> Note: After running a number of these silhouettes, we had a very hard time finding a balance between running computationally expensive code and successfully clustering. Therefore, we ended up going with faiss.Kmeans() instead of purusing sklearn further.

The below unsupervised learning sentiment analysis takes in a corpus of 514027 rows of Tweets and YouTube comments and finds word, bigram, and sentence associations. Using the associations of the tokens within the corpus, it vectorizes the distances of the associations. The overall process begins by utilizing cleaning techniques such as lemmatization, removal of NaN values and stop words. Secondly by using the Gensim library's Phrases() and Phraser() to convert the individual words into bigrams and clusters of no more than 5 words (note to self--might need to adjust the minimum number of words in a phrase) and vectorizes their associations.

The biggest leap here is that since this is unsupervised learning, none of the rows or words already have a predetermined classification. There is no already defined positive, negative, or neutral words or statement. So labeling and tagging the words by first creating associations between them is the first major step before judging the sentiment value (i.e. positive versus negative versus neutral).

In [6]:
```
import re  # For preprocessing
import pandas as pd  # For data handling
from time import time  # To time our operations
from collections import defaultdict  # For word frequency

import spacy  # For preprocessing

import logging  # Setting up the loggings to monitor gensim
logging.basicConfig(format="%(levelname)s - %(asctime)s: %(message)s", d
atefmt= '%H:%M:%S', level=logging.INFO)
```

In [8]:
```
pd.set_option('display.width', None)
pd.set_option('max_columns', None)
pd.set_option('max_colwidth', 200)
```

In [7]:
```
df = pd.read_csv('lemm.csv')
df.shape
```

```
/Users/tlipman/opt/anaconda3/envs/learn-env/lib/python3.6/site-package
s/IPython/core/interactiveshell.py:2714: DtypeWarning: Columns (3) have
mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[7]:
```
(514027, 7)
```

In [9]:
```
df.head()
```

Out[9]:

	Unnamed: 0	text	favorite_count	user_id	mentions	repost_count	
0	0	deployment 60 starlink satellite confirmed	97534.0	34743251.0	NaN	9272.0	1.3674
1	1	sn11 almost ready fly	60997.0	44196397.0	NaN	4389.0	1.3719
2	2	ive continued driving scout spot ill drop mar helicopter area get certified fli	56739.0	1.23278323762312e+18	NaN	5605.0	1.3690
3	3	honestly hadnt seen eye didnt footage would 100 think cg	40107.0	3167257102.0	NaN	5219.0	1.3719
4	4	really doe look like something 80 sci fi show credit spacex	36838.0	9.294728238480302e+17	NaN	2216.0	1.3679

In [10]:
```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
```

In [11]:
```
df.head()
```

Out[11]:

	text	favorite_count	user_id	mentions	repost_count	post_id
0	deployment 60 starlink satellite confirmed	97534.0	34743251.0	NaN	9272.0	1.367407e+18
1	sn11 almost ready fly	60997.0	44196397.0	NaN	4389.0	1.371995e+18
2	ive continued driving scout spot ill drop mar helicopter area get certified fli	56739.0	1.23278323762312e+18	NaN	5605.0	1.369068e+18
3	honestly hadnt seen eye didnt footage would 100 think cg	40107.0	3167257102.0	NaN	5219.0	1.371988e+18
4	really doe look like something 80 sci fi show credit spacex	36838.0	9.294728238480302e+17	NaN	2216.0	1.367993e+18

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
text                  81
favorite_count         0
user_id                3
mentions          181446
repost_count           0
post_id                1
dtype: int64
```

In [15]:
```
df_comments = df.drop(['favorite_count', 'user_id', 'mentions', 'repost_
count'], axis=1)
```

In [18]:
```
df_comments = df_comments.dropna().reset_index(drop=True)
df_comments.isnull().sum()
```

Out[18]:
```
text       0
post_id    0
dtype: int64
```

In [19]:
```
df_comments.head()
```

Out[19]:

	text	post_id
0	deployment 60 starlink satellite confirmed	1.367407e+18
1	sn11 almost ready fly	1.371995e+18
2	ive continued driving scout spot ill drop mar helicopter area get certified fli	1.369068e+18
3	honestly hadnt seen eye didnt footage would 100 think cg	1.371988e+18
4	really doe look like something 80 sci fi show credit spacex	1.367993e+18

In [22]:
```
nlp = spacy.load('en_core_web_sm', disable=['ner', 'parser']) # disablin
g Named Entity Recognition for speed

def cleaning(doc):
    # Lemmatizes and removes stopwords
    # doc needs to be a spacy Doc object
    txt = [token.lemma_ for token in doc if not token.is_stop]
    # Word2Vec uses context words to learn the vector representation of
 a target word,
    # if a sentence is only one or two words long,
    # the benefit for the training is very small
    if len(txt) > 2:
        return ' '.join(txt)
```

In [25]:
```
# remove non-alphabetical characters in 'text' column

brief_cleaning = (re.sub("[^A-Za-z']+", ' ', str(row)).lower() for row i
n df_comments['text'])
```

In [27]:
```
# Process texts as a stream, and yield `Doc` objects in order.

t = time()

txt = [cleaning(doc) for doc in nlp.pipe(brief_cleaning)]

print('Time to clean up everything: {} mins'.format(round((time() - t) /
60, 2)))
```

```
Time to clean up everything: 8.75 mins
```

In [28]:
```
df_clean = pd.DataFrame({'clean': txt})
df_clean = df_clean.dropna().drop_duplicates()
df_clean.shape
```

Out[28]:
```
(500924, 1)
```

In [29]:
```
df_clean.head()
```

Out[29]:

	clean
0	deployment starlink satellite confirm
1	sn ready fly
2	ve continue drive scout spot ill drop mar helicopter area certify fli
3	honestly nt see eye nt footage think cg
4	doe look like sci fi credit spacex

In [30]:
```
# Utilizing Gensim Phrases package to automatically detect common phrase
s (bigrams)
# from a list of sentences. https://radimrehurek.com/gensim/models/phras
es.html

from gensim.models.phrases import Phrases, Phraser
```

```
/Users/tlipman/opt/anaconda3/envs/learn-env/lib/python3.6/site-package
s/gensim/similarities/__init__.py:15: UserWarning: The gensim.similarit
ies.levenshtein submodule is disabled, because the optional Levenshtein
package <https://pypi.org/project/python-Levenshtein/> is unavailable.
Install Levenhstein (e.g. `pip install python-Levenshtein`) to suppress
this warning.
  warnings.warn(msg)
```

