In [1]:

```
import warnings

import matplotlib as mpl
import numpy as np
import pandas as pd  # Basic library for all of our dataset operations
import statsmodels as sm
import tensorflow as tf
from matplotlib import pyplot as plt
from pylab import rcParams
from sklearn.linear_model import LinearRegression
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa import api as smt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller


# We will use deprecated models of statmodels which throw a lot of warnings to use more
warnings.filterwarnings("ignore")


# Extra settings
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
plt.style.use('bmh')
mpl.rcParams['axes.labelsize'] = 14
mpl.rcParams['xtick.labelsize'] = 12
mpl.rcParams['ytick.labelsize'] = 12
mpl.rcParams['text.color'] = 'k'
print(tf.__version__)
```

```
2.3.0
```

📚 Time series analysis and transforms

Load the dataset and quick preview

In [2]:

```
air_pollution = pd.read_csv('C:/Users/hp/Downloads/time-series-forecasting-with-python-
air_pollution.set_index('date', inplace=True)
air_pollution.head()
```

Out[2]:

date	pollution_today	dew	temp	press	wnd_spd	snow	rain	pollution_yeste
2010-01-02	145.958333	-8.500000	-5.125000	1024.750000	24.860000	0.708333	0.0	10.04
2010-01-03	78.833333	-10.125000	-8.541667	1022.791667	70.937917	14.166667	0.0	145.95
2010-01-04	31.333333	-20.875000	-11.500000	1029.291667	111.160833	0.000000	0.0	78.83
2010-01-05	42.458333	-24.583333	-14.458333	1033.625000	56.920000	0.000000	0.0	31.33

date	pollution_today	dew	temp	press	wnd_spd	snow	rain	pollution_yeste
2010-01-06	56.416667	-23.708333	-12.541667	1033.750000	18.511667	0.000000	0.0	42.45

In [3]:

```
air_pollution.describe()
```

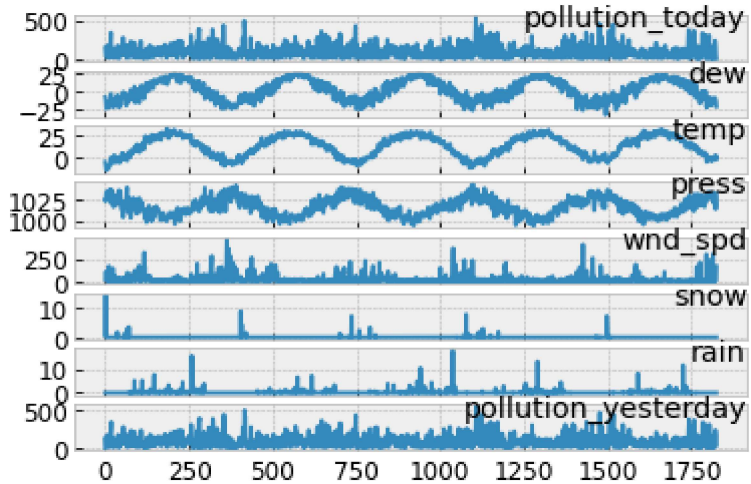
Out[3]:

	pollution_today	dew	temp	press	wnd_spd	snow	rain
count	1825.000000	1825.000000	1825.000000	1825.000000	1825.000000	1825.000000	1825.000000
mean	98.245080	1.828516	12.459041	1016.447306	23.894307	0.052763	0.195023
std	76.807697	14.163508	11.552997	10.076053	41.373161	0.546072	0.993917
min	3.166667	-33.333333	-14.458333	994.041667	1.412500	0.000000	0.000000
25%	42.333333	-10.083333	1.541667	1007.916667	5.904167	0.000000	0.000000
50%	79.166667	2.041667	13.916667	1016.208333	10.953750	0.000000	0.000000
75%	131.166667	15.083333	23.166667	1024.541667	22.235000	0.000000	0.000000
max	541.895833	26.208333	32.875000	1043.458333	463.187917	14.166667	17.583333

Check Each Feature

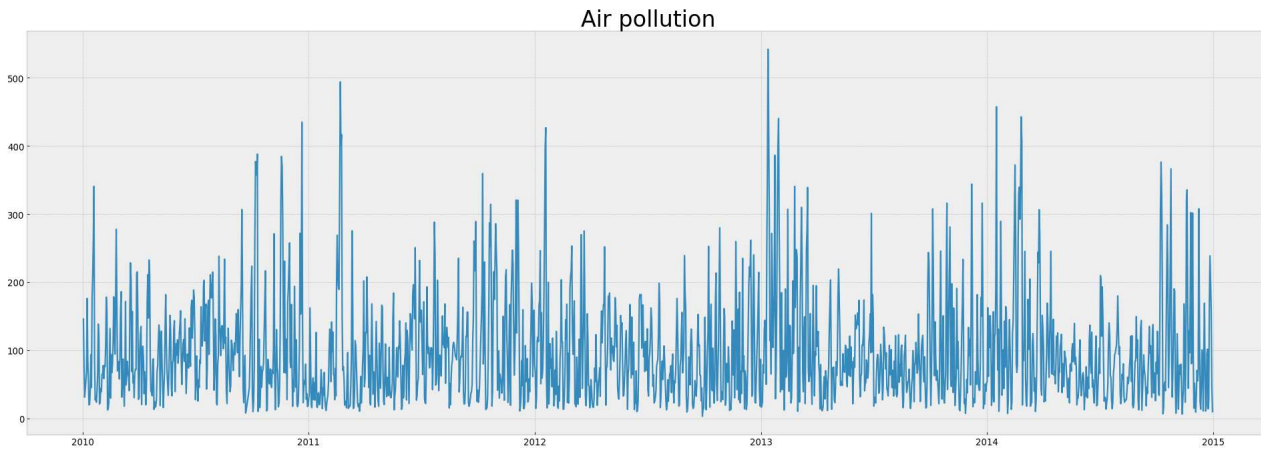
In [4]:

```
values = air_pollution.values
groups = [0, 1, 2, 3, 4, 5, 6, 7]
i = 1
# plot each column
for group in groups:
    plt.subplot(len(groups), 1, i)
    plt.plot(values[:, group])
    plt.title(air_pollution.columns[group], y=0.5, loc='right')
    i += 1


plt.show()
```



In [5]:

```
plt.figure(num=None, figsize=(30, 10), dpi=80, facecolor='w', edgecolor='k')
plt.title('Air pollution', fontsize=30)

plt.plot(air_pollution.pollution_today)
plt.savefig("results/pollution.png")
```



