QSS20: Modern Statistical Computing

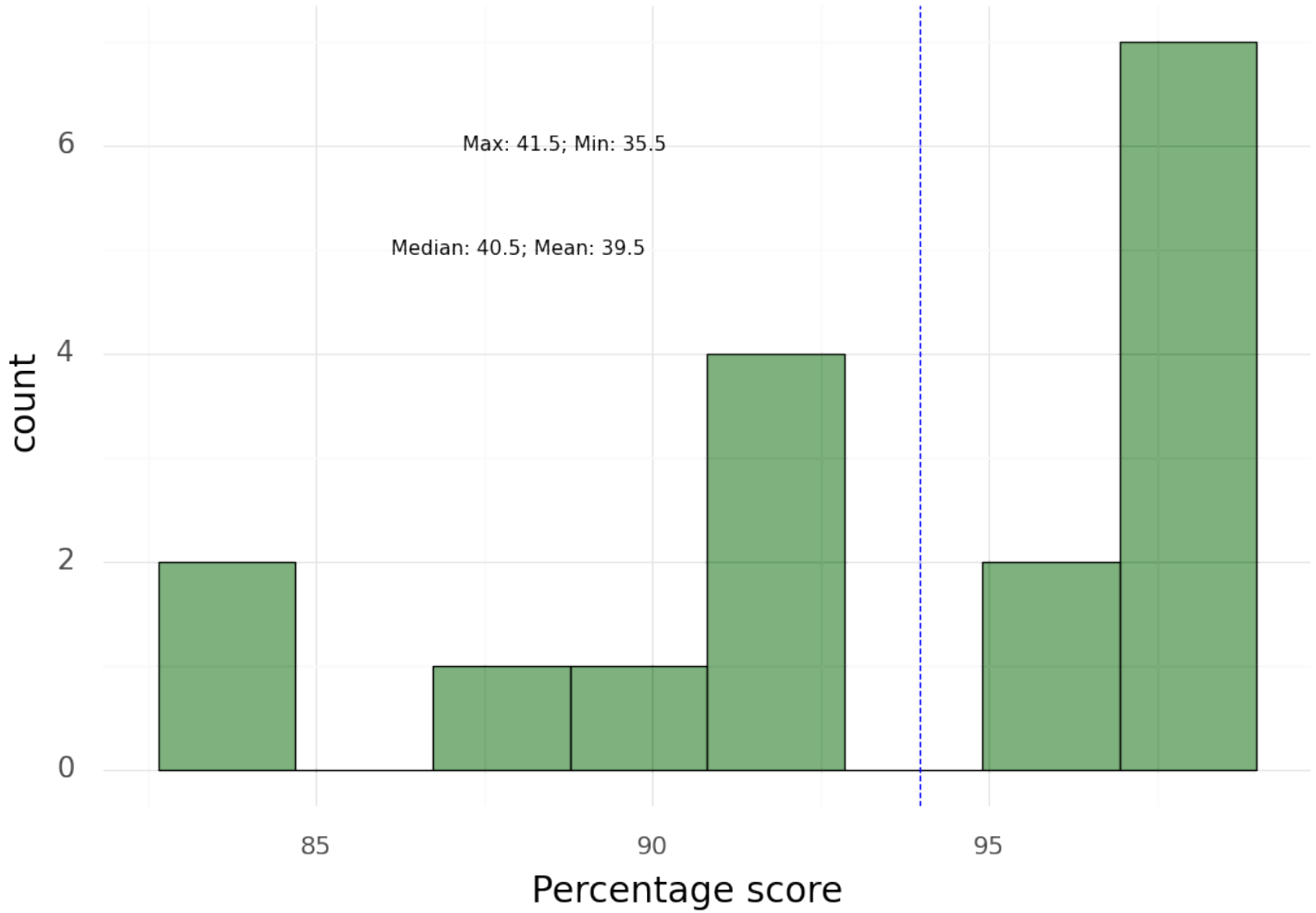
Unit 07: Basic regular expressions

Goals for today

- **Pset 1 feedback**
- Recap of exact merging
- Regex lecture & activity
- Plan ahead with final project groups (if time)

Pset 1 grade distribution

Median: 40.5%; **Mean:** 39.5%; **Max:** 41.5; **Min:** 35.5



Code to produce previous slide (for reference)

