Regular Expression in SAS

Dajun Tian

Outline

- Common tasks dealing with text
- Warmup Regular Expression(RE) example in SAS
- Overview of RE
- Two general forms of SAS RE

Text Searching

- Substring search
  - Find a particular substring from a string

Application	SAS Functions
Check if patient's comments include 'excellent'	**index, find**
Example Code	
mention_excellent = **find**("the doctor's staff is excellent", 'excellent')	

Text Searching – More complicated example

- Find orders that are prescribed every xx hours

```
data find_dose;
    input order_frequency $ 1-20;
datalines;
every 3 hour
every 5 hour
every 6 hour
every 2.5 hour
every 2 weeks
as needed
once
;;;
run;
```

Obs	order_frequency
1	every 3 hour
2	every 5 hour
3	every 6 hour
4	every 2.5 hour
5	every 2 weeks
6	as needed
7	once

- Solution 1: Use scan

```
data find_every_hr;
    set find_dose;
    if scan(order_frequency, 1, ' ') = 'every' and
       scan(order_frequency, 3, ' ') = 'hour';
run;
```

Obs	order_frequency
1	every 3 hour
2	every 5 hour
3	every 6 hour
4	every 2.5 hour

- Solution 2: Use regular expression

```
data find_every_hr;
    set find_dose;
    if prxmatch("/^every .{1,4} hour/i", order_frequency);
run;
```

Definition of regular expression

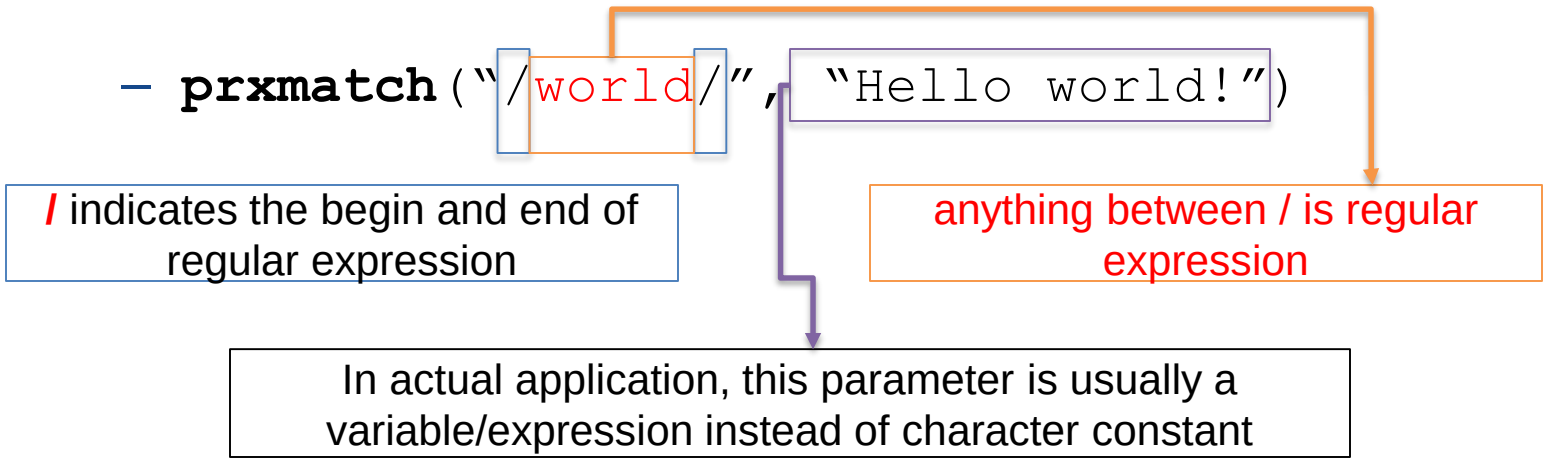
- A regular expression is a notation to specify a set of strings.

operation	example regular expression	matches	does not match
concatenation	Excellent	Excellent	*every other string*
or	good|excellent	good excellent	*every other string*
closure	AB*A	AA ABBA ABBBBBA	AB ABABA
parentheses	every(day|week)	everyday everyweek	*every other string*
	(AB)*A	A ABA ABABA	AA ABBA

https://algs4.cs.princeton.edu/lectures/54RegularExpressions.pdf

Find a Match: function forms

- Basic Syntax for Finding a Match in a String
  - general form is `/regular-expression/`
  - **`prxmatch`**`("/world/", "Hello world!")`



  - **`prxmatch`** returns the position of the first occurrence
  - returns 7 in this example
  - this is equivalent to **`find`**`("Hello world!", "world")`

Find a Match: build regular expression

- To extend the power of regular expression, we refer to Regular Expression (PRX) Metacharacters
- What is metachracter? A metacharacter is a character that has a special meaning to a regular expression. For example, * means the preceding character could occur 0 or more times.
- SAS observes these metacharacters:
  - { } [ ] ( ) ^ $. | * + ? \
  - To match metacharacter, use \ to override

```
prxmatch("/world\(you\)/", "Hello world(you)")
```

```
find("Hello world(you)", "world(you)")
```

Build Regular Expression: greedy/lazy repetition

Define Pattern	Description	Example
*	matches preceding zero or more times	zo* matches z, zo, zoo and so on
+	matches preceding one or more times	zo+ matches zo and zoo zo+ does not match “z”
?	matches preceding zero or one time	do(es)? matches do and dose only.
{n,}	matches a pattern at least n times	zo{1,} is equivalent to zo+ zo{0,} is equivalent to zo*
{n,m}	matches n-m times.	zo{0,1} is equivalent to zo?