Multiplicative model

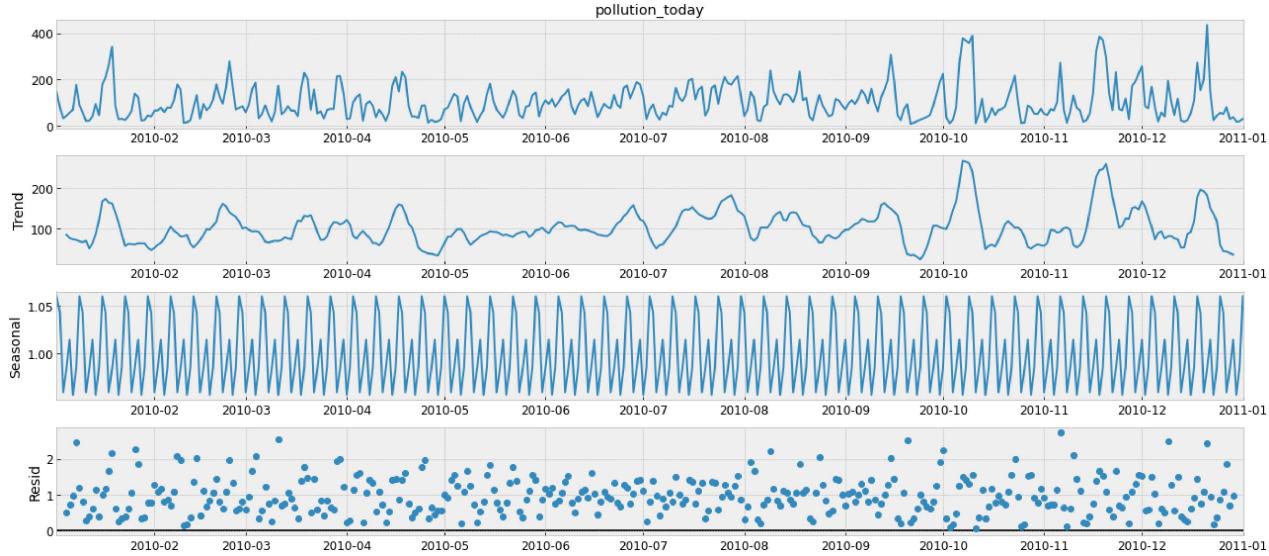
```
y(t) = Level * Trend * Seasonality * Noise
```

Multiplicatives models are nonlinear,trend is curved and seasonality is not constant. Change is not constant over time

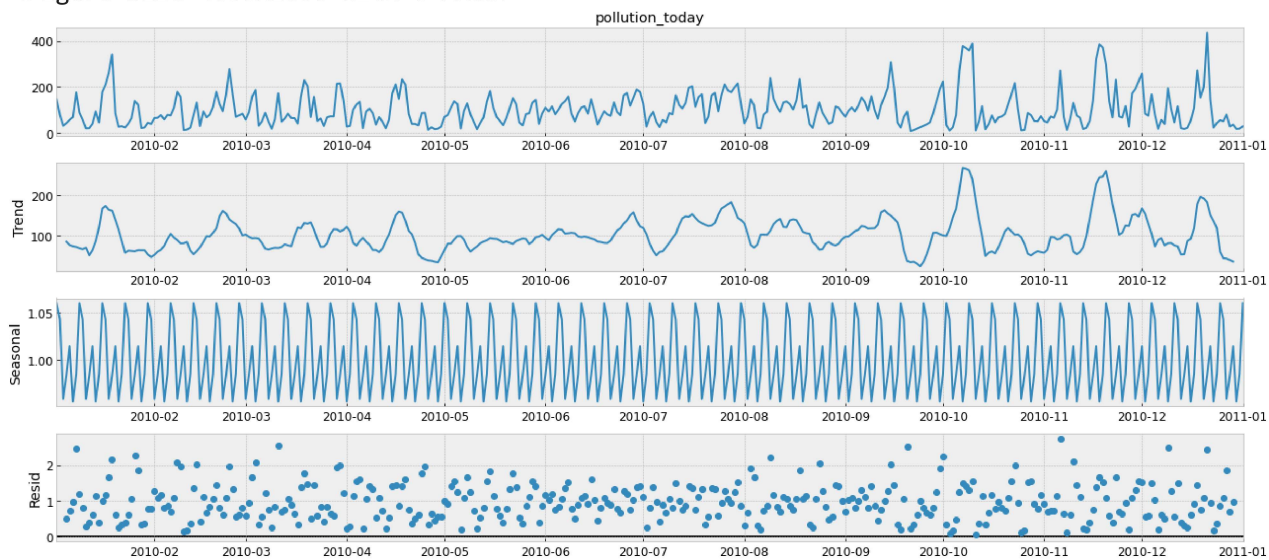
In [7]:

```
rcParams['figure.figsize'] = 18, 8
plt.figure(num=None, figsize=(50, 20), dpi=80, facecolor='w', edgecolor='k')
series = air_pollution.pollution_today[:365]
result = seasonal_decompose(series, model='multiplicative')
result.plot()
```

Out[7]:




```
<Figure size 4000x1600 with 0 Axes>
```



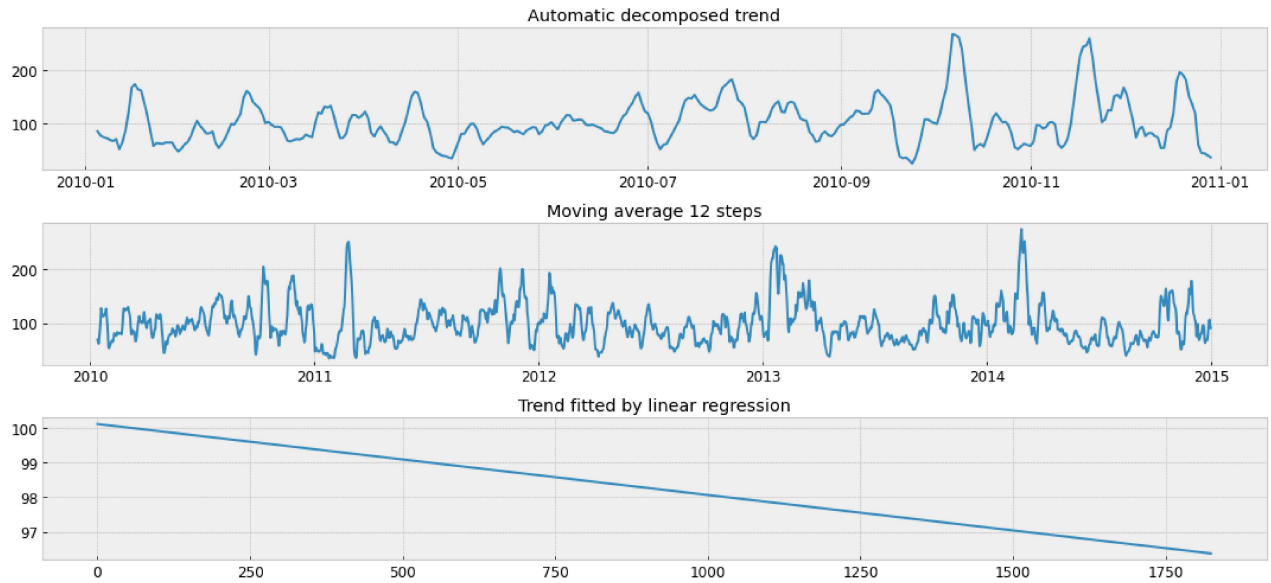
some methods to check for trend in our series:

- Automatic decomposing
- Moving average
- Fit a linear regression model to identify trend

In [8]:

```
fig = plt.figure(figsize=(15, 7))
layout = (3, 2)
pm_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
mv_ax = plt.subplot2grid(layout, (1, 0), colspan=2)
fit_ax = plt.subplot2grid(layout, (2, 0), colspan=2)

pm_ax.plot(result.trend)
pm_ax.set_title("Automatic decomposed trend")

mm = air_pollution.pollution_today.rolling(12).mean()
mv_ax.plot(mm)
mv_ax.set_title("Moving average 12 steps")


X = [i for i in range(0, len(air_pollution.pollution_today))]
```

```
X = np.reshape(X, (len(X), 1))
y = air_pollution.pollution_today.values
model = LinearRegression()
model.fit(X, y)
# calculate trend
trend = model.predict(X)
fit_ax.plot(trend)
fit_ax.set_title("Trend fitted by linear regression")

plt.tight_layout()
```



We can see our series does not have a strong trend, results from both the automatic decomposition and the moving average look more like a seasonality effect + random noise than a trend. This sort of confirmed with our linear regression, which cant find our series properly and gives us a poor trend.

