Understanding and Resolving Python’s TypeError: Subtracting Strings and Integers

Authored by

Mohammed looti

October 30, 2025

One of the most common exceptions encountered when performing data manipulation or mathematical operations in Python, particularly within the pandas DataFrame environment, is the TypeError. Specifically, developers often encounter the message:

TypeError: unsupported operand type(s) for -: 'str' and 'int'

This critical error arises when the subtraction operator (`-`) is applied between two variables that Python considers incompatible for that operation: a string (`'str'`) and an integer (`'int'`). Python, as a strongly typed language, demands that mathematical operations only occur between compatible data types, usually numerical types. When it detects an attempt to subtract textual data from numerical data, it immediately throws this exception to prevent logical corruption of the result.

Understanding the root cause--the mismatch of data types--is the first step toward resolving this issue. This comprehensive guide will demonstrate precisely why this error occurs, illustrate its reproduction using a practical pandas DataFrame example, and provide the robust, standard solution necessary to ensure successful, numerically accurate computations.

Understanding Type Coercion and the TypeError

To fully appreciate why Python raises the `TypeError: unsupported operand type(s)`, we must examine how Python handles types during runtime. Python is dynamically typed, meaning you do not explicitly declare variable types, but it is also strongly typed, meaning types are strictly enforced during operations. When you attempt to subtract a value of type string from a value of type integer, Python lacks an intrinsic mechanism for performing this conversion automatically, unlike some weakly typed languages.

The core problem is that the subtraction operator (`__sub__` method in Python's data model) is defined only for numerical operands (like `int`, `float`, etc.). A string represents text, even if that text happens to look like a number (e.g., `'18'`). Python cannot assume that the user intended for `'18'` to be treated as the number 18; this conversion must be explicit. Attempting to mix these incompatible types triggers the TypeError, signaling that the operation definition does not support the given combination of operand types.

This strict type checking is a vital feature of Python, designed to prevent silent errors where mathematical operations yield meaningless results. For instance, if Python were allowed to treat `'18'` as 18 implicitly, it might lead to unpredictable behavior, especially if the string contained non-numeric characters later in the dataset. Therefore, the solution always involves explicitly converting the non-numeric data into a recognized numeric variable type before attempting the calculation.

How to Reproduce the Error

To observe this error in a real-world data analysis context, let us utilize the widely adopted pandas DataFrame library. We will create a sample dataset intended to track sports scores, where we want to calculate the point differential (points for minus points against).

In the construction of the DataFrame below, note how the `points_for` column is intentionally initialized using strings (indicated by the quotes around the values), while `points_against` is initialized using standard integers. This initial structure is where the problem originates, often stemming from importing data from files like CSVs or Excel spreadsheets where fields are misinterpreted as text.

import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ,
'points_for': ,
'points_against': })

#view DataFrame
print(df)

team points_for points_against
0 A 18 5
1 B 22 7
2 C 19 17
3 D 14 22
4 E 14 12
5 F 11 9
6 G 20 9
7 H 28 4

#view data type of each column
print(df.dtypes)

team object
points_for object
points_against int64
dtype: object

After inspecting the output of `df.dtypes`, it is explicitly clear that the `points_for` column is

classified as `object`, which in pandas typically signifies a string or mixed data type, while `points_against` is correctly recognized as `int64`. Now, when we attempt the subtraction operation, the conflict becomes unavoidable.

We now attempt to calculate the difference by subtracting the `points_against` column from the `points_for` column, expecting a new column named `diff`. This action directly triggers the type mismatch error, as illustrated in the resulting traceback below.

#attempt to perform subtraction
df = df.points_for - df.points_against

TypeError: unsupported operand type(s) for -: 'str' and 'int'

The traceback confirms that the operation failed because the first operand (`points_for`) was interpreted as a string (`'str'`) and the second (`points_against`) was an integer (`'int'`). Since no defined subtraction method exists for this specific combination of types, the Python interpreter halts execution and raises the TypeError.

Diagnosing Data Types in Pandas

Effective data cleaning and preparation hinge on accurately diagnosing the data types within your pandas DataFrame. The `.dtypes` attribute is the most crucial tool for this diagnosis. In the previous section, we observed that `points_for` was classified as `object`. While `object` often implies text, it is also the catch-all category for columns containing mixed types, which can include a combination of strings and integers, or even null values that prevent uniform numerical assignment.

When preparing data for mathematical computation, the goal must be to ensure that all operands are explicit numeric variables--either `int64`, `float64`, or related numerical types. If a column that should contain numbers is listed as `object`, it is a strong indicator that explicit type conversion is required before any arithmetic operation can proceed safely. Failure to perform this verification step is the leading cause of this specific TypeError.

