How to calculate SMAPE in Python

Authored by

stats writer

December 23, 2025

The Symmetric Mean Absolute Percentage Error (SMAPE) stands as a sophisticated and widely utilized metric within the field of quantitative forecasts evaluation, particularly in scenarios where accurately measuring prediction accuracy is paramount, such as in supply chain management or financial modeling. Unlike simpler metrics, SMAPE provides a normalized measure of error, making it suitable for comparing the performance of forecasting models across different datasets or scales. The fundamental principle involves taking the average of the absolute percentage errors, where the denominator is the average of the absolute values of the actual and forecasted figures, ensuring symmetry regardless of whether the actual value or the forecast value is zero. This normalization helps mitigate certain biases inherent in traditional percentage error metrics, offering a more robust assessment of model quality.

Calculating SMAPE in Python is straightforward, although it requires the creation of a custom function since it is not included in standard libraries like Scikit-learn or SciPy. The calculation involves iterating through all data points, determining the absolute difference between the predicted and actual values, and then normalizing this difference by the average of their absolute magnitudes. This individual percentage error is then scaled by 200 (or 100 for a simple average percentage error, then multiplied by 2 to account for the symmetric denominator) and finally averaged across the entire dataset. The resulting value is always expressed as a percentage, which provides immediate and intuitive interpretability regarding the model's accuracy level, offering crucial insights for model optimization and selection processes.

Introduction to Symmetric Mean Absolute Percentage Error (SMAPE)

The **symmetric mean absolute percentage error (SMAPE)** is an essential statistical tool employed extensively across various industries to quantify the predictive accuracy of time series models and machine learning algorithms. Its design specifically addresses some of the mathematical limitations found in its predecessor, the Mean Absolute Percentage Error (MAPE), making it particularly useful in situations where data may contain values close to zero. By providing a percentage-based metric, SMAPE ensures that the error measurement is scale-independent, which is a critical feature when comparing the performance of models predicting vastly different volumes or magnitudes, such as sales forecasts for high-volume consumer goods versus specialized industrial equipment. Understanding and correctly implementing SMAPE is fundamental for any data scientist working in forecasting.

At its core, the utility of a percentage error metric lies in its ease of communication; stating that a model has an error of 10% is far more intuitive to stakeholders than reporting an Mean Absolute Error (MAE) of 500 units without context. However, standard MAPE suffers from a severe issue: if the actual value is zero, the division results in an infinite error, rendering the metric useless. SMAPE resolves this mathematical instability by using the average of the actual value and the

forecasted value in the denominator. This symmetry ensures that the metric is bounded (ranging from 0% to 200%) and prevents the calculation from crashing due to zero values, thereby offering a more stable and reliable measure of forecasting performance across the entire dataset, even in highly volatile scenarios or intermittent demand patterns.

In essence, SMAPE calculates the percentage difference between the forecast and the actual value relative to their sum. This normalization process ensures that the resulting error is balanced, meaning that positive and negative errors of the same magnitude relative to the data points contribute equally to the overall metric. This balanced approach is highly valuable in fields like inventory management, where under-forecasting (leading to stockouts) and over-forecasting (leading to excess inventory costs) both carry significant but often different financial penalties. By adopting SMAPE, organizations gain a standardized, reliable metric for benchmarking internal forecasting processes against industry standards and optimizing model deployment strategies.

Why Traditional Percentage Errors Fall Short

While metrics like MAPE are popular due to their intuitive percentage interpretation, they carry significant inherent flaws that often skew model evaluation, particularly when dealing with real-world business data. The most glaring issue with MAPE, as noted, is the division by zero problem, which occurs whenever the actual observed value is zero, a common occurrence in time series analysis for products with intermittent or seasonal demand. Furthermore, even when actual values are non-zero but very small, MAPE introduces a strong bias. Specifically, MAPE assigns disproportionately higher penalties to negative errors (when the forecast exceeds the actual value, resulting in positive percentage error blow-up) compared to positive errors (when the forecast is lower than the actual value), making the evaluation asymmetrical and often misleading.

