```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Extract and organize metadata and text of PCD

@author: chadheilig


Main product:
"""

#%% Import modules and set up environment
# import from 0_cdc-corpora-header.py

import time
from dateutil.parser import parse
import copy

os.chdir('/Users/chadheilig/cdc-corpora/_test')
PCD_BASE_PATH_u3 = normpath(expanduser('~/cdc-corpora/pcd_u3/'))

# PCD DataFrame, reduced to 2 columns for articles only
pcd_dframe = pickle.load(open('pickle-files/pcd_dframe.pkl', 'rb'))
pcd_art_frame = pcd_dframe.loc[pcd_dframe.level == 'article', 'mirror_path':'string']
pcd_art_frame.index = range(3542)
# [3542 rows x 2 columns]

#%% Read HTML from mirror into list of strings

# pcd_art_html = [read_uni_html(PCD_BASE_PATH_u3 + path)
#                       for path in tqdm(pcd_art_frame.mirror_path)]
# 3237/3237 [00:09<00:00, 326.39it/s]
# pickle.dump(pcd_art_html, open('pcd_art_html.pkl', 'wb'))
# pcd_art_html = pickle.load(open('pcd_art_html.pkl', 'rb'))

pcd_art_html = [html_reduce_space_u(read_uni_html(PCD_BASE_PATH_u3 + path))
                      for path in tqdm(pcd_art_frame.mirror_path)]
# 3542/3542 [00:22<00:00, 156.17it/s]
# pickle.dump(pcd_art_html, open('pcd_art_html.pkl', 'wb'))
# pcd_art_html = pickle.load(open('pcd_art_html.pkl', 'rb'))

# Storing all soup works for PCD because of its size
only_body = SoupStrainer(name='body')
pcd_art_soup = [BeautifulSoup(html, 'lxml', parse_only=only_body)
                for html in tqdm(pcd_art_html)]
# 3542/3542 [01:45<00:00, 33.50it/s]

# attempting to dump as pickle exceeds recursion depth
# crashes with sys.setrecursionlimit(50000)


#%% 3 exemplars to examine

# https://www.cdc.gov/pcd/issues/2019/19_0035.htm
# https://www.cdc.gov/pcd/issues/2014/14_0047.htm
# https://www.cdc.gov/pcd/issues/2014/14_0047_es.htm
```

```
 57 # https://www.cdc.gov/pcd/issues/2011/jul/10_0177.htm
 58 # https://www.cdc.gov/pcd/issues/2011/jul/10_0177_es.htm
 59 # https://www.cdc.gov/pcd/issues/2011/jan/09_0236.htm
 60 # https://www.cdc.gov/pcd/issues/2011/jan/09_0236_es.htm
 61 # https://www.cdc.gov/pcd/issues/2004/jan/03_0005.htm
 62
 63 with pd.option_context("display.max_colwidth", 36):
 64    display(pd.concat([pcd_art_frame.loc[pcd_art_frame.mirror_path.str.contains(stem)]
 65       for stem in ('19_0035', '14_0047', '10_0177', '09_0236', '03_0005')]))
 66 #                               mirror_path                                  string
 67 # 73           /pcd/issues/2019/19_0035.htm  Chronic Obstructive Pulmonary Di...
 68 # 968          /pcd/issues/2014/14_0047.htm  Characteristics of the Built Env...
 69 # 2533      /pcd/issues/2014/14_0047_es.htm  Características del entorno cons...
 70 # 1554     /pcd/issues/2011/jul/10_0177.htm  Forecasting Diabetes Prevalence ...
 71 # 2900  /pcd/issues/2011/jul/10_0177_es.htm  Proyección de la prevalencia de ...
 72 # 1645     /pcd/issues/2011/jan/09_0236.htm  Systems-Level Smoking Cessation ...
 73 # 2963  /pcd/issues/2011/jan/09_0236_es.htm  Actividades para la cesación del...
 74 # 2488     /pcd/issues/2004/jan/03_0005.htm  Osteoporosis and Health-Related ...
 75
 76 # pcd_art_frame.iloc[[73, 968, 2533, 1554, 2900, 1645, 2963, 2488], :]
 77
 78 with open('19_0035-pretty.html', 'w') as file_out:
 79    file_out.write(html_prettify_u(pcd_art_html[73]))
 80 with open('14_0047-pretty.html', 'w') as file_out:
 81    file_out.write(html_prettify_u(pcd_art_html[968]))
 82 with open('14_0047_es-pretty.html', 'w') as file_out:
 83    file_out.write(html_prettify_u(pcd_art_html[2533]))
 84 with open('10_0177-pretty.html', 'w') as file_out:
 85    file_out.write(html_prettify_u(pcd_art_html[1554]))
 86 with open('10_0177_es-pretty.html', 'w') as file_out:
 87    file_out.write(html_prettify_u(pcd_art_html[2900]))
 88 with open('09_0236-pretty.html', 'w') as file_out:
 89    file_out.write(html_prettify_u(pcd_art_html[1645]))
 90 with open('09_0236_es-pretty.html', 'w') as file_out:
 91    file_out.write(html_prettify_u(pcd_art_html[2963]))
 92 with open('03_0005-pretty.html', 'w') as file_out:
 93    file_out.write(html_prettify_u(pcd_art_html[2488]))
 94 del file_out
 95
 96 #%% Focus on elements in <head>
 97
 98 only_head = SoupStrainer(name='head')
 99 x = BeautifulSoup(pcd_art_html[73], 'lxml').head # 2019/19_0035.htm
100
101 def pcd_soup_head_count(soup):
102    """Process selected metadata from HTML <head> element.
103    Using SoupStrainer makes this even more efficent."""
104    citation_author = len(soup.find_all('meta', attrs={'name': 'citation_author'}))
105    citation_categories = len(soup.find_all('meta', attrs={'name': 'citation_categories'}))
106    citation_doi = len(soup.find_all('meta', attrs={'name': 'citation_doi'}))
107    citation_issn = len(soup.find_all('meta', attrs={'name': 'citation_issn'}))
108    citation_journal_abbrev = len(soup.find_all('meta', attrs={'name':
108    'citation_journal_abbrev'}))
109    citation_publication_date = len(soup.find_all('meta', attrs={'name':
109    'citation_publication_date'}))
110    citation_title = len(soup.find_all('meta', attrs={'name': 'citation_title'}))
```

```
111     citation_volume = len(soup.find_all('meta', attrs={'name': 'citation_volume'}))
112     dc_date = len(soup.find_all('meta', attrs={'name': 'DC.date'}))
113     description = len(soup.find_all('meta', attrs={'name': 'description'}))
114     keywords = len(soup.find_all('meta', attrs={'name': 'keywords'}))
115     canonical_link = len(soup.find_all('link', attrs={'rel': 'canonical'}))
116     return dict(citation_author=citation_author,
117        citation_categories=citation_categories, citation_doi=citation_doi,
118        citation_issn=citation_issn, citation_journal_abbrev=citation_journal_abbrev,
119        citation_publication_date=citation_publication_date,
120        citation_title=citation_title, citation_volume=citation_volume,
121        dc_date=dc_date, description=description, keywords=keywords,
122        canonical_link=canonical_link)
123
124 pcd_soup_head_count(x)
125
126 y = pd.DataFrame([pcd_soup_head_count(BeautifulSoup(html, 'lxml', parse_only=only_head))
127                 for html in tqdm(pcd_art_html)])
128 # 3237/3237 [00:51<00:00, 63.34it/s]
129 y.to_excel('pcd_soup_head_count.xlsx', engine='openpyxl')
130
131 # citation_categories is article type
132 # citation_issn is fixed at '1545-1151'
133 # citation_journal_abbrev is 'Prev Chronic Dis' (2012-2014)
134 #    or 'Prev. Chronic Dis.' (2015-2020)
135 # description is 1 of 2 fixed values
136 #    Preventing Chronic Disease (PCD) is a peer-reviewed electronic journal established by
136 the National Center for Chronic Disease Prevention and Health Promotion. PCD provides an
136 open exchange of information and knowledge among researchers, practitioners, policy
136 makers, and others who strive to improve the health of the public through chronic disease
136 prevention.
137 #    Preventing Chronic Disease (PCD) is a peer-reviewed electronic journal established by
137 the National Center for Chronic Disease Prevention and Health Promotion. PCD provides an
137 open exchange on the very latest in chronic disease prevention, research findings, public
137 health interventions, and the exploration of new theories and concepts.
138 # DC.date is not useful
139
140 def pcd_soup_head(soup):
141     """Process selected metadata from HTML <head> element.
142     Using SoupStrainer makes this even more efficent."""
143     title = '' if soup.title.string is None else soup.title.string
144     citation_author = soup.find('meta', attrs={'name': 'citation_author'})
145     citation_author = '' if citation_author is None else \
146        citation_author.get('content').strip()
147     citation_categories = soup.find('meta', attrs={'name': 'citation_categories'})
148     citation_categories = '' if citation_categories is None else \
149        citation_categories.get('content').strip()
150     citation_doi = soup.find('meta', attrs={'name': 'citation_doi'})
151     citation_doi = '' if citation_doi is None else citation_doi.get('content')
152     citation_publication_date = soup.find('meta', attrs={'name':
152     'citation_publication_date'})
153     citation_publication_date = '' if citation_publication_date is None else \
154        citation_publication_date.get('content')
155     citation_title = soup.find('meta', attrs={'name': 'citation_title'})
156     citation_title = '' if citation_title is None else \
157        citation_title.get('content').strip()
158     citation_volume = soup.find('meta', attrs={'name': 'citation_volume'})
```

```
159     citation_volume = '' if citation_volume is None else citation_volume.get('content')
160     keywords = soup.find('meta', attrs={'name': 'keywords'})
161     keywords = '' if keywords is None else \
162        re.sub('(, )+', '|', keywords.get('content')).strip()
163     canonical_link = soup.find('link', attrs={'rel': 'canonical'})
164     canonical_link = '' if canonical_link is None else canonical_link.get('href')
165     return dict(
166        title_head=title, type_head=citation_categories,
167        citation_doi=citation_doi, canonical_link=canonical_link,
168        citation_title=citation_title, citation_author=citation_author,
169        citation_publication_date=citation_publication_date,
170        citation_volume=citation_volume,
171        keywords=keywords)
172
173 pcd_soup_head(x)
174
175 pcd_head_data = [pcd_soup_head(BeautifulSoup(html, 'lxml', parse_only=only_head))
176                  for html in tqdm(pcd_art_html)]
177 # 3542/3542 [00:57<00:00, 61.27it/s]
178 # pickle.dump(pcd_head_data, open("pcd_head_data.pkl", "wb"))
179 # pd.DataFrame(pcd_head_data).to_excel('pcd_head_data.xlsx', engine='openpyxl')
180 # pcd_head_data = pickle.load(open("pcd_head_data.pkl", "rb"))
181
182 #%% Focus on elements in <body>
183
184 # most PCD files do not have a <main> element
185 # only_main = SoupStrainer(name='main') # contains main body of article
186 x = BeautifulSoup(pcd_art_html[73], 'lxml') # 2019/19_0035.htm
187
188 # Inspection of HTML <body>s shows which elements contain content and metatdata
189 # of interest
190
191 # Article segments largely delimited by variations of 'Back to top'
192 # Systematically assemble elements that might serve as these delimiters
193
194 #%% 1. Find <a>, <p>, and <span> elements whose text content contains word top
195
196 za_toptext = [tag for soup in tqdm(pcd_art_soup)
197                   for tag in soup.find_all('a', href=True)
198                   if tag.get_text() is not None and
199                      re.search(r'\btop\b', tag.get_text(), re.I)]
200 zp_toptext = [tag for soup in tqdm(pcd_art_soup)
201                   for tag in soup.find_all('p')
202                   if tag.find('a', href=True, recursive=False) is not None and
203                      tag.get_text() is not None and
204                      re.search(r'\btop\b', tag.get_text(), re.I)]
205 zspan_toptext = [tag for soup in tqdm(pcd_art_soup)
206                   for tag in soup.find_all('span')
207                   if tag.find('a', href=True, recursive=False) is not None and
208                      tag.get_text() is not None and
209                      re.search(r'\btop\b', tag.get_text(), re.I)]
210
211 len(set([str(tag) for tag in za_toptext])) # 11
212 len(set([tag.get_text('|', strip=True) for tag in za_toptext])) # 7
213 len(set([str(tag) for tag in zp_toptext])) # 71
214 len(set([tag.get_text('|', strip=True) for tag in zp_toptext])) # 65
```

```
215 len(set([str(tag) for tag in zspan_toptext])) # 2
216 len(set([tag.get_text('|', strip=True) for tag in zspan_toptext])) # 1
217
218 # za_toptext
219 z = za_toptext # 21957
220 z_str = sorted([str(tag) for tag in z])
221 z_text = sorted([tag.get_text('|', strip=True) for tag in z])
222 { item: z_str.count(item) for item in sorted(set(z_str)) } # 11
223 { item: z_text.count(item) for item in sorted(set(z_text)) } # 7
224 # delete 3 entries (4 occurrences) with '"top 10"' and 'top priority'
225 # 21037 <a> texts contain ['Back to top', 'Top', 'Top of Page']
226
227 {'<a class="psmall" href="#top"> Back to top </a>': 1,
228  '<a class="tp-link-policy" href="#"> Top </a>': 8,
229  '<a href="#"> Back to top </a>': 7580,
230  '<a href="#"> Top </a>': 8071,
231  '<a href="#"> Top of Page </a>': 5369,
232  '<a href="#top"> Back to top </a>': 3,
233  '<a href="#ttop"> Back to top </a>': 5,
234  '<a href="/pcd/for_authors/top_five.htm"> Top 20 Manuscript Problems </a>': 916}
235 {'Back to top': 7589,
236  'Top': 8079,
237  'Top 20 Manuscript Problems': 916,
238  'Top of Page': 5369}
239
240 # zspan_toptext
241 z = zspan_toptext # 16
242 z_str = sorted([str(tag) for tag in z])
243 z_text = sorted([tag.get_text('|', strip=True) for tag in z])
244 { item: z_str.count(item) for item in sorted(set(z_str)) }
245 { item: z_text.count(item) for item in sorted(set(z_text)) }
246 # Among <span> with <a> child, 16 texts contain 'Top'
247
248 {'<span class="text-right d-block"> <span class="icon-angle-up"> <!-- --> </span> <a
248 href="#"> Top </a> </span>': 8,
249  '<span class="toTop"> <span class="icon-angle-up"> <!-- --> </span> <a
249  class="tp-link-policy" href="#"> Top </a> </span>': 8}
250 {'Top': 16}
251
252 # zp_toptext
253 z = zp_toptext # 21080
254 z_str = sorted([str(tag) for tag in z])
255 z_text = sorted([tag.get_text('|', strip=True) for tag in z])
256
257 with open('zp_toptext-str.txt', 'w') as file_out:
258    for z_s in sorted(set(z_str)):
259       file_out.write(f'{z_s}\n')
260 with open('zp_toptext-text.txt', 'w') as file_out:
261    for z_t in sorted(set(z_text)):
262       file_out.write(f'{z_t}\n')
263 del file_out, z_s, z_t
264
265 print(sorted([len(x) for x in set(z_str)], reverse=True))
266 # z_str = sorted([str(tag) for tag in z if len(str(tag))]) # 21080
267 print(sorted([len(x) for x in set(z_text)], reverse=True))
268 # z_text = sorted([tag.get_text('|', strip=True) for tag in z
```

```
269 #                  if len(tag.get_text('|', strip=True))])   # 21080
270 # review of results suggests limiting lengths: str 66, text 13
