Dataset Agrana

In [2]:

```
# Import the library
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
import sys
from datetime import datetime

# Import statsmodel
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.tsa.stattools import adfuller

from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6
```

1) Data Munging:

After the process of transforming and mapping data, loading data using Pandas Dataframe:

In [3]:

```
import pandas as pd
agrana= pd.read_excel('agra_datetime.xlsx',
                      names=['Dates','Country','MC_Product','Total_Sales'])
```

In [4]:

```
agrana.shape
```

Out[4]:

```
(2393, 4)
```

In [5]:

```
agrana.head(10)
```

Out[5]:

	Dates	Country	MC_Product	Total_Sales
0	2016-09-01	AE	Kristall	0.0
1	2016-10-01	AE	Kristall	0.0
2	2016-11-01	AE	Kristall	0.0
3	2016-12-01	AE	Kristall	0.0
4	2016-09-01	AL	Kristall	0.0
5	2016-10-01	AL	Kristall	0.0
6	2016-11-01	AL	Kristall	0.0
7	2016-12-01	AL	Kristall	0.0
8	2016-09-01	AM	Kristall	0.0
9	2016-10-01	AM	Kristall	0.0

2) Reading as datetime format:

WE need to convert into a datetime index for time series analysis

In [6]:

```
# Change the index to the datetime index
agrana.index = pd.PeriodIndex(agrana.Dates, freq='M', inplace=True)
# Drop redundant columns
ag_drop = agrana.drop(["Dates"], axis = 1, inplace=True)
```

In [7]:

```
agrana.head()
```

Out[7]:

	Country	MC_Product	Total_Sales
Dates			
2016-09	AE	Kristall	0.0
2016-10	AE	Kristall	0.0
2016-11	AE	Kristall	0.0
2016-12	AE	Kristall	0.0
2016-09	AL	Kristall	0.0

In [8]:

```
agrana.info()
```

```
<class 'pandas.core.frame.DataFrame'>
PeriodIndex: 2393 entries, 2016-09 to 2018-01
Freq: M
Data columns (total 3 columns):
Country        2393 non-null object
MC_Product     2393 non-null object
Total_Sales    2393 non-null float64
dtypes: float64(1), object(2)
memory usage: 74.8+ KB
```

In [9]:

```
agrana.index
```

Out[9]:

```
PeriodIndex(['2016-09', '2016-10', '2016-11', '2016-12', '2016-09', '2016-
10',
             '2016-11', '2016-12', '2016-09', '2016-10',
             ...
             '2018-02', '2018-01', '2018-02', '2018-01', '2018-02', '2018-
01',
             '2018-02', '2018-01', '2018-02', '2018-01'],
            dtype='period[M]', name='Dates', length=2393, freq='M')
```

3) Explore and Visualize the Data

In [10]:

```
#number Country
agrana['Country'].value_counts().count()
```

Out[10]:

```
52
```

In [11]:

```
# Countries
pd.unique(agrana.Country)
```

Out[11]:

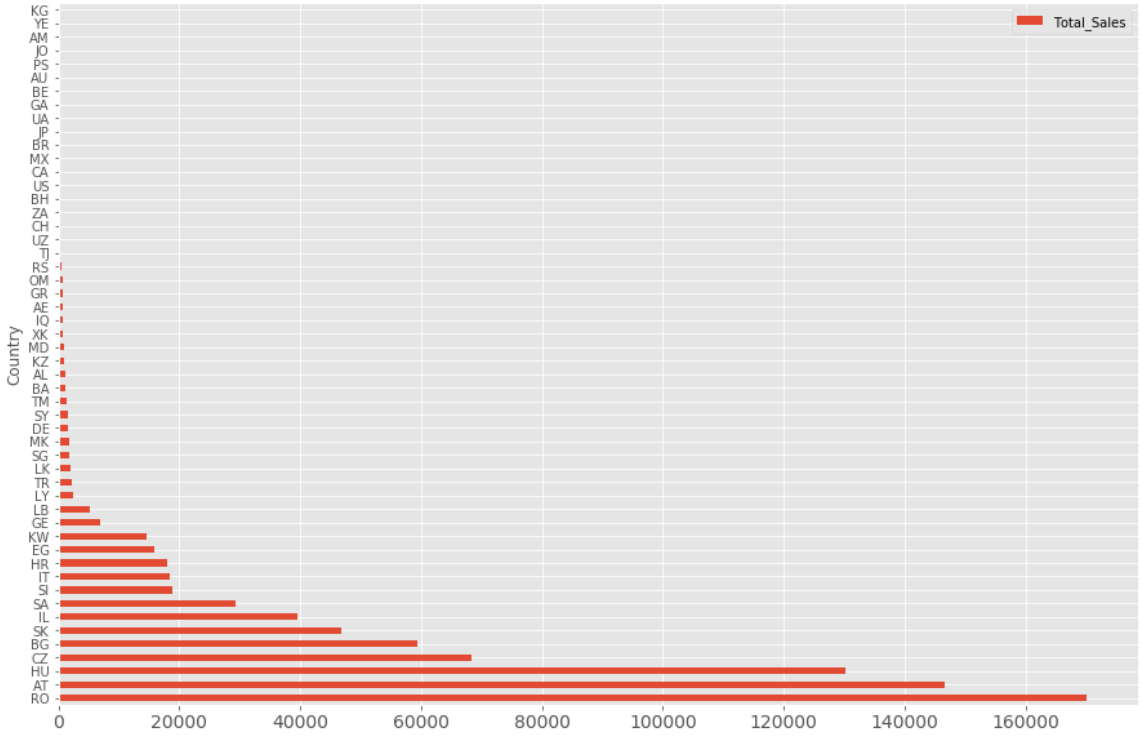
```
array(['AE', 'AL', 'AM', 'AT', 'AU', 'BA', 'BE', 'BG', 'BH', 'BR', 'CA',
       'CH', 'CZ', 'DE', 'EG', 'GA', 'GE', 'GR', 'HR', 'HU', 'IL', 'IQ',
       'IT', 'JO', 'JP', 'KG', 'KW', 'KZ', 'LB', 'LK', 'LY', 'MD', 'MK',
       'MX', 'OM', 'PS', 'RO', 'RS', 'SA', 'SG', 'SI', 'SK', 'SY', 'TJ',
       'TM', 'TR', 'UA', 'US', 'UZ', 'XK', 'YE', 'ZA'], dtype=object)
```

