Vectorized String Functions in Pandas

Many complex systems now-a-days deal with a large amount of data.

Processing such a large amount of data can be slow and time consuming. This is where vectorization comes into play.

A) What is Vectorization?

> - In programming and computer science, **vectorization is the process of applying operations to an entire set of values at once.**

> - **In other words, Vectorization is process of applying an operation on multiple items (in an array, Series or DataFrame) in one go.**

> - The vectorized version takes milliseconds to execute which is thousand times faster than applying Python function on every element one by one.

> Vectorization is used to speed up the Python code without using loop. Using such a function can help in minimizing the running time of code efficiently.

Vectorization in strings

Pandas provides a .str object on Series that lets you run various vectorized operations on strings.

B) String manipulation Functions

- Pandas provides a set of built-in string functions which make it easy to operate on string data. These are called String Manipulation Functions.

↳ *2 cells hidden*

Let's create a dataframe to learn to apply String Manipulation Functions:

```
import pandas as pd
a = {'name': ['Michael Smith', 'Ana Kors', 'Sean Bill', 'Carl Jonson', 'Bob Evan'],
                     'grade': [['A', 'A'], ['C'], ['A', 'C', 'B'], [], ['F']],
                     'age': ['19', '19', '17', '18', '-'],
                     'group': ['class 1', 'class 2', 'class 2', 'class 1', 'class 2'],
                     'suspended': [True, False, True, False, True]
                    }
df = pd.DataFrame(a)
df
```

	name	grade	age	group	suspended
0	Michael Smith	[A, A]	19	class 1	True
1	Ana Kors	[C]	19	class 2	False
2	Sean Bill	[A, C, B]	17	class 2	True
3	Carl Jonson	[]	18	class 1	False
4	Bob Evan	[F]	-	class 2	True

As a result, you have created a data frame with five columns: name, grade, age, group, and suspended. The columns we will focus on are name, age, and group as those are represented by the string values.

1. upper()

This function converts all the string into uppercase.

```
df.name.str.upper()
```

```
0    MICHAEL SMITH
1         ANA KORS
2        SEAN BILL
3      CARL JONSON
4         BOB EVAN
Name: name, dtype: object
```

> As you can see the all the letters in names have been changed to upper case.

Note the syntax of the code used to transform the string. You need to first call '.str' before calling the function upper(). The '.str' transforms the series object to its string form on which the actual string operation can be executed.

2. lower()

Lower() function works similarly to the upper() function but it does exactly the opposite, it lowers all characters in a string. Here you can see the results of calling it on the name column.

```
df.name.str.lower()
```

```
0    michael smith
1         ana kors
2        sean bill
3      carl jonson
4         bob evan
Name: name, dtype: object
```

3. isupper()

This function checks every string entry in a column, if it has all its characters in uppercase or not.

```
df.name.str.isupper()
```

```
0    False
1    False
2    False
3    False
4    False
Name: name, dtype: bool
```

4. islower()

This function checks every string entry in a column, if it has all its characters in lowercase or not.

```
df.name.str.islower()
```

```
0    False
1    False
```

```
2    False
3    False
4    False
Name: name, dtype: bool
```

5. isnumeric()

This function checks if the characters in the string are actually digits.

All of them have to be digits in order for isnumeric() to return True.

```
df.age.str.isnumeric()
```

```
0     True
1     True
2     True
3     True
4    False
Name: age, dtype: bool
```

6. len()

It computes the length of the string in each row of a column.

```
df.name.str.len()
```

```
0    13
1     8
2     9
3    11
4     8
Name: name, dtype: int64
```

7. split()

Split() function splits a string on the desired character.

