PPOL 564: Data science I

Session 06: Basic regex

Where today fits in

- **Last week:**
  - **Exact merging**: have identifier with exact matches between two datasets
- **This week:** basic regular expressions (regex) to help:
  1. Clean join fields for exact matching/merges
  2. Clean join fields for fuzzy/probabilistic matching/merges
- **Later week or optional session:** fuzzy/probabilistic merging/record linkage

Today: basic regex to improve match rates for strings as join keys

- In example below, what if we didn't have the NCES ID numeric identifier? Ways to improve match rates for spelling variations (sometimes called `entity resolution`)

Student	Year	District
Rebecca	2021	New Trier High School
Jennifer	2022	Bethesda High
Jason	2022	Homeschool
⋮		

District	% FRPL
New Trier HS	X%
Bethesda HS	Y%
Arundel HS	Z%
⋮	

Working example

Want to clean school names and classify them into different types (elementary; middle school; high school; charter; alternative; etc...). E.g.:

Central Columbia Ms
Riley County High
Jarrell H S
Trumbull School
SAN GABRIEL ELEMENTARY
Plains El
Pond Hill School
Franklin Elementary
P.S. 119
ANDREW CARNEGIE MIDDLE

Example of variety of names that all match the pattern within `str.contains`

```
cep_optin['is_elem'] =
np.where(cep_optin.schoolname_lower.str.contains("\s+elem",
regex = True), True, False)
```

Examples of True show a lot of variation that could make merges to other data difficult...

paint branch elementary

stewart county elementary school

stove prairie elementary school

winchester avenue elementary school

oak hill elem.

lewis and clark elem.

saunemin elem school

desert springs elementary school

fifth district elementary

linden elementary school

re module provides more flexible alternative to pandas str methods

- Need to import at top: `import re`
- General structure: `re.something(r‘‘somepattern’’, some_str)`
- Why the `r`? Treats string as a raw string without special interpretation of certain characters
- Challenging part is constructing the pattern that captures what you want to capture

First example

- We want to standardize the school names so that if it's an elementary school, it has the string "elemschool" after its other identifiers. E.g.:

original	cleaned
paint branch elementary	paint branch elemschool
stewart county elementary school	stewart county elemschool
stove prairie elementary school	⋮
winchester avenue elementary school	
oak hill elem.	
lewis and clark elem.	
saunemin elem school	
desert springs elementary school	
fifth district elementary	
linden elementary school	

Approach 1 using re.sub

Construct a pattern to match all identified variations you want to sub out. One natural inclination is to try to enumerate the variations and use | (or), e.g.:

```
## define pattern
elem_pattern = r"elementary|elem|elem\\.|elementary school"

## replace in one string
one_str_clean = re.sub(elem_pattern,
                "elemschool", one_str)

## replace for all strings in the column schoolname_lower
all_str_clean = [re.sub(elem_pattern, "elemschool",
                one_str)
                for one_str in df.schoolname_lower]
```

But this leads to issues...

Issues with earlier pattern

(1) Sees `elementary` and subs it out but leaves school in; (2) leaves in . after elem. since matches `elem`

orig_name	**cleaned_name**
paint branch elementary	paint branch elemschool
stewart county elementary school	stewart county elemschool school
stove prairie elementary school	stove prairie elemschool school
winchester avenue elementary school	winchester avenue elemschool school
oak hill elem.	oak hill elemschool.
lewis and clark elem.	lewis and clark elemschool.
saunemin elem school	saunemin elemschool school
desert springs elementary school	desert springs elemschool school
fifth district elementary	fifth district elemschool
linden elementary school	linden elemschool school

Making the pattern more robust to variations

```
## define pattern
elem_pattern_try2 = r"(elem.*)(\s+)?(school)?"
```

Breaking down each component:

- Creating groups using ( ): these help us define groups of characters to look at together
- `elem.*`: matches elem, elem., and elementary (would maybe want to make more restrictive if we had schools named things like element that we didn't want to match)
- Multiple spaces: uses "metacharacter" to match 1+ spaces:

  `\s+`
- ? : **optional pattern**, or match if this pattern occurs but also match if it doesn't (in this case, it's useful since schools like fifth district elementary have nothing afterwards)
- (school)?: similarly, sometimes ends with school and we want to replace; other times just ends with elementary or elem

