Paper Visualization

Emerson Johnston

```
import gc
import matplotlib.pyplot as plt
import networkx as nx
import nltk
import numpy as np
import os
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import re
import scipy
import seaborn as sns
from collections import Counter
from datetime import datetime
from itertools import combinations
from matplotlib.patches import Polygon
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from scipy.spatial import ConvexHull
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer

# Directories
output_directory = "/Users/emerson/Github/usenet_webpage"
threads_directory = os.path.join(output_directory, "CSV Files/Threads")
comments_directory = os.path.join(output_directory, "CSV Files/Comments")
images_dir = os.path.join(output_directory, "Images and Tables/Images")
tables_dir = os.path.join(output_directory, "Images and Tables/Tables")

# Load cleaned datasets
dataset1_threads = pd.read_csv(os.path.join(threads_directory, "dataset1_threads.csv"))
dataset1_comments = pd.read_csv(os.path.join(comments_directory, "dataset1_comments.csv"))
dataset2_threads = pd.read_csv(os.path.join(threads_directory, "dataset2_threads.csv"))
dataset2_comments = pd.read_csv(os.path.join(comments_directory, "dataset2_comments.csv"))
dataset3_threads = pd.read_csv(os.path.join(threads_directory, "dataset3_threads.csv"))
dataset3_comments_all = pd.read_csv(os.path.join(comments_directory, "dataset3_comments_all.csv"))
dataset3_comments_onlyinfluential = pd.read_csv(os.path.join(comments_directory, "dataset3_comments_onl
influential_authors = pd.read_csv(os.path.join(output_directory, "CSV Files/influential_authors.csv"))
```

Cool Initial Visualizations

```
print(dataset2_comments.columns)
```

```
## Index(['NG_TH_CM_ID', 'NG_TH_ID', 'TH_CM_ID', 'CM_ID', 'TH_ID', 'NG_ID',
##        'Author', 'Date.and.Time', 'Full.Text', 'URL.String', 'newsgroup',
##        'Hour', 'Date', 'SentimentScore'],
##       dtype='object')
```

```
print(dataset3_comments_all.columns)
```

```
## Index(['NG_TH_CM_ID', 'NG_TH_ID', 'TH_CM_ID', 'CM_ID', 'TH_ID', 'NG_ID',
##        'Author', 'Date.and.Time', 'Full.Text', 'URL.String', 'newsgroup',
##        'Hour', 'Date', 'SentimentScore'],
##       dtype='object')
```

Author Impact

Topic Modeling

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import numpy as np
import pandas as pd
import re

# Preprocessing function
def preprocess_text(text):
    stop_words = set(stopwords.words("english"))
    lemmatizer = WordNetLemmatizer()
    text = re.sub(r"\W+", " ", text)  # Remove non-alphanumeric characters
    tokens = word_tokenize(text.lower())
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words and len(word) > 2
    return " ".join(tokens)

# Preprocess datasets
dataset2_text = dataset2_comments["Full.Text"].dropna().astype(str).apply(preprocess_text)
dataset3_text = dataset3_comments_all["Full.Text"].dropna().astype(str).apply(preprocess_text)

# Vectorize text
def vectorize_text(data):
    vectorizer = CountVectorizer(max_df=0.9, min_df=5, stop_words="english")
    doc_term_matrix = vectorizer.fit_transform(data)
    return doc_term_matrix, vectorizer
```

```python
# Determine optimal number of topics
def find_optimal_topics(doc_term_matrix, topic_range=(1, 10)):
    coherence_values = []
    for n_topics in range(topic_range[0], topic_range[1] + 1):
        lda = LatentDirichletAllocation(n_components=n_topics, random_state=42)
        lda.fit(doc_term_matrix)
        coherence_values.append(lda.perplexity(doc_term_matrix))
    optimal_n_topics = coherence_values.index(min(coherence_values)) + topic_range[0]
    return optimal_n_topics, coherence_values

# Extract topics and top words
def extract_topics(lda_model, vectorizer, n_top_words=20):
    topics = []
    for topic_idx, topic in enumerate(lda_model.components_):
        top_words = [vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-n_top_words - 1:-
        topics.append((f"Topic {topic_idx + 1}", top_words))
    return topics

# Process Dataset 2
dataset2_matrix, dataset2_vectorizer = vectorize_text(dataset2_text)
optimal_topics_2, coherence_2 = find_optimal_topics(dataset2_matrix)
lda_dataset2 = LatentDirichletAllocation(n_components=optimal_topics_2, random_state=42)
lda_dataset2.fit(dataset2_matrix)
```

```
## LatentDirichletAllocation(random_state=42)
```

```python
topics_2 = extract_topics(lda_dataset2, dataset2_vectorizer)

