Community Benefits Code for Exploratory, Keyword, and Clustering Analysis

Manuscript Version

This notebook covers analysis for the exploratory, keyword frequency, and clustering analysis as described in the manuscript.

In [ ]:

```
import pandas as pd
import numpy as np
import os
import pathlib
import requests
import json
import re
import nltk
import geopandas as gpd
import matplotlib.pyplot as plt
import hdbscan
import seaborn as sns
import networkx as nx
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.manifold import TSNE
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
from dotenv import load_dotenv
from nltk.tokenize import sent_tokenize
```

Load relevant datasets and merge

In [ ]:

```
def load_path_to_data():
    """
    Load the path to the data location from the environment variables.

    Returns:
        str: The path to the data location.
    """
    load_dotenv()
    path_to_data = "../data/"

    if not path_to_data:
        raise EnvironmentError("PATH_TO_DATA is not set in the .env file.")

    return path_to_data

# Load the path to data
location = load_path_to_data()
```

In [ ]:

```
# Define file names
raw_filename = 'CBI_F990_SchH_20230612.txt'
q6_filename = 'CBI_Qual_Part6_20230612.txt'
mapping_filename = 'CBI_Qual_Keys_20230612.txt'

# Load the metadata DataFrame
raw_df = pd.read_csv(
    location + raw_filename,
    delimiter='|',
    encoding='latin1',
    low_memory=False
)

# Load the text for Part 6 DataFrame
df_q6 = pd.read_csv(
    location + q6_filename,
    delimiter='|',
    encoding='latin1',
    engine='python'
)

# Load the crosswalk DataFrame
mapping_df = pd.read_csv(
    location + mapping_filename,
    delimiter='|',
    encoding='latin1'
)
```

In [ ]:

```
# Group the DataFrame by 'TAX_YEAR' and count unique 'EIN' values for each g
yearly_ein_counts = raw_df.groupby('TAX_YEAR')['EIN'].nunique()

# Reset the index to turn the result into a DataFrame
yearly_ein_counts = yearly_ein_counts.reset_index()

# Display
yearly_ein_counts
```

Out[ ]:

	TAX_YEAR	EIN
0	2010	2327
1	2011	2306
2	2012	2304
3	2013	2285
4	2014	2229
5	2015	2206
6	2016	2164
7	2017	2174
8	2018	2127
9	2019	2071
10	2020	2108
11	2021	909

In [ ]:

```
# Select specific columns and rename 'ExplanationTxt' to 'Desc' in a new Dat
df = df_q6[['ObjectID', 'ExplanationTxt']].rename(columns={'ExplanationTxt':
```

In [ ]:

```
# List of columns to select
raw_col_list = ['EIN', 'F990_Name', 'Hospital_Name', 'TAX_YEAR', 'F990_State

# Create a DataFrame subset with selected columns and reset the index
raw_df_subset = raw_df[raw_col_list].reset_index(drop=True)
```

In [ ]:

```
# Merge raw data subset with the mapping DataFrame based on 'EIN' and 'TAX_Y
crosswalk_df = pd.merge(raw_df_subset, mapping_df, how='right', on=['EIN', '

# Merge the current DataFrame 'df' with 'crosswalk_df' based on 'ObjectID'
df = df.merge(crosswalk_df, how='left', on='ObjectID')
```

Text Preprocessing

The following steps were taken to clean the text data:

1. Converting all characters to lowercase
2. Removing symbols

In [ ]:

```
# Convert to lowercase for text analysis
df['clean_text'] = df['Desc'].str.lower()

# Reset the index to ensure continuous indexing
df = df.reset_index(drop=True)
```

In [ ]:
```
# Select relevant columns and remove duplicate rows
unique_df = df[['F990_Name', 'Hospital_Name', 'EIN', 'TAX_YEAR',
                'F990_State',  'clean_text']].drop_duplicates()

# Add a column for better identification
unique_df['f990_system_with_year'] = '[' + unique_df['F990_Name'].astype(str

# Remove symbols
unique_df['clean_text'] = unique_df['clean_text'].astype(str)
unique_df['clean_text'] = unique_df['clean_text'].str.replace('[,()-]', '')

print("Data Shape:", unique_df.shape)
```

```
Data Shape: (461221, 7)
```