Executing with re.sub

```python
## define pattern
elem_pattern_try2 = r"(elem.*)(\s+)?(school)?"

## replace for all strings in the column schoolname_lower
all_str_clean_try2 = [re.sub(elem_pattern_try2, "elemschool",
                one_str)
                for one_str in df.schoolname_lower]
```

Better result than our first attempt :)

orig_name	**cleaned_name**
paint branch elementary	paint branch elemschool
stewart county elementary school	stewart county elemschool
stove prairie elementary school	stove prairie elemschool
winchester avenue elementary school	winchester avenue elemschool
oak hill elem.	oak hill elemschool
lewis and clark elem.	lewis and clark elemschool
saunemin elem school	saunemin elemschool
desert springs elementary school	desert springs elemschool
fifth district elementary	fifth district elemschool
linden elementary school	linden elemschool

Other `re` operations

- In previous example, we:
  1. Defined a pattern: different variations of elementary school
  2. Used `re.sub(pattern, replacement, string)` to substitute something else when we match the pattern
- In other examples, we may want to:
  - Define a pattern that characterizes different subparts of a string (e.g., elementary is one part we want to extract; school is another)
  - Match that pattern
  - Extract the matches and do something

First, we can use “metacharacters” or shortcuts to match general types of patterns

https://www.geeksforgeeks.org/python-regex-metacharacters/

Common ones:

- \d or [0-9]: numbers
- [A-Z]: uppercase alpha (any; can do subsets like [ABC])
- [a-z]: lowercase alpha
- \w: alphanumeric (so numbers of letters)
- + : match one or more appearances (e.g., if we have two usernames—rebeccajohnson8; rebeccajohnson88; rebeccajohnson796–could do “[a-z]+\d+” to match all versions)
- *: match anything
- ^: match at beginning
- $: match at end
- {x, y}: match a pattern of length x to length y (e.g., if rather than capture something repeated 1 or more times, we wanted to capture numbers that look like ages and could be length 1-3, write as “\d{1,3}”

Then, three methods that do similar things

1. `re.findall(pattern, string)`
   - **What it does/returns**: scans the string from left to right and returns the matches in a list
   - **Useful for**: parsing a string and then recombining elements (e.g., `elementary` could be list element 0; `school` could be list element 1)
2. `re.search(pattern, string)`
   - **What it does/returns**: scans the entire string and returns the substring(s) regardless of where it appears in the string. If no matches found, returns `None`. If matches found, returns MatchObject
   - **Useful for:** checking *if* there's a match (since can check whether result is equal to None); same use above of getting substrings
3. `re.match(pattern, string)`
   - **What it does/returns:** Operates similarly to `re.search()` but rather than searching the entire string, only returns if found at beginning of string

Both `re.match()` and `re.search()` only return the first appearance of a pattern; if want to return all occurrences (e.g., count how many times "Trump" appears in a single tweet), can use either `re.findall()` or `re.finditer()`

Focusing on `re.findall`: executing

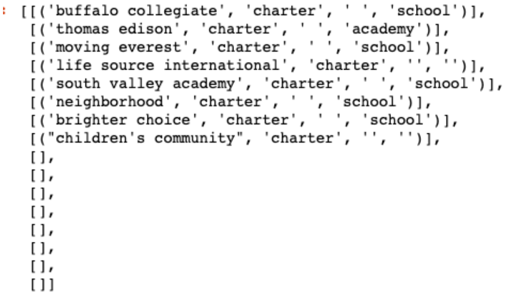
Example: match words before and after the charter pattern

```python
## charter pattern
charter_pattern = r"(.*)\s+(charter)(\s+)?(\w+)?"

## findall
test_charter_findall = [re.findall(charter_pattern,
                        school)
                        for school in combined_examples]
```