Seasonality

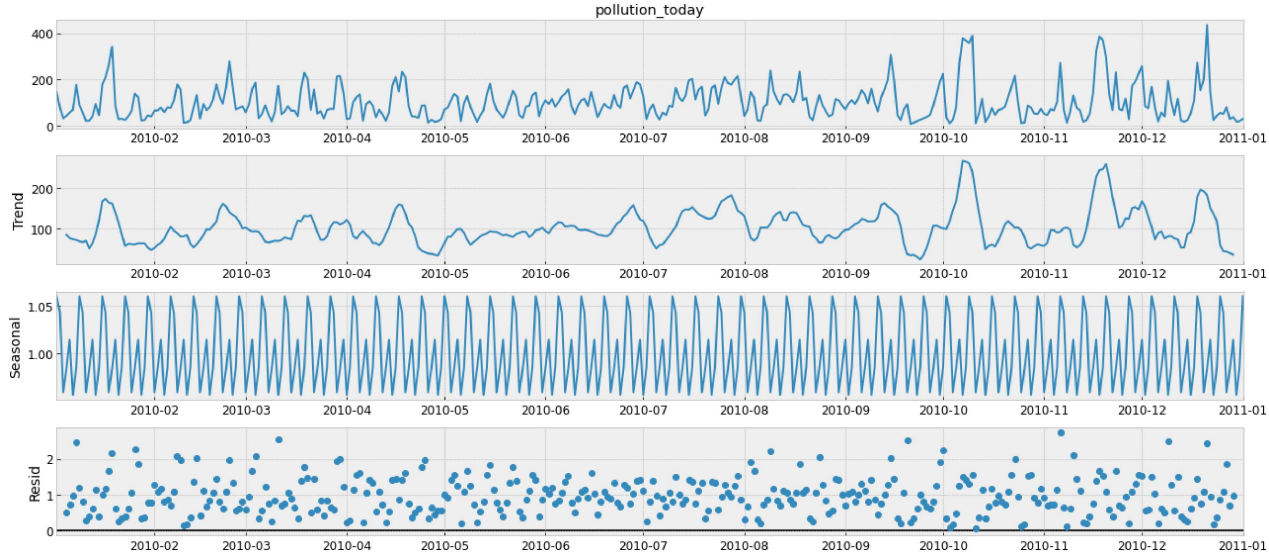
Seasonality is observed when there is a distinct repeated pattern observed between regular intervals due to seasonal factors.

The automatic decomposing chart did not gave a good look into the decomposed seasonality, decomposing smaller parts of the time series first and test seasonalities we found into the others.

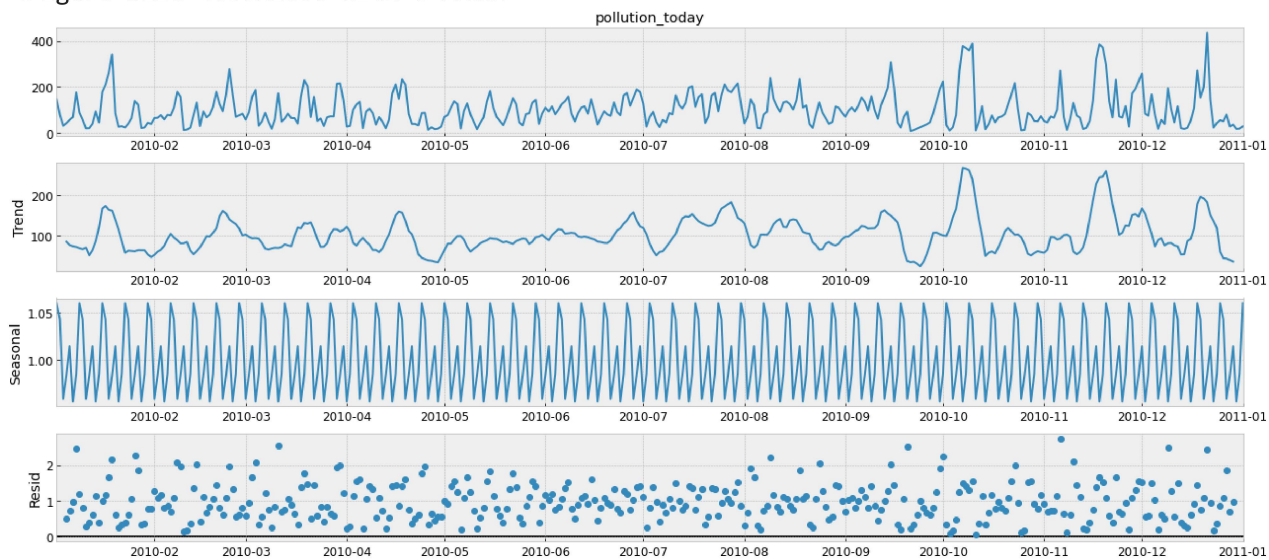
In [9]:

```
rcParams['figure.figsize'] = 18, 8
plt.figure(num=None, figsize=(50, 20), dpi=80, facecolor='w', edgecolor='k')
series = air_pollution.pollution_today[:365]
result = seasonal_decompose(series, model='multiplicative')
result.plot()
```

Out[9]:




```
<Figure size 4000x1600 with 0 Axes>
```

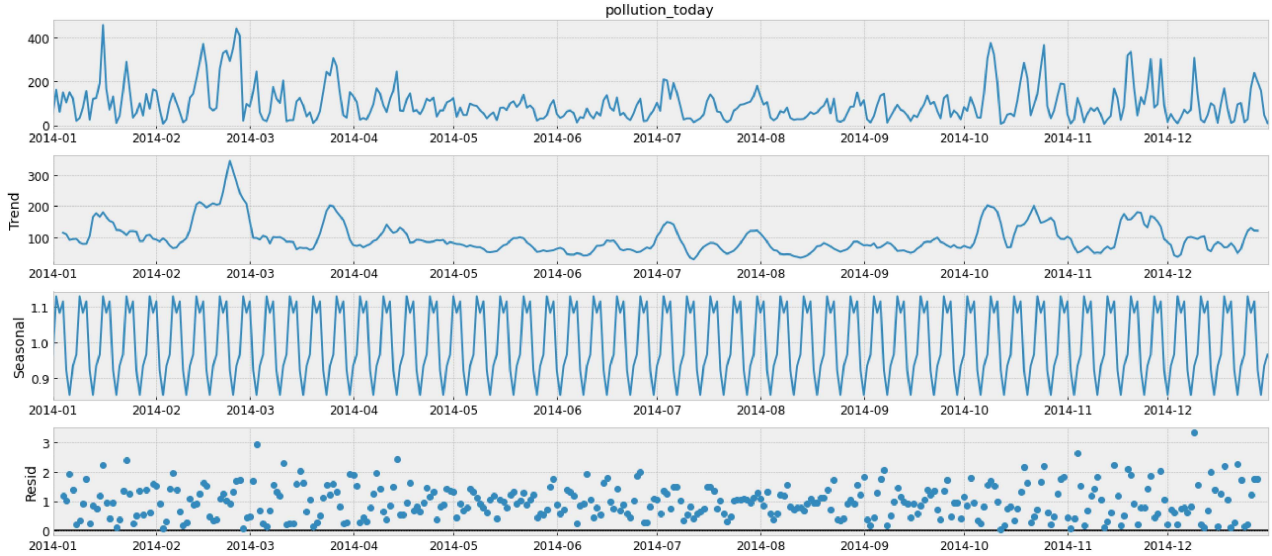


Here can see a clear weekly seasonality, 4 spikes every month (weerkly). Lets check how the last year of data looks

