Published in Towards Data Science

Marco Peixeiro Follow

Dec 6, 2021 · 7 min read · ✨ · Listen

Save

Defining the Moving Average Model for Time Series Forecasting in Python

Explore the moving average model and discover how we can use the ACF plot to identify the right MA(q) model for our time series

35 3

Photo by Pawel Czerwinski on Unsplash

One of the foundational models for time series forecasting is the moving average model, denoted as MA(q). This is one of the basic statistical models that is a building block of more complex models such as the ARMA, ARIMA, SARIMA and SARIMAX models. A deep understanding of the MA(q) is thus a key step prior to using more complex models to forecast intricate time series.

In this article, we first define the moving average process and explore its inner workings. We then use a dataset to apply our knowledge and use the ACF plot to determine the order of an MA(q) model.

> This article is an excerpt of my upcoming book Time Series Forecasting in Python. If you are interested in learning more about time series forecasting, using both statistical and deep learning models with applied scenarios, you can learn more here.

Prerequisites

You can grab the dataset here. Note that the data is synthetic, as we rarely observe real-life time series that can be modeled with a pure moving average process. This dataset is thus for learning purposes.

The full source code is available here.

Defining the moving average process

A moving average process, or the moving average model, states that the current value is linearly dependent on the current and past error terms. Again, the error terms are assumed to be mutually independent and normally distributed, just like white noise.

A moving average model is denoted as MA(q) where *q* is the order. The model expresses the present value as a linear combination of the mean of the series (*mu*), the present error term (*epsilon*), and past error terms (*epsilon*). The magnitude of the impact of past errors on the present value is quantified using a coefficient denoted with *theta*. Mathematically, we express a general moving average process as follows:

$$y_t = \mu + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q}$$

General equation of the MA(q) model

The order q of the moving average model determines the number of past error terms that affect the present value. For example, if it is of order one, meaning that we have a MA(1) process, then the model is expressed as follows:

$$y_t = \mu + \epsilon_t + \theta_1\epsilon_{t-1}$$

Equation of the MA(1) model

If we have a moving average process of order two, or MA(2), then we express the equation like this:

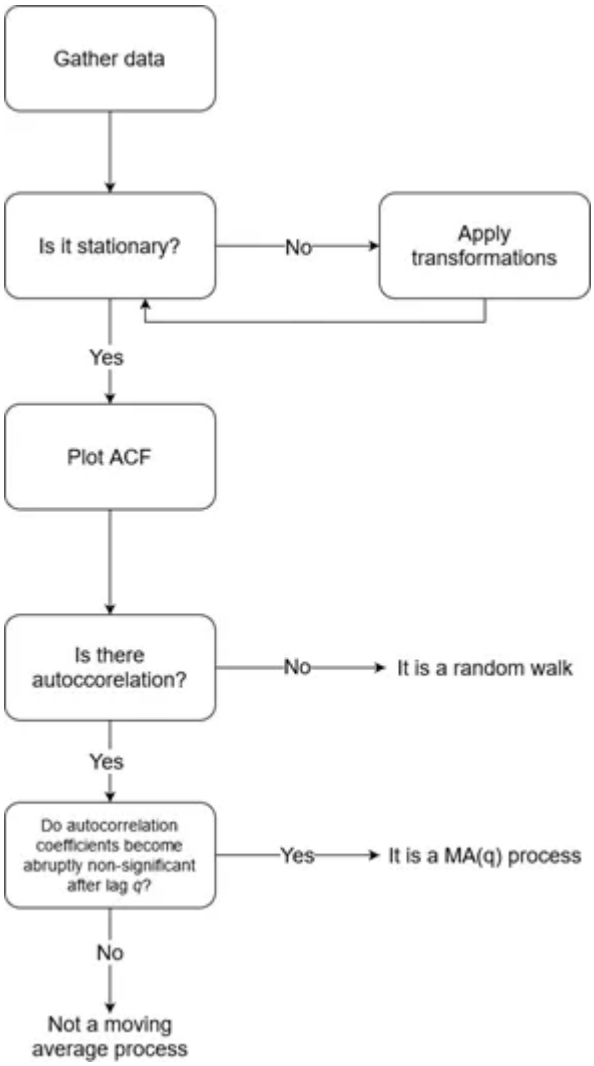
$$y_t = \mu + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2}$$

Equation of the MA(2) model

Hence, we can see how the order q of the MA(q) process affects the number of past error terms that must be included in the model. The larger q is, the more past error terms affect the present value. Therefore, it is important to determine the order of the moving average process in order to fit the appropriate model, meaning that if we have a second-order moving average process, then a second-order moving average model will be used for forecasting.

