**Feature Engineering using Regular Expression (RegEx) in Pandas DataFrame**

**Discover how to manipulate your string columns easily using Regular Expressions**



Photo by [Clark Van Der Beken](https://unsplash.com/@snapsbyclark?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

Manipulating string columns in Pandas is one of the most common operations that a data engineer will perform. Most of the time, you would do things like splitting columns, extracting key information from…

A **regular expression** (commonly referred to as **regex** or **regexp**)is a sequence of characters that specifies a search pattern in text. Regex is extremely powerful, but it can also be intimidating to the beginners. And so in this article, I will provide a gentle introduction to regex to get you started.

**Our Dataset**

As usual, my favorite dataset to use for illustrating the concepts in this article is the Titanic dataset. It contains a few string columns that are excellent candidates for performing feature engineering.

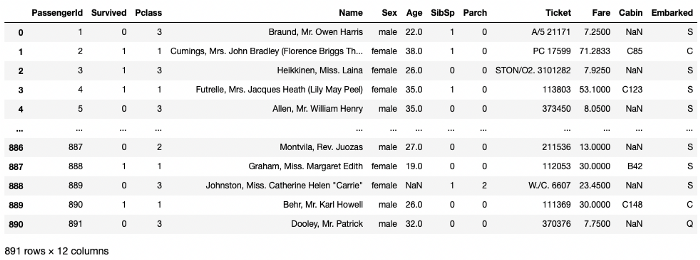
***Source of Data****: The data source for this article is from* [*https://www.kaggle.com/datasets/tedllh/titanic-train*](https://www.kaggle.com/datasets/tedllh/titanic-train)*.*

***Licensing —*** *Database Contents License (DbCL) v1.0* [*https://opendatacommons.org/licenses/dbcl/1-0/*](https://opendatacommons.org/licenses/dbcl/1-0/)

Let’s load the Titanic dataset and examine its various rows and columns:

import pandas as pd  
import numpy as np  
  
df = pd.read\_csv('titanic\_train.csv')  
df

In particular, I will perform feature engineering using the **Name** and **Cabin** columns:



All images in this article were created by the author.

**Basics of Regular Expressions**

Before we go head on to perform feature engineering on the Titanic dataset, it would be really useful to go through the basics of regex and learn how to use it in Python.

To use regex in Python, import the re package:

import re

In the following sections, I will illustrate how to use regex to do the following:

* How to search for a string within a string
* How to search for a string based on specific casing
* How to use anchors to search for strings in the beginning or end of a string
* How to search for strings with a specific pattern
* How to search for strings with repeated patterns

**Searching for a string**

Suppose you want to search for the world “William” in the string “Allen, Mr. William Henry”. You can use the search() function (from the re package) by passing in the word to search, and the string to search from:

expression = 'William'  
name\_search = re.search(expression, "Allen, Mr. William Henry")  
print(name\_search)

The search() function returns a Match object containing information about the search and the result:

<re.Match object; span=(11, 18), match='William'>

In the above output, you can see that a match for “William” was found from index of 11 to 17 (not including 18) of the string. If there is no match, None will be returned.

You can obtain the word that matches your search using the group() function of the Match object:

if name\_search:  
 # returns the part of the string where there was a match  
 print(name\_search.group())  
 # William

I will explain the concept of **groups** in more details in the later sections.

You can specify the group that matches your search using its group number:

print(name\_search.group(0)) # same as group()

The above statements both produce the same output:

William

**Searching for a string based on different casings**

Sometimes you want to search for a word based on specific casings. For example, if you want to look for “William” or “william”, you can use the [] construct to indicate a match of individual characters. For example, the following expression looks for either “W” or “w”, followed by “illiam”:

expression = '[Ww]illiam' # either W or w, followed by "illiam"  
name\_search = re.search(expression, "Allen, Mr. william Henry")  
if name\_search:  
 print(name\_search.group())   
 # william

If you want to match a word regardless of casing, use the (?i) modifier:

expression = '(?i)william' # case insensitive search  
name\_search = re.search(expression, "Allen, Mr. WILLIAM Henry")  
if name\_search:  
 print(name\_search.group())   
 # WILLIAM

