Time Series Analysis

Shuteng Ong, Chloe Roque, Jessica Gallardo, Jennie Pham

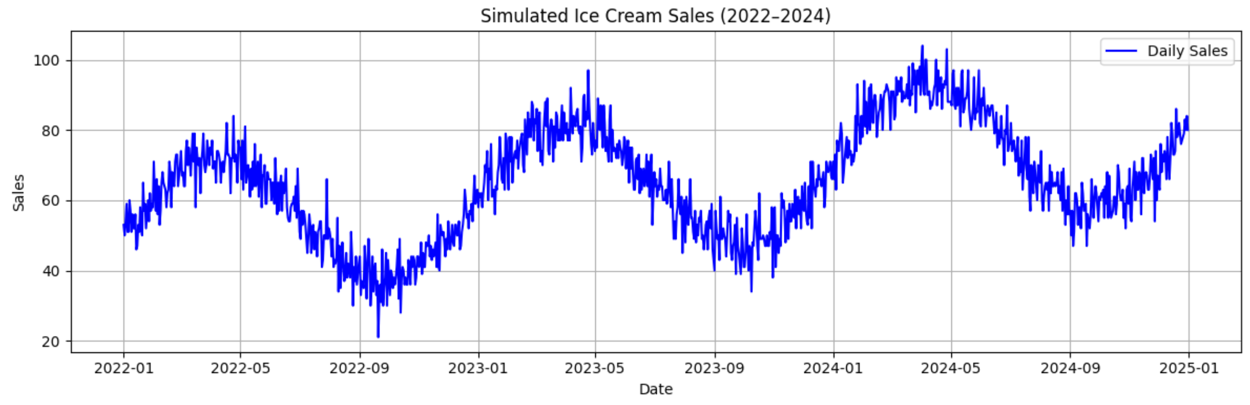
Introduction

In today's data-driven world, time plays a crucial role in how we interpret data. Whether it's monitoring stock market fluctuations, analyzing shifts in temperature, or tracking website traffic, many of the most valuable insights come from understanding how things change over time. That's where **time series analysis** comes in.

Time series analysis is the study of data points arranged in chronological order, typically recorded at regular, equally spaced intervals. These data points represent observations of a variable over time and are most commonly analyzed in a discrete-time setting, where measurements are taken at specific moments such as daily, monthly, or hourly intervals. The primary goals of time series analysis are:

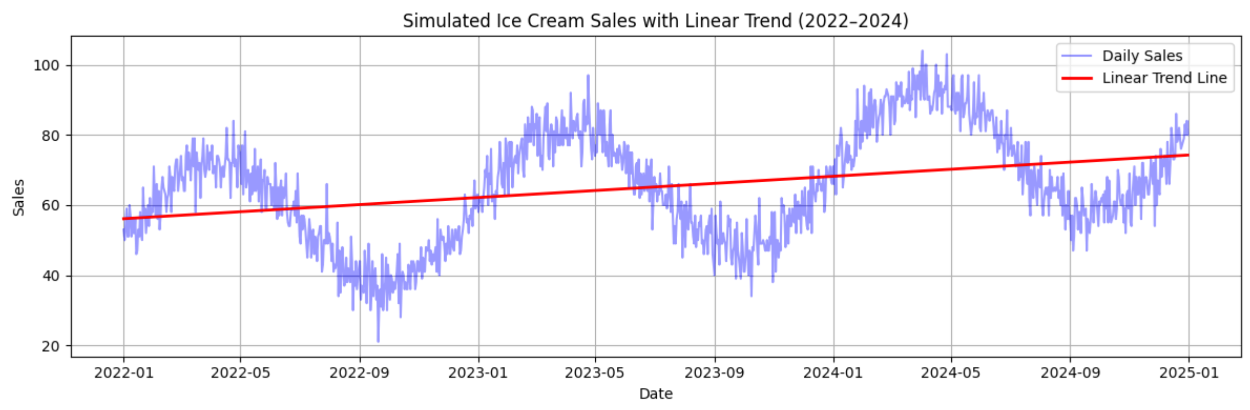
1. To understand and describe the process that generates the observed data
2. To predict how the processes will evolve in the future, using patterns found in past data

There are two main perspectives to look at a time series: a statistical perspective and a dynamical system perspective. Both approaches provide different insights and lead to distinct modeling strategies. In Python, the **statsmodels.tsa** library offers a powerful suite of tools for performing time series analysis.