# Process Dataset 3
dataset3_matrix, dataset3_vectorizer = vectorize_text(dataset3_text)
optimal_topics_3, coherence_3 = find_optimal_topics(dataset3_matrix)
lda_dataset3 = LatentDirichletAllocation(n_components=optimal_topics_3, random_state=42)
lda_dataset3.fit(dataset3_matrix)
```

```
## LatentDirichletAllocation(n_components=7, random_state=42)
```

```python
topics_3 = extract_topics(lda_dataset3, dataset3_vectorizer)

# Output results
print(f"Optimal number of topics for Dataset 2: {optimal_topics_2}")
```

```
## Optimal number of topics for Dataset 2: 10
```

```python
print(f"\nOptimal number of topics for Dataset 3: {optimal_topics_3}")
```

```
##
## Optimal number of topics for Dataset 3: 7
```

```python
def save_topics_to_html(lda_model, vectorizer, output_path, n_top_words=20):
    topics_data = []
    for topic_idx, topic in enumerate(lda_model.components_):
        top_words = [vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-n_top_words - 1:-
        top_betas = [topic[i] for i in topic.argsort()[:-n_top_words - 1:-1]]
        topic_dict = {
            f"Topic_{topic_idx + 1}_terms": top_words,
            f"Topic_{topic_idx + 1}_betas": top_betas
        }
        topics_data.append(pd.DataFrame(topic_dict))

    # Combine all topics into a single DataFrame
    topics_df = pd.concat(topics_data, axis=1)
    topics_df.to_html(output_path, index=False)
    print(f"Topics saved to: {output_path}")

# Save topics for Dataset 2 (All Comments)
save_topics_to_html(
    lda_model=lda_dataset2,
    vectorizer=dataset2_vectorizer,
    output_path="/Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_all_comments
    n_top_words=20
)
```

```
## Topics saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_all_comme
```

```
# Save topics for Dataset 3 (Influential Authors)
save_topics_to_html(
    lda_model=lda_dataset3,
    vectorizer=dataset3_vectorizer,
    output_path="/Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_influential_
    n_top_words=20
)
```

