CS334: Principles and Techniques of Data Science

Analyzing Text Data: String Manipulation & Regular Expressions

Ihsan Ayyub Qazi

Slides use information from Deborah Nolan and Joseph E. Gonzalez

Working with Text

- A lot of data resides as free-form text in books, documents, blog posts, and Internet comments
- While numerical and categorical data are often collected from physical phenomena, textual data arises from human communication
- Many techniques for working with text
  - We introduce only a small subset of these techniques: Python string manipulation and regular expressions

Use-cases for string manipulation & regular expressions

1. **Clean text**
2. **Locate/extract fields**
3. **Derive features**
4. **Analyze text**

We will give an example of each

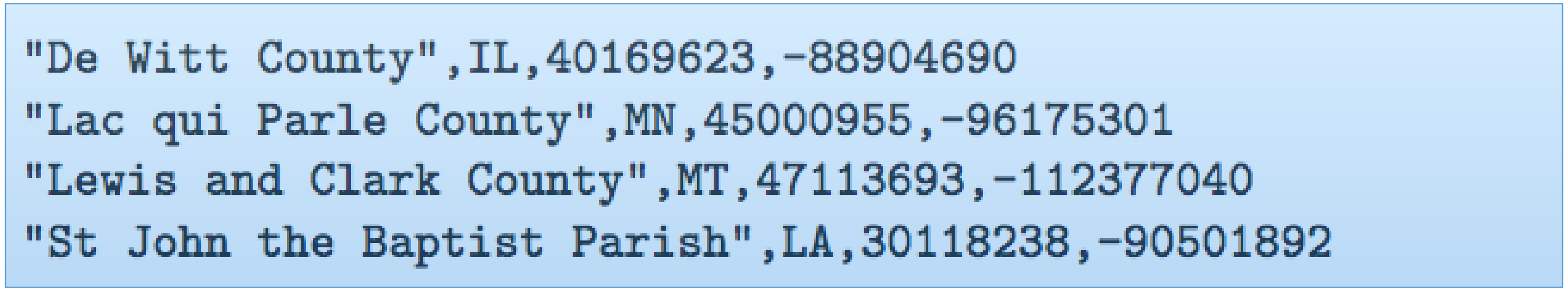
(1) Joining different sources of data

We want to make a county map with census information and election information. This information is in 3 files

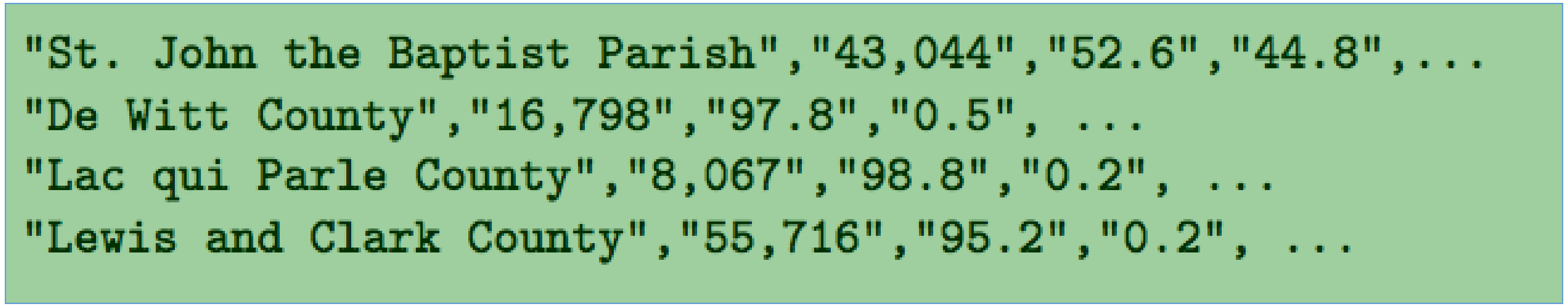
- Geographic: longitude and latitude of the county center
- Census: demographic statistics for each county
- Political: election results from each county

Need a primary key in each file to join records

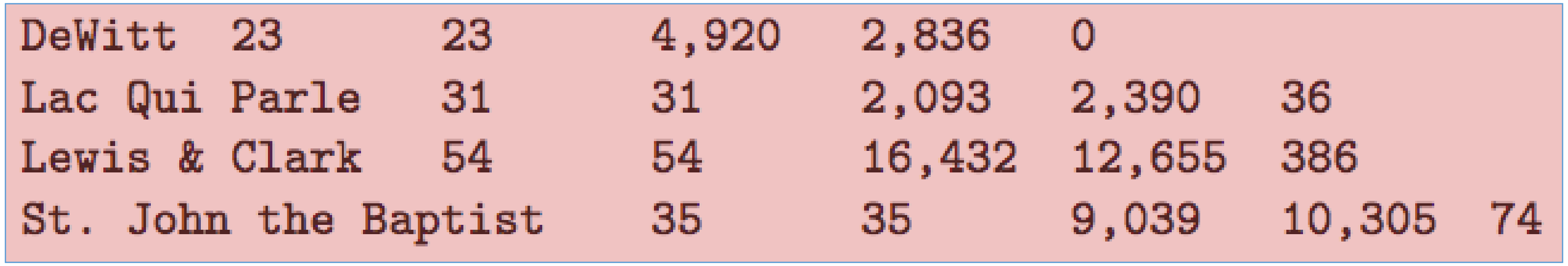
Sample records from 3 files



```
"De Witt County",IL,40169623,-88904690
"Lac qui Parle County",MN,45000955,-96175301
"Lewis and Clark County",MT,47113693,-112377040
"St John the Baptist Parish",LA,30118238,-90501892
```



```
"St. John the Baptist Parish","43,044","52.6","44.8",...
"De Witt County","16,798","97.8","0.5", ...
"Lac qui Parle County","8,067","98.8","0.2", ...
"Lewis and Clark County","55,716","95.2","0.2", ...
```



```
DeWitt  23      23      4,920   2,836   0
Lac Qui Parle   31      31      2,093   2,390   36
Lewis & Clark   54      54      16,432  12,655  386
St. John the Baptist    35      35      9,039   10,305  74
```

What problems needs resolving in order to join these sources?

- Capitalization: Qui vs. qui
- Punctuation: St. John vs. St John
- County/Parish: De Witt County vs. De Witt
- & vs. and: Lewis and Clark vs. Lewis & Clark
- Blanks: DeWitt vs. De Witt

(2) Web behavior

Every time you visit a Web site, information is recorded about the visit:

- Page visited
- Date and time of visit
- Browser used
- IP address

Two lines of a Web log

Is this a csv file? tsv?
Fixed width format?

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04 HTTP/1.1" 301 328
"http://anson.ucdavis.edu/courses/"  "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)”
```