Consider a scenario where the actual value is 10, and the forecast is 5 (50% under-forecast). Now consider a scenario where the actual value is 5, and the forecast is 10 (100% over-forecast, according to standard MAPE formula relative to the actual value). This asymmetry means that MAPE penalizes under-forecasting much less severely than over-forecasting when the actual value is used as the denominator, leading models to potentially minimize the metric by systematically under-predicting. This is an unacceptable bias in many practical applications. Financial institutions, for instance, require error metrics that treat positive and negative deviations equally to maintain risk integrity, necessitating a move toward symmetric measures like SMAPE or alternative approaches based on squared errors, such as Root Mean Squared Error (RMSE).

SMAPE effectively overcomes these systemic disadvantages by normalizing the absolute difference by the average of the actual and forecasted values. This symmetric denominator ensures that the error calculation is balanced, providing a measure that is fundamentally independent of the order of the terms (actual vs. forecast). Consequently, SMAPE provides a more

balanced perspective on the model's performance, avoiding the extreme inflation of errors near zero and maintaining a reliable, bounded range. This makes SMAPE an indispensable tool for official forecasting competitions and critical business planning where a robust and unbiased performance indicator is non-negotiable for comparing different statistical approaches, including complex deep learning models and simpler ARIMA methods.

Deconstructing the SMAPE Formula

To accurately implement and understand SMAPE, one must first grasp the mathematical components of its official formulation. The formula for the symmetric mean absolute percentage error (SMAPE) across a sample size n is defined as:

SMAPE = (1/n) * Σ(|forecast - actual| / ((|actual| + |forecast|)/2)) * 100

This expression might initially appear complex, but breaking down its elements reveals a logical structure designed for symmetry and stability. The entire calculation starts with the summation operator, indicating that we must calculate the error for every single data point and then aggregate them. This aggregated sum is then averaged by multiplying by the inverse of the sample size ($1/n$). The crucial component lies within the fraction that defines the individual percentage error for each observation.

The numerator, **|forecast - actual|**, simply represents the absolute magnitude of the prediction error--how far the forecast was from the truth, regardless of direction. The denominator, **((|actual| + |forecast|)/2)**, is the core innovation of SMAPE. This term calculates the average of the absolute values of the actual data point and its corresponding forecast. By using this average, we create a reference point for the percentage calculation that is inherently symmetric. If we multiply the entire fraction by 100, we express the result as a percentage error for that specific point. Note that in some academic literature, the factor of 2 (from the denominator average) is moved to the numerator outside the summation, leading to the alternative definition: $text{SMAPE} = frac{100}{n} sum frac{2 cdot |f_i - a_i|}{|a_i| + |f_i|}$.

Understanding the variables involved in the calculation is crucial for translation into programming code. We can summarize the terms as follows:

Σ - The mathematical symbol indicating the required "sum" across all observations in the dataset, encompassing the entire time series or sample.
n - Represents the total "sample size" or the count of data pairs (actual and forecast) being evaluated, ensuring the metric is an average error.
actual - Refers to the true, observed data value at a given point in time or observation index.
forecast - Refers to the predicted or estimated data value generated by the model for that same point in time.

This tutorial focuses specifically on implementing this mathematical definition efficiently within the Python environment, utilizing vectorization capabilities provided by the NumPy library for high performance.

Prerequisites for SMAPE Calculation in Python

Calculating SMAPE in Python requires handling array operations efficiently, which necessitates the use of the **NumPy** library. NumPy is the foundational package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a vast collection of high-level mathematical functions to operate on these arrays. Because SMAPE involves element-wise subtraction, addition, absolute value calculation, and division across two corresponding arrays (actuals and forecasts), NumPy's vectorization capabilities are ideal for writing clean, fast, and scalable code.

Before proceeding with the function definition, ensuring that NumPy is installed and imported correctly is the first essential step. Installation is typically achieved via pip (`pip install numpy`). Once installed, importing it under the standard alias `np` allows us to utilize its vectorized functions, such as `np.abs()` for calculating absolute values and `np.sum()` for summing array elements, which are both crucial components of the SMAPE formula. The use of vectorized operations bypasses slow Python loops, drastically speeding up calculations, especially when dealing with hundreds of thousands or millions of data points, which is common in large-scale forecasting projects.

It is important to emphasize that while Python does not offer a native, built-in function specifically named `smape()`, creating a reusable function using NumPy is the standard industry practice. This approach allows developers and data scientists to maintain consistency and modularity in their codebase. Furthermore, because SMAPE is defined mathematically using array operations, its translation into a vectorized NumPy function is nearly direct, minimizing the risk of implementation errors compared to complex manual looping structures. The input data must be prepared as two NumPy arrays of equal length, ensuring a one-to-one correspondence between each actual value and its respective forecast.

Implementing the Custom SMAPE Function using NumPy

The efficient calculation of SMAPE hinges on translating the mathematical formula into a vectorized Python function. We will define a function, typically named `smape`, that accepts two arrays: `a` (actual values) and `f` (forecasted values). Using NumPy allows us to perform all arithmetic operations simultaneously across the entire array, adhering perfectly to the structure of the summation ($Sigma$) in the mathematical definition.

The code below demonstrates how to construct this function. Notice how each part of the formula

is mapped directly to a NumPy operation: `np.abs(f-a)` calculates the absolute difference (the numerator), and `(np.abs(a) + np.abs(f))` calculates the sum of the absolute values (the core of the symmetric denominator). The division and multiplication by 2 (or 200, depending on the normalization style) are performed element-wise before the final summation and averaging step. This implementation provides a robust, highly performant calculation method.

import numpy as np

def smape(a, f):
return 1/len(a) * np.sum(2 * np.abs(f-a) / (np.abs(a) + np.abs(f))*100)

A crucial aspect of this implementation is the handling of the sample size, n. In Python, when working with NumPy arrays, the size n is obtained using `len(a)`. The function calculates the element-wise percentage error term: $2 cdot |f-a| / (|a| + |f|) cdot 100$. This term is summed using `np.sum()`, and then divided by the total number of observations, effectively calculating the mean. This entire structure ensures that the resultant value is the average symmetric percentage error across the entire provided sample, delivering the final SMAPE metric.

While this function is robust, specialized attention should be paid to potential edge cases, although the symmetric denominator significantly reduces risks. If both the actual and forecasted values are zero simultaneously (a rare but possible scenario), the denominator becomes zero. To handle this, professional implementations often include a small epsilon value or a conditional check to return 0% error in this specific instance, as a perfect zero forecast of a zero actual value should logically result in zero error. However, for most standard datasets and initial implementations, the presented NumPy function is sufficient and highly accurate.

Practical Application: Calculating SMAPE on Sample Data

Once the custom `smape` function is defined, we can utilize it immediately to calculate the predictive accuracy for any pair of arrays representing actual observations and model forecasts. This section demonstrates a simple, yet representative, example using small arrays to illustrate the calculation process and how the function handles the input data structure. This is the stage where the power of NumPy vectorization truly shines, allowing complex calculations to be executed with minimal lines of code.

We begin by defining our sample datasets. We require two arrays, `actual` and `forecast`, which must contain numerical data and be of the same length, ensuring that each prediction corresponds directly to an observed reality. For this example, we define seven data points, simulating a short time series or a small batch of predictions. The chosen values are designed to show varying degrees of prediction accuracy, including near-perfect matches and noticeable deviations, allowing

for a comprehensive test of the SMAPE function.

#define arrays of actual and forecasted data values
actual = np.array()
forecast = np.array()

#calculate SMAPE
smape(actual, forecast)

12.45302

Executing the function `smape(actual, forecast)` initiates the vectorized calculation. NumPy handles the element-wise processing: for the first pair (actual=12, forecast=11), the absolute difference is 1, and the symmetric denominator is $(12+11)/2 = 11.5$. The percentage error for this point is $(1 / 11.5) cdot 200 approx 17.39%$. This process is repeated for all seven data points. The individual percentage errors are summed together, and the total sum is divided by 7 (the sample size n), yielding the final, average SMAPE value. The output `12.45302` is the result of this aggregation, providing the overall measure of model performance for this specific dataset.

Interpreting the SMAPE Result and Model Evaluation

The resulting value of **12.45302%** derived from our sample calculation is the **symmetric mean absolute percentage error** for the model applied to this specific dataset. Interpretation of SMAPE is straightforward: the lower the value, the better the predictive accuracy of the model. A SMAPE of 0% indicates a perfect forecast where the predicted values exactly match the actual values for every observation. Conversely, a SMAPE approaching 200% (the theoretical maximum) indicates extremely poor forecasting performance, although values rarely exceed 100% in meaningful scenarios.