What this returns: a list of lists

orig_name
buffalo collegiate charter school
thomas edison charter academy
moving everest charter school
life source international charter
south valley academy charter school
neighborhood charter school of harle
brighter choice charter school-girls
children's community charter
frontier elementary school
columbus humanities, arts and...
okemos public montessori-central
pawhuska es
east valley senior high
glenpool es
number 27
south fork elementary



```
: [[('buffalo collegiate', 'charter', ' ', 'school')],
   [('thomas edison', 'charter', ' ', 'academy')],
   [('moving everest', 'charter', ' ', 'school')],
   [('life source international', 'charter', '', '')],
   [('south valley academy', 'charter', ' ', 'school')],
   [('neighborhood', 'charter', ' ', 'school')],
   [('brighter choice', 'charter', ' ', 'school')],
   [("children's community", 'charter', '', '')],
   [],
   [],
   [],
   [],
   [],
   [],
   [],
   []]
```

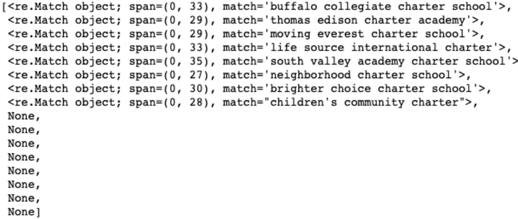
What if we use same pattern and use `re.search` instead on the list of school names? How to execute

```python
## charter pattern
charter_pattern = r"(.*)\s+(charter)(\s+)?(\w+)?"

## search
test_charter_search = [re.search(charter_pattern,
                                 school)
                                 for school in combined_examples]
```

A list of a MatchObject if match exists; None otherwise

orig_name
buffalo collegiate charter school
thomas edison charter academy
moving everest charter school
life source international charter
south valley academy charter school
neighborhood charter school of harle
brighter choice charter school-girls
children’s community charter
frontier elementary school
columbus humanities, arts and...
okemos public montessori-central
pawhuska es
east valley senior high
glenpool es
number 27
south fork elementary



```
[<re.Match object; span=(0, 33), match='buffalo collegiate charter school'>,
 <re.Match object; span=(0, 29), match='thomas edison charter academy'>,
 <re.Match object; span=(0, 29), match='moving everest charter school'>,
 <re.Match object; span=(0, 33), match='life source international charter'>,
 <re.Match object; span=(0, 35), match='south valley academy charter school'>
 <re.Match object; span=(0, 27), match='neighborhood charter school'>,
 <re.Match object; span=(0, 30), match='brighter choice charter school'>,
 <re.Match object; span=(0, 28), match="children's community charter">,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None]
```

MatchObject isn't useful in its own right; how do we extract matches? .group() method on MatchObject

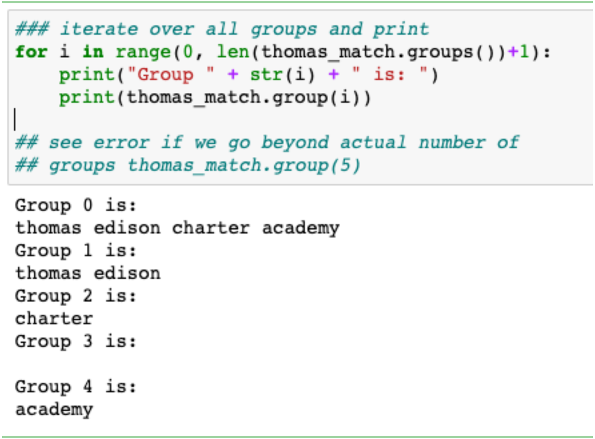
Example with: `thomas edison charter academy`

```
### here, we're just focusing on the 2nd match (thomas edison charter academy)
### and we're getting the first group from that match
thomas_match = test_charter_search[1]
thomas_match

### example where we're just getting the first group
### (name of school before charter)
thomas_firstgroup = thomas_match.group(1)
```

Prints: `thomas edison`

Generalizing to show all groups for that one match



```
### iterate over all groups and print
for i in range(0, len(thomas_match.groups())+1):
    print("Group " + str(i) + " is: ")
    print(thomas_match.group(i))

## see error if we go beyond actual number of
## groups thomas_match.group(5)
```

```
Group 0 is: 
thomas edison charter academy
Group 1 is: 
thomas edison
Group 2 is: 
charter
Group 3 is: 

Group 4 is: 
academy
```

Break for activity

ME
Why can't you just be normal?
regex
^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$

07_regex_blank.ipynb