# Abstract

Nowadays, the demand for data analysis and forecasting has been grown higher and higher. However, with that increase in demand, there is also a surge in misunderstandings. Have you heard about one of the following statements? Or do you even agree with one?

* Forecasting is only for data scientists.
* You cannot do data science without knowing Python or R.
* I need to talk to IT for everything related to Machine Learning.
* Machine Learning or Neural Network is the answer to everything.
* The more complicated the model is, the more accurate it is.
* Etc.

In this article, we are going to see how an analyst with Statistics 101 knowledge can do time series analysis in **Power BI** **without Python or R**. You do not have to know Python or R; you do not need to talk to IT department; you do not need to install anything extra; and you do not need to sweat about it.

In this model, we are going to use the **Decomposition Method** for the time series forecasting. If you are not familiar with the term, no worries, it will be covered later.

Without further ado, let’s jump right in.

# Prerequisite

1. You will need to have the Power BI Desktop client installed. If you do not have Power BI, you can download it from here.
2. Because of the properties of the **Decomposition Model**, it is highly recommended to have a data input with strong seasonality. If you do not have the data by hand, you can use the fictional data I generated from here.
3. The Dashboard needs to be in Import Mode. Otherwise, you won’t be able to create calculated columns and measures.

# Decomposition Method

According to Wikipedia, Decomposition of time series is defined as the following:

The **decomposition of time series** is a statistical task that deconstructs a time series into several components, each representing one of the underlying categories of patters.

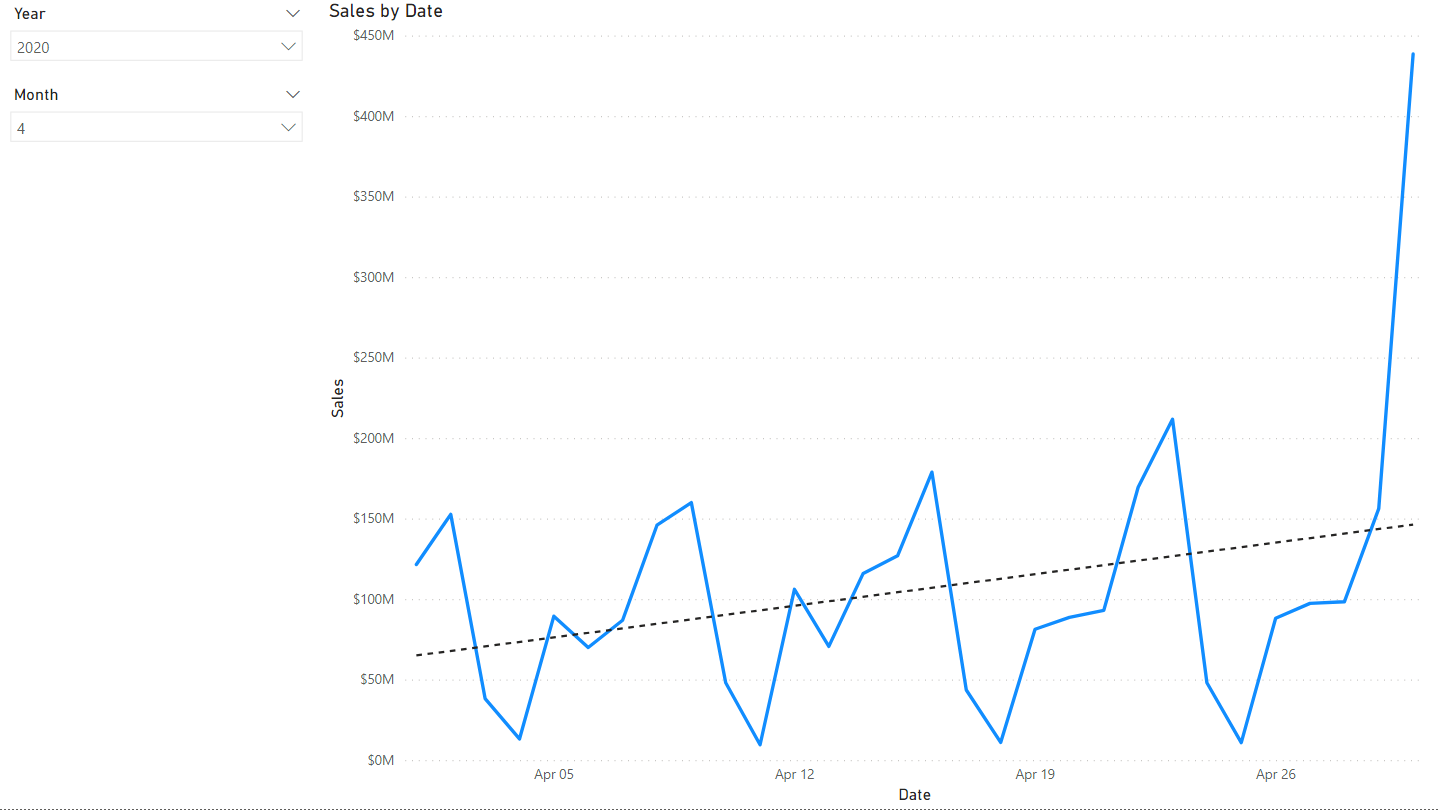
In other words, the **Decomposition Method** is striping out all the possible date related elements before building the model. After the model is created, we then add those elements back into the model.