```
169.237.6.168 - - [8/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"
```

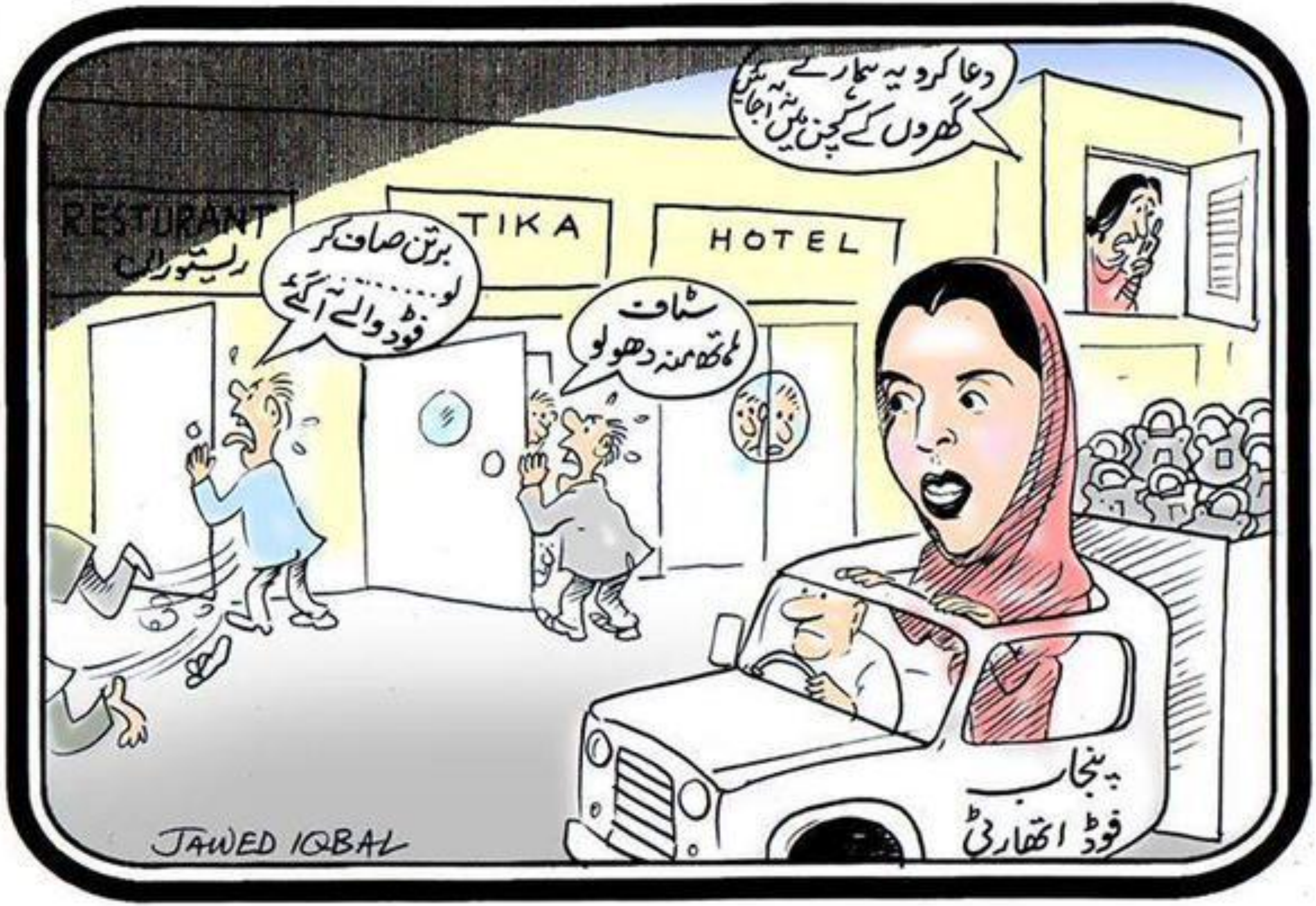
Understanding Web logs

- The information in the log has a lot of structure (e.g., the date always appears in square brackets)
- However, the information is not consistently separated by the same delimiters, as in a csv or tsv file
- And, the information is not consistently placed at the same locations in the file (e.g., the date does not always begin at the 20th character in the string)

(3) Food Safety Violations

The types of violations might be something that we want to investigate:

- What is the relationship between violation and inspection score?
- What are the most common violations?


RESTURANT
ریسٹورانٹ
TIKA
HOTEL
برتن صاف کر لو...... فوڈ والے آگئے
سٹاف ہاتھ منہ دھو لو
دعا کرو یہ ہمارے گھروں کے کچن میں نہ آجائیں
پنجاب فوڈ اتھارٹی
JAWED IQBAL

Descriptions of Violations

```
Unapproved or unmaintained equipment or utensils

Inadequate and inaccessible handwashing facilities

Food safety certificate or food handler card not available

Unclean or degraded floors walls or ceilings

No hot water or running water

Improper storage of equipment utensils or linens  [ date violation
corrected: 5/13/2016 ]

Inadequate food safety knowledge or lack of certified food safety
manager  [ date violation corrected: 5/13/2016 ]

High risk vermin infestation

Inadequately cleaned or sanitized food contact surfaces  [ date
violation corrected: 2/14/2014 ]

Employee eating or smoking
```

Are there “standard” descriptions?

- There are 15072 unique descriptions in 2016 alone
- BUT, many descriptions include the correction date
- When we remove this information, we find there are only 64 unique descriptions
- What are the top 25 violations?

Top 25 Violations

unclean or degraded floors walls or ceilings

moderate risk food holding temperature

inadequate and inaccessible handwashing facilities

unapproved or unmaintained equipment or utensils

Inadequately cleaned or sanitized food contact surfaces

wiping cloths not clean or properly stored or inadequate sanitizer

improper food storage

foods not protected from contamination

high risk food holding temperature

moderate risk vermin infestation

food safety certificate or food handler card not available

unclean nonfood contact surfaces

permit license or inspection report not posted

inadequate food safety knowledge or lack of certified food safety manager

unclean or unsanitary food contact surfaces

low risk vermin infestation

improper storage of equipment utensils or linens

unclean hands or improper use of gloves

improper or defective plumbing

improper cooling methods

improper thawing methods

Inadequate washing facilities or equipment

high risk vermin infestation

no thermometers or uncalibrated thermometers

improper storage use or identification of toxic substances

What features might you derive?

- Vermin related
- High-risk related
- Hand related (gloves, nails, …)
- Surface related, e.g., floor, wall, ceiling
- Food related, e.g., food surface, food contamination

(4) State of the Union Speeches

- Text Analysis
  - Do speeches/words/sentences have similar lengths?
  - How does the use of a word(s) change over time?
  - Which presidents give similar speeches?

What does it mean for one speech to be similar to another?

The first State of the Union Address

```
***
State of the Union Address
George Washington
December 8, 1790

Fellow-Citizens of the Senate and House of Representatives:
In meeting you again I feel much satisfaction in being able to repeat
my congratulations on the favorable prospects which continue to
distinguish our public affairs. The abundant fruits of another year
have blessed our country with plenty and with the means of a
flourishing commerce. …
```

Text Mining

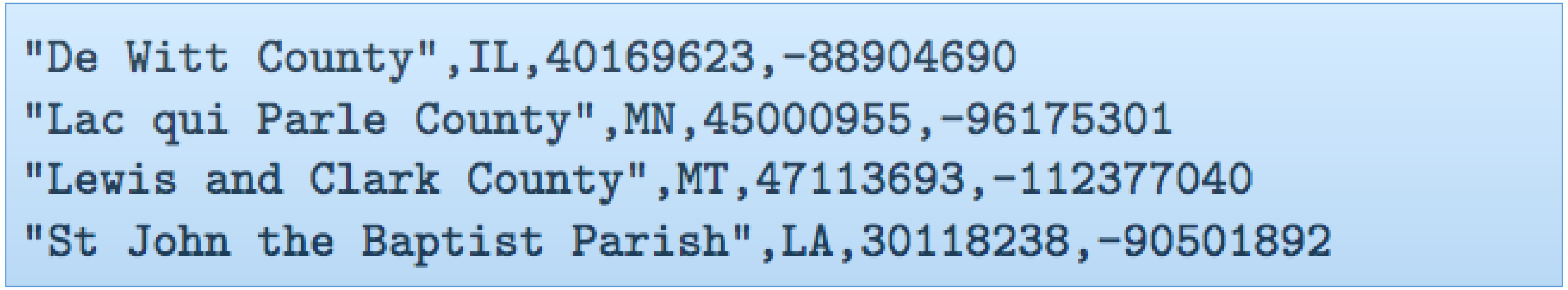
- All speeches in one large plain text file
- Each speech starts with "***" on one line, followed by 3 lines of information about who gave the speech and when
- One approach: Create a word vector for each speech that holds the counts of the number of times a particular word was used in a speech (including 0 counts)
- Words such as nation, nations, national can/should be considered the same "word"

Four Examples

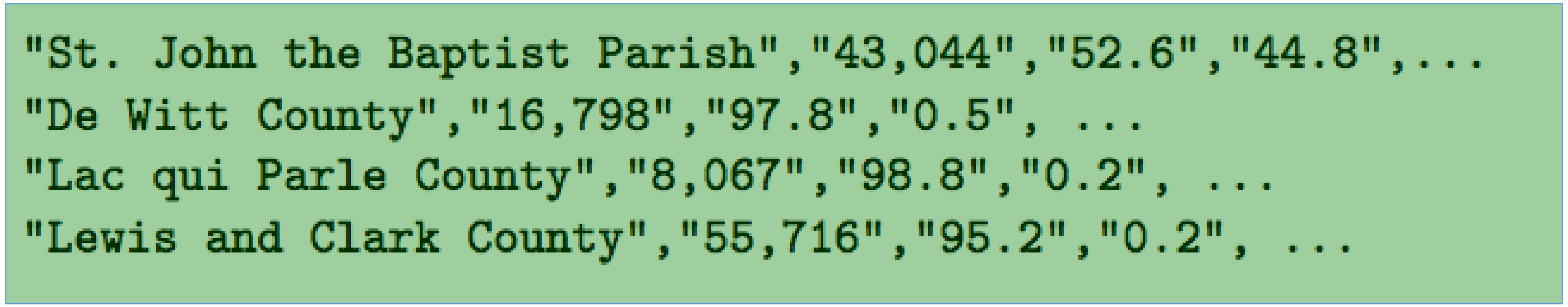
1. Census stats, Election results, Geographic coordinates Join – clean text
2. Web logs – extract fields from text
3. Food safety – derive features
4. State of the Union speeches – text analysis

String Manipulation

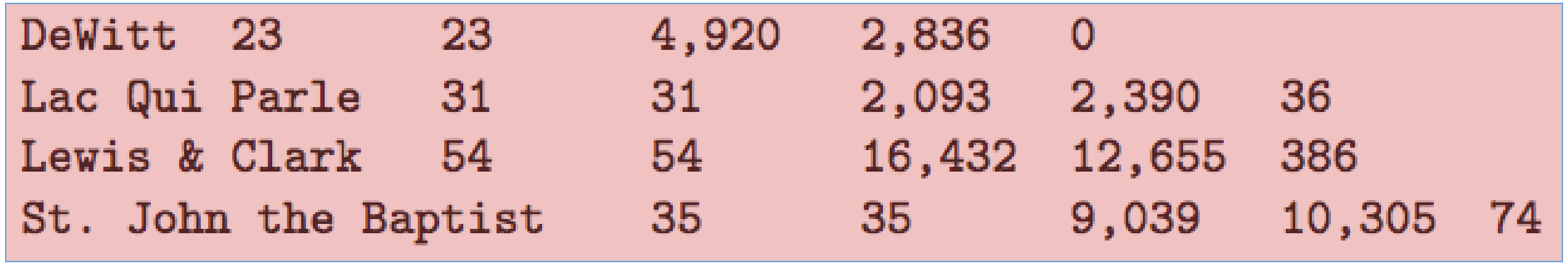
(1) Recall County Names