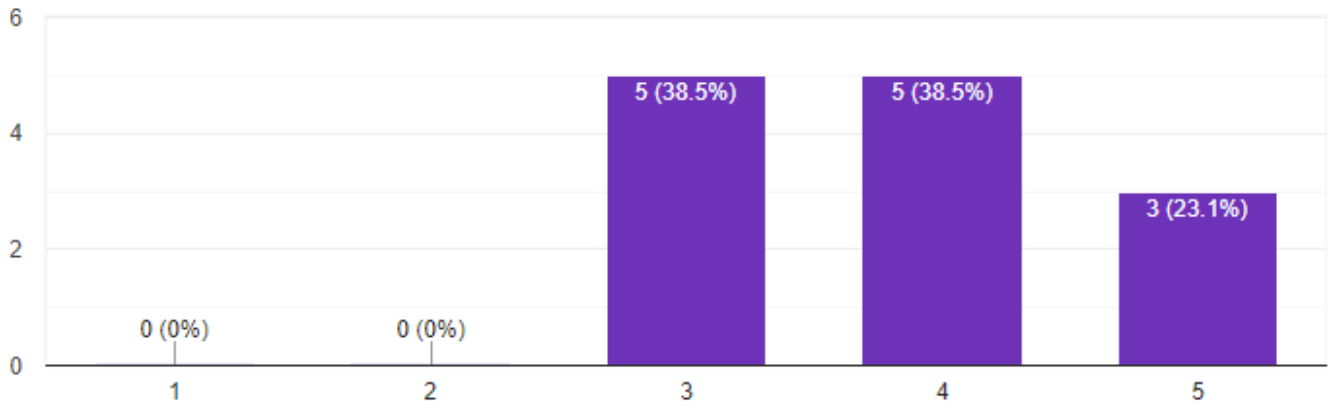
```
(ggplot(grades_df, aes(x = 'pset1_prop')) +
geom_histogram(alpha = 0.5,
               color = "black",
               bins = 8,
               fill = 'darkgreen') +
theme_minimal(base_size = 20) +
xlab("Percentage score") +
geom_vline(xintercept = grades_df['pset1_prop'].mean(),
           linetype = "dashed",
           color = "blue") +
annotate("text", x = 88,
         y = 5,
         label = "Median: 40.5; Mean: 39.5") +
annotate("text", x = 88.7,
         y = 6,
         label = "Max: 41.5; Min: 35.5") +
theme(axis_text_x = element_text(size = 14)))
```

Pset 1 feedback (1/4)

How would you rate the difficulty of problem set 1?

Copy

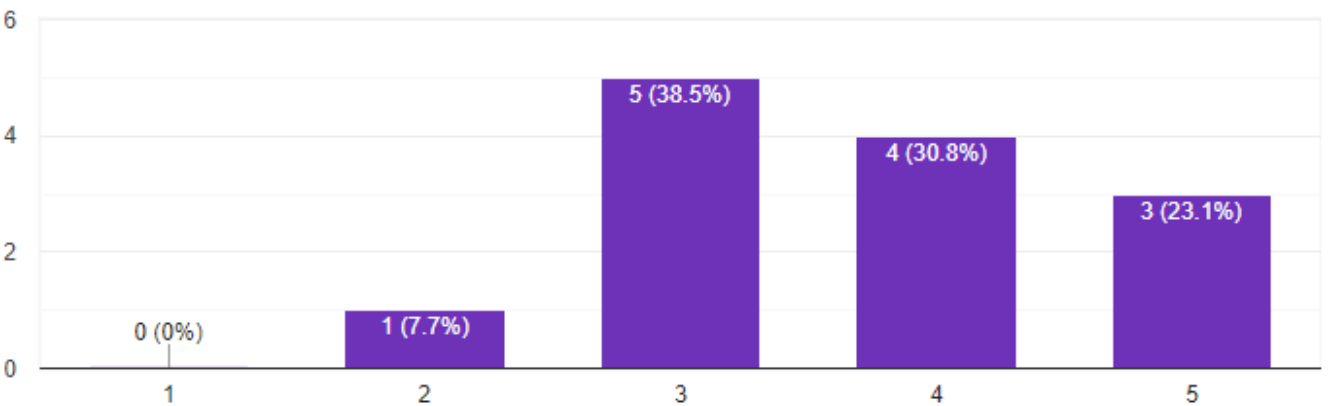
13 responses



How would you rate the overall pace of the course?

Copy

13 responses

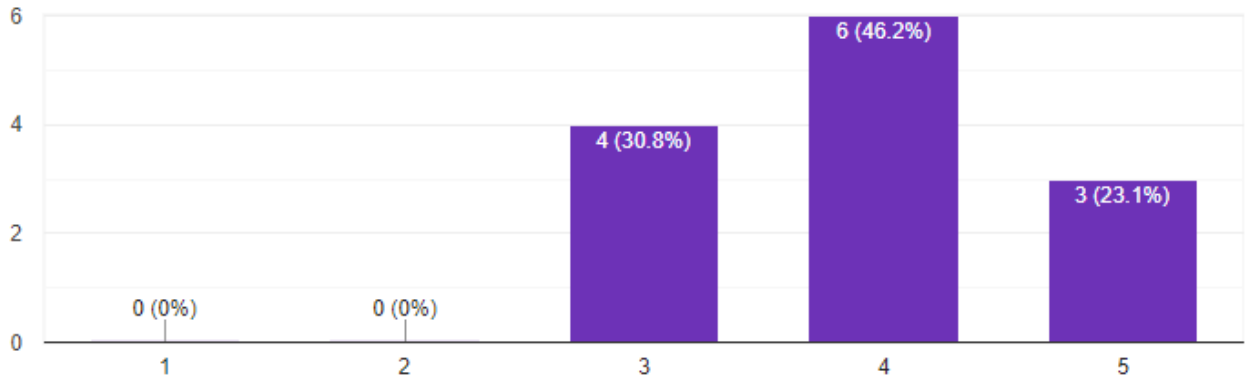


Pset 1 feedback (2/4)

How much do you agree with this statement:
"I am learning a lot from this course!"

Copy

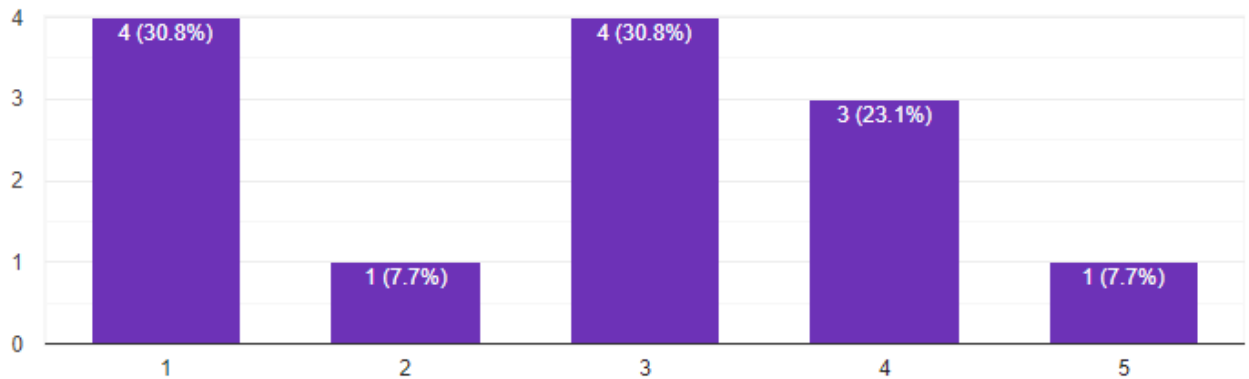
13 responses



How much do you agree with this statement:
"I don't think I have the necessary background to do well in this course."

Copy

13 responses

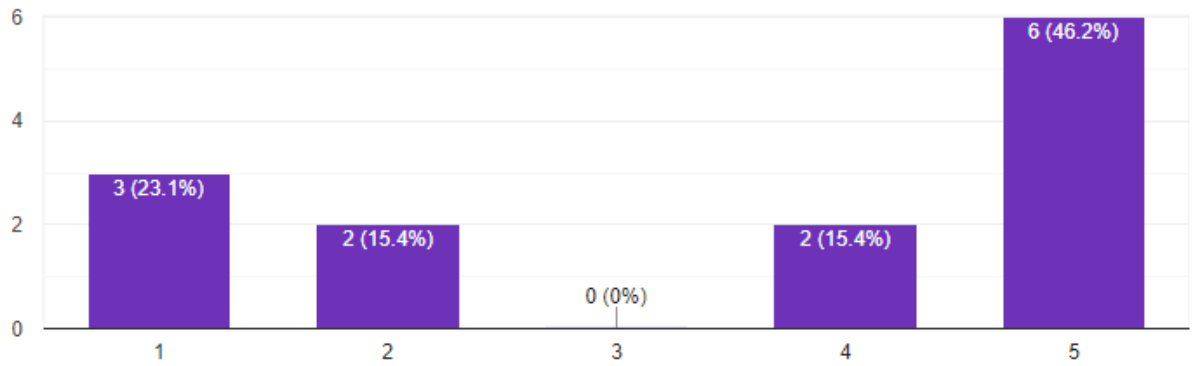


Pset 1 feedback: Collaboration (3/4)

How much do you agree with this statement:
"Collaborating with a classmate resulted in a higher quality submission of pset 1."

Copy

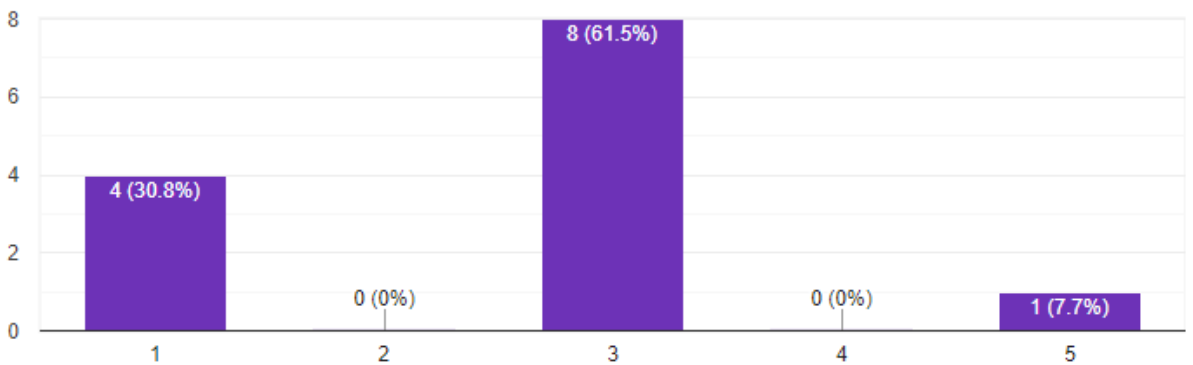
13 responses



Between you and your classmate/partner, who did more work on your submission of pset 1?
(If it was balanced, choose the middle button, 3.)

Copy

13 responses



Pset 1 feedback: Additional text comments (4/4)

What's working:

- DataCamp; exercises with real data
- Collaboration/camaraderie; partner psets
- Recaps

What's not working:

- Partner psets (make partner work optional?)
- DataCamp too heavy/not helpful
- Too much lecture

New ideas:

- In-class projects or psets
- Reviews/exercises as recaps
- Function dictionary
- More hints/tips on expected output in activities/psets
- Better guidance on setup phase

Where we are

- Pset 1 feedback
- **Recap of exact merging**
- Regex lecture & activity
- Plan ahead with final project groups (if time)

Recap of exact merging

What do you remember?

Recap of exact merging

- Tips:
  - Main data on left, aux on right
  - Ideally, unique join key with same name in main/aux (can also use multiple cols)
  - Four main kinds of joins: inner (shared keys), outer (all), left, right
  - Check # rows before & after merge
  - To debug, check for spelling variations in join key
- Useful commands:

```
pd.merge(maindf, auxdf,
         on='joinkey') # defaults to inner
pd.merge(maindf, auxdf, on='joinkey',
         how='outer') # drop no rows
pd.merge(maindf, auxdf, left_on='leftkey',
         right_on='rightkey') # diff keys
indicator = 'unit_mergesource' # merge diagnostic
suffixes = ('_main', '_aux')
  # if same non-join cols, override '_x', '_y'
```

Where we are

- Pset 1 feedback
- Recap of exact merging
- **Regex lecture & activity**
- Plan ahead with final project groups (if time)

Goal for next few sessions

- Exact matching: types of joins
  - Inner joins
  - Outer joins
  - Left joins
  - Right joins
- **Basic regex for two purposes:**
  1. Clean join fields for exact matching/merges
  2. Clean join fields for fuzzy/probabilistic matching/merges
- Fuzzy/probabilistic matching and merges

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	Hanover High
Jason	2022	Homeschool
⋮		

District	**% FRPL**
New Trier HS	X%
Hanover HS	Y%
Lebanon 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)`
- Challenging part is constructing the pattern that captures what you want to capture

Group up and open activity notebook

Work with your partner for pset 2!

