# Guide to Time Series Analysis with Python — 2: Moving Average Process

[[Buse Köseoğlu](https://buse-koseoglu13.medium.com/?source=post_page-----784328325e5f--------------------------------)](https://buse-koseoglu13.medium.com/?source=post_page-----784328325e5f--------------------------------)

[Buse Köseoğlu](https://buse-koseoglu13.medium.com/?source=post_page-----784328325e5f--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F5e1a64153870&operation=register&redirect=https%3A%2F%2Fbuse-koseoglu13.medium.com%2Fguide-to-time-series-analysis-with-python-2-moving-average-process-784328325e5f&user=Buse+K%C3%B6seo%C4%9Flu&userId=5e1a64153870&source=post_page-5e1a64153870----784328325e5f---------------------post_header-----------)

5 min read

·

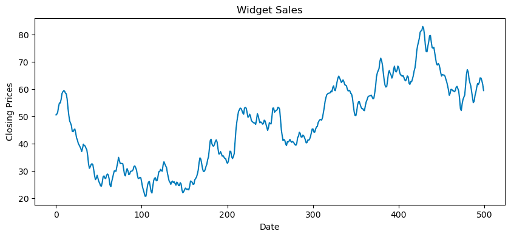
Jul 15

105

In my previous article, I talked about how time series analysis should be done, analysis methods and baseline model creation. If you do not know about these topics, you can first review this article.

1. [Guide to Time Series Analysis with Python — 1: Analysis Techniques and Baseline Model](https://medium.com/@buse-koseoglu13/guide-to-time-series-analysis-with-python-1-analysis-techniques-and-baseline-model-59e12fdd2eba)

In this article, we will examine the Moving Average Process, a model used in time series together. We will use widget sales data during these reviews. You can find all analysis codes on [github repo](https://github.com/busekoseoglu/time-series-forecasting).



# Moving Average Process (MA)

Moving average process indicates that the current value depends on the current and past error rates. If we examine the formula:



• μ: mean of the series  
• εt: Current error value  
• εt-1: Historical error value  
• Theta-q : The effect of past mistakes on the present

The important parameter we need to determine for MA is q. This parameter determines the number of historical error terms that affect the current value.

MA is used to model univariate time series. So if we’re going to use MA we only have the date and the data we want to predict, we can’t use any extra arguments.

# Specifying the q Parameter

To determine this parameter, we should use the ACF chart that we mentioned in the first article. In an ACF graph, we normally expect coefficients to be non-significant, but often this is not the case. So what should we do if there is a non-significant situation? We have a roadmap to determine this.

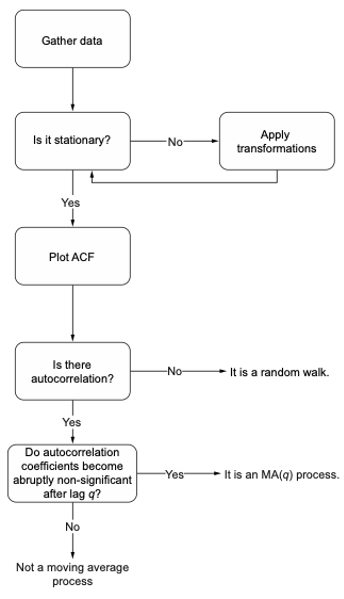
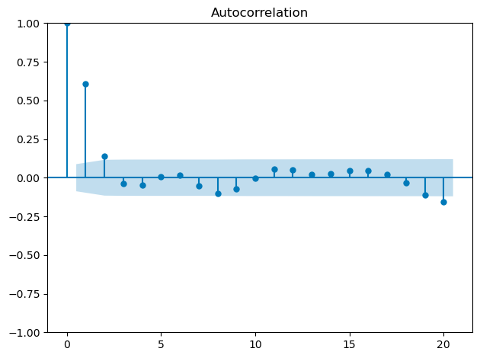


Figure from Time series forecasting in python — Marco Peixerio

The map above shows us a scenario that can develop after plotting the ACF graph. Accordingly, if there is autocorrelation, that is, if all coefficients are not non-significant, we need to find out after which lag the coefficients are non-significant. Let’s explain with an example.