```
"De Witt County",IL,40169623,-88904690
"Lac qui Parle County",MN,45000955,-96175301
"Lewis and Clark County",MT,47113693,-112377040
"St John the Baptist Parish",LA,30118238,-90501892
```



```
"St. John the Baptist Parish","43,044","52.6","44.8",...
"De Witt County","16,798","97.8","0.5", ...
"Lac qui Parle County","8,067","98.8","0.2", ...
"Lewis and Clark County","55,716","95.2","0.2", ...
```



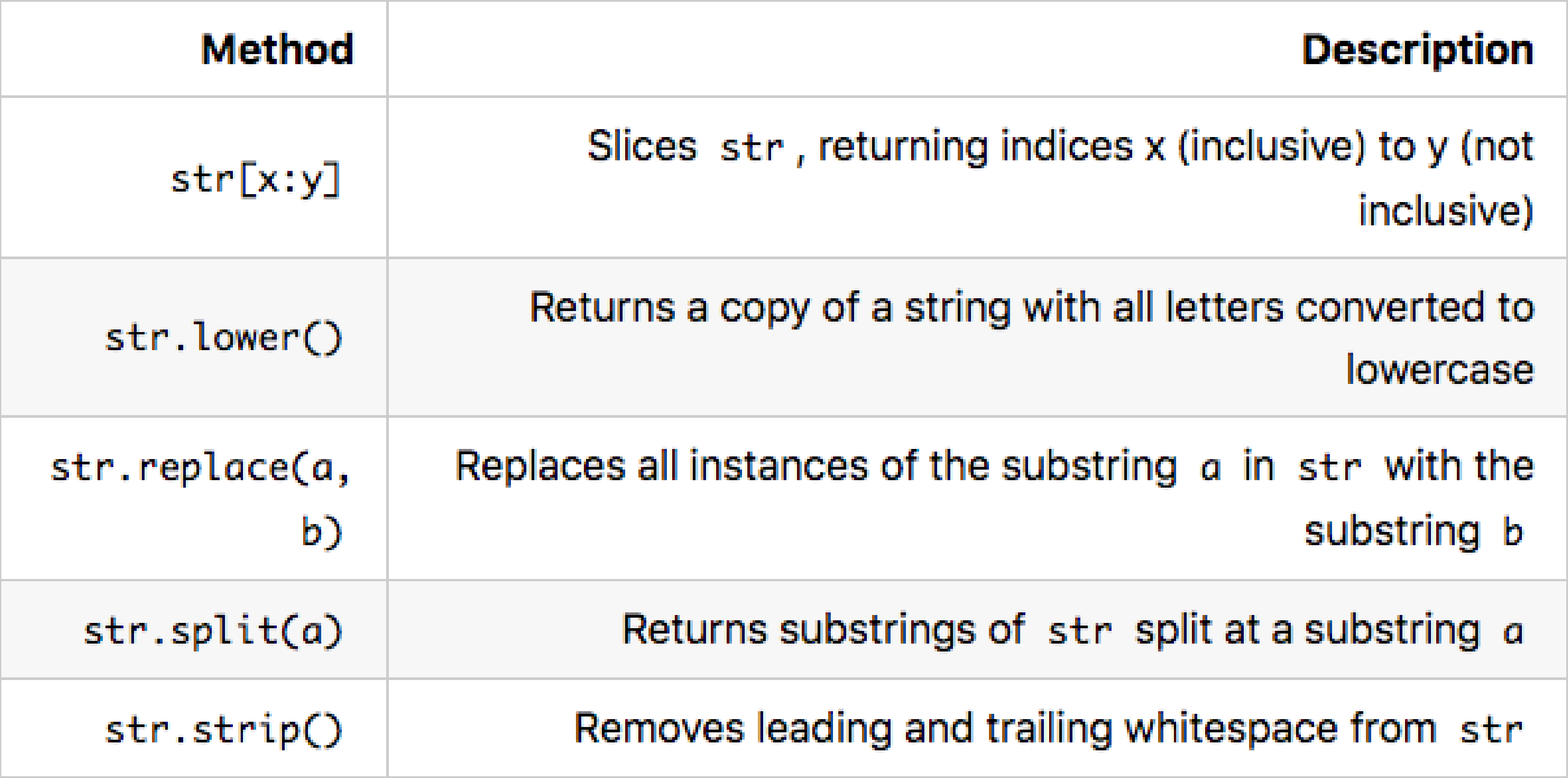
```
DeWitt   23        23        4,920    2,836    0
Lac Qui Parle     31        31       2,093    2,390    36
Lewis & Clark     54        54       16,432   12,655   386
St. John the Baptist        35       35       9,039    10,305   74
```

Create set that encapsulates various issues

- We typically handle case first as it is easy to make letters lower case
- County and Parish are both 6 letters long so we can simply drop the last 7 characters without checking.
- Then we can examine each character in turn and drop if . or space and swap for “and” if &.