```
## Topics saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_influenti
```

Topic Similarity

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import os
from itertools import combinations

def compare_topics_and_save(general_results_path, influential_results_path, output_dir):
    """
    Compare topics between general and influential author analyses
    using Jaccard similarity and create visualizations.
    """
    print("\nComparing topics with Jaccard similarity...")

    # Load general and influential results from HTML
    general_results = pd.read_html(general_results_path)[0]
```

```python
influential_results = pd.read_html(influential_results_path)[0]

# List to store topic similarity data
topic_similarities = []

# Calculate Jaccard similarity for all topic pairs
for general_topic in range(1, len(general_results.columns) // 2 + 1):
    general_terms = set(general_results[f"Topic_{general_topic}_terms"].dropna())
    for influential_topic in range(1, len(influential_results.columns) // 2 + 1):
        influential_terms = set(influential_results[f"Topic_{influential_topic}_terms"].dropna())
        similarity = len(general_terms.intersection(influential_terms)) / len(general_terms.union(in
        topic_similarities.append({
            'General_Topic': general_topic,
            'Influential_Topic': influential_topic,
            'Similarity': similarity
        })

# Create similarity DataFrame
comparison_df = pd.DataFrame(topic_similarities)
similarity_matrix = comparison_df.pivot(
    index='General_Topic',
    columns='Influential_Topic',
    values='Similarity'
)

# Reorder the similarity_matrix to reverse the order of General Topics
similarity_matrix = similarity_matrix.iloc[::-1]  # Reverse the row order

# Generate heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(similarity_matrix, annot=True, fmt='.3f', cmap='YlOrRd', cbar_kws={'label': 'Jaccard Si
plt.title('Topic Similarity Heatmap')
plt.xlabel('Influential Topics')
plt.ylabel('General Topics')

# Save heatmap
os.makedirs(output_dir, exist_ok=True)
heatmap_path = os.path.join(output_dir, "topic_similarity_heatmap.png")
plt.savefig(heatmap_path, dpi=300, bbox_inches='tight')
plt.close()
print(f"Heatmap saved to: {heatmap_path}")

# Find the best matches for general topics
best_matches = []
used_influential_topics = set()

for general_topic in range(1, len(general_results.columns) // 2 + 1):
    topic_similarities = comparison_df[comparison_df['General_Topic'] == general_topic]
    remaining_similarities = topic_similarities[~topic_similarities['Influential_Topic'].isin(used_

    if not remaining_similarities.empty:
        best_match = remaining_similarities.loc[remaining_similarities['Similarity'].idxmax()]
        best_matches.append({
            'General_Topic': general_topic,
            'Influential_Topic': int(best_match['Influential_Topic']),
            'Similarity': best_match['Similarity']
        })
        used_influential_topics.add(best_match['Influential_Topic'])

best_matches_df = pd.DataFrame(best_matches)

# Generate bar plot for best matches
plt.figure(figsize=(12, 6))
bars = plt.bar(
    best_matches_df['Influential_Topic'],
    best_matches_df['Similarity'],
    color=[plt.cm.Set2(i / 7) for i in best_matches_df['General_Topic']]
)

# Add value labels to bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2., height,
             f'{height:.2f}',
             ha='center', va='bottom')

# Customize bar plot
plt.title('Best Topic Matches Between General and Influential Authors')
plt.xlabel('Influential Topics')
plt.ylabel('Jaccard Similarity')
plt.ylim(0, max(best_matches_df['Similarity']) * 1.1)  # Add padding above highest bar

# Add legend for general topics
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor=plt.cm.Set2(i / 7), label=f'Topic {i}')
    for i in best_matches_df['General_Topic']
]
plt.legend(handles=legend_elements, title='Matching General Topic',
           bbox_to_anchor=(1.05, 1), loc='upper left')

# Save bar plot
barplot_path = os.path.join(output_dir, "topic_similarity_barplot.png")
plt.savefig(barplot_path, dpi=300, bbox_inches='tight')
plt.close()
print(f"Bar plot saved to: {barplot_path}")

# Save comparison DataFrame to CSV
comparison_csv_path = os.path.join(output_dir, "topic_similarity_comparison.csv")
comparison_df.to_csv(comparison_csv_path, index=False)
print(f"Comparison data saved to: {comparison_csv_path}")

# Print similarity statistics
print("\nTopic Similarity Statistics:")
print(f"Average similarity: {comparison_df['Similarity'].mean():.3f}")
print(f"Maximum similarity: {comparison_df['Similarity'].max():.3f}")
print(f"Minimum similarity: {comparison_df['Similarity'].min():.3f}")

return comparison_df, best_matches_df

# Example usage
general_results_path = "/Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_all_
influential_results_path = "/Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_
output_dir = "/Users/emerson/Github/usenet_webpage/Images and Tables/Images"

try:
comparison_df, best_matches_df = compare_topics_and_save(
    general_results_path,
    influential_results_path,
    output_dir
)
print("\nJaccard Similarity Analysis Complete.")
print("\nBest Matches DataFrame:")
print(best_matches_df)
except Exception as e:
print(f"An error occurred during topic comparison: {str(e)}")
```

```
##
## Comparing topics with Jaccard similarity...
## Heatmap saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_similarity_hea
## Bar plot saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_similarity_ba
## Comparison data saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_simila
##
## Topic Similarity Statistics:
## Average similarity: 0.134
## Maximum similarity: 0.481
## Minimum similarity: 0.000
##
## Jaccard Similarity Analysis Complete.
##
## Best Matches DataFrame:
##    General_Topic  Influential_Topic  Similarity
## 0              1                  6    0.379310
## 1              2                  7    0.333333
## 2              3                  1    0.481481
## 3              4                  4    0.379310
## 4              5                  5    0.379310
## 5              6                  2    0.052632
## 6              7                  3    0.081081
```