**Searching for the beginning or end of string**

In regex, you can use anchors — $ and ^, to specify the position of strings that you want to match. For example, if you want to look for a string that ends with “Henry”, you can use the $ anchor in your expression:

expression = 'Henry$' # ends with Henry   
name\_search = re.search(expression, "Allen, Mr. William Henry")  
if name\_search:  
 print(name\_search.group())  
 # Henry

Likewise, if you want to look for the string that begins with “Allen”, you can use the ^ anchor in your expression:

expression = '^Allen' # starts with Allen  
name\_search = re.search(expression, "Allen, Mr. William Henry")  
if name\_search:  
 print(name\_search.group())  
 # Allen

**Searching for a pattern**

Suppose we have a string like this: “Allen, Mr. William Henry”. We are interested to extract the title — “Mr”, in this case, from the sentence. Based on the position of the title, we can see the following pattern:

* There is a space before the title
* The title ends with a period

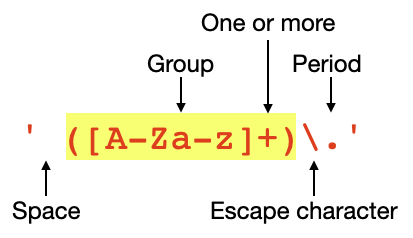
With this, we can create the following expression:

expression = ' ([A-Za-z]+)\.'  
name\_search = re.search(expression, "Allen, Mr. William Henry")  
print(name\_search)  
if name\_search:  
 print(name\_search.group(0)) # group zero is everything that matches the pattern  
 print(name\_search.group(1)) # group 1 is the first ()

The above expression needs a little explanation.

* First, there is a space indicating that we need to find the starting space in front of the title.
* Next, we have a pair of parentheses representing a group. Within this group, we have a [] indicating that we want to match any characters from “A” to “Z”, or “a” to “z”, and the + character indicates that we want to match one or more such character.
* The . (period) is a special character in RegEx that matches any character except new line (\n). So if we want to match a period in our string, we need to use the \ (escape character) to turn off the special meaning of ..

The following figure summarizes the above description:



The first output of name\_search.group(0) is (note the space in front of Mr.):

Mr.

This output contains the matches of your expression — ‘ ([A-Za-z]+)\.’. The second output is:

Mr

This output is the result of the first group (since there is only one group) in your expression — ([A-Za-z]+). Hence the output does not contain the period.

Since there is only one group in your expression, if you try to access the second group in the result, you will hit an error:

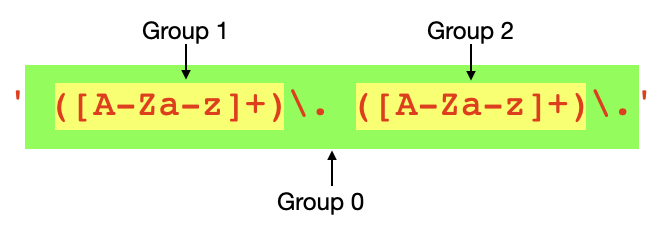
print(name\_search.group(2))  
 # IndexError: no such group

How about an example where we have more than one group in our expression?

Suppose we have a string containing a name with two titles (I know I am stretching this example a bit too far) and we want to find the two titles in this string. We could do that with the following expression:

expression = ' ([A-Za-z]+)\. ([A-Za-z]+)\.'  
titles\_search = re.search(expression, "Allen, Dr. Mr. William Henry")  
if titles\_search:  
 print(titles\_search.group(0)) # Dr. Mr.  
 print(titles\_search.group(1)) # Dr  
 print(titles\_search.group(2)) # Mr

The following figure explains the output of each group:



**Extracting based on repeat patterns**

Sometimes you just want to extract repeat patterns. For example, say you have a string containing different titles, like this: ‘Mrs. Mr. Ms. Dr.’. To extract all the titles in this string, you can use the findall() function, like this:

# returns a list of non-overlapping matches  
titles\_search = re.findall('([A-Za-z]+)\.', "Mrs. Mr. Ms. Dr.")  
print(titles\_search)

The result will be a list of non-overlapping matches. In this case, the output is:

['Mrs', 'Mr', 'Ms', 'Dr']

**Using Regular Expression on a Pandas Series str() function**

Now that you had a taste of how regex works, it is now time to see how you can use regex on Pandas. The following methods in Pandas Series’s vectorized string functions (Series.str()) allows you to use regex expressions:

* count()
* extract()
* match()
* contains()
* replace()
* findall()
* split()

Let’s go through each of these functions with examples.