Identifying the order of a moving average process

To identify the order of a moving average process, we follow the steps outlined below:



Steps to identify an MA(q) model and its order. Image by the author.

As usual, the first step is to gather the data. Then, we test for stationarity. In the event where our series is not stationary, we apply transformations, such as differencing, until the series is stationary. Then, we plot the ACF and look for significant autocorrelation coefficients. In the case of a random walk, we will not see significant coefficients after lag 0. On the other hand, if we see significant coefficients, then we must check if they become abruptly non-significant after some lag q. If that is the case, then we know that we have a moving average process of order q. Otherwise, we must follow a different set of steps to discover the underlying process of our time series.

Let's put this in action using our data for the volume of sales of widgets for the XYZ Widget Company. The dataset contains 500 days of sales volume data starting on January 1, 2019. We will follow the set of steps outlined in figure 4.3 and determine the order of the underlying moving average process.

The first step is to gather the data. While this step was already done for you, this is a great time to load the data into a DataFrame using *pandas* and display the first five

rows of data:

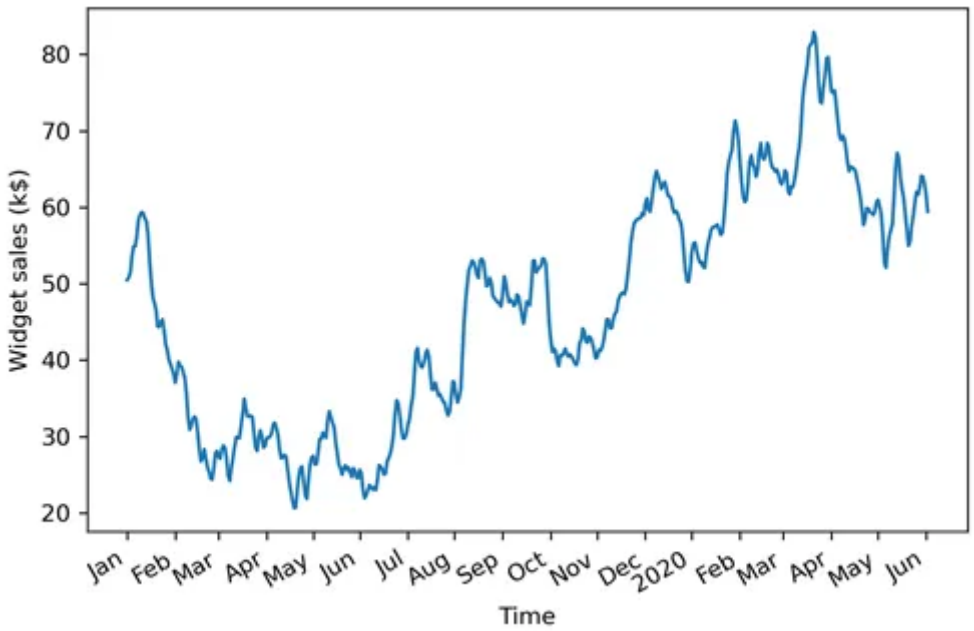
```
import pandas as p
df = pd.read_csv('data/widget_sales.csv')
df.head()
```

You see that our volume of sales is in the column *widget_sales*. Note that the volume of sales is in units of thousands of US dollars.

We can then plot the data using the following piece of code:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(df.widget_sales)
ax.set_xlabel('Time')
ax.set_ylabel('Widget sales (k$)')

plt.xticks(
[0, 30, 57, 87, 116, 145, 175, 204, 234, 264, 293, 323, 352, 382,
409, 439, 468, 498],
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
'Oct', 'Nov', 'Dec', '2020', 'Feb', 'Mar', 'Apr', 'May', 'Jun'])
#D
fig.autofmt_xdate()
plt.tight_layout()
```



Volume of widget sales for the XYZ Widget Company over 500 days, starting in January 1, 2019. Image by the author.

The next step is to test for stationarity. We intuitively know that the series is not stationary since there is an observable trend as see in the figure above. Still, we will use the ADF test to make sure. Again, we use the adfuller function from the statsmodels library and extract the ADF statistic and p-value. If the ADF statistic is a large negative number and the p-value is smaller than 0.05, then our series is stationary. Otherwise, we must apply transformations.

```
from statsmodels.tsa.stattools import adfuller
ADF_result = adfuller(df.widget_sales)
print(f'ADF Statistic: {ADF_result[0]}')
print(f'p-value: {ADF_result[1]}')
```

This results in an ADF statistic of -1.51 and a p-value of 0.53. Here, the ADF statistic is not a large negative number and the p-value is greater than 0.05. Therefore, our time series is not stationary and we must apply transformations to make it stationary.

In order to make our series stationary, we will try to stabilize the trend by applying a first-order differencing. We can do so by using the *diff* method from the *numpy* library. Remember that this method takes in a parameter *n* that specifies the order

of differencing. In this case, because it is a first-order differencing, *n* will be equal to 1.