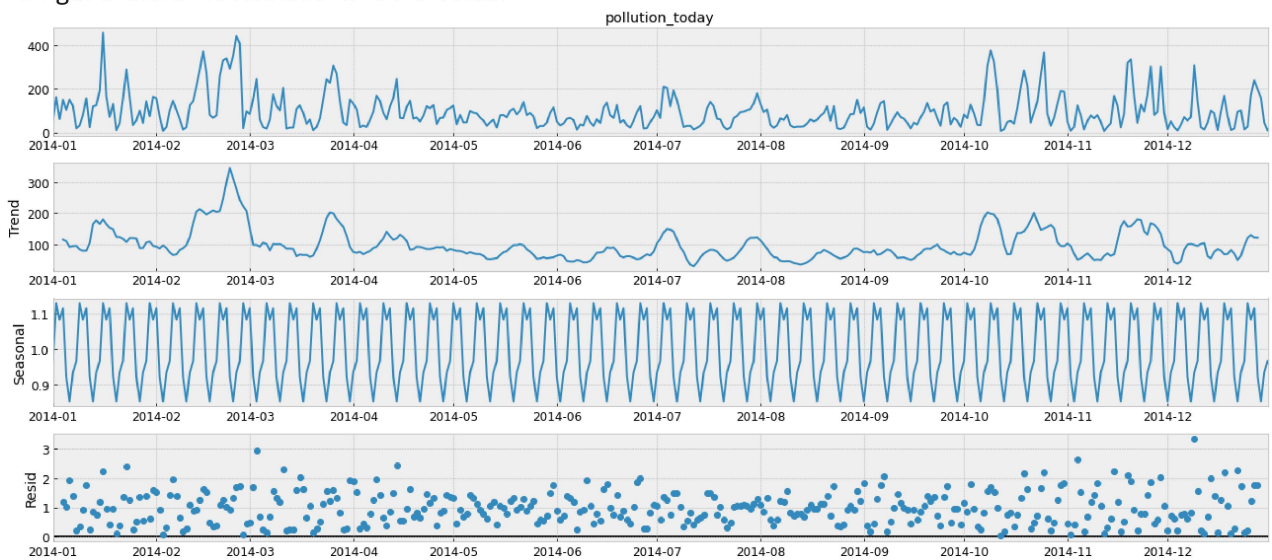
In [10]:

```
rcParams['figure.figsize'] = 18, 8
plt.figure(num=None, figsize=(50, 20), dpi=80, facecolor='w', edgecolor='k')
series = air_pollution.pollution_today[-365:]
result = seasonal_decompose(series, model='multiplicative')
result.plot()
```

Out[10]:




```
<Figure size 4000x1600 with 0 Axes>
```



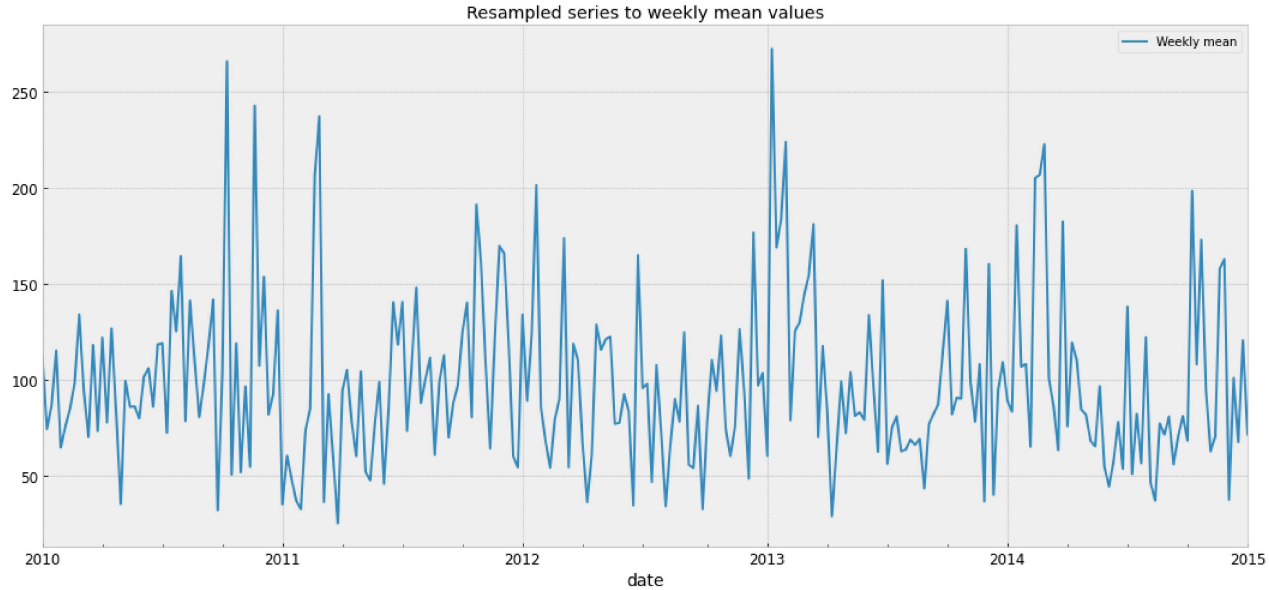
We see another weekly seasonality(4 spikes between every month) but a bit different to the original one, this is something we should always expect from real datasets as their seasonalities will never be perfect but a combination of multiples.