This result of 12.45302% implies that, on average, the model's predictions deviated from the actual values by approximately 12.45%, relative to the average of the actual and predicted values. In a business context, this metric serves as a powerful benchmark. If a company typically aims for an error rate below 10%, a SMAPE of 12.45% indicates that the current model is underperforming relative to internal targets. Such a result would necessitate further investigation into the model parameters, feature engineering, or the underlying data quality, potentially leading to the selection of an alternative forecasting methodology that minimizes this error. The percentage format makes this evaluation immediately actionable for both technical teams and business stakeholders.

Furthermore, SMAPE is often used in model comparison. If Model A produces a SMAPE of 12.45% and Model B produces a SMAPE of 9.80% on the same validation dataset, Model B is unequivocally the superior choice in terms of percentage accuracy. This objective, standardized

comparison is vital when deploying complex systems, ensuring that the final selection is based on robust statistical evidence rather than qualitative assessments. The bounded and symmetric nature of SMAPE ensures that this comparison is fair, regardless of whether the actual values in the dataset contained zeros or highly skewed distributions, reinforcing its reliability over non-symmetric metrics like MAPE.

Advantages and Disadvantages of Using SMAPE

While SMAPE is a highly valuable metric, particularly known for its symmetry and robustness against division by zero, it is crucial for expert practitioners to recognize both its strengths and weaknesses when choosing an evaluation criterion. One primary advantage is its **scale independence**, allowing error comparison across series with different units or magnitudes. Second, the **symmetry** inherent in the denominator structure ensures that errors are treated equally regardless of whether the forecast is an overestimate or an underestimate, correcting a major bias found in standard MAPE. Third, its **bounded nature** (from 0% to 200%) simplifies interpretation and prevents the metric from exploding to infinity, providing stability in model evaluation.

However, SMAPE is not without its limitations. Despite its improvement over MAPE, SMAPE can still be susceptible to extreme penalties in certain boundary conditions. When both the actual value and the forecast are very close to zero, the denominator (the average of the two absolute values) is also very small. If the absolute difference is even slightly non-zero in this scenario, the resulting percentage error can still be significantly magnified. While not infinite, this inflation can disproportionately influence the overall mean. For datasets dominated by zero or near-zero values, alternative metrics like MAE (Mean Absolute Error) or RMSE might still be preferred if the goal is to minimize absolute deviation rather than relative percentage error.

A second, subtle disadvantage is that SMAPE, like other percentage error metrics, can be challenging to optimize directly using standard machine learning optimization algorithms, which often prefer differentiable loss functions like squared error. While SMAPE is used extensively for final evaluation, models are frequently trained using Mean Squared Error (MSE) or MAE, and then SMAPE is calculated only after training for reporting purposes. Therefore, the choice of SMAPE must be balanced against the computational requirements and optimization path of the specific modeling technique being employed. Data scientists must weigh the interpretability benefits of SMAPE against the optimization challenges posed by its non-linear, non-differentiable structure.

Further Reading and Related Metrics

Understanding SMAPE often involves comparing it to a spectrum of other forecasting error metrics. While SMAPE addresses percentage error biases, other metrics like MAE, RMSE, or even MAPE

remain relevant depending on the specific business requirements and statistical properties of the data being analyzed. For instance, RMSE heavily penalizes large errors, making it suitable for scenarios where large deviations are highly costly, whereas MAE is more robust to outliers.

For those looking to deepen their expertise in forecasting evaluation and related Python implementations, we recommend exploring the detailed documentation and discussions surrounding these metrics. The official resources provide comprehensive explanations of the theoretical foundations and practical implications of choosing one metric over another. Continuous learning in this area is essential for developing robust and reliable predictive models.

Below are several high-quality resources for continuing your study of SMAPE and related forecasting accuracy measures:

Wikipedia Entry for SMAPE
Rob J. Hyndman's thoughts on SMAPE
How to Calculate MAPE in Python
How to Calculate MAPE in R
How to Calculate MAPE in Excel