In [12]:

```
# Set some parameters to get good visuals
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (15, 10)

# Plot the Data(Sales x Coutries)
agra_Country = agrana.groupby(['Country']).sum()

agra= pd.DataFrame(agra_Country.sort_values(by='Total_Sales',ascending=False))
group_data = agra.reset_index()
group_data.plot(kind ="barh", x = 'Country', y = 'Total_Sales')
plt.xticks(fontsize=14)
plt.yticks(fontsize=10)
```

Out[12]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 3
3,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 5
0,
        51]), <a list of 52 Text yticklabel objects>)
```



In [13]:

```
#number products
agrana['MC_Product'].value_counts().count()
```

Out[13]:

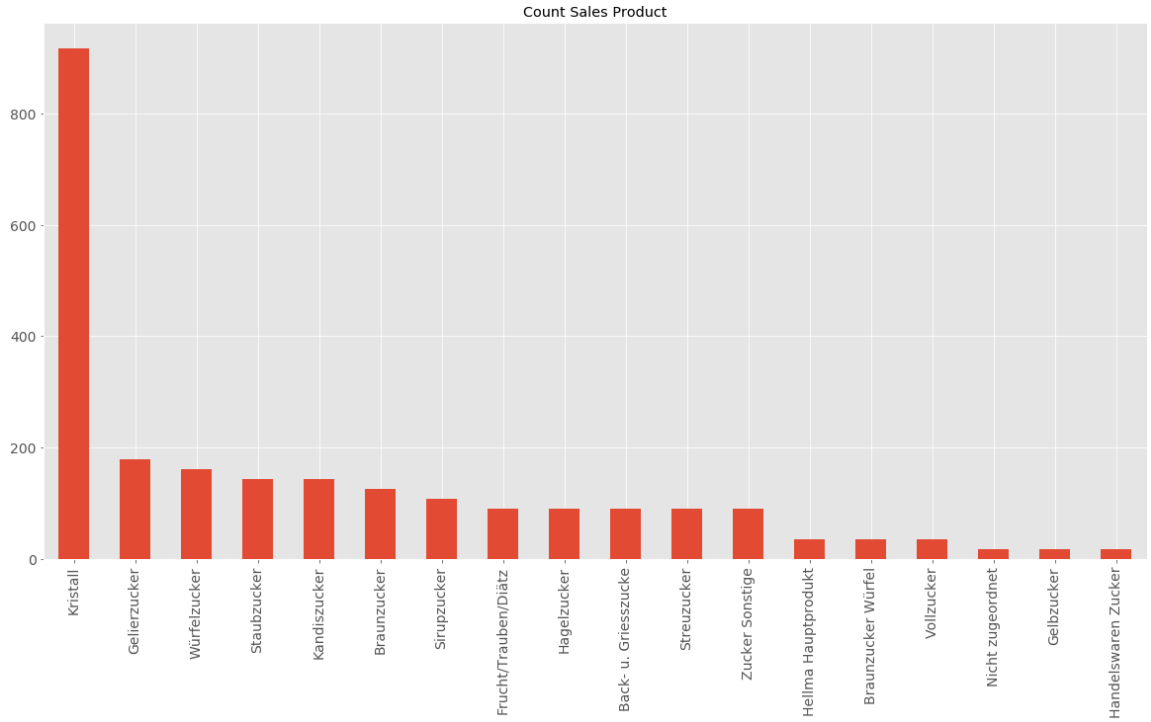
```
18
```

In [14]:

```
#Product Sales
agrana['MC_Product'].value_counts().plot(kind= 'bar',grid= True, figsize=(20,10),
                                         title='Count Sales Product')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
```

Out[14]:

```
(array([    0.,   200.,   400.,   600.,   800.,  1000.]),
 <a list of 6 Text yticklabel objects>)
```



In [15]:

