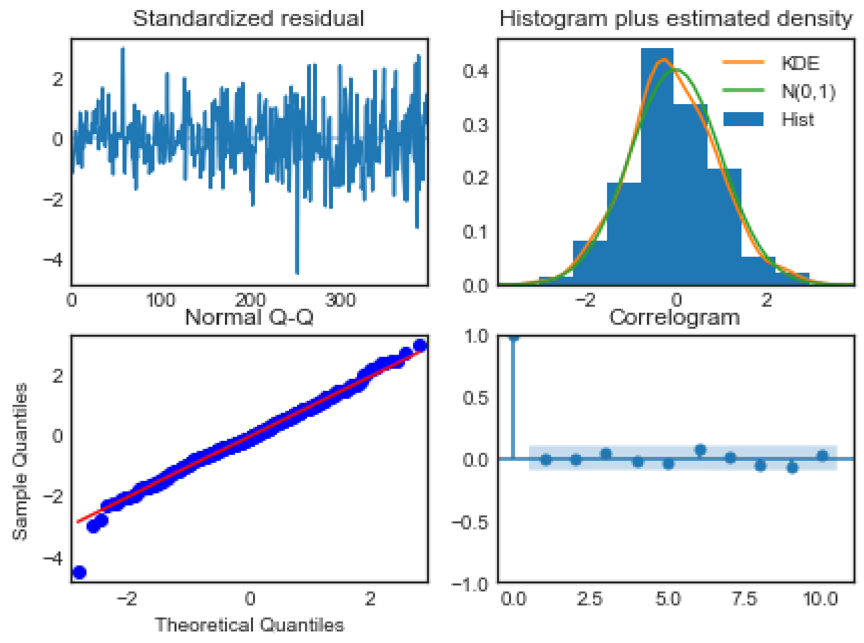
Dep. Variable:	y	**No. Observations:**	396
Model:	SARIMAX(1, 1, 2)x(2, 0, 2, 12)	**Log Likelihood**	-912.953
Date:	Thu, 02 Dec 2021	**AIC**	1841.907
Time:	01:28:31	**BIC**	1873.738
Sample:	0	**HQIC**	1854.518
	- 396		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	0.3171	0.094	3.388	0.001	0.134	0.501
ma.L1	-0.7045	0.091	-7.757	0.000	-0.883	-0.527
ma.L2	-0.2284	0.078	-2.936	0.003	-0.381	-0.076
ar.S.L12	0.5480	0.267	2.049	0.040	0.024	1.072
ar.S.L24	0.4460	0.265	1.682	0.093	-0.074	0.966
ma.S.L12	-0.1992	0.261	-0.762	0.446	-0.712	0.313
ma.S.L24	-0.4620	0.179	-2.574	0.010	-0.814	-0.110
sigma2	5.4623	0.366	14.943	0.000	4.746	6.179

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	11.66
Prob(Q):	0.88	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.74	**Skew:**	-0.09
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.82

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

In [21]:
```
smodel.plot_diagnostics(figsize=(7,5))
plt.show()
```



Looking at the above graphs, the residuals seem to have a mean of zero with well stable variance, the histogram and q-q plot suggest normally distributed data which indicates a well behaved distribution, Also the correlogram (ACF plot) indicates no correlations which means we're left with white noise and that's what we desire and expect.

We can conclude that this is a good ARIMA fit for our data, and we can use it as reference for the next steps where we will manually create our own ARIMA and compare it to this fit

4. Fit a model that includes, trend, seasonality and cyclical components.

Next we will manually decompose the data and create our own fit

In [22]:

```
# Extract the Components ----
# Actual Values = Product of (Seasonal * Trend * Resid)
data_r = pd.concat([decomposeM.seasonal, decomposeM.trend, decomposeM.resid, decomposeM.observed
data_r.columns = ['seas', 'trend', 'resid', 'actual_values']
data_r.head()
```

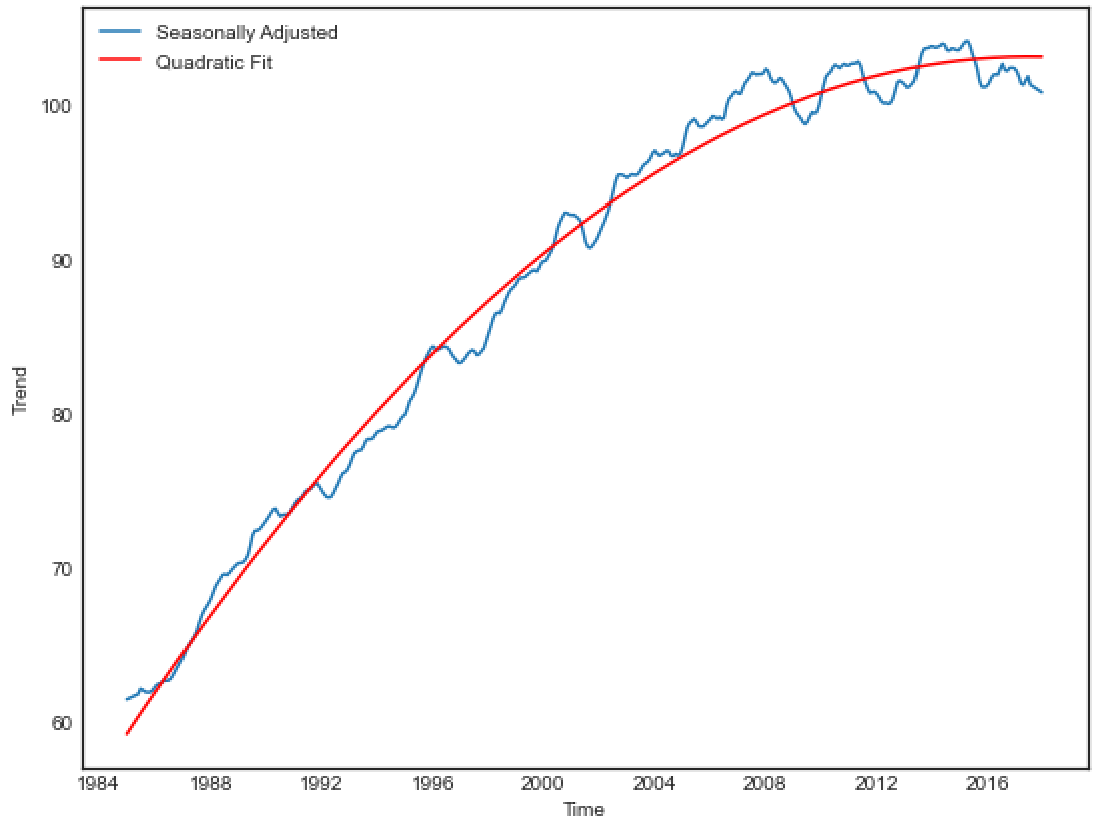
Out[22]:

	seas	trend	resid	actual_values
DATE				
1985-01-01	1.149302	61.508657	1.025649	72.5052
1985-02-01	1.067642	61.577679	1.074975	70.6720
1985-03-01	0.995838	61.646701	1.017268	62.4502
1985-04-01	0.896496	61.715723	1.038742	57.4714
1985-05-01	0.896083	61.784745	0.999112	55.3151

First we will use the decomposed trend data to find a good fit for the trend

In [23]:
```
# let's use ploynomial for our trend as prices are positively correlated with time in the long r

# Fit a quadratic trend
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

x_ts =data_r.index
y_ts = data_r.trend
n = np.shape(y_ts)[0]

# Quadratic Fit
t = range(n)
p = np.polyfit(t,y_ts,2)
y = np.polyval(p,t)
plt.plot(x_ts,y_ts,label='Seasonally Adjusted ')
plt.plot(x_ts,y,color='r',label ='Quadratic Fit')
plt.xlabel('Time')
plt.ylabel('Trend')
plt.legend()
print('The coefficients of the trend are: B0 ='+str(p[2])+' B1 = '+ str(p[1]) + ' B2 = '+str(p[0
```

```
The coefficients of the trend are: B0 =59.26090171774182 B1 = 0.22531504890843215 B2 = -0.00028913
82539889242
```



We can also do it through regressing the values on a second order time variable to confirm.

In [24]:
```
data_t = data.copy()
```

In [25]:
```
data_t["t"] = np.arange(1, len(data)+1)
X = sm.add_constant(data_t[["t"]])
X["t^2"] = data_t[["t"]]**2
Y = data_t[["Value"]]
mtrend = sm.OLS(Y, X).fit()
mtrend.summary()
```

Out[25]:

OLS Regression Results

Dep. Variable:	Value	**R-squared:**	0.739
Model:	OLS	**Adj. R-squared:**	0.738
Method:	Least Squares	**F-statistic:**	557.7
Date:	Thu, 02 Dec 2021	**Prob (F-statistic):**	1.67e-115
Time:	01:28:32	**Log-Likelihood:**	-1374.6
No. Observations:	396	**AIC:**	2755.
Df Residuals:	393	**BIC:**	2767.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	59.2796	1.184	50.063	0.000	56.952	61.608
t	0.2229	0.014	16.185	0.000	0.196	0.250
t^2	-0.0003	3.36e-05	-8.387	0.000	-0.000	-0.000

Omnibus:	31.872	**Durbin-Watson:**	0.984
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	14.509
Skew:	0.260	**Prob(JB):**	0.000707
Kurtosis:	2.220	**Cond. No.**	2.12e+05

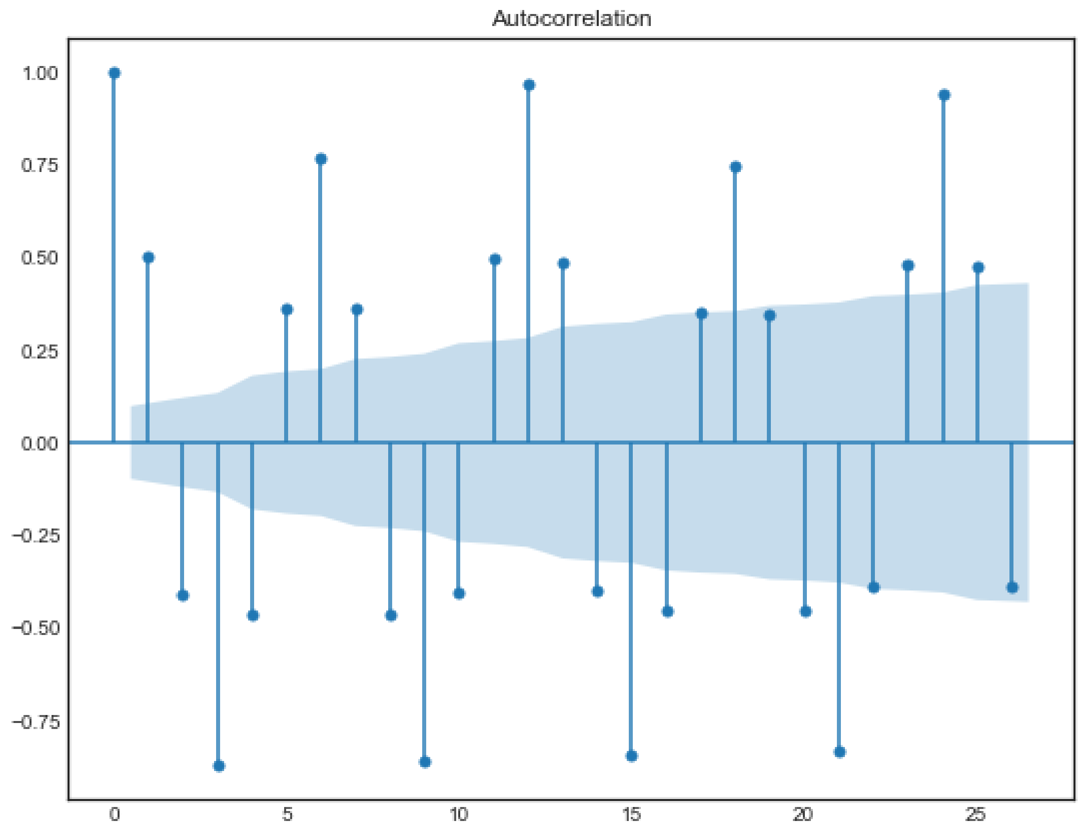
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.12e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