**Using the count() function**

Let’s start with the count() function. Say you want to find all the people in the Titanic dataset that has names that start with “Braund” and contains “Owen”. You can do so using the following expression in the count() function:

expression = '^Braund.+Owen'   
print(df['Name'].str.count(expression))

Here are the uses of the various special characters in regex:

^ Start of string or line  
. Matches any single character except the newline character.   
+ 1 or more occurrences  
\* 0 or more occurrences

You will see the result as a Series, with value of 1 for each row that matches your expression:

**0 1**  
1 0  
2 0  
3 0  
4 0  
 ..  
886 0  
887 0  
888 0  
889 0  
890 0  
Name: Name, Length: 891, dtype: int64

If you want to count the total number of rows that match your search expression, you can use the sum() function:

print(df['Name'].str.count(expression).sum())  
# 1

To display the row(s) that match the expression, you can pass the result into the original dataframe:

display(df[df['Name'].str.count(expression) == 1])

The following figure shows the row(s) that matched the expression:

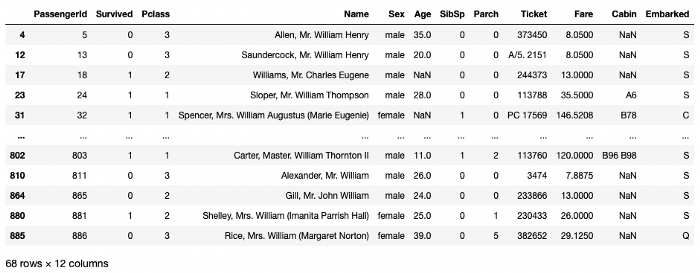
https://miro.medium.com/max/700/1*Tlg2Qu9WZengwhClnoy42Q.png

Note the + in the expression ‘^Braund.+Owen’. In this case, it means that there must be at least one character separating Braund and Owen. For example, “Braund,Owen”, or “Braund Owen”. If you want to match names such “BraundOwen”, then your expression should use \*:

expression = '^Braund.\*Owen'

If you simply want to find all the names that contains the word “William”, then simply set the expression to “William”, like this:

expression = 'William'



The following statements show row(s) that contain more than one occurrences of the word “William”:

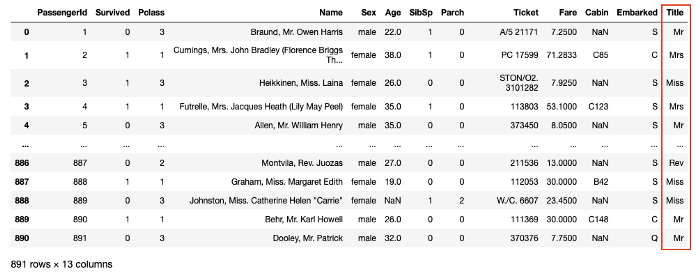
expression = 'William'  
print(df['Name'].str.count(expression))  
print(df['Name'].str.count(expression).sum())  
display(df[df['Name'].str.count(expression) > 1])

https://miro.medium.com/max/700/1*w5abxiXmQV-0zugnpCgM1Q.png

**Using the extract() function**

Remember earlier I mentioned that you can extract the title from the passenger’s name? Using the **Name** column, you can use the extract() function to pass in a regex expression to extract the title. The result can then be added as a new column in the dataframe:

expression = ' ([A-Za-z]+)\.'  
df['Title'] = df['Name'].str.extract(expression)  
df



Another interesting column that you can use the extract() function on is the **Cabin** column. If you examine the unique values of the **Cabin** column:

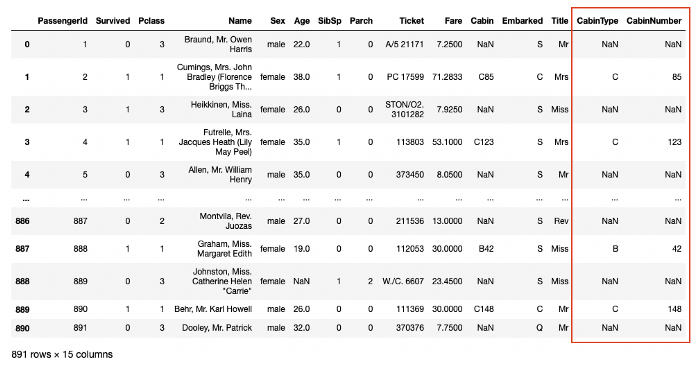
print(df['Cabin'].unique())

You will see the following:

[nan 'C85' 'C123' 'E46' 'G6' 'C103' 'D56' 'A6' 'C23 C25 C27' 'B78' 'D33'  
 'B30' 'C52' 'B28' 'C83' 'F33' 'F G73' 'E31' 'A5' 'D10 D12' 'D26' 'C110'  
 'B58 B60' 'E101' 'F E69' 'D47' 'B86' 'F2' 'C2' 'E33' 'B19' 'A7' 'C49'  
 'F4' 'A32' 'B4' 'B80' 'A31' 'D36' 'D15' 'C93' 'C78' 'D35' 'C87' 'B77'  
 'E67' 'B94' 'C125' 'C99' 'C118' 'D7' 'A19' 'B49' 'D' 'C22 C26' 'C106'  
 'C65' 'E36' 'C54' 'B57 B59 B63 B66' 'C7' 'E34' 'C32' 'B18' 'C124' 'C91'  
 'E40' 'T' 'C128' 'D37' 'B35' 'E50' 'C82' 'B96 B98' 'E10' 'E44' 'A34'  
 'C104' 'C111' 'C92' 'E38' 'D21' 'E12' 'E63' 'A14' 'B37' 'C30' 'D20' 'B79'  
 'E25' 'D46' 'B73' 'C95' 'B38' 'B39' 'B22' 'C86' 'C70' 'A16' 'C101' 'C68'  
 'A10' 'E68' 'B41' 'A20' 'D19' 'D50' 'D9' 'A23' 'B50' 'A26' 'D48' 'E58'  
 'C126' 'B71' 'B51 B53 B55' 'D49' 'B5' 'B20' 'F G63' 'C62 C64' 'E24' 'C90'  
 'C45' 'E8' 'B101' 'D45' 'C46' 'D30' 'E121' 'D11' 'E77' 'F38' 'B3' 'D6'  
 'B82 B84' 'D17' 'A36' 'B102' 'B69' 'E49' 'C47' 'D28' 'E17' 'A24' 'C50'  
 'B42' 'C148']

The cabin values start with an alphabet from A to G, followed by a numerical value. It might be useful to be able to extract the alphabet (which may represent the different decks on the Titanic or the different cabin type), followed by the numerical value. You can do the extraction as follows:

expression = '^([A-G])'  
df['CabinType'] = df['Cabin'].str.extract(expression)  
  
expression = '^.{1}(.\*)'   
df['CabinNumber'] = df['Cabin'].str.extract(expression)  
df



The expression of ‘^.{1}(.\*)’ means that the value of the **Cabin** should start with exactly one character (^.{1}) and then you want to match the rest with zero or more characters ((.\*)). This will essentially extract the cabin number.

Unfortunately, the Cabin column has too many empty cells to make this column useful.

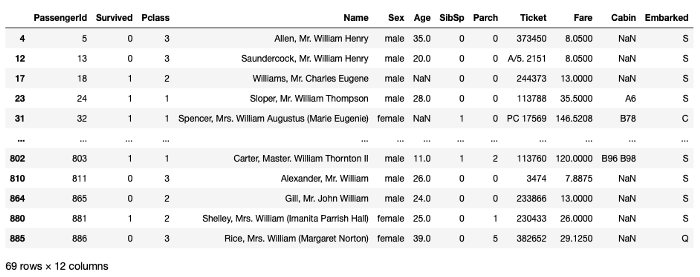
**Using the contains() and match() functions**

Recall earlier we used the count() function to find and count all rows containing the word “William”?

expression = 'William'  
print(df['Name'].str.count(expression).sum())  
display(df[df['Name'].str.count(expression) == 1])