Furthermore, simply knowing the type is not enough; one must also verify the content. Before converting an `object` column to a numeric variable using methods like `.astype(int)`, it is essential to check for non-numeric entries (e.g., currency symbols, commas, or textual errors like 'N/A'). If invalid characters are present, the conversion attempt will fail with a `ValueError`, highlighting the need for comprehensive data cleaning prior to type casting.

How to Fix the Error

The resolution for the `TypeError: unsupported operand type(s) for -: 'str' and 'int'` is straightforward: we must explicitly change the data type of the string column (`points_for`) into a numerical type compatible with subtraction. In this scenario, since the values represent whole numbers, converting the column to an integer is the appropriate action.

Pandas provides the highly efficient `.astype()` method specifically for performing these explicit type conversions across entire Series or DataFrames. By applying `.astype(int)` to the `points_for` column, we instruct pandas to interpret the string representations (like `'18'`) as true numeric variables (like 18), thereby resolving the type conflict.

Observe the corrected sequence of operations below. First, we perform the type conversion. Second, we execute the subtraction operation, which now runs successfully because both operands (`points_for` and `points_against`) are recognized as numerical data types. Finally, we verify the results and the resulting data types.

#convert points_for column to integer
df = df.astype(int)

#perform subtraction
df = df.points_for - df.points_against

#view updated DataFrame
print(df)

team points_for points_against diff
0 A 18 5 13
1 B 22 7 15
2 C 19 17 2
3 D 14 22 -8
4 E 14 12 2
5 F 11 9 2
6 G 20 9 11
7 H 28 4 24

#view data type of each column
print(df.dtypes)

team object
points_for int32
points_against int64

diff int64
dtype: object

The successful output confirms that the calculated `diff` column was generated correctly. Crucially, the final `df.dtypes` output confirms that `points_for` is now an integer type (`int32`), resolving the conflict. It is worth noting that while the original `points_against` was `int64`, the conversion of `points_for` to `int32` still allows the operation to proceed because both are numerical types, and pandas handles the internal casting for arithmetic compatibility. We no longer receive a TypeError because both columns used for subtraction are now compatible numeric variables.

Best Practices for Data Type Management

To minimize the occurrence of this and related type errors, proactive data type management is essential, especially when dealing with external data sources like CSVs or databases. The most fundamental best practice is to always inspect the data types immediately upon loading data into a pandas DataFrame using the `.dtypes` attribute. This simple step can preempt hours of debugging type-related errors downstream in your analytical pipeline.

Furthermore, when loading data via functions such as `pd.read_csv()`, consider using the `dtype` parameter to explicitly define the expected data types for specific columns right at the moment of ingestion. For instance, if you know `points_for` should be an integer, specifying `{'points_for': int}` during the load process forces pandas to attempt the conversion immediately, often leading to cleaner data loading and immediate detection of problematic non-numeric variable entries.

Finally, whenever converting string columns to numeric variables, utilize the `pd.to_numeric()` function instead of `.astype()`, particularly when dealing with potentially dirty data. `pd.to_numeric()` offers the `errors='coerce'` argument, which is incredibly useful as it replaces any values that cannot be converted to a number with `NaN` (Not a Number). This approach allows the conversion to succeed without crashing, enabling subsequent handling of the missing data (e.g., imputation or removal) rather than immediately hitting a `ValueError` or TypeError.

Summary of Solution Steps

To concisely summarize the necessary steps for resolving the `TypeError: unsupported operand type(s) for -: 'str' and 'int'`, follow this ordered checklist:

Identify the columns involved in the mathematical operation (e.g., subtraction).
Use `DataFrame.dtypes` to confirm that one operand is a string or `object` type, and the other is a numeric variable.
If the column contains valid numerical data stored as strings, use `DataFrame =`

`DataFrame.astype(int)` (or `float`) to perform the explicit type casting.
Alternatively, use `pd.to_numeric(DataFrame, errors='coerce')` if data cleaning is required, followed by handling the resulting `NaN` values.
Rerun the mathematical operation (subtraction) only after both columns have been successfully converted to compatible numerical types.

Additional Resources

For further reading on related Python and Pandas error handling and data manipulation techniques, consider exploring the following comprehensive tutorials and documentation:

Official Pandas Documentation on Data Types (Dtypes).
Tutorials explaining how to fix other common errors in Python, such as `KeyError` or `AttributeError`.
Detailed guides on using the `object` datatype effectively in data cleaning pipelines.