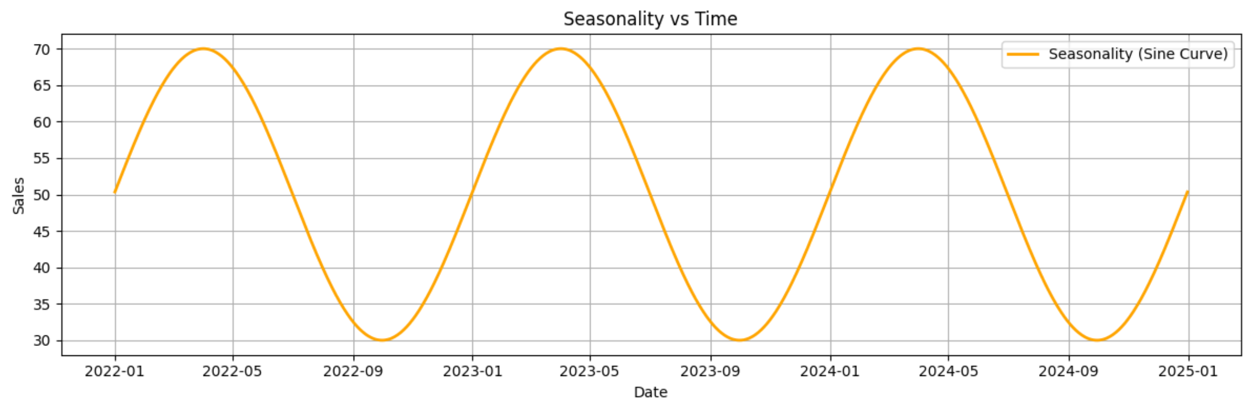


To make these ideas concrete, we'll illustrate key concepts throughout this article using a simulated dataset of daily ice cream sales from 2022 to 2024 (Figure 1). This example reveals three core components present in many real-world time series: **trend**, **seasonality**, and **residuals**.

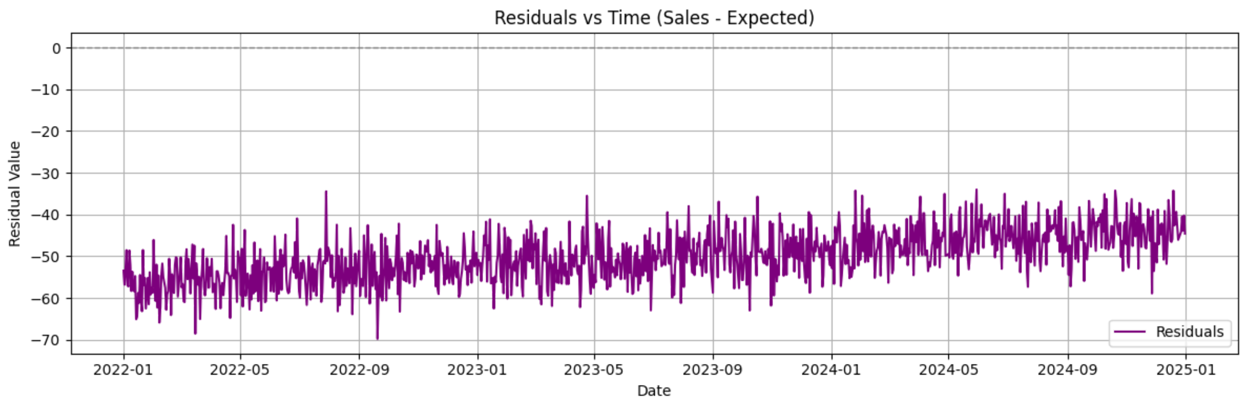
Trend refers to the long-term direction or movement in the data. In our simulated ice cream sales (Figure 2), we observe a steady upward trend across the three years, suggesting a gradual increase in overall demand. This could be due to factors such as growing popularity, expanding distribution, or even broader changes in consumer behavior.