Lazy Repetition is implemented by adding ?. For example,
zo*? will match a pattern by as few characters as possible.

Build pattern using metachracters

Define Pattern	Description
\d	matches a digit character
\D	matches a non-digit character
\w	matches any word character or alphanumeric character, including the underscore.
\s	matches any whitespace character, including space, tab, form feed, and so on, and is equivalent to [\f\n\r\t\v].
\S	matches any character that is not a whitespace character and is equivalent to [^\f\n\r\t\v].

Build Regular Expression: Groupings

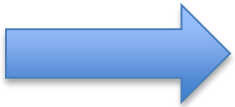
Metacharacter	Description and example
[…]	matches any one of the enclosed characters: • [abc] matches the “a” in “apple”
[^…]	matches any character that is not encloded • [^abc] matches the “p” in “apple”
[[:alpha:]]	matches an alphabetic character.
[[:alnum:]]	matches an alphanumeric character.
[[:^alpha:]]	matches a nonalphabetic character.
[[:^alnum:]]	matches a non-alphanumeric character.
[[:space:]]	matches a space.
[[:^space:]]	matches a non-space character

Section Summary

- We could use
  - Repetition
  - Build pattern as number, space etc.
  - Define groups

Check if string contains a phone number

```
data find_phone_number;
    input message $ 1-50;
datalines;
My phone is 314-xxx-xxxx.
cell is (314) 111-3456
mobile number: (322) 899-1234
;;;
run;
```



Obs	message
1	My phone is 314-xxx-xxxx.
2	cell is (314) 111-3456
3	mobile number: (322) 899-1234

```
data check_number;
    set find_phone_number;
    if prxmatch("/\(\d\d\d\) ?\d\d\d-\d\d\d\d/", message);
run;
```

```
data check_number;
    set find_phone_number;
    if prxmatch("/\(\d{3}\) ?\d{3}-\d{4}/", message);
run;
```

Obs	message
1	cell is (314) 111-3456
2	mobile number: (322) 899-1234

Match and Extract- Example Continued

- After pattern is identified (phone number for example), we also want to extract the information.

Obs	message
1	My phone is 314-xxx-xxxx.
2	cell is (314) 111-3456
3	mobile number: (322) 899-1234

```
data check_number;
    set find_phone_number;
    if prxmatch("/\(\d{3}\) ?\d{3}-\d{4}/", message);

    prxID = prxparse("/\(\d{3}\) ?\d{3}-\d{4}/");
    call prxsubstr(prxID, message, position, length);
    phone_no = substr(message, position, length);
run;
```

Return a unique pattern ID

The pattern identifier number containing the targeted pattern

The input string

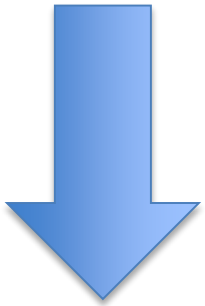
The position of the targeted pattern

The length of the targeted pattern

Obs	message	prxID	position	length	phone_no
1	cell is (314) 111-3456	1	9	14	(314) 111-3456
2	mobile number: (322) 899-1234	1	16	14	(322) 899-1234

What if multiple patterns matched.

```
data find_phone_number;
    infile datalines truncover;
    input message $ 1-100 ;
datalines;
My phone is 314-xxx-xxxx.
cell is (314) 111-3456 and my office is (314)123-4567
mobile number: (322) 899-1234
;;;
run;
```



Records having multiple phone numbers

Obs	message
1	My phone is 314-xxx-xxxx.
2	cell is (314) 111-3456 and my office is (314)123-4567
3	mobile number: (322) 899-1234

What if we want to extract multiple patterns?

