Clustering by Search Intent

Search intent is the meaning behind the search queries that users of Google type in when searching online. So you may have the following queries:

"Trench coats"
"Ladies trench coats"

"Life insurance"

"Trench coats" will share the same search intent as "Ladies trench coats" but won't share the same intent as "Life insurance." To work this out, a simple comparison of the top 10 ranking sites for both search phrases in Google will offer a strong suggestion of what Google thinks of the search intent between the two phrases.

It's not a perfect method, but it works well because you're using the ranking results which are a distillation of everything Google has learned to date on what content satisfies the search intent of the search query (based upon the trillions of global searches per year). Therefore, it's reasonable to surmise that if two search queries have similar enough SERPs, then the search intent is shared between keywords.

This is useful for a number of reasons:

- *Rank tracking costs*: If your budget is limited, then knowing the search intent means you can avoid incurring further expense by not tracking keywords with the same intent as those you're tracking. This comes with a risk as consumers change and the keyword not tracked may not share the same intent anymore.
- *Core updates*: With changing consumer search patterns come changing intents, which means you can see if keywords change clusters or not by comparing the search intent clusters of keywords before and after the update, which will help inform your response.
- *Keyword content mapping*: Knowing the intent means you can successfully map keywords to landing pages. This is especially useful in ensuring your site architecture consists of landing pages which map to user search demand.
- *Paid search ads*: Good keyword content mappings also mean you can improve the account structure and resulting quality score of your paid search activity.

Starting Point

Okay, time to cluster. We'll assume you already have the top 100 SERPs[3] results for each of your keywords stored as a Python dataframe "serps_input." The data is easily obtained from a rank tracking tool, especially if they have an API:

```
serps_input
```

This results in the following:

	keyword	rank	url	se_results_count
0	xbox one x controller	1	https://www.xbox.com/en-GB/accessories	144000000
1	xbox one x controller	2	None	144000000
2	xbox one x controller	3	https://www.xbox.com/en-GB/accessories/control...	144000000
3	xbox one x controller	4	https://www.argos.co.uk/browse/technology/vide...	144000000
4	xbox one x controller	5	https://www.game.co.uk/en/accessories/xbox-one...	144000000
5	xbox one x controller	6	https://www.currys.co.uk/gbuk/xbox-one-control...	144000000
6	xbox one x controller	7	https://www.amazon.co.uk/xbox-one-controller/s...	144000000
7	xbox one x controller	8	None	144000000
8	xbox one x controller	9	https://www.ebay.co.uk/b/Microsoft-Xbox-One-Co...	144000000
9	xbox one x controller	10	https://www.amazon.com/Xbox-Wireless-Controlle...	144000000
10	xbox one x controller	11	https://www.powera.com/product_platform/xbox-one/	144000000
11	xbox one x controller	12	https://www.pricerunner.com/sp/xbox-one-x-cont...	144000000
12	xbox one x controller	13	https://en.wikipedia.org/wiki/Xbox_Wireless_Co...	144000000
13	xbox one x controller	14	https://scufgaming.com/uk/xbox	144000000
14	xbox one x controller	15	https://www.digitaltrends.com/gaming/how-to-sy...	144000000

Here, we're using DataForSEO's SERP API,[4] and we have renamed the column from "rank_absolute" to "rank."

[3] Search Engine Results Pages (SERP)

[4] Available at https://dataforseo.com/apis/serp-api/

Filter Data for Page 1

Because DataForSEO's numbers to individual results are contained within carousels, People Also Ask, etc., we'll want to compare the top 20 results of each SERP to each other to get the approximate results for page 1. We'll also filter out URLs that have the value "None." The programming approach we'll take is "Split-Apply-Combine." What is Split-Apply-Combine?

- Split the dataframe into keyword groups
- Apply the filtering formula to each group
- Combine the keywords of each group

Here it goes:
Split:

```
serps_grpby_keyword = serps_input.groupby("keyword")
```

Apply the function, before combining:

```
def filter_twenty_urls(group_df):
    filtered_df = group_df.loc[group_df['url'].notnull()]
    filtered_df = filtered_df.loc[filtered_df['rank'] <= 20]
    return filtered_df
filtered_serps = serps_grpby_keyword.apply(filter_twenty_urls)
```

Combine and add prefix to column names:

```
normed = normed.add_prefix('normed_')
```

Concatenate with an initial dataframe:

```
filtered_serps_df = pd.concat([filtered_serps],axis=0)
```