Statistical Signifcance

```
import pandas as pd
import statsmodels.api as sm
from scipy.stats import ttest_ind
from scipy.sparse import SparseEfficiencyWarning
import warnings
```

```python
# Suppress warnings
warnings.simplefilter("ignore", category=SparseEfficiencyWarning)
warnings.simplefilter("ignore", category=FutureWarning)

# Load the comparison data
comparison_csv_path = "/Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_similarity_c
comparison_data = pd.read_csv(comparison_csv_path)

# Regression Analysis
print("Preparing data for regression analysis...")
```

```
## Preparing data for regression analysis...
```

```python
# Ensure General_Topic and Influential_Topic are categorical
comparison_data['General_Topic'] = comparison_data['General_Topic'].astype('category')
comparison_data['Influential_Topic'] = comparison_data['Influential_Topic'].astype('category')

# Create dummy variables for categorical predictors
X = pd.get_dummies(comparison_data[['General_Topic', 'Influential_Topic']], drop_first=True)

# Add a constant for the regression model
X = sm.add_constant(X)

# Convert boolean predictors to integers and ensure all data is numeric
X = X.applymap(lambda value: int(value) if isinstance(value, bool) else value)
X = X.apply(pd.to_numeric, errors='coerce')

# Dependent variable: Similarity
y = pd.to_numeric(comparison_data['Similarity'], errors='coerce')

# Drop rows with NaN in X or y
if X.isnull().any().any() or y.isnull().any():
    print("Warning: Non-numeric or missing values detected. Dropping invalid rows.")
    valid_rows = ~(X.isnull().any(axis=1) | y.isnull())
    X = X[valid_rows]
    y = y[valid_rows]

# Verify the cleaned data
print("\nCleaned Predictors (X):")
```

```
##
## Cleaned Predictors (X):
```

```python
print(X.head())
```

```
##    const  General_Topic_2  ...  Influential_Topic_6  Influential_Topic_7
## 0    1.0                0  ...                    0                    0
## 1    1.0                0  ...                    0                    0
## 2    1.0                0  ...                    0                    0
## 3    1.0                0  ...                    0                    0
## 4    1.0                0  ...                    0                    0
```

```
##
## [5 rows x 16 columns]
```

```
print("\nCleaned Dependent Variable (y):")
```

```
##
## Cleaned Dependent Variable (y):
```

```
print(y.head())
```

```
## 0    0.176471
## 1    0.081081
## 2    0.081081
## 3    0.025641
## 4    0.176471
## Name: Similarity, dtype: float64
```

```
# Fit the regression model
try:
    model = sm.OLS(y, X).fit()
    print("\nRegression Analysis Summary:")
    print(model.summary())
except Exception as e:
    print(f"An error occurred during regression analysis: {str(e)}")
```

```
##
## Regression Analysis Summary:
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:             Similarity   R-squared:                       0.223
## Model:                            OLS   Adj. R-squared:                  0.007
## Method:                 Least Squares   F-statistic:                     1.032
## Date:                Sun, 17 Nov 2024   Prob (F-statistic):              0.438
## Time:                        06:56:20   Log-Likelihood:                 62.774
## No. Observations:                  70   AIC:                            -93.55
## Df Residuals:                      54   BIC:                            -57.57
## Df Model:                          15
## Covariance Type:            nonrobust
## =======================================================================================
##                           coef    std err          t      P>|t|      [0.025      0.975]
## ---------------------------------------------------------------------------------------
## const                   0.1632      0.054      3.037      0.004       0.055       0.271
## General_Topic_2        -0.0351      0.060     -0.585      0.561      -0.156       0.085
## General_Topic_3         0.0456      0.060      0.759      0.451      -0.075       0.166
## General_Topic_4        -0.0661      0.060     -1.100      0.276      -0.187       0.054
## General_Topic_5        -0.0143      0.060     -0.239      0.812      -0.135       0.106
## General_Topic_6        -0.0155      0.060     -0.258      0.797      -0.136       0.105
## General_Topic_7        -0.0651      0.060     -1.084      0.283      -0.186       0.055
## General_Topic_8        -0.0515      0.060     -0.858      0.395      -0.172       0.069
## General_Topic_9         0.0344      0.060      0.572      0.569      -0.086       0.155
## General_Topic_10       -0.1072      0.060     -1.785      0.080      -0.228       0.013
## Influential_Topic_2    -0.0258      0.050     -0.513      0.610      -0.127       0.075
## Influential_Topic_3    -0.0475      0.050     -0.944      0.349      -0.148       0.053
## Influential_Topic_4    -0.0081      0.050     -0.161      0.873      -0.109       0.093
## Influential_Topic_5     0.0466      0.050      0.928      0.358      -0.054       0.147
## Influential_Topic_6     0.0104      0.050      0.207      0.836      -0.090       0.111
## Influential_Topic_7     0.0143      0.050      0.285      0.777      -0.086       0.115
## ==============================================================================
## Omnibus:                       21.787   Durbin-Watson:                   1.995
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):               28.694
## Skew:                           1.379   Prob(JB):                     5.88e-07
## Kurtosis:                       4.494   Cond. No.                         11.6
## ==============================================================================
##
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Pairwise T-Test Analysis
print("\nPerforming pairwise T-tests between General Topics...")
```

```
##
## Performing pairwise T-tests between General Topics...
```

```
# Group Jaccard similarities by General_Topic
grouped_data = comparison_data.groupby('General_Topic')['Similarity']

# Perform pairwise T-tests between General_Topic groups
t_test_results = []
for (group1, group2) in combinations(grouped_data.groups.keys(), 2):
    similarity_group1 = grouped_data.get_group(group1)
    similarity_group2 = grouped_data.get_group(group2)
    t_stat, p_value = ttest_ind(similarity_group1, similarity_group2, equal_var=False)
    t_test_results.append({'Group1': group1, 'Group2': group2, 't-statistic': t_stat, 'p-value': p_valu

# Convert T-test results into a DataFrame
t_test_results_df = pd.DataFrame(t_test_results)

# Display significant T-test results
significant_results = t_test_results_df[t_test_results_df['p-value'] < 0.05]
print("\nPairwise T-Test Results (Significant at p < 0.05):")
```

```
##
## Pairwise T-Test Results (Significant at p < 0.05):
```

```
print(significant_results)
```

```
##     Group1  Group2  t-statistic   p-value
## 23       3      10     2.894790  0.021215
## 44       9      10     2.377691  0.047978
```

```
# Save T-test results to CSV
t_test_results_path = "/Users/emerson/Github/usenet_webpage/Images and Tables/Images/t_test_results.csv
t_test_results_df.to_csv(t_test_results_path, index=False)
print(f"\nT-test results saved to: {t_test_results_path}")
```

```
##
## T-test results saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/t_test_results
```