For instance, if I know my sales data will have a surge every weekend, I will “remove” the rise for every weekend before we build our model. This way, I will be able to build my model with the simplest model possible without the consideration of the “anomalies.” Once the model is completed, I can “add” the surge back for every weekend for a more accurate forecast.

However, keep in mind, I’m using the term **Decomposition Method** instead of **Decomposition Model**. There is a difference between these two terms. The **Decomposition Method** is a way of thinking, where the **Decomposition Model** is a defined statistical model which is also used in *fbprophet* (a Facebook developed time series forecasting tool). However, conceptually, these two are very similar about striping elements away.

# Data Discovery

Before we jump right into the model building, we always want to do a discovery session on what data we have. With Power BI, everything can be drag and drop. Once we imported the data into Power BI, we can simply create a Line Chart with *[Date]* as *Axis* and *[Sales]* as *Values*. To make this graph more readable, we can filter the data to any time period. In our case, we screened the data to April 2020.

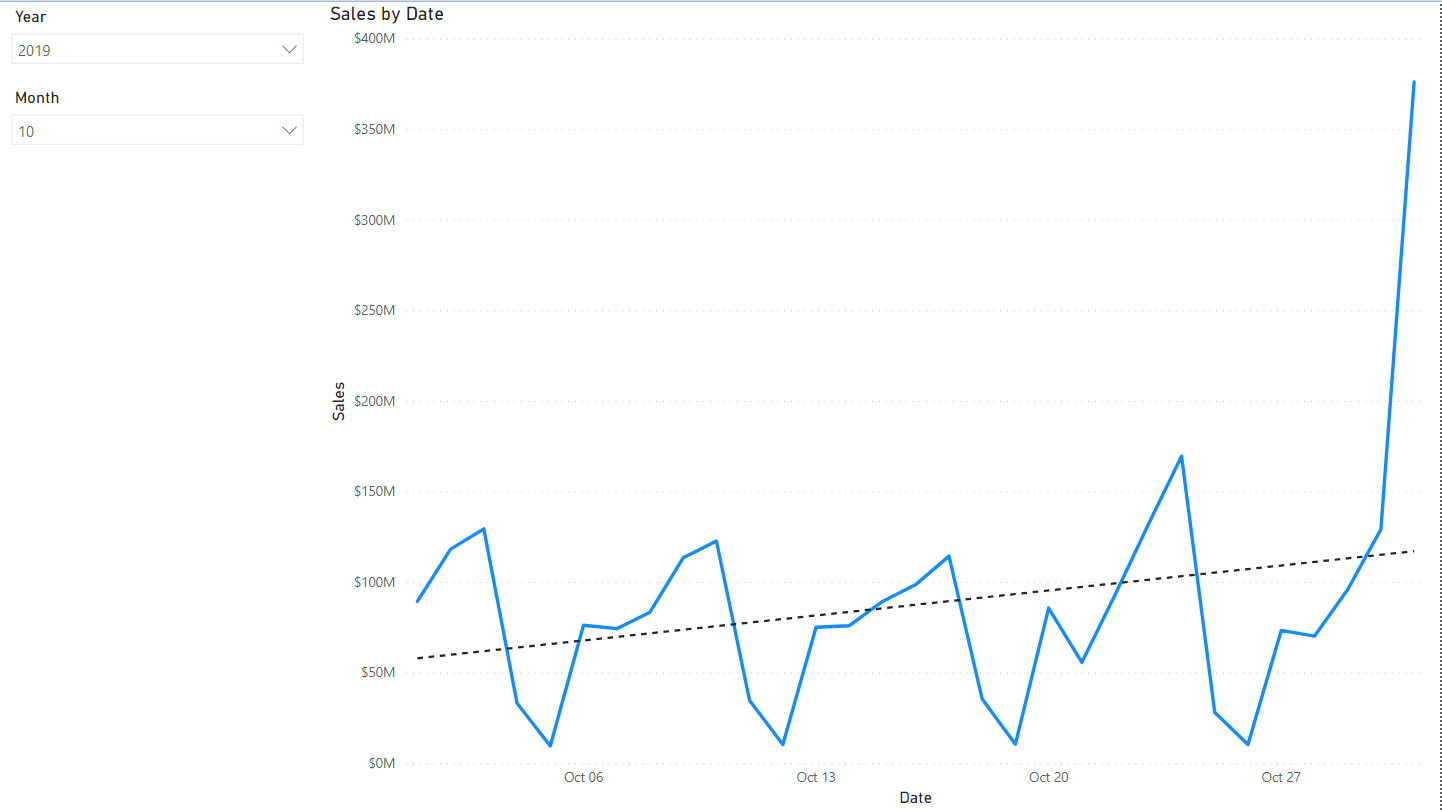


As we can see, there are a couple of features are exhibited in the data:

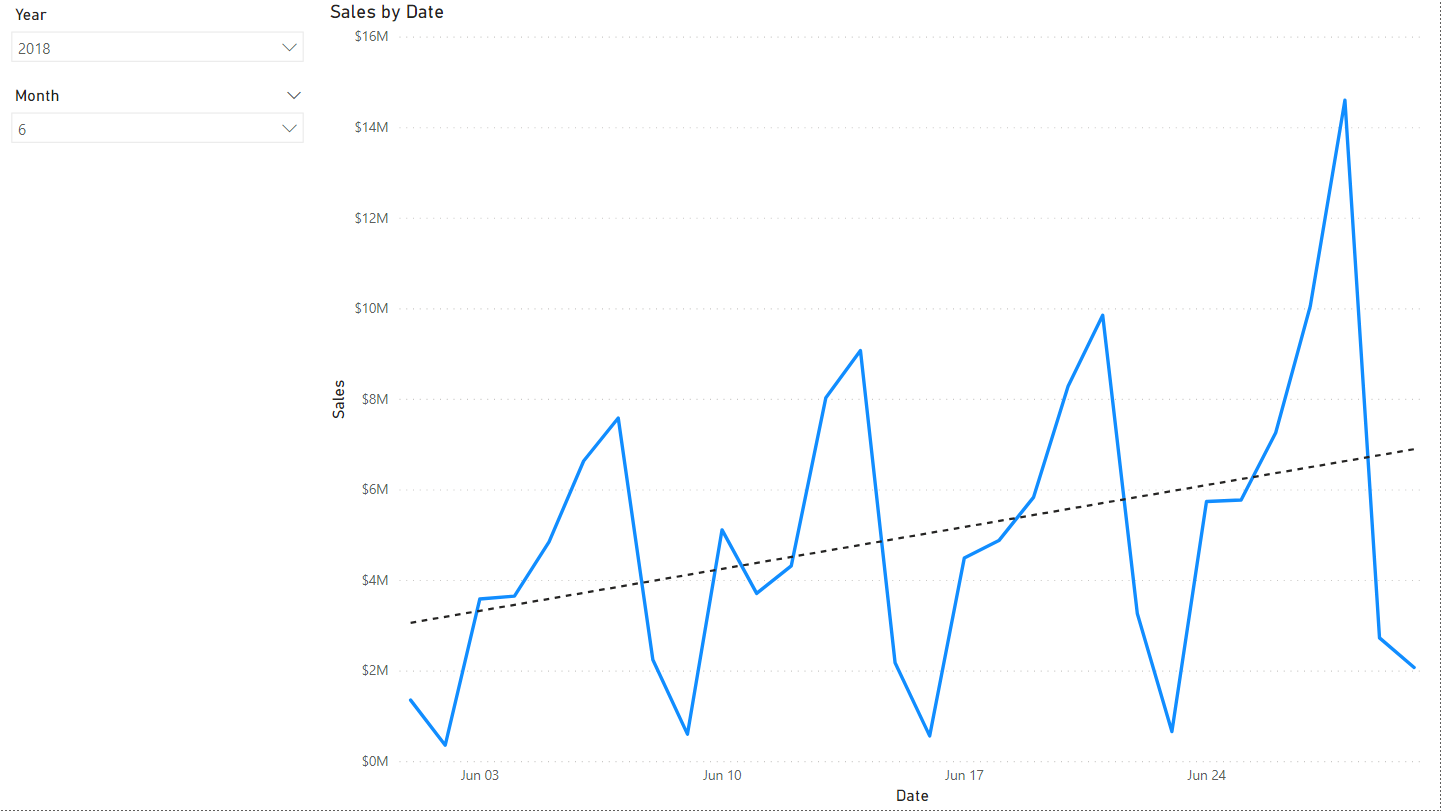
1. There is an upward trend for sales over time (dashed line);
2. There is a weekly pattern for sales;
3. And there is a spike at the month-end.

