In [83]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.cm as cm
from collections import defaultdict

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import random
```

In [79]:

```
lemmatizer = WordNetLemmatizer()

def lemma(word):
    word = word.strip()

    safe = ["ns"]
    if word in safe:
        return word

    noun = lemmatizer.lemmatize(word)
    if noun != word:
        return noun

    adjective = lemmatizer.lemmatize(word, pos="a")
    if adjective != word:
        return adjective

    verb = lemmatizer.lemmatize(word, pos="v")
    if verb != word:
        return verb

    return word

stopwordDict = defaultdict(int)
for sw in stopwords.words("english"):
    stopwordDict[sw] += 1
```

In [ ]:

```
import pymongo

client = pymongo.MongoClient(host="128.195.180.83",
                             port=27939,
                             username="db_viewer",
                             password="ucidsplab_dbviewer"
                             )
db = client.cloudflare_crawled_data

vocab = {}

corpus, corpus_index, url_list = [], [], []
startPage, endPage = 100, 1354

for i in range(startPage, endPage + 1):
    col_name = "purepage" + str(i)
    collection = db[col_name]

    for page in collection.find():
        if not page["DNS_Related"]:
            continue

        if "Other Languages" in page["labels"]:
            continue

#         date = page["created_date"].split(",")
#         year = date[1][1:5]
#         if year != "2021" and year != "2020":
#             continue

        unclassified = True
        if unclassified:

            unprocessed, processed = page["original_post"], []
            for word in unprocessed.split(" "):
                lem = lemma(word.strip().lower())

                try:
                    _ = int(lem)
                    continue
                except:
                    pass

                if lem not in stopwordDict:
                    flag = False
                    for char in lem:
                        if char < "a" or char > "z":
                            flag = True
                    if not flag:
                        processed.append(lem)
                        vocab[lem] = 1


            corpus.append(" ".join(processed))
            corpus_index.append(page["title"])
            url_list.append(page["url"])


print(corpus)
print(len(corpus))
```

In [22]:

```
from sklearn.feature_extraction.text import CountVectorizer
vocabList = vocab.keys()
#cv1 = CountVectorizer(vocabulary=vocabList, analyzer = 'word', ngram_range=(1,2))
cv1 = CountVectorizer(vocabulary=vocabList, analyzer = 'word')
corpus_vocab_count_matrix = cv1.transform(corpus)
```

In [23]:

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer(smooth_idf=True, use_idf=True)
tfidf_transformer.fit(corpus_vocab_count_matrix)
```

Out[23]:

```
TfidfTransformer()
```

In [24]:

```
df_idf = pd.DataFrame(tfidf_transformer.idf_, index=cv1.get_feature_names(), columns=
df_idf = df_idf.sort_values(by=['idf-weights'])
print(df_idf.head(10))
```

```
            idf-weights
cloudflare     1.428240
dns            1.610688
```

```
domain        1.782482
use           2.051258
com           2.068365
record        2.211908
get           2.227520
thank         2.256599
work          2.262401
server        2.291928
```

In [25]:

```
articles_vocab_count_matrix = cv1.transform(corpus)
tfidf_matrix = tfidf_transformer.transform(articles_vocab_count_matrix)
```

In [26]:

```
tfidf_array = np.asarray(tfidf_matrix.todense())
df = pd.DataFrame(tfidf_array[0], index=cv1.get_feature_names(), columns=['tfidf'])
df_descending = df.sort_values(by=['tfidf'], ascending=False)
print(df_descending.head(10))
```

```
              tfidf
cornell    0.455808
edu        0.312971
query      0.274808
confluence 0.265888
opt        0.252202
contegix   0.237834
flag       0.218305
answer     0.196958
est        0.149258
dec        0.145079
```

In [60]:

```
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine
distance_array = pairwise_distances(tfidf_array, metric='cosine')
```

In [59]:

```
distance_array
```

Out[59]:

```
array([[-27.19565918,   0.76632044,   0.5809683 , ...,  -0.87197672,
          0.31689633,   0.57934432],
       [  0.18277863, -21.75420146,   0.03796774, ...,  -2.41995114,
         -0.95927878,   0.37833943],
       [  0.0375315 ,   0.09069644, -21.2904778 , ...,  -0.51003695,
         -1.3721348 ,   0.57934432],
       ...,
       [ -1.24882979,  -1.85603471,  -0.12230912, ..., -23.73957429,
          0.75989932,   0.57934432],
       [ -0.29794515,  -0.60088823,  -0.94421591, ...,   0.32749841,
        -28.73846948,   0.57934432],
       [  0.61148409,   0.93072739,   1.04191954, ...,   0.76756499,
          1.29927718, -23.84607878]])
```

In [41]:

```
#distance_array = StandardScaler().fit_transform(distance_array)
```

In [76]:

```
import numpy as np

from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

clustering = DBSCAN(eps=0.6, min_samples=2).fit(distance_array)
```

In [72]:

```
for i in range(max(clustering.labels_)):
    print(np.where(clustering.labels_==i)[0])
```

```
[  120  1219  1776  3116  4977  6466  6701  7503  7580  8313  8721  8864
 10101]
[ 2039  9813 11947]
[2729 5499 6883]
```

In [73]:

```
core_samples_mask = np.zeros_like(clustering.labels_, dtype=bool)
core_samples_mask[clustering.core_sample_indices_] = True
labels = clustering.labels_
labels
```

Out[73]:

```
array([-1, -1, -1, ..., -1, -1, -1], dtype=int64)
```

In [74]:

```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
```

In [47]:

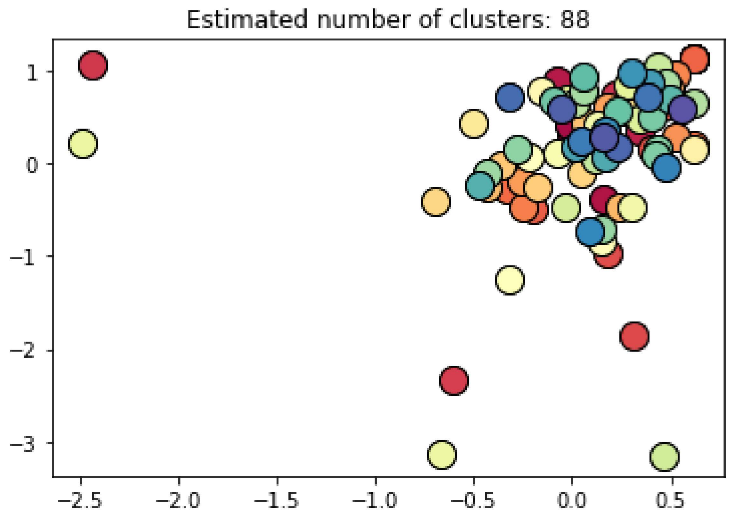
```
import matplotlib.pyplot as plt
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        continue
        col = [0, 0, 0, 1]

    class_member_mask = labels == k

    xy = distance_array[class_member_mask & core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )

    xy = distance_array[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title("Estimated number of clusters: %d" % n_clusters_)
plt.show()
```



In [48]:

```
metrics.silhouette_score(distance_array, labels)
```

Out[48]:

```
-0.20606649528416796
```

In [ ]:

```

```