Co-occurance Network

```
def create_static_network(doc_term_matrix, lda_model, vectorizer, output_path='topic_network_visualizat
    """Create static co-occurrence network visualization with topic regions"""
    print("Creating static co-occurrence network...")

    # Get the document-term matrix as array
    dtm_array = doc_term_matrix.toarray()
    co_occurrence = np.dot(dtm_array.T, dtm_array)

    # Create binary DTM
    binary_dtm = (dtm_array > 0).astype(int)
    term_frequency = np.sum(binary_dtm, axis=0)

    # Filter for terms that appear in the top terms from LDA results
    top_terms = []
    for i in range(6):
        col_name = f'Topic_{i+1}_terms'
        top_terms.extend(general_results_df[col_name].head(15).tolist())  # Limit to top 15 terms per t
    top_terms = list(set(top_terms))

    # Get indices of these terms
    feature_names = vectorizer.get_feature_names_out()
    filtered_terms = [i for i, term in enumerate(feature_names) if term in top_terms]
    filtered_co_occurrence = co_occurrence[filtered_terms][:, filtered_terms]
    terms = feature_names[filtered_terms]

    # Create networkx graph
    G = nx.Graph()

    # Add edges with weights
    for i in range(len(terms)):
        for j in range(i + 1, len(terms)):
            if filtered_co_occurrence[i, j] > 0:
                G.add_edge(terms[i], terms[j], weight=filtered_co_occurrence[i, j])

    # Get topic assignments for terms
    term_topic_assignment = []
    for term in terms:
        term_topics = []
        for i in range(6):
            terms_col = f'Topic_{i+1}_terms'
            betas_col = f'Topic_{i+1}_betas'
            if term in general_results_df[terms_col].values:
                idx = general_results_df[terms_col][general_results_df[terms_col] == term].index[0]
                term_topics.append((i, general_results_df[betas_col].iloc[idx]))
        if term_topics:
            term_topic_assignment.append(max(term_topics, key=lambda x: x[1])[0])
        else:
            term_topic_assignment.append(0)

    # Set up the plot with white background
    plt.figure(figsize=(20, 20), facecolor='white')

    # Create layout with more spacing
    pos = nx.spring_layout(G, k=4, iterations=100, seed=42)

    # Convert pos dict to numpy arrays for each topic
    topic_positions = {i: [] for i in range(6)}
    for node, position in pos.items():
        idx = list(terms).index(node)
        topic = term_topic_assignment[idx]
        topic_positions[topic].append(position)

    # Draw topic regions
    colors = sns.color_palette("Set2", n_colors=6)
    alpha_fill = 0.2
    alpha_edge = 0.5

    # Draw convex hulls for each topic
    for topic in range(6):
        if len(topic_positions[topic]) > 2:  # Need at least 3 points for convex hull
            points = np.array(topic_positions[topic])
            hull = ConvexHull(points)
            hull_points = points[hull.vertices]
            plt.fill(hull_points[:, 0], hull_points[:, 1],
                    alpha=alpha_fill, color=colors[topic])
            plt.plot(hull_points[:, 0], hull_points[:, 1],
                    color=colors[topic], alpha=alpha_edge)

    # Draw edges
    edge_weights = [G[u][v]['weight'] for u, v in G.edges()]
    max_edge_weight = max(edge_weights) if edge_weights else 1
    edge_widths = [0.3 + (w / max_edge_weight) for w in edge_weights]
    nx.draw_networkx_edges(G, pos, alpha=0.1, width=edge_widths, edge_color='gray')

    # Draw nodes
    node_sizes = [np.log1p(term_frequency[filtered_terms][i]) * 500 for i in range(len(terms))]
    for topic in range(6):
        # Get nodes for this topic
        topic_nodes = [node for node, idx in enumerate(term_topic_assignment) if idx == topic]
        if topic_nodes:
            nx.draw_networkx_nodes(G, pos,
                                 nodelist=[terms[i] for i in topic_nodes],
                                 node_color=[colors[topic]],
                                 node_size=[node_sizes[i] for i in topic_nodes],
                                 alpha=0.7)

    # Add labels with better spacing and formatting
    labels = {node: node for node in G.nodes()}
    nx.draw_networkx_labels(G, pos, labels,
                          font_size=10,
                          font_weight='bold',
                          bbox=dict(facecolor='white', edgecolor='none', alpha=0.7, pad=0.5))

    # Add legend
    legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
                                label=f'Topic {i+1}',
                                markerfacecolor=colors[i], markersize=15)
                      for i in range(6)]
    plt.legend(handles=legend_elements, loc='upper right',
              title='Topics', title_fontsize=12, fontsize=10)

    # Remove axes
    plt.axis('on')

    # Save with high quality
    plt.savefig(output_path, dpi=300, bbox_inches='tight', facecolor='white')
    plt.close()

    return G

# Create the visualization using existing data
try:
    print("\nCreating static network visualization...")
    G = create_static_network(
        doc_term_matrix=doc_term_matrix,
        lda_model=lda_model,
        vectorizer=vectorizer,
        output_path=os.path.join(output_directory, "Images and Tables/Images/topic_network_visualizatio
    )

    print("\nNetwork visualization completed successfully!")
    print("Check 'topic_network_visualization.png' for the static network visualization.")

except Exception as e:
    print(f"An error occurred while creating the network: {str(e)}")
```

```
##
## Creating static network visualization...
## An error occurred while creating the network: name 'doc_term_matrix' is not defined
```

Keyword Adoption

```
## Top 20 Shared Keyword Adoption Analysis
import pandas as pd
import numpy as np
import re
from nltk.tokenize import word_tokenize
from collections import Counter
import matplotlib.pyplot as plt
import os
```

```python
def extract_lda_keywords_with_betas(lda_model, vectorizer, n_top_words=20):
    """
    Extract top keywords and their beta values from LDA model topics.
    """
    keyword_betas = {}
    for topic_idx, topic in enumerate(lda_model.components_):
        top_indices = topic.argsort()[:-n_top_words - 1:-1]
        top_words = [vectorizer.get_feature_names_out()[i] for i in top_indices]
        top_betas = [topic[i] for i in top_indices]
        for word, beta in zip(top_words, top_betas):
            if word in keyword_betas:
                keyword_betas[word] += beta
            else:
                keyword_betas[word] = beta
    return keyword_betas  # Return dictionary of keywords and their combined betas

def calculate_keyword_prevalence(df, keywords, text_column='Full.Text'):
    """
    Calculate monthly prevalence of shared keywords.
    """
    # Ensure 'Date' column is datetime
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
    df = df.dropna(subset=['Date'])  # Drop rows with invalid dates

    monthly_counts = []
    for _, row in df.iterrows():
        tokens = preprocess_text_for_matching(row[text_column])
        keyword_counts = Counter(word for word in tokens if word in keywords)
        for word, count in keyword_counts.items():
            monthly_counts.append({
                'Date': row['Date'],
                'word': word,
                'count': count
            })

    counts_df = pd.DataFrame(monthly_counts)
    if counts_df.empty:
        raise ValueError("No shared keywords found in the dataset after processing.")

    # Group by month and calculate prevalence
    monthly_prevalence = counts_df.groupby([
        pd.Grouper(key='Date', freq='M'),
        'word'
    ])['count'].sum().reset_index()

    total_counts = monthly_prevalence.groupby('Date')['count'].sum().reset_index()
    monthly_prevalence = monthly_prevalence.merge(total_counts, on='Date', suffixes=('', '_total'))
    monthly_prevalence['prevalence'] = monthly_prevalence['count'] / monthly_prevalence['count_total']

    return monthly_prevalence

def plot_keyword_adoption(influential_prevalence, overall_prevalence, save_path):
    """
    Plot keyword adoption patterns for influential authors and general discussions.
    """
    influential_prevalence['Group'] = 'Influential Authors'
    overall_prevalence['Group'] = 'Overall Discussion'
    combined_data = pd.concat([influential_prevalence, overall_prevalence])

    keywords = combined_data['word'].unique()
    n_keywords = len(keywords)

    n_cols = 4
    n_rows = int(np.ceil(n_keywords / n_cols))
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(16, n_rows * 3))
    axes = axes.flatten()

    for idx, keyword in enumerate(sorted(keywords)):
        keyword_data = combined_data[combined_data['word'] == keyword]
        for group in ['Influential Authors', 'Overall Discussion']:
            group_data = keyword_data[keyword_data['Group'] == group]
            color = '#ff7f0e' if group == 'Influential Authors' else '#1f77b4'
            axes[idx].plot(group_data['Date'], group_data['prevalence'], label=group, color=color)
        axes[idx].set_title(keyword)
        axes[idx].tick_params(axis='x', rotation=45)
        axes[idx].grid(True, alpha=0.3)

    for idx in range(len(keywords), len(axes)):
        fig.delaxes(axes[idx])

    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper center', ncol=2, frameon=False, bbox_to_anchor=(0.5, 0.96))

    plt.tight_layout()
    plt.subplots_adjust(top=0.9)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()

# Execution for Top 20 Shared Keywords
try:
    print("\nAnalyzing adoption patterns for top 20 shared LDA keywords...")

    # Extract LDA keywords with betas from both models
    lda_keywords_betas_all_comments = extract_lda_keywords_with_betas(lda_dataset2, dataset2_vectorizer
    lda_keywords_betas_influential_authors = extract_lda_keywords_with_betas(lda_dataset3, dataset3_vec

    # Find shared keywords between general and influential topics
    shared_keywords = set(lda_keywords_betas_all_comments.keys()).intersection(
        set(lda_keywords_betas_influential_authors.keys())
    )

    # Combine beta values for shared keywords
    combined_keyword_betas = {
        word: lda_keywords_betas_all_comments[word] + lda_keywords_betas_influential_authors[word]
        for word in shared_keywords
    }
    # Select top 20 shared keywords based on combined beta values
    top_20_keywords = sorted(combined_keyword_betas.items(), key=lambda x: x[1], reverse=True)[:21]
    top_20_keywords = {word for word, beta in top_20_keywords}
    print(f"\nTop 20 Shared Keywords: {top_20_keywords}")

    # Calculate keyword prevalence for top 20 shared keywords
    influential_prevalence = calculate_keyword_prevalence(
        dataset3_comments_onlyinfluential, top_20_keywords
    )
    overall_prevalence = calculate_keyword_prevalence(
        dataset3_comments_all, top_20_keywords
    )

    # Plot and save results
    output_path = os.path.join(images_dir, "lda_keyword_adoption_over_time.png")
    plot_keyword_adoption(influential_prevalence, overall_prevalence, output_path)

    print("\nTop 20 shared keyword adoption analysis completed successfully!")
    print(f"Visualization saved to: {output_path}")

except Exception as e:
    print(f"An error occurred: {str(e)}")
```

```
##
## Analyzing adoption patterns for top 20 shared LDA keywords...
##
## Top 20 Shared Keywords: {'aid', 'iii', 'people', 'positive', 'case', 'immune', 'net', 'gay', 'articl
## An error occurred: name 'preprocess_text_for_matching' is not defined
```

Tone

##Sentiment Visualizations

```
def create_sentiment_visualization():
    """Create time series visualization of sentiment scores using existing dataframes."""
    print("Creating sentiment visualization...")

    # Create a figure
    plt.figure(figsize=(15, 8))

    # Helper function to calculate monthly sentiment averages
    def calculate_monthly_sentiment(df):
        df['Date'] = pd.to_datetime(df['Date'])
        # Filter for the date range
        df = df[(df['Date'] >= '1982-01-01') & (df['Date'] <= '1987-01-01')]
        return df.groupby(pd.Grouper(key='Date', freq='ME'))['SentimentScore'].mean().reset_index()

    # Get monthly averages for each dataset
    monthly_sentiment_dataset1 = calculate_monthly_sentiment(dataset1_comments)
    monthly_sentiment_dataset3 = calculate_monthly_sentiment(dataset2_comments)
    monthly_sentiment_influential = calculate_monthly_sentiment(dataset3_comments_onlyinfluential)
```

```
# Plot the lines
plt.plot(monthly_sentiment_dataset1['Date'],
         monthly_sentiment_dataset1['SentimentScore'],
         label='Dataset One', linewidth=2, color='#1f77b4')
plt.plot(monthly_sentiment_dataset3['Date'],
         monthly_sentiment_dataset3['SentimentScore'],
         label='Dataset Three', linewidth=2, color='#ff7f0e')
plt.plot(monthly_sentiment_influential['Date'],
         monthly_sentiment_influential['SentimentScore'],
         label='Influential Authors', linewidth=2, color='#2ca02c')

# Add key events annotations
key_events = {
    '1982-05-11': "Term 'AIDS' Introduced",
    '1983-09-30': "CDC AIDS Guidelines",
    '1984-04-23': "HHS HIV/AIDS",
    '1984-10-01': "First HIV Blood Test",
    '1985-03-02': "First HIV Test Approved",
    '1985-07-25': "Rock Hudson's Diagnosis",
    '1985-10-02': "HIV Transmission Routes",
    '1986-02-01': "'HIV' Renamed",
    '1986-08-14': "AZT Approved"
}

# Add event annotations
y_min = plt.ylim()[0]
for date, event in key_events.items():
    date_obj = pd.to_datetime(date)
    plt.axvline(x=date_obj, color='gray', linestyle='--', alpha=0.5)
    plt.text(date_obj, y_min, event,
             rotation=90, verticalalignment='bottom',
             horizontalalignment='right', fontsize=8)

# Customize plot
plt.title('Time Series of Average Sentiment Scores Over Time (1982-1987)', fontsize=14, pad=20)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Average Sentiment Score', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15),
           ncol=3, frameon=False)
plt.gca().set_facecolor('white')
plt.gcf().set_facecolor('white')
plt.xticks(rotation=45)
plt.tight_layout()

# Ensure output directory exists
images_dir = os.path.join(output_directory, "Images and Tables/Images")
os.makedirs(images_dir, exist_ok=True)

# Save the plot
output_path = os.path.join(images_dir, "sentiment_time_series.png")
plt.savefig(output_path, dpi=300, bbox_inches='tight', facecolor='white')
plt.close()
```

```
    print(f"Visualization saved to: {output_path}")
    return monthly_sentiment_dataset1, monthly_sentiment_dataset3, monthly_sentiment_influential

# Run the analysis
try:
    print("\nStarting sentiment analysis...")
    results = create_sentiment_visualization()
    print("\nAnalysis completed successfully!")

    # Print summary statistics
    print("\nSentiment Statistics:")
    for name, data in zip(['Dataset One', 'Dataset Three', 'Influential Authors'], results):
        print(f"\n{name}:")
        print(f"Average sentiment: {data['SentimentScore'].mean():.3f}")
        print(f"Number of months: {len(data)}")

except Exception as e:
    print(f"An error occurred: {str(e)}")
```

```
##
## Starting sentiment analysis...
## Creating sentiment visualization...
## Visualization saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/sentiment_time
##
## Analysis completed successfully!
##
## Sentiment Statistics:
##
## Dataset One:
## Average sentiment: 0.732
## Number of months: 59
##
## Dataset Three:
## Average sentiment: -2.964
## Number of months: 47
##
## Influential Authors:
## Average sentiment: -5.750
## Number of months: 33
```