```
county_ex = [
  'De witt County',
  'Lac qui Parle County',
  'St. John the Baptist Parish',
  'Stone County',
  'Lewis & Clark County'
]
```

Some String Methods



Method	Description
`str[x:y]`	Slices `str`, returning indices x (inclusive) to y (not inclusive)
`str.lower()`	Returns a copy of a string with all letters converted to lowercase
`str.replace(a, b)`	Replaces all instances of the substring `a` in `str` with the substring `b`
`str.split(a)`	Returns substrings of `str` split at a substring `a`
`str.strip()`	Removes leading and trailing whitespace from `str`

Operate on literals one at a time

```
for s in county_ex:
    lower = s.lower()
    wo_county = lower[:-7]
    def dropchange(l):
        if l == "&":
            return "and"
        elif l == " " or l == ".":
            return ""
        else:
            return l
    final_word = "".join([dropchange(l) for
                          l in wo_county])
    final.append(final_word)
```

Change to
lower case

Drop parish and county
at end of string

Examine **one literal**,
If it's "&" return "and"
If its blank or period
return empty string

Process characters
one at a time
in the string

String manipulation in a DataFrame

```
df = pd.DataFrame(data = county_ex,
                  columns=["County"])
(
    df['County'].str.lower()
    .str.replace("&","and”)
    .str.replace(".","")
    .str.replace(" ", "")

    .str.replace("county","")
    .str.replace("parish","")
)
```

The **replace** method for strings in Pandas accepts a regular expression.

These patterns are single literals: ampersand, period, and blank

Here the pattern is a sequence of 6 literals: **county** or **parish**

The .str property on pandas Series exposes the same string methods as Python does. Calling a method on the .str property calls the method on each item in the series.

(2) Recall the Web log file

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04 HTTP/1.1" 301 328
"http://anson.ucdavis.edu/courses/"  "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)”
```

```
169.237.6.168 - - [8/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"
```

Reading a Web log

Given the formatting, we simply want to read in each record as a string

```
with open("data/smallLog.txt", "r") as f:
    lines = f.readlines()
```

We can place these strings in a column of a data frame:

```
df = pd.DataFrame(pd.Series(lines, name="raw"))
```

One record

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04 HTTP/1.1" 301 328
"http://anson.ucdavis.edu/courses/"  "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)”
```

How can we extract the

- *Day of month*
- *Month*
- *Year*

from the log entry?

What structure in the record is useful?

- *Date is within a left and right square bracket [ ]*
- *Day, month and year are separated by forward slash*
- *Year is separated from time with a colon :*

How can we use these observations to extract the desired information?

Splitting strings – Simple & Useful

- If we split on [and take the second substring we have

```
26/Jan/2014:10:47:8 -0800] "GET /stat141/Winter…
```

- Next, we split this second string on : and take the first substring we have

```
26/Jan/2014
```

- Lastly, we split this first element on / to get the three strings

```
[26, Jan, 2014]
```

Splitting strings – Simple & Useful

```
date_pieces = (df['raw']
               .str.split(r'[').str[1]
               .str.split(r':').str[0]
               .str.split(r'/').str[0:3]
              )
df['day'] = date_pieces.str[0]
df['mon'] = date_pieces.str[1]
df['Yr'] = date_pieces.str[2]
```

Can we do better?