Next is **seasonality**. **Seasonality** refers to the repeating, predictable patterns that occur at regular intervals in our data. In our simulated dataset (Figure 3), we see a clear seasonal pattern where ice cream sales in the summer months and fall in the winter months. This mirrors real-world behaviors, since people tend to buy more ice cream when it's hot outside. In fact, in our model, this seasonal effect is captured mathematically using a sine curve, which creates that smooth, wave-like pattern we see repeating each year.



Lastly, we have **residuals**. **Residuals** represent the irregular, random fluctuations that remain after removing both the trend and seasonality from the time series. They capture the unpredictable variations in data that don't follow a specific pattern. In our ice cream sales example (Figure 4), this might reflect unexpected disruptions like a storm that shut down the store for a few days, employee emergencies, or supply issues. These variations are often referred to as noise, and while they may not follow a consistent rhythm, they're important to account for.



Stationarity

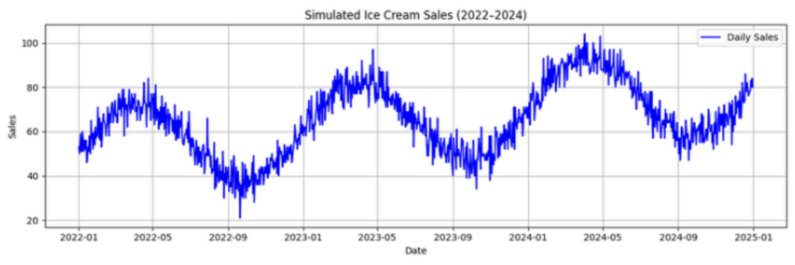
Before we can build an effective time series model, we need to talk about an important concept: **stationarity**. A time series is considered stationary if its average value (mean) if the series stays constant, the variability (standard deviation) remains constant, and there are no seasonal or repeating patterns that shift over time. In other words, the process that generates the data behaves the same way throughout the entire dataset. This matters because many forecasting models assume the data is stationary. If this assumption is violated, the model's predictions can become unreliable or misleading.

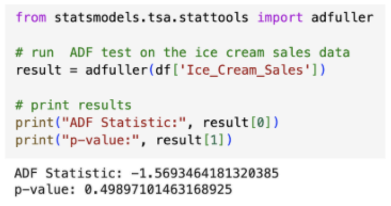
To determine whether a time series is stationary, we can start by plotting the data and examining its statistical behavior over different time windows. However, for a more rigorous assessment, we use the **Augmented Dickey-Fuller (ADF) test**, a formal statistical test that checks for the presence of a **unit root**, an indicator of nonstationarity. If the test returns a high p-value (typically greater than 0.01 or 0.05), we fail to reject the null hypothesis and conclude that the series is likely nonstationary. Fortunately, Python's statsmodels.tsa library provides a built-in ADF function, making it easy to perform this analysis and formally assess the stationarity of the data.

When a time series is found to be nonstationary, there are several techniques we can use to transform it into a stationary series. Common methods include subtracting the mean to stabilize the level, applying a log transformation to stabilize the variance, and differencing the series, which removes trends and seasonality by subtracting each data point from its previous value. These transformations help ensure that the time series meets the assumptions needed for accurate modeling and forecasting.

In our simulated ice cream sales data, we observed both an upward trend and clear seasonal spikes during the summer months (Figure 5). Unsurprisingly, the ADF test confirms that the original series is non-stationary, with a test statistic of -1.57 and a p-value of 0.499. To address this, we applied first-order differencing, a method that subtracts each data point from the previous one. This transformation removes the seasonality and stabilizes the mean. After differencing, the ADF test result improved significantly (test statistic = -6.36, p-value < 0.01), indicating that the data is now stationary and ready for modeling.