# We made stationary series  
widget\_sales\_diff = np.diff(df['widget\_sales'], n=1)  
plot\_acf(widget\_sales\_diff, lags=20);  
  
plt.tight\_layout()



The above ACF graph was drawn after the data was stationary. We can determine the MA(q) value according to this graph. In our previous review, all coefficients were non-significant (i.e. they were all in the blue confidence interval). But now, after the second lag, the coefficients have entered the confidence interval. So we can accept the parameter q as 2. There is only one significant coefficient seen in Lag 20. We can ignore this as it has no continuity and has never been seen before 20.

# Modelling

There is an important point that we should not forget when we are going to model for MA(q). The MA model does not allow us to predict 50 days ahead. The MA(q) model can only predict the future q steps. Therefore, if we want to predict the far future, we need to add the predicted q value to the data and then re-estimate it.

An important point here is that we should do the modeling on the data that we have made stationary. In other words, we will use the widget\_sales\_diff data above while modeling.

df\_diff = pd.DataFrame({'widget\_sales\_diff': widget\_sales\_diff})  
  
train = df\_diff[:449]  
test = df\_diff[449:]

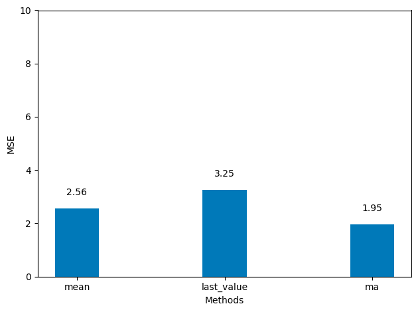
We create a dataframe with the data we have made stationary and divide it into train and test.

With the following function, we will create both the MA model and our base models for comparison. This function will be used for both moving average and autoregressive processes.

def rolling\_forecast(df: pd.DataFrame, train\_len: int, horizon: int, window: int, method: str) -> list:  
   
 total\_len = train\_len + horizon  
   
 if method == 'mean':  
 pred\_mean = []  
   
 for i in range(train\_len, total\_len, window):  
 mean = np.mean(df[:i].values)  
 pred\_mean.extend(mean for \_ in range(window))  
  
 return pred\_mean  
  
 elif method == 'last':  
 pred\_last\_value = []  
   
 for i in range(train\_len, total\_len, window):  
 last\_value = df[:i].iloc[-1].values[0]  
 pred\_last\_value.extend(last\_value for \_ in range(window))  
   
 return pred\_last\_value  
   
 elif method == 'MA':  
 pred\_MA = []  
   
 for i in range(train\_len, total\_len, window):  
 model = SARIMAX(df[:i], order=(0,0,2))  
 res = model.fit(disp=False)  
 predictions = res.get\_prediction(0, i + window - 1)  
 oos\_pred = predictions.predicted\_mean.iloc[-window:]  
 pred\_MA.extend(oos\_pred)  
   
 return pred\_MA

**“model = SARIMAX(df[:i], order=(0,0,2))”** is the part where we create the model and specify the q parameter we found. After doing all this, only the training process remains.

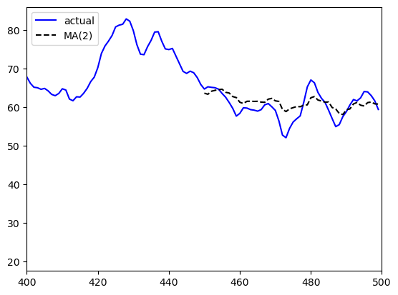
Above, we created both our base models and our MA(2) model. We can now compare these three models.



When we examine the model comparisons, we see that the best model is the MA(2) model, as we expected. But to get the real metrics, we have to reverse the difference we applied.

df['pred\_widget\_sales'] = pd.Series()  
df['pred\_widget\_sales'][450:] = df['widget\_sales'].iloc[450] + pred\_df['pred\_MA'].cumsum()

When we reverse this process, we can now obtain the actual sales values predicted by the model. You can see the actual values and the estimation of the model in the chart below.



Thank you for reading. In the next article, we will make predictions using Autoregressive Process. Stay tuned!

# References

* Time series forecasting in python — Marco Peixerio
* Tan,Y.-F.;Ong,L.-Y.;Leow, M.-C.; Goh, Y.-X. Exploring Time-Series Forecasting Models for Dynamic Pricing in Digital Signage Advertising. Future Internet 2021, 13, 241. <https://doi.org/10.3390/> fi13100241
* Machine learning with time series — Markus Löning
* <https://www.influxdata.com/time-series-forecasting-methods/>
* <https://builtin.com/data-science/time-series-forecasting-python>
* <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
* <https://www.projectpro.io/article/moving-average-time-series-model/716>