In [31]:
```
sent = [row.split() for row in df_clean['clean']]
```

In [32]:

```
# Detect phrases based on collocation counts.
phrases = Phrases(sent)
```

```
INFO - 06:56:31: collecting all words and their counts
INFO - 06:56:31: PROGRESS: at sentence #0, processed 0 words and 0 word
types
INFO - 06:56:32: PROGRESS: at sentence #10000, processed 100114 words a
nd 87342 word types
INFO - 06:56:32: PROGRESS: at sentence #20000, processed 198794 words a
nd 162683 word types
INFO - 06:56:32: PROGRESS: at sentence #30000, processed 297712 words a
nd 234295 word types
INFO - 06:56:32: PROGRESS: at sentence #40000, processed 395270 words a
nd 301893 word types
INFO - 06:56:32: PROGRESS: at sentence #50000, processed 492039 words a
nd 365707 word types
INFO - 06:56:32: PROGRESS: at sentence #60000, processed 588500 words a
nd 427226 word types
INFO - 06:56:33: PROGRESS: at sentence #70000, processed 683840 words a
nd 486520 word types
INFO - 06:56:33: PROGRESS: at sentence #80000, processed 778614 words a
nd 544369 word types
INFO - 06:56:33: PROGRESS: at sentence #90000, processed 871368 words a
nd 601094 word types
INFO - 06:56:33: PROGRESS: at sentence #100000, processed 963971 words
and 656715 word types
INFO - 06:56:33: PROGRESS: at sentence #110000, processed 1055932 words
and 710347 word types
INFO - 06:56:34: PROGRESS: at sentence #120000, processed 1145787 words
and 761448 word types
INFO - 06:56:34: PROGRESS: at sentence #130000, processed 1233804 words
and 809946 word types
INFO - 06:56:34: PROGRESS: at sentence #140000, processed 1320930 words
and 857300 word types
INFO - 06:56:34: PROGRESS: at sentence #150000, processed 1407901 words
and 904914 word types
INFO - 06:56:34: PROGRESS: at sentence #160000, processed 1494247 words
and 953396 word types
INFO - 06:56:34: PROGRESS: at sentence #170000, processed 1580180 words
and 1001455 word types
INFO - 06:56:35: PROGRESS: at sentence #180000, processed 1666651 words
and 1049221 word types
INFO - 06:56:35: PROGRESS: at sentence #190000, processed 1753861 words
and 1095365 word types
INFO - 06:56:35: PROGRESS: at sentence #200000, processed 1840973 words
and 1140831 word types
INFO - 06:56:35: PROGRESS: at sentence #210000, processed 1929337 words
and 1187140 word types
INFO - 06:56:35: PROGRESS: at sentence #220000, processed 2022876 words
and 1232496 word types
INFO - 06:56:35: PROGRESS: at sentence #230000, processed 2117724 words
and 1275305 word types
INFO - 06:56:36: PROGRESS: at sentence #240000, processed 2217516 words
and 1314984 word types
INFO - 06:56:36: PROGRESS: at sentence #250000, processed 2311978 words
and 1353013 word types
INFO - 06:56:36: PROGRESS: at sentence #260000, processed 2412904 words
and 1389296 word types
INFO - 06:56:37: PROGRESS: at sentence #270000, processed 2510409 words
and 1428175 word types
```

```
INFO - 06:56:37: PROGRESS: at sentence #280000, processed 2611381 words
and 1468633 word types
INFO - 06:56:37: PROGRESS: at sentence #290000, processed 2715933 words
and 1509817 word types
INFO - 06:56:37: PROGRESS: at sentence #300000, processed 2817663 words
and 1547801 word types
INFO - 06:56:37: PROGRESS: at sentence #310000, processed 2915474 words
and 1588508 word types
INFO - 06:56:38: PROGRESS: at sentence #320000, processed 3012327 words
and 1633348 word types
INFO - 06:56:38: PROGRESS: at sentence #330000, processed 3109904 words
and 1671660 word types
INFO - 06:56:38: PROGRESS: at sentence #340000, processed 3208825 words
and 1711522 word types
INFO - 06:56:38: PROGRESS: at sentence #350000, processed 3314596 words
and 1754506 word types
INFO - 06:56:38: PROGRESS: at sentence #360000, processed 3412319 words
and 1793626 word types
INFO - 06:56:38: PROGRESS: at sentence #370000, processed 3512462 words
and 1832530 word types
INFO - 06:56:39: PROGRESS: at sentence #380000, processed 3614138 words
and 1868337 word types
INFO - 06:56:39: PROGRESS: at sentence #390000, processed 3709086 words
and 1909353 word types
INFO - 06:56:39: PROGRESS: at sentence #400000, processed 3809764 words
and 1950103 word types
INFO - 06:56:39: PROGRESS: at sentence #410000, processed 3903895 words
and 1990292 word types
INFO - 06:56:39: PROGRESS: at sentence #420000, processed 3998432 words
and 2031016 word types
INFO - 06:56:40: PROGRESS: at sentence #430000, processed 4097164 words
and 2075753 word types
INFO - 06:56:40: PROGRESS: at sentence #440000, processed 4195227 words
and 2114695 word types
INFO - 06:56:40: PROGRESS: at sentence #450000, processed 4299809 words
and 2149637 word types
INFO - 06:56:40: PROGRESS: at sentence #460000, processed 4400418 words
and 2184346 word types
INFO - 06:56:40: PROGRESS: at sentence #470000, processed 4496443 words
and 2223460 word types
INFO - 06:56:41: PROGRESS: at sentence #480000, processed 4595097 words
and 2266443 word types
INFO - 06:56:41: PROGRESS: at sentence #490000, processed 4693712 words
and 2313718 word types
INFO - 06:56:41: PROGRESS: at sentence #500000, processed 4790923 words
and 2357515 word types
INFO - 06:56:41: collected 2361650 token types (unigram + bigrams) from
a corpus of 4799749 words and 500924 sentences
INFO - 06:56:41: merged Phrases<2361650 vocab, min_count=5, threshold=1
0.0, max_vocab_size=40000000>
INFO - 06:56:41: Phrases lifecycle event {'msg': 'built Phrases<2361650
vocab, min_count=5, threshold=10.0, max_vocab_size=40000000> in 9.62s',
'datetime': '2021-04-07T06:56:41.566997', 'gensim': '4.0.1', 'python':
'3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17) \n[GCC 4.2.1 C
ompatible Clang 4.0.1 (tags/RELEASE_401/final)]', 'platform': 'Darwin-1
9.6.0-x86_64-i386-64bit', 'event': 'created'}
```

In [33]:
```
# The goal of Phraser() is to cut down memory consumption of Phrases(),

bigram = Phraser(phrases)
```

```
INFO - 06:58:47: exporting phrases from Phrases<2361650 vocab, min_coun
t=5, threshold=10.0, max_vocab_size=40000000>
INFO - 06:58:53: FrozenPhrases lifecycle event {'msg': 'exported Frozen
Phrases<24851 phrases, min_count=5, threshold=10.0> from Phrases<236165
0 vocab, min_count=5, threshold=10.0, max_vocab_size=40000000> in 6.10
s', 'datetime': '2021-04-07T06:58:53.684451', 'gensim': '4.0.1', 'pytho
n': '3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17) \n[GCC 4.
2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]', 'platform': 'Dar
win-19.6.0-x86_64-i386-64bit', 'event': 'created'}
```

