Extracting Week Numbers from Dates: A Pandas DataFrame Tutorial

Authored by

Mohammed looti

November 13, 2025

When conducting **time-series analysis** or generating reports based on cyclical data, data professionals often require the precise extraction of the **week number** from a date column stored within a **Pandas DataFrame**. This specific operation is fundamental for correctly grouping, aggregating, and visualizing data based on standardized weekly periods. Fortunately, the widely used **Pandas** library offers robust, highly optimized tools designed specifically for the efficient manipulation of date and time data.

This comprehensive guide will detail the primary methods utilized to accurately determine the week number. We will focus specifically on leveraging the powerful `.dt` accessor, combined with Python's standard **strftime** formatting codes. By exploring these two fundamental techniques, we cover the majority of requirements needed for effective weekly data analysis.

Understanding Date/Time Formatting in Pandas

Before attempting any date manipulation, it is essential to grasp the underlying mechanism **Pandas** employs. For these techniques to work, the target column must first be stored as a **datetime object**. Once the data type is correct, we gain access to the crucial `.dt` accessor. This accessor is the gateway that exposes various time-series components--such as the year, month, day, and, most importantly for this task, the ability to apply **strftime**.

The **strftime** function (which stands for String Format Time) is a standard utility inherited from Python's core library. Its purpose is to convert **datetime objects** into strings, formatted precisely according to specified directive codes. When extracting week numbers, the selection of the correct directive is critical, as different standards exist globally for designating the start of the week (e.g., Sunday vs. Monday) and defining which week constitutes "Week 1" of the year.

In the two methods detailed below, we will utilize the `%U` directive. This directive adheres to the common standard where **Sunday** is designated as the first day of the week, and Week 1 begins specifically with the first Sunday of the calendar year.

Method 1: Extracting the Simple Sequential Week Number (%U Directive)

The most straightforward approach for retrieving the week number involves applying the `%U` format code directly. This method is highly suitable for scenarios where all data points fall within the same calendar year, or where the explicit context of the year is not necessary for the immediate analysis (such as comparing weekly performance within a single year).

The general syntax is concise and powerful: select the date column, apply the `.dt` accessor, and then execute the **strftime** method using the required format code. This operation transforms the date object into a string representation of the week.

df = df.dt.strftime('%U')

Executing this line of code efficiently creates a new column, typically named `week_num`, that holds the corresponding week number for every date entry in the source column. The resulting output is a string displayed as a two-digit decimal number (e.g., '01', '15'), representing the sequence of the week within the year based on the `%U` standard.

Important Context: Understanding the %U Calculation Standard

It is critical for analysts to fully understand the specific logic governing the `%U` directive, as misunderstanding this standard can lead to miscalculations during weekly data aggregation. The `%U` format code follows a very precise set of rules for week calculation:

The start of the week is strictly designated as **Sunday**.
Week 01 is defined as the week that contains the **first Sunday** of that specific calendar year.
Any days falling in January, starting from the 1st, but occurring before the first Sunday are grouped into **Week 00**.

If your analytical requirements dictate adherence to the international standard, such as **ISO 8601** (where Monday is the first day and Week 1 must contain January 4th), you should instead investigate the `%W` or `%V` directives. However, for demonstrating the standard Python implementation readily available through Pandas' date utilities, `%U` remains the most common and simplest method for calculating sequential week numbers.

Method 2: Including the Year for Unique Identification (%Y-%U)

When working with a **DataFrame** that spans across multiple years, relying solely on the week number (1 through 53) is insufficient to uniquely identify a specific weekly period. For example, Week 10 in 2024 is entirely distinct from Week 10 in 2025. To ensure unique and unambiguous weekly identifiers across longitudinal time periods, we must combine the year with the week number.

This is accomplished by combining the `%Y` directive (which represents the four-digit year) and the `%U` directive (for the week number), joined together by a suitable delimiter, such as a hyphen. This concatenation creates a single, highly effective composite key.

df = df.dt.strftime('%Y-%U')

This enhanced method generates a new column (e.g., `week_num`) containing the year followed by the zero-padded week number (e.g., "2024-04"). This unique composite key ensures that every

single weekly period, irrespective of the year boundary, possesses a unique label. This reliability is paramount for aggregation and comparison across multi-year datasets, especially useful in financial reporting or longitudinal trend analysis.

The following practical examples illustrate how to implement both techniques effectively in a real-world context using simulated sales data.

Practical Application: Detailed Example 1 (Simple Week Number)

We begin by constructing a sample **Pandas DataFrame**. This dataset contains hypothetical sales figures recorded bi-weekly throughout the early months of 2024. This setup provides a clear, practical environment for demonstrating the extraction of the simple sequential week number using the `%U` directive.

import pandas as pd

#create DataFrame
df = pd.DataFrame({'date': pd.date_range('1/1/2024', periods=10, freq='2W'),
'sales': })

#view DataFrame
print(df)

date sales
0 2024-01-07 2
1 2024-01-21 5
2 2024-02-04 5
3 2024-02-18 4
4 2024-03-03 7
5 2024-03-17 8
6 2024-03-31 9
7 2024-04-14 12
8 2024-04-28 10
9 2024-05-12 14

Our objective is now to accurately isolate the week number corresponding to each transaction date listed in the `date` column. We will apply the `.dt.strftime('%U')` method, as detailed in Method 1.

Applying the syntax shown below performs the required data type conversion and appends the results as a new feature column to our existing DataFrame:

#create new column that contains week number of date
df = df.dt.strftime('%U')

#view updated DataFrame
print(df)

date sales week_num
0 2024-01-07 2 01
1 2024-01-21 5 03
2 2024-02-04 5 05
3 2024-02-18 4 07
4 2024-03-03 7 09
5 2024-03-17 8 11
6 2024-03-31 9 13
7 2024-04-14 12 15
8 2024-04-28 10 17
9 2024-05-12 14 19

The resulting `week_num` column successfully displays the sequential week number within the year 2024 for every date record. This structure allows for straightforward and accurate weekly aggregation of the sales data.

The date **2024-01-07** falls within **Week 1** of the year, adhering to the Sunday-start calculation standard.
The date **2024-01-21** corresponds to **Week 3**.
The date **2024-02-04** is correctly assigned **Week 5**.
The date **2024-02-18** is designated **Week 7**.

This outcome clearly demonstrates the utility of the `%U` directive for standard weekly reporting purposes. Always remember the underlying calculation rules: Sunday is the first day, and Week 0 is reserved for any days preceding the first Sunday of January.

Practical Application: Detailed Example 2 (Year and Week Number)

For this second detailed example, we construct a **Pandas DataFrame** whose dates intentionally span across two separate calendar years (2024 and 2025). This scenario explicitly requires the inclusion of the year component to prevent conflating weekly data points originating from different annual periods.

import pandas as pd

```
#create DataFrame
df = pd.DataFrame({'date': pd.date_range('1/1/2024', periods=10, freq='2M'),
'sales': })

#view DataFrame
print(df)
```

```
date sales
0 2024-01-31 2
1 2024-03-31 5
2 2024-05-31 5
3 2024-07-31 4
4 2024-09-30 7
5 2024-11-30 8
6 2025-01-31 9
7 2025-03-31 12
8 2025-05-31 10
9 2025-07-31 14
```

Our objective here is to extract a combined, unique identifier that incorporates both the year and the week number for every entry in the `date` column. This composite identifier is essential for accurate, long-term time-series analysis across the entire multi-year period.

We utilize the combined format string `'%Y-%U'` within the `.dt.strftime()` method to achieve this precise labeling:

```
#create new column that contains week number and year of date
df = df.dt.strftime('%Y-%U')

#view updated DataFrame
print(df)
```

```
date sales week_num
0 2024-01-31 2 2024-04
1 2024-03-31 5 2024-13
2 2024-05-31 5 2024-21
3 2024-07-31 4 2024-30
4 2024-09-30 7 2024-39
5 2024-11-30 8 2024-47
6 2025-01-31 9 2025-04
7 2025-03-31 12 2025-13
```

8 2025-05-31 10 2025-21
9 2025-07-31 14 2025-30

The final `week_num` column successfully provides clear and distinct labels for every data point, completely resolving any potential ambiguity across the two years present in the dataset. Note carefully how the sequential week numbers reset in the 2025 entries, but the year prefix guarantees chronological distinction and accuracy.

A quick analysis of the output confirms the successful application of the combined formatting:

The date **2024-01-31** is uniquely identified as **Week 4** in the year **2024** (`2024-04`).
The date **2024-03-31** corresponds to **Week 13** in **2024** (`2024-13`).
The date **2025-01-31** is identified as **Week 4** in the year **2025** (`2025-04`), which clearly and successfully separates it from the data point corresponding to the same sequential week in the preceding year.

This powerful formatting technique is indispensable for data analysts performing year-over-year comparisons or running complex weekly reports that span multiple fiscal cycles or extended time frames.

Additional Resources for Time Series Analysis

Mastering date and time manipulation is a fundamental skill set required for any data scientist or analyst working extensively with **Pandas**. The following related tutorials provide further guidance on essential data processing tasks:

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