271 z_str = sorted([str(tag) for tag in z if len(str(tag)) <= 66]) # 21019
272 z_text = sorted([tag.get_text('|', strip=True) for tag in z
273                  if len(tag.get_text('|', strip=True)) <= 13]) # 21019
274 { item: z_str.count(item) for item in sorted(set(z_str)) }
275 { item: z_text.count(item) for item in sorted(set(z_text)) }
276 # Among <p> with <a> child, 21019 texts contain
277 #   ['Back to top', 'Back to top|]', 'Top', 'Top of Page']
278
279 {'<p align="left" class="psmall"> <a href="#"> Back to top </a> </p>': 1,
280  '<p class="float-right"> <a href="#"> Top </a> </p>': 8063,
281  '<p class="psmall" dir="ltr"> <a href="#"> Back to top </a> </p>': 9,
282  '<p class="psmall"> <a href="#"> Back to top </a> </p>': 7555,
283  '<p class="psmall"> <a href="#"> Back to top </a> ] </p>': 2,
284  '<p class="psmall"> <a href="#top"> Back to top </a> </p>': 3,
285  '<p class="psmall"> <a href="#ttop"> Back to top </a> </p>': 5,
286  '<p class="topOPage"> <a href="#"> Top of Page </a> </p>': 5369,
287  '<p> <a class="psmall" href="#top"> Back to top </a> </p>': 1,
288  '<p> <a href="#"> Back to top </a> </p>': 11}
289 {'Back to top': 7585, 'Back to top|]': 2, 'Top': 8063, 'Top of Page': 5369}
290
291 pd.DataFrame(\
292    [{ 'markup': z_s,
293       'content': BeautifulSoup(z_s, 'lxml').p.get_text('|', strip=True),
294       'freq': z_str.count(z_s) }
295          for z_s in sorted(set(z_str))]).to_excel('zp_toptext.xlsx', engine='openpyxl')
296
297 # The 20137 <a> texts that contain ['Back to top', 'Top', 'Top of Page']
298 #    include 16 <span> texts with 'Top'
299 #    and 21019 <p> texts with ['Back to top', 'Back to top|]', 'Top', 'Top of Page']
300 # Seek 2 other instances of <a> with 'top' in content but not in <p> or <span>
301
302 # Elements with child <a> element and text with word 'top' (ingore case)
303 z_all_toptext = [tag for soup in tqdm(pcd_art_soup)
304                  for tag in soup.find_all(True)
305                  if tag.find('a', href=True, recursive=False) is not None and
306                     tag.get_text() is not None and
307                     re.search(r'\btop\b', tag.get_text(), re.I)]
308 # 22965 items in list
309 set([z.name for z in z_all_toptext])
310 # ('caption', 'div', 'h5', 'li', 'p', 'span', 'td')
311 { elem: [z.name for z in z_all_toptext].count(elem)
312          for elem in ('caption', 'div', 'h5', 'li', 'p', 'span', 'td') }
313 {'caption': 2, 'div': 2, 'h5': 8, 'li': 1855, 'p': 21080, 'span': 16, 'td': 2}
314
315 # Review <p>s not in zp_toptext, based on text length; there are 61
316 z_all_toptext_p = [tag for tag in z_all_toptext
317                    if (tag.name == 'p' and len(tag.get_text('|', strip=True)) > 13)]
318 # Inspect <a> elements that are direct children of the <p> element
319 [(i, tag.find_all('a', recursive=False)) for (i, tag) in enumerate(z_all_toptext_p)]
320 # (46, [<a href="#"> Back to top </a>])
321 z_all_toptext_p[46].find('a', recursive=False).parent
322 print(z_all_toptext_p[46].find('a', recursive=False).parent.\
323       prettify(formatter='minimal'))
324 # Go up DOM tree to find enclosing <body> element
```

```
z_all_toptext_p[46].find('a', recursive=False).parent.parent.parent.parent.parent.name
# Write <body> to file to inspect
with open('z_all_toptext_p46.htm', 'w') as file_out:
   file_out.write(z_all_toptext_p[46].find('a', recursive=False).\
                  parent.parent.parent.parent.parent.prettify(formatter='minimal'))
del file_out
# Pinpointed: http://www.cdc.gov/pcd/issues/2010/mar/09_0106.htm,
# which has '<a href="#">Back to top</a>' stuck at the end of a
# <p class="caption">, not <p class="small"> as in the rest of the doc.

# Review <div>s not in zp_toptext
z_all_toptext_div = [tag for tag in z_all_toptext
                  if (tag.name == 'div')]
# Inspect <a> elements that are direct children of the <div> element
[tag.find_all('a', recursive=False) for tag in z_all_toptext_div]
# Zero in on the 2nd result, which has '<a href="#"> Back to top </a>'
z_all_toptext_div[1].find('a', recursive=False).parent
# Pinpointed: http://www.cdc.gov/pcd/issues/2010/nov/10_0104.htm,
# which has '<a href="#">Back to top</a>' not inside <p> or <span>.
# Rendering engine inserts missing <p> to correct for malformed HTML.

# Conclusion 1:
# Each <a> element with href that begins '#' and content including 'top'
#    ends a document segment.
# All but 1 are direct children of <p> or <span>.
# All but 2 are direct children of <p> or <span> that contain no additional text.

#%% 2. Find elements that might contain corresponding Spanish-language text
#      <a> with href in {'#top', '#ttop'}; <p> with class 'topOPage'
#      inspection of PCD Spanish-language pages

za_toptext = [tag for soup in tqdm(pcd_art_soup)
                  for tag in soup.find_all('a', href=['#top', "#ttop"])]
# 415; sum([x.get_text() is None for x in za_toptext])
z = za_toptext # 415
z_str = sorted([str(tag) for tag in z])
z_text = sorted([tag.get_text('|', strip=True) for tag in z])
{ item: z_str.count(item) for item in sorted(set(z_str)) }
# { item: z_text.count(item) for item in sorted(set(z_text)) }

{'<a class="psmall" href="#top"> Back to top </a>': 1,
 '<a href="#top"> Back to top </a>': 3,
 '<a href="#top"> Inicio de página </a>': 28,
 '<a href="#top"> Volver al Inicio </a>': 58,
 '<a href="#top"> Volver al comienzo </a>': 282,
 '<a href="#top"> Volver al comienzoo </a>': 1,
 '<a href="#top"> Volver al inicio </a>': 37,
 '<a href="#ttop"> Back to top </a>': 5}
['Back to top', 'Inicio de página', 'Volver al Inicio', 'Volver al comienzo',
 'Volver al comienzoo', 'Volver al inicio']

zp_toptext = [tag for soup in tqdm(pcd_art_soup)
                  for tag in soup.find_all('p', class_='topOPage')]
# [str(tag) for tag in zp_toptext if tag.find('a', href=True, recursive=False) is None]
# ['<p class="topOPage"> </p>']
z = zp_toptext # 5572
```

```
z_str = sorted([str(tag) for tag in z])
z_text = sorted([tag.get_text('|', strip=True) for tag in z])
{ item: z_str.count(item) for item in sorted(set(z_str)) }
# { item: z_text.count(item) for item in sorted(set(z_text)) }

{'<p class="topOPage"> </p>': 1,
 '<p class="topOPage"> <a href="#"> Inicio de la página </a> </p>': 153,
 '<p class="topOPage"> <a href="#"> Top of Page </a> </p>': 5369,
 '<p class="topOPage"> <a href="#"> Volver al comienzo </a> </p>': 49}
['', 'Inicio de la página', 'Top of Page', 'Volver al comienzo']

# Conclusion 2: Add 'comienzo' and 'inicio' to 'top'

#%% 3. Find elements that have <a> child with 'top', 'comienzo', or 'inicio'
#      especially <p> and <span> elements

z_all_toptext = [tag for soup in tqdm(pcd_art_soup)
                 for tag in soup.find_all(True)
                 if tag.find('a', href=True, recursive=False) is not None and
                    tag.get_text() is not None and
                    re.search(r'\b(top|inicio|comienzoo?)\b', tag.get_text(), re.I)]
# 23587; 3542/3542 [00:35<00:00, 100.63it/s]

z = z_all_toptext # 23587
z_str = sorted([str(tag) for tag in z])
z_text = sorted([tag.get_text('|', strip=True) for tag in z])
{ item: z_str.count(item) for item in sorted(set(z_str)) }
# { item: z_text.count(item) for item in sorted(set(z_text)) }

with open('z_all_toptext-str.txt', 'w') as file_out:
   for z_s in sorted(set(z_str)):
      file_out.write(f'{z_s}\n')
with open('z_all_toptext-text.txt', 'w') as file_out:
   for z_t in sorted(set(z_text)):
      file_out.write(f'{z_t}\n')
del file_out, z_s, z_t

z_toptext = [tag for tag in z_all_toptext
             if re.search(r'\b(top|inicio|comienzoo?)\b', tag.a.get_text(), re.I)
                and len(tag.a.get_text('|', strip=True)) < 20]
# 21644
# hard limit of length 20; reducing upper bound loses 'Inicio de la página'

# z_toptext_ = [tag for tag in tqdm(z_all_toptext)
#               if len(tag.a.get_text('|', strip=True)) < 20
#                  and re.search(r'\b(top|inicio|comienzoo?)\b', tag.a.get_text(), re.I) is
None]
# 987: {'': 57, 'Authorâ€™s Corner': 2, 'Author’s Corner': 914, 'CrossRef': 7,
#       'Figure 2': 1, 'Home': 2, 'PubMed': 3, 'Table 3': 1}

z = z_toptext # 21644
z_str = sorted([str(tag) for tag in z])
z_text = sorted([tag.get_text('|', strip=True) for tag in z])
{ item: z_str.count(item) for item in sorted(set(z_str)) }
# { item: z_text.count(item) for item in sorted(set(z_text)) }

```

```
{'<p align="left" class="psmall"> <a href="#"> Back to top </a> </p>': 1,
 '<p class="caption"> <a href="#top"> Volver al comienzo </a> </p>': 1,
 '<p class="caption"> [...] <a href="#"> Back to top </a> </p>': 1,
 '<p class="float-right"> <a href="#"> Top </a> </p>': 8063,
 '<p class="psmall" dir="ltr"> <a href="#"> Back to top </a> </p>': 9,
 '<p class="psmall" dir="ltr"> <a href="#top"> Volver al comienzo </a> </p>': 1,
 '<p class="psmall"> <a href="#"> Back to top </a> </p>': 7555,
 '<p class="psmall"> <a href="#"> Back to top </a> ] </p>': 2,
 '<p class="psmall"> <a href="#top"> Back to top </a> </p>': 3,
 '<p class="psmall"> <a href="#top"> Inicio de página </a> </p>': 28,
 '<p class="psmall"> <a href="#top"> Volver al Inicio </a> </p>': 58,
 '<p class="psmall"> <a href="#top"> Volver al comienzo </a> </p>': 279,
 '<p class="psmall"> <a href="#top"> Volver al comienzoo </a> </p>': 1,
 '<p class="psmall"> <a href="#top"> Volver al inicio </a> </p>': 37,
 '<p class="psmall"> <a href="#ttop"> Back to top </a> </p>': 5,
 '<p class="topOPage"> <a href="#"> Inicio de la página </a> </p>': 153,
 '<p class="topOPage"> <a href="#"> Top of Page </a> </p>': 5369,
 '<p class="topOPage"> <a href="#"> Volver al comienzo </a> </p>': 49,
 '<p> <a class="psmall" href="#top"> Back to top </a> </p>': 1,
 '<p> <a href="#"> Back to top </a> </p>': 11,
 '<p> <a href="#top"> Volver al comienzo </a> </p>': 1,
 '<span class="text-right d-block"> <span class="icon-angle-up"> <!-- --> </span> <a
 href="#"> Top </a> </span>': 8,
 '<span class="toTop"> <span class="icon-angle-up"> <!-- --> </span> <a
 class="tp-link-policy" href="#"> Top </a> </span>': 8}

['Abbreviation: ...|Back to top',
 'Back to top', 'Back to top|]', 'Inicio de la página', 'Inicio de página',
 'Top', 'Top of Page', 'Volver al Inicio', 'Volver al comienzo',
 'Volver al comienzoo', 'Volver al inicio']

pcd_toptext = [
   dict(year=int(path[12:16]), markup=str(tag), content=tag.a.get_text('|', strip=True))
      for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
      for tag in soup.find_all(True)
          if tag.find('a', href=True, recursive=False) is not None
             and tag.get_text() is not None
             and re.search(r'\b(top|inicio|comienzoo?)\b', tag.get_text(), re.I)
             and len(tag.a.get_text('|', strip=True)) < 20]
# 22631
# 3542/3542 [00:57<00:00, 61.24it/s]
# pd.DataFrame(pcd_toptext).to_excel('pcd_toptext.xlsx', engine='openpyxl')

pcd_toptext = [
   dict(year=int(path[12:16]), markup=str(every_tag), content=a_tag.get_text('|',
   strip=True))
      for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
      for every_tag in soup.find_all(True)
      for a_tag in every_tag.find_all('a', href=True, recursive=False)
          if a_tag.get_text() is not None
             and len(a_tag.get_text('|', strip=True)) < 20
             and re.search(r'\b(top|inicio|comienzoo?)\b', a_tag.get_text(), re.I)
             and len(str(every_tag)) < 120]
# length limits of 120 and 20 are empirically determined hard bounds
# 21643
# 3542/3542 [00:51<00:00, 69.18it/s]
```

```
489 pcd_toptext_dframe = pd.DataFrame(pcd_toptext)
490 # pcd_toptext_dframe.to_excel('pcd_toptext.xlsx', engine='openpyxl')
491
492 pd.crosstab(pcd_toptext_dframe.markup, pcd_toptext_dframe.year, margins=True).\
493    iloc[[22, 5, 0, 8, 18, 13, 3, 9, 12, 7, 10, 6, 1, 4, 11, 17, 19, 15, 16, 14, 2, 20,
493    21], :]
494
495 pcd_topstrings = set([x['content'] for x in pcd_toptext])
496 # {'Back to top', 'Inicio de la página', 'Inicio de página', 'Top', 'Top of Page',
497 #  'Volver al Inicio', 'Volver al comienzo', 'Volver al comienzoo', 'Volver al inicio'}
498 # in rder by language and approximate order of appearance:
499 pcd_topstrings = {'Back to top', 'Top of Page', 'Top',
500    'Inicio de página', 'Volver al inicio', 'Volver al Inicio',
501    'Volver al comienzo', 'Volver al comienzoo', 'Inicio de la página'}
502
503 # Conclusion 3: 9 anchor content strings delineate end-of-section
504
505 #%% Explore <h1>, <h2> and other elements that dellinete document segments
506
507 x = pcd_art_soup[73] # 2019/19_0035.htm
508
509 def depth(object, ancestor_name = 'body'):
510    for depth, ancestor in enumerate(object.parents, start=1):
511       if ancestor.name == ancestor_name: break
512    if ancestor.name is None: depth = None
513    return depth
514
515 [(depth(tag), tag.name, tag.get_text('|', strip=True))
516  for tag in x.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])]
517
518 def children(object):
519    child_names = [y.name for y in object.children if y.name is not None]
520    uniq_names = sorted(set(child_names))
521    # print(uniq_names)
522    if len(uniq_names) > 0:
523       children = '|'.join(['|'.join([child_name, str(child_names.count(child_name))])
524                            for child_name in uniq_names])
525    else: children = ''
526    return children
527
528 children(x.find('table'))
529
530 [(depth(tag), tag.name, tag.get_text('|', strip=True), children(tag))
531  for tag in x.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])]
532
533 h_children = [{'path': path, 'name': tag.name, 'depth': depth(tag),
534                'string': tag.get_text('|', strip=True),
535                'attrs': '' if tag.name is None else str(tag.attrs),
536                'children': children(tag)}
537    for (path, soup) in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
538    for tag in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])]
539 # 3542/3542 3542/3542 [00:52<00:00, 67.90it/s]
540 pd.DataFrame(h_children).to_excel('h_children.xlsx', engine='openpyxl')
541
542 # Which articles have a <main> element?