In [34]:
```
# transform the corpus based upon bigrams detected
sentences = bigram[sent]
```

In [35]:
```
# creating a word frequency count for each individual word
# ensuring that lemmatization, removal of stop words, and bigrams reduce
d
# the total diversity of sentiment to be able to be more accurately meas
ured and understood

word_freq = defaultdict(int)
for sent in sentences:
    for i in sent:
        word_freq[i] += 1
len(word_freq)
```

Out[35]:
```
369182
```

In [36]:
```
sorted(word_freq, key=word_freq.get, reverse=True)[:10]
```

Out[36]:
```
['space', 'mar', 'nasa', 'http_co', 'nt', 'spacex', 'wa', 'm', 'like',
'ha']
```

The latter approach would be an unsupervised one, and this one is an object of interest in this article. The main idea behind unsupervised learning is that you don’t give any previous assumptions and definitions to the model about the outcome of variables you feed into it — you simply insert the data (of course preprocessed before), and want the model to learn the structure of the data itself. It is extremely useful in cases when you don’t have labeled data, or you are not sure about the structure of the data, and you want to learn more about the nature of process you are analyzing, without making any previous assumptions about its outcome.

Gensim Word2Vec Implementation: Model Training

In [37]:
```
import multiprocessing

from gensim.models import Word2Vec
```

In [38]:
```
vanilla_model = Word2Vec()
```

```
INFO - 08:58:59: Word2Vec lifecycle event {'params': 'Word2Vec(vocab=0,
vector_size=100, alpha=0.025)', 'datetime': '2021-04-08T08:58:59.69087
2', 'gensim': '4.0.1', 'python': '3.6.9 |Anaconda, Inc.| (default, Jul
30 2019, 13:42:17) \n[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_40
1/final)]', 'platform': 'Darwin-19.6.0-x86_64-i386-64bit', 'event': 'cr
eated'}
```

Word2Vec requires us to build a vocabulary table:

- taking in all the words
- filtering out the unique words
- conducting a word count):

In [40]:

```
t = time()

vanilla_model.build_vocab(sentences)

print('Time to build vocab: {} mins'.format(round((time() - t) / 60, 2
)))
```

```
INFO - 09:01:36: collecting all words and their counts
INFO - 09:01:36: PROGRESS: at sentence #0, processed 0 words, keeping 0
word types
INFO - 09:01:36: PROGRESS: at sentence #10000, processed 87487 words, k
eeping 21107 word types
INFO - 09:01:37: PROGRESS: at sentence #20000, processed 175523 words,
keeping 34764 word types
INFO - 09:01:37: PROGRESS: at sentence #30000, processed 264009 words,
keeping 46776 word types
INFO - 09:01:37: PROGRESS: at sentence #40000, processed 351216 words,
keeping 57989 word types
INFO - 09:01:37: PROGRESS: at sentence #50000, processed 437368 words,
keeping 68267 word types
INFO - 09:01:37: PROGRESS: at sentence #60000, processed 523355 words,
keeping 77963 word types
INFO - 09:01:38: PROGRESS: at sentence #70000, processed 607879 words,
keeping 87322 word types
INFO - 09:01:38: PROGRESS: at sentence #80000, processed 692052 words,
keeping 96364 word types
INFO - 09:01:38: PROGRESS: at sentence #90000, processed 775477 words,
keeping 105552 word types
INFO - 09:01:38: PROGRESS: at sentence #100000, processed 858560 words,
keeping 114122 word types
INFO - 09:01:39: PROGRESS: at sentence #110000, processed 940554 words,
keeping 122451 word types
INFO - 09:01:39: PROGRESS: at sentence #120000, processed 1020619 word
s, keeping 130415 word types
INFO - 09:01:39: PROGRESS: at sentence #130000, processed 1098885 word
s, keeping 137969 word types
INFO - 09:01:39: PROGRESS: at sentence #140000, processed 1176645 word
s, keeping 145363 word types
INFO - 09:01:39: PROGRESS: at sentence #150000, processed 1254848 word
s, keeping 152569 word types
INFO - 09:01:39: PROGRESS: at sentence #160000, processed 1332806 word
s, keeping 160144 word types
INFO - 09:01:40: PROGRESS: at sentence #170000, processed 1410229 word
s, keeping 167551 word types
INFO - 09:01:40: PROGRESS: at sentence #180000, processed 1488518 word
s, keeping 174753 word types
INFO - 09:01:40: PROGRESS: at sentence #190000, processed 1566898 word
s, keeping 181964 word types
INFO - 09:01:40: PROGRESS: at sentence #200000, processed 1646005 word
s, keeping 189220 word types
INFO - 09:01:40: PROGRESS: at sentence #210000, processed 1726646 word
s, keeping 196622 word types
INFO - 09:01:41: PROGRESS: at sentence #220000, processed 1810677 word
s, keeping 203394 word types
INFO - 09:01:41: PROGRESS: at sentence #230000, processed 1894382 word
s, keeping 209180 word types
INFO - 09:01:41: PROGRESS: at sentence #240000, processed 1981120 word
s, keeping 214516 word types
INFO - 09:01:41: PROGRESS: at sentence #250000, processed 2064354 word
s, keeping 220035 word types
INFO - 09:01:42: PROGRESS: at sentence #260000, processed 2151238 word
s, keeping 225089 word types
INFO - 09:01:42: PROGRESS: at sentence #270000, processed 2235857 word
s, keeping 231195 word types
```

```
INFO - 09:01:42: PROGRESS: at sentence #280000, processed 2324160 word
s, keeping 236708 word types
INFO - 09:01:42: PROGRESS: at sentence #290000, processed 2415613 word
s, keeping 242638 word types
INFO - 09:01:42: PROGRESS: at sentence #300000, processed 2504551 word
s, keeping 248121 word types
INFO - 09:01:43: PROGRESS: at sentence #310000, processed 2591885 word
s, keeping 254259 word types
INFO - 09:01:43: PROGRESS: at sentence #320000, processed 2678735 word
s, keeping 261570 word types
INFO - 09:01:43: PROGRESS: at sentence #330000, processed 2765529 word
s, keeping 268019 word types
INFO - 09:01:43: PROGRESS: at sentence #340000, processed 2853585 word
s, keeping 273935 word types
INFO - 09:01:43: PROGRESS: at sentence #350000, processed 2947059 word
s, keeping 280065 word types
INFO - 09:01:44: PROGRESS: at sentence #360000, processed 3033078 word
s, keeping 285472 word types
INFO - 09:01:44: PROGRESS: at sentence #370000, processed 3120848 word
s, keeping 290622 word types
INFO - 09:01:44: PROGRESS: at sentence #380000, processed 3209082 word
s, keeping 295331 word types
INFO - 09:01:44: PROGRESS: at sentence #390000, processed 3293916 word
s, keeping 301214 word types
INFO - 09:01:45: PROGRESS: at sentence #400000, processed 3383697 word
s, keeping 307429 word types
INFO - 09:01:45: PROGRESS: at sentence #410000, processed 3468709 word
s, keeping 313114 word types
INFO - 09:01:45: PROGRESS: at sentence #420000, processed 3554684 word
s, keeping 319605 word types
INFO - 09:01:45: PROGRESS: at sentence #430000, processed 3644055 word
s, keeping 326832 word types
INFO - 09:01:45: PROGRESS: at sentence #440000, processed 3731484 word
s, keeping 332596 word types
INFO - 09:01:46: PROGRESS: at sentence #450000, processed 3821979 word
s, keeping 337450 word types
INFO - 09:01:46: PROGRESS: at sentence #460000, processed 3909128 word
s, keeping 342444 word types
INFO - 09:01:46: PROGRESS: at sentence #470000, processed 3993098 word
s, keeping 348418 word types
INFO - 09:01:46: PROGRESS: at sentence #480000, processed 4081596 word
s, keeping 354901 word types
INFO - 09:01:46: PROGRESS: at sentence #490000, processed 4172530 word
s, keeping 362245 word types
INFO - 09:01:47: PROGRESS: at sentence #500000, processed 4261457 word
s, keeping 368543 word types
INFO - 09:01:47: collected 369182 word types from a corpus of 4269583 r
aw words and 500924 sentences
INFO - 09:01:47: Creating a fresh vocabulary
INFO - 09:01:47: Word2Vec lifecycle event {'msg': 'effective_min_count=
5 retains 63580 unique words (17.221858053751266%% of original 369182,
drops 305602)', 'datetime': '2021-04-08T09:01:47.576368', 'gensim': '4.
0.1', 'python': '3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:1
7) \n[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]', 'pla
tform': 'Darwin-19.6.0-x86_64-i386-64bit', 'event': 'prepare_vocab'}
INFO - 09:01:47: Word2Vec lifecycle event {'msg': 'effective_min_count=
5 leaves 3824181 word corpus (89.568021045615%% of original 4269583, dr
```

```
ops 445402)', 'datetime': '2021-04-08T09:01:47.577522', 'gensim': '4.0.
1', 'python': '3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17)
\n[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]', 'platfo
rm': 'Darwin-19.6.0-x86_64-i386-64bit', 'event': 'prepare_vocab'}
INFO - 09:01:48: deleting the raw counts dictionary of 369182 items
INFO - 09:01:48: sample=0.001 downsamples 30 most-common words
INFO - 09:01:48: Word2Vec lifecycle event {'msg': 'downsampling leaves
estimated 3578747.1810065126 word corpus (93.6%% of prior 3824181)', 'd
atetime': '2021-04-08T09:01:48.108706', 'gensim': '4.0.1', 'python':
'3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17) \n[GCC 4.2.1 C
ompatible Clang 4.0.1 (tags/RELEASE_401/final)]', 'platform': 'Darwin-1
9.6.0-x86_64-i386-64bit', 'event': 'prepare_vocab'}
INFO - 09:01:48: estimated required memory for 63580 words and 100 dime
nsions: 82654000 bytes
INFO - 09:01:48: resetting layer weights
INFO - 09:01:49: Word2Vec lifecycle event {'update': False, 'trim_rul
e': 'None', 'datetime': '2021-04-08T09:01:49.090883', 'gensim': '4.0.
1', 'python': '3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 13:42:17)
\n[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]', 'platfo
rm': 'Darwin-19.6.0-x86_64-i386-64bit', 'event': 'build_vocab'}

Time to build vocab: 0.21 mins
```