Create Relevant DF

In [ ]:
```
# Group 'unique_df' by 'EIN' and 'TAX_YEAR', joining 'clean_text' with comma
system_df = unique_df.groupby(['EIN', 'TAX_YEAR'])['clean_text'].apply(','.j

# Reset the index and rename 'index' column to 'id'
system_df = system_df.reset_index().rename(columns={'index': 'id'})
```

In [ ]:
```
# Fill missing values in 'F990_State' with 'No State Reported'
unique_df['F990_State'] = unique_df['F990_State'].fillna('No State Reported'
```

In [ ]:
```
# Find the most common 'F990_State' for each 'EIN'
system_most_common_state = unique_df.groupby(['EIN'])['F990_State'].agg(lamb

# Fill missing values in 'Hospital_Name' with 'No Name Reported'
unique_df['Hospital_Name'] = unique_df['Hospital_Name'].fillna('No Name Repo


# Find the most common 'Hospital_Name' for each 'EIN'
system_most_common_hospital_name = unique_df.groupby(['EIN'])['Hospital_Name
```

In [ ]:
```
# Merge 'system_df' with 'system_most_common_state' based on 'EIN' (left joi
system_df = system_df.merge(system_most_common_state, on='EIN', how='left')

# Merge 'system_df' with 'system_most_common_hospital_name' based on 'EIN' (
system_df = system_df.merge(system_most_common_hospital_name, on='EIN', how=
```

Keyword Frequency

In [ ]:
```
# Define keyword names and groups
keyword_names = ["covid"]
keyword_groups = [["covid", "coronavirus",
                   "pandemic", "telehealth", "mask", "vaccin",
                   " ppe ", "personal protective equipment",
                   "covid test"]]

# Flatten the keyword_groups into a single list
```

```python
keyword_list = [item for sublist in keyword_groups for item in sublist]

# Calculate the length of keyword_names
len_keyword_names = len(keyword_names)

# Calculate the total number of keywords in keyword_groups
total_keywords_count = sum(len(listElem) for listElem in keyword_groups)
```

In [ ]:

```python
# Iterate through each keyword in keyword_list
for keyword in keyword_list:
    print(keyword)
    # Create columns for keyword counts and flags
    count_col = 'count_' + keyword
    flag_col = 'flag_' + keyword

    # Count occurrences of the keyword in 'clean_text' and set flags
    system_df[count_col] = system_df['clean_text'].apply(lambda x: x.count(k
    system_df[flag_col] = np.where(system_df[count_col] > 0, 1, 0)
```

```
covid
coronavirus
pandemic
telehealth
mask
vaccin
 ppe
personal protective equipment
covid test
```

In [ ]:

```python
# Calculate the length of 'clean_text' in terms of word count
system_df['clean_text_length'] = system_df['clean_text'].str.split().str.len

# Convert 'TAX_YEAR' to datetime format
system_df['year_as_date'] = pd.to_datetime(system_df['TAX_YEAR'], format='%Y
```

In [ ]:

```python
# Export df for review
# system_df.to_csv('r_and_r_data/manuscript_covid_analysis_r_and_r.csv', inc
```

In [ ]:

```python
system_df.shape
```

Out[ ]:

```
(25131, 26)
```

Which words are used most by year?