It is very useful if you have a sentence and want to get a list of individual words. You can do that by splitting the string on the empty space (' ').

```
df.name.str.split(" ")
```

```
0    [Michael, Smith]
1         [Ana, Kors]
2        [Sean, Bill]
3      [Carl, Jonson]
4         [Bob, Evan]
Name: name, dtype: object
```

8. startswith(pattern)

Returns true if the element in the Given column starts with the pattern.

```
df.name.str.startswith ('M')
```

```
Strings that start with 'T':
0     True
1    False
2    False
3    False
```

```
4    False
Name: name, dtype: bool
```

9. endswith(pattern)

Returns true if the element in the Given column ends with the pattern.

```
df.name.str.endswith('l')
```

```
0    False
1    False
2     True
3    False
4    False
Name: name, dtype: bool
```

10. find(pattern)

Returns the first position of the first occurrence of the pattern.

"-1" indicates that there is no such pattern available in the element.

```
df.name.str.find('s')
```

```
0   -1
1    7
2   -1
3    8
4   -1
Name: name, dtype: int64
```

11. findall()

findall() return a list of matching substrings.

It similar to find() but it will search a string for existing substring instead of one index.

```
df.name.str.findall('B')
```

```
0     []
1     []
2    [B]
3     []
4    [B]
Name: name, dtype: object
```

12. contains()

Contains() function can check if the string contains a particular substring.

It just returns the boolean value True or False.

```
df.name.str.contains('e')
```

```
0     True
1    False
2     True
3    False
4    False
Name: name, dtype: bool
```

Let's create a dataframe using .csv file to learn to apply String Manipulation Functions:

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv")

# Add new column 'First_Name' and write the output in that column
data["First Name"]= data["Name"].str.lower()

# display
data
```

	Unnamed: 0	Name	Team	Number	Position	Age	Height	Weight	College	Salary	Fi N
0	152	Aaron Brooks	Chicago Bulls	0.0	PG	31.0	6-0	161.0	Oregon	2250000.0	aa bro
1	356	Aaron Gordon	Orlando Magic	0.0	PF	20.0	6-9	220.0	Arizona	4171680.0	aa gor
2	328	Aaron Harrison	Charlotte Hornets	9.0	SG	21.0	6-6	210.0	Kentucky	525093.0	aa harri
3	404	Adreian Payne	Minnesota Timberwolves	33.0	PF	25.0	6-10	237.0	Michigan State	1938840.0	adre pa
4	312	Al Horford	Atlanta Hawks	15.0	C	30.0	6-10	245.0	Florida	12000000.0	hor
...	...	...	...	...	...	...	...	...	...	...	
453	270	Xavier Munford	Memphis Grizzlies	14.0	PG	24.0	6-3	180.0	Rhode Island	NaN	xa mun
454	402	Zach LaVine	Minnesota Timberwolves	8.0	PG	21.0	6-5	189.0	UCLA	2148360.0	z la

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv")

# Add new column 'Team_upper' and write the output in that column
data["Team_upper"]= data["Team"].str.upper()

# display
data
```

	Unnamed: 0	Name	Team	Number	Position	Age	Height	Weight	College	Salary	Team_upper
0	152	Aaron Brooks	Chicago Bulls	0.0	PG	31.0	6-0	161.0	Oregon	2250000.0	CHICAGO BULLS
1	356	Aaron Gordon	Orlando Magic	0.0	PF	20.0	6-9	220.0	Arizona	4171680.0	ORLANDO MAGIC
2	328	Aaron Harrison	Charlotte Hornets	9.0	SG	21.0	6-6	210.0	Kentucky	525093.0	CHARLOTTE HORNETS
		Payne	Timberwolves						State		TIMBERWOLVES
4	312	Al Horford	Atlanta Hawks	15.0	C	30.0	6-10	245.0	Florida	12000000.0	ATLANTA HAWKS
...	...	...	...	...	...	...	...	...	...	...	...
453	270	Xavier Munford	Memphis Grizzlies	14.0	PG	24.0	6-3	180.0	Rhode Island	NaN	MEMPHIS GRIZZLIES
		Zach	Minnesota								MINNESOTA

Colab paid products - Cancel contracts here