

How to Perform Univariate Analysis in Python with Pandas and Seaborn

Authored by
stats writer

December 2, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Perform Univariate Analysis in Python with Pandas and Seaborn*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=103641>

Univariate analysis is a fundamental and essential technique in the initial stages of data exploration. It focuses exclusively on analyzing data from a single variable or feature at a time, providing crucial insights into its underlying structure, distribution, and central tendencies. The core purpose of this method is to describe the data, rather than explain relationships between variables (which is the domain of bivariate or multivariate analysis). By isolating one variable, we can accurately measure its characteristics without interference from other factors.

Executing effective univariate analysis often requires powerful computational tools. In the Python ecosystem, this process is streamlined using robust libraries such as Pandas for data manipulation, and visualization tools like Seaborn and Matplotlib for graphical representation. Examples of standard univariate tasks include exploring the shape of the data's distribution, calculating key measures of central tendency (like the mean, median, and mode), and quantifying the variability or spread (such as standard deviation and range). This preliminary exploration is vital for identifying errors, handling outliers, and ensuring the data is properly understood before complex modeling begins.

The Core Methods of Univariate Analysis

Understanding a single variable involves more than just calculating an average; it requires a comprehensive approach covering central location, frequency, and visual distribution. The term "univariate" itself is derived from the prefix "uni," meaning "one," clearly indicating its focus on a singular data characteristic. Mastering these techniques is the first step toward effective statistical modeling and machine learning preparation.

The three most common and powerful ways to perform a thorough univariate analysis on any given variable are through numerical summaries, tabular frequency counts, and graphical visualization. These methods work synergistically to provide a complete picture of the data's characteristics, helping analysts determine if the data is normally distributed, skewed, or contains problematic values.

We categorize these standard univariate approaches into three distinct areas of study:

Summary Statistics: This involves calculating descriptive statistics--the measures that quantify the center, position, and dispersion (spread) of the values within the dataset.

Frequency Table: This method systematically describes how often each distinct value or value range occurs within the variable, providing a direct count of observations.

Charts and Visualization: This crucial step uses graphical tools to visually represent the distribution of values, allowing for quick identification of shape, symmetry, and the presence of outliers.

Setting Up the Python Environment and Data

Before diving into the analysis, we must first establish a working environment and load the sample data. For statistical computation and manipulation, the [Pandas](#) library is indispensable, as it provides high-performance, easy-to-use data structures, most notably the [DataFrame](#). Our example utilizes a simple dataset containing fictional performance metrics (points, assists, and rebounds) for a small group of players.

This tutorial will specifically use the 'points' column as our target variable for univariate analysis. It is essential to inspect the initial structure of the data to confirm proper loading and formatting before applying statistical methods. This preliminary step ensures that subsequent calculations are based on clean and correctly structured data elements.

The following code snippet demonstrates the creation and inspection of the sample [DataFrame](#) in Python:

```
import pandas as pd
```

```
#create DataFrame
```

```
df = pd.DataFrame({'points': ,  
'assists': ,  
'rebounds': })
```

```
#view first five rows of DataFrame
```

```
df.head()
```

```
points assists rebounds
```

```
0 1.0 5 11
```

```
1 1.0 7 8
```

```
2 2.0 7 10
```

```
3 3.5 9 6
```

```
4 4.0 12 6
```

Method 1: Calculating Descriptive Statistics

The first and most critical component of univariate analysis involves calculating [descriptive statistics](#). These numerical summaries provide quantitative insights into the typical value (central tendency) and the spread of the data (dispersion). Understanding these metrics helps determine the overall characteristics of the variable, such as whether it is clustered tightly around the average or spread widely across a range of values.

We will focus on the most commonly used statistics: the mean, the median, and the standard deviation. The **mean** (average) is sensitive to outliers and provides the central balancing point of the distribution. The **median** represents the 50th percentile, acting as a robust measure of the center that is unaffected by extreme values. Comparing the mean and median can often reveal skewness in the data distribution.

Furthermore, the **standard deviation** is essential as it measures the average amount of variability or dispersion from the mean. A small standard deviation indicates that the data points tend to be close to the mean, while a high standard deviation suggests the data points are spread out over a wider range. Using the [Pandas](#) library, these calculations are straightforward using built-in methods applied directly to the series (column) of interest, in this case, 'points'.