In [11]:

```
# Looking for weekly seasonality
resample = air_pollution.resample('W')
weekly_mean = resample.mean()
weekly_mean.pollution_today.plot(label='Weekly mean')
plt.title("Resampled series to weekly mean values")
plt.legend()
```

Out[11]:

```
<matplotlib.legend.Legend at 0x136cc9d33a0>
```

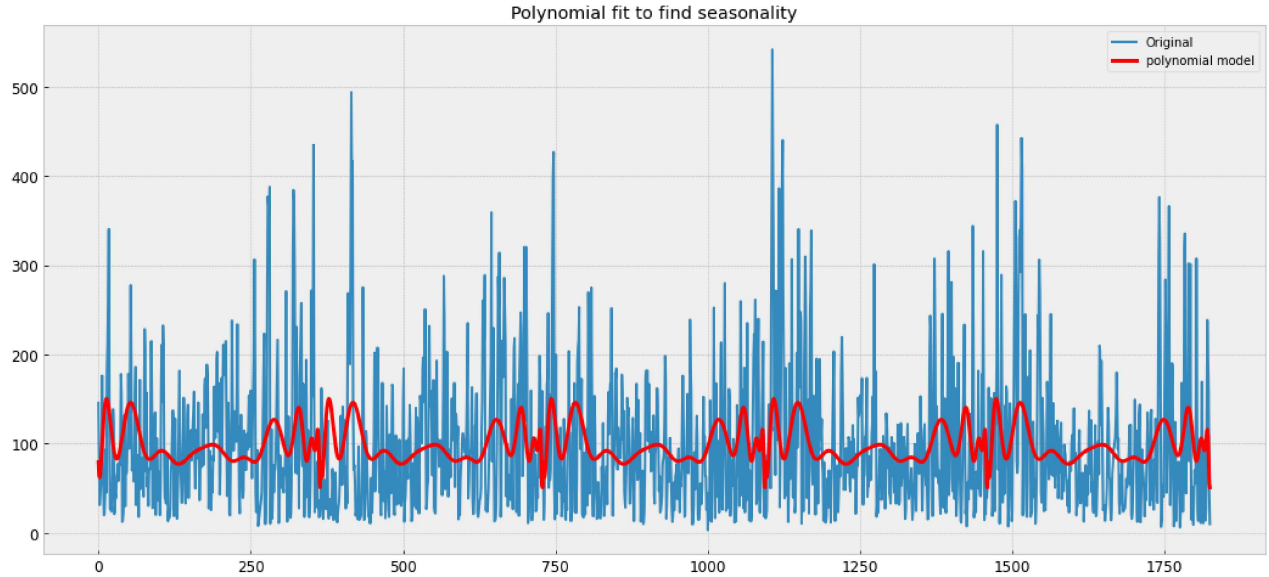


Manual methods to find seasonalities

We can try to generate a model to find the seasonalities for us. One of the most common to use is a simple polynomial model.

In [12]:

```
# Fix xticks to show dates
# fit polynomial: x^2*b1 + x*b2 + ... + bn
series = air_pollution.pollution_today.values
X = [i % 365 for i in range(0, len(series))]
y = series
degree = 100
coef = np.polyfit(X, y, degree)
# create curve
curve = list()
for i in range(len(X)):
    value = coef[-1]
    for d in range(degree):
        value += X[i]**(degree-d) * coef[d]
    curve.append(value)
# plot curve over original data
plt.plot(series, label='Original')
plt.plot(curve, color='red', linewidth=3, label='polynomial model')
plt.legend()
plt.title("Polynomial fit to find seasonality")
plt.show()
```



We can see how the model to find a seasonality fits poorly to our data. Is going to be a complicate time series to model :P

Noise

check if our series has white noise?

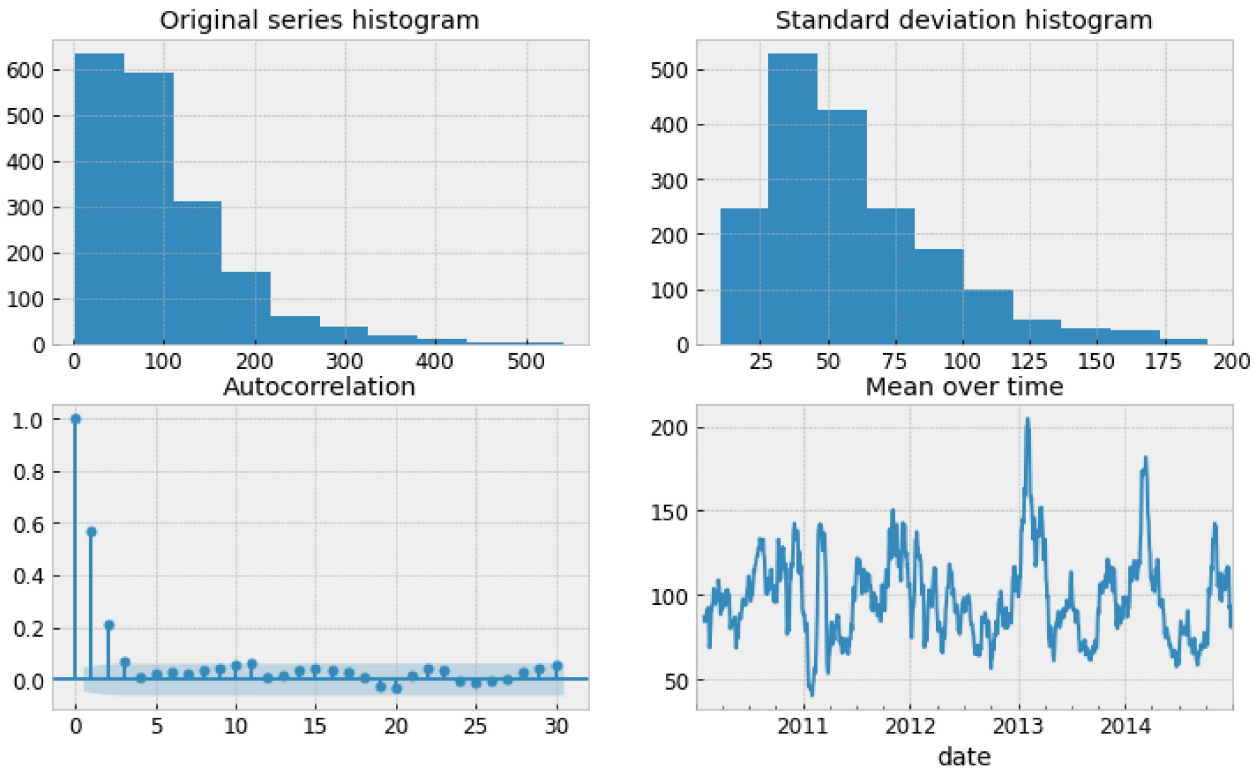
- Check our series histogram, does it look like a Gaussian distribution? Mean=0 and constand std
- Correlation plots
- Standard deviation distribution, is it a Gaussian distribution?
- Does the mean or level change over time?

In [13]:

```
fig = plt.figure(figsize=(12, 7))
layout = (2, 2)
hist_ax = plt.subplot2grid(layout, (0, 0))
ac_ax = plt.subplot2grid(layout, (1, 0))
hist_std_ax = plt.subplot2grid(layout, (0, 1))
mean_ax = plt.subplot2grid(layout, (1, 1))

air_pollution.pollution_today.hist(ax=hist_ax)
hist_ax.set_title("Original series histogram")

plot_acf(series, lags=30, ax=ac_ax)
ac_ax.set_title("Autocorrelation")

mm = air_pollution.pollution_today.rolling(7).std()
mm.hist(ax=hist_std_ax)
hist_std_ax.set_title("Standard deviation histogram")

mm = air_pollution.pollution_today.rolling(30).mean()
mm.plot(ax=mean_ax)
mean_ax.set_title("Mean over time")
```

Out[13]:

```
Text(0.5, 1.0, 'Mean over time')
```



We can see our series do not follow a Gaussian distribution from the histogram and neither the standard deviation, though the std does has the mean more centered which shows a small part of white noise that is not possible to split from the original series.