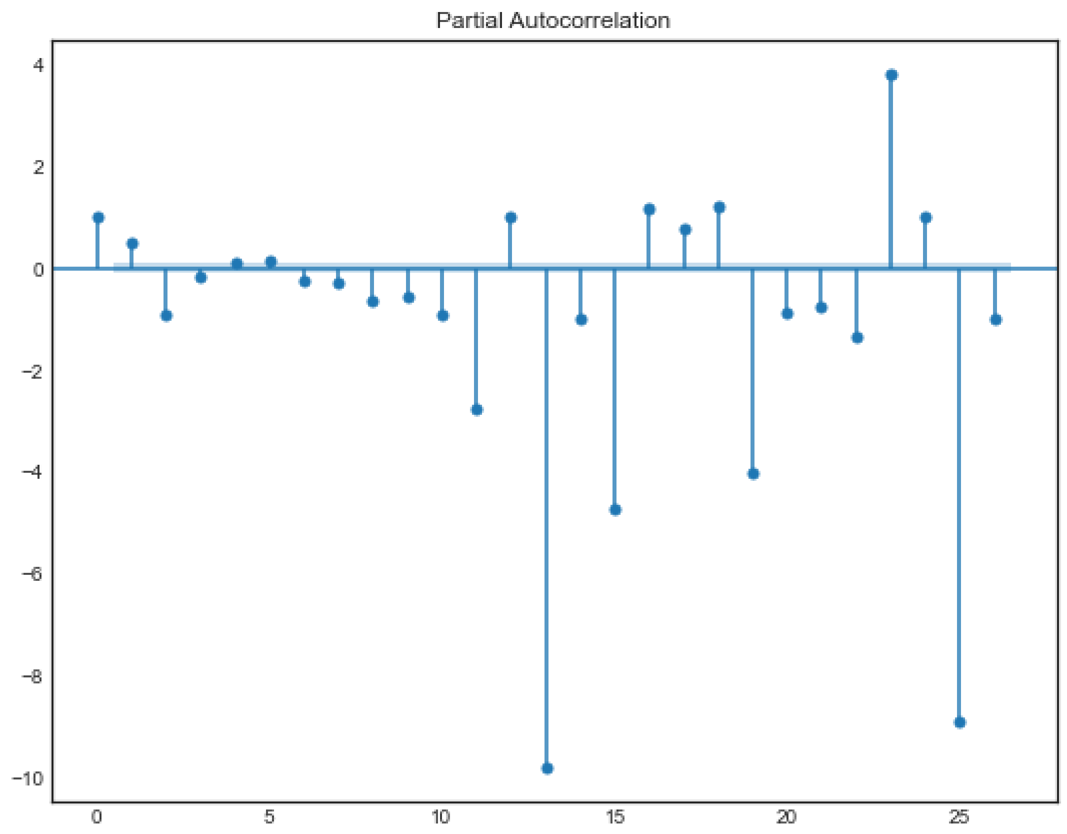
The regression results seem consistent with our polyfit results and clearly statistically significant. With that we have modeled our trend component and acquired the coefficients.

Now we will fit the seasonality, by reading the ACF and PACF of the seasonal component and deciding on the ARMIA model to go with

In [26]:
```
from statsmodels.tsa.statespace.tools import diff
dseas = diff(data_r['seas'],k_diff=12)
```

In [27]:
```
plot_acf(data_r['seas']);
plot_pacf(data_r['seas']);
```





From the above ACF and PACF plots it clear that there are AR behavior since the ACF is seasonally fluctuating and there are seasonal spikes on the PACF. However, it is maybe difficult to read the order from the PACF, but there seems to be a spike at the 24 mark which indicates at least and S-AR(2) which will be modeled next and tested.

In [28]:

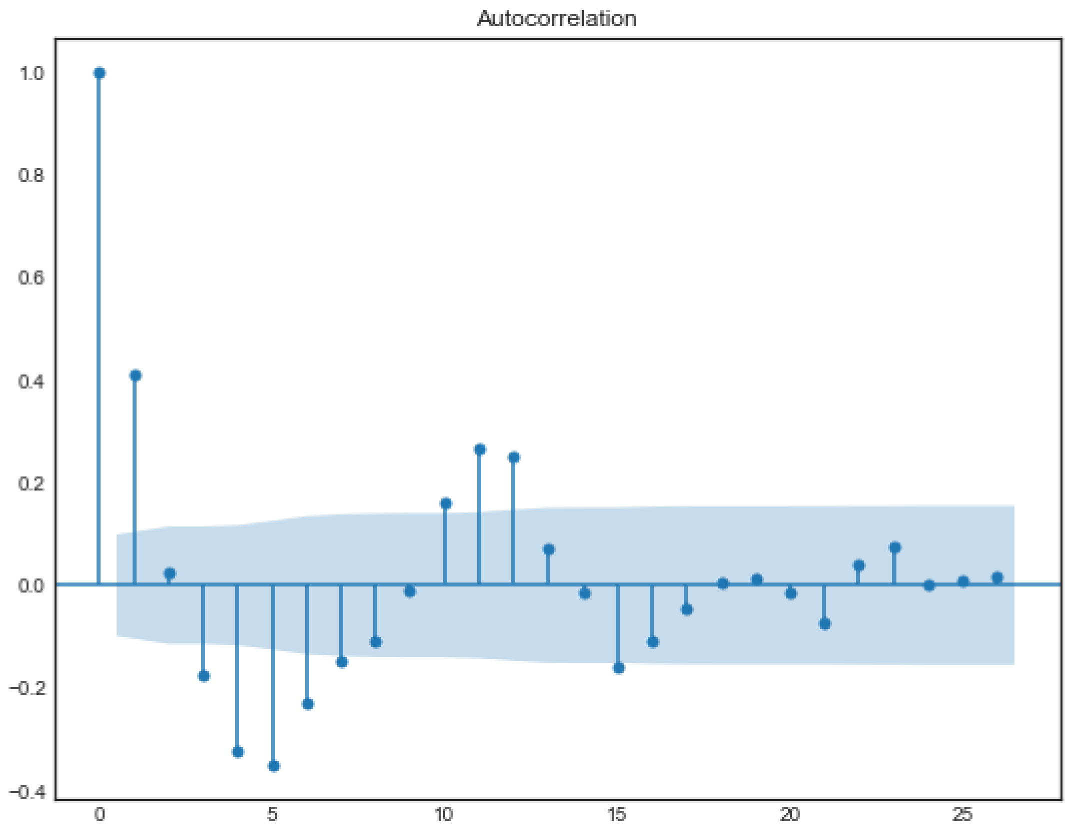
```python
from statsmodels.tsa.arima.model import ARIMA
model_s = ARIMA(data_r["seas"], order=(2,0,0))
fitted_s = model_s.fit()
fitted_s.summary()
```

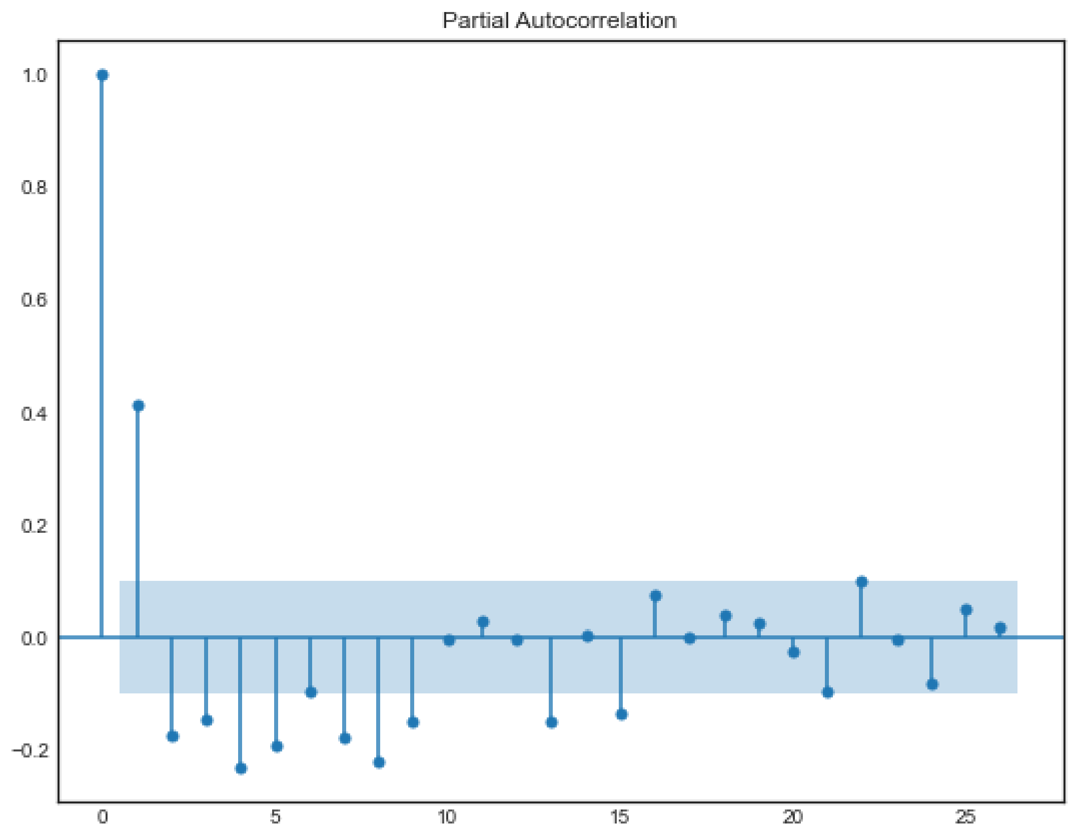
Out[28]:

```
SARIMAX Results

   Dep. Variable:            seas  No. Observations:       396
          Model:     ARIMA(2, 0, 0)   Log Likelihood    834.025
           Date:    Thu, 02 Dec 2021             AIC  -1660.049
           Time:           01:28:32              BIC  -1644.123
         Sample:         01-01-1985             HQIC  -1653.740
                       - 12-01-2017
Covariance Type:                opg

            coef   std err         z   P>|z|   [0.025   0.975]
  const   1.0000     0.002   458.816   0.000    0.996    1.004
  ar.L1   0.9694     0.042    23.047   0.000    0.887    1.052
  ar.L2  -0.9063     0.025   -36.558   0.000   -0.955   -0.858
 sigma2   0.0009  8.22e-05    10.447   0.000    0.001    0.001

    Ljung-Box (L1) (Q):  4.34  Jarque-Bera (JB):  62.43
               Prob(Q):  0.04          Prob(JB):   0.00
Heteroskedasticity (H):  0.98              Skew:   0.97
  Prob(H) (two-sided):   0.90          Kurtosis:   3.11


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

We will now fit a model for the cycles in the same manner as the seasons, by first looking at the ACF & PACF and then select the best model

In [29]:
```
plot_acf(data_r['resid']);
plot_pacf(data_r['resid']);
```





From our earlier observation, since ACF is decaying while there is a clear spike at 1 on the PACF, the proposed model for the cycles will be AR(1)

In [30]:
```python
from statsmodels.tsa.arima.model import ARIMA
model_c = ARIMA(data_r["resid"], order=(1,0,0))
fitted_c = model_c.fit()
fitted_c.summary()

```

Out[30]:

SARIMAX Results

Dep. Variable:	resid	**No. Observations:**	396
Model:	ARIMA(1, 0, 0)	**Log Likelihood**	946.880
Date:	Thu, 02 Dec 2021	**AIC**	-1887.759
Time:	01:28:32	**BIC**	-1875.815
Sample:	01-01-1985	**HQIC**	-1883.027
	- 12-01-2017		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	1.0007	0.002	522.906	0.000	0.997	1.004
ar.L1	0.4186	0.046	9.125	0.000	0.329	0.508
sigma2	0.0005	3.1e-05	15.828	0.000	0.000	0.001

Ljung-Box (L1) (Q):	2.09	**Jarque-Bera (JB):**	4.85
Prob(Q):	0.15	**Prob(JB):**	0.09
Heteroskedasticity (H):	1.45	**Skew:**	-0.05
Prob(H) (two-sided):	0.04	**Kurtosis:**	3.53

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

With that, we have all components required to reconstruct the model and test it. We will compile an ARIMA model with (1,0,0) Cycles a polynomial trend with coefficients of B0 =59.26090171774182 B1 = 0.22531504890843215 B2 = -0.0002891382539889242 and a (2,0,0) seasonal component.

In [31]:
```
model_F = pm.ARIMA(order=(1,0,0),seasonal_order=(2,0,0,12),trend=[59.26,0.225,-0.000289])
fitted_F = model_F.fit(data)
fitted_F.plot_diagnostics()
fitted_F.summary()
```

Out[31]:

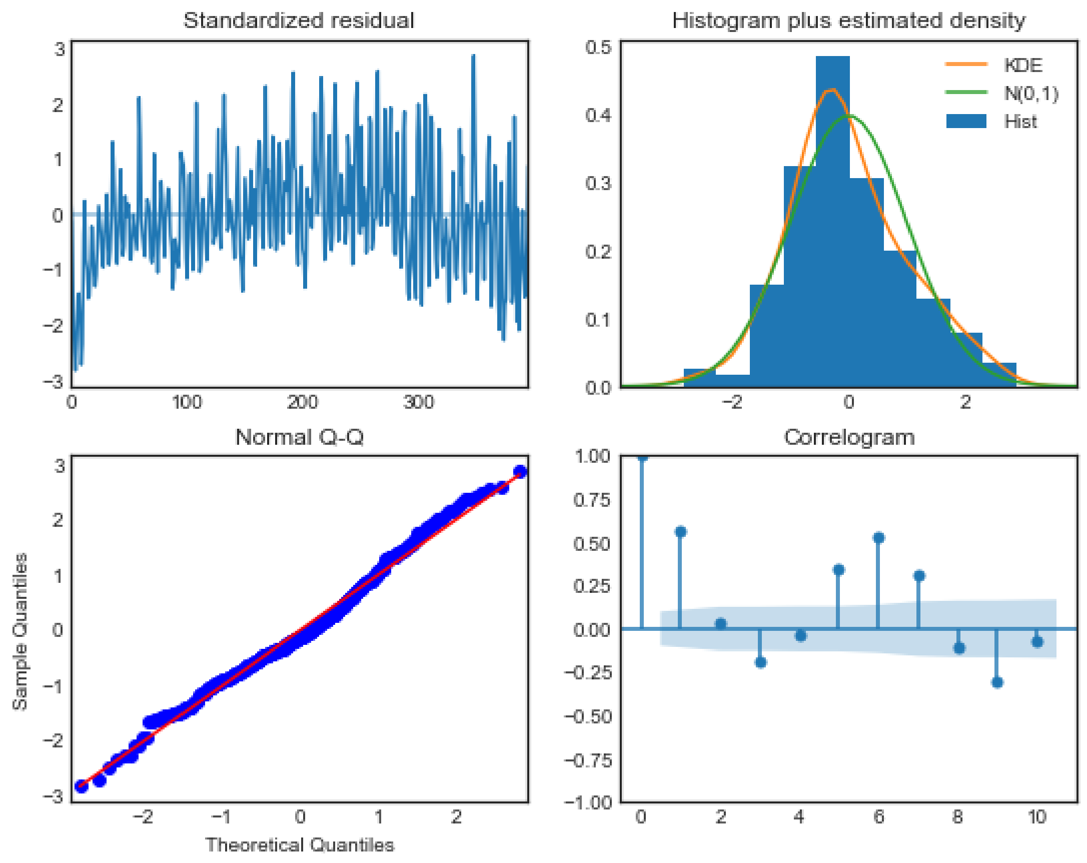
SARIMAX Results

Dep. Variable:	y	**No. Observations:**	396
Model:	SARIMAX(1, 0, 0)x(2, 0, 0, 12)	**Log Likelihood**	-1150.877
Date:	Thu, 02 Dec 2021	**AIC**	2313.754
Time:	01:28:34	**BIC**	2337.642
Sample:	0	**HQIC**	2323.218
	- 396		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	27.2936	2.057	13.266	0.000	23.261	31.326
drift	0.0373	0.003	11.303	0.000	0.031	0.044
ar.L1	0.0858	0.052	1.638	0.101	-0.017	0.189
ar.S.L12	0.5453	0.061	8.899	0.000	0.425	0.665
ar.S.L24	0.0356	0.061	0.586	0.558	-0.084	0.155
sigma2	18.9632	1.860	10.195	0.000	15.318	22.609

Ljung-Box (L1) (Q):	126.85	**Jarque-Bera (JB):**	6.03
Prob(Q):	0.00	**Prob(JB):**	0.05
Heteroskedasticity (H):	1.58	**Skew:**	0.30
Prob(H) (two-sided):	0.01	**Kurtosis:**	3.05

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



Looking at our model, it seems that the second order seasonal component may have not been needed as it's statistically insignificant and may have adversely impacted our model. Also, The ACF and residuals do not seem to be good enough, as there seem to be spikes on the ACF and the residuals have a clear structure. We will confirm by reducing the order from 2 to 1 and compare the fits.

In [32]:
```
model_F = pm.ARIMA(order=(1,0,0),seasonal_order=(1,0,0,12),trend=[59.26,0.225,-0.000289])
fitted_F = model_F.fit(data)
fitted_F.plot_diagnostics()
fitted_F.summary()
```

Out[32]:

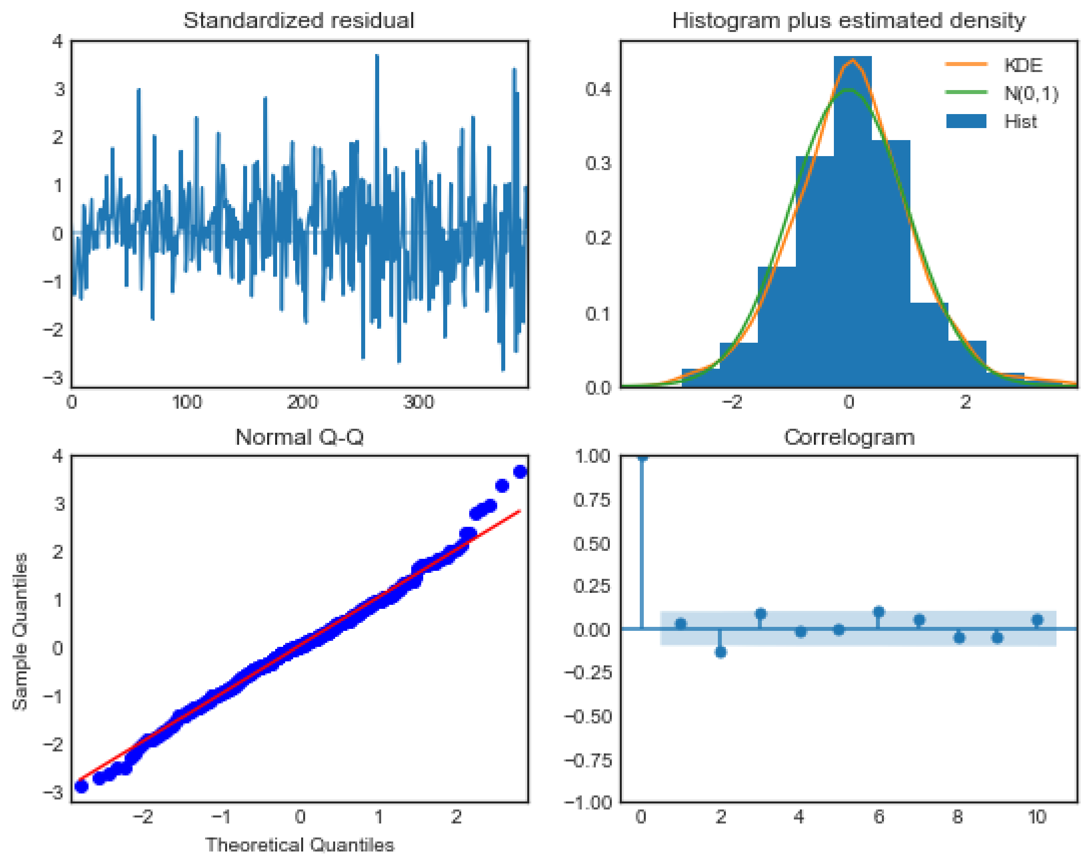
SARIMAX Results

Dep. Variable:	y	**No. Observations:**	396
Model:	SARIMAX(1, 0, 0)x(1, 0, 0, 12)	**Log Likelihood**	-982.418
Date:	Thu, 02 Dec 2021	**AIC**	1974.836
Time:	01:28:35	**BIC**	1994.743
Sample:	0	**HQIC**	1982.723
	- 396		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	3.0229	0.647	4.669	0.000	1.754	4.292
drift	0.0054	0.001	4.175	0.000	0.003	0.008
ar.L1	0.5790	0.037	15.661	0.000	0.507	0.651
ar.S.L12	0.8978	0.020	44.933	0.000	0.859	0.937
sigma2	8.0052	0.500	16.001	0.000	7.025	8.986

Ljung-Box (L1) (Q):	0.39	**Jarque-Bera (JB):**	9.59
Prob(Q):	0.53	**Prob(JB):**	0.01
Heteroskedasticity (H):	2.60	**Skew:**	0.13
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.72

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



The above shows the full fitted model with all parameters being statistically significant after combining all components.

Now we will plot our fitted model vs our actual series to compare

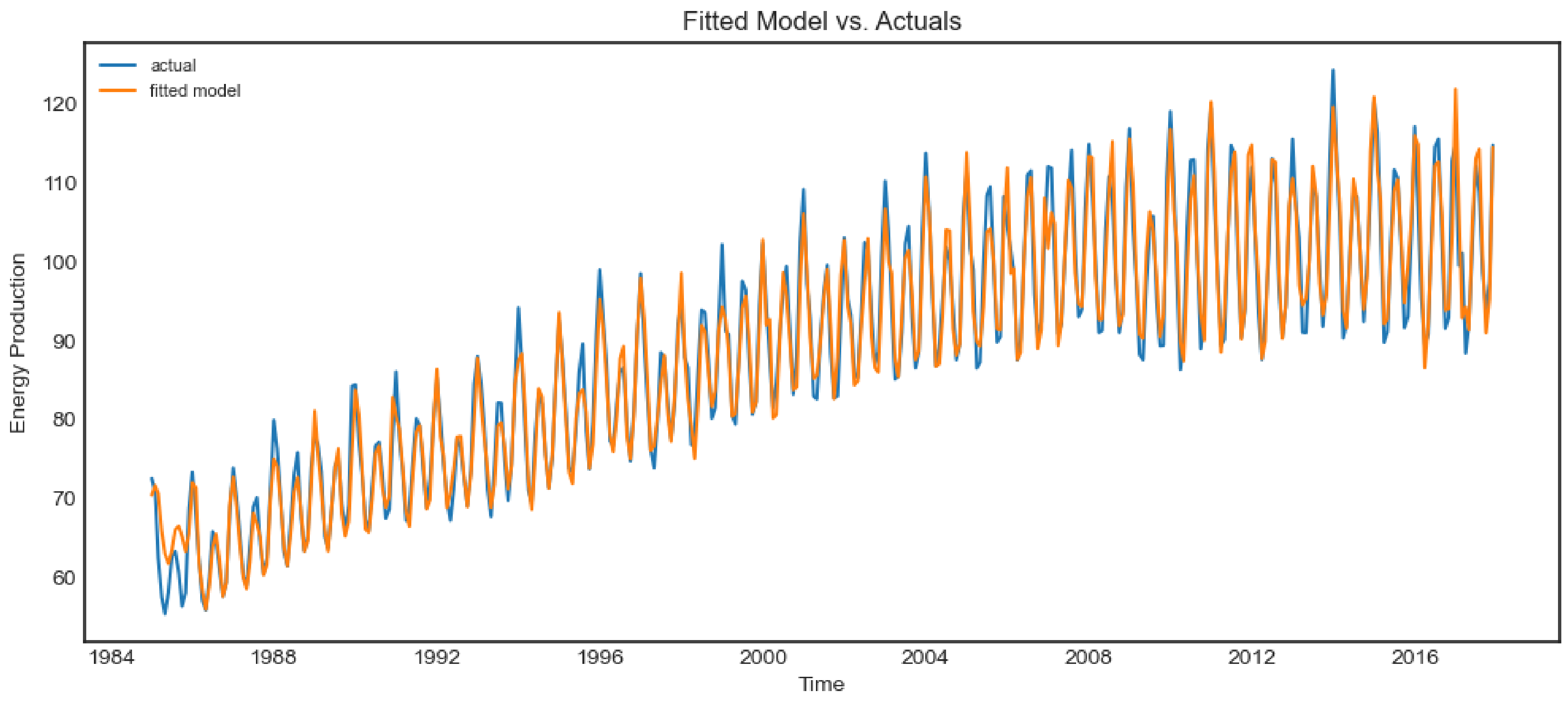
Creating a data frame that would hold the fitted values to be used later.

In [33]:
```
data2 =data.copy()
data2['fittedvalue']=fitted_F.predict_in_sample()
data2.drop('Value',axis=1,inplace=True)
data2
```

Out[33]:

	fittedvalue
DATE	
1985-01-01	70.375301
1985-02-01	71.611695
1985-03-01	70.557880
1985-04-01	65.806638
1985-05-01	62.936420
1985-06-01	61.706519
1985-07-01	63.342287
1985-08-01	66.010324
1985-09-01	66.458073
1985-10-01	65.115155

In [34]:
```python
plt.figure(figsize=(12,5), dpi=100)
plt.plot(data, label='actual')
plt.plot(data2, label='fitted model')
plt.title('Fitted Model vs. Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.xlabel('Time')
plt.ylabel('Energy Production')
plt.show()
```



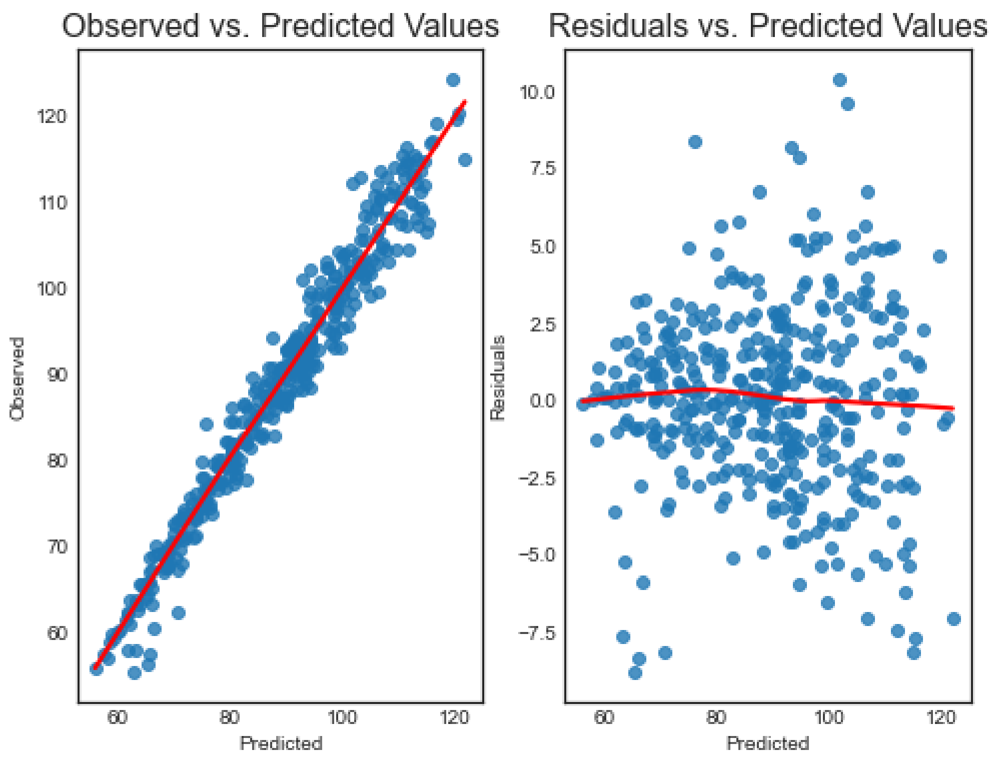
Looking at the above plot, it seems that our fitted model is doing a decent job predicting our existing data.

5. Plot the respective residuals vs. fitted values and discuss your observations.

Next, we will plot the fitted values against the observations and the residuals to measure the goodness of fit

In [35]:
```
fig, ax = plt.subplots(1,2,figsize=(8, 6))
sns.regplot(x=data2, y=data, lowess=True, ax=ax[0], line_kws={'color': 'red'})
ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
ax[0].set(xlabel='Predicted', ylabel='Observed')

