Module-03, Python for Data Analysis

Data Preparation(String Manipulation)

Dostdar Ali
Instructor

Data science and Artificial Intelligence
3-Months Course
at
Karakaroum international Univrsity

January 11, 2024

Table of Contents

String Manipulation

- Python has long been a popular raw data manipulation language in part due to its ease of use for string and text processing.
- Most text operations are made simple with the string object's built-in methods.
- For more complex pattern matching and text manipulations, regular expressions may be needed. pandas adds to the mix by enabling you to apply string and regular expressions concisely on whole arrays of data.

String Manipulation

- Python has long been a popular raw data manipulation language in part due to its ease of use for string and text processing.
- Most text operations are made simple with the string object's built-in methods.
- For more complex pattern matching and text manipulations, regular expressions may be needed. pandas adds to the mix by enabling you to apply string and regular expressions concisely on whole arrays of data.

String Manipulation

- Python has long been a popular raw data manipulation language in part due to its ease of use for string and text processing.
- Most text operations are made simple with the string object's built-in methods.
- For more complex pattern matching and text manipulations, regular expressions may be needed. pandas adds to the mix by enabling you to apply string and regular expressions concisely on whole arrays of data.

String Object Methods

- In many string munging and scripting applications, built-in string methods are sufficient.
- As an example, a comma-separated string can be broken into pieces with split. Split is often combined with strip to trim whitespace (including line breaks).
- These substrings could be concatenated together with a two-colon delimiter using addition.
- But this isn't a practical generic method. A faster and more Pythonic way is to pass a list or tuple to the join method on the string '::'.
- Other methods are concerned with locating substrings. Using Python's in keyword is the best way to detect a substring, though index and find can also be used.
- Replace will substitute occurrences of one pattern for another. It is commonly used to delete patterns, too, by passing an empty strin

String Object Methods

- In many string munging and scripting applications, built-in string methods are sufficient.
- As an example, a comma-separated string can be broken into pieces with split. Split is often combined with strip to trim whitespace (including line breaks).
- These substrings could be concatenated together with a two-colon delimiter using addition.
- But this isn't a practical generic method. A faster and more Pythonic way is to pass a list or tuple to the join method on the string '::'.
- Other methods are concerned with locating substrings. Using Python's in keyword is the best way to detect a substring, though index and find can also be used.
- Replace will substitute occurrences of one pattern for another. It is commonly used to delete patterns, too, by passing an empty strin

String Object Methods

- In many string munging and scripting applications, built-in string methods are sufficient.
- As an example, a comma-separated string can be broken into pieces with split. Split is often combined with strip to trim whitespace (including line breaks).
- These substrings could be concatenated together with a two-colon delimiter using addition.
- But this isn't a practical generic method. A faster and more Pythonic way is to pass a list or tuple to the join method on the string '::'.
- Other methods are concerned with locating substrings. Using Python's in keyword is the best way to detect a substring, though index and find can also be used.
- Replace will substitute occurrences of one pattern for another. It is commonly used to delete patterns, too, by passing an empty strin

String Object Methods

- In many string munging and scripting applications, built-in string methods are sufficient.
- As an example, a comma-separated string can be broken into pieces with split. Split is often combined with strip to trim whitespace (including line breaks).
- These substrings could be concatenated together with a two-colon delimiter using addition.
- But this isn't a practical generic method. A faster and more Pythonic way is to pass a list or tuple to the join method on the string '::'.
- Other methods are concerned with locating substrings. Using Python's in keyword is the best way to detect a substring, though index and find can also be used.
- Replace will substitute occurrences of one pattern for another. It is commonly used to delete patterns, too, by passing an empty strin

