Perform Runs Test in R

Authored by

Mohammed looti

November 7, 2025

The Wald-Wolfowitz Runs Test: An Essential Tool for Assessing Data Randomness

The Runs test, formally recognized as the **Wald-Wolfowitz runs test**, stands as a fundamental non-parametric statistical test crucial for robust data analysis, particularly within fields like quality control, finance, and scientific research. Its primary utility lies in rigorously evaluating whether a sequence of observed data points originates from a truly random process. This assessment of randomness is not merely academic; it forms the foundation for many advanced statistical methodologies, nearly all of which rely on the crucial assumption that the underlying data points are independently and identically distributed (i.i.d.). If this assumption is violated, the reliability of subsequent statistical inferences, such as regression analysis or time series modeling, is severely compromised.

When executing the Runs test, the focus shifts away from the specific numerical magnitudes of the observations and instead concentrates entirely on the sequential order in which they appear. The test operates by meticulously counting the number of "runs" present in the data sequence. A run is defined as a consecutive sequence of identical events, or, more commonly in quantitative analysis, a continuous sequence of values that are either all above or all below a designated central point, typically the **median** or the **mean** of the dataset. This counting mechanism provides a quantified measure of the degree of sequential structure within the data.

The number of runs observed dictates the nature of potential non-randomness. A sequence exhibiting significantly too few runs suggests strong clustering or the presence of **positive correlation**. For example, a scenario where long stretches of high values are consistently followed by long stretches of low values indicates a clear pattern rather than pure chance. Conversely, a dataset displaying significantly too many runs implies systematic oscillation or **negative correlation**, where values rapidly alternate between high and low states. In both extreme cases--too few or too many runs--the observed sequence order is statistically suspicious, leading to the conclusion that the core statistical assumption of independence among observations has been violated. Therefore, the Runs test provides an initial, powerful filter to validate data integrity before proceeding with more complex modeling.

Establishing Formal Hypotheses and Defining the Concept of a Run

To structure the statistical inference provided by the Runs test, we must first establish the formal null and alternative hypotheses, adhering to the standard framework of statistical hypothesis testing. These hypotheses clearly articulate the opposing perspectives regarding the underlying mechanism that generated the observed dataset, thereby framing the interpretation of the test results.

The hypotheses are meticulously defined as follows:

H0 (Null Hypothesis): The data sequence was generated in a truly **random manner**. The observed number of runs and the sequence order are statistically indistinguishable from what would be expected purely by chance.

Ha (Alternative Hypothesis): The data sequence was **not** generated in a random manner. The sequence exhibits a statistically significant systematic pattern, manifesting either as excessive clustering (too few runs) or rapid oscillation (too many runs).

The inference process hinges on a comparison between the actual, observed number of runs in the sample data and the expected number of runs, calculated under the strict assumption that the null hypothesis is true. A substantial deviation--either higher or lower--from this expected value results in a low p-value, leading to the rejection of H0 and confirming the presence of systematic non-randomness within the sequence. If the deviation is minor, the null hypothesis is retained, suggesting the data sequence is consistent with a random process.

This comprehensive tutorial will guide users through two distinct, yet equally robust, methods available within the powerful R programming environment for executing the Runs test. While these methods employ different external libraries, they are designed to perform the exact same statistical calculation, ensuring that the final conclusion regarding the data's inherent randomness remains consistent and verifiable across platforms.

Prerequisites: Configuring the R Environment for Non-Parametric Testing

Successful execution of the Runs test in R necessitates the correct installation and loading of specific external statistical packages. Unlike many foundational statistical operations, R's base installation does not include a native function dedicated solely to the Runs test. Consequently, reliance on high-quality, community-developed packages is mandatory. For this guide, we will focus on two of the most reliable and widely adopted options for non-parametric analysis: the **snpar** package and the **randtests** package.

Before attempting to run the practical code examples provided in the subsequent sections, it is essential that these libraries are correctly installed and loaded into your current R session. The standard procedure requires executing the `install.packages()` function once to permanently place the library files on your system, followed by using the `library()` function at the start of every new R session where the functions are needed. This preparation step ensures all necessary computational tools are immediately accessible.

Although both libraries achieve the same statistical goal, understanding their subtle differences is vital for informed analysis. The snpar package is particularly valued because it offers an explicit

option for calculating an **exact p-value**, a feature which becomes critically important when dealing with smaller sample sizes where asymptotic approximations might be unreliable. In contrast, the randtests package provides a broader, more comprehensive suite of randomness checks and time series tests, positioning it as a versatile foundation for a wide range of non-parametric analytical requirements.

Method 1: Utilizing the snpar Package and the runs.test() Function

Our first recommended approach leverages the robust capabilities of the snpar library, a package specifically designed for conducting various non-parametric statistical tests with precision. Within this specialized library, the function dedicated to assessing sequential randomness is logically named `runs.test()`. This function is known for its clarity of implementation and the clear, interpretable output it provides, making statistical decision-making straightforward.

The generalized syntax for invoking the `runs.test()` function within the **snpar** package offers crucial flexibility, allowing users to specify approximation methods and the directionality of the test:

```
runs.test(x, exact = FALSE, alternative = c("two.sided", "less", "greater"))
```

To illustrate the practical application of this function, we will now apply it to a standardized sample dataset. This example clearly demonstrates the entire workflow, starting from the necessary library inclusion and dataset definition, through to the final calculation of the test statistic and corresponding p-value.

The following R code block provides the necessary sequential commands. It includes loading the specified library, defining a simple numeric vector representing a sequence of observations, and subsequently executing the Runs test using the **snpar** implementation:

library(snpar)

# Create the sample dataset (numeric vector)
data <- c(12, 16, 16, 15, 14, 18, 19, 21, 13, 13)

# Perform the Runs test using snpar's function
runs.test(data)

Approximate runs rest

data: data
Runs = 5, p-value = 0.5023
alternative hypothesis: two.sided

The resulting output clearly presents the key findings: the calculated number of runs (in this case, 5) and the resultant p-value of 0.5023. This critical output immediately facilitates the transition into the final phase of statistical interpretation, where the result is compared against the pre-selected significance level, conventionally set at $alpha = 0.05$, to determine whether the null hypothesis should be rejected or retained.

Understanding the Parameters of snpar's runs.test()

Effective utilization of the `runs.test()` function requires a deep understanding of its primary arguments, which enable researchers to precisely tailor the statistical rigor and the directional focus of the test based on specific research questions and data characteristics.

The critical parameters governing the **snpar** implementation are threefold:

x: This mandatory argument represents the core input data. It must be a numeric vector containing the exact sequence of data values whose randomness is being subjected to the test.
exact: This is a logical parameter, accepting either **TRUE** or **FALSE**. By default, this is set to **FALSE**, which instructs the function to calculate an approximate p-value based on the normal distribution--an approach that is computationally efficient and generally suitable for large samples. However, if the sample size is limited, or if the number of calculated runs is small, setting this parameter to **TRUE** is strongly recommended. This forces the function to calculate the exact p-value using combinatorial methods, ensuring superior precision and reliability in the test result.
alternative: This parameter specifies the type of alternative hypothesis being evaluated. The default setting is **two.sided**, which is used to test for any deviation from randomness (i.e., detecting either too few runs, indicating clustering, or too many runs, indicating oscillation). Researchers can specify **less** to specifically test for clustering or **greater** to test specifically for oscillation, thus focusing the statistical power on a particular pattern of non-randomness.

The choice regarding the `exact` parameter is perhaps the most crucial decision. While the asymptotic approximation offers speed, the exact calculation, though relying on more intensive combinatorial mathematics, provides the highest degree of accuracy, especially vital for rigorous statistical studies involving limited datasets where the assumption of normality for the test statistic may not hold. Therefore, in cases demanding absolute certainty, accuracy should always be prioritized over computational speed.

Method 2: Leveraging the randtests Package for Comprehensive Randomness Testing

The second widely accepted and powerful method for conducting the Runs test in R involves implementing the specialized randtests package. As implied by its name, this library is meticulously

curated to provide a comprehensive array of statistical checks specifically designed to evaluate the randomness properties of various data sequences. The core function for the Runs test within this package retains the same name, `runs.test()`, though its internal structure and the required input parameters differ slightly when compared to the snpar version.

The simplified syntax for the `runs.test()` function within the **randtests** package reflects its focus on asymptotic methods:

```
runs.test(x, alternative = c("two.sided", "less", "greater"))
```

It is noteworthy that the `exact` parameter, which was central to precision in the **snpar** version, is explicitly absent here. The **randtests** package primarily relies on asymptotic approximations, suitable for larger samples, although it often incorporates internal logic to switch to exact methods depending on the data characteristics and sample size constraints, aiming to balance computational efficiency with statistical reliability.

The following R code block demonstrates the application of the **randtests** function using the identical sample dataset previously analyzed with the **snpar** package. This parallel execution is vital for confirming the consistency of the statistical conclusions, ensuring that the results are independent of the choice of external library.

```
library(randtests)

# Create the sample dataset
data <- c(12, 16, 16, 15, 14, 18, 19, 21, 13, 13)

# Perform the Runs test using randtests' function
runs.test(data)
```

```
Runs Test

data: data
statistic = -0.67082, runs = 5, n1 = 5, n2 = 5, n = 10, p-value =
0.5023
alternative hypothesis: nonrandomness
```

The output generated by the **randtests** package is typically more comprehensive and diagnostic. It includes the calculated test statistic (often presented as a z-score), the total number of runs (5), the counts of observations above (n1) and below (n2) the specified cutoff (usually the median), and the total sample size (n). Crucially, the reported p-value remains exactly **0.5023**, perfectly matching the result obtained using the **snpar** package, thus confirming the robustness of the finding.

Interpreting the Statistical Results and Final Conclusion

Irrespective of whether the snpar or the randtests package was employed, the final and most decisive stage in the hypothesis testing framework is the careful interpretation of the resulting p-value against the established significance level, designated as $alpha$. For almost all statistical analyses in academic and industry settings, the standard threshold for $alpha$ is set at 0.05.

In both demonstrations presented, the calculated p-value derived from the Runs test is consistently **0.5023**. We must now apply the standard and universally accepted statistical decision rule based on this comparison:

If the calculated p-value is less than or equal to $alpha$ (i.e., $p le 0.05$), we must **reject** the null hypothesis (H0). This implies that the observed data sequence is statistically incompatible with the assumption of randomness.
If the calculated p-value is greater than $alpha$ (i.e., $p > 0.05$), we **fail to reject** the null hypothesis (H0). This suggests that there is insufficient statistical evidence to conclude that the data sequence is non-random.

Since the resulting p-value of 0.5023 is considerably larger than the conventional significance level of 0.05, we decisively **fail to reject the null hypothesis**. This crucial statistical outcome signifies that, based on the rigorous scrutiny of the Runs test, we lack the statistically sufficient evidence required to assert that the data sequence is non-random or exhibits any systematic pattern of dependency.

Therefore, the definitive conclusion drawn from this analysis is that the data was produced in a manner entirely consistent with a random process. The observed ordering of the data points--neither showing excessive clustering nor pronounced oscillation--does not present statistically significant evidence that would violate the fundamental assumption of independence, thereby validating the sequence for further parametric statistical modeling.