NONSTATIONARY data





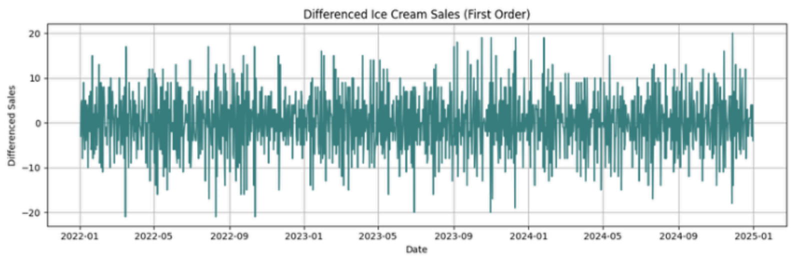
```python
from statsmodels.tsa.stattools import adfuller

# run  ADF test on the ice cream sales data
result = adfuller(df['Ice_Cream_Sales'])

# print results
print("ADF Statistic:", result[0])
print("p-value:", result[1])
```

```
ADF Statistic: -1.5693464181320385
p-value: 0.49897101463168925
```

STATIONARY data



```
ADF Statistic: -6.360430869180391
p-value: 2.482246498091736e-08
```

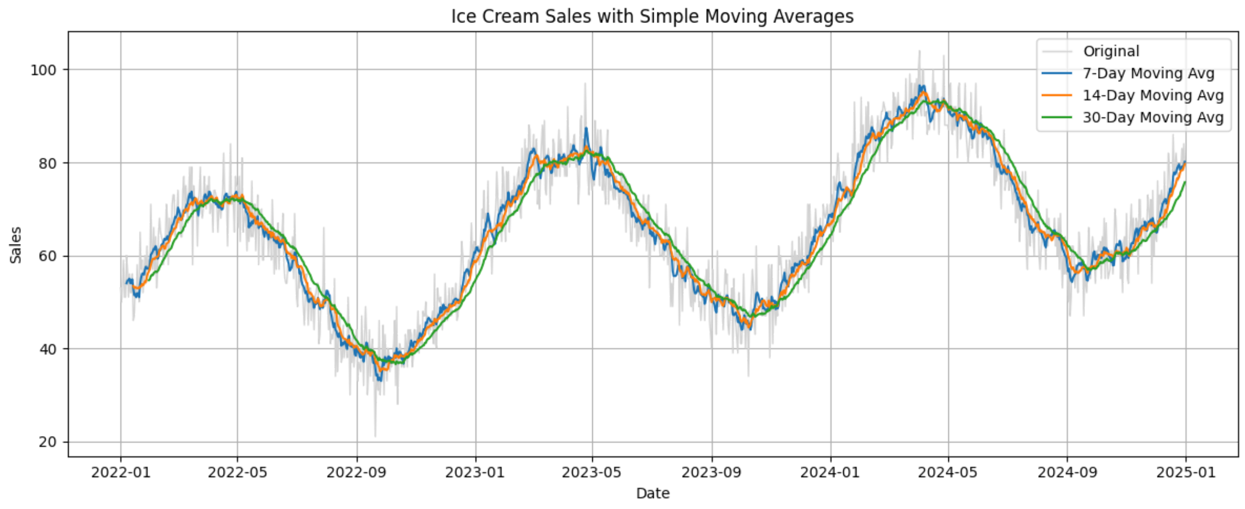
Smoothing

Now that we've addressed stationarity, the next concept we'll explore is **smoothing**. Smoothing is a technique used to reduce noise and irregular functions in a time series, helping us better visualize the underlying trend or pattern in the data. While random variation is a natural part of most time series, too much of it can make it hard to identify meaningful insights. Smoothing is not only useful for visualization but also plays an important role in forecasting, where the recovered patterns are projected into the future to predict upcoming values. There are two common types of simple smoothing: **simple smoothing** and **exponential smoothing**.