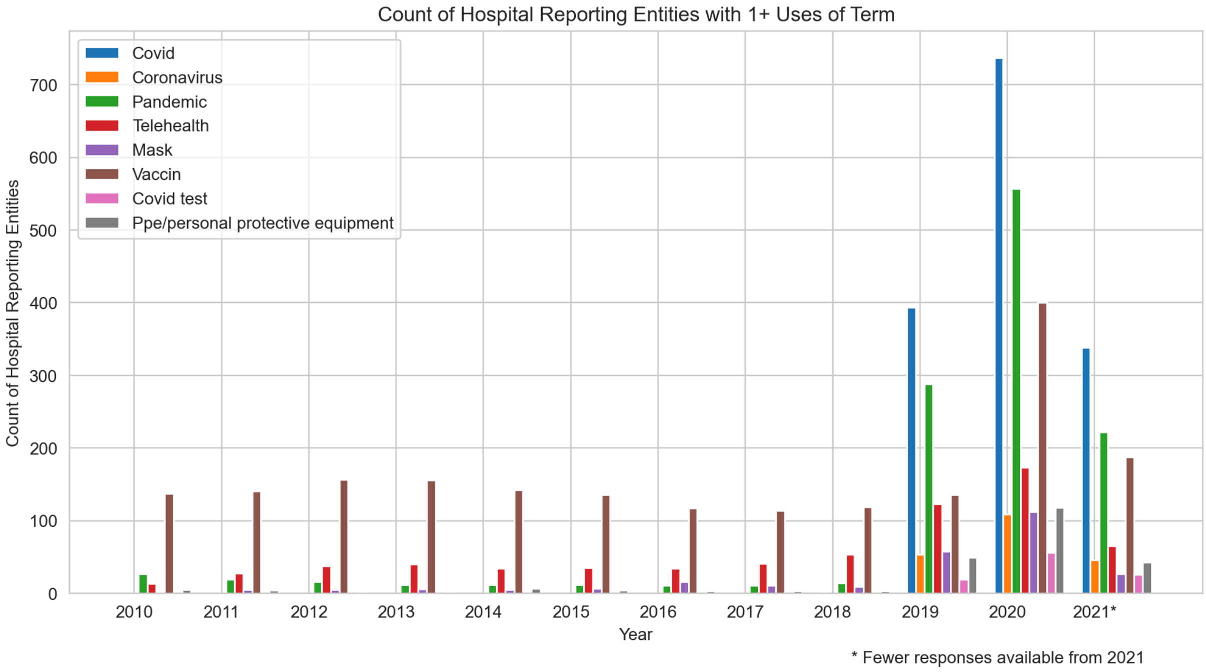
In [ ]:

```python
# Create a flag for 'PPE/Personal Protective Equipment' by combining flags f
system_df['flag_PPE/Personal Protective Equipment'] = ((system_df['flag_ ppe

# Create a list of flag columns to analyze
flag_cols = [col for col in system_df.columns if col.startswith('flag_') and

print(flag_cols)

# Count the number of unique 'EIN' by year for each flag
count_by_year = {flag: system_df[system_df[flag] == 1].groupby('TAX_YEAR')['
```

```
['flag_covid', 'flag_coronavirus', 'flag_pandemic', 'flag_telehealth', 'flag
_mask', 'flag_vaccin', 'flag_covid test', 'flag_PPE/Personal Protective Equi
pment']
```

In [ ]:

```
%config InlineBackend.figure_format = 'retina'  # For high-resolution displa

plt.rcParams['figure.figsize'] = (12, 6)  # Set the desired width and height
sns.set_style("whitegrid")

years = sorted(df['TAX_YEAR'].unique())

# Set parameters for bar width
bar_width = 0.1
r = list(range(len(years)))

fig, ax = plt.subplots()

for idx, (flag, count) in enumerate(count_by_year.items()):
    legend_label = flag.split('flag_')[1].capitalize()
    plt.bar(
        [x + idx*bar_width for x in r],
        [count.get(year, 0) for year in years],
        width=bar_width,
        label=legend_label
    )

plt.xlabel('Year')
plt.ylabel('Count of Hospital Reporting Entities')
plt.title('Count of Hospital Reporting Entities with 1+ Uses of Term')
plt.xticks([r + bar_width for r in range(len(years))], years)

# Modify x-axis labels to add an asterisk for the year 2021
ax.set_xticklabels([str(year) if year != 2021 else str(year) + '*' for year

plt.legend()
plt.figure(figsize=(12, 6))

# Add a caption below the plot
fig.supxlabel('* Fewer responses available from 2021', fontsize=10, x=0.76)

plt.tight_layout()
plt.show()
```



```
<Figure size 1200x600 with 0 Axes>
```