String Object Methods

- In many string munging and scripting applications, built-in string methods are sufficient.
- As an example, a comma-separated string can be broken into pieces with split. Split is often combined with strip to trim whitespace (including line breaks).
- These substrings could be concatenated together with a two-colon delimiter using addition.
- But this isn't a practical generic method. A faster and more Pythonic way is to pass a list or tuple to the join method on the string '::'.
- Other methods are concerned with locating substrings. Using Python's in keyword is the best way to detect a substring, though index and find can also be used.
- Replace will substitute occurrences of one pattern for another. It is commonly used to delete patterns, too, by passing an empty strin

String Object Methods

- In many string munging and scripting applications, built-in string methods are sufficient.
- As an example, a comma-separated string can be broken into pieces with split. Split is often combined with strip to trim whitespace (including line breaks).
- These substrings could be concatenated together with a two-colon delimiter using addition.
- But this isn't a practical generic method. A faster and more Pythonic way is to pass a list or tuple to the join method on the string '::'.
- Other methods are concerned with locating substrings. Using Python's in keyword is the best way to detect a substring, though index and find can also be used.
- Replace will substitute occurrences of one pattern for another. It is commonly used to delete patterns, too, by passing an empty string.

String Object Methods

Argument	Description
`count`	Return the number of non-overlapping occurrences of substring in the string.
`endswith`	Returns `True` if string ends with suffix.
`startswith`	Returns `True` if string starts with prefix.
`join`	Use string as delimiter for concatenating a sequence of other strings.
`index`	Return position of first character in substring if found in the string; raises `ValueError` if not found.
`find`	Return position of first character of *first* occurrence of substring in the string; like `index`, but returns –1 if not found.
`rfind`	Return position of first character of *last* occurrence of substring in the string; returns –1 if not found.
`replace`	Replace occurrences of string with another string.
`strip`, `rstrip`, `lstrip`	Trim whitespace, including newlines; equivalent to `x.strip()` (and `rstrip`, `lstrip`, respectively) for each element.
`split`	Break string into list of substrings using passed delimiter.
`lower`	Convert alphabet characters to lowercase.
`upper`	Convert alphabet characters to uppercase.
`casefold`	Convert characters to lowercase, and convert any region-specific variable character combinations to a common comparable form.
`ljust`, `rjust`	Left justify or right justify, respectively; pad opposite side of string with spaces (or some other fill character) to return a string with a minimum width.

Regular Expressions

Regular expressions provide a flexible way to search or match (often more complex) string patterns in text. A single expression, commonly called a regex, is a string formed according to the regular expression language. Python's built-in re module is responsible for applying regular expressions to strings.

The re module functions fall into three categories: pattern matching, substitution, and splitting. Naturally these are all related; a regex describes a pattern to locate in the text, which can then be used for many purposes. Let's look at a simple example:

- Suppose we wanted to split a string with a variable number of whitespace characters (tabs, spaces, and newlines). The regex describing one or more whitespace characters is \s+.
- Match and search are closely related to findall. While findall returns all matches in a string, search returns only the first match. Let's consider a block of text and a regular expression capable of identifying most email addresses: text = """Dave dave@google.com Steve steve@gmail.com Rob rob@gmail.com Ryan ryan@yahoo.com """

Regular Expressions

Regular expressions provide a flexible way to search or match (often more complex) string patterns in text. A single expression, commonly called a regex, is a string formed according to the regular expression language. Python's built-in re module is responsible for applying regular expressions to strings.

The re module functions fall into three categories: pattern matching, substitution, and splitting. Naturally these are all related; a regex describes a pattern to locate in the text, which can then be used for many purposes. Let's look at a simple example:

- Suppose we wanted to split a string with a variable number of whitespace characters (tabs, spaces, and newlines). The regex describing one or more whitespace characters is \s+.
- Match and search are closely related to findall. While findall returns all matches in a string, search returns only the first match. Let's consider a block of text and a regular expression capable of identifying most email addresses: text = """Dave dave@google.com Steve steve@gmail.com Rob rob@gmail.com Ryan ryan@yahoo.com """