Convert Ranking URLs to a String

To compare the SERPs for each keyword, we need to convert the SERPs URL into a string. That's because there's a one (keyword) to many (SERP URLs) relationship. The way we achieve that is by simply concatenating the URL strings for each keyword, using the Split-Apply-Combine approach (again). Convert results to strings using SAC:

```
filtserps_grpby_keyword = filtered_serps_df.groupby("keyword")

def string_serps(df):
    df['serp_string'] = ''.join(df['url'])
    return df

    Combine
strung_serps = filtserps_grpby_keyword.apply(string_serps)
```

Concatenate with an initial dataframe and clean:

```
strung_serps = pd.concat([strung_serps],axis=0)
strung_serps = strung_serps[['keyword', 'serp_string']]#.head(30)
strung_serps = strung_serps.drop_duplicates()
strung_serps
```

This results in the following:

	keyword	serp_string
0	fifa 19 ps4	https://www.amazon.co.uk/Electronic-Arts-22154...
18	gaming broadband	https://www.bt.com/products/broadband/gaminght...
37	playstation vr	https://www.playstation.com/en-gb/ps-vr/https:...
54	ps4	https://www.playstation.com/en-gb/ps4/https://...
72	ps4 console	https://www.game.co.uk/en/hardware/playstation...
91	ps4 controller	https://www.playstation.com/en-gb/ps4/ps4-acce...
109	ps4 controllers	https://www.playstation.com/en-gb/ps4/ps4-acce...
127	ps4 vr	https://www.playstation.com/en-gb/ps-vr/https:...
146	ps5	https://direct.playstation.com/en-us/ps5https:...
162	ps5 console	https://direct.playstation.com/en-us/ps5https:...

Now we have a table showing the keyword and their SERP string, we're ready to compare SERPs. Here's an example of the SERP string for "fifa 19 ps4":

```
strung_serps.loc[1, 'serp_string']
```

This results in the following:

```
'https://www.amazon.co.uk/Electronic-Arts-221545-FIFA-PS4/dp/
B07DLXBGN8https://www.amazon.co.uk/FIFA-19-GAMES/dp/B07DL2SY2Bhttps://
www.game.co.uk/en/fifa-19-2380636https://www.ebay.co.uk/b/FIFA-19-Sony-
PlayStation-4-Video-Games/139973/bn_711513427Ohttps://www.pricerunner.com/
pl/1422-4602670/PlayStation-4-Games/FIFA-19-Compare-Priceshttps://pricespy.
co.uk/games-consoles/computer-video-games/ps4/fifa-19-ps4--p4766432https://
store.playstation.com/en-gb/search/fifa%2019https://www.amazon.com/FIFA-19-
Standard-PlayStation-4/dp/B07DL2SY2Bhttps://www.tesco.com/groceries/
en-GB/products/301926084https://groceries.asda.com/product/ps-4-games/
ps-4-fifa-19/1000076097883https://uk.webuy.com/product-detail/?id=503094
5121916&categoryName=playstation4-software&superCatName=gaming&title=fi
fa-19https://www.pushsquare.com/reviews/ps4/fifa_19https://en.wikipedia.
org/wiki/FIFA_19https://www.amazon.in/Electronic-Arts-Fifa19SEPS4-Fifa-
PS4/dp/B07DVWWF44https://www.vgchartz.com/game/222165/fifa-19/https://www.
metacritic.com/game/playstation-4/fifa-19https://www.johnlewis.com/fifa-19-
ps4/p3755803https://www.ebay.com/p/22045274968'
```

Compare SERP Distance

The SERPs comparison will use string distance techniques which allow us to see how similar or dissimilar one keyword's SERPs are. This technique is similar to how geneticists would compare one DNA sequence to another.

Naturally, we need to get the SERPs into a format ready for Python to compare SERPs. To do this, we need to convert each SERP to a string and then put them side by side. Group the table by keyword:

```
filtserps_grpby_keyword = filtered_serps_df.groupby("keyword")
def string_serps(df):
    df['serp_string'] = ' '.join(df['url'])
    return df
```

Combine using the preceding function:

```
strung_serps = filtserps_grpby_keyword.apply(string_serps)
```

Concatenate with an initial dataframe and clean:

```
strung_serps = pd.concat([strung_serps],axis=0)
```

```
strung_serps = strung_serps[['keyword', 'serp_string']]#.head(30)
strung_serps = strung_serps.drop_duplicates()
#strung_serps['serp_string'] = strung_serps.serp_string.str.
replace("https://www\.", "")
strung_serps.head(15)
```