```
date_pieces  = df['raw'].str.split(r'[\[/:]').str[1:4]
```

What the heck is:

[\[\]/:] ?????

A regular expression
(a.k.a. regex)!

Regular Expressions

How does the pattern matching work?

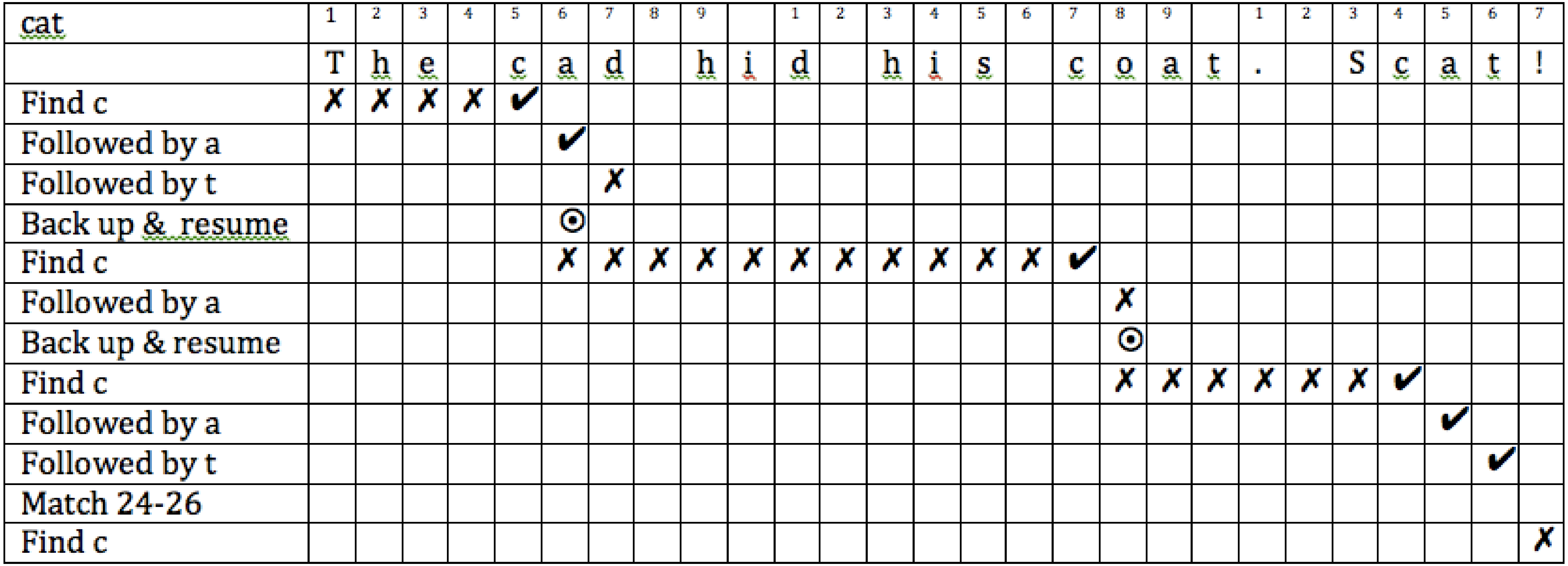
- We have a pattern and a string
- We are looking for the pattern in the string
- Proceed through the string from start to end (left to right) one character at a time
- All of the pattern must be matched
- Not all of the string needs to match the pattern

Pattern: Cat

- Look at each character one at a time, starting at the left.
- When you find a "C", then look at the next character and check to see if it is "a".
- If it is an "a", then look at the next character and check to see if it is "t". If it is, then we have a match!
- At any point if a literal does not match, back up and continue the search for "C" at the character immediately following the first character matched (i.e., the first "C")

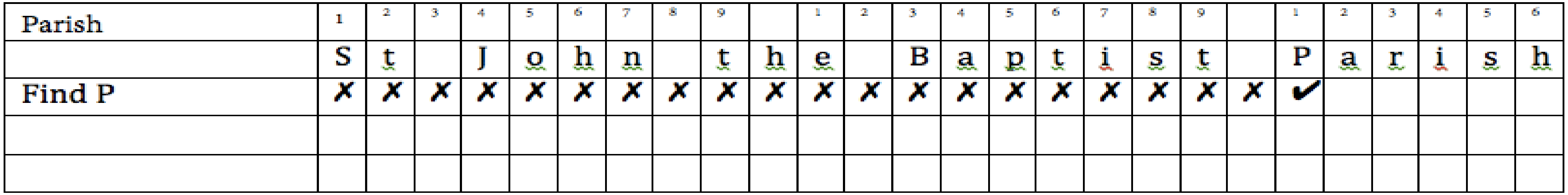
Search String One Character at a Time

PATTERN **STRING**



cat	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7
	T	h	e		c	a	d		h	i	d		h	i	s		c	o	a	t	.		S	c	a	t	!
Find c	✗	✗	✗	✗	✔																						
Followed by a						✔																					
Followed by t							✗																				
Back up & resume						⊙																					
Find c						✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✔										
Followed by a																		✗									
Back up & resume																		⊙									
Find c																		✗	✗	✗	✗	✗	✗	✔			
Followed by a																									✔		
Followed by t																										✔	
Match 24-26																											
Find c																											✗

pattern string

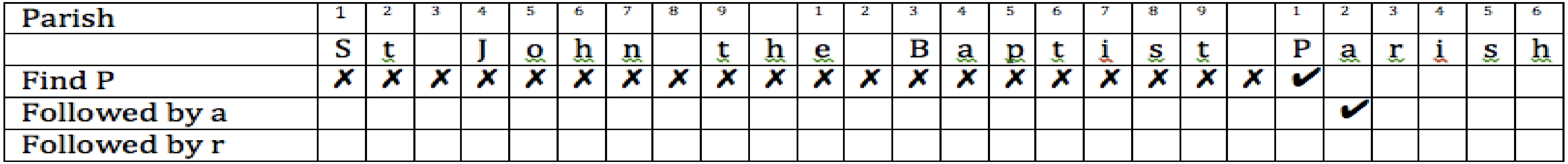


Parish	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6
	S	t		J	o	h	n		t	h	e		B	a	p	t	i	s	t		P	a	r	i	s	h
Find P	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✔					

Search proceeds one literal at a time.

Begin with a search for “P”

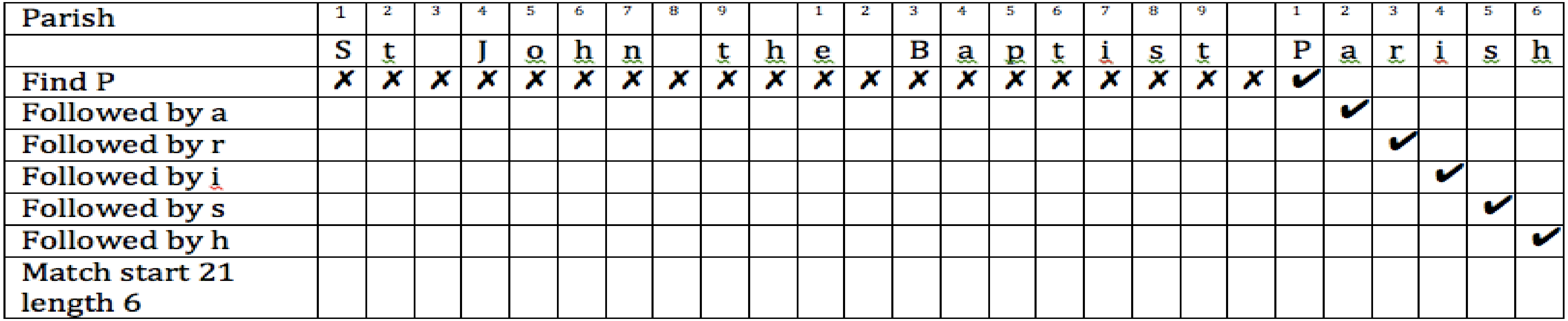
pattern string



Parish	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6
	S	t		J	o	h	n		t	h	e		B	a	p	t	i	s	t		P	a	r	i	s	h
Find P	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓					
Followed by a																						✓				
Followed by r																										

When find the first literal, look for the second immediately following it.

pattern string



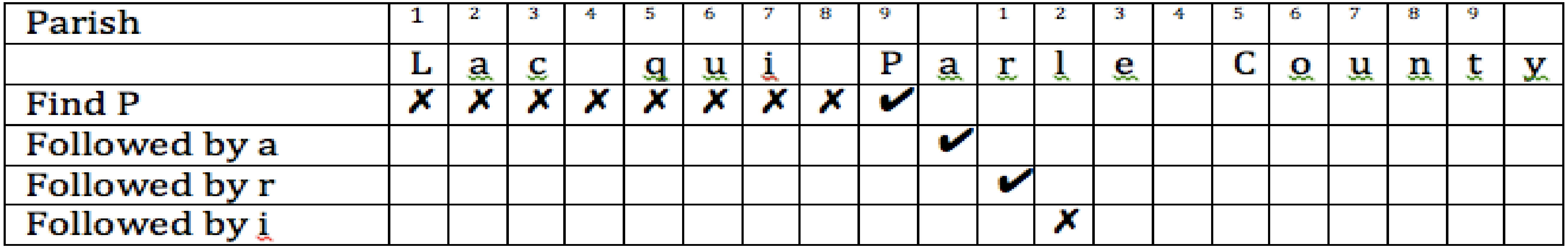
Parish	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6
	S	t		J	o	h	n		t	h	e		B	a	p	t	i	s	t		P	a	r	i	s	h
Find P	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓					
Followed by a																						✓				
Followed by r																							✓			
Followed by i																								✓		
Followed by s																									✓	
Followed by h																										✓
Match start 21 length 6																										

Continue matching the 3rd,4th, 5th, and 6th literals, one after another.

Pattern Matching

Let’s Try Another String

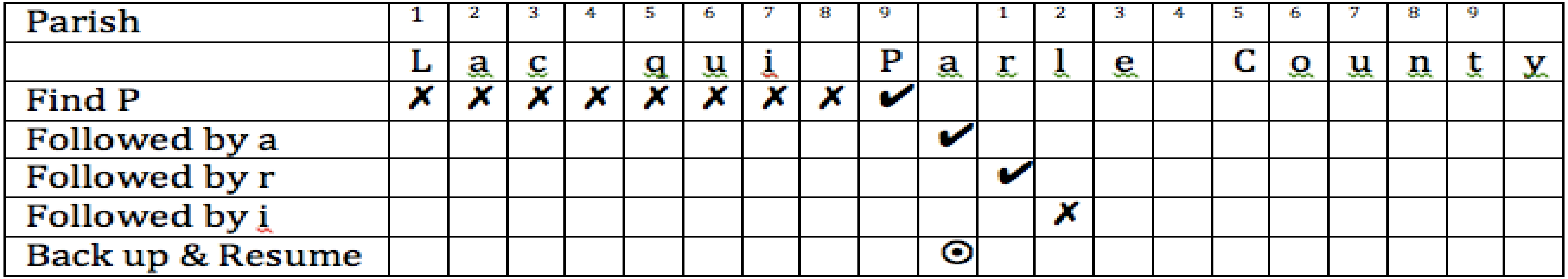
pattern string



Parish	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9	
	L	a	c		q	u	i		P	a	r	l	e		C	o	u	n	t	y
Find P	✗	✗	✗	✗	✗	✗	✗	✗	✓											
Followed by a										✓										
Followed by r											✓									
Followed by i												✗								

Search proceeds one literal at a time.
Begin with a search for “P”.
When find “P”, continue with match for the 2nd literal and so on.

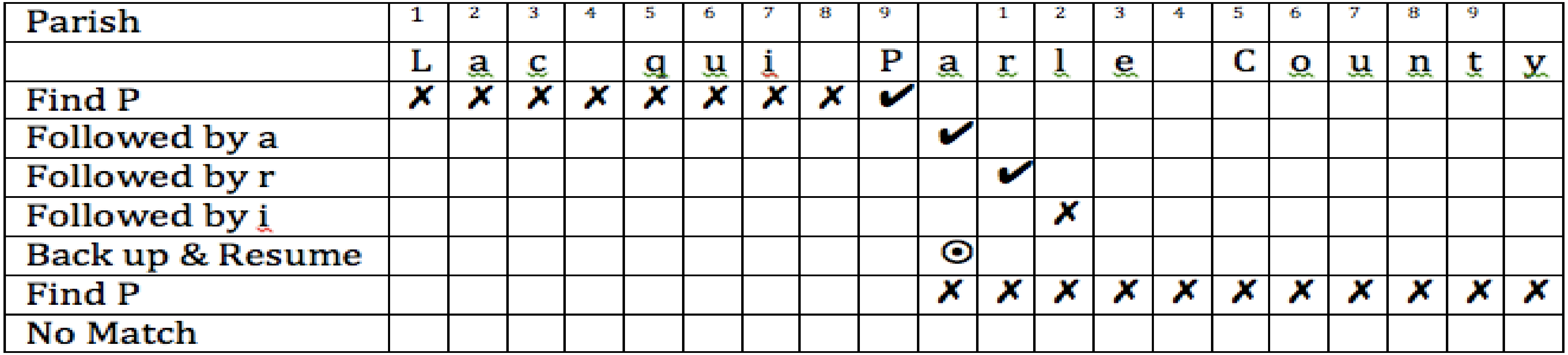