Clustering

In [43]:

```
--NotebookApp.iopub_data_rate_limit=1.0e10
```

```
  File "<ipython-input-43-25d083b58976>", line 1
    --NotebookApp.iopub_data_rate_limit=1.0e10
                                              ^
SyntaxError: can't assign to operator
```

In [41]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# establish the of text documents
text = df_comments['text']
# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)
# summarize
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])
# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [47]:
```
# add column to dataframe
for text in df_comments['text']:
    text = vector.toarray()
```

In [48]:
```
df_comments.head()
```

Out[48]:

	text	post_id
0	deployment 60 starlink satellite confirmed	1.367407e+18
1	sn11 almost ready fly	1.371995e+18
2	ive continued driving scout spot ill drop mar helicopter area get certified fli	1.369068e+18
3	honestly hadnt seen eye didnt footage would 100 think cg	1.371988e+18
4	really doe look like something 80 sci fi show credit spacex	1.367993e+18

NOTES: If the text was correctly vectorized using TFIDF, wouldn't the column values 'text' be displayed as vectors?

In [52]:
```
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
```

In [50]:
```
x = df_comments['text']
y = vectorizer.fit_transform(x)
```

In [51]:
```
# displaying the multidimentionality of the dataset
y.shape
```

Out[51]:
```
(513945, 382688)
```

In [53]:
```
x_train, x_test, y_train, y_test = train_test_split(y, x, test_size=0.2,
train_size=0.8, random_state=0)
```

In [56]:
```
clusters = KMeans(n_clusters=3,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)

y_km = clusters.fit_predict(x_train)
clusters.inertia_
```

Out[56]:
```
407058.2041046221
```

Visualizing the clusters

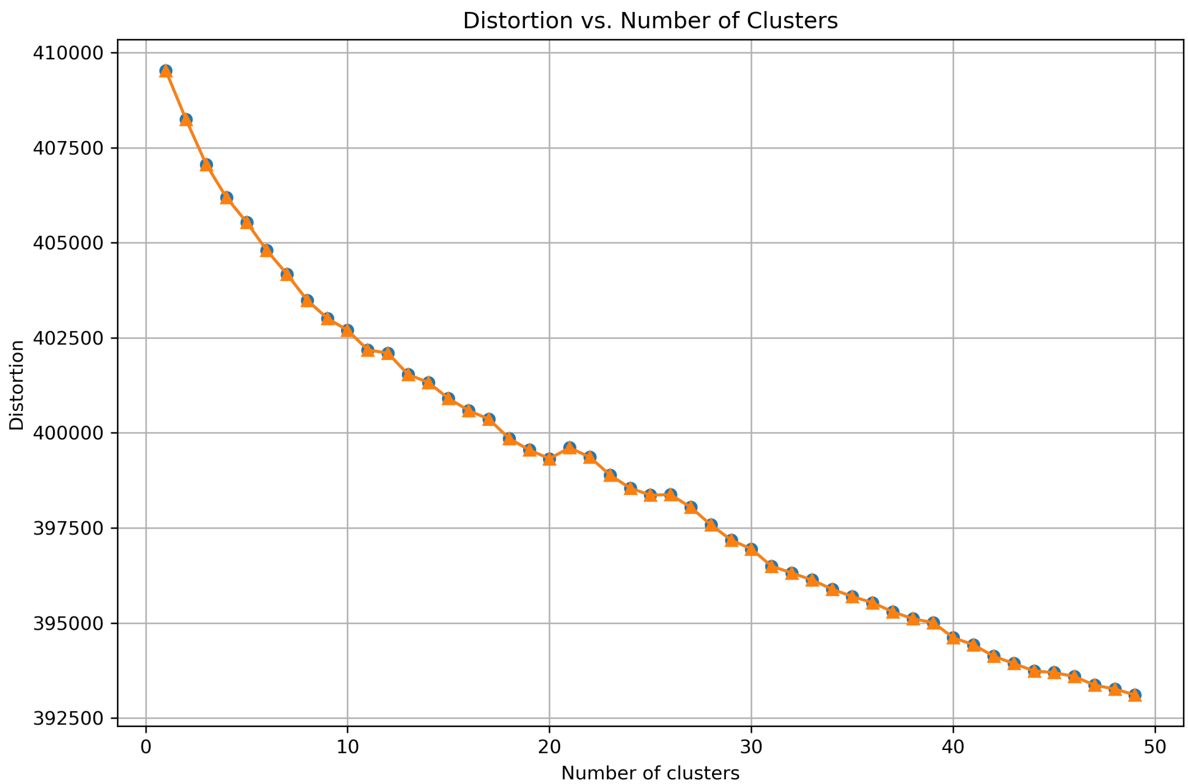
Elbow Method | Quantifying Distortion

In [61]:
```
%matplotlib inline
from IPython.display import Image
import matplotlib.pyplot as plt
```

In [59]:
```
distortions = []
ScoreList   = []
maxNumberOfClusters=50

for i in range(1, maxNumberOfClusters):
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=0)
    km.fit(x_train)
    distortions.append(km.inertia_)
    ScoreList.append(-km.score(x_train))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-59-ea1cf397e938> in <module>()
     14 
     15 
---> 16 plt.plot(range(1, maxNumberOfClusters), distortions, marker='o')
     17 plt.plot(range(1, maxNumberOfClusters), ScoreList, marker='^')
     18 plt.xlabel('Number of clusters')

AttributeError: module 'matplotlib' has no attribute 'plot'
```

In [66]:
```
plt.figure(figsize=(9, 6), dpi=300)
plt.plot(range(1, maxNumberOfClusters), distortions, marker='o')
plt.plot(range(1, maxNumberOfClusters), ScoreList, marker='^')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.title('Distortion vs. Number of Clusters')
plt.tight_layout()
plt.grid(True)
#plt.savefig('images/11_03.png', dpi=300)
plt.show()
```



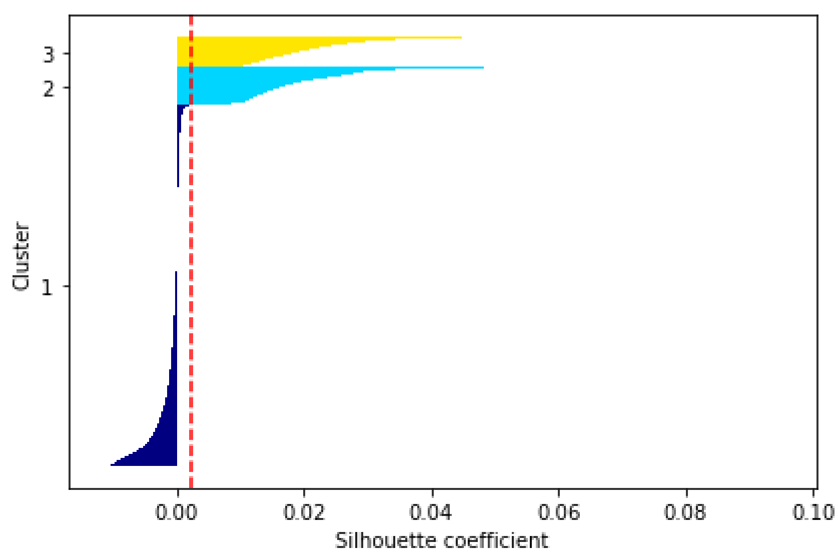
In [60]:
```
list(distortions)
```

Out[60]:
```
[409523.43945019826,
 408239.9419649572,
 407058.2041046221,
 406189.3036780986,
 405541.65203667124,
 404798.3639729706,
 404174.17582473025,
 403487.9384772645,
 403012.4649344421,
 402699.65013360966,
 402181.56058032165,
 402096.13828038273,
 401534.6971363215,
 401325.7474762759,
 400905.4600150742,
 400587.9970817581,
 400360.06836105976,
 399853.9422180477,
 399549.75253139937,
 399322.2658109628,
 399609.85515683383,
 399360.9206275439,
 398891.77300744696,
 398545.73672280693,
 398363.38362102455,
 398374.952972023,
 398042.1835232826,
 397578.4563243436,
 397181.694931716,
 396946.3024974996,
 396494.3448806715,
 396318.15194258007,
 396130.5684008789,
 395890.8713717635,
 395696.80283337564,
 395534.7654026135,
 395295.6575261435,
 395110.15984973044,
 395013.067259124,
 394612.9919097628,
 394428.3409459194,
 394126.1383374245,
 393934.38422859524,
 393738.703281618,
 393696.99965549103,
 393591.8818625619,
 393365.6189636897,
 393263.9352830617,
 393108.47383202077]
```

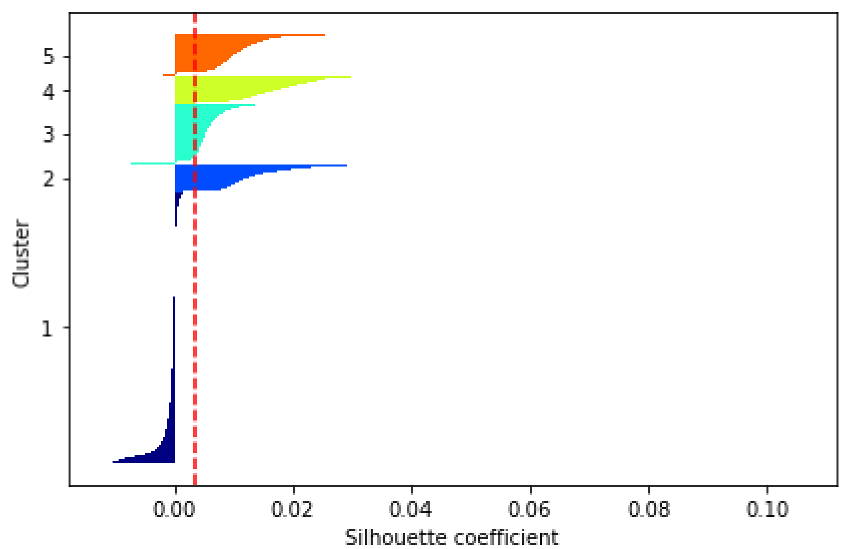
Silhouette Plot | evaluating the distance the clusters are from each other and their respective densities