Example Sentences for Each Group of Terms

```
In [ ]: nltk.download('punkt')
        nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /Users/ehadley/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/ehadley/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
Out[ ]: True
```

```
In [ ]: # Prepare the DataFrame for sentence extraction
        min_df = system_df[['id', 'EIN', 'Hospital_Name', 'TAX_YEAR', 'F990_State',
```

```
In [ ]: system_df.shape
```

```
Out[ ]: (25131, 27)
```

```
In [ ]: min_df.shape
```

```
Out[ ]: (25131, 6)
```

```
In [ ]: def count_sentences_nltk(text):
            sentences = sent_tokenize(text)
            return len(sentences)

        total_sentence_count = min_df['clean_text'].apply(count_sentences_nltk).sum(
print(total_sentence_count)
```

```
2735378
```

In [ ]:

```
keyword_groups = [["covid", "coronavirus", "pandemic", "telehealth", "mask",
keyword_names = ["covid"]

# Loop through each keyword group
for i in range(len(keyword_groups)):
    searched_words = keyword_groups[i]
    col_name = keyword_names[i] + '_sentences'

    # Extract sentences containing the searched words using NLTK's sent_toke
    min_df[col_name] = min_df['clean_text'].apply(lambda text: [sentence for
```

```
/var/folders/hz/h2x015256dx58q12mx7ft6680000gn/T/ipykernel_46895/2015770836.
py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  min_df[col_name] = min_df['clean_text'].apply(lambda text: [sentence for s
entence in sent_tokenize(text) if any(searched_word in sentence for searched
_word in searched_words)])
```

In [ ]:

```
min_df.shape
```

Out[ ]:

```
(25131, 7)
```

In [ ]:

```
# min_df.to_csv('r_and_r_data/min_df_r_and_r.csv', index=False)
```

In [ ]:

```
# Remove the 'clean_text' column
export_df = min_df.drop(columns='clean_text')
# Filter the DataFrame to keep rows with at least 1 COVID-related sentence
export_df = export_df[export_df['covid_sentences'].str.len() >= 1]
# Export the DataFrame to a CSV file
# export_df.to_csv('r_and_r_data/r_and_rcovid_extracted_sentences.csv', inde
```

In [ ]:

```
export_df.shape
```

Out[ ]:

```
(8658, 6)
```

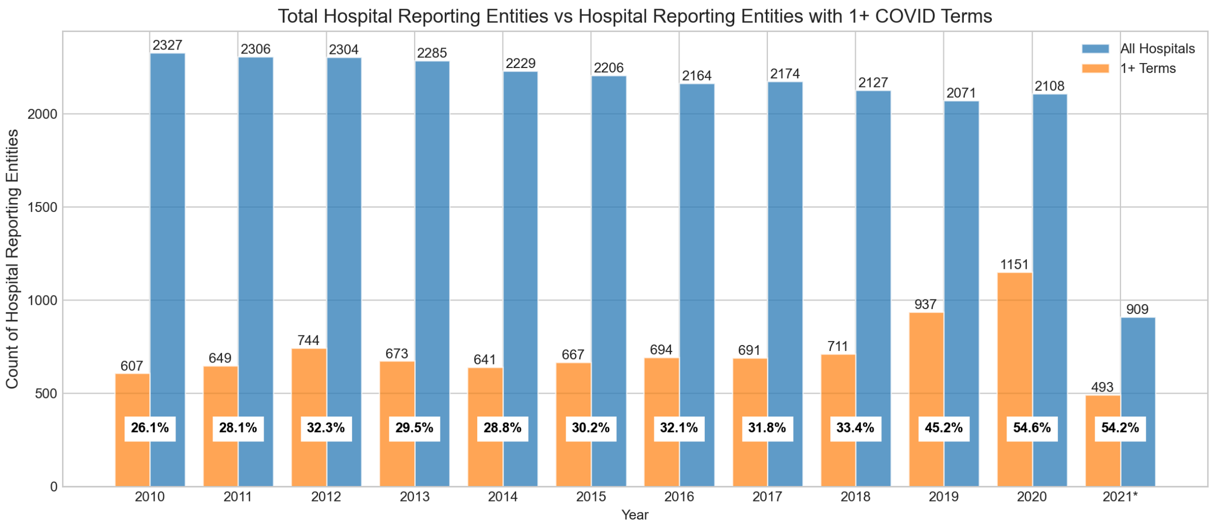
How many hospitals are using COVID terms by year?