543 has_main = sorted([\
```

```
544    path for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup))
545         if soup.main is not None])
546 # Answer: exactly those published in 2015 or later
547 # Among those with a <main> element, what are the attributes of child <div>s?
548 has_main_sub = [(path, soup.main is not None,
549                         soup.find('div', class_='content') is not None,
550                         soup.find('div', class_='content-fullwidth') is not None)
551    for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)]
552 # pd.DataFrame(has_main_sub).to_clipboard()
553 # All 916 documents with <main> also have <div class="content content-fullwidth">.
554
555 # How many lack <main> but have <div class="main-inner">?
556 has_main_inner = [(path, soup.main is not None,
557                         soup.find('div', class_='main-inner') is not None)
558    for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)]
559 # pd.DataFrame(has_main_inner).to_clipboard()
560 # All 928 documents with <div class="main-inner"> do not have <main>
561 # Is it feasible to remove <div class="onthispageChrono"> elements?
562 onthispageChrono = [
563    dict(i=i, j=j, k=k, parent_name=elem.parent.name, name_=elem.name, attrs=elem.attrs,
563    depth=depth(elem))
564    for i, soup in tqdm(enumerate(pcd_art_soup), total=3542)
565    for j, chrono in enumerate(soup.find_all('div', class_='onthispageChrono'))
566    for k, elem in enumerate(chrono.descendants)
567       if isinstance(elem, Tag)]
568 # pd.DataFrame(onthispageChrono).to_excel('onthispageChrono.xlsx', engine='openpyxl')
569
570 onthispageChrono_text = [
571    dict(i=i, j=j, text=chrono.get_text('|', strip=True), depth=depth(chrono))
572    for i, soup in tqdm(enumerate(pcd_art_soup), total=3542)
573    for j, chrono in enumerate(soup.find_all('div', class_='onthispageChrono'))]
574 # pd.DataFrame(onthispageChrono_text).to_excel('onthispageChrono_text.xlsx',
574 engine='openpyxl')
575
576 # Which articles have an element <td width="80%">?
577 has_width_80 = sorted([\
578    path for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
579         if soup.find('td', width="80%") is not None])
580 # Answer: exactly those published through 2011
581 has_width_80_sub = [(path, soup.find('td', width="80%") is not None)
582    for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)]
583 # pd.DataFrame(has_width_80_sub).to_clipboard()
584 # The remaining 1698 documents have <td width="80%">
585 # Among these 1698 documents, which <td width="80%"> contain content of interest?
586 has_width_80_n = [\
587    dict(year=int(path[12:16]), n_width_80=len(soup.find_all('td', width="80%")))
588       for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)]
589 # pd.DataFrame(has_width_80_n).to_clipboard()
590
591 has_width_80_children = [\
592    dict(path=path, year=int(path[12:16]), i=i, j=j, k=k, n_width_80=len(td),
592    child_name=child.name,
593         child_attrs=child.attrs)
594       for i, (path, soup) in tqdm(enumerate(zip(pcd_art_frame.mirror_path, pcd_art_soup)),
594       total=3542)
595       for j, td in enumerate(soup.find_all('td', width="80%"))
```

```
596       for k, child in enumerate(td.children)
597          if td.name is not None and child.name is not None]
598 # pd.DataFrame(has_width_80_children).to_clipboard()
599
600 has_style_width_80 = [(path, soup.find('td', style=re.compile('80%')) is not None)
601    for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)]
602 # occurs with 3 files
603 # /pcd/issues/2011/nov/11_0062.htm
604 # /pcd/issues/2011/nov/10_0191.htm
605 # /pcd/issues/2011/nov/10_0276.htm
606 has_style_width_80 = [(i, path)
607    for i, (path, soup) in tqdm(enumerate(zip(pcd_art_frame.mirror_path, pcd_art_soup)),
607    total=3542)
608       if soup.find('td', style=re.compile('80%')) is not None]
609 # [(1514, '/pcd/issues/2011/nov/11_0062.htm'),
610 #  (1521, '/pcd/issues/2011/nov/10_0191.htm'),
611 #  (1522, '/pcd/issues/2011/nov/10_0276.htm')]
612
613 # list of 12:
614 [(path, td.attrs) for i, path in has_style_width_80
615     for td in pcd_art_soup[i].find_all('td')
616     if (td.has_attr('width') and re.search('80%', td['width'])) or
617        (td.has_attr('style') and re.search('80%', td['style']))]
618 # list of 3 lists of 4 each:
619 [ [(path, td.attrs) for td in pcd_art_soup[i].find_all('td')
620      if (td.has_attr('width') and re.search('80%', td['width'])) or
621        (td.has_attr('style') and re.search('80%', td['style']))]
622     for i, path in has_style_width_80]
623 # in all 3 cases, <td style="width: 80%"> appears once,
624 # followed by <td width="80%" 4 times>
625
626 [dict(i=i, path=path, k=k, child_name=child.name, child_attrs=child.attrs)
627     for i, path in has_style_width_80
628     for k, child in enumerate(pcd_art_soup[i].find('td', style='width: 80%').children)
629          if child.name is not None]
630 # same structure as many <td width="80%">: <img>, <br>, <table>,
631 #    <p align="right" class="psmall"> (with vol, iss, date),
632 #    <div class="syndicate"> (with type, title),
633 #    <form name="eMailer">,
634 #    <div class="syndicate">
635
636 table_width_soup = [(i, soup)
637    for i, soup in tqdm(enumerate(pcd_art_soup), total=3542)
638    if soup.find('div', class_=['content-fullwidth', 'main-inner']) is None]
639 # 3542/3542 [00:15<00:00, 236.02it/s]
640 table_width_info = [
641    dict(i=i, j=j, elem=elem.name, depth=depth(elem),
642         width=no(elem.get('width')), style=no(elem.get('style')))
643    for i, soup in tqdm(table_width_soup)
644    for j, elem in enumerate(soup.find_all(name=['table', 'td']))
645       if elem.has_attr('width') or elem.has_attr('style')]
646 # pd.DataFrame(table_width_info).to_excel('table_width_info.xlsx', engine='openpyxl')
647
648
649 # Summary
650 # 2004-2011: 1698 <table><tr><td width="80%">
```

```
651 # 2012-2014:  928 <div class="main-inner">
652 # 2015-2020:  916 <main><div class="content">
653
654 # Which articles have a <div class="syndicate"> element?
655 has_syndicate = sorted([\
656    path for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
657         if soup.find('div', class_="syndicate") is not None])
658 # Answer: mostly those published in July 2010 or later
659 #    Some exceptions: through May 2010, 6/1279 have this element, 14/2242 do not
660
661 # How can datelines be identified?
662 from bs4 import Comment, Tag
663
664 z = [dateline for soup in tqdm(pcd_art_soup)
665      for volume in soup.find_all('p', string=re.compile('Volume'))
666      for dateline in volume]
667 # 1572
668
669 # For the following markup, soup.string returns None because of the comment
670 # <p align="right" class="psmall">
671 #  <!-- Write the date of the issue here -->
672 #  Volume 1: No. 1, January 2004
673 # </p>
674 # So find_all('p', string=...) doesn't catch those
675
676 z = [(path, dateline.get_text('|', strip=True), dateline.get('class'))
677      for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
678      for dateline in soup.find_all('p')
679         if dateline.strings is not None and re.search('Volume', dateline.get_text())]
680 # 3277
681 len(set([path for path, text, class_ in z]))
682 # This includes 6 where "Volume" is in longer text
683 # 3270 that look like datelines, max string length 65
684 # Is there a better way to capture those 3270?
685 # What about the other 272? All exceptions published in 2012, but not all 2012
686 # publications are excepted. There are 8 errata, and the rest have
687 # <div class="syndicate">.
688
689 # Proposed rule: Use <div class="syndicate">, if it is available (in which case,
690 # check also for dateline in absence of <div class="dateline">).
691 # When <div class="syndicate"> is not available, look for <td width="80%">.