In [67]:
```
from sklearn.metrics import silhouette_samples
from matplotlib import cm
import numpy as np
```

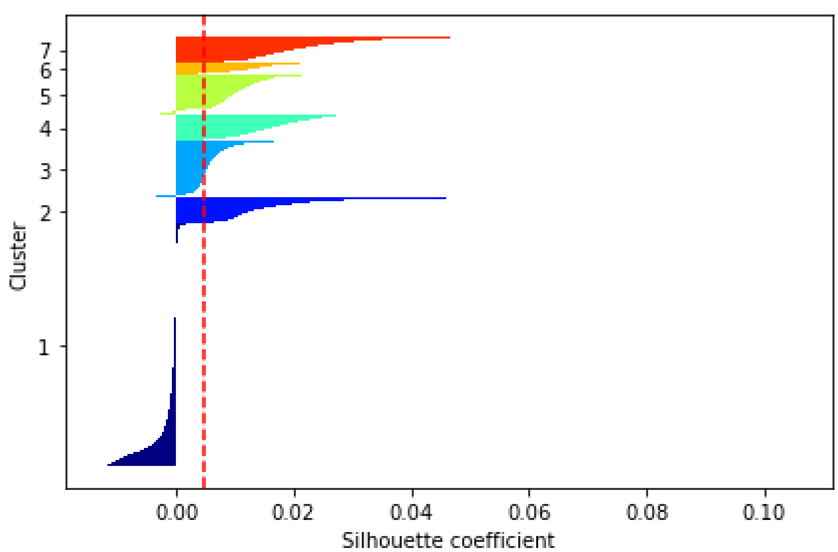
In [76]:
```
km3 = KMeans(n_clusters=3,
             init='k-means++',
             n_init=10,
             max_iter=300,
             tol=1e-04,
             random_state=0)
y_km3 = km3.fit_predict(x_train)

cluster_labels = np.unique(y_km3)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km3, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km3 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
              edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```



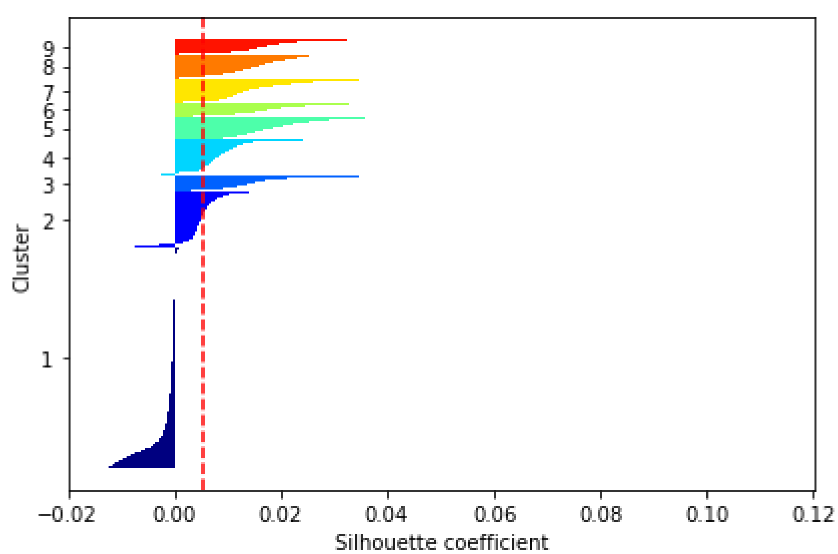
In [68]:
```
km5 = KMeans(n_clusters=5,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km5 = km5.fit_predict(x_train)

cluster_labels = np.unique(y_km5)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km5, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km5 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```



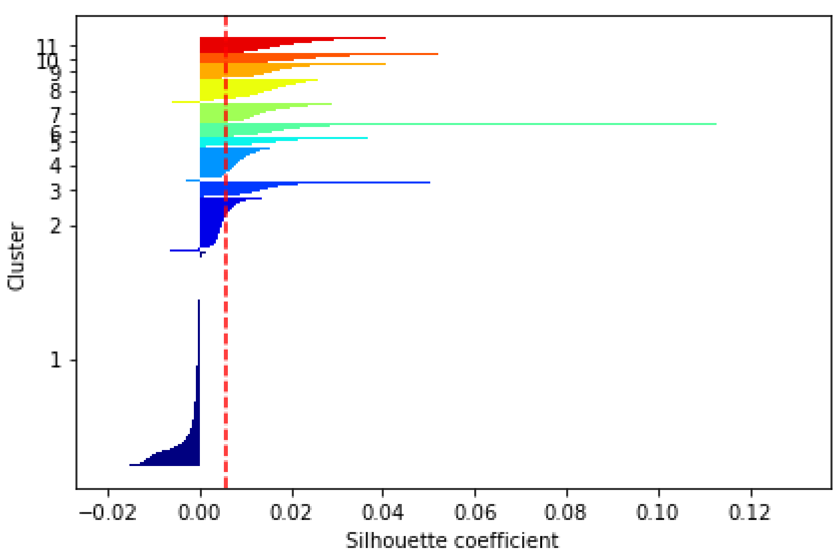
```
In [69]: km7 = KMeans(n_clusters=7,
                     init='k-means++',
                     n_init=10,
                     max_iter=300,
                     tol=1e-04,
                     random_state=0)
         y_km7 = km7.fit_predict(x_train)

         cluster_labels = np.unique(y_km7)
         n_clusters = cluster_labels.shape[0]
         silhouette_vals = silhouette_samples(x_train, y_km7, metric='euclidean')
         y_ax_lower, y_ax_upper = 0, 0
         yticks = []
         for i, c in enumerate(cluster_labels):
             c_silhouette_vals = silhouette_vals[y_km7 == c]
             c_silhouette_vals.sort()
             y_ax_upper += len(c_silhouette_vals)
             color = cm.jet(float(i) / n_clusters)
             plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
         0,
                       edgecolor='none', color=color)

             yticks.append((y_ax_lower + y_ax_upper) / 2.)
             y_ax_lower += len(c_silhouette_vals)

         silhouette_avg = np.mean(silhouette_vals)
         plt.axvline(silhouette_avg, color="red", linestyle="--")

         plt.yticks(yticks, cluster_labels + 1)
         plt.ylabel('Cluster')
         plt.xlabel('Silhouette coefficient')

         plt.tight_layout()
         #plt.savefig('images/11_04.png', dpi=300)
         plt.show()
```