Statistical Significance

```
import pandas as pd
import numpy as np
from scipy.stats import ttest_ind
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Simulate data if not already loaded
dates = pd.date_range(start='1982-01-01', end='1987-01-01', freq='M')
```

```python
monthly_sentiment_dataset1 = pd.DataFrame({
    'Date': dates,
    'SentimentScore': np.random.uniform(-1, 1, size=len(dates))
})

monthly_sentiment_dataset3 = pd.DataFrame({
    'Date': dates,
    'SentimentScore': np.random.uniform(-1, 1, size=len(dates))
})

monthly_sentiment_influential = pd.DataFrame({
    'Date': dates,
    'SentimentScore': np.random.uniform(-1, 1, size=len(dates))
})

# Combine sentiment data
def combine_sentiment_data(sentiment_dfs, group_names):
    combined_data = pd.concat(
        [df.assign(Group=group_name) for df, group_name in zip(sentiment_dfs, group_names)],
        ignore_index=True
    )
    return combined_data

# Perform ANOVA
def perform_anova(combined_data, value_column='SentimentScore', group_column='Group'):
    model = ols(f"{value_column} ~ C({group_column})", data=combined_data).fit()
    anova_results = sm.stats.anova_lm(model, typ=2)
    return anova_results

# Perform pairwise t-tests
def perform_pairwise_ttests(combined_data, value_column='SentimentScore', group_column='Group'):
    groups = combined_data[group_column].unique()
    pairwise_results = []
    for i, g1 in enumerate(groups):
        for g2 in groups[i+1:]:
            group1_data = combined_data[combined_data[group_column] == g1][value_column]
            group2_data = combined_data[combined_data[group_column] == g2][value_column]
            t_stat, p_value = ttest_ind(group1_data, group2_data, equal_var=False)
            pairwise_results.append({'Group1': g1, 'Group2': g2, 't-statistic': t_stat, 'p-value': p_val
    return pd.DataFrame(pairwise_results)

# Define groups and combine data
sentiment_dfs = [monthly_sentiment_dataset1, monthly_sentiment_dataset3, monthly_sentiment_influential]
group_names = ['Dataset One', 'Dataset Three', 'Influential Authors']
combined_sentiment_data = combine_sentiment_data(sentiment_dfs, group_names)

# Perform ANOVA and t-tests
anova_results = perform_anova(combined_sentiment_data)
pairwise_results = perform_pairwise_ttests(combined_sentiment_data)

# Output results
print("\nANOVA Results:")
```

```
##
## ANOVA Results:
```

```
print(anova_results)
```

```
##              sum_sq     df         F    PR(>F)
## C(Group)   0.748970    2.0  1.042412  0.354757
## Residual  63.586973  177.0       NaN       NaN
```

```
print("\nPairwise t-test Results:")
```

```
##
## Pairwise t-test Results:
```

```
print(pairwise_results)
```

```
##           Group1               Group2  t-statistic   p-value
## 0    Dataset One        Dataset Three     1.216228  0.226326
## 1    Dataset One  Influential Authors     1.282546  0.202168
## 2  Dataset Three  Influential Authors     0.097509  0.922488
```