692 # When both are lacking, use <div class="main-inner">
693
694 dateline.name is not None and
695
696 z = [psmall.get_text('|', strip=True) for soup in tqdm(pcd_art_soup)
697      for all_psmall in soup.find_all('p', class_='psmall')
698      for psmall in all_psmall
699         if isinstance(psmall, Tag) and psmall.get_text() is not None
700            and re.search('\bVolume', psmall.get_text())]
701 dateline1 = [ dateline.get_text('|', strip=True)
702               for dateline in soup.find('p', class_='psmall')
703
704    path for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
705         if soup.find('div', class_="syndicate") is not None]
706
```

```
707 isinstance(tag.next_element, Comment)
708 # <p <div class="dateline">
709
710 # Elements <h1>, ..., <h6> and their descendants
711 h_all = [dict(path=path, year=int(path[12:16]),
712               h_name=h_elem.name, h_depth=depth(h_elem), h_str=str(h_elem),
713               h_text=h_elem.get_text('|', strip=True),
714               h_attrs=str(h_elem.attrs))
715          for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
716          for h_elem in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])]
717 pd.DataFrame(h_all).to_excel('h_all.xlsx', engine='openpyxl')
718
719 h_descendants = [dict(path=path, year=int(path[12:16]),
720               h_name=h_elem.name, h_depth=depth(h_elem), h_attrs=str(h_elem.attrs),
721               h_desc_name=h_desc.name, h_desc_depth=depth(h_desc),
721               h_desc_attrs=str(h_desc.attrs),
722               h_str=str(h_elem), h_desc_str=str(h_desc),
723               h_text=h_elem.get_text('|', strip=True),
724               h_desc_text = '' if h_desc.get_text() is None else h_desc.get_text('|',
724               strip=True))
725          for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542)
726          for h_elem in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
727          for h_desc in h_elem.descendants
728              if h_desc.name is not None]
729 # 3542/3542 [01:05<00:00, 54.17it/s]
730 pd.DataFrame(h_descendants).to_excel('h_descendants.xlsx', engine='openpyxl')
731
732 # conclusion: every <h[1-6]> is of interest;
733 # descendant <a> or <span> could be, as well
734
735 #%% Assemble forgoing information into comprehensive inventory
736 #      content containers: class="content" OR "main-inner", td width="80%"
737 #      within content containers:
738 #         <div class="syndicate">, <div class="dateline">
739 #         content: p, h[1-6], r'\bVolumen?\b'
740 #         extraneous: Comment, Script, Stylesheet; eMailer, onthispageChrono,
740 tp-on-this-page
741 #         delimiters: p/span with short content that includes top, comienzo, inicio
742
743 pcd_year = [int(path[12:16]) for path in pcd_art_frame.mirror_path]
744
745 # Inventory candidate containers
746 def pcd_containers(soup):
747    # soup_main = soup.main
748    main = len(soup.find_all('main'))
749    div_content = len(soup.find_all('div', class_='content'))
750    div_content_fw = len(soup.find_all('div', class_='content-fullwidth'))
751    div_main_inner = len(soup.find_all('div', class_='main-inner'))
752    td_w_80 = len(soup.find_all('td', width='80%'))
753    td_w_70 = len(soup.find_all('td', width='70%'))
754    td_w_80_s = len(soup.find_all('td', style='width: 80%'))
755    div_syndicate = len(soup.find_all('div', class_='syndicate'))
756    return dict(main=main, div_content=div_content,
757       div_content_fw=div_content_fw, div_main_inner=div_main_inner,
758       td_w_80=td_w_80, td_w_80_s=td_w_80_s, td_w_70=td_w_70,
759       div_syndicate=div_syndicate)
```

```
760
761 pcd_container_freq = [pcd_containers(soup) for soup in tqdm(pcd_art_soup)]
762 # 3542/3542 [02:00<00:00, 29.38it/s]
763 pd.concat([pd.Series(pcd_art_frame.mirror_path, name='path'),
764            pd.Series(pcd_year, name='year'),
765            pd.DataFrame(pcd_container_freq)],
766           axis=1).to_excel('pcd_container_freq.xlsx', engine='openpyxl')
767 # [3542 rows x 9 columns]
768
769 # Anchor text that links back to top of page, thus delimits document segments
770 # Cast as a set for efficient calculation of membership
771 pcd_topstrings = {'Back to top', 'Top', 'Top of Page',
772    'Inicio de la página', 'Inicio de página', 'Volver al Inicio',
773    'Volver al comienzo', 'Volver al comienzoo', 'Volver al inicio'}
774
775 def td_w_80_fn(tag):
776    if tag.name == 'td':
777       td_width = tag.has_attr('width') and tag['width'] in {'80%', '70%'}
778       td_style = tag.has_attr('style') and tag['style'] == 'width: 80%'
779       td_w_80 = td_width or td_style
780    else:
781       td_w_80 = False
782    return td_w_80
783
784 #%% Split into 4 phases and process each separately
785
786 # Start over with HTML, sorting filenames for convenience
787
788 # x = ['/pcd/issues/2007/apr/06_0105_es.htm', '/pcd/issues/2019/19_0199.htm']
789 # re.search(r'/(?P<year>\d{4})/(?:[a-z]{3}/)?(?P<file>[\w-]+?).htm', x[1]).groupdict()
790 # '/pcd/issues/2007/apr/06_0105_es.htm' # {'year': '2007', 'file': '06_0105_es'}
791 # '/pcd/issues/2019/19_0199.htm'        # {'year': '2019', 'file': '19_0199'}
792
793 # Sort in descending order by year of publication and article filename
794 # This is close to but not exactly the same as reverse of order published
795 year_file_re = re.compile(r'''
796    /pcd/issues/(?P<year>\d{4})/ # match year
797    (?:[a-z]{3}/)?               # match and discard month, if it exists
798    (?P<file>[\w-]+?).htm        # match filename root
799    ''', re.VERBOSE)
800 sorted_paths = sorted(pcd_art_frame.mirror_path,
801        key=lambda path: re.search(year_file_re, path).groups(),
802        reverse=True)
803
804 # Epochs fall out (roughly) as
805 #    pcd1: 2004-2010/05; pcd2: 2010/07-2011; pcd3: 2012-2014; pcd4: 2015-2020
806
807 pcd1_html = list(); pcd2_html = list(); pcd3_html = list(); pcd4_html = list()
808 for path in tqdm(sorted_paths):
809    html = html_reduce_space_u(read_uni_html(PCD_BASE_PATH_u3 + path))
810    soup = BeautifulSoup(html, 'lxml')
811
812    if soup.find('div', class_='content-fullwidth') is not None:
813       pcd4_html.append(dict(path=path, html=html))
814    elif soup.find('div', class_='main-inner') is not None:
815       pcd3_html.append(dict(path=path, html=html))
```

```
816     elif soup.find('div', class_='syndicate') is not None:
817        pcd2_html.append(dict(path=path, html=html))
818     else:
819        pcd1_html.append(dict(path=path, html=html))
820 del year_file_re, path, html, soup
821 # 3542/3542 [02:42<00:00, 21.76it/s]
822 # [len(x) for x in [pcd1_html, pcd2_html, pcd3_html, pcd4_html]]
823 # [1284, 414, 928, 916]
824 # pickle.dump(pcd1_html, open('pcd1_html.pkl', 'wb'))
825 # pickle.dump(pcd2_html, open('pcd2_html.pkl', 'wb'))
826 # pickle.dump(pcd3_html, open('pcd3_html.pkl', 'wb'))
827 # pickle.dump(pcd4_html, open('pcd4_html.pkl', 'wb'))
828 # pcd1_html = pickle.load(open('pcd1_html.pkl', 'rb'))
829 # pcd2_html = pickle.load(open('pcd2_html.pkl', 'rb'))
830 # pcd3_html = pickle.load(open('pcd3_html.pkl', 'rb'))
831 # pcd4_html = pickle.load(open('pcd4_html.pkl', 'rb'))
832
833 # Storing all soup works for PCD because of its size
834 pcd1_soup = [BeautifulSoup(html['html'], 'lxml') for html in tqdm(pcd1_html)]
835 # 1284/1284 [00:27<00:00, 47.05it/s]
836 pcd2_soup = [BeautifulSoup(html['html'], 'lxml') for html in tqdm(pcd2_html)]
837 # 414/414 [00:10<00:00, 38.52it/s]
838 pcd3_soup = [BeautifulSoup(html['html'], 'lxml') for html in tqdm(pcd3_html)]
839 # 928/928 [00:24<00:00, 37.48it/s]
840 pcd4_soup = [BeautifulSoup(html['html'], 'lxml') for html in tqdm(pcd4_html)]
841 # 916/916 [00:56<00:00, 16.08it/s]
842
843 # Epochs 3 and 4 appear to be the most straightforward to parse
844 pcd4_html[0]['path'] # '/pcd/issues/2020/19_0391.htm'
845 with open('2020-19_0391-pretty.htm', 'w') as file_out:
846    file_out.write(html_prettify_u(pcd4_html[0]['html']))
847 x = pcd4_soup[0].find('div', class_='syndicate')
848 for j in range(30):
849    print(f'{j}: \'{len(str(x))}\'')
850    if x is None: break
851    x = x.next_sibling
852
853 print([z for z in x.find_all_next(string=True) if z != ' '])
854
855 y = [len(soup.find_all('div', class_='col-md-4')) for soup in tqdm(pcd4_soup)]
856 # 153 have 1; 763 have 0
857 y = [dict(n=len(soup.find_all('div', class_='col-md-4')),
858       dateline=soup.find('div', class_='dateline').div.p.get_text(strip=True))
859      for soup in tqdm(pcd4_soup)]
860 pd.DataFrame(y).to_clipboard()
861
862 pcd4_html[179]['path'] # '/pcd/issues/2019/18_0288.htm'
863 with open('pretty/2019-18_0288-pretty.htm', 'w') as file_out:
864    file_out.write(html_prettify_u(pcd4_html[179]['html']))
865
866 y = [dict(i=i, j=j, k=k, text=div_synd_div.get_text('|', strip=True),
867           parent_name=div_synd_div.parent.name, attrs=div_synd_div.attrs)
868      for i, soup in tqdm(enumerate(pcd4_soup), total=916)
869      for j, div_synd in enumerate(soup.find_all('div', class_='syndicate'))
870      for k, div_synd_div in enumerate(div_synd.find_all('div'))
871      if len(div_synd.find_all('div')) > 0]
```

```
872
873 div_bg_secondary = [soup.find('div', class_='bg-secondary').get_text('|', strip=True)
874    for soup in tqdm(pcd4_soup)
875    if soup.find('div', class_='bg-secondary') is not None]
876 # set(div_bg_secondary) # {'Summary'}
877 # div_bg_secondary.count('Summary') # 153
878
879 div_card_text = [(i, card_text)
880    for i, soup in tqdm(enumerate(pcd4_soup), total=916)
881    for card_text in soup.find_all('div', class_='card-text')
882    # for p in card_text.find_all('p')
883    # if card_text.p.string is not None and card_text.p.string[:4] == 'What']
884    if card_text.p.get_text() is not None
885    and card_text.p.get_text(strip=True)[:4] == 'What'
886    and len(card_text) != 13]
887 # 66, 99
888 card_text_0 = '|'.join([p.get_text(strip=True)
889                                for i, p in enumerate(div_card_text[0][1].find_all('p'))
890                                if i in (0, 2, 4)])
891 card_text_0 = card_text_0.replace('What is already known on this topic?', '', 1)
892 card_text_1 = '|'.join([p.get_text(strip=True)
893                                for i, p in enumerate(div_card_text[1][1].find_all('p'))
894                                if i in (1, 3, 5)])
895
896 y = [len(soup.find_all('div', class_='card-text')) for soup in tqdm(pcd4_soup)]
897 [y.count(x) for x in range(4)] # [763, 147, 0, 6]
898
899 y = [soup.find_all('div', class_='card-text') for soup in tqdm(pcd4_soup)
900      if len(soup.find_all('div', class_='card-text')) > 1]
901
902 def div_card_text_fn(soup):
903    div_card_text = soup.find_all('div', class_='card-text')
904    div_len = len(div_card_text)
905
906    # Length is 0, 1, or 3; extract usable div element
907    if div_len == 3:
908       div_card_text = div_card_text[2]
909    elif div_len == 1:
910       div_card_text = div_card_text[0]
911
912    if div_len > 0:
913       card_text_p = div_card_text.find_all('p')
914       # Length is 5, 6, or 7; kludge lengths 5 and 7
915       if len(card_text_p) == 5:
916          card_text = '|'.join([p.get_text(strip=True)
917                                for i, p in enumerate(card_text_p)
918                                if i in (0, 2, 4)])
919          card_text = card_text.replace('What is already known on this topic?', '')
920       elif len(card_text_p) >= 6:
921          card_text = '|'.join([p.get_text(strip=True)
922                                for i, p in enumerate(card_text_p)
923                                if i in (1, 3, 5)])
924    else: card_text = ''
925    return card_text
926
927 div_card_text_fn(pcd4_soup[66])
```

```
928 div_card_text_fn(pcd4_soup[99])
929 [(x, div_card_text_fn(pcd4_soup[x])) for x in range(66, 99+11, 11)]
930
931 card_texts = [div_card_text_fn(soup) for soup in tqdm(pcd4_soup)]
932 len([x for x in card_texts if x != '']) # 153
933
934 pcd4_soup[415].find('div', class_='dateline').p.string
935 # ' EDITOR IN CHIEFÂ€™S COLUMN â€” Volume 14 â€” February 2, 2017 '
936 pcd4_soup[734].find('div', class_='dateline').p.string
937 # ' EDITORÂ€™S CHOICE â€” Volume 12 â€” June 25, 2015 '
938
939 x = str(pcd4_soup[415].find('div', class_='dateline').p.string)
940 x.replace('Â€™', "'").replace('â€”', '—') # em dash \u2014
941 x = str(pcd4_soup[734].find('div', class_='dateline').p.string)
942 x.replace('Â€™', "'").replace('â€”', '—') # em dash \u2014
943
944 x.replace('Â€™', "'").replace('â€”', '—').strip().split(' — ')
945 dict(zip(['type', 'volume', 'date'], \
946     x.replace('Â€™', "'").replace('â€”', '—').strip().split(' — ')))
947
948 def process_dateline(soup):
949    div_dateline = soup.find('div', class_='dateline')
950    if div_dateline.get_text() is None:
951       dateline = dict(type='', volume=None, date=None)
952    else:
953       dateline_text = div_dateline.get_text(strip=True)
954       if 'â€”' in dateline_text:
955          dateline_text = dateline_text.replace('Â€™', "'").replace('â€”', '—')
956       dateline = dateline_text.split(' — ')
957       dateline = dict(type=dateline[0],
958          volume=int(re.search(r'Volume (?P<vol>\d{1,2})', dateline[1]).group('vol')),
959          date=parse(dateline[2]).strftime('%Y-%m-%d'))
960
961    return dateline
962
963 process_dateline(pcd4_soup[415])
964 process_dateline(pcd4_soup[734])
965 datelines = [process_dateline(soup) for soup in tqdm(pcd4_soup)]
966
967 x = [len(soup.find_all('h1')) for soup in tqdm(pcd4_soup)] # all 916
968 x = [soup.find('h1').attrs for soup in tqdm(pcd4_soup)] # all "{'id': 'content'}"
969
970 h1_content = [soup.find('h1').get_text('|', strip=True) for soup in tqdm(pcd4_soup)]
971
972 x = [len(soup.find_all('h4')) for soup in tqdm(pcd4_soup)] # all 916
973 set(x) # 1-17, mode at 2; 1st appears to be authors
974 [(y, x.count(y)) for y in range(1, 18)]
975 # [(1, 36), (2, 781), (3, 30), (4, 12), (5, 17), (6, 16), (7, 12), (8, 3),
976 #  (9, 3), (10, 0), (11, 1), (12, 1), (13, 1), (14, 2), (15, 0), (16, 0), (17, 1)]
977
978 x = [soup.find('h4').get_text('|', strip=True).\
979      replace('(|View author affiliations|)','')
980      for soup in tqdm(pcd4_soup)]
981 # begins and ends with |, with mix of digit, space, comma, hyphen
982 xx = [re.sub(r'\|[|\d, -]+\|', '', y) for y in x]
983 # |2|,3|,4|,5|; |1|, 2|; |1-4|
```

```
 984
 985 # What's up with 'Exit Notification/Disclaimer Policy'?
 986 y = [soup.find_all('h4')
 987      for soup in tqdm(pcd4_soup)
 988      if soup.find('h4').get_text('|', strip=True) == 'Exit Notification/Disclaimer
 988      Policy']
 989 # occurs 36 times; only <h4> when it does
 990 y = [soup.find('h1')
 991      for soup in tqdm(pcd4_soup)
 992      if soup.find('h4').get_text('|', strip=True) == 'Exit Notification/Disclaimer
 992      Policy']
 993 # errata, addenda, retractions
 994
 995 # If the first <h4> is authors and 2 <h4>s is the mode, what's the 2nd <h4>?