One of the most basic forms of simple smoothing is the **simple average**, where we calculate the average of all past observations and use it as a constant prediction. While easy to compute the simple average often struggles with dynamic data because it doesn't adapt to local changes. A more responsive method is the **moving average (MA)**. A moving average smooths a time series by averaging values over a sliding window of fixed size. However, the moving average also has its limitations: it assigns equal weight to all observations within the window, meaning it treats older and newer data with equal importance. Additionally, choosing the right window size is critical. A window too small leaves too much noise, while too large a window can overly flatten meaningful patterns. Another drawback is that moving averages do not automatically adjust for changes in trend or seasonality.

To address the issue of equal weighting, we can use a **weighted moving average (WMA)**. Like a standard moving average, WMA smooths the series over a window, but it assigns greater importance to more recent observations, allowing the smoothed curve to respond more effectively to recent changes while still filtering out noise from older data points. This weighted WMA captures shifts in the underlying pattern more accurately than a simple moving average.

To illustrate how simple smoothing works, we applied simple smoothing moving averages to our simulated ice cream sales data (Figure 6) using three different window sizes: weekly (7 days), biweekly (14 days), and monthly (30 days). The original sales data (gray line) is noisy with lots of daily fluctuations that make it hard to see the underlying pattern. The smoothed lines show how moving averages clarify the bigger picture: the weekly average (blue) reacts quickly to changes, the biweekly (orange) smooths the data more moderately, and the monthly (green) heavily smooths out noise but lags behind recent shifts. This example highlights the trade-off between responsiveness and stability when selecting a window size. While effective at revealing trends and seasonality, simple moving averages assign equal weight to all values in the window and do not adjust for trend or seasonal dynamics. To address this limitation, we can turn to methods like weighted moving averages and exponential smoothing.



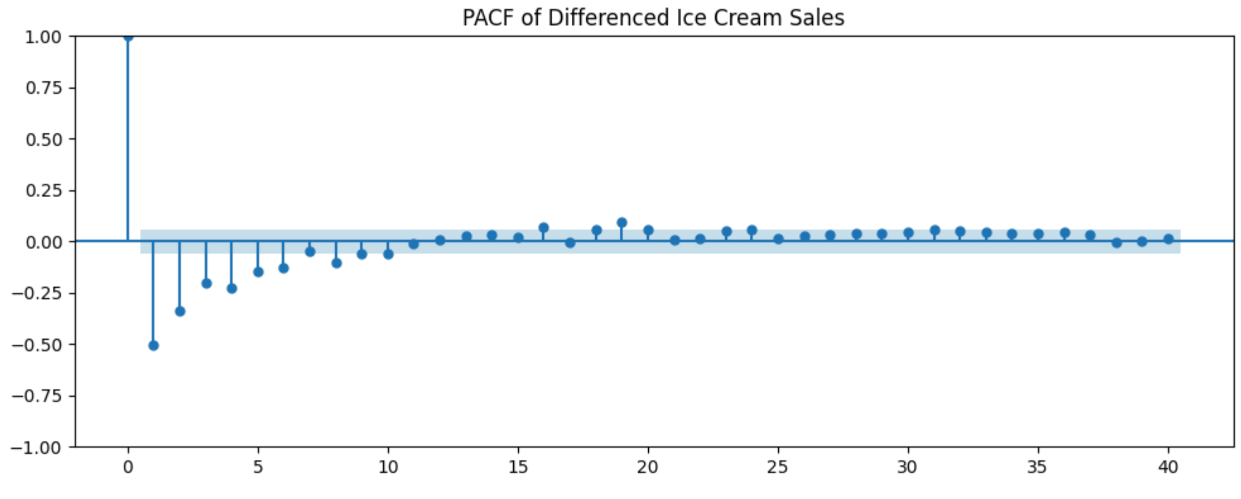
Exponential smoothing is a more flexible smoothing technique that, unlike simple moving averages, assigns more weight to recent observations while still considering past data. This makes it especially useful for capturing recent trends without completely discarding older information. The smoothing is controlled by a parameter, α, which ranges from 0 to 1. Higher values of α give more emphasis to recent data, making the data more responsive to change. Exponential smoothing is effective for short-term forecasting and works well when there is no clear seasonality or complex patterns. Its ability to adapt to changing trends, while still filtering out noise, makes it an effective and intuitive method in time series analysis. For data with stronger trends or seasonal components, we can extend this method using double or triple exponential smoothing.