sns.regplot(x=data2, y=fitted_F.resid(), lowess=True, ax=ax[1], line_kws={'color': 'red'})
ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
ax[1].set(xlabel='Predicted', ylabel='Residuals')
```

Out[35]:
```
[Text(0.5, 0, 'Predicted'), Text(0, 0.5, 'Residuals')]
```

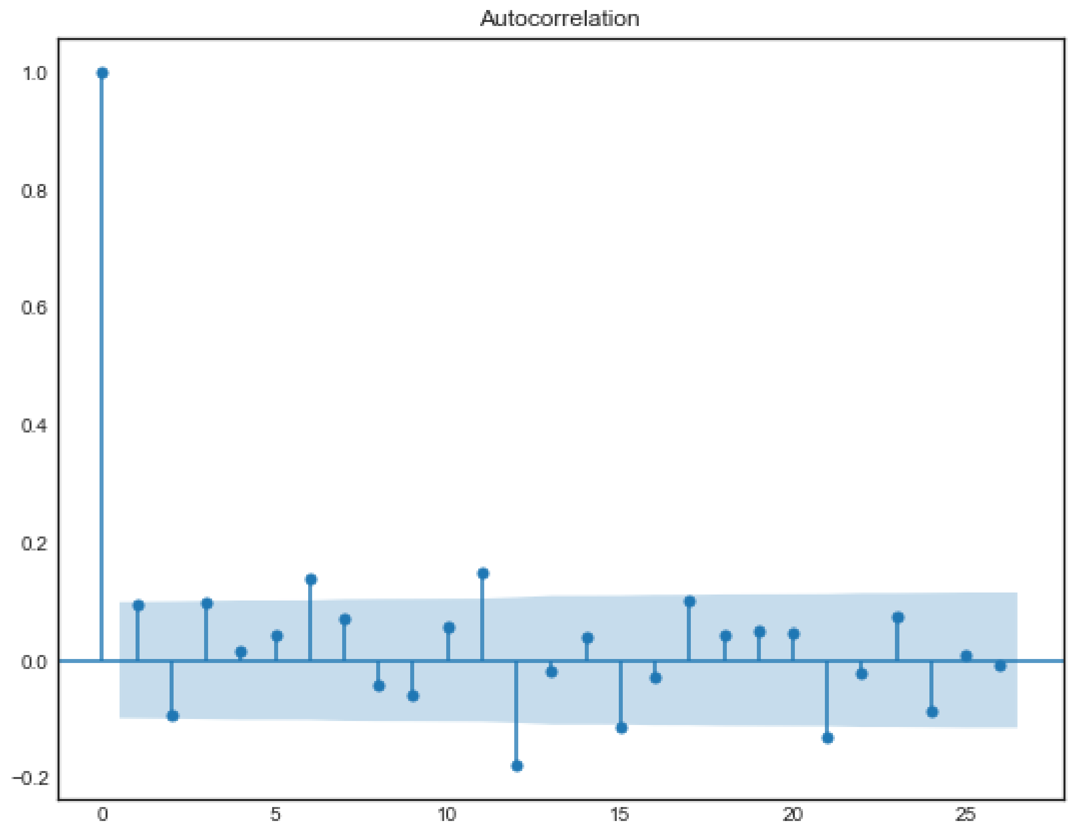


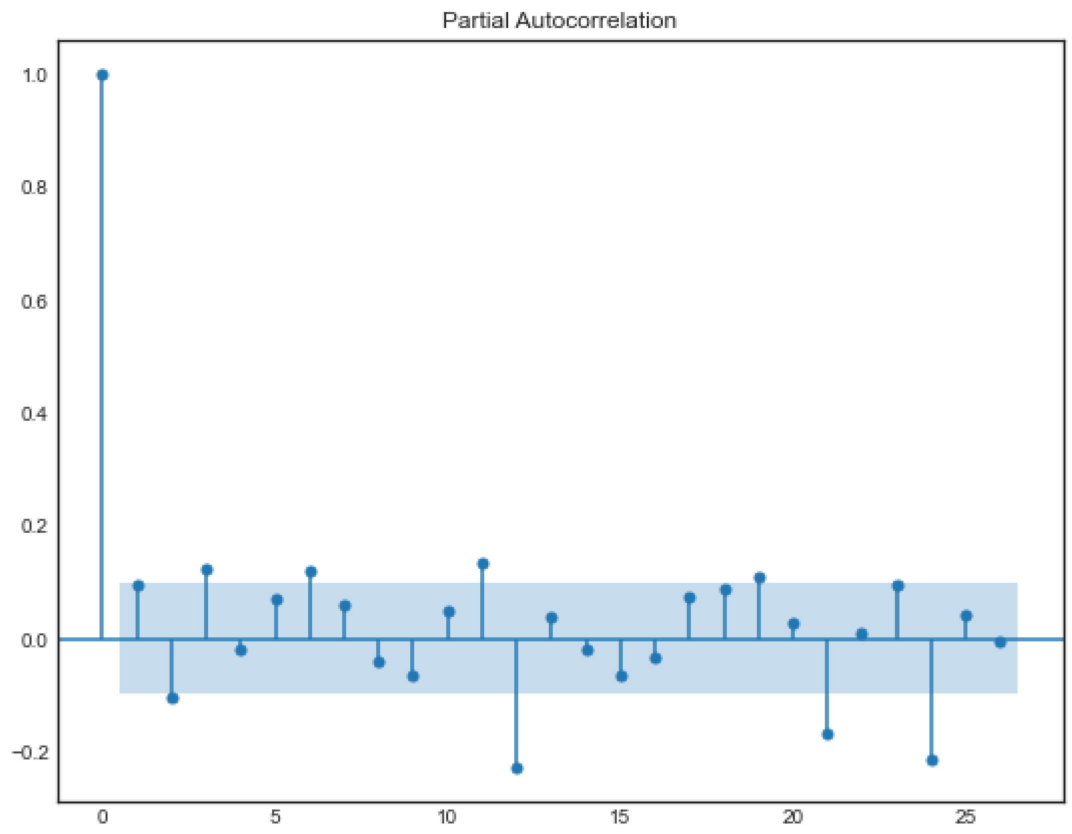
Looking at the above it is clear that our model does a great job of predicting the observed values, while the residuals lack any structure, hence, our model is sufficiently defined.

6. Plot the ACF and PACF of the respective residuals and interpret the plots.

Next, we will plot the ACF and PACF of the fitted model residual to check if more can be done to improve our model.

In [36]:
```
plot_acf(fitted_F.resid());
plot_pacf(fitted_F.resid());
```





Looking at the above PCF and PACF plots, we fail to detect any substantial spikes which is assuring us that the fit of the model was good. However, we are also not seeing only white noise, which means there are potentially better models than ours that we can look for and fit instead.

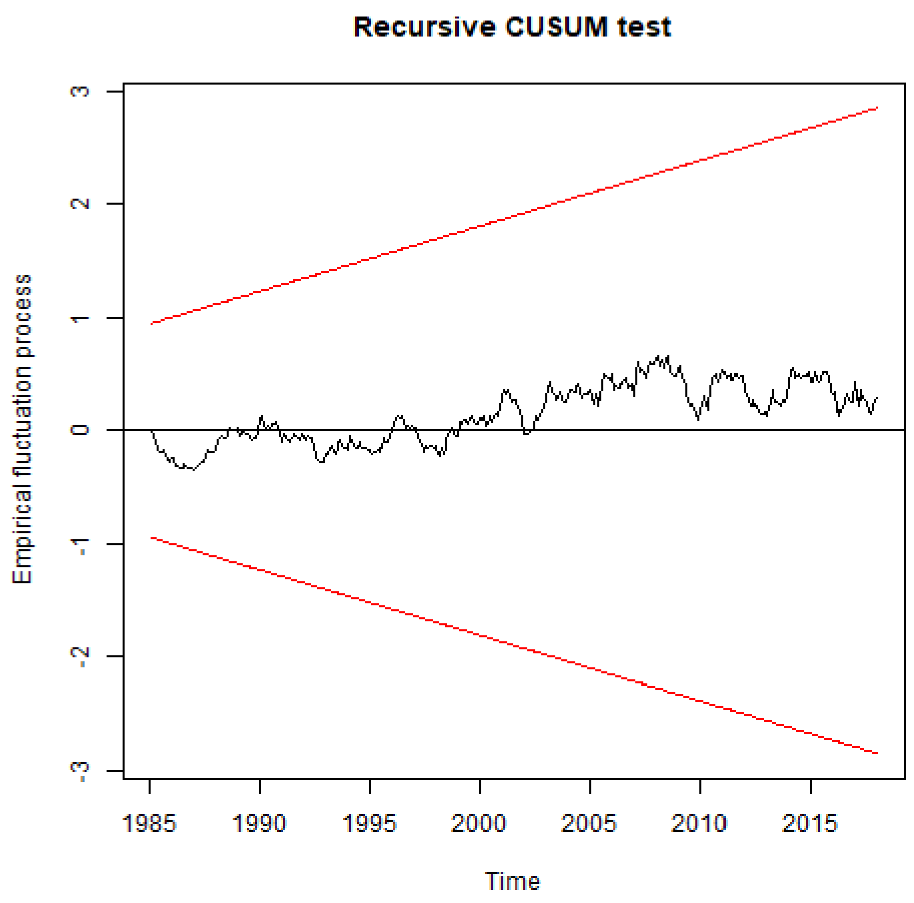
7. Plot the respective CUSUM and interpret the plot.

Next, we will plot the CUSUM of the residuals for further confirmation of our goodness of fit.

First we export our fitted residuals to a csv file, then call rpy2 to read our data in R and construct the CUSUM plot

In [37]:
```
import rpy2

import warnings
warnings.filterwarnings('ignore')

from rpy2.robjects import pandas2ri
import rpy2.rinterface as rinterface
pandas2ri.activate()

%load_ext rpy2.ipython
```

In [38]:
```
%%R

# Load Libraries
library(tis)
require("datasets")
require(graphics)
library("forecast")
library(tseries)
library(stats)
library(fpp)
library(strucchange)