 996 y = [str(soup.find_all('h4')[1])
 997      for soup in tqdm(pcd4_soup)
 998      if len(soup.find_all('h4')) > 1]
 999 # mostly "<h4 class=""modal-title"" id=""extModalTitle""> Exit Notification/Disclaimer
 999 Policy </h4>"
1000
1001 z = [dict(soup_num=i, child_num=j, child_name=child.name,
1002         child_attrs=('' if child.name is None else str(child.attrs)),
1003         grandchildren=(0 if child.name is None else len(child.contents)))
1004       for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1005       for j, child in enumerate(soup.find('div', class_='content-fullwidth').children)]
1006
1007 # Direct children of <div class="content-fullwidth">
1008 # Children 1, 2, 4, and 5 below appear in all 916; child appears in only 39
1009 # <div class="syndicate">
1010 # <div class="row col-12 dateline">
1011 # <div class="language-link mb-3 fs0875">
1012 # <div class="mb-3 card tp-related-pages fourth-level-nav d-none">
1013 # <div class="syndicate">
1014
1015 z_lank_link = [(i, str(soup.find('div', class_='language-link')))
1016    for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1017    if soup.find('div', class_='language-link') is not None]
1018 # [ 10,  50, 375, 412, 420, 421, 422, 428, 434, 436, 437, 440, 444, 445, 447,
1019 #  449, 455, 463, 470, 472, 480, 483, 486, 512, 515, 536, 598, 649, 687, 769,
1020 #  834, 840, 870, 871, 879, 880, 885, 886, 887]
1021 # for child 3, content (if any) is noninformative
1022 #    class="language-link" is unique to these 39; other classes are not
1023
1024 block_names = ('h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'p', 'ul', 'ol', 'blockquote')
1025 z = [dict(soup_num=i, child_num=j, child_name=child.name,
1026         grchildren=(0 if child.name is None else len(child.contents)),
1027         grchild_num=k, grchild_name=grchild.name, grchild_depth=depth(grchild),
1028         len=len('' if grchild.name is None else str(grchild.attrs)),
1029         grchild_attrs=('' if grchild.name is None else str(grchild.attrs))
1030         )
1031       for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1032       for j, child in enumerate(soup.find('div', class_='content-fullwidth').\
1033                           find_all('div', class_=('syndicate', 'dateline'),
1033                           recursive=False))
1034       for k, grchild in enumerate(child.find_all(name=True, recursive=False))]
1035 pd.DataFrame(z).to_clipboard()
```

```
1036
1037 z_div = [dict(soup_num=i, child_num=j, child_name=child.name,
1038         grchildren=(0 if child.name is None else len(child.contents)),
1039         grchild_num=k, grchild_name=grchild.name, grchild_depth=depth(grchild),
1040         len=len('' if grchild.name is None else str(grchild.attrs)),
1041         grchild_attrs=('' if grchild.name is None else str(grchild.attrs)),
1042         grchild_str=repr(grchild)
1043         )
1044       for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1045       for j, child in enumerate(soup.find('div', class_='content-fullwidth').\
1046                           find_all('div', class_=('syndicate', 'dateline'),
1046                           recursive=False))
1047       for k, grchild in enumerate(child.find_all(name='div', recursive=False))]
1048 pd.DataFrame(z_div).to_clipboard()
1049
1050 # In 2 instances, there is a 3rd 'syndicate', but it is not a direct child
1051 [pcd4_html[x]['path'] for x in (270, 288)]
1052 ['/pcd/issues/2018/17_0535.htm', '/pcd/issues/2018/17_0434.htm']
1053 list(pcd4_soup[288].find('div', class_='content-fullwidth').find_all('div',
1053 class_=('syndicate', 'dateline')))[3]
1054
1055 [pcd4_html[x]['path'] for x in (439, 441, 447, 53)]
1056 [pcd4_html[x]['path'] for x in (447,441,858,769,574,439,477,415)]
1057
1058 # CME
1059 y = [(i, soup.find('h1').get_text(strip=True))
1060      for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1061      if soup.find('div', class_=('medscapeCME', 'cme')) is not None]
1062 y = [(i, soup.find('div', class_='dateline').get_text(strip=True))
1063      for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1064      if soup.find('div', class_=('medscapeCME', 'cme')) is not None]
1065
1066 [pcd4_html[x]['path'] for x in (73,135,544,852,858,894,901,904)]
1067
1068 p_smallgrey = [len(soup.find_all('p', class_='smallgrey'))
1069                for soup in tqdm(pcd4_soup)]
1070 # occurs 2x in 914; 3x in 2
1071
1072 p_smallgrey = [[str(p)[:60] for p in soup.find_all('p', class_='smallgrey')]
1073                for soup in tqdm(pcd4_soup)]
1074 [p for p in p_smallgrey if len(p)==3]
1075 [['<p class="smallgrey"> <i> Suggested citation for this articl',
1076   '<p class="smallgrey"> The opinions expressed by authors cont',
1077   '<p class="smallgrey"> The opinions expressed by authors cont'],
1078  ['<p class="smallgrey"> <em> Suggested citation for this artic',
1079   '<p class="smallgrey"> <em> Retracted by authors in: </em> Re',
1080   '<p class="smallgrey"> The opinions expressed by authors cont']]
1081
1082 [p[0] for p in p_smallgrey].count('<p class="smallgrey"> <em> Suggested citation for this
1082 artic')
1083 # 117
1084 [p[0] for p in p_smallgrey].count('<p class="smallgrey"> <i> Suggested citation for this
1084 articl')
1085 # 799
1086 [p[1] for p in p_smallgrey].count('<p class="smallgrey"> The opinions expressed by authors
1086 cont')
```

```
1087 # 915, plus the 1 above where it's p[2]
1088
1089 p_smallgrey = [(i, j, k, p_smallgrey.get_text('|', strip=True)[:37])
1090                for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1091                for j, div_synd in enumerate(soup.find_all('div', class_='syndicate'))
1092                for k, p_smallgrey in enumerate(div_synd.find_all('p',
1092                class_='smallgrey'))]
1093 pd.DataFrame(p_smallgrey).to_clipboard()
1094
1095 p_peervwd = [(html['path'], '' if peer_text.get_text() is None else
1095 peer_text.get_text(strip=True))
1096    for html, soup in tqdm(zip(pcd4_html, pcd4_soup), total=916)
1097    for peer_text in soup.find_all('p', class_='peerreviewed') ]
1098 pd.DataFrame(p_peervwd).to_clipboard()
1099
1100 from bs4 import SoupStrainer
1101 cont_fw_strain = SoupStrainer('div', class_='content-fullwidth')
1102
1103 # check documents with >2 <div class="syndicate">
1104 [str(synd)[:200]
1105  for soup in tqdm(pcd4_soup)
1106  for synd in soup.find('div', class_='content-fullwidth').find_all('div',
1106  class_='syndicate')
1107  if len(soup.find('div', class_='content-fullwidth').find_all('div',
1107  class_='syndicate'))>2]
1108 # 2 cases, both where 3rd is JPG
1109 # [<div class="syndicate"> <p class="text-center"> <a
1109 href="/pcd/issues/2018/images/17_0535_01.jpg" target="new"> High-resolution JPG for print
1109 <span class="sr-only"> image icon </span> <span aria-hidden="true" class="fi
1109 cdc-icon-image x16 fill-image"> </span> <span class="file-details"> </span> </a> </p>
1109 </div>,
1110 #  <div class="syndicate"> <p class="text-center"> <a
1110 href="/pcd/issues/2018/images/17_0434_large.gif" target="new"> High-resolution JPG for
1110 print <span class="sr-only"> image icon </span> <span aria-hidden="true" class="fi
1110 cdc-icon-image x16 fill-image"> </span> <span class="file-details"> </span> </a> </p>
1110 </div>]
1111
1112 def preprocess_pcd4(soup):
1113    soup = copy.copy(soup.find('div', class_='content-fullwidth'))
1114    # 3 child <div> elements of interest: syndicate, dateline, syndicate
1115    # dateline
1116    dateline = process_dateline(soup)
1117    # syndicate <div>s
1118    div_synd1, div_synd2 = soup.find_all('div', class_='syndicate')[:2]
1119    title = div_synd1.h1.get_text('|', strip=True) # syndicate 1
1120    summary = div_card_text_fn(div_synd2) # syndicate 2
1121
1122    # process the rest of syndicate 2
1123    syn_children = div_synd2.find_all(name=True, recursive=False)
1124    syn_child_attrs = [(x.name, '|'.join(x.get_attribute_list('class'))
1125                           if x.has_attr('class') else '',
1126                           x.get_text('|', strip=True)[:40])
1127                           for x in syn_children]
1128
1129    syn_child_names = [':'.join(x[:2]).rstrip(':') for x in syn_child_attrs]
1130    # Run-length encoding: initialize and iterate to completion
```

```
1131     elems = [[syn_child_names[0], 1]]
1132     for name in syn_child_names[1:]:
1133        if name == elems[-1][0]:
1134           elems[-1][1] += 1
1135        else:
1136           elems += [[name, 1]]
1137     suite = '+'.join([elem[0] if elem[1]==1 else elem[0]+'*'+str(elem[1]) for elem in
1137     elems])
1138 
1139     return dict(title=title, **dateline, summary=summary,
1140                 # syn_attrs=syn_child_attrs,
1141                 suite=suite)
1142 
1143 preprocess_pcd4(pcd4_soup[0])
1144 
1145 suites = [preprocess_pcd4(soup)['suite'] for soup in tqdm(pcd4_soup)]
1146 
1147 x = pd.DataFrame(dict(path=[html['path'] for html in pcd4_html], suite=suites))
1148 
1149 #%%
1150 
1151 # pcd4_html = pickle.load(open('pcd4_html.pkl', 'rb'))
1152 pcd4_soup = [BeautifulSoup(html['html'], 'lxml') for html in tqdm(pcd4_html)]
1153 # 916/916 [00:56<00:00, 16.08it/s]
1154 
1155 ## review <div>s that typically appear on first part of page
1156 
1157 # div:col-md-4|float-right|cr as ('col-md-4', 'cr')
1158 x = [str(div.attrs) for soup in tqdm(pcd4_soup)
1159      for div in soup.find('div', class_='content-fullwidth').\
1160         find_all('div', class_='syndicate', recursive=False)[1].\
1161         find_all('div', class_=('col-md-4', 'cr'), recursive=False)]
1162 # set(x) # {"{'class': ['col-md-4', 'float-right', 'cr']}"}
1163 #    searching on only 'col-md-4' or only on 'cr' yields the same results
1164 #    including 'float-right' does not
1165 # 153 occurrences of syndicate child <div class="col-md-4 float-right cr">
1166 #    these occurrences are unique within each file where they occur
1167 #    all "Summary" -- keep
1168 
1169 # div:card|tp-on-this-page|mb-3|w-33|float-right|d-none|d-lg-block
1170 # ('card', 'tp-on-this-page', 'mb-3', 'w-33', 'float-right', 'd-none', 'd-lg-block')
1171 x = [str(div.attrs) for soup in tqdm(pcd4_soup)
1172      for div in soup.find('div', class_='content-fullwidth').\
1173         find_all('div', class_='syndicate', recursive=False)[1].\
1174         find_all('div', class_=('tp-on-this-page', 'w-33', 'd-none', 'd-lg-block'),
1175                  recursive=False)]
1176 # set(x) # {"{'class': ['card', 'tp-on-this-page', 'mb-3', 'w-33', 'float-right',
1176 'd-none', 'd-lg-block']}"}
1177 #    or searcing on only 'tp-on-this-page', 'w-33', 'd-none', or 'd-lg-block'
1178 #    these 4 classes occur exclusively together within syndicate
1179 # 877 occurrences of syndicate child; unique in file when it exists
1180 #    <div class="card tp-on-this-page mb-3 w-33 float-right d-none d-lg-block">
1181 # one get_text: 'On This
1181 Page|Abstract|Introduction|Methods|Results|Discussion|Acknowledgments|Author
1181 Information|References|Tables'
1182 
```

```
1183 # div:row|bg-gray-l1|medscapeCME, div:medscapeCME, div:cme
1184 x = [str(div.attrs) for soup in tqdm(pcd4_soup)
1185      for div in soup.find('div', class_='content-fullwidth').\
1186         find_all('div', class_='syndicate', recursive=False)[1].\
1187         find_all('div', class_=('cme', 'medscapeCME'), recursive=False)]
1188         # find_all('div', class_=re.compile('cme', re.IGNORECASE), recursive=False)]
1189 # 46 occurrences; set(x) returns 3 unique values; unique in file when they exist
1190 # <div class="cme"> 17 times
1191 # <div class="medscapeCME"> 23 times
1192 # <div class="row bg-gray-l1 medscapeCME"> 6 times
1193
1194 # div:edNote
1195 x = [str(div.attrs) for soup in tqdm(pcd4_soup)
1196      for div in soup.find('div', class_='content-fullwidth').\
1197         find_all('div', class_='syndicate', recursive=False)[1].\
1198         find_all('div', class_=('edNote'), recursive=False)]
1199 # 4 occurrences; set(x) returns 1 unique value; unique in file when it exists
1200 # <div class="edNote">
1201 # winners of '2017 Student Research Paper Contest'
1202
1203 # div:d-block|text-center|pt-2
1204 x = [str(div.attrs) for soup in tqdm(pcd4_soup)
1205      for div in soup.find('div', class_='content-fullwidth').\
1206         find_all('div', class_='syndicate', recursive=False)[1].\
1207         # find_all('div', class_=('d-block', 'text-center', 'pt-2'),
1208         find_all('div', class_=('pt-2',), recursive=False)]
1209 # 373 for ('d-block', 'text-center', 'pt-2'); 372 for ('pt-2',)
1210 # {y: x.count(y) for y in sorted(set(x))}
1211 # "{'class': ['d-block', 'text-center', 'pt-2']}"       366 times
1212 # "{'class': ['d-block', 'text-center']}"                 1 time - omit
1213 # "{'class': ['float-md-left', 'cl', 'mr-md-3', 'pt-2']}" 6 times
1214 x = [len(soup.find('div', class_='content-fullwidth').\
1215         find_all('div', class_='syndicate', recursive=False)[1].\
1216         find_all('div', class_=('pt-2',), recursive=False))
1217      for soup in tqdm(pcd4_soup)]
1218 # {y: x.count(y) for y in sorted(set(x))}
1219 # occurs 0-4 times per document: {0: 665, 1: 148, 2: 88, 3: 12, 4: 3}
1220 # 372 strings in 251 files
1221 x = [soup.find('div', class_='content-fullwidth').\
1222         find_all('div', class_='syndicate', recursive=False)[1].\
1223         find_all('div', class_=('pt-2',), recursive=False)
1224      for soup in tqdm(pcd4_soup)]
1225 # [[z.img['alt'].strip() for z in y] for y in x if len(y) > 2] # 15 examples
1226 # [z.img['alt'].strip() for y in x for z in y if len(y) > 2] # single list of 48
1227
1228 # alt_exclude is from below; exclusions remove only 3 instances
1229 xx = [[z.img['alt'].strip(' \n') for z in y
1230        if z.img['alt'].strip(' \n') not in alt_exclude]
1231       for y in x]
1232 # { y: [len(z) for z in xx].count(y) for y in range(5) }
1233 # {0: 668, 1: 145, 2: 88, 3: 12, 4: 3}
1234
1235 # harvest <img alt=""> attribute text for these
1236 # <img> alt attribute text for images under <div class="pt-2">
1237 xx = [div.find('img').get('alt') if div.find('img') is not None else ''
1238       for soup in tqdm(pcd4_soup)
```

```
1239       for div in soup.find('div', class_='content-fullwidth').\
1240         find_all('div', class_='syndicate', recursive=False)[1].\
1241         find_all('div', class_=('pt-2',), recursive=False)]
1242
1243 # <img> alt attribute text for all images under <div class="syndicate">
1244 alt_text = [img.get('alt').strip(' \n') for soup in tqdm(pcd4_soup)
1245    for img in soup.find('div', class_='content-fullwidth').\
1246       find_all('div', class_='syndicate', recursive=False)[1].\
1247       find_all('img', alt=True)]
1248 # len(alt_text); len(set(alt_text)) # 2887 images, 770 unique strings
1249
1250 alt_text_freq = {text: alt_text.count(text) for text in sorted(set(alt_text))}
1251 set(alt_text_freq.values()) # {1, 2, 3, 4, 8, 16, 2080}
1252 [(freq, alt_text) for alt_text, freq in alt_text_freq.items() if freq > 1]
1253 # [(3, ''),
1254 #  (2, '14_0299'),
1255 #  (2, '14_0378'),
1256 #  (2, 'A map of New York State counties shows geographic distribution of obesity among
1256 men based on ordinary least squares (OLS) coefficients. OLS coefficients are larger in the
1256 southeast and smaller in the west.'),
1257 #  (2, 'Change in patients' general knowledge of diabetes over time, measured with the
1257 Diabetes, Hypertension and Hyperlipidemia (DHL) knowledge instrument (15), for patients
1257 participating in the health coaching program. The change in score (possible range, 0–28)
1257 was assessed over time in A) all patients (n = 238), and in B) patients who completed the
1257 assessment at all time points. Scores for A at each time point after baseline were
1257 compared with baseline scores by using the Wilcoxon matched-pairs signed-rank test. Scores
1257 for B at each time point after baseline were compared with baseline scores by using the
1257 Friedman test. Error bars indicate standard deviation. '),
1258 #  (3, 'Changes in Density of On-Premises'),
1259 #  (16, 'Joint Accredited Provider Interprofessional Continuing Education '),
1260 #  (4, 'Jointly Accredited Provider Interprofessional Continuing Education Logo'),
1261 #  (3, 'Leonard Jack'),
1262 #  (8, 'Leonard_Jack'),
1263 #  (2, 'Mean Sitting Time, by United Hospital Fund, 34 Neighborhoods in New York City,
1263 Physical Activity and Transit Survey,'),
1264 #  (2080, 'Return to your place in the text'),
1265 #  (3, 'Three advertisements used in New York State Department of Health promotion of
1265 tobacco cessation patient interventions among health care providers. ')]
1266 # plus 757 unique strings # sum([freq for alt_text, freq in alt_text_freq.items() if freq
1266 == 1])
1267 pd.DataFrame([alt_text for alt_text, freq in alt_text_freq.items() if freq ==
1267 1]).to_clipboard()
1268
1269 # [text for text, freq in alt_text_freq.items() if freq > 1 and len(text) > 10 and
1269 len(text) < 100]
1270 # rule to remove noninformative values
1271 alt_exclude = ('Joint Accredited Provider Interprofessional Continuing Education',
1272    'Jointly Accredited Provider Interprofessional Continuing Education Logo',
1273    'Leonard Jack', 'Leonard_Jack', 'Return to your place in the text', '')
1274 alt_text = [x for x in alt_text if len(x) > 10 and x not in alt_exclude]
1275 # len(alt_text); len(set(alt_text)) # 753 and 746, down from 2887 and 770
1276 pd.DataFrame([dict(len=len(x), alt_text=x) for x in alt_text]).to_clipboard()
1277
1278 alt_text_elems = [img for soup in tqdm(pcd4_soup)
1279    for img in soup.find('div', class_='content-fullwidth').\
1280       find_all('div', class_='syndicate', recursive=False)[1].\
```

```
1281       find_all('img', alt=True)
1282       if len(img.get('alt').strip(' \n')) > 10
1283          and img.get('alt').strip(' \n') not in alt_exclude]
1284
1285 x = [(depth(y), y.parent.name) for y in alt_text_elems]
1286 {y: x.count(y) for y in sorted(set(x))}
1287 {(6, 'div'): 22, (7, 'center'): 9, (7, 'div'): 374, (7, 'p'): 336,
1288  (8, 'a'): 5, (10, 'td'): 7}
1289
1290 def branch(object, ancestor_name = 'body'):
1291    branch = [object.name]
1292    for depth, ancestor in enumerate(object.parents, start=1):
1293       branch = [ancestor.name] + branch # could use deque
1294       if ancestor.name == ancestor_name: break
1295    if ancestor.name is None: depth = None
1296    return dict(depth = depth, branch='>'.join(branch))
1297
1298 # branch(alt_text_elems[0])
1299 x = [branch(elem) for elem in alt_text_elems]
1300 xx = [tuple(y.values()) for y in x]
1301 {y: xx.count(y) for y in sorted(set(xx))}
1302 # {( 6, 'body>div>main>div>div>div>img'):                  22,
1303 #  ( 7, 'body>div>main>div>div>div>center>img'):            9,
1304 #  ( 7, 'body>div>main>div>div>div>div>img'):             374,
1305 #  ( 7, 'body>div>main>div>div>div>p>img'):               336,
1306 #  ( 8, 'body>div>main>div>div>div>div>a>img'):             3,
1307 #  ( 8, 'body>div>main>div>div>div>p>a>img'):               2,
1308 #  (10, 'body>div>main>div>div>div>table>tbody>tr>td>img'): 7}
1309 # depth 6 is child of <div class="syndicate">
1310
1311 # example of 2nd div:syndicate, race lineage up and down
1312 y = pcd4_soup[0].find('div', class_='content-fullwidth').\
1313       find_all('div', class_='syndicate', recursive=False)[1]
1314 branch(y)
1315 # {'depth': 5, 'branch': 'body>div>main>div>div>div'}
1316
1317 (lambda x: (x.name, x.attrs))(y)
1318 ('div', {'class': ['syndicate']})
1319 (lambda x: (x.name, x.attrs))(y.parent)
1320 ('div', {'class': ['col', 'content', 'content-fullwidth']})
1321  (lambda x: (x.name, x.attrs))(y.parent.parent)
1322 ('div', {'class': ['row']})
1323 (lambda x: (x.name, x.attrs))(y.parent.parent.parent)
1324 ('main', {'class': ['col-12', 'order-lg-2']})
1325 (lambda x: (x.name, x.attrs))(y.parent.parent.parent.parent)
1326 ('div', {'class': ['container', 'd-flex', 'flex-wrap', 'body-wrapper', 'bg-white']})
1327 (lambda x: (x.name, x.attrs))(y.parent.parent.parent.parent.parent)
1328 ('body', {'class': ['no-js']})
1329 (lambda x: (x.name, x.attrs))(y.parent.parent.parent.parent.parent.parent)
1330 ('html', {'class': ['theme-green'], 'lang': 'en-us'})
1331
1332 def branch2(object, ancestor_name = 'body'):
1333    branch = [(object.name, object.attrs)]
1334    for depth, ancestor in enumerate(object.parents, start=1):
1335       branch = [(ancestor.name, ancestor.attrs)] + branch # could use deque
1336       if ancestor.name == ancestor_name: break
```

```
1337     if ancestor.name is None: depth = None
1338     return dict(depth = depth, branch = branch)
1339 branch2(y.h4)
1340 
1341 # gather ancestry for all <img>s
1342 x = [branch2(elem) for elem in alt_text_elems]
1343 # trim by removing ancestry through <div class="syndicate"> and terminal <img>
1344 xx = [(y['depth']-6, y['branch'][6:-1]) for y in x]
1345 xy = [(y[0], str(y[1])) for y in xx]
1346 # xx = [tuple(y.values()) for y in x]
1347 {y: xy.count(y) for y in sorted(set(xy))}
1348 
1349 x = [(i, j, h2.get_text('|', strip=True))
1350     for i, soup in tqdm(enumerate(pcd4_soup), total=916)
1351     for j, h2 in enumerate(soup.find('div', class_='content-fullwidth').\
1352        find_all('div', class_='syndicate', recursive=False)[1].\
1353        find_all('h2'))
1354     if soup.find('div', class_='content-fullwidth').\
1355          find('div', class_=('cme', 'medscapeCME'))
1356          is not None]
1357 
1358     x = [str(div.attrs) for soup in tqdm(pcd4_soup)
1359       for div in s]
1360 
1361 
1362 block_names = ('h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'p', 'ul', 'ol', 'blockquote')
1363 
1364 def process_pcd4(soup):
1365     soup_ = copy.copy(soup.find('div', class_='content_fullwidth'))
1366     div_synd1, div_dateline, div_synd2 = soup_.find_all('div',
1367        class_=('syndicate', 'dateline'))[:3]
1368     # div_synd1: title
1369     # div_dateline: article type, volume number, publication date
1370     # div_synd2: authors, citation, peer-reviewed, summary; content
1371     pass
1372 
1373 
1374 #%%
1375 from bs4 import Comment, Script, Stylesheet
1376 def count_subs(elem):
1377     elem_name = elem.name
1378     elem_parent_name = '' if elem.parent.name is None else elem.parent.name
1379     elem_depth = depth(elem)
1380     elem_descendants = list(elem.descendants)
1381     elem_desc_types = [str(type(desc)) for desc in elem_descendants]
1382     elem_desc_type_freq = { type_: elem_desc_types.count(type_)
1383                             for type_ in sorted(set(elem_desc_types)) }
1384     elem_desc_names = ['' if desc.name is None else desc.name
1385                        for desc in elem_descendants]
1386     elem_desc_name_freq = { name_: elem_desc_names.count(name_)
1387                             for name_ in sorted(set(elem_desc_names)) }
1388     div_classes = ['' if class_ is None else class_
1389                    for div_elem in elem.find_all('div')
1390                    for class_ in div_elem.get_attribute_list('class')]
1391     div_class_freq = { class_: div_classes.count(class_)
1392                        for class_ in sorted(set(div_classes)) }
```

```
1393
1394     h_classes = ['' if class_ is None else class_
1395                  for h_elem in elem.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
1396                  for class_ in h_elem.get_attribute_list('class')]
1397     h_class_freq = { class_: h_classes.count(class_)
1398                      for class_ in sorted(set(h_classes)) }
1399     h_depths = [depth(h_elem) for h_elem in elem.find_all(['h1', 'h2', 'h3', 'h4', 'h5',
1399     'h6'])]
1400     h_depth_min = min(h_depths, default = -1)
1401     h_depth_max = max(h_depths, default = -1)
1402
1403     p_classes = ['' if class_ is None else class_
1404                  for p_elem in elem.find_all('p')
1405                  for class_ in p_elem.get_attribute_list('class')]
1406     p_class_freq = { class_: p_classes.count(class_)
1407                      for class_ in sorted(set(p_classes)) }
1408     p_depths = [depth(p_elem) for p_elem in elem.find_all('p')]
1409     p_depth_min = min(p_depths, default = -1)
1410     p_depth_max = max(p_depths, default = -1)
1411     p_topstrings = [p_elem
1412                          for p_elem in elem.find_all(name=['p', 'span'])
1413                          if p_elem.a is not None
1414                          and p_elem.a.get_text() is not None
1415                          and p_elem.a.get_text(strip=True) in pcd_topstrings]
1416     n_topstrings = len(p_topstrings)
1417     p_ts_depths = [depth(p_elem) for p_elem in p_topstrings]
1418     p_ts_depth_min = min(p_ts_depths, default=-1)
1419     p_ts_depth_max = max(p_ts_depths, default=-1)
1420     n_form_eMailer = len(elem.find_all('form', attrs={'name': 'eMailer'}))
1421     n_div_onthispageChrono = len(elem.find_all('div', class_='onthispageChrono'))
1422     n_div_tponthispage = len(elem.find_all('div', class_='tp-on-this-page'))
1423     n_span_es = len(elem.find_all('span', lang='es'))
1424     return dict(elem_name=elem_name, elem_parent_name=elem_parent_name,
1425        elem_depth=elem_depth, elem_desc_names=elem_desc_name_freq,
1426        elem_desc_types=elem_desc_type_freq,
1427        div_classes=div_class_freq,
1428        p_classes=p_class_freq,
1429        p_depth_min=p_depth_min, p_depth_max=p_depth_max,
1430        n_topstrings=n_topstrings,
1431        p_ts_depth_min=p_ts_depth_min, p_ts_depth_max=p_ts_depth_max,
1432        n_form_eMailer=n_form_eMailer,
1433        n_div_onthispageChrono=n_div_onthispageChrono,
1434        n_div_tponthispage=n_div_tponthispage,
1435        n_span_es=n_span_es)
1436
1437
1438 # Inventory components of containers
1439 def pcd_container_components(soup):
1440     td_w_80 = soup.find_all(td_w_80_fn)
1441     div_main_inner = soup.find_all('div', class_='main-inner')
