Learning Time Series Analysis: A Practical Guide to the KPSS Test in Python

Authored by

Mohammed looti

October 31, 2025

Introduction to Time Series Stationarity and the KPSS Test

Time series analysis stands as a fundamental pillar of modern data science, finance, and **econometrics**, focusing intently on sequences of data points indexed, most often, in time order. A foundational concept that dictates the appropriate selection of models in this domain is **stationarity**. A time series is considered stationary if its statistical properties--such as the mean, variance, and autocorrelation structure--remain constant over time. This constancy is not merely an academic ideal; it is a critical prerequisite because the underlying assumptions of many powerful **time series models**, including **ARIMA** (Autoregressive Integrated Moving Average), require the data generating process to exhibit stationarity.

To accurately characterize the behavior of a time series, it is essential to differentiate between various types of stationarity. A series is typically categorized as either **level stationary** (fluctuations around a constant mean) or **trend stationary**. A trend-stationary series possesses a deterministic trend component, meaning the series fluctuates around a predictable, non-random trend (e.g., a straight line). If this deterministic trend is successfully removed, the remaining residual series must be stationary. This contrasts sharply with a difference-stationary series, where non-stationarity is caused by a stochastic trend or a **unit root**, necessitating differencing to achieve stationarity. Making this crucial distinction is the first step toward selecting an effective time series modeling strategy.

To statistically confirm the nature of a time series, researchers rely on formal hypothesis tests. The **KPSS test**, named after Kwiatkowski, Phillips, Schmidt, and Shin, is a widely utilized statistical procedure specifically designed to assess if a time series is trend stationary. Its formulation is unique and highly valuable because, unlike traditional unit root tests such as the **Augmented Dickey-Fuller (ADF) test**, the KPSS test employs a null hypothesis that the series is stationary. This inverted perspective makes the KPSS test particularly effective for confirming stationarity rather than simply rejecting non-stationarity.

This comprehensive guide is designed to serve as a practical walkthrough for implementing the KPSS test efficiently using **Python**. We will meticulously break down the underlying statistical hypotheses, demonstrate how to interpret the numerical results, and provide hands-on examples using both stationary and non-stationary synthetic data. By the conclusion of this tutorial, you will possess a clear, functional understanding of how to leverage the KPSS test to conduct robust and reliable time series analysis.

Understanding the KPSS Test: Hypotheses and Interpretation

The core strength of the KPSS test lies in the structure of its hypotheses, which fundamentally differs from most other unit root tests. Understanding the definition of the **null hypothesis** (H0)

and the **alternative hypothesis** (HA) is absolutely essential for correctly interpreting the test's outcomes.

The hypotheses for the KPSS test are formally defined as follows, assuming the test is configured to account for a linear trend (the `regression='ct'` setting in Python):

H0 (Null Hypothesis): The time series is **trend stationary**. This implies that the series can be made stationary by modeling and removing a deterministic linear trend and/or a constant mean. Non-stationarity is not due to a unit root.
HA (Alternative Hypothesis): The time series is *not* trend stationary. This conclusion suggests that the series contains a unit root (a stochastic trend) and is therefore difference-stationary, requiring differencing before standard modeling techniques can be applied.

The process of interpreting the test results centers on analyzing the **p-value** generated by the test and comparing it against a predetermined **significance level**, conventionally denoted as alpha ($alpha$) and often set at 0.05. If the calculated p-value is smaller than the chosen significance level (p < $alpha$), then the result is deemed statistically significant, providing enough evidence to **reject the null hypothesis** (H0). In the specific context of the KPSS test, rejecting H0 means that the data is non-stationary, likely containing a unit root. Conversely, if the p-value is greater than or equal to the significance level (p geq $alpha$), we **fail to reject the null hypothesis**. This outcome suggests that there is insufficient evidence to conclude that the series is non-stationary, and we proceed under the assumption that it is trend stationary. It is important to emphasize that "failing to reject" H0 does not equate to "proving" or "accepting" stationarity; it simply means the data does not offer strong proof of non-stationarity.

Beyond the p-value, the KPSS test output also provides a calculated **test statistic** and a set of **critical values** corresponding to various significance levels (e.g., 1%, 5%, 10%). An alternative, and often complementary, method for decision-making is to compare the calculated test statistic directly to these critical thresholds. If the test statistic is greater than the critical value at a chosen significance level, this result also leads to the rejection of the null hypothesis. This comparison is particularly useful when the reported p-value is truncated (e.g., reported simply as "> 0.1" or "< 0.01") or if the exact p-value is not precisely known due to limitations in the statistical tables used by the software.

Setting Up Your Python Environment for KPSS

Before we transition into the practical application of the KPSS test, it is paramount to ensure that your Python environment is correctly configured with the necessary libraries. The implementation of the KPSS test, along with the visualization and data manipulation required for time series analysis, primarily relies on three robust and widely adopted libraries: **NumPy**, **Matplotlib**, and **statsmodels**.

Each of these libraries plays a distinct and crucial role in our workflow. **NumPy** is the cornerstone of numerical computing in Python, providing essential support for high-performance, multi-dimensional array objects and a vast collection of sophisticated mathematical functions. We will utilize NumPy extensively for the efficient generation and manipulation of our synthetic time series data examples. Complementing this is **Matplotlib**, the industry-standard library for generating static, animated, and interactive visualizations. Plotting the time series data allows for crucial visual inspection of underlying trends, seasonality, or lack thereof, which serves as an important visual check alongside the formal statistical test.

The statistical heavy lifting is performed by **statsmodels**, a specialized Python module that offers extensive classes and functions for estimating statistical models, conducting various statistical tests, and facilitating statistical data exploration. Specifically, the function required for our analysis, `sm.tsa.stattools.kpss`, is housed within this indispensable library. Statsmodels ensures that the complex mathematics of the KPSS test are implemented accurately and reliably.

If these packages are not yet available in your environment, they can be installed quickly and easily using `pip`, the standard Python package installer. Execute the following single command in your terminal or command prompt to prepare your workspace:

```
pip install numpy matplotlib statsmodels
```

Once the installation is complete, you are fully prepared to import these libraries into your preferred Python environment (such as a Jupyter Notebook or script) and begin applying the KPSS test to real-world or synthetic time series data.

Example 1: Performing the KPSS Test on Stationary Data

To establish a baseline understanding of how the KPSS test functions when its null hypothesis holds true, we will first generate a synthetic dataset that is inherently stationary. This data, often referred to as white noise, is constructed using random draws from a **normal distribution** via **NumPy**. By definition, this type of data lacks any deterministic trend, stochastic trend (unit root), or time-varying variance, confirming its stationary nature.

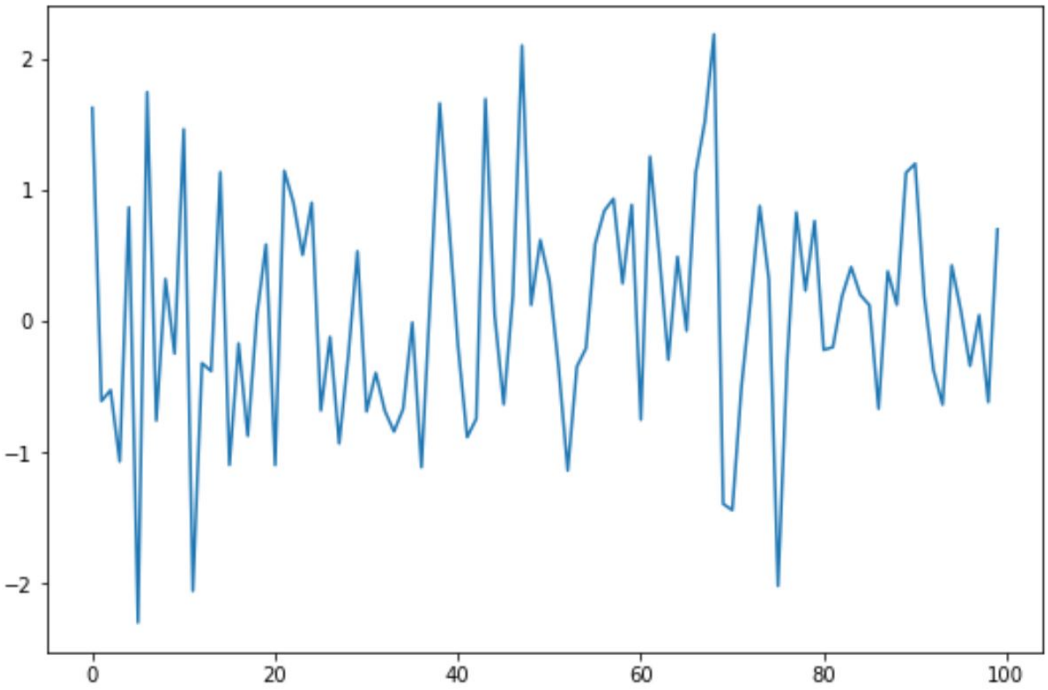
The code below initializes a reproducible state by setting a random seed, ensuring consistent results across multiple executions. It then generates 100 data points conforming to a standard normal distribution. Following data generation, **Matplotlib** is used to visualize the series. A quick visual inspection of the resulting plot should confirm that the data fluctuates randomly around a constant mean (zero), exhibiting no obvious upward or downward trend.

import numpy as np
import matplotlib.pyplot as plt

```
#make this example reproducible
np.random.seed(1)

#create time series data
data = np.random.normal(size=100)

#create line plot of time series data
plt.plot(data)
```



With the stationary data successfully generated, we proceed to execute the KPSS test using the **`kpss()` function** imported from the **statsmodels** package. Crucially, we set the parameter `regression='ct'`. This parameter instructs the test to assume trend stationarity under the null hypothesis, meaning the test accounts for both a constant mean and a linear time trend when assessing stability.

```
import statsmodels.api as sm

#perform KPSS test
sm.tsa.stattools.kpss(data, regression='ct')

(0.0477617848370993,
0.1,
1,
```

{'10%': 0.119, '5%': 0.146, '2.5%': 0.176, '1%': 0.216})

InterpolationWarning: The test statistic is outside of the range of p-values available
in the look-up table. The actual p-value is greater than the p-value returned.

The output is a tuple containing four elements critical for interpretation. The primary elements are the **KPSS test statistic** (0.04776) and the **p-value** (0.1). The remaining elements are the truncation lag parameter (1) and the set of **critical values**. Using the standard significance level of $alpha = 0.05$, we compare the p-value: $0.1 geq 0.05$. Since the p-value is not smaller than $alpha$, we **fail to reject the null hypothesis**. This statistical conclusion confirms that the time series is consistent with the assumption of **trend stationarity**, matching our expectations for white noise data. Additionally, we can confirm this result by comparing the test statistic (0.04776) to the 5% critical value (0.146); since 0.04776 is less than 0.146, we again fail to reject H0. The accompanying `InterpolationWarning` simply reinforces the decision, indicating that the true p-value is even greater than 0.1.

Example 2: Performing the KPSS Test on Non-Stationary Data

Our second example explores a scenario where the data clearly violates the assumption of stationarity, typically exhibiting a strong upward or downward trend. This exercise demonstrates how the KPSS test effectively identifies non-stationarity and leads to the rejection of the null hypothesis. We construct a simple dataset that incorporates a persistent, non-random upward movement over time. Such data is highly representative of many real-world economic and financial time series that display long-term growth.

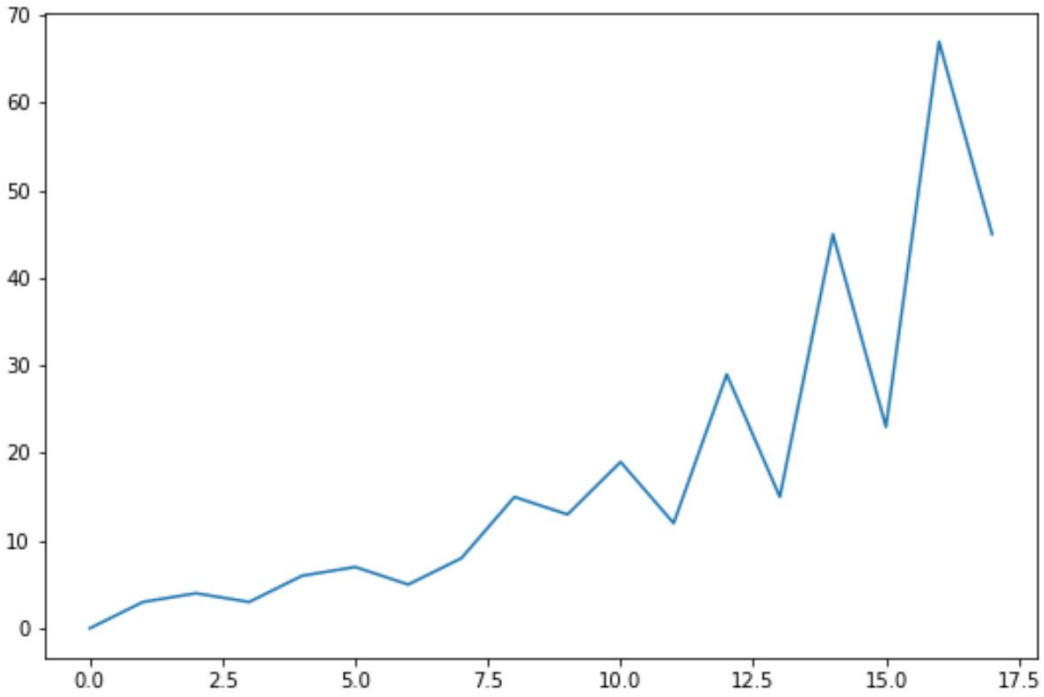
We again use **NumPy** to define a short series of values that exhibit a clear, increasing trend. Setting the random seed ensures the reproducibility of this example. The subsequent **Matplotlib** visualization will visually confirm the non-stationary nature of the series, showing that the mean of the data changes significantly over the observation period.

import numpy as np
import matplotlib.pyplot as plt

#make this example reproducible
np.random.seed(1)

#create time series data
data =np.array()

#create line plot of time series data
plt.plot(data)



With the non-stationary data prepared and visually confirmed, we proceed to apply the **kpss() function** from the **statsmodels** library. As before, the `regression='ct'` parameter is used, ensuring the test is attempting to fit a constant and a linear trend under the null hypothesis of stationarity.

import statsmodels.api as sm

#perform KPSS test
sm.tsa.stattools.kpss(data, regression='ct')

(0.15096358910843685,
0.04586367574296928,
3,
{'10%': 0.119, '5%': 0.146, '2.5%': 0.176, '1%': 0.216})

Interpreting the output for this non-stationary series is straightforward. We observe the **KPSS test statistic** is **0.1509**, and the resulting **p-value** is **0.0458**. When comparing the p-value (0.0458) to the standard significance level ($alpha = 0.05$), we find that $0.0458 < 0.05$. Because the p-value is less than alpha, we have statistically significant evidence to **reject the null hypothesis** of trend stationarity. This decisive rejection leads us to the conclusion that the time series is non-stationary, likely containing a unit root, which is consistent with the clear upward trend observed in the plot. Furthermore, if we compare the test statistic (0.1509) to the 5% critical value (0.146), we see that $0.1509 > 0.146$, which provides alternative confirmation that H0 must be rejected at the

5% level.

Interpreting KPSS Results and Next Steps

Successful application of the KPSS test requires more than just executing a function; it demands a clear understanding of the decision rules and the necessary subsequent actions in **time series analysis**. The fundamental decision rule is unequivocal: if the p-value falls below the chosen significance level ($alpha$), or if the test statistic exceeds the relevant **critical value**, the null hypothesis of stationarity is rejected, and the series is flagged as non-stationary.

When the KPSS test strongly indicates non-stationarity, it signals that the data requires transformation before being fed into models that rely on stationarity (e.g., standard ARMA or certain regression models). The most common and effective technique for stabilizing a non-stationary series is **differencing**. First-order differencing, calculated as the difference between an observation and the previous observation ($X_t - X_{t-1}$), often eliminates a linear trend. For series exhibiting more complex structures, such as quadratic trends or strong seasonal patterns, higher-order or seasonal **differencing** may be required to achieve the desired level of stability. After transformation, the KPSS test should be re-run on the differenced data to confirm that stationarity has been achieved.

It is crucial to adopt a holistic approach when analyzing time series data. Statistical tests should always be complemented by a thorough visual inspection of your **time series plots**. Visual examination can rapidly reveal persistent trends, seasonality, or structural breaks that might not be fully captured or explained by a single test statistic. Furthermore, because the KPSS test has a null hypothesis of stationarity, it is often employed alongside tests that have a null hypothesis of non-stationarity, such as the **Augmented Dickey-Fuller (ADF) test**. Consistent results from both tests (e.g., KPSS rejects H0, ADF fails to reject H0) provide the strongest evidence regarding the true nature of the series. Contradictory results, however, warrant deeper investigation into the data structure and the specific type of non-stationarity present.

Conclusion and Further Resources

The **KPSS test** is an indispensable and unique instrument for any professional engaged in **time series data** analysis. By offering a robust method to assess **trend stationarity** through the inversion of the standard null hypothesis, it greatly enhances the reliability of subsequent model selection and forecasting efforts. Mastery of its implementation in **Python**, particularly using the **statsmodels** library, ensures that data analysts can confidently identify and address the stationarity requirements of their datasets.

We have successfully navigated the theoretical underpinnings, the practical interpretation of the p-value and test statistic, and the hands-on application of the KPSS test using reproducible

examples. Remember that ensuring stationarity is a foundational step for unlocking the full potential of many predictive time series models. By integrating the KPSS test into your data preprocessing workflow, you are better equipped to produce accurate models and derive meaningful insights from temporal data.

For those interested in exploring the technical specifics of the `kpss()` function and expanding their knowledge of time series functionalities available within the `statsmodels` package, the following official documentation links are highly recommended:

KPSS function documentation (statsmodels)

Additionally, deepen your expertise in time series analysis with these valuable resources:

statsmodels Time Series Analysis documentation
Wikipedia: Time Series Analysis