In [1]:
```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
```

In [2]:
```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!'
]

labels = ['weather', 'weather', 'animals', 'animals', 'weather', 'animals']
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

Out[2]:

	Document	Category
0	The sky is blue and beautiful.	weather
1	Love this blue and beautiful sky!	weather
2	The quick brown fox jumps over the lazy dog.	animals
3	The brown fox is quick and the blue dog is lazy!	animals
4	The sky is very blue and the sky is very beaut...	weather
5	The dog is lazy but the brown fox is quick!	animals

In [3]:
```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

In [4]:
```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

Out[4]:
```
array(['sky blue beautiful', 'love blue beautiful sky',
       'quick brown fox jumps lazy dog', 'brown fox quick blue dog lazy',
       'sky blue sky beautiful today', 'dog lazy brown fox quick'],
      dtype='<U30')
```

In [5]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[5]:
```
array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1],
       [0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]], dtype=int64)
```

In [6]:
```
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

Out[6]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	1	0	1	0
2	0	0	1	1	1	1	1	0	1	0	0
3	0	1	1	1	1	0	1	0	1	0	0
4	1	1	0	0	0	0	0	0	0	2	1
5	0	0	1	1	1	0	1	0	1	0	0

In [7]:
```
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

Out[7]:

	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	brown fox	dog lazy	fox jumps	fox quick	jumps lazy	lazy brown	lazy dog	lov blu
0	0	0	1	0	0	0	0	0	0	0	0	0	
1	1	0	1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	1	0	1	0	1	0	1	
3	0	0	0	1	0	1	1	0	1	0	0	0	
4	0	1	0	0	1	0	0	0	0	0	0	0	
5	0	0	0	0	0	1	1	0	1	0	1	0	

In [8]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

Out[8]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	0.60	0.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00
1	0.46	0.39	0.00	0.00	0.00	0.00	0.00	0.66	0.00	0.46	0.00
2	0.00	0.00	0.38	0.38	0.38	0.54	0.38	0.00	0.38	0.00	0.00
3	0.00	0.36	0.42	0.42	0.42	0.00	0.42	0.00	0.42	0.00	0.00
4	0.36	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.52
5	0.00	0.00	0.45	0.45	0.45	0.00	0.45	0.00	0.45	0.00	0.00

In [9]:
```
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix = cosine_similarity(tv_matrix)
similarity_df = pd.DataFrame(similarity_matrix)
similarity_df
```

Out[9]:

	0	1	2	3	4	5
0	1.000000	0.753128	0.000000	0.185447	0.807539	0.000000
1	0.753128	1.000000	0.000000	0.139665	0.608181	0.000000
2	0.000000	0.000000	1.000000	0.784362	0.000000	0.839987
3	0.185447	0.139665	0.784362	1.000000	0.109653	0.933779
4	0.807539	0.608181	0.000000	0.109653	1.000000	0.000000
5	0.000000	0.000000	0.839987	0.933779	0.000000	1.000000

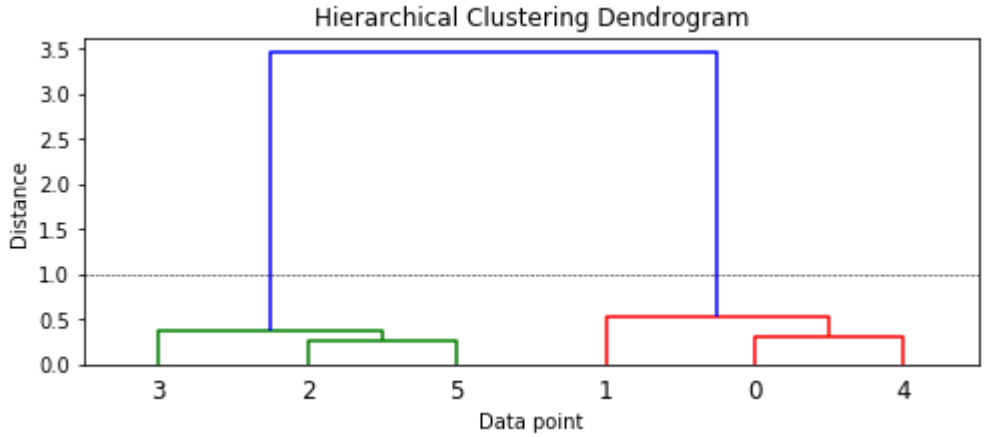
In [10]:
```
from scipy.cluster.hierarchy import dendrogram, linkage

Z = linkage(similarity_matrix, 'ward')
pd.DataFrame(Z, columns=['Document\Cluster 1', 'Document\Cluster 2',
                         'Distance', 'Cluster Size'], dtype='object')
```

Out[10]:

	Document\Cluster 1	Document\Cluster 2	Distance	Cluster Size
0	2	5	0.271171	2
1	0	4	0.317548	2
2	3	6	0.373037	3
3	1	7	0.531801	3
4	8	9	3.44916	6

In [11]:
```
plt.figure(figsize=(8, 3))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point')
plt.ylabel('Distance')
dendrogram(Z)
plt.axhline(y=1.0, c='k', ls='--', lw=0.5)
```

Out[11]: <matplotlib.lines.Line2D at 0x29e996604a8>



In [12]:
```
from sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(n_topics=3, max_iter=10000, random_state=0)
dt_matrix = lda.fit_transform(cv_matrix)
features = pd.DataFrame(dt_matrix, columns=['T1', 'T2', 'T3'])
features
```

```
C:\Users\finan\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\de
composition\online_lda.py:314: DeprecationWarning: n_topics has been renamed
to n_components in version 0.19 and will be removed in 0.21
  DeprecationWarning)
```

Out[12]:

	T1	T2	T3
0	0.831570	0.084281	0.084149
1	0.864945	0.067312	0.067743
2	0.047801	0.903651	0.048548
3	0.055404	0.896033	0.048563
4	0.887660	0.055993	0.056347
5	0.055710	0.887959	0.056331

In [13]:
```
tt_matrix = lda.components_
for topic_weights in tt_matrix:
    topic = [(token, weight) for token, weight in zip(vocab, topic_weights)]
    topic = sorted(topic, key=lambda x: -x[1])
    topic = [item for item in topic if item[1] > 0.6]
    print(topic)
    print()
```

```
[('sky', 4.330354318739757), ('blue', 3.3755171944376308), ('beautiful', 3.33
0118419346211), ('today', 1.330700279870604), ('love', 1.329975060395897)]

[('brown', 3.3302367122958025), ('dog', 3.3302367122958025), ('fox', 3.330236
7122958025), ('lazy', 3.3302367122958025), ('quick', 3.3302367122958025), ('j
umps', 1.3302792634397713), ('blue', 1.2856996815109725)]

[]
```

In [14]:
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, random_state=0)
km.fit_transform(features)
cluster_labels = km.labels_
cluster_labels = pd.DataFrame(cluster_labels, columns=['ClusterLabel'])
pd.concat([corpus_df, cluster_labels], axis=1)
```

Out[14]:

	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	2
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	The brown fox is quick and the blue dog is lazy!	animals	1
4	The sky is very blue and the sky is very beaut...	weather	0
5	The dog is lazy but the brown fox is quick!	animals	1

In [ ]:

In [ ]: