1. Plotting a Dendogram

In [1]:
```
import pandas as pd

# Read in the CSV file
data = pd.read_csv('dailykos.csv')

# Print the first few rows of the data to verify everything looks good
print(data.head())
```

```
   abandon  abc  ability  abortion  absolute  abstain  abu  abuse  accept  \
0        0    0        0         0         0        0    0      0       0
1        0    0        0         0         0        0    0      0       0
2        0    0        0         0         0        0    1      0       0
3        0    0        0         0         0        0    0      0       0
4        0    0        0         0         0        0    0      0       0

   access  ...  yeah  year  yesterday  york  youll  young  youre  youve  \
0       0  ...     0     0          0     0      0      0      0      0
1       0  ...     0     0          0     0      0      0      0      0
2       0  ...     0     0          0     0      0      0      0      0
3       0  ...     0     0          2     0      0      1      0      0
4       0  ...     0     1          1     0      0      1      0      0

   zogby  zone
0      0     1
1      0     0
2      0     0
3      0     0
4      0     0

[5 rows x 1545 columns]
```

In [2]:
```
import pandas as pd

# Drop any rows with missing values
data.dropna(inplace=True)
```

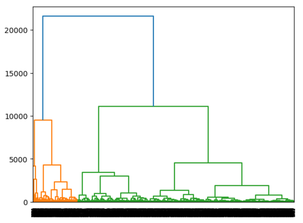
In [3]:
```
from scipy.spatial.distance import pdist, squareform
# Compute the Euclidean distances between all pairs of articles
distances = pdist(data.iloc[:, 2:], metric='euclidean')
```

In [4]:
```
# Convert the distances to a square matrix
dist_matrix = squareform(distances)
```

In [5]:
```
from scipy.cluster.hierarchy import linkage, dendrogram
# Generate the linkage matrix using the ward method
linkage_matrix = linkage(dist_matrix, method='ward')
```

```
/var/folders/31/1nz5gg1549b5s7vbknbc011r0000gn/T/ipykernel_18548/2580125479.py:3: ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation matrix looks suspiciously like an uncondensed distance matrix
  linkage_matrix = linkage(dist_matrix, method='ward')
```

In [6]:
```
import matplotlib.pyplot as plt
# Plot the dendrogram
dendrogram(linkage_matrix)
plt.show()
```



The computation of Euclidean distances between all pairs of articles may take a while because the dataset contains a large number of articles and each article has many features. This means that the distances need to be computed between a large number of high-dimensional points, which is a computationally very expensive task. Also using of the pdist function in the scipy library computes the pairwise distances for all possible combinations, which increases the computational complexity. That's why this takes a lot of time.

2. The number of clusters for grouping news articles or blog posts would depend on the objective. If the goal is to have general categories that has a wide range of topics, a smaller number of clusters such as around 4 or 5 may be suitable. However, if more specific categories are needed to capture and if more detail is necessary, then a larger number of clusters like around 9 or 10 or more may be necessary.

In [7]:
```
from scipy.cluster.hierarchy import linkage, fcluster

# Performing hierarchical clustering
Z = linkage(distances, method='ward')
```

In [8]:
```
# Assigning cluster labels
labels = fcluster(Z, t=7, criterion='maxclust')
```

In [9]:
```
# Adding cluster labels to the data
data['cluster'] = labels
```

In [10]:
```
# Printing the number of observations in cluster 3
len(data[data['cluster'] == 3])
```

Out[10]:
```
346
```

In [11]:
```
# Printing the cluster with the most num of observations
data['cluster'].value_counts().idxmax()
```

Out[11]:
```
2
```

In [12]:
```
# Printing the cluster with least num of observations
data['cluster'].value_counts().idxmin()
```

Out[12]:
```
5
```

3. So, looks like there are 346 number of obersvations in cluster 3. cluster 2 has the most number of observations. cluster 5 has the least number of observations.

In [13]:
```
from scipy.cluster.hierarchy import fcluster

# Performing hierarchical clustering with 7 clusters as in the previous case (assumption as no of clusters in not given in the question)
clusters = fcluster(Z, t=7, criterion='distance')
```

In [14]:
```
# Adding the cluster assignments to the data DataFrame
data['cluster'] = clusters
```

In [15]:
```
# Selecting only the articles in cluster 1
cluster1 = data[data['cluster'] == 1]
```

In [16]:
```
# Computing the mean frequency of each word in cluster 1
mean_freqs = cluster1.iloc[:, 2:-1].mean()
```

In [17]:
```
# printing the 6 words that occur most frequently
top_words = mean_freqs.sort_values(ascending=False)[:6].index.tolist()
print(top_words)
```

```
['november', 'vote', 'bush', 'kerry', 'elect', 'challenge']
```

4. These are the top 6 words that occured most frequently in cluster 1 - 'november', 'vote', 'bush', 'kerry', 'elect', 'challenge'. Also the most frequent word in this cluster, in terms of average value is November.

In [38]:
```
#building a loop in order to make things easier. This is to find out answers to the give questions
from scipy.cluster.hierarchy import fcluster

# Performing hierarchical clustering with 7 clusters (using 7 clusters again)
clusters = fcluster(Z, t=7, criterion='maxclust')

# Assigning the cluster elemeent to the data DataFrame
data['cluster'] = clusters

# Looping over each cluster and compute the top 6 words
for i in range(1, 8):
    # Selecting only the articles in the current cluster
    cluster_i = data[data['cluster'] == i]

    # Computing the mean frequency of each word in the current cluster so to find the most frquent words
    mean_freqs = cluster_i.iloc[:, 2:-1].mean()

    # printing the 6 words that occur most frequently in the current cluster in a loop
    top_words = mean_freqs.sort_values(ascending=False)[:6].index.tolist()
    print(f"Top 6 words in cluster {i}:", top_words)
```

```
Top 6 words in cluster 1: ['november', 'poll', 'vote', 'challenge', 'democrat', 'bush']
Top 6 words in cluster 2: ['bush', 'democrat', 'elect', 'kerry', 'republican', 'state']
Top 6 words in cluster 3: ['bush', 'iraq', 'war', 'administration', 'presided', 'american']
Top 6 words in cluster 4: ['republican', 'democrat', 'vote', 'bush', 'state', 'campaign']
Top 6 words in cluster 5: ['democrat', 'parties', 'state', 'republican', 'senate', 'seat']
Top 6 words in cluster 6: ['kerry', 'bush', 'poll', 'percent', 'voter', 'general']
Top 6 words in cluster 7: ['dean', 'kerry', 'edward', 'democrat', 'poll', 'clark']
```

5 .Which cluster could best be described as the cluster related to the Iraq war? - Cluster 3

which cluster best corresponds to the democratic party? - Cluster 7

6. Using KNN

In [40]:
```
from sklearn.cluster import KMeans

# Performing k-means clustering with 7 clusters
kmeans = KMeans(n_clusters=7, random_state=1000).fit(data.iloc[:, 2:-1])
```

In [41]:
```
# Assigning the cluster elemeent to the data DataFrame
data['cluster'] = kmeans.labels_
```

In [42]:
```
# Selecting only the articles in cluster 3
cluster3 = data[data['cluster'] == 2]
```

In [44]:
```
# printing the number of observations in cluster 3
print(len(cluster3))
```

```
329
```

In [46]:
```
# printing the cluster with the most no of observations
max_cluster = data['cluster'].value_counts().idxmax()
print(max_cluster)
```

```
4
```

In [47]:
```
# Printing the cluster with the least no of observations
min_cluster = data['cluster'].value_counts().idxmin()
print(min_cluster)
```

```
3
```

So, looks like there are 329 number of obersvations in cluster 3. cluster 4 has the most number of observations. cluster 3 has the least number of observations. All these values are the outcomes if KNN algo is used instead of Hierarchical Clustering algo

7.

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

# Performing k-means clustering with 7 clusters
kmeans = KMeans(n_clusters=7, random_state=1000).fit(data.iloc[:, 2:-1])

# Assigning the cluster labels to the data DataFrame
data['cluster'] = kmeans.labels_

# Looping over each cluster and computing the top 6 words
for i in range(1, 8):
    # Selecting only the articles in the current cluster
    cluster_i = data[data['cluster'] == i]

    # Computing the mean frequency of each word in the current cluster
    mean_freqs = cluster_i.iloc[:, 2:-1].mean()

    # Printing the 6 words that occur most frequently in the current cluster
    top_words = mean_freqs.sort_values(ascending=False)[:6].index.tolist()
    print(f"Top 6 words in cluster {i}:", top_words)
```

```
Top 6 words in cluster 1: ['bush', 'iraq', 'war', 'administration', 'american', 'iraqi']
Top 6 words in cluster 2: ['november', 'poll', 'vote', 'challenge', 'bush', 'democrat']
Top 6 words in cluster 3: ['democrat', 'parties', 'republican', 'state', 'seat', 'senate']
Top 6 words in cluster 4: ['bush', 'kerry', 'poll', 'democrat', 'general', 'elect']
Top 6 words in cluster 5: ['bush', 'kerry', 'poll', 'presided', 'democrat', 'campaign']
Top 6 words in cluster 6: ['democrat', 'republican', 'elect', 'state', 'senate', 'parties']
Top 6 words in cluster 7: ['ability', 'abortion', 'absolute', 'abstain', 'abu', 'abuse']
```

7. Which k-means cluster best corresponds to the Iraq War? - Cluster 1

which cluster best corresponds to the democratic party? - Cluster 4/5 (cannot conclude perfectly, output words look similar)

In [54]:
```
import pandas as pd

# Assigning the hierarchical clustering labels to a variable
hierarchical_labels = data['cluster']

# Assigning the k-means clustering labels to a variable
kmeans_labels = kmeans.labels_

# Creating the contingency table
ct = pd.crosstab(hierarchical_labels, kmeans_labels)

# Printing the contingency table
print(ct)
```

```
col_0      0    1    2  3    4    5    6
cluster
0        152    0    0  0    0    0    0
1          0  333    0  0    0    0    0
2          0    0  329  0    0    0    0
3          0    0    0 49    0    0    0
4          0    0    0  0 1060    0    0
5          0    0    0  0    0  305    0
6          0    0    0  0    0    0  402
```