Follow along the first part (before group activity) of
05_basicregex_blank.ipynb

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: `re.sub` with stubborn listing

Basic syntax:

`re.sub(pattern, str_to_sub_in, str_to_match)`

Step 1: Construct a pattern to match all identified variations for which you want to substitute. Regex is very flexible, so there are *many ways to do this!* To start out, let's write out all the variations we can think of and stubbornly include them all with | (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 exhaustive approach leads to issues...

Limits to trying to include each option separately

(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

A better idea: Using "metacharacters" or shortcuts to match general types of patterns

The basic 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 several usernames—jhaber-zz; jhaber-zzz; jhaber-zzzz—could do "`[a-z]+-z+`" to match all versions)
- `*`: match any number
- `^`: match at beginning of string or line
- `$`: match at end of string or line
- `{x, y}`: match a pattern of length between x and y (e.g., to capture numbers that look like ages and could be length 1-3, write as "`\d{1,3}`"

Using metacharacters to make the previous 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 re.sub

```
## 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]

## then assign the resulting list to the df
df['cleaned_name_try2'] = all_str_clean_try2

## could easily use same approach to create new column directly
df['cleaned_name_try2'] = df['schoolname_lower'].apply(
                    lambda name:
                    re.sub(elem_pattern_try2, "elemschool", name))
```

How does the result compare to our first, more exhaustive method?

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 the string 'elementary'
  2. Used `re.sub(pattern, replacement, string)` to substitute the target string with something else (in this case, 'elemschool') when we match the pattern
- In other examples, we may want to:
  - Define a pattern that characterizes different subparts of a string (e.g., maybe 'elementary' is one part we want to extract, and 'school' is another)
  - Match that pattern
  - Extract the matches and do something

Three similar matching methods for regex

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()`

Executing `re.findall`

Example: match words before and after the word 'charter'

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

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

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', '', '')],
 [],
 [],
 [],
 [],
 [],
 [],
 [],
 []]
```

Executing `re.search` using same pattern and school names list

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

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

`re.search` returns `MatchObject` if match found, else `None`

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
 <re.Match object; span=(0, 29), match='thomas edison char
 <re.Match object; span=(0, 29), match='moving everest cha
 <re.Match object; span=(0, 33), match='life source intern
 <re.Match object; span=(0, 35), match='south valley acade
 <re.Match object; span=(0, 27), match='neighborhood chart
 <re.Match object; span=(0, 30), match='brighter choice ch
 <re.Match object; span=(0, 28), match="children's communi
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None]
```

Extracting matches from `MatchObject` with `.group()` method

Example: `thomas edison charter academy`

```
### here, we're just focusing on the 3rd match = 6th entry (thomas
    edison charter academy)
### and we're getting the first group from that match
thomas_match = test_charter_search[5]
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`

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))
```

Prints:

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

Group 4 is:
academy
```

Can also extract the groups as a tuple:

```
thomas_match.groups()
```

```
('thomas edison', 'charter', ' ', 'academy')
```

Break for activity

Work with your partner for pset 2!

Group activity section at end of 05_basicregex_blank.ipynb

Where we are

- Pset 1 feedback
- Recap of exact merging
- Regex lecture & activity
- **Plan ahead with final project groups (if time)**

Final project groups

Project	Partner A	Partner B	Partner C	Partner D
SIP: SIRS	Andrew Cho	~~Nate Haile~~	Justin Sapun	Anish Sikhinam
SIP: Medical IDD training	Saige Gitlin	Kayla Hamann	Rachael Williams	
SIP: Medical IDD training	Emma Johnson	Omario Corral-Willians	Max Konzerowsky	
Felony sentencing	Luca D'Ambrosio	Filippo de Min	Nick Romans	Daniel Xu
Felony sentencing	Daniel Céspedes	Giulio Frey	Andy Ilie	