```
#ProductxCountry
pivot_table= agrana.pivot_table(values='Total_Sales',index=['Country'],columns=['MC_Pro
duct'])
pivot_table.fillna('0.0')
```

Out[15]:

MC_Product	Back- u. Griesszucke	Braunzucker	Braunzucker Würfel	Frucht/Trauben/Diätz	Gelbzucl
Country					
AE	0.0	0.0	0.0	0.0	0.0
AL	0.0	0.0	0.0	0.0	0.0
AM	0.0	0.0	0.0	0.0	0.0
AT	85.9603	17.4961	0.0	5.89806	16.8758
AU	0.0	0.0	0.0	0.0	0.0
BA	0.0	0.0	0.0	0.0	0.0
BE	0.0	0.0	0.0	0.0	0.0
BG	0.0	1.88889	0.0	0.000222222	0.0
BH	0.0	0.0	0.0	0.0	0.0
BR	0.0	0.0	0.0	0.0	0.0
CA	0.0	0.0	0.0	0.0	0.0
CH	0.0	0.0	0.0	0.0	0.0
CZ	82.6771	3.74639	0.0	1.12622	0.0
DE	0	0.0	0.0	0.0	0.0
EG	0.0	0.0	0.0	0.0	0.0
GA	0.0	0.0	0.0	0.0	0.0
GE	0.0	0.0	0.0	0.0	0.0
GR	0.0	0.0	0.0	0.0	0.0
HR	0.0	0.0	0.0	0.0	0.0
HU	0.0	21.2292	0.0	0.0	0.0
IL	0.0	0.0	0.0	0.0	0.0
IQ	0.0	0.0	0.0	0.0	0.0
IT	0.0	0.0	0.0	0.07	0.0
JO	0.0	0.0	0.0	0.0	0.0
JP	0.0	0.0	0.0	0.0	0.0
KG	0.0	0.0	0.0	0.0	0.0
KW	0.0	0.0	0.0	0.0	0.0
KZ	0.0	0.0	0.0	0.0	0.0
LB	0.0	0.0	0.0	0.0	0.0
LK	0.0	0.0	0.0	0.0	0.0
LY	0.0	0.0	0.0	0.0	0.0

MC_Product	Back- u. Griesszucke	Braunzucker	Braunzucker Würfel	Frucht/Trauben/Diätz	Gelbzucl
Country					
MD	0.0	0.0	0.0	0.0	0.0
MK	0.0	0.0	0.0	0.0	0.0
MX	0.0	0.0	0.0	0.0	0.0
OM	0.0	0.0	0.0	0.0	0.0
PS	0.0	0.0	0.0	0.0	0.0
RO	0.0	48.7693	2.72961	0.0	0.0
RS	0.0	0.0	0.0	0.0	0.0
SA	0.0	0.0	0.0	0.0	0.0
SG	0.0	0.0	0.0	0.0	0.0
SI	0.8	0	0.0	0.0	0.0
SK	2.44444	7.4145	1.64361	0.950222	0.0
SY	0.0	0.0	0.0	0.0	0.0
TJ	0.0	0.0	0.0	0.0	0.0
TM	0.0	0.0	0.0	0.0	0.0
TR	0.0	0.0	0.0	0.0	0.0
UA	0.0	0.0	0.0	0.0	0.0
US	0.0	0.0	0.0	0.0	0.0
UZ	0.0	0.0	0.0	0.0	0.0
XK	0.0	0.0	0.0	0.0	0.0
YE	0.0	0.0	0.0	0.0	0.0
ZA	0.0	0.0	0.0	0.0	0.0

4) Using Statsmodels and SMA to identifying trend

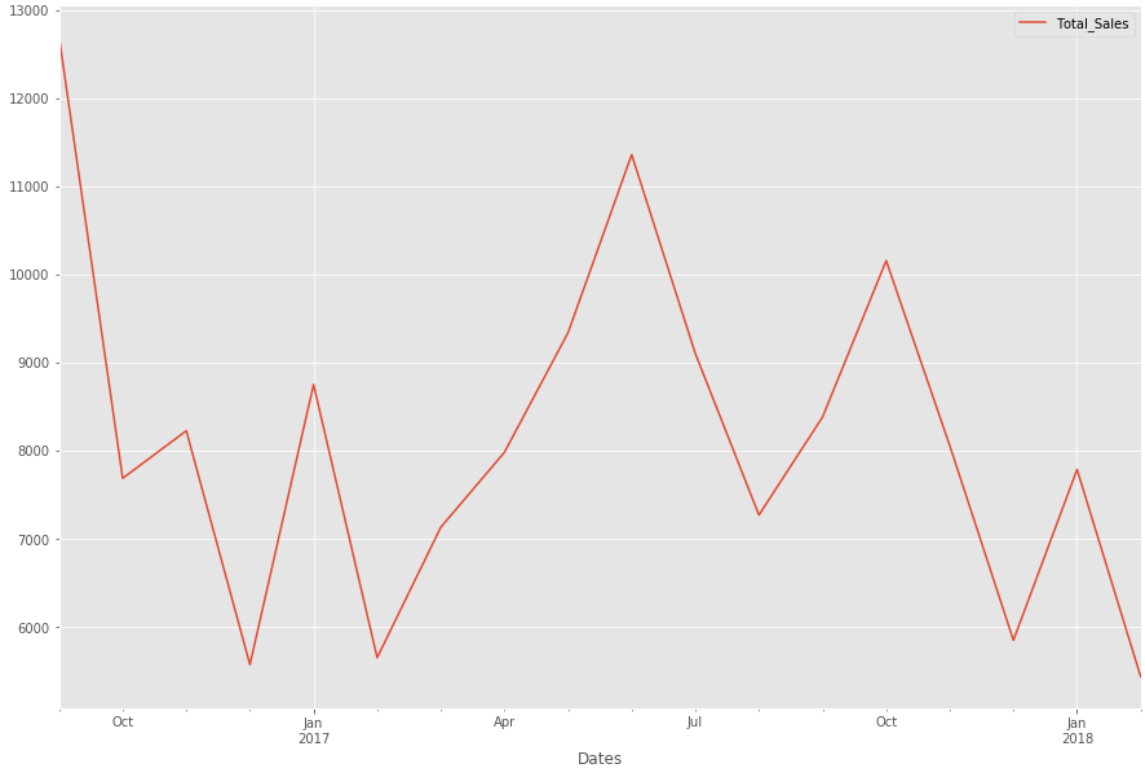
The Hodrick-Prescott filter separates a time-series y_t into a trend τ_t and a cyclical component ζt

In [16]:

```
#choose a Country (AT)
ag_AT= pd.DataFrame(agrana.loc[agrana["Country"]== "AT"])
ag_AT=ag_AT.groupby(by='Dates').agg({'Total_Sales': 'sum'})
ag_AT.plot()
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd384630>
```



In [17]:

```
ag_AT
```

Out[17]:

	Total_Sales
Dates	
2016-09	12701.362517
2016-10	7692.363681
2016-11	8230.521884
2016-12	5578.705081
2017-01	8756.173072
2017-02	5656.894628
2017-03	7136.642718
2017-04	7986.522652
2017-05	9343.598791
2017-06	11363.984839
2017-07	9108.219124
2017-08	7275.591929
2017-09	8386.536281
2017-10	10160.813405
2017-11	8061.613937
2017-12	5854.690375
2018-01	7791.795260
2018-02	5441.803327

In [18]:

```
#use the filter to return a trend
result = sm.tsa.filters.hpfilter(ag_AT.Total_Sales)
```

In [19]:

```
# Tuple unpacking
gdp_ciclo, gdp_trend = sm.tsa.filters.hpfilter(ag_AT.Total_Sales)
```

In [20]:

```
#create a new column for a trend
ag_AT["Trend"] = gdp_trend
ag_AT.head()
```

Out[20]:

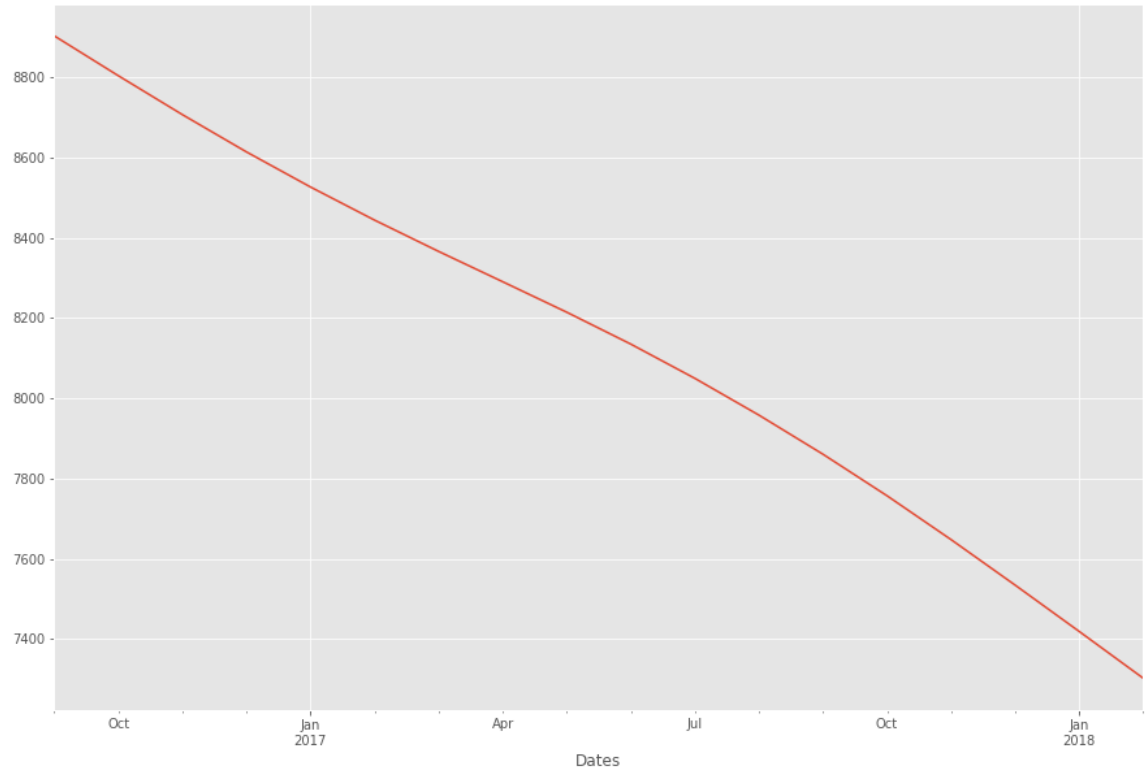
	Total_Sales	Trend
Dates		
2016-09	12701.362517	8902.487840
2016-10	7692.363681	8803.207271
2016-11	8230.521884	8706.301000
2016-12	5578.705081	8613.449044
2017-01	8756.173072	8526.034062

In [21]:

```
#decreasing trend
ag_AT['Trend'].plot()
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd433b00>
```

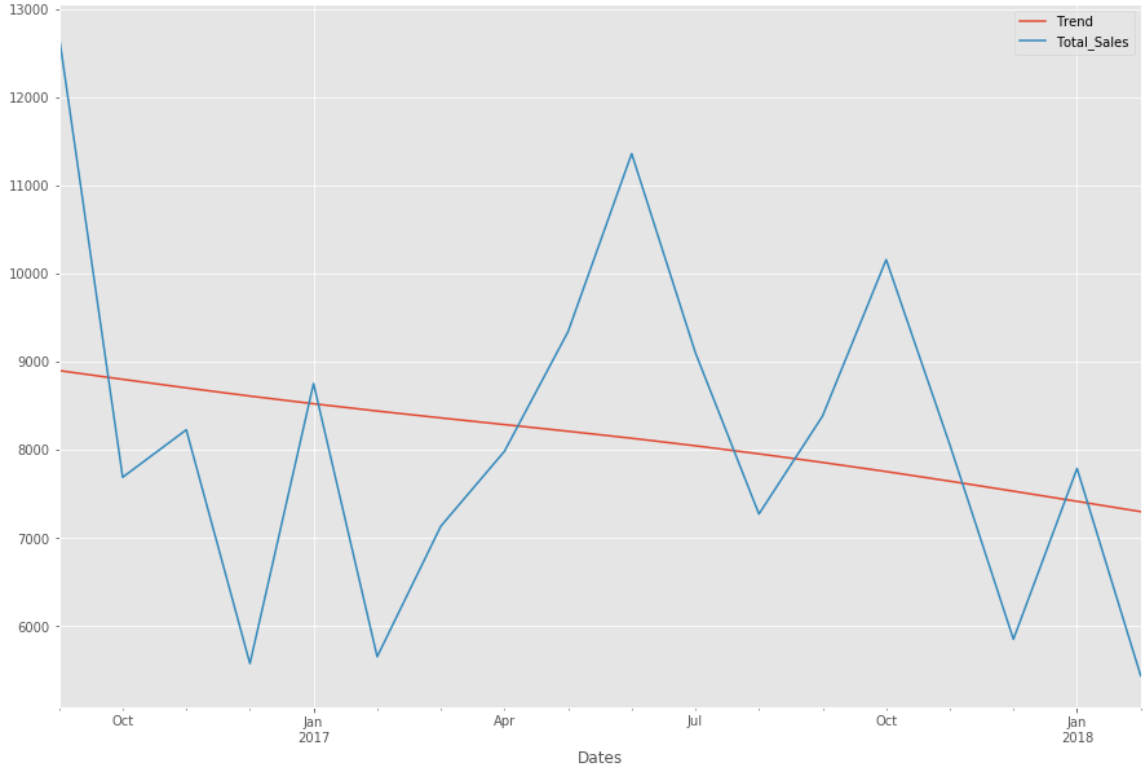


In [22]:

```
#trend x Total Sales
ag_AT[['Trend','Total_Sales']].plot()
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd4b98d0>
```



In [23]:

```
ag_AT.head()
```

Out[23]:

	Total_Sales	Trend
Dates		
2016-09	12701.362517	8902.487840
2016-10	7692.363681	8803.207271
2016-11	8230.521884	8706.301000
2016-12	5578.705081	8613.449044
2017-01	8756.173072	8526.034062

ETS Decomposition

5) Time Series Decomposition

We can also decompose the time series into trend and seasonality

In [24]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

ag_AT.index = ag_AT.index.to_datetime()

ag_AT.head()
```

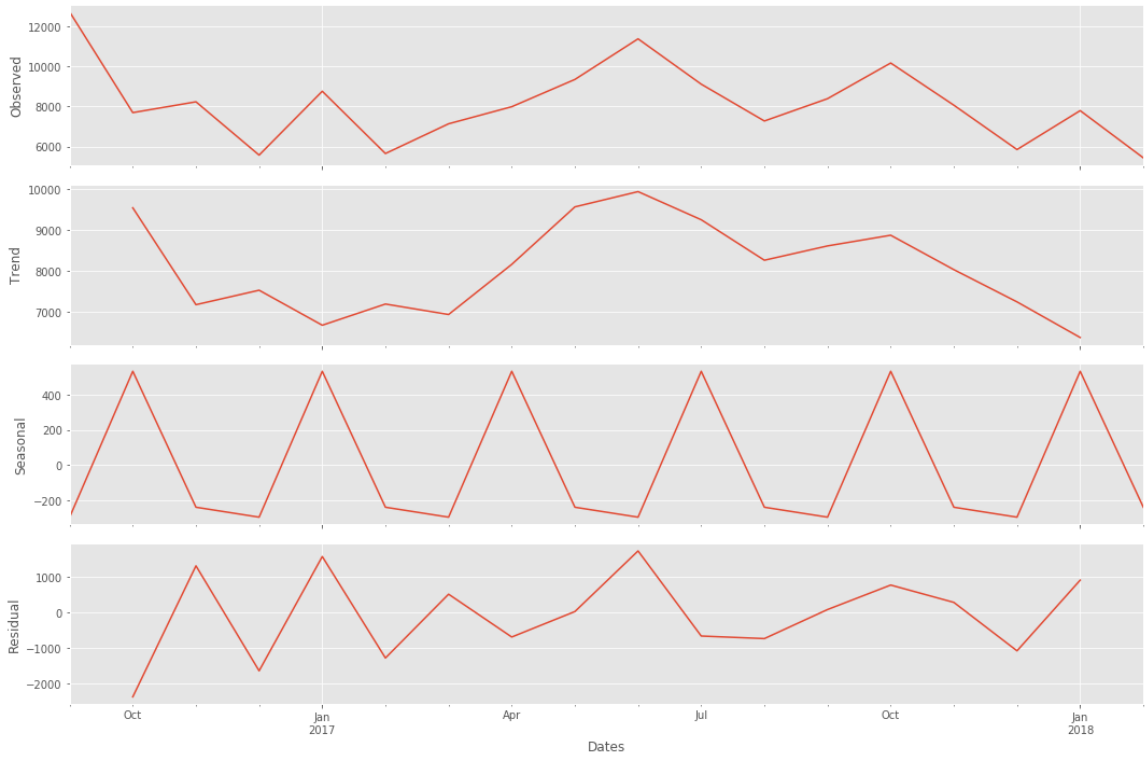
```
C:\Users\Gabriele\Anaconda3\lib\site-packages\ipykernel\__main__.py:3: Fut
ureWarning: to_datetime is deprecated. Use self.to_timestamp(...)
  app.launch_new_instance()
```

Out[24]:

	Total_Sales	Trend
Dates		
2016-09-01	12701.362517	8902.487840
2016-10-01	7692.363681	8803.207271
2016-11-01	8230.521884	8706.301000
2016-12-01	5578.705081	8613.449044
2017-01-01	8756.173072	8526.034062

In [27]:

```
decomposition = seasonal_decompose(ag_AT.Total_Sales, model = "additive", freq=3)

fig= decomposition.plot()

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



Apply Test Duckey Fuller to confirm the series is not stationary

In [38]:

```
import warnings
warnings.filterwarnings('ignore')

ts= ag_AT.Total_Sales
def adf_check(time_series):
    """
    Pass in a time series, returns ADF report
    Passar em uma série temporal, retorna o relatório do ADF
    """
    result = adfuller(time_series)
    print('Augmented Dickey-Fuller Test:')
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used'
]

    for value,label in zip(result,labels):
        print(label+' : '+str(value) )

    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis, reject the null hypothesis.
 Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indi
cating it is non-stationary ")
```

In [39]:

```
import warnings
warnings.filterwarnings('ignore')

adf_check(ts)
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : -0.0
p-value : 0.95853208606
#Lags Used : 8
Number of Observations Used : 9
weak evidence against null hypothesis, time series has a unit root, indica
ting it is non-stationary
```

Test Statistic <0.05 so is not stationary serie

6) Forecasting Steps

The underlying principle is to model or estimate the trend and seasonality in the series and remove those from the series to get a stationary series. Then statistical forecasting techniques can be implemented on this series. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back.

Estimating and Eliminating Trend

1. Transformation - Take a log, sqrt, cuberoot etc. transformation

Transformation

In [32]:

```
ag_AT['SalesLog'] = np.log(ag_AT.Total_Sales)
ag_AT.head()
```

Out[32]:

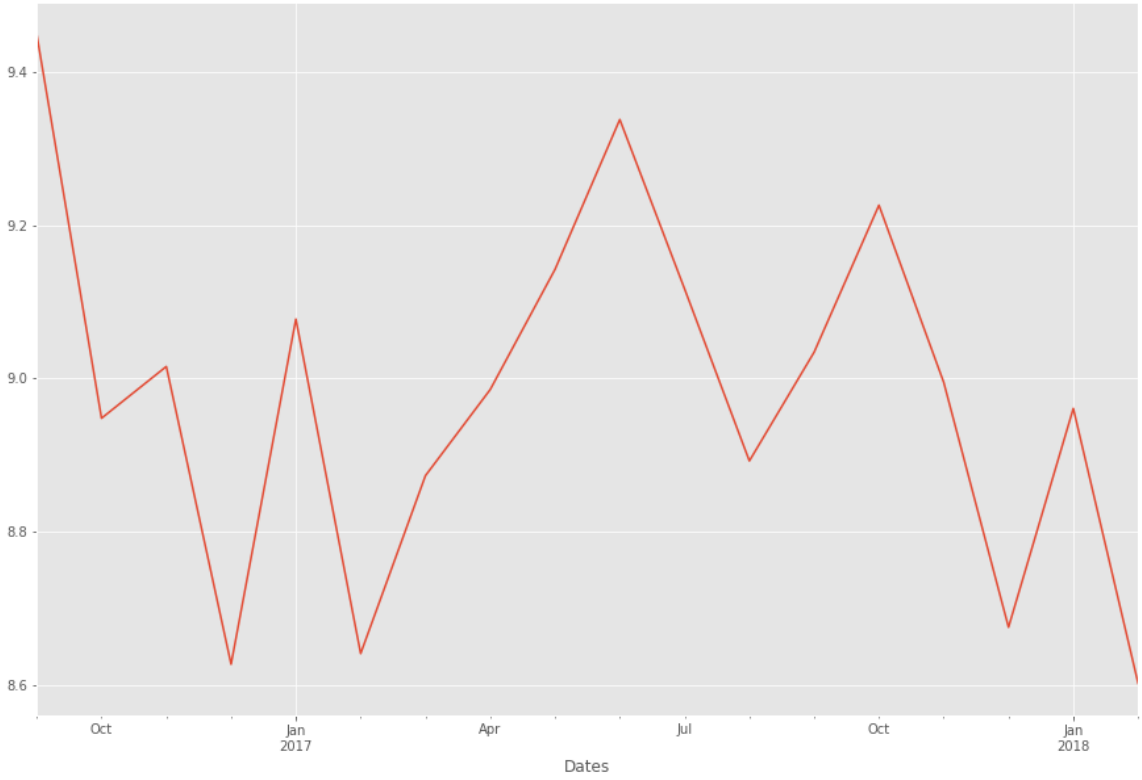
	Total_Sales	Trend	SalesDecomp	SalesLog
Dates				
2016-09-01	12701.362517	8902.487840	NaN	9.449465
2016-10-01	7692.363681	8803.207271	inf	8.947983
2016-11-01	8230.521884	8706.301000	inf	9.015605
2016-12-01	5578.705081	8613.449044	inf	8.626712
2017-01-01	8756.173072	8526.034062	inf	9.077514

In [33]:

```
ag_AT.SalesLog.plot()
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe689400>
```



Basic Time Series Model

We will build a time-series forecasting model to get a forecast for AT. Let us start with the three most basic models -

1. Mean Constant Model

Mean Model

This very simple forecasting model will be called the "mean model"

In [34]:

```
model_mean_pred = ag_AT.SalesLog.mean()

# Let us store this as our Mean Predication Value
ag_AT["SalesMean"] = np.exp(model_mean_pred)

ag_AT.head()
```

Out[34]:

	Total_Sales	Trend	SalesDecomp	SalesLog	SalesMean
Dates					
2016-09-01	12701.362517	8902.487840	NaN	9.449465	7924.789309
2016-10-01	7692.363681	8803.207271	inf	8.947983	7924.789309
2016-11-01	8230.521884	8706.301000	inf	9.015605	7924.789309
2016-12-01	5578.705081	8613.449044	inf	8.626712	7924.789309
2017-01-01	8756.173072	8526.034062	inf	9.077514	7924.789309

Can we measure the error rate?

We will use Root Mean Squared Error (RMSE) to calculate our error values

$RMSE = \Sigma\sqrt{(\hat{y} - y)^2/n}$, where $\hat{y}$ is predicted value of y

In [36]:

```
def RMSE(predicted, actual):
    mse = (predicted - actual)**2
    rmse = np.sqrt(mse.sum()/mse.count())
    return rmse
```

In [37]:

```
model_mean_RMSE = RMSE(ag_AT.SalesMean, ag_AT.Total_Sales)
model_mean_RMSE

# Save this in a dataframe
ag_AT_ResultsMean = pd.DataFrame(columns = ["Model", "Forecast", "RMSE"])
ag_AT_ResultsMean.head()