We also have a small correlation with close measurements in time but not present with distant measurements (this could also indicate low seasonality). The mean over time also shows something similar with a constant value and high peaks in the same moments for the 4 years (smaller in 2012)

We could say our series does contain a small part of white noise but it is really small and hard to remove

Stationarity

Stationarity is an important characteristic of time series. A time series is stationarity if it has constant mean and variance over time. Most models work only with stationary data as this makes it easier to model. Not all time series are stationary but we can transform them into stationary series in different ways.

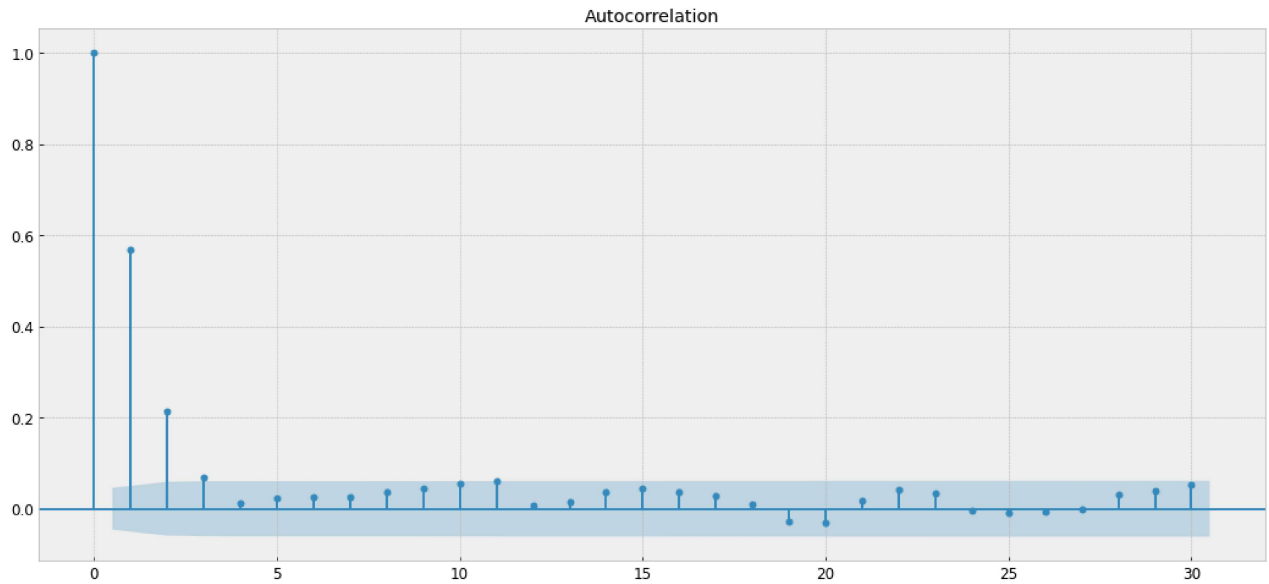
Often, stock prices are not a stationary process, since we might see a growing trend, or its volatility might increase over time (meaning that variance is changing).

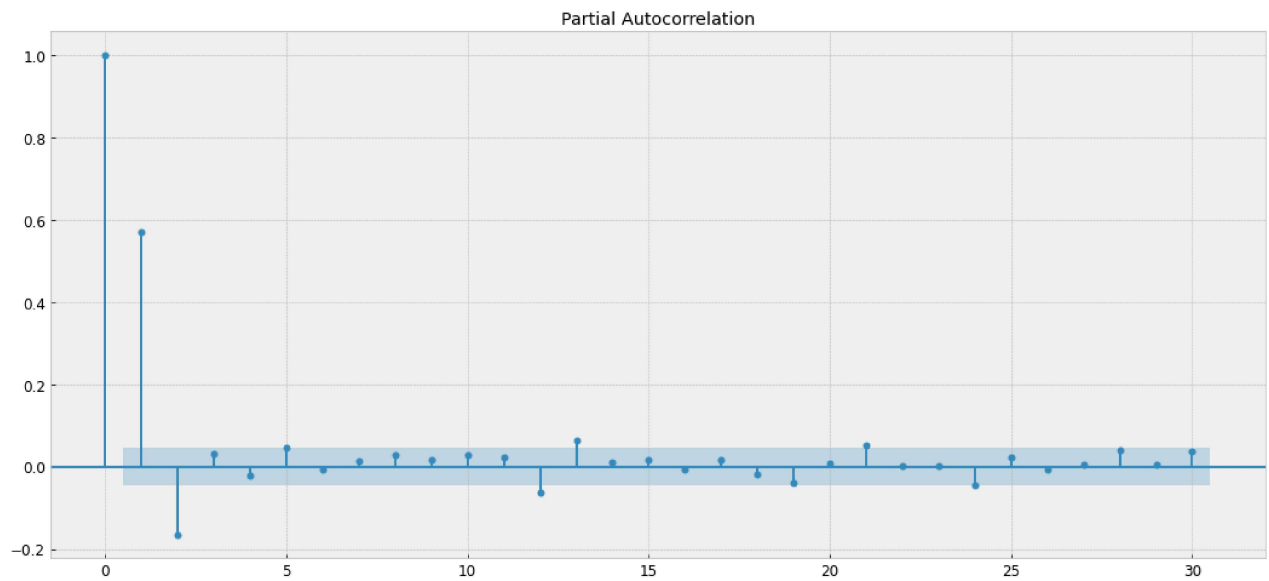
Check for sationarity

Autocorrelation and Partial autocorrelation plots

In [14]:

```
plot_acf(series, lags=30)
plot_pacf(series, lags=30)
plt.show()
```



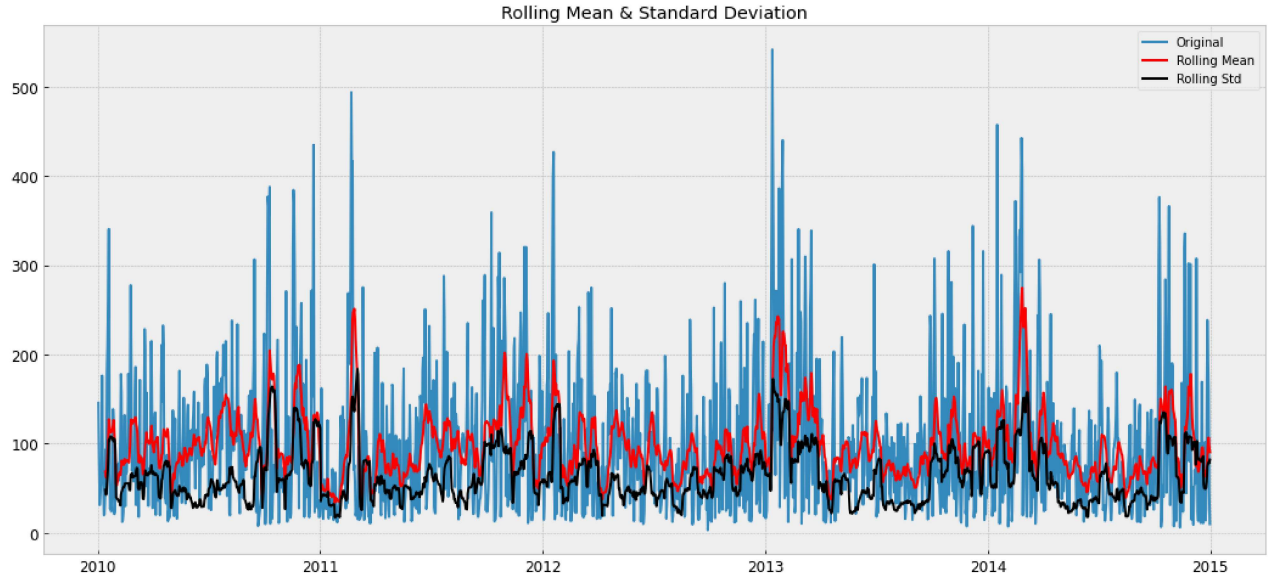


our time series values are not correlated with distant points in time, this is good and shows us our series should be stationary but for the shake of learning and confirming we will test with some other methods