1442     div_content_fw = soup.find_all('div', class_='content-fullwidth')
1443
1444     # the following 3 conditions are mutually exclusive
1445     if len(td_w_80):
1446        components = ['td_w_80'] + [count_subs(elem) for elem in td_w_80]
1447     elif len(div_main_inner):
```

```
1448        components = ['div_main_inner'] + [count_subs(elem) for elem in div_main_inner]
1449     elif len(div_content_fw):
1450        components = ['div_content_fw'] + [count_subs(elem) for elem in div_content_fw]
1451     # else:
1452     #    pass
1453
1454     # div_syndicate = len(soup.find_all('div', class_='syndicate'))
1455     return components
1456
1457 # pcd_container_freq_ = [pcd_container_components(soup) for soup in tqdm(pcd_art_soup)]
1458 # 3542/3542 [01:18<00:00, 44.94it/s]
1459 # confirmed that the 3 categories are mutually exclusive and exhaustive
1460
1461 pcd_container_list = [pcd_container_components(soup) for soup in tqdm(pcd_art_soup)]
1462 # 3542/3542 [02:07<00:00, 27.81it/s]
1463 # pickle.dump(pcd_container_list, open("pcd_container_list.pkl", "wb"))
1464
1465 [pcd_container_list[i] for i in [73, 968, 2533, 1554, 2900, 1645, 2963, 2488]]
1466
1467 # set([len(x) for x in pcd_container_list])
1468 { j: [len(x) for x in pcd_container_list].count(j) for j in range(2, 9) }
1469 # {2: 1844, 3: 5, 4: 1057, 5: 623, 6: 9, 7: 3, 8: 1}
1470
1471 { struct_: [x[0] for x in pcd_container_list].count(struct_)
1472            for struct_ in ('td_w_80', 'div_main_inner', 'div_content_fw') }
1473 # {'td_w_80': 1698, 'div_main_inner': 928, 'div_content_fw': 916}
1474
1475 # single-value elements
1476 { k: v for k, v in pcd_container_list[0][1].items() if type(v) is not dict }
1477 # ['elem_name', 'elem_parent_name', 'elem_depth', 'p_depth_min', 'p_depth_max',
1478 #  'n_topstrings', 'p_ts_depth_min', 'p_ts_depth_max', 'n_form_eMailer',
1479 #  'n_div_onthispageChrono', 'n_div_tponthispage', 'n_span_es']
1480
1481 # dictionary-value elements
1482 { k: v for k, v in pcd_container_list[0][1].items() if type(v) is dict }
1483 # ['elem_desc_names', 'elem_desc_types', 'div_classes', 'p_classes']
1484
1485 pcd_comps_per_file = [len(x)-1 for x in pcd_container_list]
1486 { j: pcd_comps_per_file.count(j) for j in range(1, 8) }
1487 # {1: 1844, 2: 5, 3: 1057, 4: 623, 5: 9, 6: 3, 7: 1}
1488
1489 pcd_comps_dframe = pd.concat([
1490    pd.DataFrame(
1491       [dict(path = path, year = int(path[12:16]), container = comps[0])
1492           for path, comps in zip(pcd_art_frame.mirror_path, pcd_container_list)
1493           for comp_dict in comps[1:]]),
1494    pd.DataFrame(
1495       [{ k: v for k, v in comp_dict.items() if type(v) is not dict }
1496          for comps in pcd_container_list
1497          for comp_dict in comps[1:]])
1498    ], axis=1)
1499 # [7587 rows x 15 columns]
1500 pcd_comps_dframe.to_excel('pcd_comps_dframe.xlsx', engine='openpyxl')
1501
1502 pd.concat([
1503    pcd_comps_dframe.iloc[:, 0:3],
```

```
1504     pd.DataFrame(
1505        [comp_dict['elem_desc_names']
1506           for comps in pcd_container_list
1507           for comp_dict in comps[1:]]).fillna(0)
1508     ], axis=1).to_excel('pcd_comps_desc_names.xlsx', engine='openpyxl')
1509  # [7587 rows x 777 columns]
1510  pd.concat([
1511     pcd_comps_dframe.iloc[:, 0:3],
1512     pd.DataFrame(
1513        [comp_dict['elem_desc_types']
1514           for comps in pcd_container_list
1515           for comp_dict in comps[1:]]).fillna(0)
1516     ], axis=1).to_excel('pcd_comps_desc_types.xlsx', engine='openpyxl')
1517  # [7587 rows x 6 columns]
1518  pd.concat([
1519     pcd_comps_dframe.iloc[:, 0:3],
1520     pd.DataFrame(
1521        [comp_dict['p_classes']
1522           for comps in pcd_container_list
1523           for comp_dict in comps[1:]]).fillna(0)
1524     ], axis=1).to_excel('pcd_comps_p_classes.xlsx', engine='openpyxl')
1525  # [7587 rows x 44 columns]
1526  pd.concat([
1527     pcd_comps_dframe.iloc[:, 0:3],
1528     pd.DataFrame(
1529        [comp_dict['div_classes']
1530           for comps in pcd_container_list
1531           for comp_dict in comps[1:]]).fillna(0)
1532     ], axis=1).to_excel('pcd_comps_div_classes.xlsx', engine='openpyxl')
1533  # [7587 rows x 116 columns]
1534
1535  # Test conditions on child element and its descendants
1536  def child_keep(child):
1537     child_truth = child.name is not None \
1538        and (child.name in {'h1', 'h2', 'h3', 'h4', 'h5', 'h6'}
1539           or child.find(['h1', 'h2', 'h3', 'h4', 'h5', 'h6']) is not None
1540           or (child.name == 'div' and child.has_attr('class')
1541              and {'syndicate', 'dateline'}.intersection(child.get('class')))
1542           or child.find('div', class_=['syndicate', 'dateline']) is not None)
1543     return child_truth
1544
1545  # Inventory components of containers
1546  def pcd_container_components2(path, soup):
1547     td_w_80 = soup.find_all(td_w_80_fn)
1548     div_main_inner = soup.find_all('div', class_='main-inner')
1549     div_content_fw = soup.find_all('div', class_='content-fullwidth')
1550
1551     if len(td_w_80):
1552        for i, elem in enumerate(td_w_80):
1553           for j, child in enumerate(elem.children):
1554              if child.name is not None:
1555                 if child_keep(child):
1556                    td_width_80_in.write(f'<!--{path} elem {i}, child {j},
1556                    {child.name}-->\n')
1557                    td_width_80_in.write(child.prettify())
1558                 else:
```

```
1559                    td_width_80_out.write(f'<!--{path} elem {i}, child {j},
1559                    {child.name}-->\n')
1560                    td_width_80_out.write(child.prettify())
1561
1562     elif len(div_main_inner):
1563        # div_main_inner should be list of length 1
1564        for i, elem in enumerate(div_main_inner):
1565           for j, child in enumerate(elem.children):
1566              if child.name is not None:
1567                 if child_keep(child):
1568                    div_main_inner_in.write(f'<!--{path} elem {i}, child {j},
1568                    {child.name}-->\n')
1569                    div_main_inner_in.write(child.prettify())
1570                 else:
1571                    div_main_inner_out.write(f'<!--{path} elem {i}, child {j},
1571                    {child.name}-->\n')
1572                    div_main_inner_out.write(child.prettify())
1573
1574     elif len(div_content_fw):
1575        # div_content_fw should be list of length 1
1576        for i, elem in enumerate(div_content_fw):
1577           for j, child in enumerate(elem.children):
1578              if child.name is not None:
1579                 if child_keep(child):
1580                    div_content_fw_in.write(f'<!--{path} elem {i}, child {j},
1580                    {child.name}-->\n')
1581                    div_content_fw_in.write(child.prettify())
1582                 else:
1583                    div_content_fw_out.write(f'<!--{path} elem {i}, child {j},
1583                    {child.name}-->\n')
1584                    div_content_fw_out.write(child.prettify())
1585
1586     else:
1587        other_components.write('<!--' + path + '-->\n')
1588
1589     # div_syndicate = len(soup.find_all('div', class_='syndicate'))
1590     return None
1591
1592  div_content_fw_in  = open('div_content-fullwidth_in.html', 'w')
1593  div_content_fw_out = open('div_content-fullwidth_out.html', 'w')
1594  div_main_inner_in  = open('div_main-inner_in.html', 'w')
1595  div_main_inner_out = open('div_main-inner_out.html', 'w')
1596  td_width_80_in     = open('td_width_80_in.html', 'w')
1597  td_width_80_out    = open('td_width_80_out.html', 'w')
1598  other_components   = open('other-components.html', 'w')
1599
1600  for file_ in [div_content_fw_in, div_content_fw_out,
1601     div_main_inner_in, div_main_inner_out, td_width_80_in, td_width_80_out,
1602     other_components]:
1603     file_.write('<html><body>\n')
1604
1605  for path, soup in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_soup), total=3542):
1606     pcd_container_components2(path, soup)
1607  # 3542/3542 [02:16<00:00, 25.87it/s]
1608
1609  for file_ in [div_content_fw_in, div_content_fw_out,
```

```
1610     div_main_inner_in, div_main_inner_out, td_width_80_in, td_width_80_out,
1611     other_components]:
1612     file_.write('</body></html>\n')
1613
1614 div_content_fw_in.close()
1615 div_content_fw_out.close()
1616 div_main_inner_in.close()
1617 div_main_inner_out.close()
1618 td_width_80_in.close()
1619 td_width_80_out.close()
1620 other_components.close()
1621 del div_content_fw_in, div_content_fw_out, div_main_inner_in, div_main_inner_out, \
1622     td_width_80_in, td_width_80_out, other_components, file_
1623
1624 #%% Reduce each html.body soup for further processing
1625
1626 # 1. Reduce to smallest element enclosing content of interest
1627 # 2. Remove remaining components of no interest (e.g., navigaion <div>s)
1628 # 3. Write reduced soup as html strings to new list object; pickle it
1629 # 4. Reread reduced-soup html and parse into soup
1630 # 5. Assess equivalence of reduced html to html->soup->html (lose any info?)
1631
1632 # When <main> occurs in a file, it occurs exactly once
1633 def main_descendants(soup):
1634     import copy
1635     soup_ = copy.copy(soup)
1636     try:
1637         content = soup_.find('main').find_all('div', class_=['syndicate', 'dateline'])
1638         div_ml_3 = soup_.find('div', class_='ml-3')
1639         if div_ml_3 is not None: div_ml_3.decompose()
1640         div_tp_page = soup_.find('div', class_='tp-on-this-page')
1641         if div_tp_page is not None: div_tp_page.decompose()
1642         content = re.sub(r'<!--.*?-->', '', ' '.join([str(elem) for elem in content]))
1643         content = re.sub(r'\s+', ' ', content)
1644     except:
1645         content = None
1646     return content
1647
1648 # When <div class="main-inner"> occurs in a file, it occurs exactly once
1649 from bs4 import Comment
1650
1651 # use BeautifulSoup to extract <div class="main-inner">
1652 # convert to string, use regex to remove comments, parse back to soup
1653 # remove elements <div class="onthispageChrono">
1654 # convert back to string
1655
1656 def main_inner_descendants(soup):
1657     try:
1658         content = soup.find('div', class_='main-inner')
1659         soup_ = BeautifulSoup(re.sub(r'<!--.*?-->', '', str(content)), 'lxml').\
1660             find('div', class_='main-inner')
1661         if soup_.find('div', class_='onthispageChrono') is not None:
1662             for j in range(len(soup_.find_all('div', class_='onthispageChrono'))):
1663                 soup_.find('div', class_='onthispageChrono').decompose()
1664         content = re.sub(r'\s+', ' ', str(soup_))
1665     except:
```

```
1666        content = ''
1667     return content
1668
1669 xx = copy.copy(pcd_art_soup[1088]) # len(str(xx)) # 26663
1670 yy = main_inner_descendants(xx) # len(_) # 8491
1671 # str(BeautifulSoup(main_descendants(x), 'lxml'))
1672
1673 [(i, path) for i, (path, soup) in enumerate(zip(pcd_art_frame.mirror_path, pcd_art_soup))
1674  if len(soup.find_all('div', class_='onthispageChrono')) > 4]
1675 xx = copy.copy(pcd_art_soup[1088])
1676
1677 len(xx.find_all('div', class_='onthispageChrono')) # 5
1678 xx.find('div', class_='onthispageChrono').decompose()
1679 len(xx.find_all('div', class_='onthispageChrono')) # 4
1680 xx.find('div', class_='onthispageChrono').decompose()
1681 len(xx.find_all('div', class_='onthispageChrono')) # 3
1682 xx.find('div', class_='onthispageChrono').decompose()
1683 len(xx.find_all('div', class_='onthispageChrono')) # 2
1684 xx.find('div', class_='onthispageChrono').decompose()
1685 len(xx.find_all('div', class_='onthispageChrono')) # 1
1686 xx.find('div', class_='onthispageChrono').decompose()
1687 len(xx.find_all('div', class_='onthispageChrono')) # 0
1688
1689 xx = copy.copy(pcd_art_soup[1088])
1690 for j in range(len(xx.find_all('div', class_='onthispageChrono'))):
1691    xx.find('div', class_='onthispageChrono').decompose()
1692    print(len(xx.find_all('div', class_='onthispageChrono')))
1693
1694 def pcd_soup_body(soup):
1695    has_main = (soup.main is not None)
1696    has_main_inner = (soup.find('div', class_='main-inner') is not None)
1697    has_width_80 = (soup.find('td', width='80%') is not None)
1698
1699
1700
1701
1702
1703 #%% miscellaneous bits from past efforts -- junk yard
1704
1705 z = za_toptext # 21957
1706 z_str = sorted([str(tag) for tag in z])
1707 z_text = sorted([tag.get_text('|', strip=True) for tag in z])
1708 z_str_freq = { item: z_str.count(item) for item in sorted(set(z_str)) } # 11
1709 z_text_freq = { item: z_text.count(item) for item in sorted(set(z_text)) } # 7
1710
1711 # [text for text in sorted(set(z_text)) if len(text) <= 13]
1712 # ['Back to top', 'Top', 'Top of Page']
1713 za_toptext_ = [tag for tag in za_toptext
1714    if tag.get_text('|', strip=True) in {'Back to top', 'Top', 'Top of Page'}]
1715 # 21037
1716
1717 set([za.parent.name for za in za_toptext_]) # {'div', 'p', 'span'}
1718 [[za.parent.name for za in za_toptext_].count(elem)
1719    for elem in ['div', 'p', 'span']]
1720 # [1, 21020, 16]
1721
```

```
1722
1723 za_toptext = [tag for soup in tqdm(pcd_art_soup)
1724                   for tag in soup.find_all('a', href=True)
1725                   if tag.get_text() is not None and
1726                      re.search(r'\btop\b', tag.get_text(), re.I)]
1727
1728 [len(x) for x in set([z.get_text('|', strip=True) for z in z_all_toptext])]
1729 [len(x) for x in set([str(z) for z in z_all_toptext])]
1730 with open('z_all_toptext-str.txt', 'w') as file_out:
1731    for z_s in sorted(set([str(z) for z in z_all_toptext])):
1732       file_out.write(f'{z_s}\n')
1733
1734
1735 x_attrs = [tag.attrs for tag in x.descendants if tag.name is not None]
1736 len(x.main) # 6
1737 len(list(x.main.children)) # 6
1738
1739 import json
1740 with open('pcd-example-attrs.json', 'w') as json_out:
1741    json.dump(x_attrs, json_out)
1742
1743 # a bit that is not in
1744 x.find(name='div', class_='bl-primary').get_text('|', strip=True)
1745
1746
1747 # need to figure out Spanish version(s), so see full English versions
1748 def top_list(tag):
1749    # cond1 = tag.name in ('h1', 'h2', 'h3', 'h4', 'h5', 'h6')
1750    cond2 = (tag.name == 'p') and \
1751       (tag.get_text(strip=True) in ('Back to top', 'Top', 'Top of Page'))
1752    return cond1 or cond2
1753
1754 pcd_body_p_tops = [str(tag) for html in tqdm(pcd_art_html)
1755                     for tag in BeautifulSoup(html, 'lxml').find_all('p')
1756                     if tag.get_text(strip=True) in ('Back to top', 'Top', 'Top of Page')]
1757 { elem: pcd_body_p_tops.count(elem) for elem in sorted(set(pcd_body_p_tops)) }
1758
1759 def top_list(tag):
1760    p_classes = {'float-right', 'psmall', 'topOPage'} # p classes
1761    a_hrefs = {'#', '#toTop', '#ttop', '#top'} # a hrefs
1762    top_strings = {'Back to top', 'Top', 'Top of Page'}
1763    cond1 = (tag.name == 'p') and tag.find('p', class_=p_classes)
1764    cond2 = (tag.name == 'a') and tag.find('a', href=a_hrefs)
1765    cond3 = (tag.name == 'p') and \
1766       (tag.get_text(strip=True) in top_strings)
1767    return cond1 or cond2 or cond3
1768
1769 p_classes = {'float-right', 'psmall', 'topOPage'} # p classes
1770 a_hrefs = {'#toTop', '#ttop', '#top'} # a hrefs; also '#'
1771 top_strings = {'Back to top', 'Top', 'Top of Page'}
1772 # top_strings_re = {re.compile(string) for string in top_strings}
1773 top_strings_re = re.compile(r'.*(top|volver|inicio|comienzo).*', re.I)
1774 # {re.compile(string) for string in ['top', 'Top', 'Volver', 'Inicio', 'inicio',
1774 'comienzo']}
1775 # ['top', 'Top', 'Volver', 'Inicio', 'inicio', 'comienzo']
1776
```

```
1777 zp_topOPage = [tag for html in tqdm(pcd_art_html)
1778                    for tag in BeautifulSoup(html, 'lxml').find_all('p',
1778                    class_='topOPage')]
1779 # 3540/3540 [03:03<00:00, 19.25it/s]
1780 z_uniq = sorted(set([str(zz) for zz in zp_topOPage])) # 4
1781 ['<p class="topOPage"> </p>',
1782  '<p class="topOPage"> <a href="#"> Inicio de la página </a> </p>',
1783  '<p class="topOPage"> <a href="#"> Top of Page </a> </p>',
1784  '<p class="topOPage"> <a href="#"> Volver al comienzo </a> </p>']
1785 z = [str(zz) for zz in zp_topOPage] # 4
1786 { string: z.count(string) for string in z_uniq }
1787 sorted(set([zz.get_text(strip=True) for zz in zp_topOPage])) # 4
1788 ['', 'Inicio de la página', 'Top of Page', 'Volver al comienzo']
1789
1790 za_top = [tag for html in tqdm(pcd_art_html)
1791                    for tag in BeautifulSoup(html, 'lxml').find_all('a', href=a_hrefs)]
1792 # 3540/3540 [03:41<00:00, 15.95it/s]
1793 z_uniq = sorted(set([str(zz) for zz in za_top])) # 8
1794 ['<a class="psmall" href="#top"> Back to top </a>',
1795  '<a href="#top"> Back to top </a>',
1796  '<a href="#top"> Inicio de página </a>',
1797  '<a href="#top"> Volver al Inicio </a>',
1798  '<a href="#top"> Volver al comienzo </a>',
1799  '<a href="#top"> Volver al comienzoo </a>',
1800  '<a href="#top"> Volver al inicio </a>',
1801  '<a href="#ttop"> Back to top </a>']
1802 z = [str(zz) for zz in za_top] # 4
1803 { string: z.count(string) for string in z_uniq }
1804 sorted(set([zz.get_text(strip=True) for zz in za_top])) # 6
1805 ['Back to top', 'Inicio de página', 'Volver al Inicio', 'Volver al comienzo',
1806  'Volver al comienzoo', 'Volver al inicio']
1807
1808 # check string, which is not as meaningful as checking get_text()
1809 zstr_top_a = [tag for html in tqdm(pcd_art_html)
1810                    for tag in BeautifulSoup(html, 'lxml').\
1811                       find_all('a', string=top_strings_re)]
1812 # 3540/3540 [03:09<00:00, 18.69it/s]
1813 z_uniq0 = set([str(zz) for zz in zstr_top_a if len(str(zz)) < 55]) # 16
1814 z_uniq1 = set([str(zz) for zz in zstr_top_a if len(zz.string) < 30]) # 18
1815 # sorted(z_uniq0 & z_uniq1) # intersection # 14
1816 ['<a class="psmall" href="#top"> Back to top </a>',
1817  '<a class="tp-link-policy" href="#"> Top </a>',
1818  '<a href="#"> Back to top </a>',
1819  '<a href="#"> Inicio de la página </a>',
1820  '<a href="#"> Top </a>',
1821  '<a href="#"> Top of Page </a>',
1822  '<a href="#"> Volver al comienzo </a>',
1823  '<a href="#top"> Back to top </a>',
1824  '<a href="#top"> Inicio de página </a>',
1825  '<a href="#top"> Volver al Inicio </a>',
1826  '<a href="#top"> Volver al comienzo </a>',
1827  '<a href="#top"> Volver al comienzoo </a>',
1828  '<a href="#top"> Volver al inicio </a>',
1829  '<a href="#ttop"> Back to top </a>']
1830 z = [str(zz) for zz in zstr_top_a]
1831 { string: z.count(string) for string in sorted(z_uniq0 & z_uniq1) }
```

```
sorted(set([zz.get_text(strip=True) for zz in zstr_top_a
            if len(zz.get_text(strip=True)) < 25])) # 0
['Back to top', 'Inicio de la página', 'Inicio de página', 'Top',
 'Top of Page', 'Volver al Inicio', 'Volver al comienzo',
 'Volver al comienzoo', 'Volver al inicio']

top_strings = {'Back to top', 'Inicio de la página', 'Inicio de página', 'Top',
   'Top of Page', 'Volver al Inicio', 'Volver al comienzo',
   'Volver al comienzoo', 'Volver al inicio'}

zstr_top_p = [tag for html in tqdm(pcd_art_html[:10])
                    for tag in BeautifulSoup(html, 'lxml').\
                       find_all('p')#, string=top_strings_re)
                       if tag.a is not None and \
                          tag.a.get_text(strip=True) in top_strings]
z_uniq = sorted(set([str(zz) for zz in zstr_top_p
            if len(zz.get_text(strip=True)) < 50]))
# 3540/3540 [06:36<00:00,  8.94it/s]
['<p align="left" class="psmall"> <a href="#"> Back to top </a> </p>',
 '<p class="caption"> <a href="#top"> Volver al comienzo </a> </p>',
 '<p class="float-right"> <a href="#"> Top </a> </p>',
 '<p class="float-right"> <span class="text-right d-block"> <span class="icon-angle-up">
 <!-- --> </span> <a href="#"> Top </a> </span> </p>',
 '<p class="psmall" dir="ltr"> <a href="#"> Back to top </a> </p>',
 '<p class="psmall" dir="ltr"> <a href="#top"> Volver al comienzo </a> </p>',
 '<p class="psmall"> <a href="#"> Back to top </a> </p>',
 '<p class="psmall"> <a href="#"> Back to top </a> ] </p>',
 '<p class="psmall"> <a href="#top"> Back to top </a> </p>',
 '<p class="psmall"> <a href="#top"> Inicio de página </a> </p>',
 '<p class="psmall"> <a href="#top"> Volver al Inicio </a> </p>',
 '<p class="psmall"> <a href="#top"> Volver al comienzo </a> </p>',
 '<p class="psmall"> <a href="#top"> Volver al comienzoo </a> </p>',
 '<p class="psmall"> <a href="#top"> Volver al inicio </a> </p>',
 '<p class="psmall"> <a href="#ttop"> Back to top </a> </p>',
 '<p class="pull-right"> <span class="toTop"> <span class="icon-angle-up"> <!-- -->
 </span> <a class="tp-link-policy" href="#"> Top </a> </span> </p>',
 '<p class="topOPage"> <a href="#"> Inicio de la página </a> </p>',
 '<p class="topOPage"> <a href="#"> Top of Page </a> </p>',
 '<p class="topOPage"> <a href="#"> Volver al comienzo </a> </p>',
 '<p> <a class="psmall" href="#top"> Back to top </a> </p>',
 '<p> <a href="#"> Back to top </a> </p>',
 '<p> <a href="#top"> Volver al comienzo </a> </p>']
z = [str(zz) for zz in zstr_top_p] # 4
{ string: z.count(string) for string in z_uniq }

# evaluate id attributes for the word top (ignore case); result: no concern
z_id = [tag for html in tqdm(pcd_art_html)
                    for tag in BeautifulSoup(html, 'lxml').\
                       find_all(id=True)]
z_id_names = sorted(set([x.name for x in tqdm(z_id)])) # 24
z_id_vals = [x['id'] for x in tqdm(z_id)] # 1733
z_id_vals_uniq = sorted(set(z_id_vals)) # 1733
z_id_val_freqs = { string: z_id_vals.count(string) for string in z_id_vals_uniq }
print(sorted(z_id_val_freqs.values(), reverse=True)[:25])
print({k: v for k, v in z_id_val_freqs.items() if v > 800})

```

```
z_id_tops = [z for z in tqdm(z_id) if z.get_text() is not None and \
             re.search(r'\btop\b', z.get_text(), re.I)]
z_id_top_vals = [x['id'] for x in tqdm(z_id_tops)] # 1733
z_id_top_vals_uniq = sorted(set(z_id_top_vals)) # 19
z_id_top_val_freqs = { string: z_id_top_vals.count(string) for string in
z_id_top_vals_uniq }
print(sorted(z_id_top_val_freqs.values(), reverse=True)[:25])
print(z_id_top_val_freqs)

z_id_top_texts = [x.get_text('|', strip=True) for x in tqdm(z_id_tops)] # 1733
z_id_top_texts_uniq = sorted(set(z_id_top_texts)) # 2852
z_id_top_text_freqs = { string: z_id_top_texts.count(string) for string in
z_id_top_texts_uniq }
print(sorted(z_id_top_text_freqs.values(), reverse=True)[:25])
print(z_id_top_text_freqs)
print({k: v for k, v in z_id_top_text_freqs.items() if v > 800})

# evaluate images for alt text; result: no concern
z_img = [tag for html in tqdm(pcd_art_html)
                    for tag in BeautifulSoup(html, 'lxml').\
                       find_all('img', alt=True)]
z_img_alts = [x['alt'] for x in tqdm(z_img)] #
z_img_alts_uniq = sorted(set(z_img_alts)) # 1249
z_img_alt_freqs = { string: z_img_alts.count(string) for string in z_img_alts_uniq }
print(sorted(z_img_alt_freqs.values(), reverse=True)[:25])
print({k: v for k, v in z_img_alt_freqs.items() if v > 300})



z = [tag for html in tqdm(pcd_art_html)
                    for tag in BeautifulSoup(html, 'lxml').find_all('a', href=a_hrefs)]
z_str = [str(zz) for zz in z]
z_string = [zz.get_text(strip=True) for zz in z]
{ elem: z_str.count(elem) for elem in sorted(set(z_str)) }
{ elem: z_string.count(elem) for elem in sorted(set(z_string)) }

z2 = [tag for html in tqdm(pcd_art_html)
                    for tag in BeautifulSoup(html, 'lxml').find_all('p',
                    class_='topOPage')]
z2_str = [str(zz) for zz in z2]
z2_string = [zz.get_text(strip=True) for zz in z2]
{ elem: z2_str.count(elem) for elem in sorted(set(z2_str)) }
{ elem: z2_string.count(elem) for elem in sorted(set(z2_string)) }

pcd_art_frame.loc[pcd_art_frame.mirror_path.str.contains('13_0137')]

def in_block_list(tag):
   cond1 = tag.name in ('h1', 'h2', 'h3', 'h4', 'h5', 'h6')
   cond2 = (tag.name == 'p') and \
      (tag.get_text(strip=True) in ('Back to top', 'Top', 'Top of Page'))
   return cond1 or cond2


[str(y) for y in x.find_all(in_block_list)]

def pcd_soup_body_blox(soup):
```

```
1939    # check for empty list?
1940    # block_names = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6']
1941    # block_elems += ['div', 'p', 'ul', 'ol']
1942    block_elems = soup.find_all(in_block_list)
1943 #    block_info = [{
1944 #       'name': el.name,
1945 # #      'attrs': el.attrs,
1946 #       'attrs': '|'.join(el.attrs.keys()),
1947 #       'text': el.get_text('|', strip=True)
1948 # #      'text_len': len(el.get_text(' ', strip=True))
1949 #       } for el in block_elems]
1950    block_info = [[el.name, '|'.join(el.attrs.keys()), el.get_text('|', strip=True)]
1951       for el in block_elems]
1952    return block_info
1953
1954 pcd_soup_body_blox(x)
1955
1956 pcd_body_blocks = [[path] + block
1957    for (path, html) in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_html), total=3237)
1958    for block in pcd_soup_body_blox(BeautifulSoup(html, 'lxml'))]
1959 # 3237/3237 [02:51<00:00, 18.83it/s]
1960 pd.DataFrame(pcd_body_blocks).to_excel('pcd_body_blocks.xlsx', engine='openpyxl')
1961
1962 # changed pcd_soup_body_blox between previous call and this one
1963 pcd_body_blocks2 = [[path] + block
1964    for (path, html) in tqdm(zip(pcd_art_frame.mirror_path, pcd_art_html), total=3237)
1965    for block in pcd_soup_body_blox(BeautifulSoup(html, 'lxml'))]
1966 # 3237/3237 [02:27<00:00, 21.91it/s]
1967 pd.DataFrame(pcd_body_blocks2).to_excel('pcd_body_blocks2.xlsx', engine='openpyxl')
1968
1969 sum([y[3] in ('Back to top', 'Top of Page', 'Top') for y in pcd_body_blocks2])
1970 # 15646 # 21010 when including  'Top of Page'
```