Below is the syntax used to compute the mean, median, and standard deviation for the 'points' variable:

```
#calculate mean of 'points'
```

```
df.mean()
```

```
5.706666666666667
```

```
#calculate median of 'points'
```

```
df.median()
```

```
5.0
```

```
#calculate standard deviation of 'points'
```

```
df.std()
```

```
3.858287308169384
```

Method 2: Generating Frequency Distributions

While descriptive statistics give us a general sense of the center and spread, a frequency distribution provides a granular look at how observations are distributed across specific values. A frequency table lists all unique values in the dataset and counts how many times each value appears. This is especially useful for categorical or discrete numerical data, although it can be applied to continuous data as well, often after binning the values into ranges.

For discrete variables like the 'points' column in our example, a frequency table immediately highlights the most and least common scores. The most frequently occurring value, known as the mode, can be easily identified from this table. In data analysis, understanding the frequency distribution is crucial for identifying modes (peaks) in the data and spotting potential data entry

errors or unusual concentrations of scores.

In Pandas, the `.value_counts()` method is the most efficient way to generate this distribution. It returns a series containing counts of unique values in descending order, making it simple to determine which values dominate the dataset.

We use the following syntax to create a frequency table for the 'points' variable:

```
#create frequency table for 'points'
```

```
df.value_counts()
```

```
4.0 3
```

```
1.0 2
```

```
5.0 2
```

```
2.0 1
```

```
3.5 1
```

```
6.5 1
```

```
7.0 1
```

```
7.4 1
```

```
8.0 1
```

```
13.0 1
```

```
14.2 1
```

```
Name: points, dtype: int64
```

The generated table clearly shows the count for each unique score observed in the dataset. This immediate visualization of frequency allows us to draw specific conclusions about the data's composition:

The value **4.0** occurs 3 times, making it the mode of the dataset.

The values **1.0** and **5.0** both occur 2 times.

The remaining values, such as **2.0**, **3.5**, **6.5**, **7.0**, **7.4**, **8.0**, **13.0**, and **14.2**, each occur only 1 time.

This level of detail confirms that while 4.0 is the most common score, many scores are unique or nearly unique, suggesting a wide spread, which aligns with the moderately high standard deviation calculated earlier.

Method 3: Visualizing Data Distribution with Charts

While numerical summaries are informative, visualizing the data distribution is often the fastest and most intuitive way to understand a variable's characteristics. Charts are invaluable in identifying the shape of the distribution, detecting skewness, and spotting potential outliers that might skew the

mean or standard deviation. We will explore three crucial visualization types for continuous data: the box plot, the histogram, and the Kernel Density Estimate (KDE) plot.

All visualizations in this section rely on Python's powerful plotting libraries, primarily Matplotlib (often accessed implicitly via Pandas plotting methods) and Seaborn, which is built on Matplotlib but provides a higher-level, more aesthetic interface for statistical graphics.

3a. Utilizing Box Plots for Spread and Outlier Detection

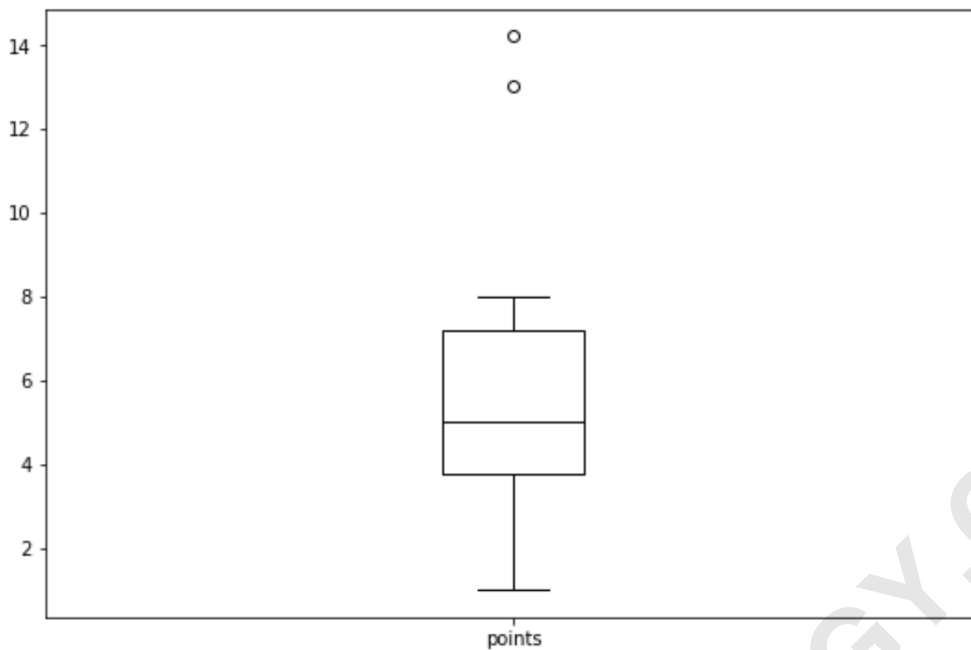
A box plot, or box-and-whisker plot, is a standardized way of displaying the distribution of data based on a five-number summary: the minimum, the first quartile (Q1), the median (Q2), the third quartile (Q3), and the maximum. It is particularly effective for visualizing the spread and identifying potential outliers--values that lie far outside the central body of the data.

The box itself represents the Interquartile Range (IQR), encompassing the middle 50% of the data. The line inside the box marks the median. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, and any data points lying outside the whiskers are considered outliers and plotted individually. This concise visual summary makes the box plot an excellent tool for comparing distributions across different groups, though here we use it to characterize a single variable.

We use the DataFrame's built-in plotting method (which utilizes Matplotlib) to generate the box plot for the 'points' variable:

```
import matplotlib.pyplot as plt
```

```
df.boxplot(column=, grid=False, color='black')
```



The resulting box plot visually confirms the central tendency and spread we calculated previously. We can see the median line near 5.0, and the long upper whisker, combined with the presence of higher points outside the main box, suggests that the distribution is slightly positively skewed (tailing off to the right).

3b. Histograms for Understanding Distribution Shape

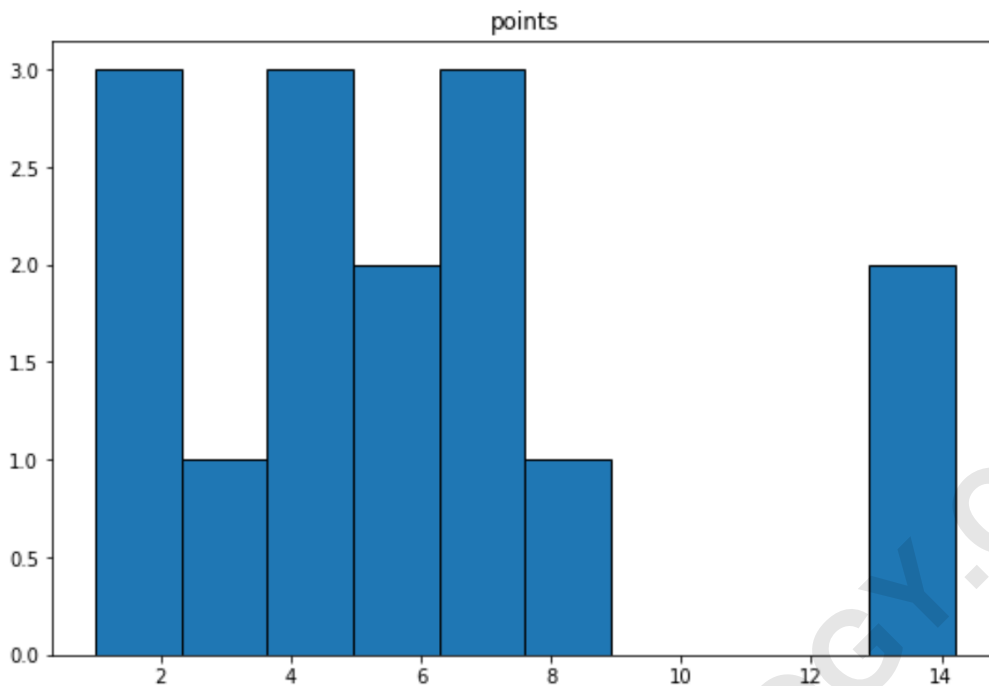
The histogram is arguably the most common graphical tool for univariate analysis of continuous variables. It approximates the probability distribution of a continuous variable by dividing the entire range of values into a series of intervals (bins) and counting how many data points fall into each interval. The height of each bar represents the frequency (or count) of observations within that bin.

Histograms are crucial for quickly assessing the shape of the distribution--is it symmetric (like a normal distribution), or is it skewed left or right? Does it have one peak (unimodal) or multiple peaks (bimodal or multimodal)? These visual cues are essential for choosing appropriate statistical tests and models later in the analysis pipeline.

We generate the histogram for the 'points' variable using the Pandas `.hist()` method:

```
import matplotlib.pyplot as plt
```

```
df.hist(column='points', grid=False, edgecolor='black')
```



The histogram confirms the positive skewness observed in the box plot. The majority of the observations are clustered toward the lower end of the scores (0 to 8), with a noticeable tail extending toward the higher scores (13 and 14.2). This visual confirmation is significantly more impactful than relying solely on the numerical difference between the mean (5.71) and the median (5.0).

3c. Kernel Density Estimates (KDE) for Smoothed Distributions

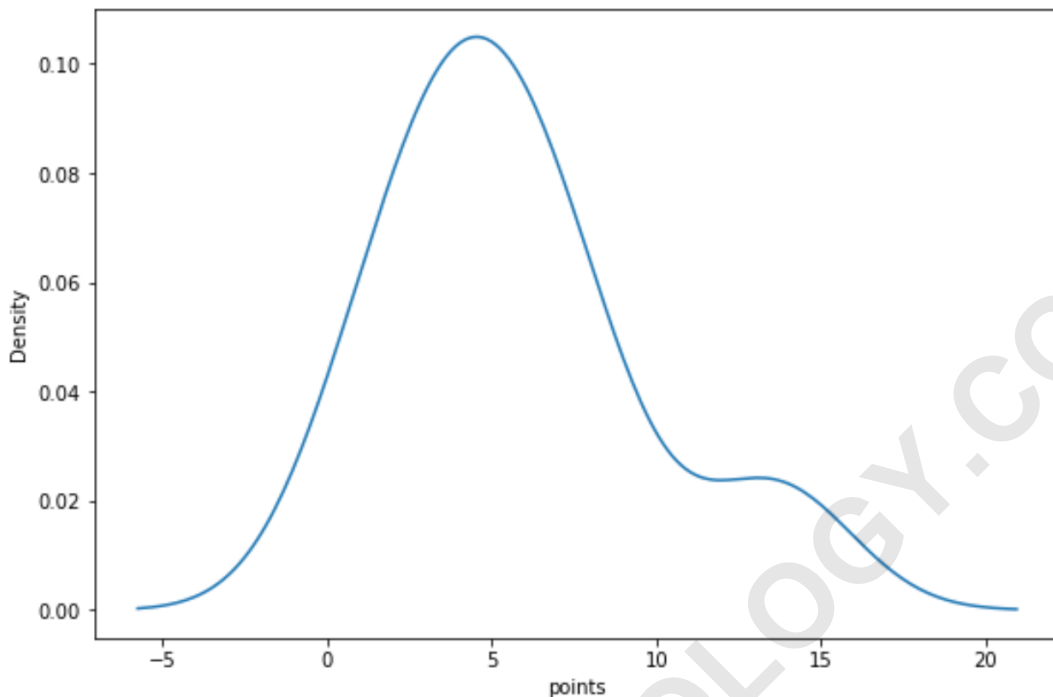
While histograms show discrete bins, the Kernel Density Estimate (KDE) plot provides a non-parametric way to estimate the probability density function of a continuous variable. Essentially, it smooths out the distribution, offering a continuous curve that visually represents the variable's density. KDE plots are excellent for identifying the underlying shape without the influence of bin size, which can sometimes artificially affect the appearance of a histogram.

KDE plots are particularly useful when comparing the distribution of one variable against theoretical distributions, or when comparing the shape of the variable across different subgroups. By using the [Seaborn](#) library, we can easily generate a high-quality, smoothed density curve for the 'points' variable, providing a final, refined visual summary of its distribution.

We utilize the [Seaborn](#) `kdeplot` function to create this smoothed visualization:

```
import seaborn as sns
```

```
sns.kdeplot(df)
```



The KDE plot clearly illustrates the distribution density. The curve shows a high concentration of data points between 0 and 8, with the highest peak density occurring just below 5.0. The long tail trailing off to the right (positive skew) is visually striking, confirming that there are several observations with much higher values than the majority of the data.

Each of these charts--the box plot, the histogram, and the KDE plot--provides a unique and complementary perspective on the distribution of values for the 'points' variable. Together with the calculated numerical statistics, they form a complete and robust univariate analysis, preparing the data scientist for deeper, more complex modeling tasks.