Rolling means and standard deviation of our series

In [15]:

```
# Determing rolling statistics
rolmean = air_pollution.pollution_today.rolling(window=12).mean()
rolstd = air_pollution.pollution_today.rolling(window=12).std()

# Plot rolling statistics:
orig = plt.plot(air_pollution.pollution_today, label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```



our mean and standar deviation have a constant behaviour over the years, even if they change over the year this behaviour is then repeated next year. This proves us again a stationary series

Making Time Series Stationary

we got lucky with this dataset and is already stationary, but what happens when this is not the case? We included a dummy dataset called `international_airline_passengers.csv` on the datasets folders which is not stationary and we will apply some methods in this section to transform it into a stationary series.

passengers = pd.read_csv("C:/Users/hp/Downloads/time-series-forecasting-with-python-master/datasets/international_airline_passengers.csv") passengers.passengers.plot(label='Original') passengers.passengers.rolling(window=12).mean().plot(color='red', label='Windowed mean') passengers.passengers.rolling(window=12).std().plot(color='black', label='Std mean') plt.legend() plt.title('Original vs Windowed mean vs Windowed std')

Difference transform

Applying a difference transform to a time series could help remove the series dependence on time.

This transform is done by substracting the previous obesvation to the current one.

`difference(t) = observation(t) - observation(t-1)`

Taking the difference between consecutive observations would be a lag-1 difference, we can tweek this lag value to fit our series.

In [18]:

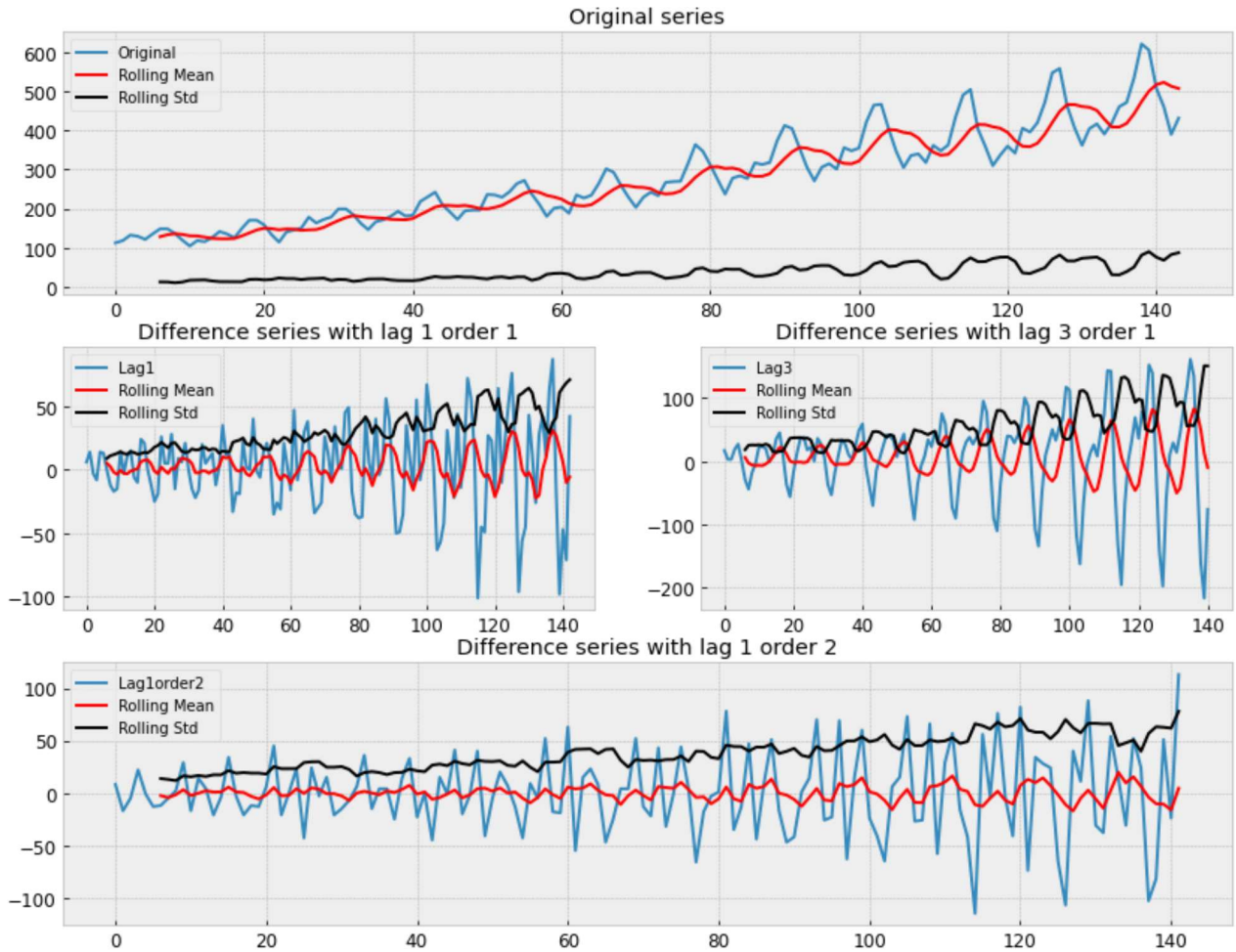
```
def difference(dataset, interval=1, order=1):
    for u in range(order):
        diff = list()
        for i in range(interval, len(dataset)):
            value = dataset[i] - dataset[i - interval]
            diff.append(value)
        dataset = diff
    return diff
```

In [19]:

```
lag1series = pd.Series(difference(passengers.passengers, interval=1, order=1))
lag3series = pd.Series(difference(passengers.passengers, interval=3, order=1))
lag1order2series = pd.Series(difference(
    passengers.passengers, interval=1, order=2))

fig = plt.figure(figsize=(14, 11))
layout = (3, 2)
original = plt.subplot2grid(layout, (0, 0), colspan=2)
lag1 = plt.subplot2grid(layout, (1, 0))
lag3 = plt.subplot2grid(layout, (1, 1))
lag1order2 = plt.subplot2grid(layout, (2, 0), colspan=2)

original.set_title('Original series')
original.plot(passengers.passengers, label='Original')
original.plot(passengers.passengers.rolling(
    7).mean(), color='red', label='Rolling Mean')
original.plot(passengers.passengers.rolling(7).std(),
              color='black', label='Rolling Std')
original.legend(loc='best')

lag1.set_title('Difference series with lag 1 order 1')
lag1.plot(lag1series, label="Lag1")
lag1.plot(lag1series.rolling(7).mean(), color='red', label='Rolling Mean')
lag1.plot(lag1series.rolling(7).std(), color='black', label='Rolling Std')
lag1.legend(loc='best')

lag3.set_title('Difference series with lag 3 order 1')
lag3.plot(lag3series, label="Lag3")
lag3.plot(lag3series.rolling(7).mean(), color='red', label='Rolling Mean')
lag3.plot(lag3series.rolling(7).std(), color='black', label='Rolling Std')
lag3.legend(loc='best')

lag1order2.set_title('Difference series with lag 1 order 2')
lag1order2.plot(lag1order2series, label="Lag1order2")
lag1order2.plot(lag1order2series.rolling(7).mean(),
                color='red', label='Rolling Mean')
lag1order2.plot(lag1order2series.rolling(7).std(),
                color='black', label='Rolling Std')
lag1order2.legend(loc='best')
```

Out[19]:

```
<matplotlib.legend.Legend at 0x136d0025520>
```



We can see how 1 order differencing doesnt really remove stationary but once we go with a order 2 difference it looks closer to a stationary series

Log scale transformation

Applying a log scale transform to a time series could also help remove the series dependence on time.

This transform is done by substracting the previous obesvation to the current one.

```
LogScaleTransform(t)= Log(t)
```

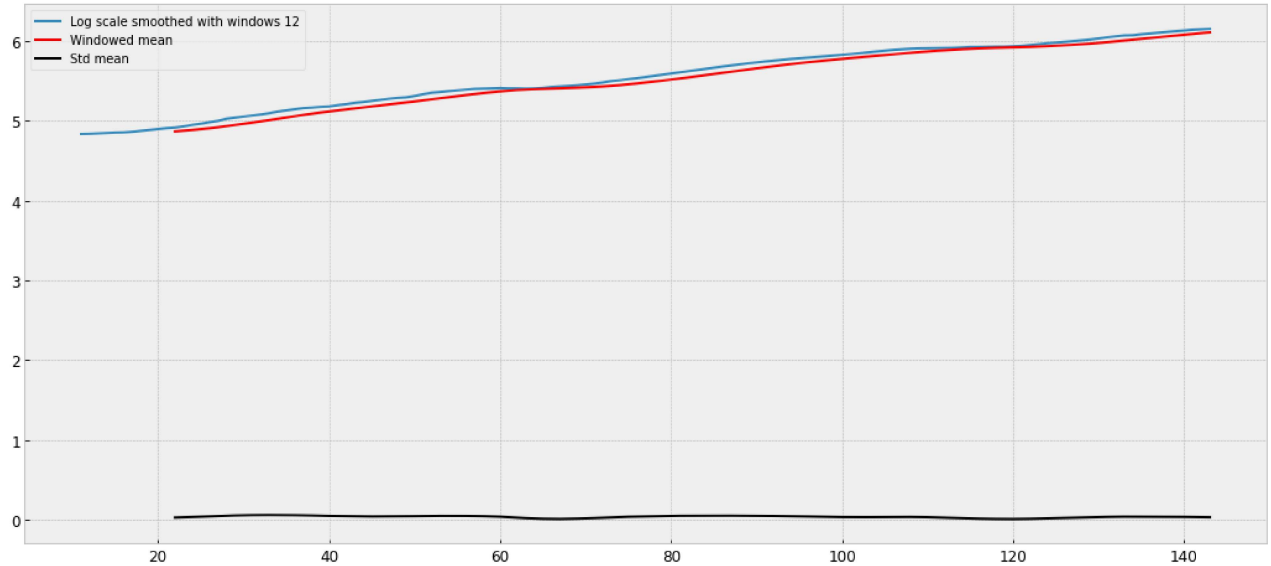
ts_log = np.log(passengers.passengers) ts_log.plot(label='Log scale result') ts_log.rolling(window=12).mean().plot(color='red', label='Windowed mean') ts_log.rolling(window=12).std().plot(color='black', label='Std mean') plt.legend() plt.title('Log scale transformation into original series')

In [22]:

```
#Smoothing
avg = pd.Series(ts_log).rolling(12).mean()
plt.plot(avg, label='Log scale smoothed with windows 12')
avg.rolling(window=12).mean().plot(color='red', label='Windowed mean')
avg.rolling(window=12).std().plot(color='black', label='Std mean')
plt.legend()
```

Out[22]:

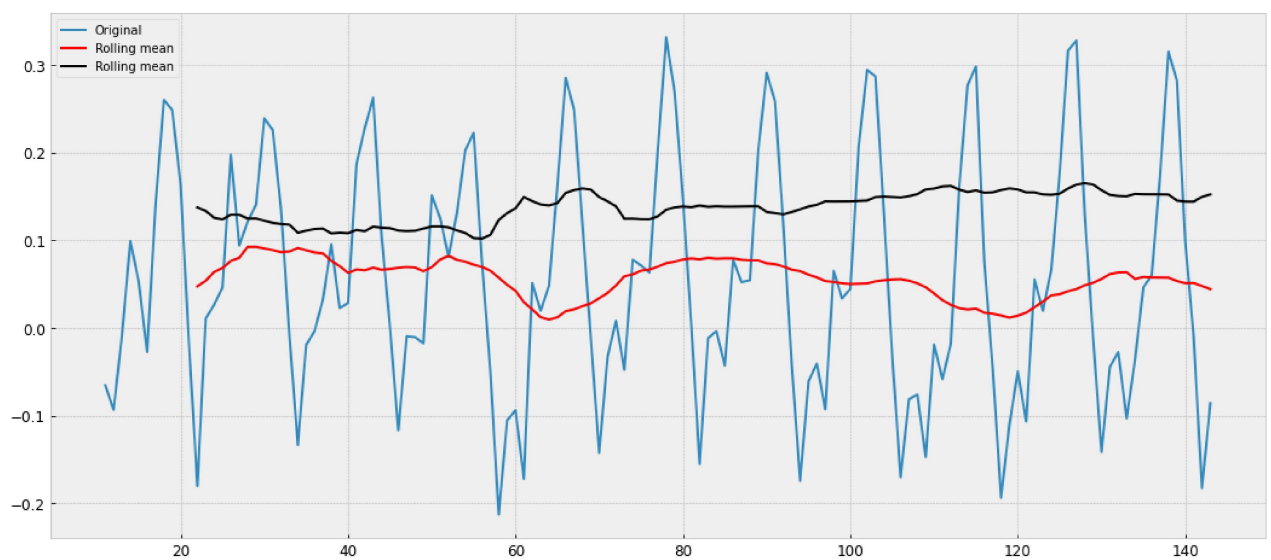
```
<matplotlib.legend.Legend at 0x136d09c3dc0>
```



In [23]:

```
#combine it with our previous log scale and apply differencing
ts_log_moving_avg_diff = ts_log - avg

ts_log_moving_avg_diff.plot(label='Original')
ts_log_moving_avg_diff.rolling(12).mean().plot(
    color='red', label="Rolling mean")
ts_log_moving_avg_diff.rolling(12).std().plot(
    color='black', label="Rolling mean")
plt.legend(loc='best')
```

Out[23]: <matplotlib.legend.Legend at 0x136d0cd1130>



This looks like a stationary time series

In [ ]: