How to Easily Drop Columns Containing Specific Strings in Pandas

Authored by

stats writer

November 20, 2025

Data manipulation is a foundational skill in the field of data science, and the pandas library is the undisputed champion for handling structured data in Python. One common yet crucial task involves cleaning datasets by removing irrelevant or redundant columns. While dropping columns by exact name is straightforward, real-world data often requires more sophisticated techniques, such as identifying and dropping columns based on whether their names contain a specific substring or pattern.

This tutorial details the expert methodology for efficiently purging columns from a DataFrame using string matching and regular expressions (regex). We will leverage the powerful combination of the `DataFrame.filter()` method, which is superb for selecting labels based on patterns, and the `DataFrame.drop()` method, which handles the final removal. Mastering this technique ensures your data preparation pipeline is robust, scalable, and highly adaptable to dynamic naming conventions.

Although other methods, like boolean indexing combined with list comprehensions, exist for selecting columns, the approach using `.filter()` with the `regex` parameter is generally considered the most idiomatic and readable solution in pandas when dealing with name patterns. This method abstracts away complex looping and provides a direct, declarative way to target columns based on textual content.

The Strategy: Combining Filtering and Dropping

To selectively drop columns whose names match a certain pattern, we employ a two-step process that is both elegant and efficient. First, we need to identify the target columns, and second, we need to execute the removal. The identification step is handled by `DataFrame.filter()`, which allows us to select column labels based on a regex pattern provided to its `regex` argument.

The `filter()` method, when used with `regex`, returns a list of labels (column names) that satisfy the pattern match. Critically, we must ensure that the output of `filter()` is interpreted correctly by the subsequent step. Since `drop()` expects a list of column names to remove, we simply pass the list returned by the filter operation directly into the `DataFrame.drop()` function. This tight integration minimizes boilerplate code and streamlines the column management process significantly.

Understanding the role of the `regex` parameter is key. This parameter accepts a pattern that is applied against the index labels (which are the column names by default). If a column name contains the string defined in the pattern, it is selected. If multiple columns match, all their names are returned in a list, ready for deletion by the `drop()` function.

Method 1: Dropping Columns Containing a Single Specific String

The simplest application of this technique involves searching for a single, fixed substring within all column names. This is ideal when dealing with automatically generated metrics, temporary identifiers, or specific prefixes/suffixes that need to be eliminated from the final dataset.

You can use the following methods to drop columns from a pandas DataFrame whose name contains specific strings:

Method 1: Drop Columns if Name Contains Specific String

df.drop(list(df.filter(regex='this_string')), axis=1, inplace=True)

This single line of code is highly effective. The internal call to `df.filter(regex='this_string')` returns the column names matching the pattern. Wrapping this in `list()` ensures that the result is passed as an iterable list of column identifiers to the `df.drop()` method. The remaining parameters, `axis=1` and `inplace=True`, finalize the operation by specifying that we are targeting columns (axis 1) and modifying the DataFrame in place, respectively.

Method 2: Dropping Columns Containing One of Several Specific Strings

Data cleaning often requires the removal of columns based on multiple criteria simultaneously. For instance, you might want to remove all columns related to 'team' statistics or 'player' metadata. Using standard Python string methods would require multiple calls to `drop()` or complex list comprehensions, but pandas simplifies this through regex.

When working with regex, the pipe symbol (`|`) acts as a logical OR operator. By inserting `|` between distinct substrings, the filter operation will select any column name that contains *at least one* of the specified strings. This dramatically increases the power and flexibility of the filtering step, allowing comprehensive removal criteria in a single command.

Method 2: Drop Columns if Name Contains One of Several Specific Strings

df.drop(list(df.filter(regex='string1|string2|string3')), axis=1, inplace=True)

This technique is particularly useful in large datasets where column names might vary slightly but share common descriptive components. For example, using `regex='date|timestamp|id'` would target all columns related to temporal data or unique identifiers for removal, streamlining large-scale data schema modifications.

Setting Up the Demonstration DataFrame

To illustrate these methods in practice, we first need a sample DataFrame that contains columns with descriptive names suitable for pattern matching. We will create a simple dataset representing fictional sports team statistics, ensuring that some columns share a common substring, like "team" or "name."

The following examples show how to use each method in practice with the following pandas DataFrame:

import pandas as pd

#create DataFrame
df = pd.DataFrame({'team_name': ,
'team_location': ,
'player_name': ,
'points': })

#view DataFrame
print(df)

team_name team_location player_name points
0 A AU Andy 22
1 B AU Bob 29
2 C EU Chad 35
3 D EU Dan 30
4 E AU Ed 18
5 F EU Fran 12

This initial DataFrame contains four columns: `team_name`, `team_location`, `player_name`, and `points`. Notice that two columns contain the string 'team' and two columns contain the string 'name'. This structure allows us to demonstrate both single and multiple pattern matching effectively.

Example 1: Dropping Columns Containing 'team'

In this first demonstration, our goal is to eliminate all columns that contain the substring `team`. This operation would effectively remove the contextual team identifiers, leaving only player-specific details and scores. We apply Method 1, ensuring the filter() function uses `regex='team'`.

The execution of this command is instantaneous, even on much larger datasets, demonstrating the

efficiency of the vectorized operations within pandas. By setting the pattern to a simple substring, we instruct the regex engine to look for that string anywhere within the column label.

Example 1: Drop Columns if Name Contains Specific String

We can use the following syntax to drop all columns in the DataFrame that contain 'team' anywhere in the column name:

```
#drop columns whose name contains 'team'
df.drop(list(df.filter(regex='team')), axis=1, inplace=True)

#view updated DataFrame
print(df)

  player_name  points
0         Andy      22
1          Bob      29
2         Chad      35
3          Dan      30
4           Ed      18
5         Fran      12
```

Notice that both columns that contained 'team' in the name (`team_name` and `team_location`) have been successfully dropped from the DataFrame. The resulting DataFrame now consists solely of `player_name` and `points`, achieving our targeted cleaning objective.

Example 2: Dropping Columns Containing 'player' OR 'points'

Now, let us tackle a more complex scenario using the logical OR operator (`|`). Suppose we want to keep only the geographical or structural team information and remove all columns related to individual performance metrics. This means dropping columns that contain either the string 'player' or the string 'points'.

This scenario highlights the power of regex in data cleaning. Instead of executing two separate drop operations, the single pattern `player|points` efficiently identifies both sets of target columns. The `filter()` method returns the union of columns matching either criterion, which is then passed to `drop()`.

Example 2: Drop Columns if Name Contains One of Several Specific Strings

We can use the following syntax to drop all columns in the DataFrame that contain 'player' or

'points' anywhere in the column name:

#drop columns whose name contains 'player' or 'points'
df.drop(list(df.filter(regex='player|points')), axis=1, inplace=True)

#view updated DataFrame
print(df)

team_name team_location
0 A AU
1 B AU
2 C EU
3 D EU
4 E AU
5 F EU

Notice that both columns that contained either 'player' or 'points' in the name (`player_name` and `points`) have been dropped from the DataFrame. The resulting structure now retains only `team_name` and `team_location`. This confirms the successful application of the OR logic within the regex pattern.

Understanding the Key Parameters: axis and inplace

The success of the `drop()` method relies heavily on two critical parameters: `axis` and `inplace`. Failing to correctly specify these can lead to errors or, worse, unintended modifications to the dataset.

The `axis` parameter dictates whether the operation should target rows or columns. In pandas, `axis=0` refers to the index (rows), and `axis=1` refers to the columns. Since we are providing a list of column names derived from the filter step, we must explicitly set `axis=1`. If this parameter were omitted or incorrectly set to 0, pandas would attempt to drop rows whose indices match the column names, resulting in a `KeyError` or an incorrect operation.

The `inplace=True` parameter controls whether the operation modifies the original DataFrame directly. When set to `True`, the changes are applied immediately, and the function returns `None`. If `inplace` is set to `False` (the default), the function returns a new DataFrame with the dropped columns, leaving the original DataFrame untouched. For memory efficiency in large processing pipelines, using `inplace=True` is common, but caution is advised as it permanently alters the data object.

Note: The **|** symbol in pandas regex is used as an "OR" operator, crucial for selecting labels that

match any one of several defined patterns.