# Look at the data and log(data)
data=read.table("C:\\Users\\cuber\\OneDrive\\Course Work\\430\\Project 3\\EP.dat")
data_ts<-ts(data[,1],start=1985.1,freq=12)
t<-seq(1985, 2018.1,length=length(data_ts))


t2<-t^2
m5=arima(data_ts,order=c(1,0,0),xreg = cbind(t, t2),seasonal=list(order=c(1,0,0)))


plot(efp(m5$res~1, type = "Rec-CUSUM"))

```



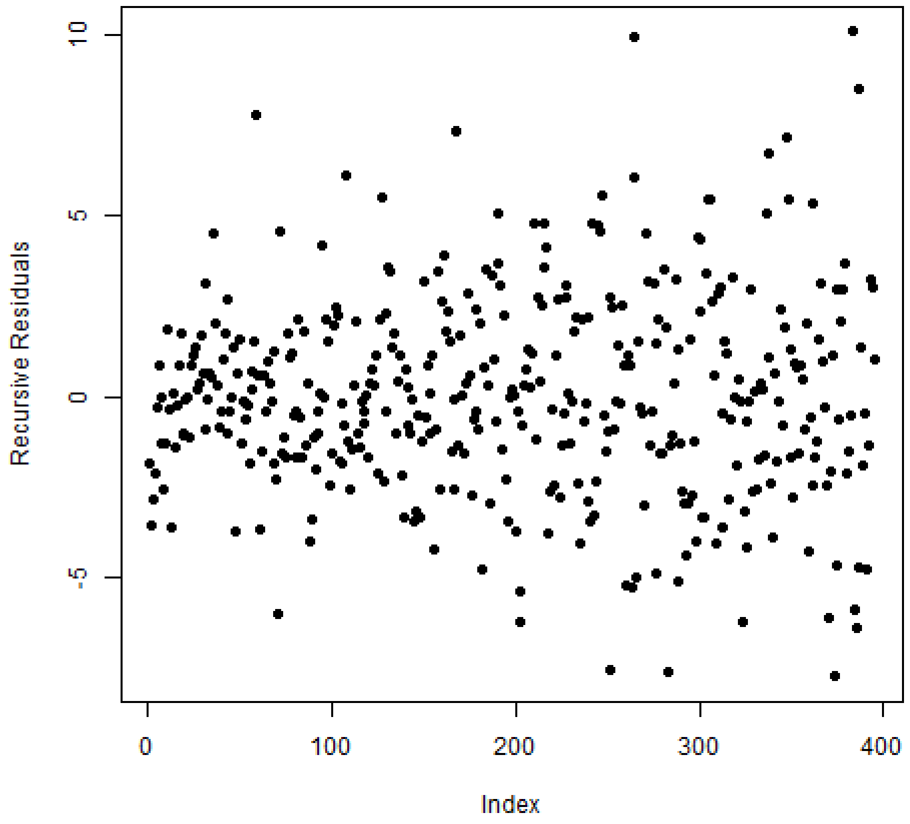
Looking at the recursive CUSUM plot, it seems that the cumulative sum is stable over the whole period and its mean is around fluctuates around zero and within the confidence interval. Accordingly, our model seems to be stable enough in tracking our data and can be used without concern.

8. Plot the respective Recursive Residuals and interpret the plot.

Next we will plot the recursive residuals using R

In [39]:

```
%%R

y=recresid(m5$res~1)
plot(y, pch=16,ylab="Recursive Residuals")
```



The Recursive residuals seems to be randomly scattered with mean 0 and constant variance without a structure, which is what we exactly want to confirm.

9. For your model, discuss the associated diagnostic statistics.

Now we will look at the model diagnostic statistics

In [40]:
```
fitted_F.plot_diagnostics(figsize = (12,10));
fitted_F.summary()
```

Out[40]:

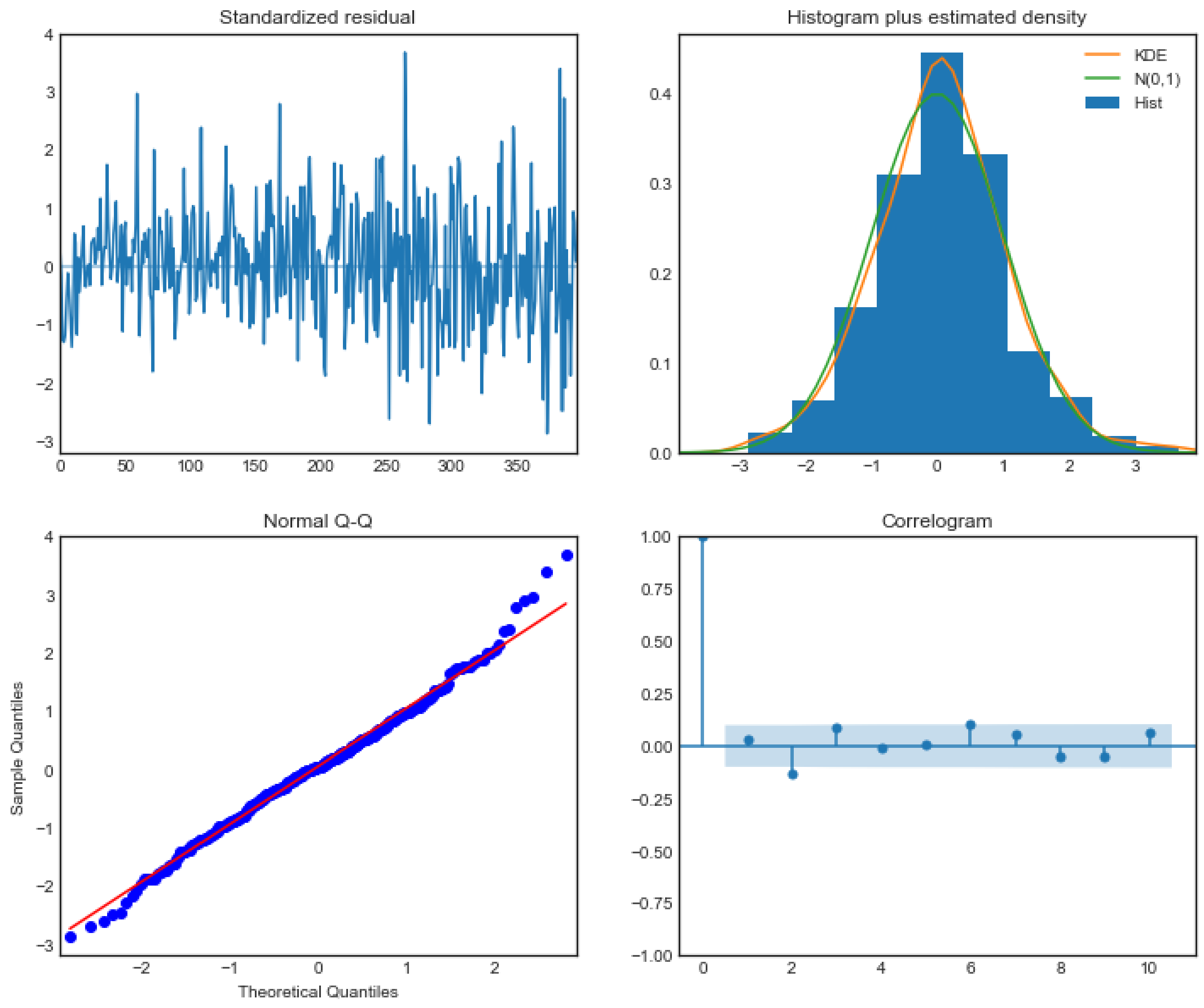
SARIMAX Results

Dep. Variable:	y	**No. Observations:**	396
Model:	SARIMAX(1, 0, 0)x(1, 0, 0, 12)	**Log Likelihood**	-982.418
Date:	Thu, 02 Dec 2021	**AIC**	1974.836
Time:	01:28:37	**BIC**	1994.743
Sample:	0	**HQIC**	1982.723
	- 396		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	3.0229	0.647	4.669	0.000	1.754	4.292
drift	0.0054	0.001	4.175	0.000	0.003	0.008
ar.L1	0.5790	0.037	15.661	0.000	0.507	0.651
ar.S.L12	0.8978	0.020	44.933	0.000	0.859	0.937
sigma2	8.0052	0.500	16.001	0.000	7.025	8.986

Ljung-Box (L1) (Q):	0.39	**Jarque-Bera (JB):**	9.59
Prob(Q):	0.53	**Prob(JB):**	0.01
Heteroskedasticity (H):	2.60	**Skew:**	0.13
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.72

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



The coefficients of the model have relatively small standard errors and narrow confidence interval while being statistically significant at above 99% confidence as evident from the very low p-values across the board.

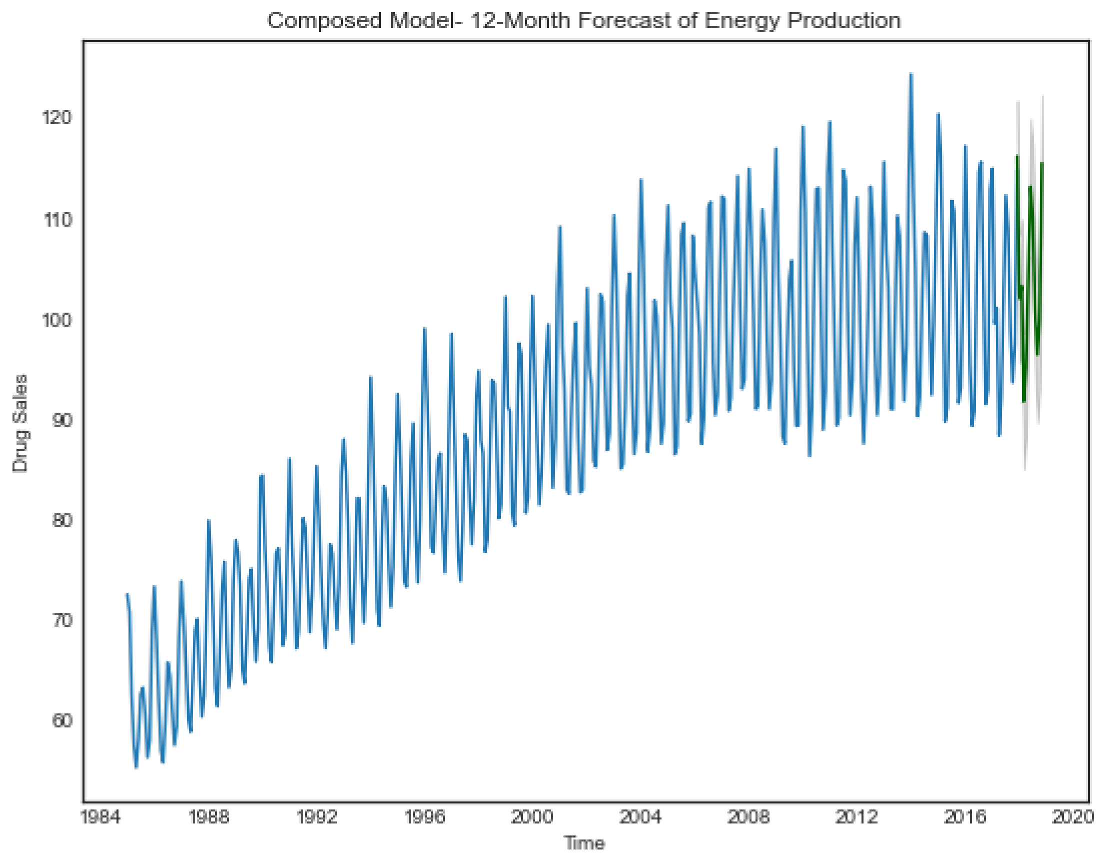
Additionally, we've verified the goodness of its fit above using multiple diagnostic plots (fitted vs observed and residuals, AFC & PAFC, CUSUM and recursive residuals), all of which told the same story about how good the fit of the model is from different angles. The histogram and q-q plot also demonstrate the normality our residual, which is what we would like to see.

10. Use your model to forecast 12-steps ahead. Your forecast should include the respective error bands. Compare this forecast against a forecast from your ARIMA model estimated in (3). Which one is better?

Now we will move on to test our model's ability to forecast the future and compare it against the auto-ARIMA produced model.

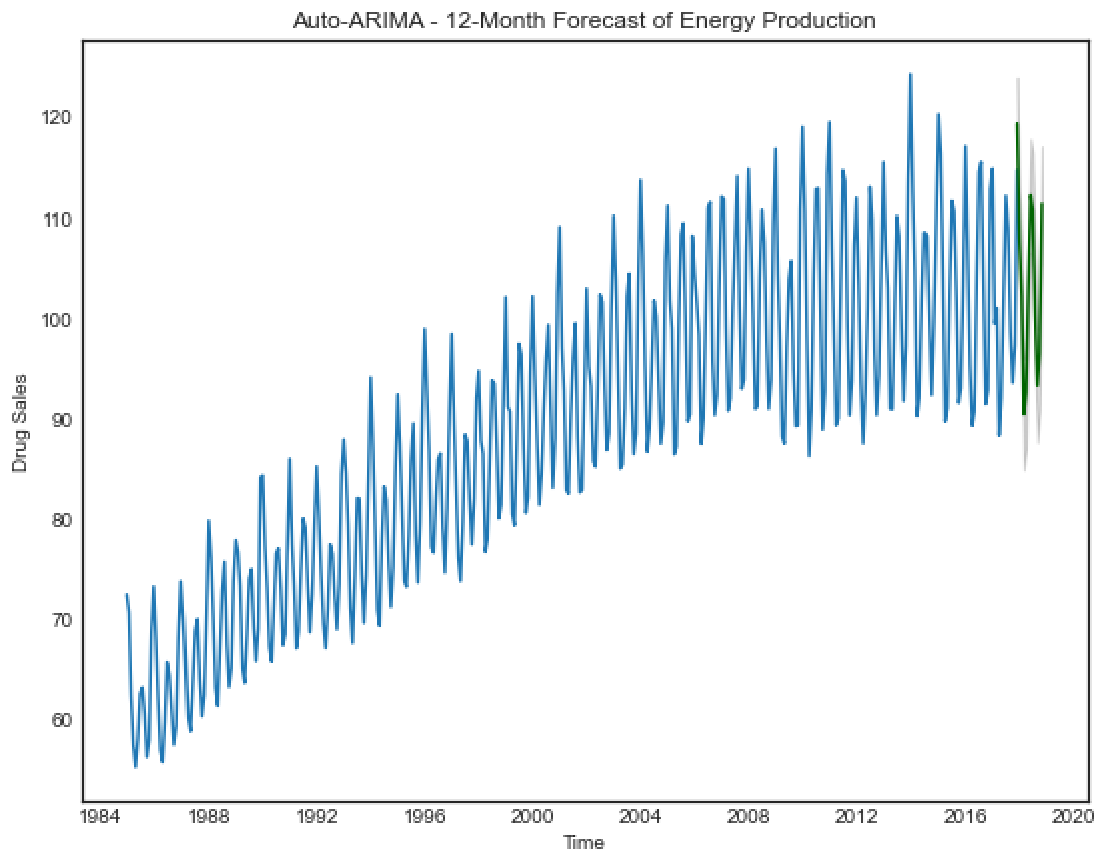
First we start by forecasting the next 12 months using our own model

In [41]:
```
n_periods = 12
fitted, confint = fitted_F.predict(n_periods=n_periods, return_conf_int=True)
index_of_fc = pd.date_range(data.index[-1], periods = n_periods, freq='MS')