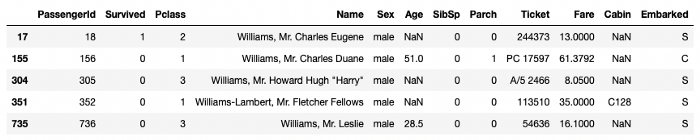
If you don’t need to count the rows and simply want the rows, you can use the contains() function:

expression = 'William'  
df[df['Name'].str.contains(expression)]



The above result shows all the rows whose name contains the word “William”. Of course, you can use the ^anchor in your expression to indicate that you want all rows whose name starts with “William”:

expression = '^William'  
df[df['Name'].str.contains(expression)]



If you simply want to find strings at the beginning, you can actually use the match() function. The following code snippet produces the same output as the above:

expression = 'William'  
df[df['Name'].str.match(expression)] # Determine if string starts with   
 # a match of a regular expression

**Using the replace() function**

Earlier, we extracted the title of the passenger using their names. Let’s examine the different titles that we have extracted:

print(df['Title'].unique())  
# ['Mr' 'Mrs' 'Miss' 'Master' 'Don' 'Rev' 'Dr' 'Mme' 'Ms' 'Major' 'Lady'  
# 'Sir' 'Mlle' 'Col' 'Capt' 'Countess' 'Jonkheer']

You can see that some of the titles are duplicates — titles such as *Mlle*, *Ms*, and *Mme* can be abbreviated as **Miss** using the replace() function:

df['Title'] = df['Title'].str.replace('Mlle', 'Miss')  
df['Title'] = df['Title'].str.replace('Ms', 'Miss')  
df['Title'] = df['Title'].str.replace('Mme', 'Miss')

Here’s the meaning of Mme and Mlle:

**Mme** (Madame) — french word for a woman

**Mlle (**Mademoiselle**) —** French courtesy title, traditionally given to an unmarried woman.

If you want to replace the title in the **Name** column directly, you can pass in a regex expression into the replace() function

df['Name'] = df['Name'].str.replace(' (Mlle)\.', ' Miss.', regex=True)  
df['Name'] = df['Name'].str.replace(' (Ms)\.', ' Miss.', regex=True)  
df['Name'] = df['Name'].str.replace(' (Mme)\.', ' Miss.', regex=True)

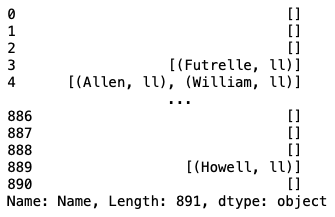
The titles in the **Name** column will now be updated.

**Using the findall() function**

If you want to quickly find all the names in the **Name** column that matches certain patterns, you can use the findall() function:

expression = '([A-Za-z]\*(ll)[a-z]\*)'  
df['Name'].str.findall(expression)

The above expression finds all the names that contain the letters “**ll**”. The findall() function returns a series of list values:



Each item in the list is a list containing zero or more tuples. Each tuple contains two elements — the first is the matching word and the second is the value of the group.

To print all matching rows, you can use list comprehension:

[i for i in df['Name'].str.findall(expression) if len(i)>0]

The above expression produce the following output:

[[('Futrelle', 'll')],  
 [('Allen', 'll'), ('William', 'll')],  
 [('Bonnell', 'll')],  
 [('William', 'll')],  
 [('Williams', 'll')],  
 [('William', 'll')],  
 [('Ellen', 'll'), ('Nellie', 'll')],  
 [('William', 'll')],  
 [('William', 'll')],  
 ...

If you want to print out the names nicely, here is a little code snippet to do so:

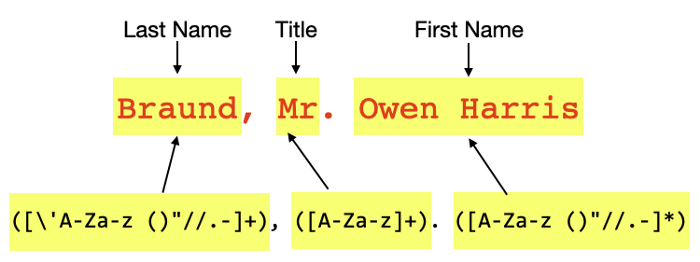
expression = '([A-Za-z]\*(ll)[A-Za-z]\*)'  
for matches in df['Name'].str.findall(expression):  
 if len(matches)>0:  
 for names in matches:  
 print(names)  
 print('----')

And the result looks like this:

('Futrelle', 'll')  
----  
('Allen', 'll')  
('William', 'll')  
----  
('Bonnell', 'll')  
----  
('William', 'll')  
----  
('Williams', 'll')  
----  
('William', 'll')  
----  
('Ellen', 'll')  
('Nellie', 'll')  
...

