Learning Percentage Change Calculation with Pandas: A Step-by-Step Guide

Authored by

Mohammed looti

November 13, 2025

When conducting thorough analysis of quantitative datasets, particularly those involving sequential observations such as time-series data or financial metrics, the calculation of proportional change between data points is fundamental. This calculation, commonly referred to as the percentage change, is indispensable for accurately assessing metrics like growth rates, underlying volatility, and overall performance trends across defined intervals.

For data scientists and analysts utilizing Python, the Pandas data analysis library provides an optimized and highly efficient method to execute this calculation: the built-in Series and DataFrame function, **pct_change()**. This powerful function abstracts the necessity of manually implementing complex shifting and division operations, simplifying the process of deriving relative differences across a dataset.

Understanding the Syntax and Core Parameters

The **pct_change()** function is specifically designed to calculate the relative difference between the current element and a preceding element within a data structure. Mastering its fundamental syntax is the first step toward leveraging its capabilities in any analytical workflow. The core signature of the function, when applied to a Pandas DataFrame, is straightforward:

DataFrame.pct_change(periods=1, fill_method='pad', limit=None, freq=None, **kwargs)

While the function offers several arguments for nuanced control, the single most critical parameter for defining the calculation window is the `periods` argument. This parameter dictates how far back in the sequence the comparison should be made.

periods: This mandatory integer specifies the number of shifts, or time periods, that will be used as the denominator in the percentage change calculation.

By default, the value for `periods` is set to **1**. This standard configuration ensures that Pandas calculates the percentage change between the current row's value and the value immediately preceding it. This is the required behavior for analyzing consecutive, period-over-period growth, such as calculating day-to-day stock returns or month-over-month sales figures. If your analytical needs require a comparison against an observation further in the past--for instance, calculating year-over-year change in quarterly data, or comparing a stock price to its value five days prior--the `periods` parameter can be easily adjusted to any positive or negative integer to establish the required lag.

Practical Application: Calculating Consecutive Period Changes

To fully grasp how the **pct_change()** function operates, we will utilize a practical example. Let us begin by constructing a sample DataFrame that simulates sequential financial data across eight

distinct periods, tracking two key metrics: total sales and associated refunds. This setup allows us to demonstrate how proportional change is calculated on a column of quantitative data.

import pandas as pd

#create DataFrame simulating sequential retail data
df = pd.DataFrame({'period': ,
'sales': ,
'refunds': })

#view DataFrame structure
print(df)

period sales refunds
0 1 122 10
1 2 140 22
2 3 188 24
3 4 134 20
4 5 199 14
5 6 215 18
6 7 200 10
7 8 249 12

Our primary objective is to calculate the **percentage change** in sales between each consecutive period, thereby measuring the proportional growth or decline relative to the immediate preceding observation. Since we are focused on consecutive changes, we rely entirely on the default configuration where `periods=1`. We apply the **pct_change()** function directly to the `sales` column and assign the resulting values to a new column named `sales_change` for easy comparison.

#calculate percent change between each period in 'sales' column using default periods=1
df = df.pct_change()

#view updated DataFrame with calculated changes
print(df)

period sales refunds sales_change
0 1 122 10 NaN
1 2 140 22 0.147541
2 3 188 24 0.342857
3 4 134 20 -0.287234
4 5 199 14 0.485075

5 6 215 18 0.080402
6 7 200 10 -0.069767
7 8 249 12 0.245000

The output demonstrates that the `sales_change` column now holds the proportional change expressed as a decimal value. For instance, the value in row 1 (corresponding to Period 2) reflects the change from Period 1 to Period 2, calculated as (140 - 122) / 122, which yields approximately **0.1475**, representing 14.75% growth. Conversely, the value in row 3 (Period 4) shows a sharp decline, calculated as (134 - 188) / 188, resulting in **-0.2872**, indicating a 28.72% decrease in sales between Period 3 and Period 4. This simple, single-line function allows for remarkably rapid analysis of sequential performance metrics.

A critical observation is the first row of the `sales_change` column, which displays **NaN** (Not a Number). This result is expected behavior in rolling window and shifting calculations because there is no preceding observation for Period 1 against which to calculate a percentage change. The function automatically handles this boundary condition.

Formatting Results as True Percentages

While the default output provides the proportional change as a decimal--which is mathematically correct--many analysts, especially those preparing reports, prefer to see the results explicitly formatted as percentages. This transformation is exceptionally simple and is achieved by multiplying the result of the **pct_change()** function by 100 before assignment.

#calculate percent change and multiply by 100 for percentage representation
df = df.pct_change() * 100

#view updated DataFrame
print(df)

period sales refunds sales_change_percent
0 1 122 10 NaN
1 2 140 22 14.754098
2 3 188 24 34.285714
3 4 134 20 -28.723404
4 5 199 14 48.507463
5 6 215 18 8.040201
6 7 200 10 -6.976744
7 8 249 12 24.500000

With this minor adjustment, the values in the new column are clearly presented in terms of percentages, making the growth and decline figures immediately interpretable for reporting and dashboarding purposes without requiring further manual formatting or mental calculation.

Analyzing Non-Consecutive Periods using the periods Parameter

The true power and flexibility of the **pct_change()** function become apparent when the `periods` parameter is utilized to specify a non-consecutive comparison. Instead of always comparing the current observation to the immediate predecessor (using `periods=1`), we can set a larger lag, such as `periods=2`, to calculate the percentage change relative to the value that occurred two periods prior.

For example, to calculate the sales change comparing the current period to the period two steps before it, we simply pass the integer 2 to the function:

#calculate percent change between every 2 periods in 'sales' column
df = df.pct_change(periods=2)

#view updated DataFrame
print(df)

period sales refunds sales_change_2_period
0 1 122 10 NaN
1 2 140 22 NaN
2 3 188 24 0.540984
3 4 134 20 -0.042857
4 5 199 14 0.058511
5 6 215 18 0.604478
6 7 200 10 0.005025
7 8 249 12 0.158140

A key difference in this output is that the first two rows now contain **NaN** values. This is entirely logical because setting `periods=2` dictates that the calculation requires two prior data points. Periods 1 and 2 lack sufficient preceding data, thus the calculation cannot be executed for those initial rows. The resulting calculated values reflect the two-period lag: for Period 3 (row 2), the calculation uses the Period 1 sales figure: (188 - 122) / 122, yielding approximately **0.5409**, or 54.09% growth over the two periods. Similarly, the value for Period 4 (row 3) compares its sales to Period 2: (134 - 140) / 140, resulting in **-0.0428**, a decrease of 4.28%. This flexibility makes the function invaluable for seasonal adjustments and long-term comparisons.

Conclusion and Next Steps

The **pct_change()** function within the Pandas DataFrame structure provides a highly robust, clean, and concise methodology for calculating proportional differences in sequential data. Whether the analytical task involves calculating short-term stock returns, measuring sales growth across consecutive months, or comparing values over extended, non-consecutive intervals, this tool drastically simplifies potentially complex data manipulation tasks. By mastering the fundamental use and adjustment of the `periods` parameter, analysts can ensure that the calculated growth rate aligns precisely with the required analytical timeframe.

For users seeking to explore the full depth of capabilities and advanced use cases, including handling missing data (NaN) imputation methods, the complete official documentation for the **pct_change()** function offers comprehensive details and further parameters.

Featured Posts

5 Statistical Biases to Avoid

April 25, 2024

5 Free Statistics Courses for Beginners

April 19, 2024

5 MIT Statistics Courses That Are Free

April 18, 2024

5 Free Books to Learn Statistics

April 18, 2024

How to Use the info() Method in Pandas

April 12, 2024

How to Use pct_change() in Pandas

April 12, 2024