pattern string



Parish	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9	
	L	a	c		q	u	i		P	a	r	l	e		C	o	u	n	t	y
Find P	✗	✗	✗	✗	✗	✗	✗	✗	✓											
Followed by a										✓										
Followed by r											✓									
Followed by i												✗								
Back up & Resume										⊙										

When we get a mismatch,
back up and resume the search for “P” at the literal immediately following our earlier first match.

pattern string



Parish	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9	
	L	a	c		q	u	i		P	a	r	l	e		C	o	u	n	t	y
Find P	✗	✗	✗	✗	✗	✗	✗	✗	✓											
Followed by a										✓										
Followed by r											✓									
Followed by i												✗								
Back up & Resume										⊙										
Find P										✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
No Match																				

The search for the “P” beginning at position 10 doesn’t find a match.

Regular Expressions

- Regular expressions give us a powerful way of matching patterns in text data
- Importantly, we do this all “programatically” rather than by “hand” looping over characters in a string

Literals

- Literals in a pattern match on the character itself in the string
- So far we have worked with literals only

Character Class

Character Class

- Collection of characters that are considered equivalent
- Any single literal in the character class can be matched in the string
- The character class is denoted by [ ]
  - so [2ax3z] will search in the string for a 2 OR a OR x OR 3 OR z
  - Any **one** of these will constitute a match
- We can use a range in the class, e.g., [A-Z] for any capital letter, [0-9] for any digit, and [A-Z0-9] for any capital or digit

Match
‘o’ or ‘a’

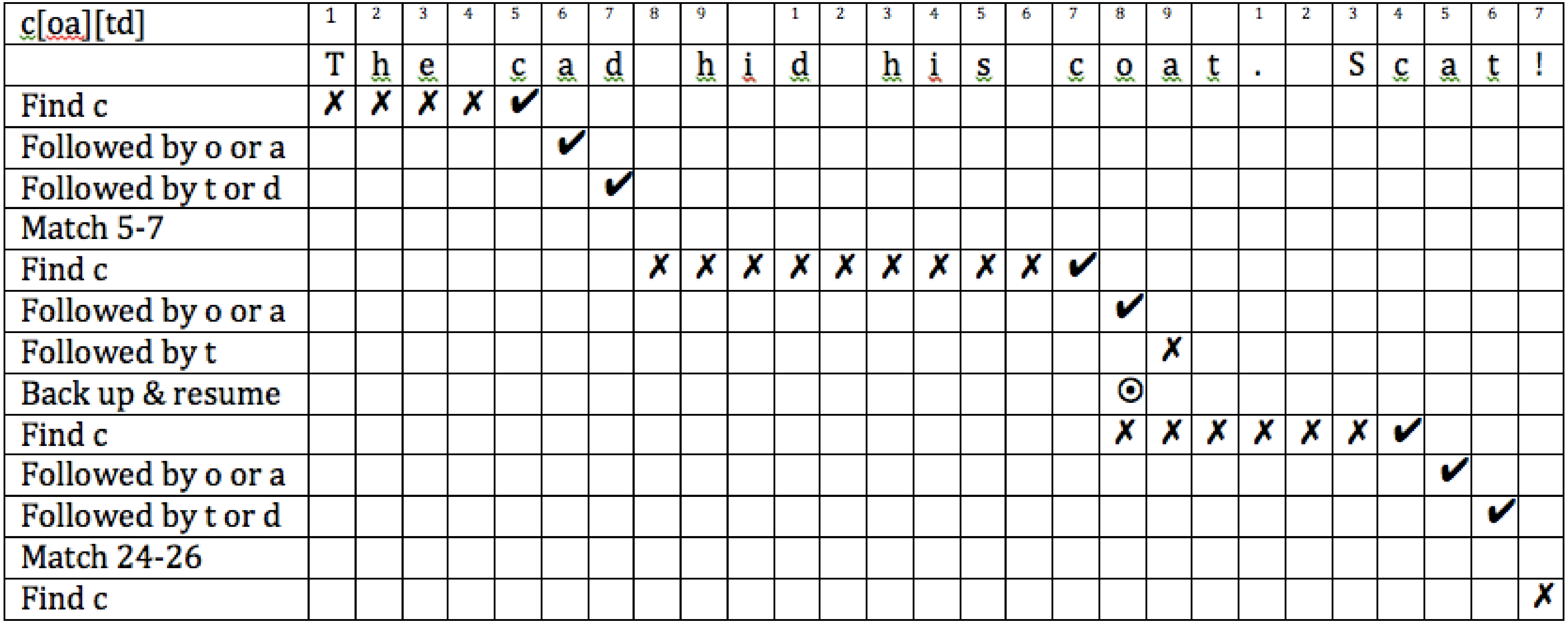
Looking for
3
characters

Equivalent Characters: c[oa][td]

Match
‘t’ or ‘d’

PATTERN

STRING



c[oa][td]	1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7	8	9		1	2	3	4	5	6	7
	T	h	e		c	a	d		h	i	d		h	i	s		c	o	a	t	.		S	c	a	t	!
Find c	✗	✗	✗	✗	✔																						
Followed by o or a						✔																					
Followed by t or d							✔																				
Match 5-7																											
Find c								✗	✗	✗	✗	✗	✗	✗	✗	✗	✔										
Followed by o or a																		✔									
Followed by t																			✗								
Back up & resume																		⊙									
Find c																		✗	✗	✗	✗	✗	✗	✔			
Followed by o or a																									✔		
Followed by t or d																										✔	
Match 24-26																											
Find c																											✗

Remember '[\[/:]'

- The outer square brackets **[…]** says that all of the literals within are equivalent so split on any one of them
- The backslash says treat the next character as a plain character, i.e., not a special one. So **\[** is really a plain (aka literal) **[**
- So the characters within the square brackets are the left bracket **[,** the forward slash **/**, and colon **:** - each is treated as a splitting character.

Remember:

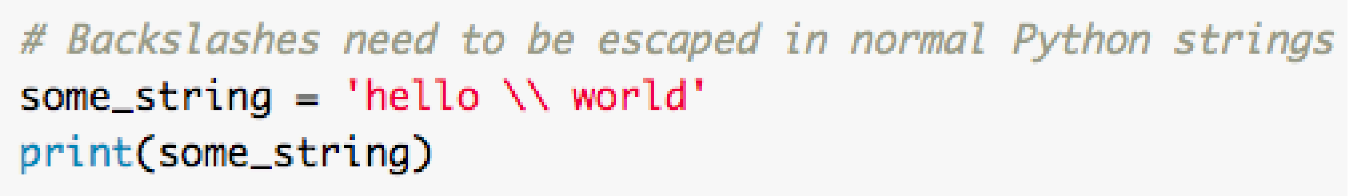
```
date_pieces  = df['raw'].str.split(r'[\[/:]').str[1:4]
```

Now we see that [\[/:] is a character class that matches [OR / OR :

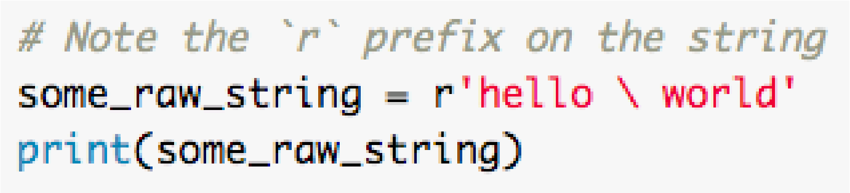
The \ in front of [is used to escape the meaning of [as the start of a character class.

Raw strings

- In Python, regular expressions are most commonly stored as raw strings. Raw strings behave like normal Python strings without special handling for backslashes