To confirm these hypothesis/observations, let’s take a look at some other months:

For October 2019:



For June 2018:



As we can see, those three observations are generally correct. However, the month-spike for June 2018 was not on the last day of that month. This is because I had another logic implemented when I was creating this dataset, which is the stores do not open On Sundays. If the month-end happens to be on a Sunday, then the month-end day will be the day prior – Saturday.

# Model

With all those steps, we are finally at the model building step. Just like building a house, we need to have a blueprint/plan. In our case, we know we are going to use the Decomposition Method. And we know we have three different elements: 1) Month-End, 2) Weekly Cycle, and 3) Upward Trend. With these being said, here is our pseudo math model:

In this model:

1. is the Weekly Cycle amount adjustment (value or modifier) based on time,
2. is the Month-End amount adjustment (value or modifier) based on time,
3. is the Upward Trend of Sales over time,
4. And is the error term.

In this scenario, we are going to use the Multiplicative Decomposition Method because it requires fewer calculations for the data we have.

Now it is time for us to strip the elements from the data.

## Decomposing the Elements

The first step we do is to create 3 calculated columns for the three elements:

1. Weekly Cycle

This column is used to identify the weekday of every date. This column will also be used by the Month-End column.

//Code

1. Month-End

This column is used to identify which day is the last day of every month. But remember, we had a business rule that the store would not open on Sundays.

//Code

Because in DAX Calculated Column, we cannot refer to another row without using Table Functions, which are very resource-intensive. Instead of searching Month-End Sundays then update the previous day, we search for the Saturdays with the Month-End day as the next day and mark that Saturday as the Month-End.

1. Upward Trend

To make the regression calculation less resource-intensive, we just put an index over the *Date* column. But it is not required.

//Code

I created a separate table for forecasting because I wanted to make this dashboard as easy to modify as possible in the future. However, you can use the single ‘*Fact* Sales’ for this demo. Since we are going to need the same columns for forecasting as well, we also need to create the same columns in the forecasting table. If you decide to use the index column, remember to generate the Index from the same starting date.

## Decomposing the Elements – Month End

The Striping process for both the Month-End and Weekly Cycle is the same. We want to identify the Sales Amount modifier applied to Month-End/Weekly Cycle. The easiest way of doing that would be using the average of the sales amount on every Month-End/Weekday divided by the overall average amount.

ATTENTION: Keep in mind, because the Month-End modifier and Weekly Cycle modifier are affecting each other, the order of the decomposition will affect the forecast accuracy. It is always a good idea to try different orders.

For the sake of demonstration, we are going to decompose the Month-End modifier using this method first. To put this step into pseudo-code:

With this pseudo-code, we can compose the following measure:

//Code

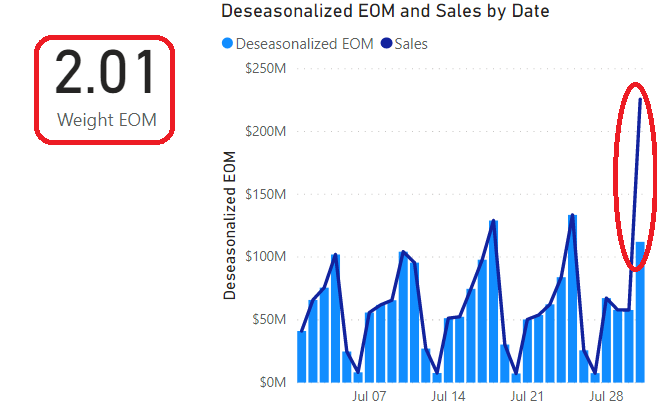
ATTENTION: Because the modifiers are aggregated scalar values, we need to create the modifiers as measures. Also, because we want this modifier to adapt over time, we used *ALL(‘Fact Sales’)*. If there is a historical data usage requirement, you can either add a dynamic filter in the calculation or filter out the data from import.

Because we used the *ALL* function on the *‘Fact Sales’* dataset, no matter how we filter the dataset, the measure will be static.

Now we have the Month-End Modifier. We can start to remove the modifier on the Month-End amounts. We will be doing this by creating a new column:

//Code

With that being done, let’s take a look at how the data looks now for July 2019:

Add alt text

As we can see here, we had a Month-End modifier of 2.01. And after the decomposition, the amount for the last day of the month has been reduced by half.