In [ ]:

```
%matplotlib inline

plt.style.use('seaborn-whitegrid')

all_count = raw_df.groupby('TAX_YEAR')['EIN'].nunique().reset_index()
counts_by_year = export_df.groupby('TAX_YEAR')['EIN'].nunique().reset_index(

# Create a figure and axis
```

```
fig, ax = plt.subplots(figsize=(15, 6))

# Plot the bars for 'All Hospitals'
bars1 = ax.bar(all_count['TAX_YEAR'] + 0.2, all_count['EIN'], label='All Hos

# Plot the bars for '1+ Terms'
bars2 = ax.bar(counts_by_year['TAX_YEAR'] - 0.2, counts_by_year['count'], la

# Set custom tick marks for every year
years = all_count['TAX_YEAR'].tolist()
ax.set_xticks(years)

# Set the X-label and Y-label
ax.set_xlabel('Year')
ax.set_ylabel('Count of Hospital Reporting Entities', fontsize=12)

# Set the title
ax.set_title('Total Hospital Reporting Entities vs Hospital Reporting Entiti

# Add a legend
ax.legend(loc='upper right')

# Add annotations to each bar and percentage overlay
for bar1, bar2 in zip(bars1, bars2):
    height1 = bar1.get_height()
    height2 = bar2.get_height()

    # Annotations for bar heights
    ax.annotate(f'{int(height1)}', (bar1.get_x() + bar1.get_width() / 2, hei
                ha='center', va='bottom', fontsize=10)
    ax.annotate(f'{int(height2)}', (bar2.get_x() + bar2.get_width() / 2, hei
                ha='center', va='bottom', fontsize=10)

    # Percentage overlay annotation
    percent = (height2 / height1) * 100 if height1 != 0 else 0
    ax.annotate(f'{percent:.1f}%', ((bar1.get_x() + bar2.get_x()) / 2 + 0.2,
                ha='center', va='bottom', fontsize=10, color='black', fontwe
                backgroundcolor='white')


# Modify x-axis labels to add an asterisk for the year 2021
ax.set_xticklabels([str(year) if year != 2021 else str(year) + '*' for year

# Add a caption below the plot
fig.supxlabel('* Fewer responses available from 2021', fontsize=10, x=0.76)

# Display the plot
plt.show()
```

```
/var/folders/hz/h2x015256dx58q12mx7ft6680000gn/T/ipykernel_46895/2154627249.
py:3: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib
are deprecated since 3.6, as they no longer correspond to the styles shipped
by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. A
lternatively, directly use the seaborn API instead.
  plt.style.use('seaborn-whitegrid')
```



* Fewer responses available from 2021

Can we cluster COVID response into topics?

In [ ]:

```
# Prepare data for cluster analysis
covid_df = min_df.drop(columns='clean_text')
covid_df = covid_df[covid_df['covid_sentences'].str.len() >= 1]
```

In [ ]:

```
covid_df.shape
```

Out[ ]:

```
(8658, 6)
```

In [ ]:

```
# Cluster preprocessing
covid_df_exploded = covid_df.explode('covid_sentences')[['covid_sentences']]

def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)
    text = ' '.join([word for word in text.split() if word not in ENGLISH_ST
    return text

covid_df_exploded['processed'] = covid_df_exploded['covid_sentences'].apply(
```

In [ ]:

```
covid_df_exploded[covid_df_exploded['processed'].str.len() >= 1].shape
```

Out[ ]:

```
(10909, 2)
```

In [ ]:

```
# TF-IDF Vectorization
vectorizer = TfidfVectorizer(stop_words='english')
tfidf = vectorizer.fit_transform(covid_df_exploded['processed'])
```

In [ ]:

```
# Clustering using HDBSCAN
clusterer = hdbscan.HDBSCAN(min_cluster_size=40, min_samples=10)
covid_df_exploded['clusters'] = clusterer.fit_predict(tfidf.toarray())
```

In [ ]:
```
# Save results
covid_df_exploded.to_csv('r_and_r_data/covid_df_exploded_sentences_manuscrip
```