This results in the following:

	keyword	serp_string
0	beige trench coats	https://www.zalando.co.uk/womens-clothing-coat...
9	blue trench coats	https://www.johnlewis.com/browse/women/womens-...
19	buy ps4	https://www.playstation.com/en-gb/ps4/buy-ps4/...
24	ladies trench coats	https://www.johnlewis.com/browse/women/womens-...
34	mens trench coats	https://uk.burberry.com/mens-trench-coats/ htt...
43	ps4	https://www.playstation.com/en-gb/ps4/ https:/...
51	ps4 console	https://www.game.co.uk/en/hardware/playstation...
60	ps4 vr	https://www.playstation.com/en-gb/ps-vr/ https...
69	ps5	https://direct.playstation.com/en-us/ps5 https...
78	ps5 console	https://www.game.co.uk/playstation-5 https://d...
86	ps5 news	https://www.pushsquare.com/ps5 https://www.pla...
95	ps5 pre order	https://www.playstation.com/en-gb/ps5/buy-now/...
104	trench coats	https://uk.burberry.com/womens-trench-coats/ h...
112	xbox controller	https://www.xbox.com/en-GB/accessories/control...
120	xbox one	https://www.xbox.com/ https://www.xbox.com/en-...

Here, we now have the keywords and their respective SERPs all converted into a string which fits into a single cell. For example, the search result for "beige trench coats" is

```
'https://www.zalando.co.uk/womens-clothing-coats-trench-coats/_beige/
https://www.asos.com/women/coats-jackets/trench-coats/cat/?cid=15143
https://uk.burberry.com/womens-trench-coats/beige/ https://www2.hm.com/
```

```
en_gb/productpage.0751992002.html https://www.hobbs.com/clothing/
coats-jackets/trench/beige/ https://www.zara.com/uk/en/woman-outerwear-
trench-l1202.html https://www.ebay.co.uk/b/Beige-Trench-Coats-for-
Women/63862/bn_7028370345 https://www.johnlewis.com/browse/women/womens-
coats-jackets/trench-coats/_/N-flvZ1z0rnyl https://www.elle.com/uk/fashion/
what-to-wear/articles/g30975/best-trench-coats-beige-navy-black/'
```

Time to put these side by side. What we're effectively doing here is taking a product of the column to itself, that is, squaring it, so that we get all the SERPs combinations possible to put the SERPs side by side.

Add a function to align SERPs:

```
def serps_align(k, df):
    prime_df = df.loc[df.keyword == k]
    prime_df = prime_df.rename(columns = {"serp_string" : "serp_string_a",
    'keyword': 'keyword_a'})
    comp_df = df.loc[df.keyword != k].reset_index(drop=True)
    prime_df = prime_df.loc[prime_df.index.repeat(len(comp_df.index))].
    reset_index(drop=True)
    prime_df = pd.concat([prime_df, comp_df], axis=1)
    prime_df = prime_df.rename(columns = {"serp_string" : "serp_string_b",
    'keyword': 'keyword_b', "serp_string_a" : "serp_string", 'keyword_a':
    'keyword'})
    return prime_df
```

Test the function on a single keyword:

```
serps_align('ps4', strung_serps)
```

Set up desired dataframe columns:

```
columns = ['keyword', 'serp_string', 'keyword_b', 'serp_string_b']
matched_serps = pd.DataFrame(columns=columns)
matched_serps = matched_serps.fillna(0)
```

Call the function for each keyword:

```
for q in queries:
    temp_df = serps_align(q, strung_serps)
    matched_serps = matched_serps.append(temp_df)
```

This results in the following:

	keyword	serp_string	keyword_b	serp_string_b
0	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	beige trench coats	https://www.zalando.co.uk/womens-clothing-coat...
1	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	blue trench coats	https://www.johnlewis.com/browse/women/womens-...
2	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	buy ps4	https://www.playstation.com/en-gb/ps4/buy-ps4/...
3	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	ladies trench coats	https://www.johnlewis.com/browse/women/womens-...
4	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	mens trench coats	https://uk.burberry.com/mens-trench-coats/ htt...
...	...	...	...	...
267	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	trench coats	https://uk.burberry.com/womens-trench-coats/ h...
268	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox controller	https://www.xbox.com/en-GB/accessories/control...
269	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox one	https://www.xbox.com/ https://www.xbox.com/en-...
270	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox one controller	https://www.xbox.com/en-GB/accessories https:/...
271	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox series x	https://www.xbox.com/en-GB/consoles/xbox-serie...