Modeling

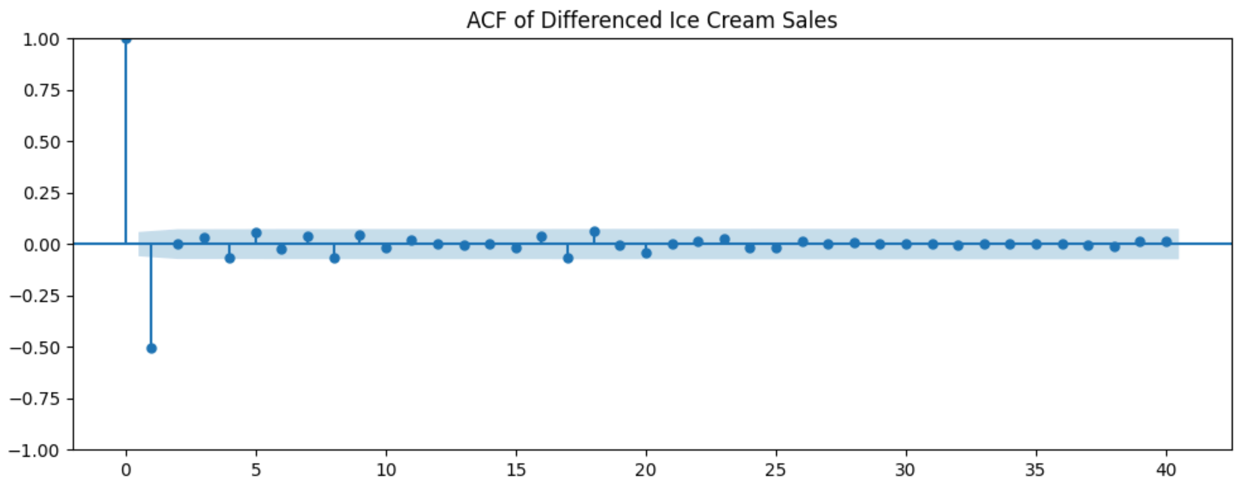
So far, we've smoothed the noise, uncovered hidden patterns, and tamed the chaos in our time series. Now it's time to turn those insights into actions. Welcome to the heart of time series analysis: **modeling**. In this section, we'll dive into the models that let us move beyond visual interpretation and start making meaningful forecasts on historical data.

Time series models use past and current values to explain patterns in the data or forecast future behavior. Before we can build effective models, it's important to ensure the data is stationary, since most models assume constant statistical properties over time. Some of the most common and basic types of time series models include **AR (Autoregressive)**, **MA (Moving Average)**, **ARMA (Autoregressive Moving Average)**, **ARIMA (Autoregressive Integrated Moving Average)**, and **SARIMA (Seasonal ARIMA)**. While we won't dive into the technical details of each model here, it's important to know that tools like the **Autocorrelation Function (ACF)** and **Partial Autocorrelation Function (PACF)** are especially helpful when setting up these models, as they guide in selecting appropriate parameters.