**Using the split() function**

If you examine the name for each passenger, you will see the following pattern and the associated regex for finding them:



More interesting are the expressions

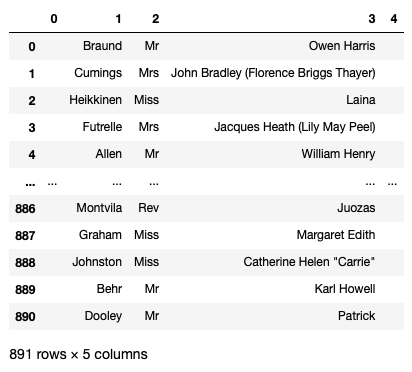
* ([\’A-Za-z ()”//.-]+)
* ([A-Za-z ()”//.-]\*)

This is because the first and last name of a passenger may include the following special characters:

* — Space
* ‘ — Apostrophe (example — “O**’**Driscoll” for last name)
* () — parentheses (example — “John Bradley **(**Florence Briggs Thayer**)**” for first name)
* “ — Double quotes (example — **“**Anna “Annie**”**” for first name)
* / — Slash (example — “Carl**/**Charles Peter” for first name)
* . — Period (example “Martin (Elizabeth L**.** Barrett)” for first name)
* - — Hyphen (example —“ Countess. of (Lucy Noel Martha Dyer**-**Edwards)” for first name)

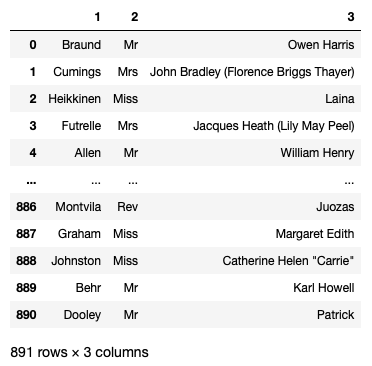
You can now use the following code snippet to extract the last name, title, and first name of the passengers:

expression = '([\'A-Za-z ()"//.-]+), ([A-Za-z]+). ([A-Za-z ()"//.-]\*)'  
df1 = df['Name'].str.split(expression, expand=True)  
display(df1)



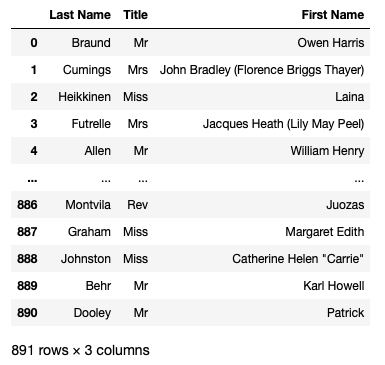
Columns 0 and 4 can be dropped as they are empty:

df1.drop([0,4], axis=1, inplace=True)  
display(df1)



Finally, you can assign the dataframe some meaningful column names:

df1.columns = ['Last Name','Title','First Name']  
display(df1)



The extraction of title is helpful in determining the age of a passenger if the age value is missing. On the other hand, extracting the last name is helpful to determine if two or more passengers are from the same family.

**Summary**

This has been quite a whirlwind tour of using regular expression in Python, as well as in Pandas. As there are quite a number of functions that I have used, I thought it might be useful to summarize the functions that I have covered and when you should use them:

* count() — use this function if you want to count the number of rows matching your patterns
* extract() — use this function if you want to extract strings from a string
* match() — use this function if you want to check if a string starts with a specific string
* contains() — use this function if you want to check if a string contains a specific string
* replace() — use this function to replace a string with another string
* findall() — use this function to find all occurrences of a pattern within a string
* split() — use this function to split the strings into multiple columns

Have fun with regex and let me know if this is useful to you!