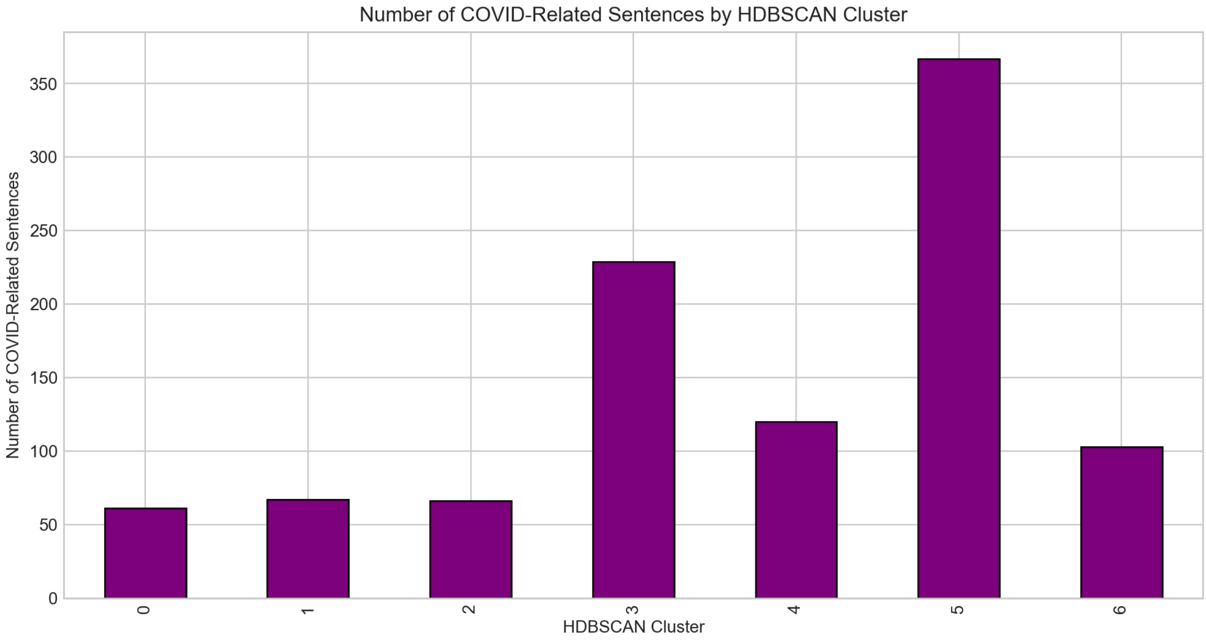
In [ ]:
```
# View clusters and shapes

cluster_counts = covid_df_exploded['clusters'].value_counts()
dataframe_shape = covid_df_exploded.shape

cluster_counts, dataframe_shape
```

Out[ ]:
```
(clusters
 -1    9896
  5     367
  3     229
  4     120
  6     103
  1      67
  2      66
  0      61
 Name: count, dtype: int64,
 (10909, 3))
```

In [ ]:
```
# Calculate the sum of cluster counts excluding cluster -1
sum_cluster_counts = cluster_counts[cluster_counts.index != -1].sum()

# Print the sum of cluster counts
print(f'Sum of cluster counts (excluding -1): {sum_cluster_counts}')
```

```
Sum of cluster counts (excluding -1): 1013
```

In [ ]:
```
%matplotlib inline
# Prepare data
counts_by_cluster = covid_df_exploded.groupby('clusters').size().reset_index
counts_by_cluster = counts_by_cluster[(counts_by_cluster["count"] < 1000)]
counts_by_cluster.plot(x='clusters', y='count', kind='bar', edgecolor="black

# Add labels and a title
plt.xlabel('HDBSCAN Cluster')
plt.ylabel('Number of COVID-Related Sentences')
plt.title('Number of COVID-Related Sentences by HDBSCAN Cluster')

# Show the plot
plt.show()
```



In [ ]:

```
# Limit to smaller clusters
interested_clusters = counts_by_cluster[counts_by_cluster["count"] < 1000]["
filtered_df = covid_df_exploded[covid_df_exploded['clusters'].isin(intereste
```

In [ ]:

```
vectorizer = TfidfVectorizer(max_df=0.5, stop_words='english', max_features=
X = vectorizer.fit_transform(filtered_df['covid_sentences'])
```