```
# Backslashes need to be escaped in normal Python strings
some_string = 'hello \\ world'
print(some_string)
```



```
# Note the `r` prefix on the string
some_raw_string = r'hello \ world'
print(some_raw_string)
```

Remember the county names

```
df = pd.DataFrame(data = county_ex,
                    columns=["County"])

(
    df['County'].str.lower()
    .str.replace("&","and”)
    .str.replace("[. ]","")
    .str.replace("county|parish","")

)
```

Now the pattern **[.]** says either a period or blank is to be replaced by an empty string.

Here the pattern searched for is either **county** or **parish**. This is alternation

Modifiers

Modifiers

- Modifiers operate on literal characters, character classes, or combinations of the two
- Modifiers are used for:
  - Repetition
  - Wild cards
  - Anchors at beginning and ending of string
  - Alternation
  - Group

Meta characters

^	As the first character in the pattern, anchor for the beginning of the string/line e.g. ^[lg]ame matches “lame” and “game” but not the last four characters in “flame” When ^ is the first character in [ ], any character matches except these characters e.g. [^A-Z0-9] matches any single character that’s not a capital letter or number
$	End of string/line anchor e.g. cat$ matches “Scat” and “my black cat” but not “Scat!” and not “my cat is black”

Meta characters can control how many times something is repeated

?	Preceding element *zero or one* time, e.g., ba? matches “b” or “ba”
+	Preceding element *one or more* times, e.g., ba+ matches “ba”, “baa”, “baaa”, and so on, but not “b”
*	Preceding element *zero or more* times, e.g., ba* matches “b”, “ba”, “baa”, and so on.

More Meta characters

<table>
<tr><td>.</td><td>Any single character
e.g. .* matches any character, any number of times (like * as a UNIX wildcard)</td></tr>
<tr><td>[ ]</td><td>Character class
e.g. [a-cx-z] matches “a”, “b”, “c”, “x”, “y”, or “z”</td></tr>
<tr><td>-</td><td>Range within a character class</td></tr>
<tr><td>|</td><td>Alternation, i.e. one subpattern or another
e.g. abc|vwxyz matches “abc” or “vwxyz”</td></tr>
<tr><td>( )</td><td>Identify a subpattern
e.g. ab(cd|x)yz matches “abcdyz” or “abxyz”</td></tr>
</table>

Pattern ”^[^a-z]+$"

Which strings does it match?

1	2	3	4
” ”	”HELP!”	”Hi”	”123”

A. 1 2

B. 1 3

C. 1 4

D. 2 3

E. 2 4

F. 3 4

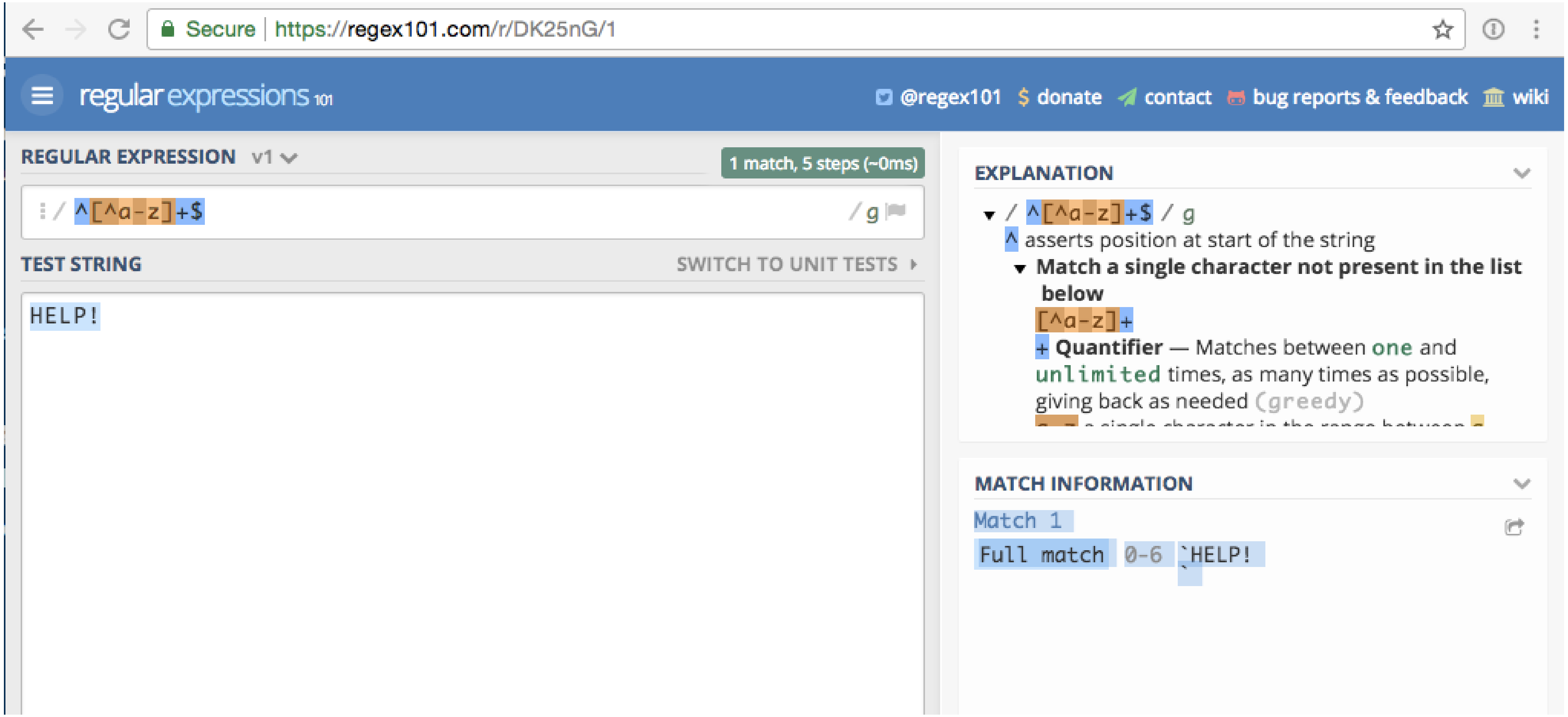
G. 1 2 3

H. 1 2 4

I. 1 3 4

J. 2 3 4

*Online Regular Expression Checkers



REGULAR EXPRESSION v1

1 match, 5 steps (~0ms)

```
/ ^[^a-z]+$ / g
```

TEST STRING

```
HELP!
```

EXPLANATION

/ ^[^a-z]+$ / g

^ asserts position at start of the string

Match a single character not present in the list below

[^a-z]+

+ **Quantifier** — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

a-z a single character in the range between a

MATCH INFORMATION