The preceding result shows all of the keywords with SERPs compared side by side with other keywords and their SERPs. Next, we'll infer keyword intent similarity by comparing serp_strings, but first here's a note on the methods like Levenshtein, Jaccard, etc.

Levenshtein distance is edit based, meaning the number of edits required to transform one string (in our case, serp_string) into the other string (serps_string_b). This doesn't work very well because the websites within the SERP strings are individual tokens, that is, not a single continuous string.

Sorensen-Dice is better because it is token based, that is, it treats the individual websites as individual items or tokens. Using set similarity methods, the logic is to find the common tokens and divide them by the total number of tokens present by combining both sets. It doesn't take the order into account, so we must go one better.

M Measure which looks at both the token overlap and the order of the tokens, that is, weighting the order tokens earlier (i.e., the higher ranking sites/tokens) more than the later tokens. There is no API for this unfortunately, so we wrote the function for you here:

```
import py_stringmatching as sm
ws_tok = sm.WhitespaceTokenizer()
```

Only compare the top k_urls results:

```
def serps_similarity(serps_str1, serps_str2, k=15):
    denom = k+1
    norm = sum([2*(1/i - 1.0/(denom)) for i in range(1, denom)])
    #use to tokenize the URLs
```

```
    ws_tok = sm.WhitespaceTokenizer()
    #keep only first k URLs
    serps_1 = ws_tok.tokenize(serps_str1)[:k]
    serps_2 = ws_tok.tokenize(serps_str2)[:k]
    #get positions of matches
    match = lambda a, b: [b.index(x)+1 if x in b else None for x in a]
    #positions intersections of form [(pos_1, pos_2), ...]
    pos_intersections = [(i+1,j) for i,j in enumerate(match(serps_1,
    serps_2)) if j is not None]
    pos_in1_not_in2 = [i+1 for i,j in enumerate(match(serps_1, serps_2)) if
    j is None]
    pos_in2_not_in1 = [i+1 for i,j in enumerate(match(serps_2, serps_1)) if
    j is None]

    a_sum = sum([abs(1/i -1/j) for i,j in pos_intersections])
    b_sum = sum([abs(1/i -1/denom) for i in pos_in1_not_in2])
    c_sum = sum([abs(1/i -1/denom) for i in pos_in2_not_in1])

    intent_prime = a_sum + b_sum + c_sum
    intent_dist = 1 - (intent_prime/norm)
    return intent_dist
```

Apply the function:

```
matched_serps['si_simi'] = matched_serps.apply(lambda x: serps_
similarity(x.serp_string, x.serp_string_b), axis=1)
matched_serps[["keyword", "keyword_b", "si_simi"]]
```

This is the resulting dataframe:

	keyword	keyword_b	si_simi
0	ps4	beige trench coats	0.058203
1	ps4	blue trench coats	0.050328
2	ps4	buy ps4	0.314561
3	ps4	ladies trench coats	0.050328
4	ps4	mens trench coats	0.058203
...	...	...	...
267	blue trench coats	trench coats	0.096999
268	blue trench coats	xbox controller	0.050328
269	blue trench coats	xbox one	0.050328
270	blue trench coats	xbox one controller	0.040118
271	blue trench coats	xbox series x	0.063454

272 rows × 3 columns

Before sorting the keywords into topic groups, let's add search volumes for each. This could be an imported table like the following one called "keysv_df":

```
keysv_df
```

This results in the following:

	keyword	search_volume
0	best isa rates	40500
1	isa	49500
2	savings account	60500
3	cash isa	14800
4	isa account	9900
5	child savings account	14800
6	fixed rate bonds	12100
7	isa rates	8100
8	savings account interest rate	9900
9	fixed rate isa	5400
10	isa interest rates	5400
11	savings accounts uk	6600
12	cash isa rates	4400
13	easy access savings account	3600
14	savings rates	5400
15	easy access savings	3600
16	fixed rate savings	4400
17	isa savings	3600
18	kids savings account	4400
19	online savings account	2400

Let's now join the data. What we're doing here is giving Python the ability to group keywords according to SERP similarity and name the topic groups according to the keyword with the highest search volume.