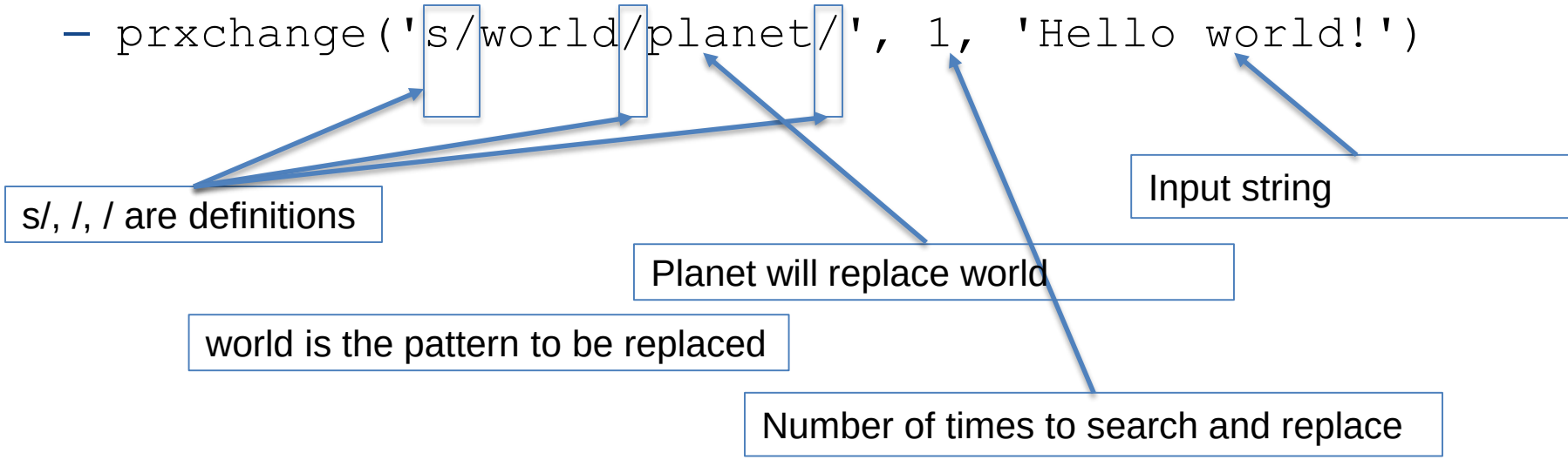
Obs	message
1	My phone is 314-xxx-xxxx.
2	cell is (314) 111-3456 and my office is (314)123-4567
3	mobile number: (322) 899-1234

```
data check_number;
    set find_phone_number;
    if prxmatch("/\(\d{3}\) ?\d{3}-\d{4}/", message);

    prxID = prxparse("/\(\d{3}\) ?\d{3}-\d{4}/");

    call prxsubstr(prxID, message, position, length);
    message_rest = message;

    do while(position > 0);
        phone_no = substr(message_rest, position, length);
        output;
        message_rest = substrn(message_rest, position + length);
        call prxsubstr(prxID, message_rest, position, length);
    end;
run;
```

Obs	message	prxID	position	length	message_rest	phone_no
1	cell is (314) 111-3456 and my office is (314)123-4567	1	9	14	cell is (314) 111-3456 and my office is (314)123-4567	(314) 111-3456
2	cell is (314) 111-3456 and my office is (314)123-4567	1	19	13	and my office is (314)123-4567	(314)123-4567
3	mobile number: (322) 899-1234	1	16	14	mobile number: (322) 899-1234	(322) 899-1234

Find and replace in SAS

- Basic Syntax for Searching and Replacing Text
  - general form is `s/regular-expression/replacement-string/`
  - `prxchange('s/world/planet/', 1, 'Hello world!')`



  - The function returns "`Hello planet!`".

Revisit the phone number example

Obs	message
1	My phone is 314-xxx-xxxx.
2	cell is (314) 111-3456
3	mobile number: (322) 899-1234

```
data check_number;
    set find_phone_number;
    if prxmatch("/\(\d{3}\) ?\d{3}-\d{4}/", message);

    prxID = prxparse("/\(\d{3}\) ?\d{3}-\d{4}/");
    call prxsubstr(prxID, message, position, length);
    phone_no = substr(message, position, length);
run;
```

- Our task is to replace phone number with xxx

```
data check_number;
    set find_phone_number;
    message2 = prxchange("s/\(\d{3}\) ?\d{3}-\d{4}/xxx/", 1, message);
run;
```

Obs	message	message2
1	My phone is 314-xxx-xxxx.	My phone is 314-xxx-xxxx.
2	cell is (314) 111-3456	cell is xxx
3	mobile number: (322) 899-1234	mobile number: xxx

Summary

- Use prxmatch to find patterns
- Refer to metacharacter table to construct RE
- Use call prxsubstr to locate position and length of matched pattern
  - /regular expression/
- Use prxchange to search and replace
  - s/pattern-to-match/pattern-to-replace/
- How to use SAS documentation for RE
  - https://regexr.com
  - http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#p1vz3ljudbd756n19502acxazevk.htm
  - http://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p0s9ilagexmjl8n1u7e1t1jfnzlk.htm&docsetVersion=9.4&locale=en

Reference

- http://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=p0s9ilagexmjl8n1u7e1t1jfnzlk.htm&docsetVersion=9.4&locale=en

- https://algs4.cs.princeton.edu/lectures/54RegularExpressions.pdf

Thank you!
Any questions are welcome.