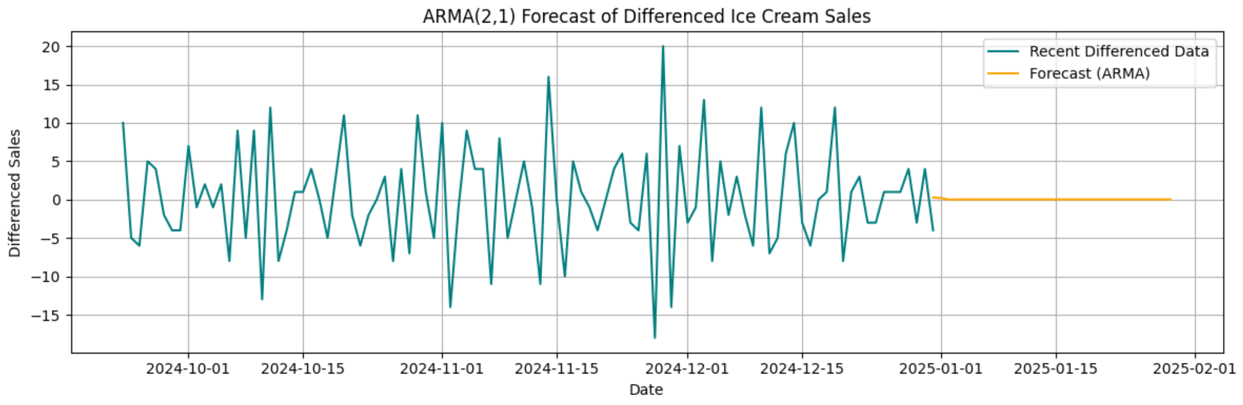
Time series models often begin with the idea of **autoregression**, where we use past values to predict future ones. These are called Autoregressive (AR) models, and they assume that the current value in a series depends on its previous values. But this raises an important question: how many past values should we include in the model? This is where the concept of **lag** comes in. A lag refers to how far back in time we look. For example, in our dataset, lag 1 compares today's sales to yesterday's sales, lag 2 looks back two days, and so on. To decide the appropriate number of lags to include in an AR model, we use the Partial Autocorrelation Function (PACF), which measures the correlation between the time series and its lagged value after removing the effects of the shorter lags. Specifically, in our example, the direct relationship between today and n days ago, after removing the effects of everything in between. In our example, the PACF plot of our differenced ice cream sales data showed significant spikes at lags 1 and 2, with all subsequent bars falling within the confidence bands. This suggested that including two past values would be sufficient, so we set the autoregressive parameter to $p = 2$ (AR(2)).



Another important type of time series model is the moving average (MA) model, where instead of using past values to predict the current one, we use past forecast errors. In an MA model, the idea is that the current value depends on random shocks or surprises from previous steps. This leads to the question: How many past errors should we include in the model? To help answer this, we use the Autocorrelation Function (ACF), which measures how correlated the current value is with its past values at different lags. In our example, the ACF plot of the differenced series showed a strong spike at lag 1, with later lags dropping off quickly. This indicated that the error from one time step back was important, so we set the moving average parameter to q = 1 (MA(1)).



Bringing these two ideas together, an ARMA model combines both the autoregressive (AR) and moving average (MA) components into a single, more powerful model. Mathematically, an ARMA model expresses the current value as a linear combination of both past observations and past errors. Specifically, it combines the lagged values (from the AR part) and the lagged forecast errors (from the MA part) to explain the behavior of the series. We fitted an ARMA(2,1) model to our stationary, differenced ice cream sales data using the statsmodels.tsa library in Python.



After fitting the model, we forecasted the next 30 days (orange). The forecast, however, quickly flattened around zero, suggesting that the differenced series lacks strong upward or downward momentum. This result is typical for stationary series dominated by randomness, and it highlights that while ARMA can capture short-term dependencies, it may not fully model more complex structures like seasonality or long-term growth, something that more advanced models like SARIMA are better suited for.

Conclusion

In this walkthrough, we explored the foundational steps of time series analysis. We began by visually examining our dataset to identify patterns such as trends, seasonality, and noise. We then used smoothing techniques to clarify these patterns and applied the ADF test to assess stationarity. Once our data was properly transformed, we used ACF and PACF plots to guide the selection of model parameters and fitted an ARMA model to make forecasts. These steps highlight a typical modeling pipeline: explore → transform → model → forecast.

While this gives a solid starting point, it’s important to note that what we’ve covered here only scratches the surface of time series analysis. Real-world applications often involve more complex structures, external variables, and more advanced models. Still, mastering the basics lays the groundwork for tackling those more complex challenges with confidence. Much of the content in this article was adapted from Filippo Maria Bianchi’s excellent *Python Time Series Handbook*. For a more detailed and technical exploration, you can view the full resource here.