Group keywords by search intent according to a similarity limit. In this case, keyword search results must be 40% or more similar. This is a number based on trial and error of which the right number can vary by the search space, language, or other factors.

```
simi_lim = 0.4
```

Append topic vols:

```
keywords_crossed_vols = serps_compared.merge(keysv_df, on = 'keyword', how
= 'left')
keywords_crossed_vols = keywords_crossed_vols.rename(columns = {'keyword':
'topic', 'keyword_b': 'keyword', 'search_volume': 'topic_volume'})
```

Append keyword vols:

```
keywords_crossed_vols = keywords_crossed_vols.merge(keysv_df, on =
'keyword', how = 'left')
```

Simulate si_simi:

```
#keywords_crossed_vols['si_simi'] = np.random.rand(len(keywords_crossed_
vols.index))
keywords_crossed_vols.sort_values('topic_volume', ascending = False)
```

Strip the dataframe of NAN:

```
keywords_filtered_nonnan = keywords_crossed_vols.dropna()
```

We now have the potential topic name, keyword SERP similarity, and search volumes of each. You'll note the keyword and keyword_b have been renamed to topic and keyword, respectively. Now we're going to iterate over the columns in the dataframe using list comprehensions.

List comprehension is a technique for looping over lists. We applied it to the Pandas dataframe because it's much quicker than the .iterrows() function. Here it goes.

Add a dictionary comprehension to create numbered topic groups from keywords_ filtered_nonnan:

```
# {1: [k1, k2, ..., kn], 2: [k1, k2, ..., kn], ..., n: [k1, k2, ..., kn]}
```

Convert the top names into a list:

```
queries_in_df = list(set(keywords_filtered_nonnan.topic.to_list()))
```

Set empty lists and dictionaries:

```
topic_groups_numbered = {}
topics_added = []
```

Define a function to find the topic number:

```
def latest_index(dicto):
    if topic_groups_numbered == {}:
        i = 0
    else:
        i = list(topic_groups_numbered)[-1]
    return i
```

Define a function to allocate keyword to topic:

```
def find_topics(si, keyw, topc):
    i = latest_index(topic_groups_numbered)
    if (si >= simi_lim) and (not keyw in topics_added) and (not topc in
    topics_added):
        #print(si, ', kw=' , keyw,', tpc=', topc,', ', i,', ', topic_
        groups_numbered)
        i += 1
        topics_added.extend([keyw, topc])
        topic_groups_numbered[i] = [keyw, topc]
    elif si >= simi_lim and (keyw in topics_added) and (not topc in
    topics_added):
        #print(si, ', kw=' , keyw,', tpc=', topc,', ', i,', ', topic_
        groups_numbered)
        j = [key for key, value in topic_groups_numbered.items() if keyw
        in value]
        topics_added.extend(topc)
        topic_groups_numbered[j[0]].append(topc)
    elif si >= simi_lim and (not keyw in topics_added) and (not topc in
    topics_added):
        #print(si, ', kw=' , keyw,', tpc=', topc,', ', i,', ', topic_
        groups_numbered)
        j = list(mydict.keys())[list(mydict.values()).index(keyw)]
        topic_groups_numbered[j[0]].append(topc)
```

The list comprehension will now apply the function to group keywords into clusters:

```
[find_topics(x, y, z) for x, y, z in zip(keywords_filtered_nonnan.si_simi,
keywords_filtered_nonnan.keyword, keywords_filtered_nonnan.topic)]
topic_groups_numbered
```

This results in the following:

```
{1: ['easy access savings',
  'savings account',
  'savings accounts uk',
  'savings rates',
  'online savings account',
  'online savings account',
  'online savings account'],
 2: ['isa account', 'isa', 'isa savings', 'isa savings'],
 3: ['kids savings account', 'child savings account'],
 4: ['best isa rates',
  'cash isa',
  'fixed rate isa',
  'fixed rate isa',
  'isa rates',
  'isa rates',
  'isa rates'],
 5: ['savings account interest rate',
  'savings accounts uk',
  'online savings account'],
 6: ['easy access savings account', 'savings rates', 'online savings
    account'],
 7: ['cash isa rates', 'fixed rate isa', 'isa rates'],
 8: ['isa interest rates', 'isa rates'],
 9: ['fixed rate savings', 'fixed rate bonds', 'online savings account']}
```

The preceding results are statements printing out what keywords are in which topic group. We do this to make sure we don't have duplicates or errors, which is crucial for the next step to perform properly. Now we're going to convert the dictionary into a dataframe so you can see all of your keywords grouped by search intent:

```
topic_groups_lst = []
for k, l in topic_groups_numbered.items():
    for v in l:
        topic_groups_lst.append([k, v])

topic_groups_dictdf = pd.DataFrame(topic_groups_lst, columns=['topic_group_
no', 'keyword'])
topic_groups_dictdf
```

This results in the following:

	topic_group_no	keyword
0	1	easy access savings
1	1	savings account
2	1	savings accounts uk
3	1	savings rates
4	1	online savings account
5	1	online savings account
6	1	online savings account
7	2	isa account
8	2	isa
9	2	isa savings
10	2	isa savings
11	3	kids savings account
12	3	child savings account
13	4	best isa rates
14	4	cash isa
15	4	fixed rate isa
16	4	fixed rate isa
17	4	isa rates

As you can see, the keywords are grouped intelligently, much like a human SEO analyst would group these, except these have been done at scale using the wisdom of Google which is distilled from its vast number of users. Name the clusters:

```
topic_groups_vols = topic_groups_dictdf.merge(keysv_df, on = 'keyword', how
= 'left')

def highest_demand(df):
```

```
    df = df.sort_values('search_volume', ascending = False)
    del df['topic_group_no']
    max_sv = df.search_volume.max()
    df = df.loc[df.search_volume == max_sv]
    return df

topic_groups_vols_keywgrp = topic_groups_vols.groupby('topic_group_no')
topic_groups_vols_keywgrp.get_group(1)
```

Apply and combine:

```
high_demand_topics = topic_groups_vols_keywgrp.apply(highest_demand).
reset_index()
del high_demand_topics['level_1']
high_demand_topics = high_demand_topics.rename(columns = {'keyword':
'topic'})

def shortest_name(df):
    df['k_len'] = df.topic.str.len()
    min_kl = df.k_len.min()
    df = df.loc[df.k_len == min_kl]
    del df['topic_group_no']
    del df['k_len']
    del df['search_volume']
    return df

high_demand_topics_spl = high_demand_topics.groupby('topic_group_no')
```

Apply and combine:

```
named_topics = high_demand_topics_spl.apply(shortest_name).reset_index()
del named_topics['level_1']
```

Name topic numbered keywords:

```
topic_keyw_map = pd.merge(named_topics, topic_groups_dictdf, on = 'topic_
group_no', how = 'left')
topic_keyw_map
```

The resulting table shows that we now have keywords clustered by topic:

	topic_group_no	topic	keyword
0	1	savings account	savings accounts uk
1	1	savings account	savings account
2	1	savings account	savings account interest rate
3	1	savings account	easy access savings
4	1	savings account	savings rates
5	1	savings account	fixed rate savings
6	1	savings account	fixed rate bonds
7	1	savings account	online savings account
8	1	savings account	easy access savings account
9	2	isa	isa
10	2	isa	isa account
11	2	isa	isa savings
12	3	child savings account	kids savings account
13	3	child savings account	child savings account
14	4	best isa rates	cash isa
15	4	best isa rates	best isa rates

Let's add keyword search volumes:

```
topic_keyw_vol_map = pd.merge(topic_keyw_map, keysv_df, on = 'keyword', how
= 'left')
topic_keyw_vol_map
```

This results in the following:

	topic_group_no	topic	keyword	search_volume
0	1	savings account	savings accounts uk	6600
1	1	savings account	savings account	60500
2	1	savings account	savings account interest rate	9900
3	1	savings account	easy access savings	3600
4	1	savings account	savings rates	5400
5	1	savings account	fixed rate savings	4400
6	1	savings account	fixed rate bonds	12100
7	1	savings account	online savings account	2400
8	1	savings account	easy access savings account	3600
9	2	isa	isa	49500
10	2	isa	isa account	9900
11	2	isa	isa savings	3600
12	3	child savings account	kids savings account	4400
13	3	child savings account	child savings account	14800
14	4	best isa rates	cash isa	14800
15	4	best isa rates	best isa rates	40500

This is really starting to take shape, and you can quickly see opportunities emerging.

SERP Competitor Titles

If you don't have much Google Search Console data or Google Ads data to mine, then you may need to resort to your competitors. You may or may not want to use third-party keyword research tools such as SEMRush. And you don't have to.

Tools like SEMRush, Keyword.io, etc., certainly have a place in the SEO industry. In the absence of any other data, they are a decent ready source of intelligence on what search queries generate relevant traffic.

However, some work will need to be done in order to weed out the noise and extract high value phrases – assuming a competitive market. Otherwise, if your website (or