# make series for plotting purpose
fitted_series = pd.Series(fitted, index=index_of_fc)
lower_series = pd.Series(confint[:, 0], index=index_of_fc)
upper_series = pd.Series(confint[:, 1], index=index_of_fc)

# Plot
plt.plot(data)
plt.plot(fitted_series, color='darkgreen')
plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k', alpha=.15)

plt.title("Composed Model- 12-Month Forecast of Energy Production")
plt.xlabel('Time')
plt.ylabel('Drug Sales')
plt.show()
```



Next we forecast the Auto-ARIMA for the next 12 months

In [42]:
```
n_periods = 12
fitted, confint = smodel.predict(n_periods=n_periods, return_conf_int=True)
index_of_fc = pd.date_range(data.index[-1], periods = n_periods, freq='MS')

# make series for plotting purpose
fitted_series = pd.Series(fitted, index=index_of_fc)
lower_series = pd.Series(confint[:, 0], index=index_of_fc)
upper_series = pd.Series(confint[:, 1], index=index_of_fc)

# Plot
plt.plot(data)
plt.plot(fitted_series, color='darkgreen')
plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k', alpha=.15)

plt.title("Auto-ARIMA - 12-Month Forecast of Energy Production")
plt.xlabel('Time')
plt.ylabel('Drug Sales')
plt.show()
```



Looking at the above two forecasts, it is very difficult to discern any difference, as both of which seem to exhibit similar forecasting values.

11. Backtest your ARIMA model and your model from (3). Begin by partitioning your data set into an estimation set and a prediction set.

Next we will move on to backtesting the models using our samples

(a) Use a recursive backtesting scheme, and forecast 12-steps ahead at each iteration. Compute the mean absolute percentage error at each step. Provide a plot showing the MAPE over each iteration.

We'll first define a function to take in the forecast and computes the mean absolute percentage error (MAPE)

In [43]:
```
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    return(mape)
```

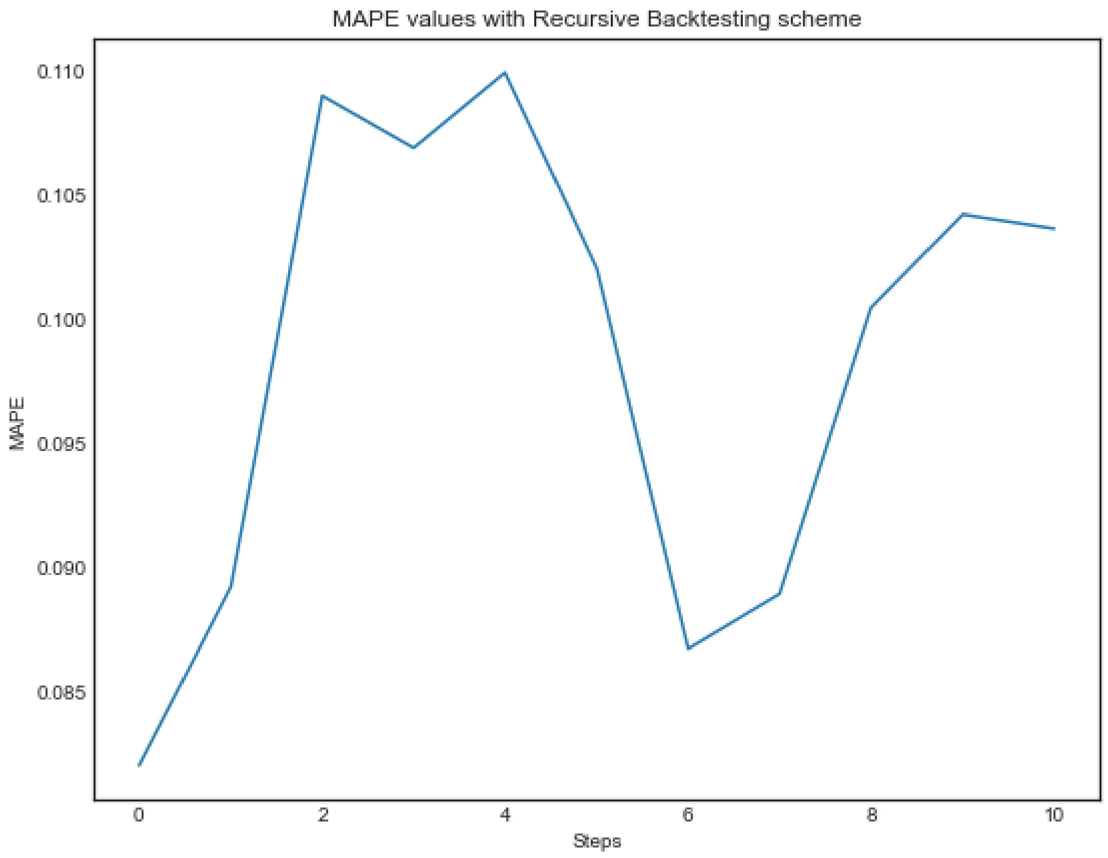
Now we will create two functions that takes the partial data and number of forecast steps, fits it into our two models and return the next 12 step forecast for direct comparison with actuals.

In [44]:
```
def ARIMA_Fitter(data,h):
    model = pm.ARIMA(order=(1,0,0),seasonal_order=(1,0,0,12),trend=[59.26,0.225,-0.000289])
    fitted = model.fit(data)
    return fitted.predict(h)
```

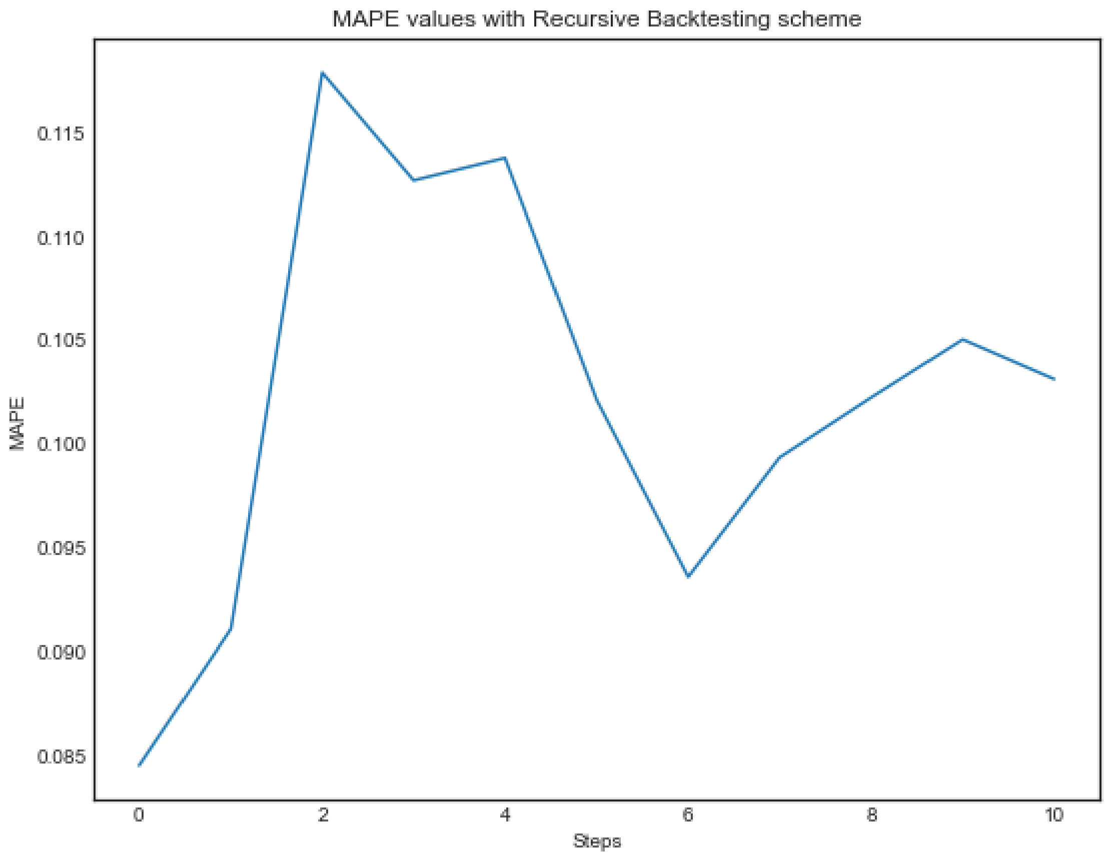
In [45]:
```
def ARIMA_FitterA(data,h):
    model = pm.ARIMA(order=(1,1,2),seasonal_order=(2,0,2,12))
    fitted = model.fit(data)
    return fitted.predict(h)
```

Now we will use a loop to forecast 12 steps within the data and measure it against the actual data to calculate the MAPE at each step, since we have 33 years' worth of data, we will divide them into 22 for fitting and 11 for this part which equals 132 months, this will be subtracted from the end point and the value will be assigned to n as a starting point.

In [46]:
```python
#Recursive backtesting scheme for our model (from 4)
MAPE = []
n=396-(11*12) # end point less the number of months representing 11 years
while n < 396:
    fitted = ARIMA_Fitter(data[:n],12)
    test = data[n:n+12]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=12
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Recursive Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```



In [47]:
```python
#Recursive backtesting scheme for Auto-ARIMA model (from 3)
MAPE = []
n=396-(11*12) # end point less the number of months representing 11 years
while n < 396:
    fitted = ARIMA_FitterA(data[:n],12)
    test = data[n:n+12]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=12
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Recursive Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```



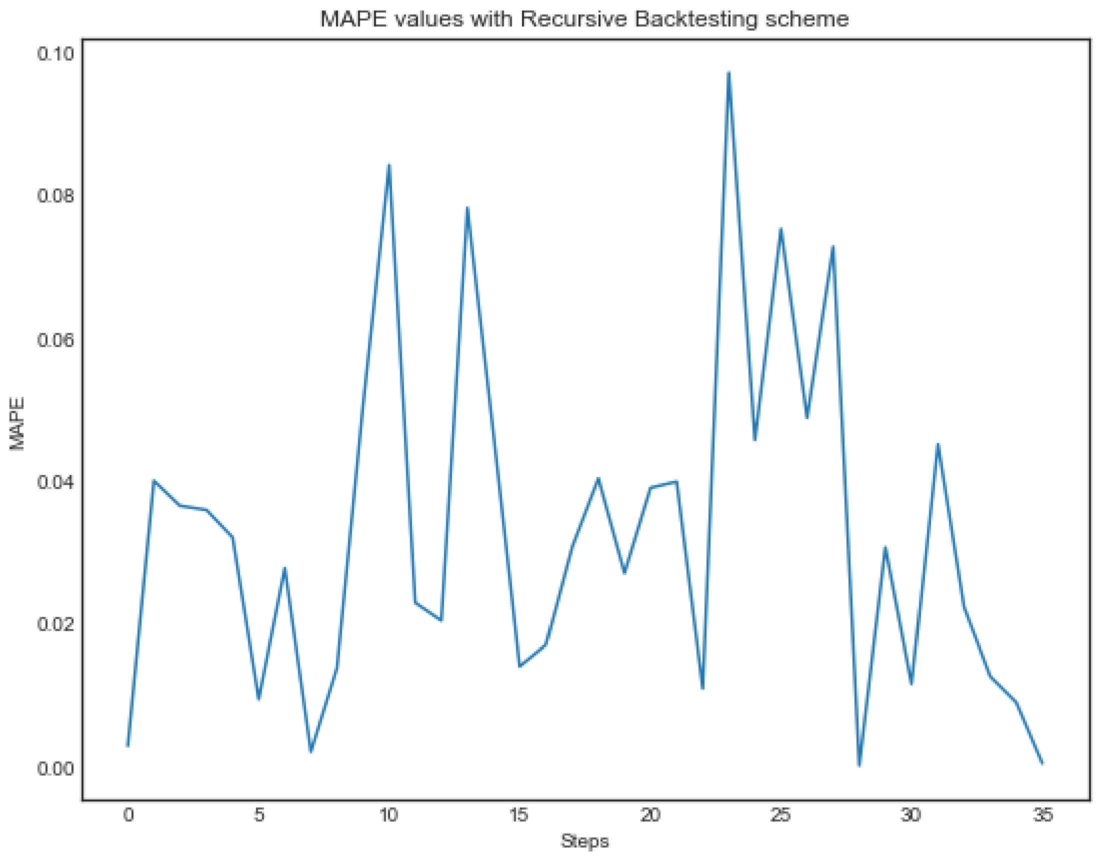
Looking at the MAPE trends above, they are generally very close which means that our fitted model is very close to the best fit produced by our reference (Auto_ARIMA).

Aside from that both models are around 90% accurate in predicting up to 11 years in the future, this is a great result which was expected given the previous tests we've done to analyze the fit, performance and stability of the model.

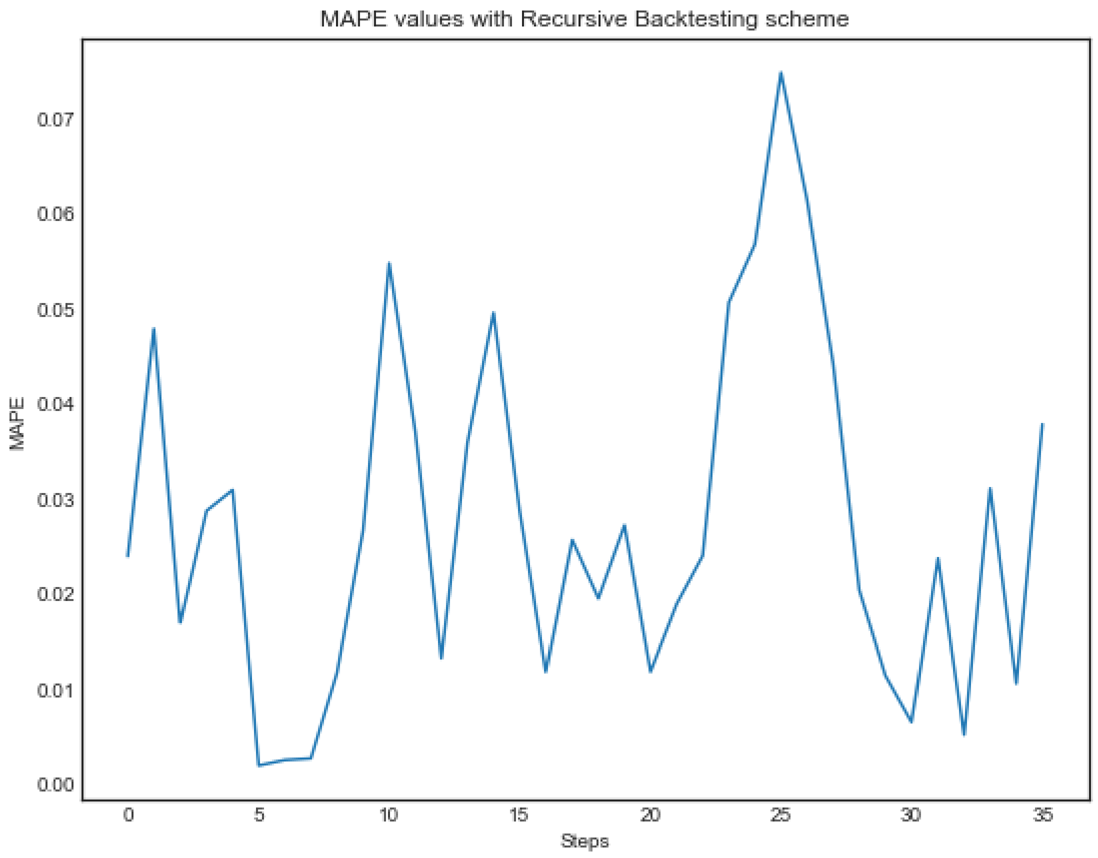
(b) Shorten your forecast horizon to only 1-step ahead. Compute the absolute percentage error at each iteration, and plot.

Now we will reduce the step size from 12 months (1 year) to 1 month only and check the impact on MAPE, however we will reduce the test data from 11 years to 3 years, since the iterations take very long time with 11 years, while the conclusion shouldn't change with this reduction in test data.

In [48]:
```
#Recursive backtesting scheme for our model (from 4) with 1 month step size
MAPE = []
n=396-(3*12) # end point less the number of months representing 3 years
while n < 396:
    fitted = ARIMA_Fitter(data[:n],1)
    test = data[n:n+1]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=1
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Recursive Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```



In [49]:
```
#Recursive backtesting scheme for Auto-ARIMA model (from 3) with 1 month step size
MAPE = []
n=396-(3*12) # end point less the number of months representing 3 years
while n < 396:
    fitted = ARIMA_FitterA(data[:n],1)
    test = data[n:n+1]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=1
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Recursive Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```



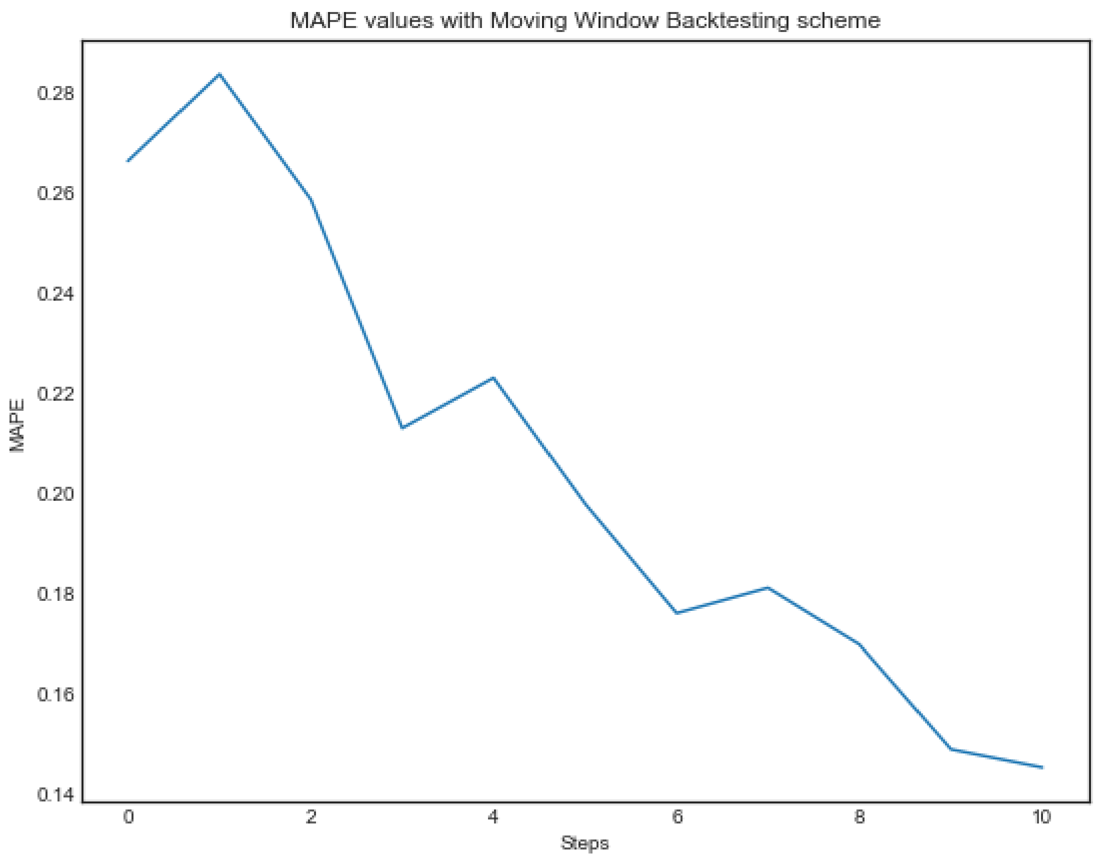
(c) Based on your findings above, does your model perform better at longer or shorter horizon forecasts?

The models generally perform better with shorter horizon forecast, since the models are based on AR(1) which uses a one step lag to predict the next and hence, 1 step prediction accuracy is generally higher for those models.

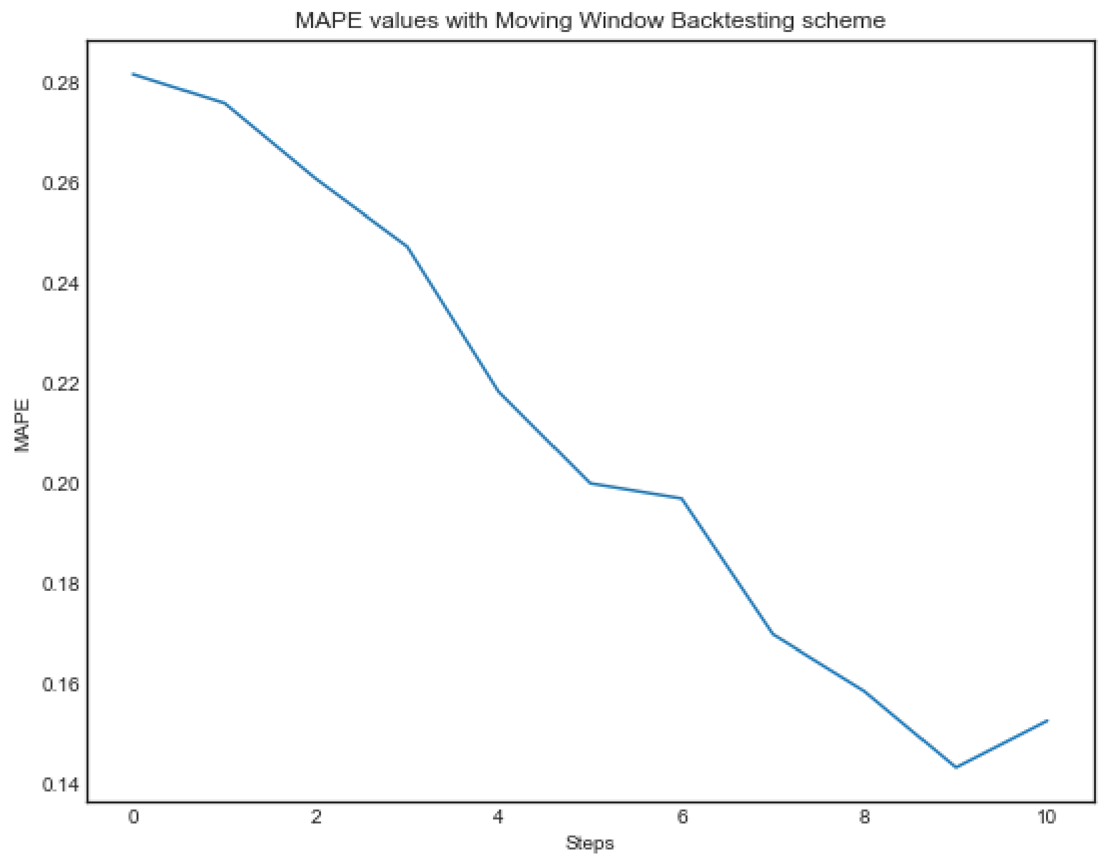
(d) Now test your model using a moving window backtesting scheme. Forecast out 12-steps ahead

We will now change the backtesting scheme to moving window and repeat the analysis done above.

In [50]:
```
#Moving window backtesting scheme for our model (from 4)
MAPE = []
n=396-(11*12) # end point less the number of months representing 11 years
i= 0 #
while n < 396:
    fitted = ARIMA_Fitter(data[i:n],12)
    test = data[i:n]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=12
    i+=12
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Moving Window Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```

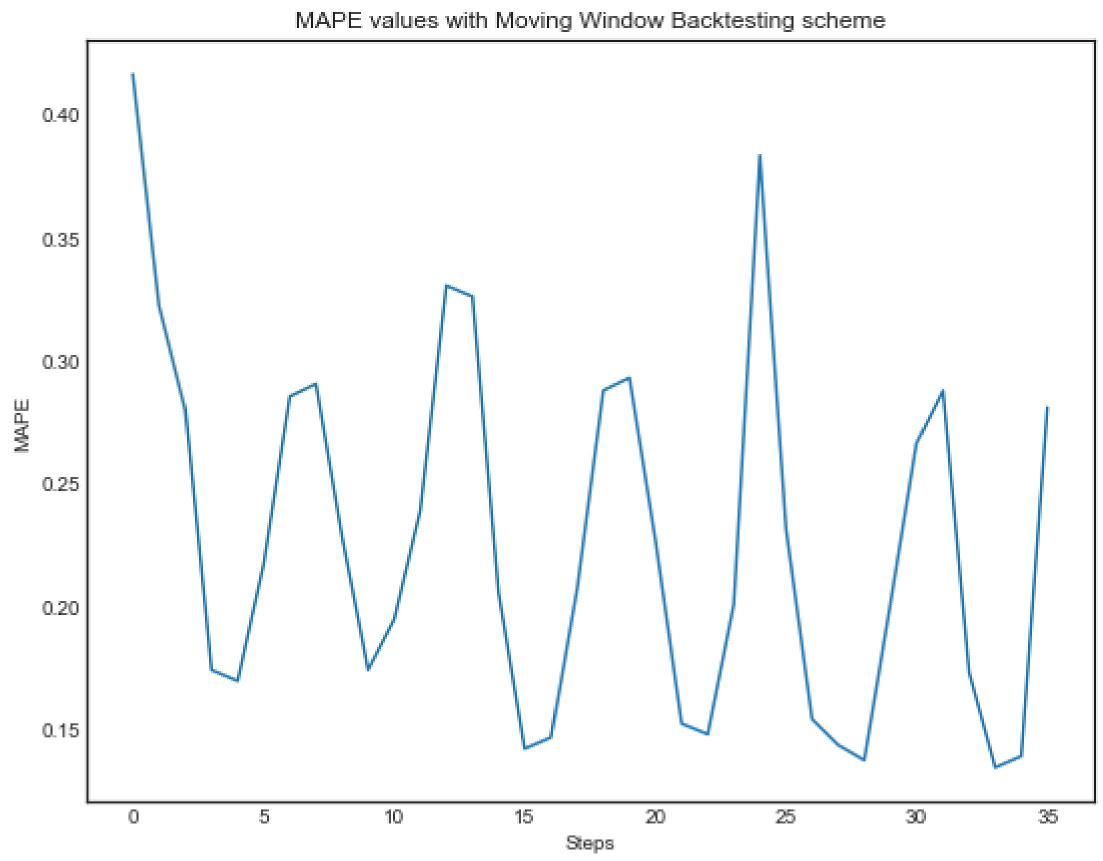

MAPE values with Moving Window Backtesting scheme
MAPE
Steps
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0
2
4
6
8
10

In [51]:
```
#Moving window backtesting scheme for Auto-ARIMA Model (from 3)
MAPE = []
n=396-(11*12) # end point less the number of months representing 11 years
i= 0
while n < 396:
    fitted = ARIMA_FitterA(data[i:n],12)
    test = data[i:n]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=12
    i+=12
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Moving Window Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```

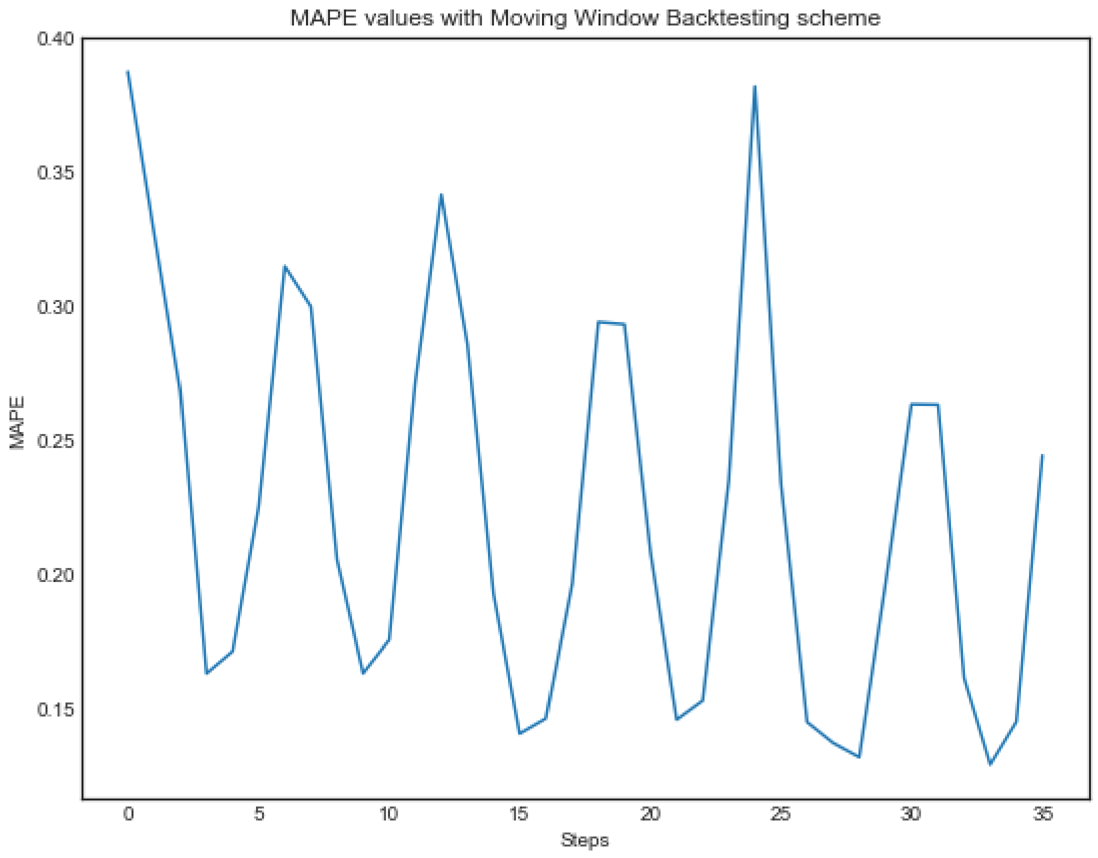

MAPE values with Moving Window Backtesting scheme
MAPE
Steps
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0
2
4
6
8
10

In [52]:

```
#Moving window backtesting scheme for our model (from 4) with 1 step size and reduced testing da
MAPE = []
n=396-(3*12) # end point less the number of months representing 3 years
i= 0
while n < 396:
    fitted = ARIMA_Fitter(data[i:n],1)
    test = data[i:n]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=1
    i+=1
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Moving Window Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```


MAPE values with Moving Window Backtesting scheme
MAPE
0.40
0.35
0.30
0.25
0.20
0.15
0
5
10
15
20
25
30
35
Steps

In [53]:
```
#Moving window backtesting scheme for our model (from 4) with 1 step size and reduced testing da
MAPE = []
n=396-(3*12) # end point less the number of months representing 3 years
i= 0
while n < 396:
    fitted = ARIMA_FitterA(data[i:n],1)
    test = data[i:n]
    MAPE.append(forecast_accuracy(fitted,test.values))
    n+=1
    i+=1
MAPE
plt.plot(MAPE)
plt.title("MAPE values with Moving Window Backtesting scheme")
plt.xlabel('Steps')
plt.ylabel('MAPE')
plt.show()
```



(e) How do the errors found using a recursive backtesting scheme compare with the errors observed using a moving average backtesting scheme? Which scheme showed higher errors overall, and what does that tell you about your model?

In general, the errors associated with moving window backtesting scheme were higher compared to the recursive scheme. This tells us that the initial values of the data were more critical to the fit of our model as the error increases as those values being omitted.

Additionally, the errors in general were relatively small across different testing data with both methods, which establishes more trust in the capability of our models.

(f) Finally, which model do you prefer and why?

Additionally, the errors in general were relatively small across different testing data with both methods, which establishes more trust in the capability of our models. Both models exhibited similar performance, without a significant gap in any of the tests used. However, the Auto-ARIMA model was slightly better overall, and would be the best fit in this case.

Yet, we would most likely choose the model we made ourselves from scratch just because we know exactly why we chose it and we can adjust it as we see fit if needed (as we did earlier), which also technically applies for the other model as well, less the sentimental value.

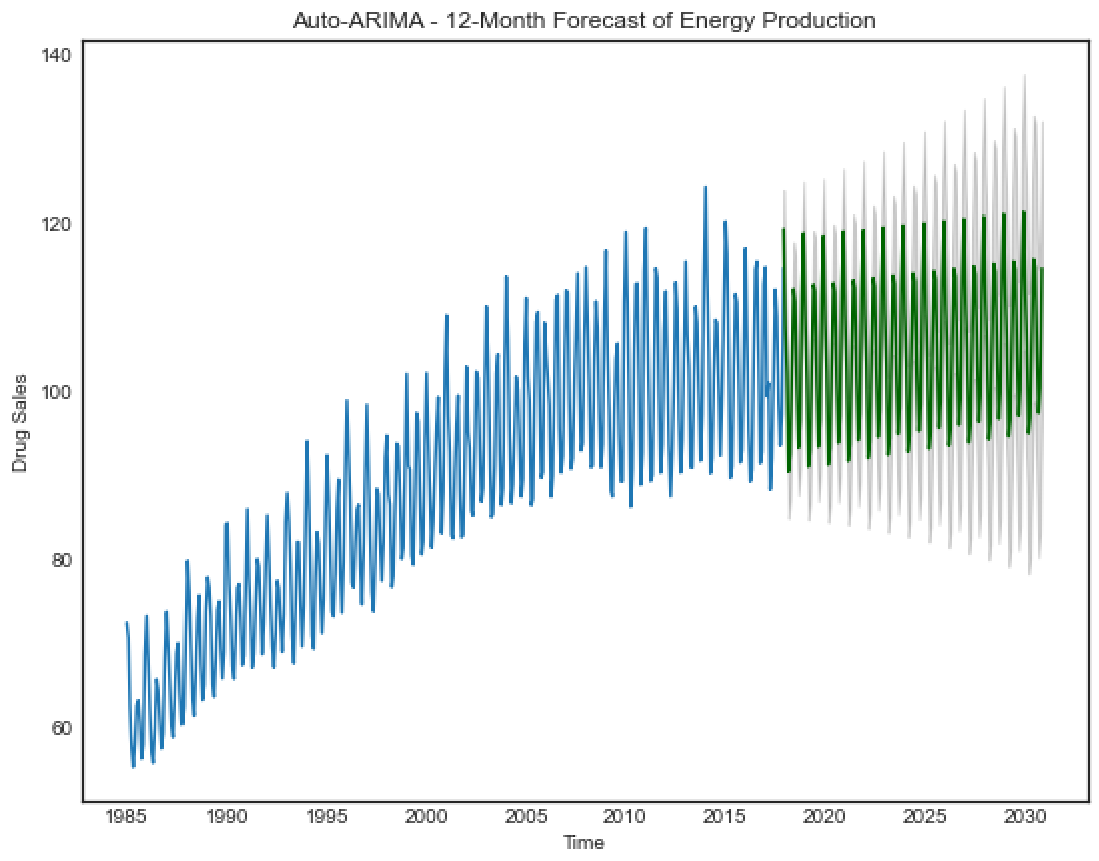
Part III: Conclusion

The objective of this project was to model the electricity consumption in an undisclosed region given the past 33 years data and study its behavior to help us understand where are we today and what to expect in the future, and decide whether to pursue an upgrade in energy generation capabilities is required or not.

Through the steps taken in this project, we managed to study the data, identify its components, decompose it, and create a great fit to model the behavior of the data. The fit was tested along the way and passed most major tests required to be identified as a good fit and accordingly was approved for use for future projections.

Now that the model is approved, we will use it to peek into the future and forecast the electricity consumption up to 2030 and check against our initial observation that energy consumption may have plateaued already and not much gain is expected going forward, indicated a steady state in energy demand and/or increase in energy efficiency.

In [54]:
```python
n_periods = (12*13) # projecting up to 2030
fitted, confint = smodel.predict(n_periods=n_periods, return_conf_int=True)
index_of_fc = pd.date_range(data.index[-1], periods = n_periods, freq='MS')

# make series for plotting purpose
fitted_series = pd.Series(fitted, index=index_of_fc)
lower_series = pd.Series(confint[:, 0], index=index_of_fc)
upper_series = pd.Series(confint[:, 1], index=index_of_fc)

# Plot
plt.plot(data)
plt.plot(fitted_series, color='darkgreen')
plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k', alpha=.15)

plt.title("Auto-ARIMA - 12-Month Forecast of Energy Production")
plt.xlabel('Time')
plt.ylabel('Drug Sales')
plt.show()
```



Looking at the prediction presented by the best fit model, we can see the energy consumption while slightly increasing, it is not expected to rise significantly in the next decade or so, and thus concluding that the region may have approached that steady state and immediate expansion in energy generation may not be required.

Part IV: References

Data

The data was sourced from kaggle.com which was submitted by Kandi Jagadish and it's accessible through the below link. https://www.kaggle.com/kandij/electric-production (https://www.kaggle.com/kandij/electric-production)

Code

Most of the code used within this project was included as part of Fall 2021 UCLA MQE Econ 430 course, which is taught by Professor Randal R. Rojas.