ag_AT_ResultsMean.loc[0,"Model"] = "Mean"
ag_AT_ResultsMean.loc[0,"Forecast"] = np.exp(model_mean_pred)
ag_AT_ResultsMean.loc[0,"RMSE"] = model_mean_RMSE
ag_AT_ResultsMean
```

Out[37]:

	Model	Forecast	RMSE
0	Mean	7924.79	1919.05

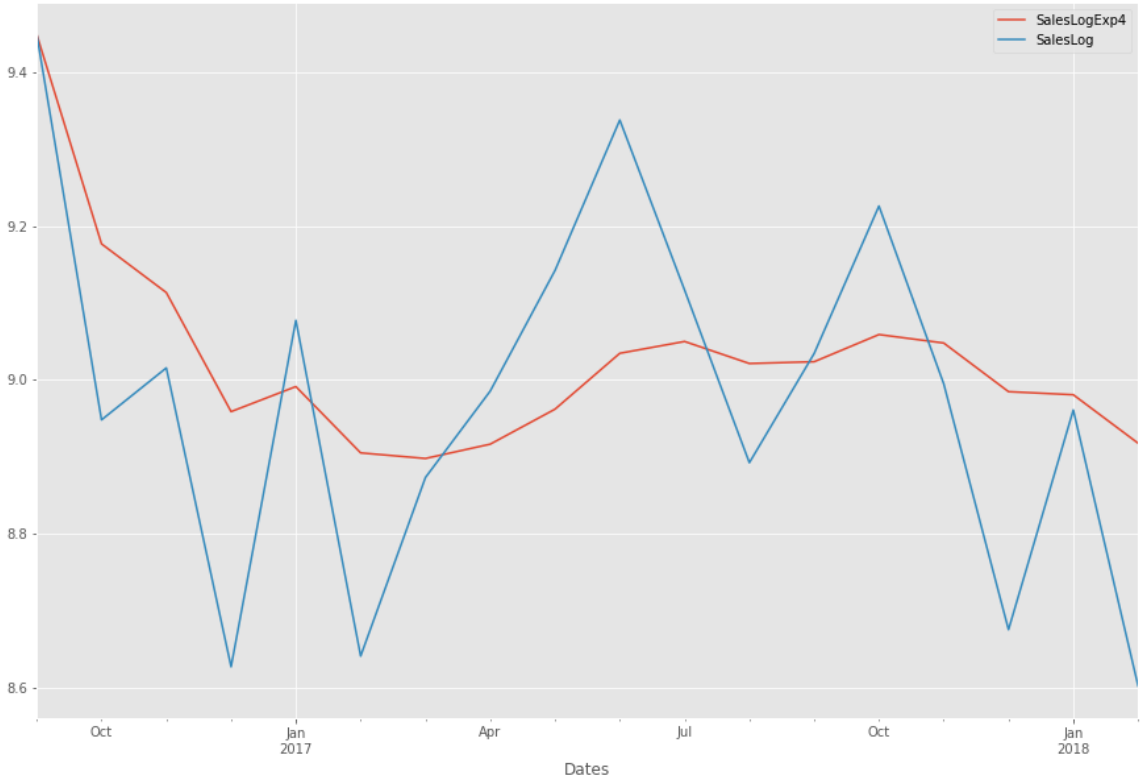
Simple Exponential Smoothing Model (SES)

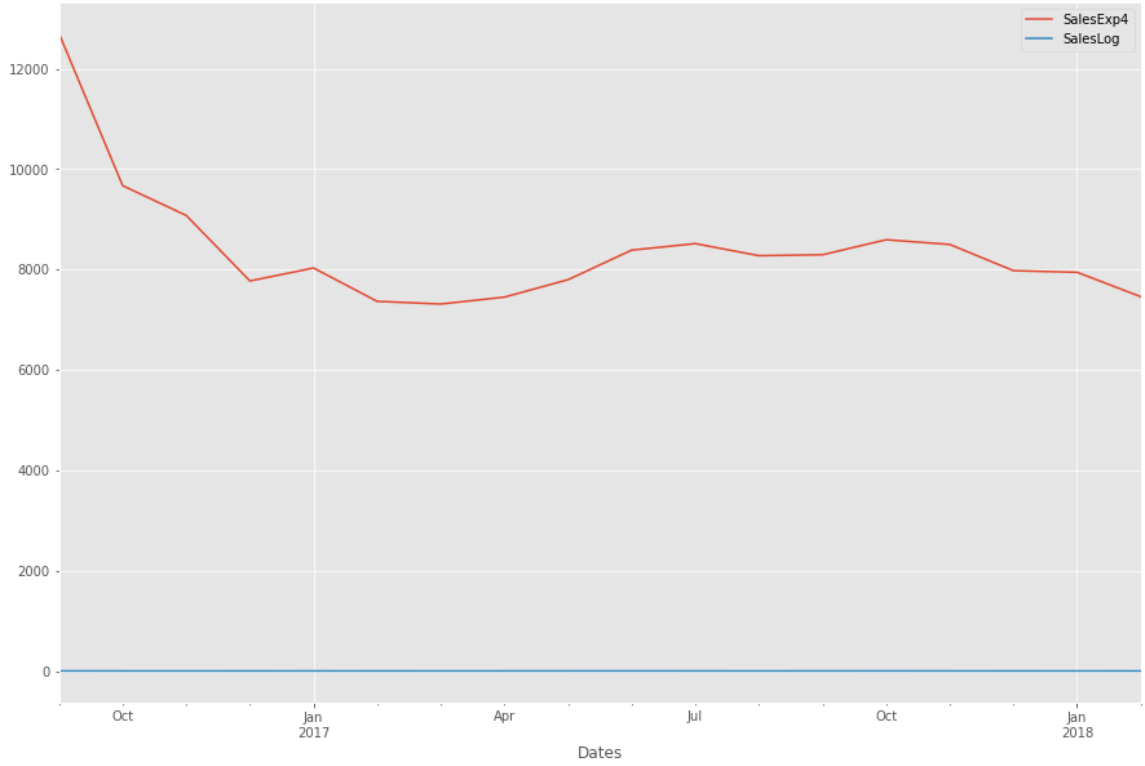
$$\hat{y_t} = \alpha y_{t-1} + (1-\alpha)\hat{y_{t-1}}$$

In [67]:

```
ag_AT['SalesLogExp4'] = pd.ewma(ag_AT.SalesLog, halflife=4)

halflife = 4
alpha = 1 - np.exp(np.log(0.5)/halflife)
alpha

#dfBang.plot(kind ="line", y=["priceModLogExp12", "priceModLog"])
ag_AT[['SalesLogExp4','SalesLog']].plot()

#cria 1 coluna para EXmean
ag_AT["SalesExp4"] = np.exp(ag_AT.SalesLogExp4)
ag_AT.tail()

# Root Mean Squared Error (RMSE)
model_Exp4_RMSE = RMSE(ag_AT.SalesExp4, ag_AT.Total_Sales)
model_Exp4_RMSE

y_exp = ag_AT.SalesLog[-1]
y_exp

y_for = ag_AT.SalesLogExp4[-1]
y_for

model_Exp4_forecast = alpha * y_exp + (1 - alpha) * y_for


ag_AT_ResultsMean.loc[2,"Model"] = "Exp Smoothing 4"
ag_AT_ResultsMean.loc[2,"Forecast"] = np.exp(model_Exp4_forecast)
ag_AT_ResultsMean.loc[2,"RMSE"] = model_Exp4_RMSE
ag_AT_ResultsMean.head()


ag_AT[['SalesExp4','SalesLog']].plot()

# Test remaining part for Stationary (Teste a parte restante para estacionário)
ts = ag_AT.SalesLog - ag_AT.SalesLogExp4
ts.dropna(inplace = True)
adf_check(ts)
```