Now, because we don’t want to double count the modifiers, we will use the result from Month-End decomposition to do the Weekly Cycle Modifiers.

## Decomposing the Elements – Weekly Cycle

For the Weekly Cycle Modifiers, we will apply the same logic as Month-End Modifier. There will be only two differences: 1) We will be using the [*Deseasonlized EOM*] instead of [*Sales*] for the calculation, and 2) we will need to create a modifier for each weekday to mimic the cycle.

For instance, the pseudo-code for Monday will be:

And the actual code for Mondays would be:

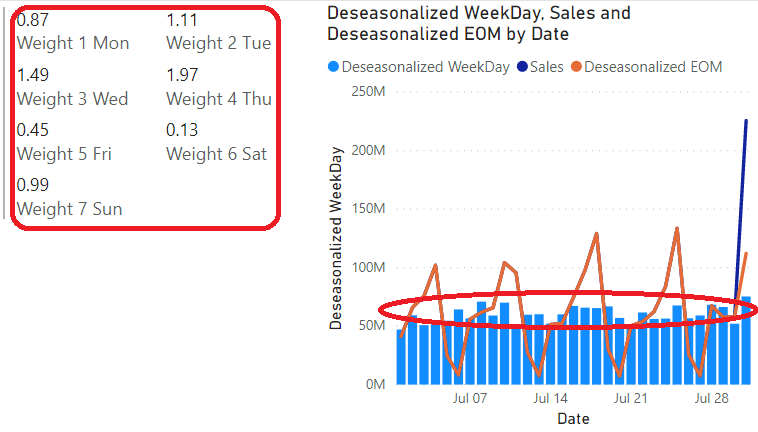
//Code

As you can see, it is very similar to the Month-End Modifier.

After we create a Modifier measure for each weekday, we can then apply them to [*Deseasonalized EOM*] the same way as how we applied the Month-End Modifier to [*Sales*] by creating a new column:

//Code

ATTENTION: The *SWITCH* function here is just a personal preference. The core concept here is to divide the [*Sales*] for each workday by the corresponding modifier. It is okay to use a line of *IF* statements. Using the *SWITCH* statement is just to make the debugging easier down the line.

Now let’s take a look at the result after we take away the weekday modifier: 

As we can see, the value [*Deseasonalized WeekDay*] is much more smooth compared to the [*Sales*]. Now we can start building a Regression Model on top to represent the trend.

## Decomposing the Elements – Trend

Before we start to write the code directly, you know the drill, we need to have a pseudo-code. For doing that, we need to have the general knowledge of the model we are about to use first. In our case, we will be using the Linear Regression Model. We choose the Linear Regression Model for two reasons: 1) at the data discovery step, we can see that the trend is or is close to linear, and 2) I used a linear model to generate the data 😊.

Since we know we are going to use the Linear Regression Model, let’s recall the Statistics 101 knowledge on Linear Regression:

Where *y* is the dependent variable (or ‘outcome variable’), *t* is the independent variable (‘predictors’, ‘covariates’, or ‘features’), *a* is the slope, and *c* is the intercept.

With the function given, all we have to do is to figure out what *a* and *c* are with the data we have ([*Deseasonalized WeekDay*] and [*Date*]). Luckily, we do have the formulas for slope and intercept prepared:

Where *t* is the [*Date*] column, *y* is the [*Deseasonalized WorkDay*], and *N* is the number of records in our data. I won’t bore you with where these formulas come from, so let’s just use them.

As we can see, we are still missing two values in these two formulas: *ty* and *tt*. So, let’s create them as **Columns**:

//Code

ATTENTION: We used calculated columns for *ty* and *tt* is because these to values need to be generated for EVERY data point. Also, remember the [*Index*] column we created earlier? Here is where it can reduce the complexity of the calculation.

Now, we have everything to assemble the *slope* as a **measure**:

//Code

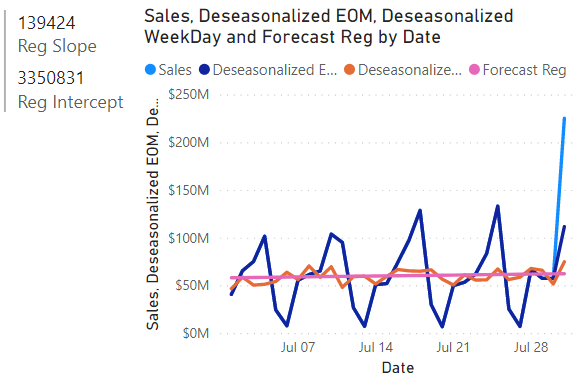
And *intercept* as a **measure**:

//Code

With *slope* and *intercept*, we can finally assemble the linear regression as a new column:

//Code

ATTENTION: Remember, I created a forecast table for forecasting. Since scoring/drawing the regression is the first step for our model, I created this column in the forecasting table. If you didn’t create a separate table, remember to change the ‘*Fact Forecast*’ in the code snippet to ‘*Fact Sales*’.