```
Match 1
Full match  0-6  `HELP!`
```

Pattern ”^[^a-z]+$"

Which strings does it match?

1	2	3	4
” ”	"HELP!”	"Hi”	"123”

A. 1 2

B. 1 3

C. 1 4

D. 2 3

E. 2 4

F. 3 4

G. 1 2 3

H. 1 2 4

I. 1 3 4

J. 2 3 4

Meta characters can represent string-related concepts

\b	Boundary between a word and non-word
\w	A word character, that is a letter, digit, or underscore
\W	A non-word character
\s	White space, including space, new line, return, tab \t, \n, \r tab, new line, and return, respectively
\d	Digit, i.e., [0-9]

Ex: find the username in an email address

Strings

"xyz alice-b@google.com purple”
“yellow john_smith@comcast.net one two 3”
“123! 456- terry123@aol.com”
“mary.zhang@berkeley.edu another email address”

What should the pattern be?

Ex: find the username in an email address

Strings

"xyz alice-b@google.com purple”
“yellow john_smith@comcast.net one two 3”
“123! 456- terry123@aol.com”
“mary.zhang@berkeley.edu another email address”

What should the pattern be?

‘[\w|.|-]+@’

Additional quantifiers

{n}	Preceding item exactly **n** times
{n,}	Preceding item **n** or more times
{n,m}	Preceding item between **n** and **m** times (inclusive)
{,m}	Preceding item up to **m** times

Practice: Indicate which strings contain a match to the pattern

	“hi mabc”	“abc”	“ abcd”	“abccd”	“abcabcdx”	“cab”	“abd”	“cad”
abc								
^abc								
abc.d								
abc+d								
abc?d								
abc$								
abc.*d								
abc?								
a[b?d]								

Practice: Indicate which strings contain a match to the pattern

	“hi mabc”	“abc”	“ abcd”	“abccd”	“abcabcdx”	“cab”	“abd”	“cad”
abc	✓	✓	✓	✓	✓			
^abc								
abc.d								
abc+d								
abc?d								
abc$								
abc.*d								
abc?								
a[b?d]								

Practice: Indicate which strings contain a match to the pattern

Do the
Next 3
rows

	“hi mabc”	“abc”	“ abcd”	“abccd”	“abcabcdx”	“cab”	“abd”	“cad”
abc	✓	✓	✓	✓	✓			
^abc		✓	✓	✓	✓			
abc.d								
abc+d								
abc?d								
abc$								
abc.*d								
abc?								
a[b?d]								

Practice: Indicate which strings contain a match to the pattern

	“hi mabc”	“abc”	“ abcd”	“abccd”	“abcabcdx”	“cab”	“abd”	“cad”
abc	✓	✓	✓	✓	✓			
^abc		✓	✓	✓	✓			
abc.d				✓				
abc+d			✓	✓				
abc?d			✓				✓	
abc$								
abc.*d								
abc?								
a[b?d]								

Complete
The
Table

Practice: Indicate which strings contain a match to the pattern

	“hi mabc”	“abc”	“ abcd”	“abccd”	“abcabcdx”	“cab”	“abd”	“cad”
abc	✓	✓	✓	✓	✓			
^abc		✓	✓	✓	✓			
abc.d				✓				
abc+d			✓	✓				
abc?d			✓				✓	
abc$	✓	✓						
abc.*d			✓	✓	✓			
abc?	✓	✓	✓	✓	✓	✓	✓	
a[b?d]	✓	✓	✓	✓	✓	✓	✓	✓

Food Safety

Top 25 Violations

unclean or degraded floors walls or ceilings

moderate risk food holding temperature

inadequate and inaccessible handwashing facilities

unapproved or unmaintained equipment or utensils

Inadequately cleaned or sanitized food contact surfaces

wiping cloths not clean or properly stored or inadequate sanitizer

improper food storage

foods not protected from contamination

high risk food holding temperature

moderate risk vermin infestation

food safety certificate or food handler card not available

unclean nonfood contact surfaces

permit license or inspection report not posted

inadequate food safety knowledge or lack of certified food safety manager

unclean or unsanitary food contact surfaces

low risk vermin infestation

improper storage of equipment utensils or linens

unclean hands or improper use of gloves

improper or defective plumbing

improper cooling methods

improper thawing methods

Inadequate washing facilities or equipment

high risk vermin infestation

no thermometers or uncalibrated thermometers

improper storage use or identification of toxic substances

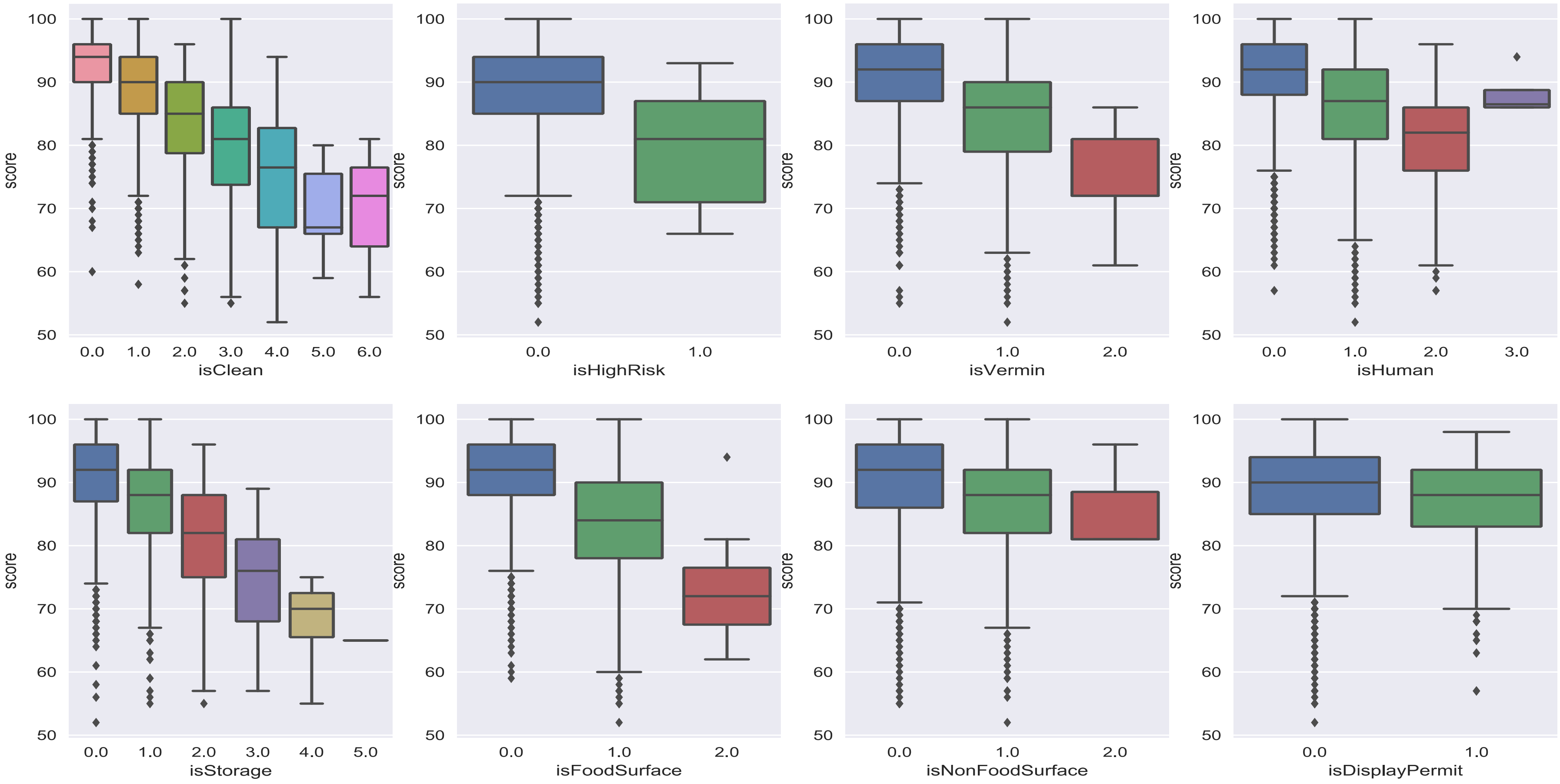
Derive features that capture type of violation

- Vermin
- Risk – high, medium, low
- People – hand, glove, hair, nail
- Food surfaces
- Nonfood surfaces, floor, wall, ceiling
- Storage, thermometer, cooling, thawing
- Equipment
- Permit, certificate

We want to find occurrences of words or parts of words

```
vio['desc'].str.contains(r"clean|sanit")
vio['desc'].str.contains(r"high risk")
vio['desc'].str.contains(r"vermin")
vio['desc'].str.contains(r"wall|ceiling|floor|nonfood surface")
vio['desc'].str.contains(r"hand|glove|hair|nail")
vio['desc'].str.contains(r"\Wfood")
```

How does a violation change the inspection score?



Useful Regular Expression Functions

- split - breaks a string at the supplied pattern
- replace – finds and replaces the pattern with supplied string
- contains – finds the pattern
- extract – extracts the substring that matches the pattern
- count – counts the number of occurrences of the pattern

Useful String Manipulation

- split
- join
- strip, rstrip, lstrip – strips characters from string (or left end or right end of string)
- lower, upper, swapcase

Advice on pattern matching

- Start with simple strings to see what the pattern matches
- If a pattern matches nothing, weaken it by dropping part of the pattern to get too many matches and then tighten incrementally
- Matching is greedy in that a match will be as large as possible. This is especially relevant when using meta characters such as * and +

Greedy Matching Example

- Goal: find html tags such as <p> and <head> in a string

Pattern: “<.*>”

String: “<html> <head> My file </head> <p> This is a para </p> … </html>

State of the Union Address

The first State of the Union Speech

***

State of the Union Address
George Washington
December 8, 1790

Fellow-Citizens of the Senate and House of Representatives:

In meeting you again I feel much satisfaction in being able to repeat my congratulations on the favorable prospects which continue to distinguish our public affairs. The abundant fruits of another year have blessed our country with plenty and with the means of a flourishing commerce. …

Processing with Regular Expressions

- Read speeches as a string
- Divide speeches according to the occurrence of “***”
- Split the speech up at new lines “\n” to extract name and year
- Clean text by turning upper case to lower case and eliminating punctuation

We have seen how to perform these operations.

Now What?

How to analyze speeches?

- We can derive features as with the food safety violation descriptions
- Instead, we can try to analyze the speech as a collection of words
  - Pool together all of the unique words across all of the speeches – *Bag of Words*
  - For each speech, count the number of occurrences of each word in the bag of words – *word vector*
  - We have lost the juxtaposition of words, but there may be something interesting in vector of word counts

Thanks!