```
Augmented Dickey-Fuller Test:
ADF Test Statistic : 0.0
p-value : 0.95853208606
#Lags Used : 8
Number of Observations Used : 9
weak evidence against null hypothesis, time series has a unit root, indica
ting it is non-stationary
```





Periodicity and Autocorrelation

Find values (p,d,q) to plot ARIMA model

In [83]:

```
#ACF and PACF plots:
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(ts, nlags=16)
lag_pacf = pacf(ts, nlags=16, method='ols')
```

In [84]:

```
#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')

#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```

ARIMA

The computed initial AR coefficients are not stationary You should induce stationarity, choose a different model order, or you can pass your own start_params. Was used 3 models:

In [76]:

```
from statsmodels.tsa.arima_model import ARIMA
ts= ag_AT['Total_Sales']
model = ARIMA(ts, order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

ARIMA Model Results

Dep. Variable:	D.Total_Sales	No. Observations:	17
Model:	ARIMA(5, 1, 0)	Log Likelihood	-153.206
Method:	css-mle	S.D. of innovations	1936.039
Date:	Thu, 14 Jun 2018	AIC	320.411
Time:	12:05:08	BIC	326.244
Sample:	10-01-2016 - 02-01-2018	HQIC	320.991

	coef	std err	z	P>|z|	[0.025	0.975]
const	-280.2441	338.331	-0.828	0.425	-943.361	382.873
ar.L1.D.Total_Sales	-0.4858	0.277	-1.754	0.107	-1.029	0.057
ar.L2.D.Total_Sales	-0.1199	0.306	-0.391	0.703	-0.721	0.481
ar.L3.D.Total_Sales	-0.2486	0.309	-0.805	0.438	-0.854	0.357
ar.L4.D.Total_Sales	0.1919	0.321	0.598	0.562	-0.437	0.821
ar.L5.D.Total_Sales	0.0804	0.302	0.266	0.795	-0.511	0.672

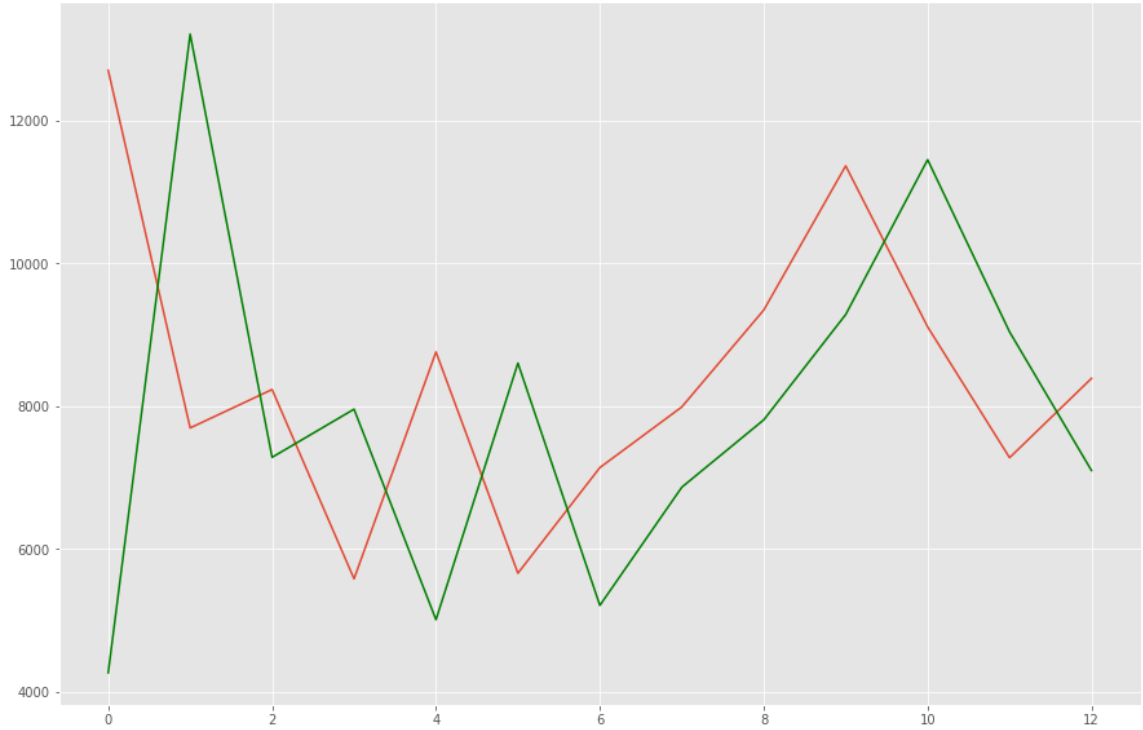
Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.1094	-1.3251j	1.3296	-0.2369
AR.2	0.1094	+1.3251j	1.3296	0.2369
AR.3	1.8321	-0.0000j	1.8321	-0.0000
AR.4	-1.1783	-0.0000j	1.1783	-0.5000
AR.5	-3.2607	-0.0000j	3.2607	-0.5000

In [80]:

```
#PLOT predictions ARIMA
size = int(len(X) * 0.76)
test, train = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
predictions
for t in range(len(test)):
    model= ARIMA(history, order= (0,1,0)) #this is pdq? wich values I must use? 5,0,1
 o q ele usou
    model_fit= model.fit(disp=1)# what mean disp?
    output= model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)

# plot
from matplotlib import pyplot
pyplot.plot(test)
pyplot.plot(predictions, color='g')#previsoes em verde
pyplot.show()
```

```
predicted=7095.265587, expected=8386.536281
Test MSE: 11760657.559
```



We can see the graph of forecasts arima in green color

In [81]:

```
forecasts = np.array(predictions)
forecasts
```

Out[81]:

```
array([[  4262.05080697],
       [ 13209.47233999],
       [  7280.95539321],
       [  7954.76595217],
       [  5005.94154072],
       [  8600.10192387],
       [  5206.50275042],
       [  6861.71810967],
       [  7805.33175544],
       [  9280.736128  ],
       [ 11449.92565553],
       [  9038.04617202],
       [  7095.26558721]])
```

SEries is not stationary.