```
import numpy as np
widget_sales_diff = np.diff(df.widget_sales, n=1)
```

With a transformation applied to our series, we can test for stationarity again using the ADF test. This time, make sure to run the test on the differenced data stored in the *widget_sales_diff* variable.

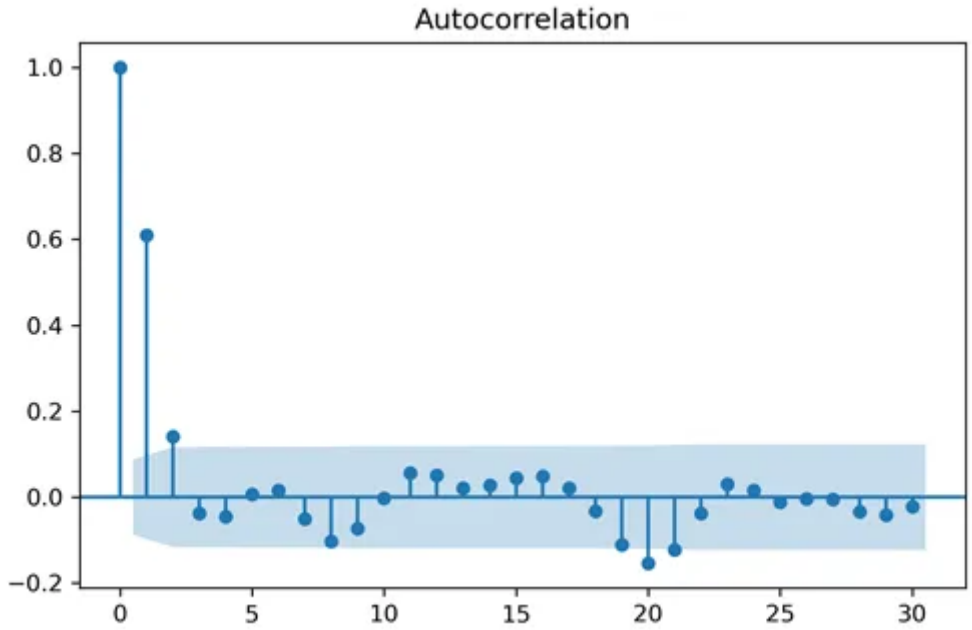
```
ADF_result = adfuller(widget_sales_diff)
print(f'ADF Statistic: {ADF_result[0]}')
print(f'p-value: {ADF_result[1]}')
```

This gives an ADF statistic of -10.6 and a p-value of virtually 0. Therefore, with a large negative ADF statistic and a p-value much smaller than 0.05, we can say that our series is stationary.

Our next step is to plot the autocorrelation function. The *statsmodels* library conveniently includes the *plot_acf* function for us. We simply pass in our differenced series and specify the number of lags in the lags parameter. Remember that the number of lags determines the range of values on the x-axis.

```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(widget_sales_diff, lags=30);
plt.tight_layout()
```

The resulting ACF plot is shown below. We notice that there are significant coefficients after lag 0. In fact, they are significant up until lag 2. Then, they abruptly become non-significant as they remain in the shaded area of the plot. We can see some significance around lag 20, but this is likely due to chance, as the following coefficients are not significant.



ACF plot of the differenced series. Notice how coefficients are significant up until lag 2 before falling abruptly in the non-significance zone (shaded area) of the plot. We see some significant coefficients around lag 20, but this is likely due to chance since they are non-significant between lag 3 and 20 and after lag 20. Image by the author.

Since we have significant autocorrelation coefficients up until lag 2, this means that we have a stationary moving average process of order 2. Therefore, we can use a second-order moving average model, or MA(2) model, to forecast our stationary time series.

Thus, we can see how the ACF plot helps us determine the order of a moving average process. The ACF plot will show significant autocorrelation coefficients up until lag q, after which all coefficients will be non-significant. We can then conclude that we have a moving average process of order q or MA(q) process. In our case, working with the volume of widget sales, we discovered that the stationary process is a second-order moving average process since the ACF plot showed significant coefficients up until lag 2.

Conclusion

In this article, we defined the moving average process and experienced how the ACF plot can be used to find the right order of the MA(q) model. This model can be used to forecast the time series.

I hope you enjoyed the read!

Cheers 🍺!

Source: Time Series Forecasting in Python

Time Series Analysis Python Machine Learning Artificial Intelligence

Data Science