Regular expression Methods

- Regular expression Methods

Argument	Description
`findall`	Return all non-overlapping matching patterns in a string as a list
`finditer`	Like `findall`, but returns an iterator
`match`	Match pattern at start of string and optionally segment pattern components into groups; if the pattern matches, returns a match object, and otherwise `None`
`search`	Scan string for match to pattern; returning a match object if so; unlike `match`, the match can be anywhere in the string as opposed to only at the beginning
`split`	Break string into pieces at each occurrence of pattern
`sub, subn`	Replace all (`sub`) or first `n` occurrences (`subn`) of pattern in string with replacement expression; use symbols `\1, \2, ...` to refer to match group elements in the replacement string

Vectorized String Functions in pandas

- Cleaning up a messy dataset for analysis often requires a lot of string munging and regularization. To complicate matters, a column containing strings will sometimes have missing data.
- data = 'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com', 'Rob': 'rob@gmail.com', 'Wes': np.nan
- We can apply string and regular expression methods can be applied (passing a lambda or other function) to each value using data.map, but it will fail on the NA (null) values. To cope with this, Series has array-oriented methods for string operations that skip NA values. These are accessed through Series's str attribute; for example, we could check whether each email address has 'gmail' in it with str.contains.
- There are a couple of ways to do vectorized element retrieval. Either use str.get or index into the str attribute.

Vectorized String Functions in pandas

- Cleaning up a messy dataset for analysis often requires a lot of string munging and regularization. To complicate matters, a column containing strings will sometimes have missing data.
- data = 'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com', 'Rob': 'rob@gmail.com', 'Wes': np.nan
- We can apply string and regular expression methods can be applied (passing a lambda or other function) to each value using data.map, but it will fail on the NA (null) values. To cope with this, Series has array-oriented methods for string operations that skip NA values. These are accessed through Series's str attribute; for example, we could check whether each email address has 'gmail' in it with str.contains.
- There are a couple of ways to do vectorized element retrieval. Either use str.get or index into the str attribute.

Vectorized String Functions in pandas

- Cleaning up a messy dataset for analysis often requires a lot of string munging and regularization. To complicate matters, a column containing strings will sometimes have missing data.
- data = 'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com', 'Rob': 'rob@gmail.com', 'Wes': np.nan
- We can apply string and regular expression methods can be applied (passing a lambda or other function) to each value using data.map, but it will fail on the NA (null) values. To cope with this, Series has array-oriented methods for string operations that skip NA values. These are accessed through Series's str attribute; for example, we could check whether each email address has 'gmail' in it with str.contains.
- There are a couple of ways to do vectorized element retrieval. Either use str.get or index into the str attribute.

Partial listing of vectorized string methods

- Partial listing of vectorized string methods

Method	Description
`cat`	Concatenate strings element-wise with optional delimiter
`contains`	Return boolean array if each string contains pattern/regex
`count`	Count occurrences of pattern
`extract`	Use a regular expression with groups to extract one or more strings from a Series of strings; the result will be a DataFrame with one column per group
`endswith`	Equivalent to `x.endswith(pattern)` for each element
`startswith`	Equivalent to `x.startswith(pattern)` for each element
`findall`	Compute list of all occurrences of pattern/regex for each string
`get`	Index into each element (retrieve *i*-th element)
`isalnum`	Equivalent to built-in `str.alnum`
`isalpha`	Equivalent to built-in `str.isalpha`
`isdecimal`	Equivalent to built-in `str.isdecimal`
`isdigit`	Equivalent to built-in `str.isdigit`
`islower`	Equivalent to built-in `str.islower`
`isnumeric`	Equivalent to built-in `str.isnumeric`
`isupper`	Equivalent to built-in `str.isupper`
`join`	Join strings in each element of the Series with passed separator
`len`	Compute length of each string
`lower, upper`	Convert cases; equivalent to `x.lower()` or `x.upper()` for each element

Great Job
Thank you