It looks somewhat like it!

# Assembling

Now, we have everything needed to do our forecast. All we have to do is to reverse the decomposition process using the [*Forecast Reg*] **column**, literally. If we stripped away the Month-End Modifier first, then we put it back last. If we stripped away the Weekly Cycle Modifiers by division, then we add them back by multiplication. It’s just a step-by-step reverse process.

For instance, when we are putting back the Weekly Cycle Modifiers to generate [*Forecast Weekday*] **column**:

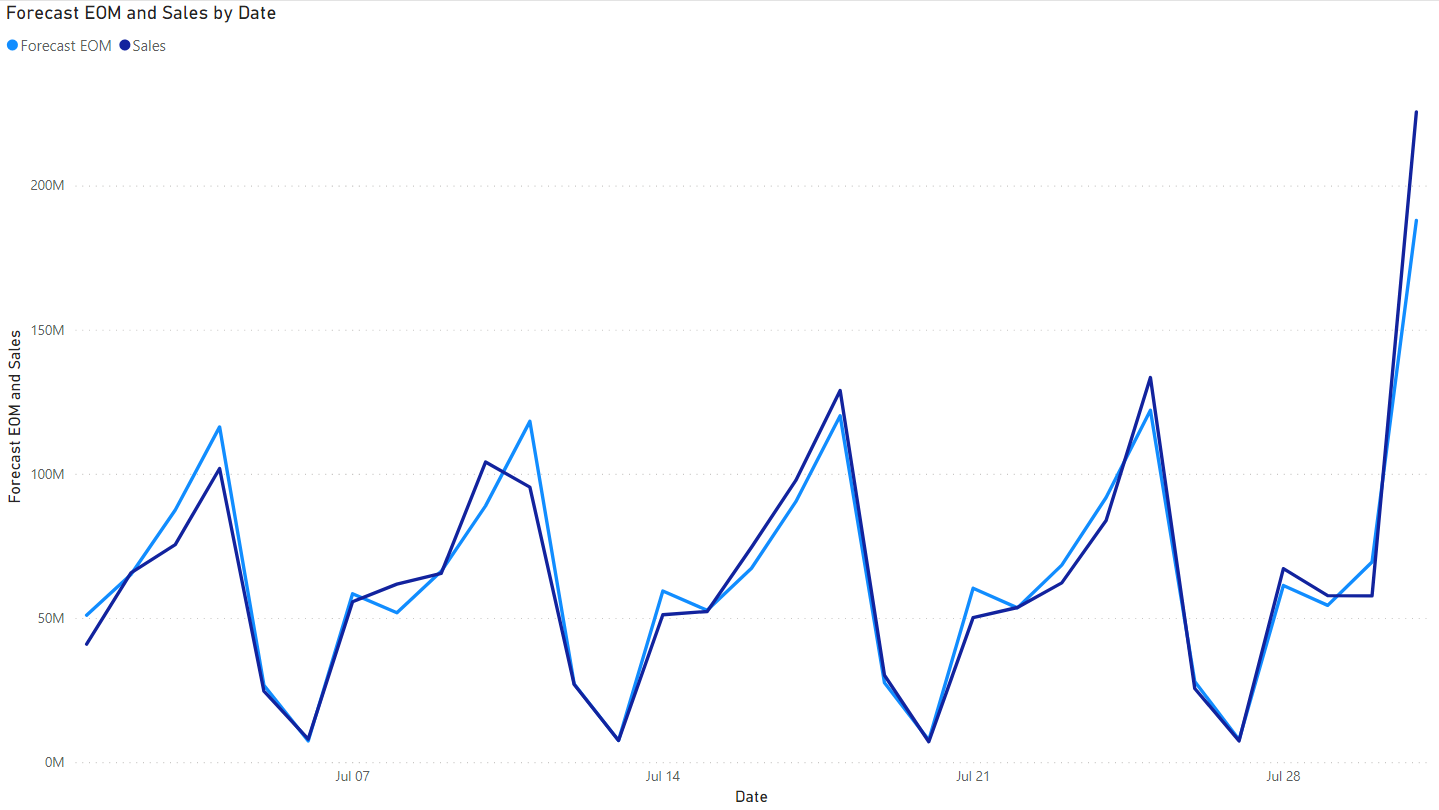
//Code

Then we put back the Month-End Modifier to generate [*Forecast EOM*] **column**:

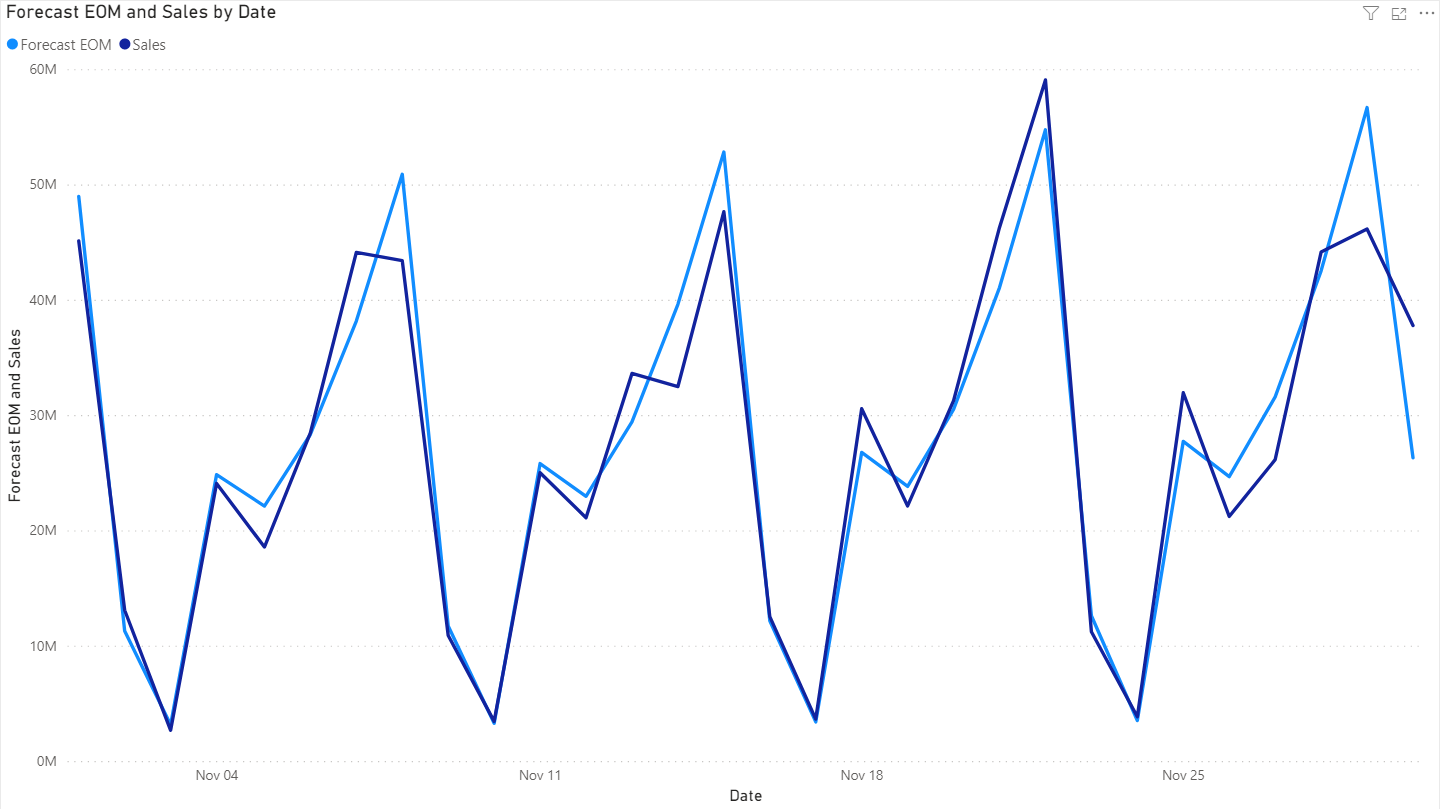
//Code

# Forecasting

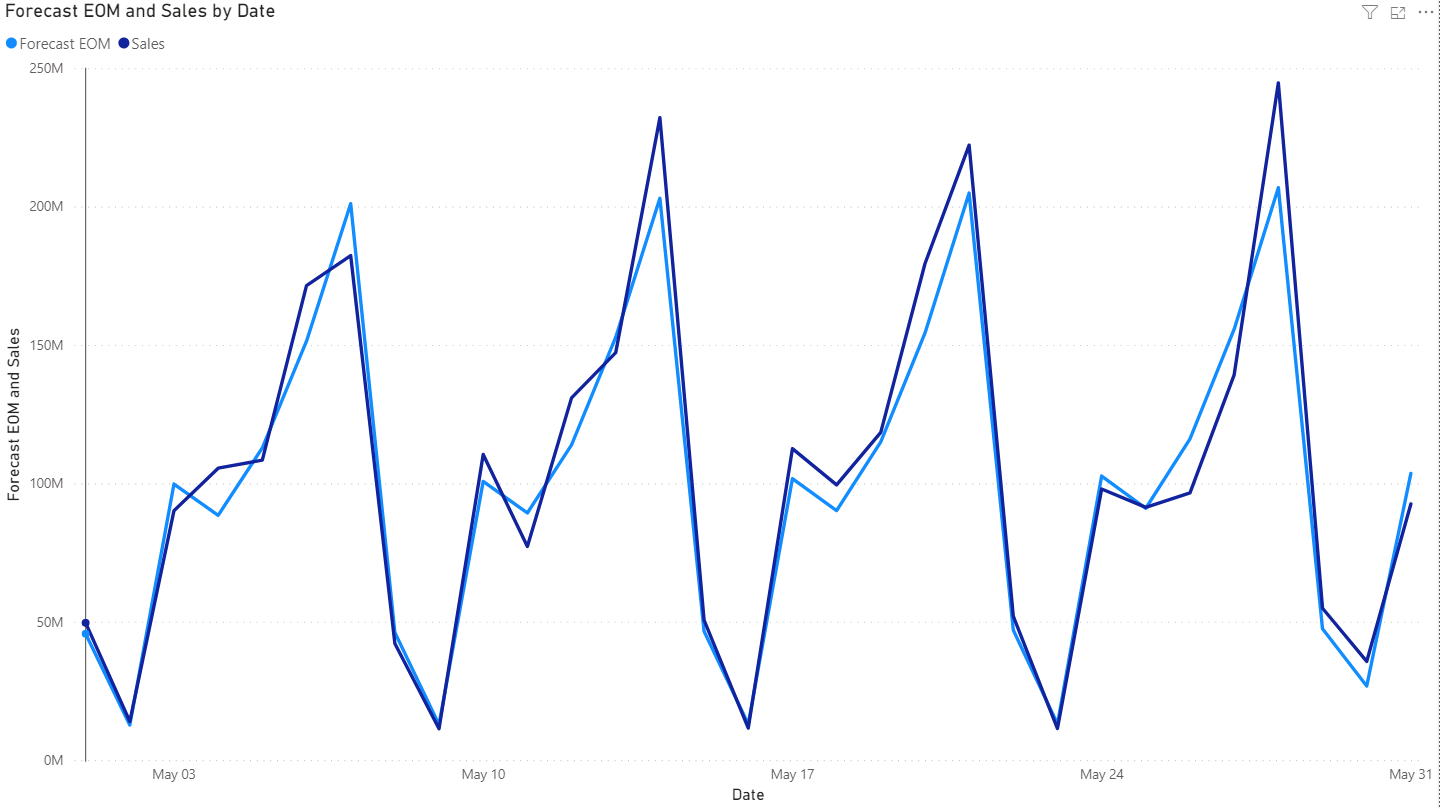
FINALLY, we have completed all the steps for making a forecasting model. Let’s take a look at how it performs against training data (July 2019):



Not half bad! How about some other months, such as November 2018:



Or a recent one, May 2020:



As we can tell, the result is impressive already. How about a little bit of forecasting? Such as June 2020?



It seems like June would be a not-so-hot month, I guess.

# Conclusion

At this point, I think we can confidently state that we can do time series analysis and forecasting in Power BI without Python and R.

Here are some personal thoughts around this exercise:

* Anyone with a solid foundation on Statistics 101 should be able to build easy statistical models in Power BI using native tools.
* For some entry-level models, it is NOT necessary to provision ML Servers or Power BI Premium capacity. It’s even not necessary to provision Enterprise/Personal Gateway just to run R or Python on-premise since they are not required at all.
* In this exercise, I could not emphasize enough that we only did an easy time series forecasting using the **Decomposition Method**. It is, by all means, not the same as the **Decomposition Model**, which is far more complex and advanced.
* With that being said, we also don’t need sophisticated models to accomplish some business objectives. For instance, this model built with the **Decomposition Method** can achieve the forecasting goal just fine.
* Machine Learning is not the solution to everything. If you can do something with simpler systems, you will save both money and effort. Simpler systems are less likely to break and more straightforward to fix if they did.
* It is fun to create stuff 😊

Thank you for reading.

Everything used in this article is accessible from here.