In [70]:
```
km9 = KMeans(n_clusters=9,
             init='k-means++',
             n_init=10,
             max_iter=300,
             tol=1e-04,
             random_state=0)
y_km9 = km9.fit_predict(x_train)

cluster_labels = np.unique(y_km9)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km9, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km9 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```

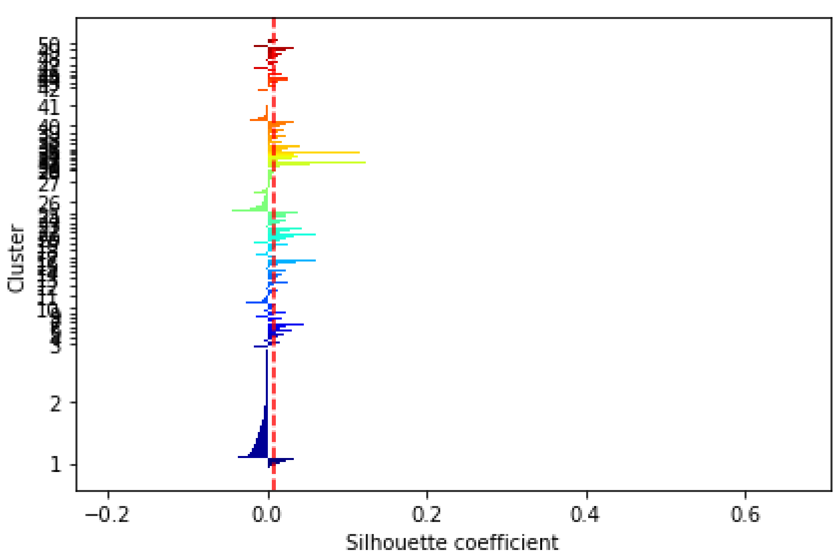


In [71]:
```
km11 = KMeans(n_clusters=11,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km11 = km11.fit_predict(x_train)

cluster_labels = np.unique(y_km11)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km11, metric='euclidean'
)
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km11 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```

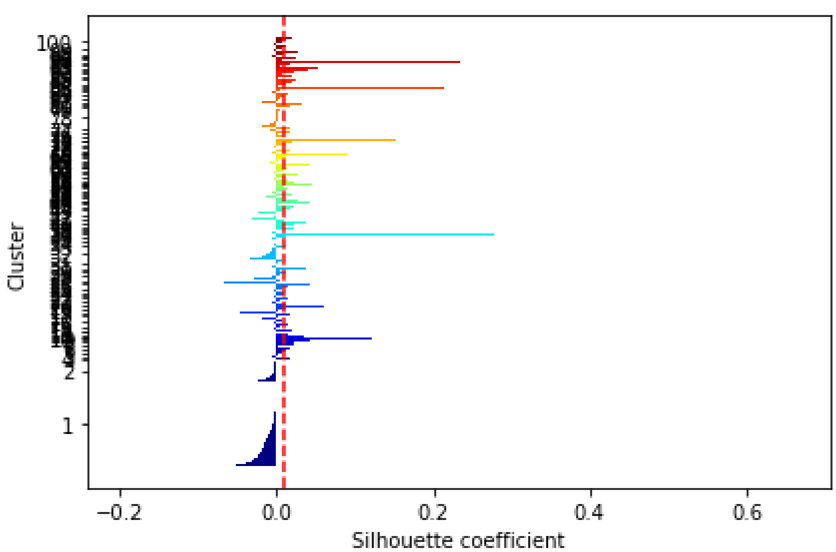


In [73]:

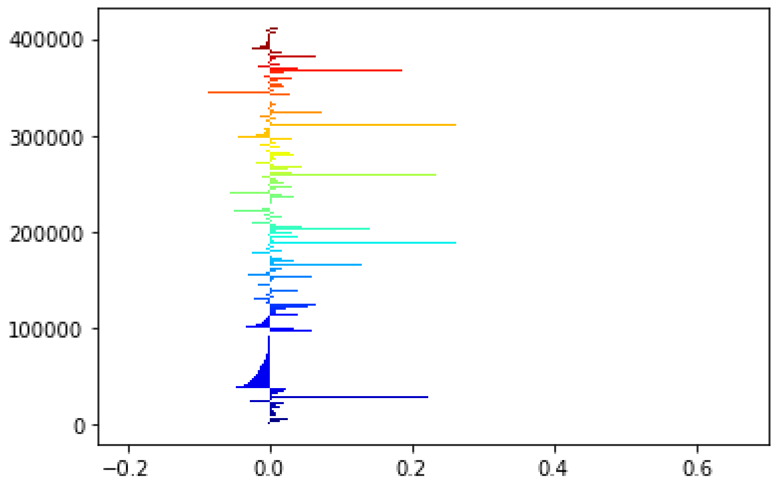
```
km50 = KMeans(n_clusters=50,
              init='k-means++',
              n_init=10,
              max_iter=300,
              tol=1e-04,
              random_state=0)
y_km50 = km50.fit_predict(x_train)

cluster_labels = np.unique(y_km50)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km50, metric='euclidean'
)
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km50 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```

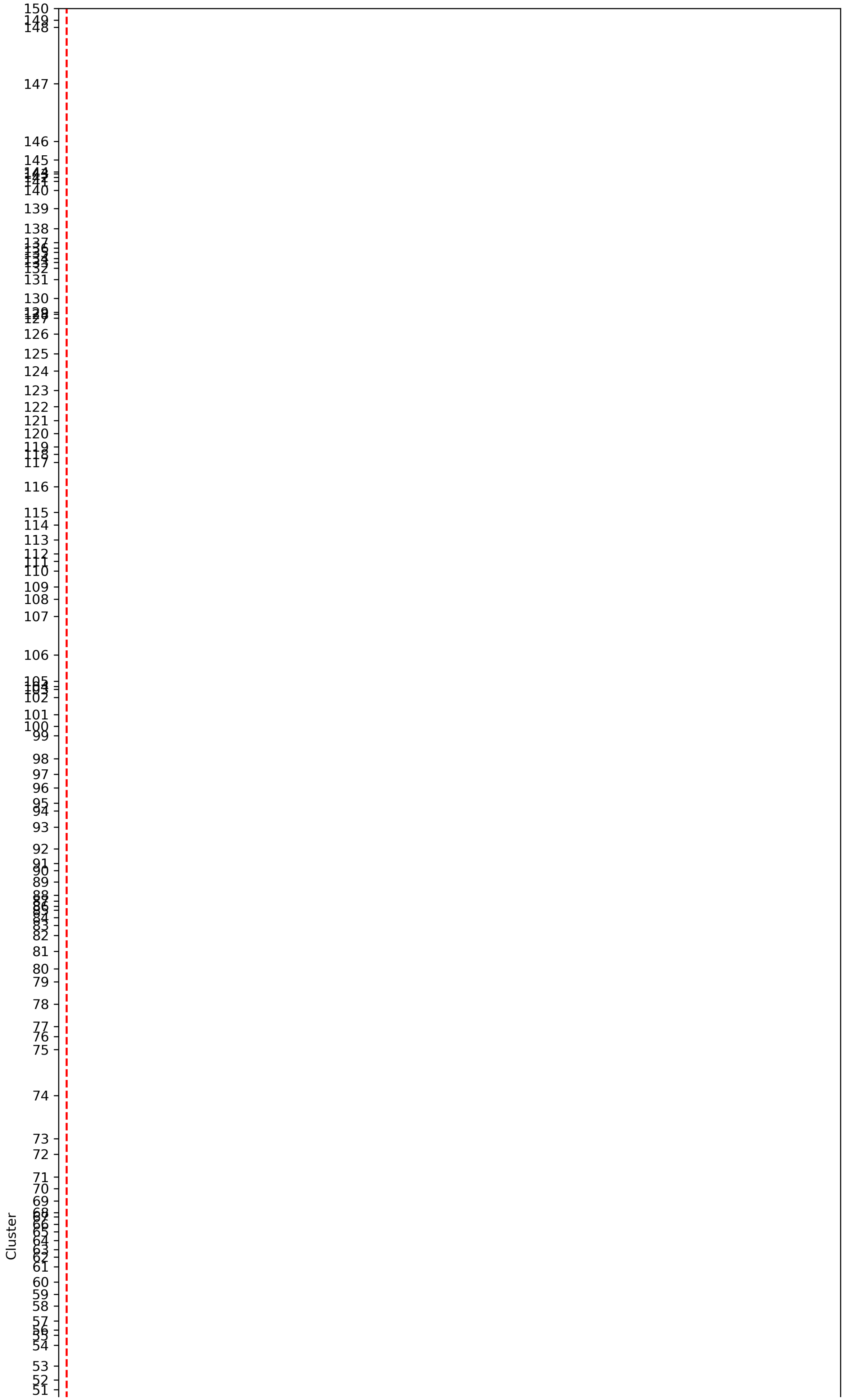


In [74]:
```
km100 = KMeans(n_clusters=100,
               init='k-means++',
               n_init=10,
               max_iter=300,
               tol=1e-04,
               random_state=0)
y_km100 = km100.fit_predict(x_train)

cluster_labels = np.unique(y_km100)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km100, metric='euclidea
n')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km100 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```

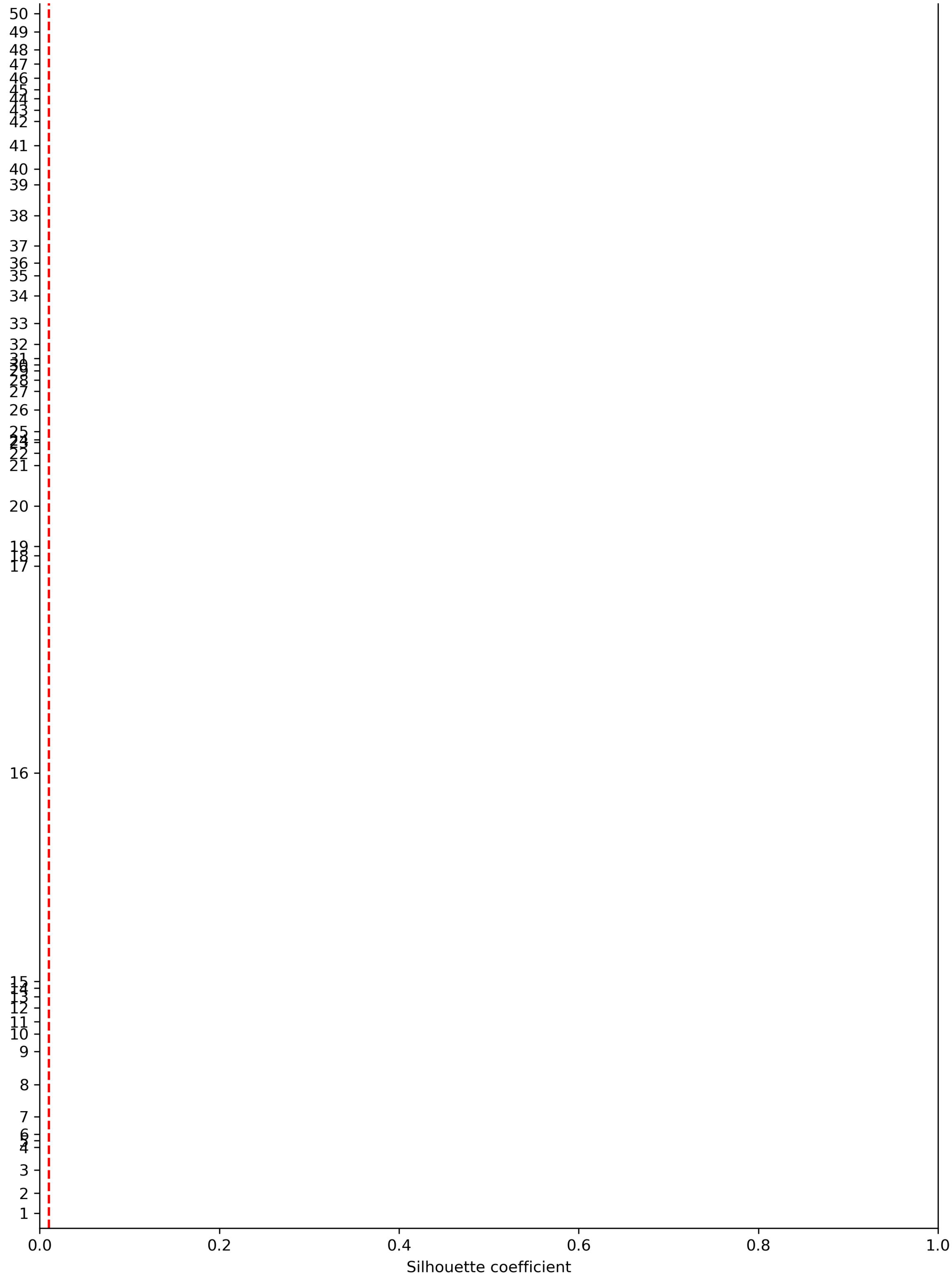


In [75]:
```
km150 = KMeans(n_clusters=150,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km150 = km150.fit_predict(x_train)

cluster_labels = np.unique(y_km150)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(x_train, y_km150, metric='euclidea
n')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km150 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.
0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```


400000
300000
200000
100000
0
−0.2
0.0
0.2
0.4
0.6



Cluster



Notes: Based upon the silhouette scores, the first cluster appears to contain a large number of outliers or unrelated information, which is giving rise to the negative silhouette score. Let's further inspect each cluster using a series of word plots to analyze what words are existing within the clusters.

Add k-means predicted clusters in a column to the dataframe

In [ ]:
```
km = KMeans(n_clusters=5, random_state=0)
km.fit(x_train)
predict=km.predict(y_train)
```