TABLE 1

In [ ]:

```
def get_top_keywords(data, clusters, labels, n_terms):
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()
    cluster_summaries = []

    for i, r in df.iterrows():
        keywords = ','.join([labels[t] for t in np.argsort(r)[-n_terms:]])
        cluster_summaries.append((i, keywords))
        print('\nCluster {}'.format(i))
        print(keywords)

    summary_df = pd.DataFrame(cluster_summaries, columns=['Cluster', 'Top_Ke

    return summary_df

export_hdbscan_summary = get_top_keywords(X, filtered_df['clusters'].values,
Cluster 0
social,systems,member,largest,country,delivery,catholic,community,health,tri
nity

Cluster 1
upper,assistance,financial,ummc,revised,chesapeake,areas,patient,patients,av
ailable

Cluster 2
improvement,director,partnership,st,dept,joseph,health,chip,county,chippewa

Cluster 3
adults,houston,vaccines,administered,2020,vaccinations,held,pandemic,19,covi
d

Cluster 4
2019,implementation,strategy,health,assessment,list,needs,community,appendi
x,chna

Cluster 5
cost,assistance,line,charity,patient,bad,care,debt,expense,financial

Cluster 6
individuals,medical,reading,registration,non,speaking,clerks,translators,rea
d,incapable
```

In [ ]:

```
export_hdbscan_summary.to_csv('r_and_r_data/r_and_r_export_hdbscan_summary_m
```

In [ ]:

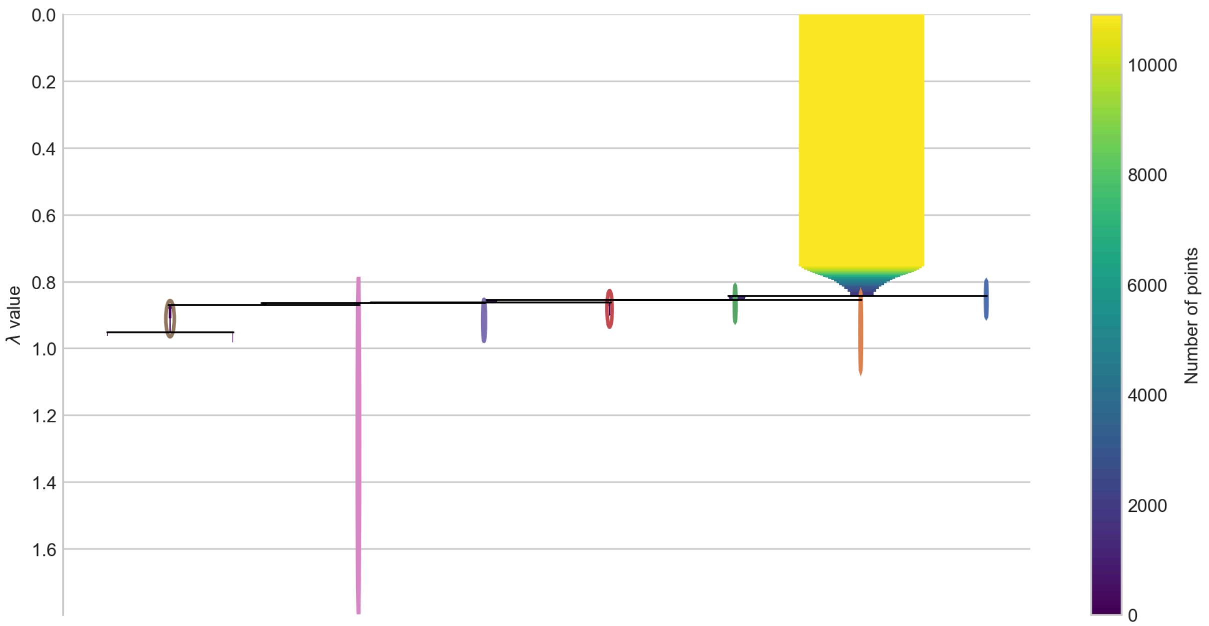
```
covid_df_exploded[covid_df_exploded.clusters != -1].to_csv('r_and_r_data/r_a
```

In [ ]:

```
clusterer.condensed_tree_.plot(select_clusters=True, selection_palette=sns.c
```

Out[ ]:

```
<Axes: ylabel='$\\lambda$ value'>
```


λ value
Number of points