Paper Visualization

Emerson Johnston

```
import gc
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
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.gridspec import GridSpec
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
import statsmodels.api as sm
from statsmodels.formula.api import ols
from scipy.stats import ttest_ind, skew, kurtosis
from nltk.stem import WordNetLemmatizer
from scipy.sparse import SparseEfficiencyWarning
import warnings

# 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_only
influential_authors = pd.read_csv(os.path.join(output_directory, "CSV Files/influential_authors.csv"))
```

Initial Visualizations

Cross-Dataset Content Focus Analysis

```
topic_categories = {
    'medical': [
        'virus', 'immune', 'infection', 'symptom', 'treatment', 'disease',
        'patient', 'doctor', 'hospital', 'drug', 'clinical', 'diagnosis',
        'vaccine', 'transmission', 'blood', 'test', 'positive', 'negative',
        'medication', 'therapy', 'cure', 'health', 'medical', 'medicine',
        'syndrome', 'cell', 'viral', 'antibody', 'immune system', 'insurance'
    ],
    'social': [
        'community', 'support', 'family', 'friend', 'relationship', 'stigma',
        'discrimination', 'education', 'awareness', 'fear', 'anxiety', 'help',
        'care', 'society', 'public', 'impact', 'life', 'work', 'social',
        'personal', 'experience', 'gay', 'lesbian', 'sexuality', 'identity',
        'support group', 'counseling', 'lifestyle', 'isolation'
    ],
    'political': [
        'policy', 'government', 'funding', 'law', 'legislation', 'right',
        'activist', 'advocacy', 'campaign', 'research', 'organization',
        'regulation', 'public health', 'cost', 'access', 'insurance',
        'initiative', 'program', 'reform', 'budget', 'protest', 'rights', 'reagan'
    ],
    'scientific': [
        'research', 'study', 'data', 'evidence', 'trial', 'experiment',
        'laboratory', 'clinical', 'result', 'hypothesis', 'analysis',
        'scientific', 'researcher', 'publication', 'paper', 'journal',
        'methodology', 'statistical'
    ],
    'emotional': [
        'fear', 'hope', 'worry', 'anxiety', 'depression', 'anger',
        'frustration', 'support', 'care', 'love', 'concern', 'stress',
        'courage', 'strength', 'emotional', 'feeling', 'scared', 'afraid',
        'hopeful', 'grateful'
    ]
}

def calculate_topic_scores(text):
    """Calculate topic scores for a single text"""
    if pd.isna(text):
        return {cat: 0.0 for cat in topic_categories.keys()}

    text = str(text).lower()
    scores = {}
```

```
    for category, keywords in topic_categories.items():
        score = sum(1 for keyword in keywords if keyword in text)
        scores[category] = float(score)

    total = sum(scores.values())
    if total > 0:
        scores = {k: v/total for k, v in scores.items()}

    return scores

def analyze_dataset(df, name):
    """Analyze content for a dataset"""
    print(f"Analyzing {name}...")

    topic_scores = []
    for _, row in df.iterrows():
        scores = calculate_topic_scores(row['Full.Text'])
        scores['Date'] = pd.to_datetime(row['Date'])
        scores['Dataset'] = name
        scores['Newsgroup'] = row['newsgroup']
        topic_scores.append(scores)

    return pd.DataFrame(topic_scores)

def create_enhanced_visualizations(results_dict, output_dir):
    """Create detailed visualizations"""
    print("Creating enhanced visualizations...")

    sns.set_style("whitegrid")
    sns.set_context("talk")
    colors = sns.color_palette("Set2", len(topic_categories))

    fig = plt.figure(figsize=(15, 10))
    gs = GridSpec(2, 1, height_ratios=[2, 1])

    ax1 = fig.add_subplot(gs[0])
    topic_means = {name: df[list(topic_categories.keys())].mean()
                   for name, df in results_dict.items()}
    topic_comparison = pd.DataFrame(topic_means)  # Notice the change here

    n_topics = len(topic_categories)
    n_datasets = len(results_dict)
    bar_width = 0.8 / n_datasets
    positions = np.arange(n_topics)

    for i, (dataset_name, values) in enumerate(topic_means.items()):
        x_pos = positions + (i * bar_width)
        rects = ax1.bar(x_pos,
                        values,
                        bar_width,
                        label=dataset_name,
                        alpha=0.8)
        # Add value labels on top of bars
```

```
        for rect in rects:
            height = rect.get_height()
            ax1.text(rect.get_x() + rect.get_width()/2., height,
                    f'{height:.2f}',
                    ha='center', va='bottom',
                    rotation=90)

    ax1.set_ylabel('Proportion', fontsize=12)
    ax1.set_title('Topic Distribution Comparison Across Datasets', fontsize=14, pad=20)
    ax1.set_xticks(positions + bar_width * (n_datasets-1)/2)
    ax1.set_xticklabels(topic_categories.keys(), rotation=45, ha='right')
    ax1.legend(title='Datasets', bbox_to_anchor=(1.05, 1))

    ax2 = fig.add_subplot(gs[1])
    baseline = topic_comparison.iloc[:,0]  # First dataset as baseline

    for i in range(1, topic_comparison.shape[1]):
        relative_diff = (topic_comparison.iloc[:,i] - baseline) / baseline * 100
        ax2.plot(range(len(topic_categories)), relative_diff,
                marker='o', label=f'{topic_comparison.columns[i]} vs {topic_comparison.columns[0]}')

    ax2.axhline(y=0, color='k', linestyle='--', alpha=0.3)
    ax2.set_xticks(range(len(topic_categories)))
    ax2.set_xticklabels(topic_categories.keys(), rotation=45, ha='right')
    ax2.set_ylabel('% Difference from General Usenet')
    ax2.legend(bbox_to_anchor=(1.05, 1))

    plt.tight_layout()
    plt.savefig(f'{output_dir}/enhanced_topic_distribution.png', dpi=300, bbox_inches='tight')
    plt.close()

    fig, axes = plt.subplots(3, 1, figsize=(15, 15))
    fig.suptitle('Topic Evolution Over Time (Pre-1987)', fontsize=16, y=1.00)

    key_events = {
        '1982-05-11': "Term 'AIDS' Introduced",
        '1983-09-30': "CDC Guidelines",
        '1984-04-23': "HIV Identified",
        '1985-03-02': "First Test Approved",
        '1985-07-25': "Rock Hudson",
        '1986-02-01': "'HIV' Named"
    }

    for idx, (dataset_name, df) in enumerate(results_dict.items()):
        df = df[df['Date'] < '1987-01-01'].copy()

        df_monthly = df.groupby(pd.Grouper(key='Date', freq='M'))[list(topic_categories.keys())].mean()

        for topic_idx, topic in enumerate(topic_categories.keys()):
            if topic in df_monthly.columns:  # Check if topic exists in the data
                axes[idx].plot(df_monthly.index, df_monthly[topic],
                               label=topic, color=colors[topic_idx],
                               linewidth=2, marker='o', markersize=4)
```

```
        axes[idx].set_title(f'{dataset_name}', fontsize=12)
        axes[idx].set_ylabel('Topic Proportion')
        axes[idx].grid(True, alpha=0.3)
        axes[idx].xaxis.set_major_locator(mdates.MonthLocator(interval=3))
        axes[idx].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))

        for date, event in key_events.items():
            event_date = pd.to_datetime(date)
            if event_date in df_monthly.index:
                axes[idx].axvline(x=event_date, color='gray',
                                  linestyle='--', alpha=0.3)
                axes[idx].text(event_date, axes[idx].get_ylim()[1],
                               event, rotation=90, va='top', fontsize=8)

    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='center right',
               bbox_to_anchor=(0.98, 0.5), title='Topics')

    plt.tight_layout()
    plt.savefig(f'{output_dir}/enhanced_temporal_evolution.png', dpi=300, bbox_inches='tight')
    plt.close()

    fig = plt.figure(figsize=(20, 8))
    gs = GridSpec(1, 3)

    for idx, (dataset_name, df) in enumerate(results_dict.items()):
        ax = fig.add_subplot(gs[idx])

        if len(df) > 0:
            pivot_data = df.groupby('Newsgroup')[list(topic_categories.keys())].mean()

            cmap = sns.color_palette("YlOrRd", as_cmap=True)
            sns.heatmap(pivot_data, annot=True, fmt='.2f',
                        cmap=cmap, ax=ax, cbar=True)

            ax.set_title(f'{dataset_name}', fontsize=12)
            plt.setp(ax.get_xticklabels(), rotation=45, ha='right')
        else:
            ax.text(0.5, 0.5, 'No data available',
                    ha='center', va='center')
            ax.set_title(f'{dataset_name}', fontsize=12)

    plt.suptitle('Topic Distribution by Newsgroup', fontsize=14, y=1.05)
    plt.tight_layout()
    plt.savefig(f'{output_dir}/enhanced_newsgroup_comparison.png', dpi=300, bbox_inches='tight')
    plt.close()

    fig, axes = plt.subplots(1, 3, figsize=(20, 6))
    fig.suptitle('Topic Correlations Across Datasets', fontsize=14)

    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    for idx, (dataset_name, df) in enumerate(results_dict.items()):
```

```
        if len(df) > 0:
            topic_corr = df[list(topic_categories.keys())].corr()
            sns.heatmap(topic_corr, annot=True, fmt='.2f',
                        cmap=cmap, vmin=-1, vmax=1,
                        center=0, ax=axes[idx])
        axes[idx].set_title(dataset_name)

    plt.tight_layout()
    plt.savefig(f'{output_dir}/topic_correlations.png', dpi=300, bbox_inches='tight')
    plt.close()

def print_summary_statistics(results_dict):
    """Print detailed summary statistics for each dataset"""
    print("\nDetailed Summary Statistics:")

    for name, df in results_dict.items():
        print(f"\n{name} Analysis:")
        print("-" * 50)

        print(f"Total posts: {len(df)}")
        print(f"Date range: {df['Date'].min().date()} to {df['Date'].max().date()}")
        print(f"Number of unique newsgroups: {df['Newsgroup'].nunique()}")

        print("\nTopic Distributions:")
        topic_means = df[list(topic_categories.keys())].mean()
        topic_stds = df[list(topic_categories.keys())].std()

        for topic in topic_categories.keys():
            print(f"{topic}:")
            print(f"  Mean: {topic_means[topic]:.3f}")
            print(f"  Std:  {topic_stds[topic]:.3f}")

        print("\nMost prevalent topic pairs:")
        topic_corr = df[list(topic_categories.keys())].corr()
        np.fill_diagonal(topic_corr.values, 0)
        top_pairs = np.unravel_index(np.argsort(topic_corr.values, axis=None)[-3:], topic_corr.shape)
        for i, j in zip(top_pairs[0], top_pairs[1]):
            if i < j:
                print(f"{topic_corr.index[i]} - {topic_corr.columns[j]}: {topic_corr.iloc[i, j]:.3f}")

def main():
    """Main analysis function"""
    results = {
        'General Usenet': analyze_dataset(dataset1_comments, "General Usenet"),
        'AIDS Discussions': analyze_dataset(dataset2_comments, "AIDS Discussions"),
        'Influential Authors': analyze_dataset(dataset3_comments_onlyinfluential, "Influential Authors"
    }

    create_enhanced_visualizations(results, "/Users/emerson/Github/usenet_webpage/Images and Tables/Ima

    print_summary_statistics(results)

    return results
```

```python
if __name__ == "__main__":
    results = main()
```

```
## Analyzing General Usenet...
## Analyzing AIDS Discussions...
## Analyzing Influential Authors...
## Creating enhanced visualizations...
## <string>:77: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
## <string>:77: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
## <string>:77: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
##
## Detailed Summary Statistics:
##
## General Usenet Analysis:
## --------------------------------------------------
## Total posts: 43891
## Date range: 1980-05-16 to 1997-05-23
## Number of unique newsgroups: 6
##
## Topic Distributions:
## medical:
##   Mean: 0.094
##   Std:  0.206
## social:
##   Mean: 0.318
##   Std:  0.320
## political:
##   Mean: 0.187
##   Std:  0.274
## scientific:
##   Mean: 0.079
##   Std:  0.179
## emotional:
##   Mean: 0.143
##   Std:  0.220
##
## Most prevalent topic pairs:
##
## AIDS Discussions Analysis:
## --------------------------------------------------
## Total posts: 348
## Date range: 1982-11-10 to 1986-09-25
## Number of unique newsgroups: 5
##
```

```
## Topic Distributions:
## medical:
##   Mean: 0.418
##   Std:  0.299
## social:
##   Mean: 0.175
##   Std:  0.186
## political:
##   Mean: 0.102
##   Std:  0.175
## scientific:
##   Mean: 0.097
##   Std:  0.125
## emotional:
##   Mean: 0.099
##   Std:  0.171
##
## Most prevalent topic pairs:
## political - scientific: 0.019
## social - emotional: 0.117
##
## Influential Authors Analysis:
## --------------------------------------------------
## Total posts: 158
## Date range: 1984-01-18 to 1986-09-21
## Number of unique newsgroups: 3
##
## Topic Distributions:
## medical:
##   Mean: 0.488
##   Std:  0.279
## social:
##   Mean: 0.169
##   Std:  0.148
## political:
##   Mean: 0.078
##   Std:  0.139
## scientific:
##   Mean: 0.102
##   Std:  0.116
## emotional:
##   Mean: 0.100
##   Std:  0.161
##
## Most prevalent topic pairs:
## political - scientific: 0.070
## social - emotional: 0.192
```

Distribution of influential authors' comments across newsgroups

```
def analyze_influential_distribution(dataset1_comments, influential_authors):
    """Analyze how influential authors participated across different newsgroups"""
```

```
    influential_comments = dataset1_comments[dataset1_comments['Author'].isin(influential_authors['Autho

    author_newsgroup_counts = influential_comments.groupby(['Author', 'newsgroup']).size().reset_index(n

    pivot_data = author_newsgroup_counts.pivot(index='Author', columns='newsgroup', values='comment_cou
    pivot_data = pivot_data.fillna(0)

    pivot_data_pct = pivot_data.div(pivot_data.sum(axis=1), axis=0) * 100

    total_comments = pivot_data.sum(axis=1)
    pivot_data_pct = pivot_data_pct.loc[total_comments.sort_values(ascending=False).index]

    plt.figure(figsize=(20, 20))

    sns.heatmap(pivot_data_pct,
                annot=True,
                fmt='.1f',
                cmap='YlOrRd',
                cbar_kws={'label': 'Percentage of Author\'s Comments (%)'},
                vmin=0,
                vmax=100)

    plt.title('Distribution of Influential Authors\' Comments Across Newsgroups\n(General Usenet Dataset
              pad=20, fontsize=14)
    plt.ylabel('Author (Sorted by Total Comments)')
    plt.xlabel('Newsgroup')

    plt.xticks(rotation=45, ha='right')
    plt.yticks(rotation=0)

    plt.tight_layout()
    plt.savefig('/Users/emerson/Github/usenet_webpage/Images and Tables/Images/influential_distribution
                dpi=300, bbox_inches='tight')
    plt.close()

    summary_stats = {
        'Total Authors': len(pivot_data),
        'Most Active Newsgroup': pivot_data.sum().idxmax(),
        'Least Active Newsgroup': pivot_data.sum().idxmin(),
        'Most Prolific Author': total_comments.idxmax(),
        'Average Comments per Author': total_comments.mean(),
        'Median Comments per Author': total_comments.median(),
        'Authors with Cross-posting': (pivot_data > 0).sum(axis=1).value_counts()
    }

    return pivot_data, pivot_data_pct, summary_stats

pivot_data, pivot_data_pct, summary_stats = analyze_influential_distribution(dataset1_comments, influent

print("\nSummary Statistics:")
```

```
##
## Summary Statistics:
```

```
print("-" * 50)
```

```
## --------------------------------------------------
```

```
print(f"Total Influential Authors: {summary_stats['Total Authors']}")
```

```
## Total Influential Authors: 20
```

```
print(f"Most Active Newsgroup: {summary_stats['Most Active Newsgroup']}")
```

```
## Most Active Newsgroup: netmed
```

```
print(f"Least Active Newsgroup: {summary_stats['Least Active Newsgroup']}")
```

```
## Least Active Newsgroup: netnews
```

```
print(f"Most Prolific Author: {summary_stats['Most Prolific Author']}")
```

```
## Most Prolific Author: Steve Dyer
```

```
print(f"Average Comments per Author: {summary_stats['Average Comments per Author']:.1f}")
```

```
## Average Comments per Author: 76.3
```

```
print(f"Median Comments per Author: {summary_stats['Median Comments per Author']:.1f}")
```

```
## Median Comments per Author: 22.0
```

```
print("\nNumber of Authors by Newsgroups Posted In:")
```

```
##
## Number of Authors by Newsgroups Posted In:
```

```
print(summary_stats['Authors with Cross-posting'].sort_index().to_string())
```

```
## 2    4
## 3    2
## 4    6
## 5    4
## 6    4
```

Author Impact

Topic Modeling

```
def preprocess_text(text):
    stop_words = set(stopwords.words("english"))
    lemmatizer = WordNetLemmatizer()
    text = re.sub(r"\W+", " ", text)
    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)

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)

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

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

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:-1]]
        topics.append((f"Topic {topic_idx + 1}", top_words))
    return topics

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)
```

```
topics_2 = extract_topics(lda_dataset2, dataset2_vectorizer)

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)
```

```
topics_3 = extract_topics(lda_dataset3, dataset3_vectorizer)

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

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

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

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

```
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:-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))

    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_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_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

```
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...")

    general_results = pd.read_html(general_results_path)[0]
```

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

topic_similarities = []

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
        })

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

similarity_matrix = similarity_matrix.iloc[::-1]

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

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}")

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_i

    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)
```

```
    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']]
    )

    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')

    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

    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')

    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}")

    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("\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

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_heat
## Bar plot saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_similarity_bar
## Comparison data saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_similar
##
## 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

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

comparison_csv_path = "/Users/emerson/Github/usenet_webpage/Images and Tables/Images/topic_similarity_c
comparison_data = pd.read_csv(comparison_csv_path)

print("Preparing data for regression analysis...")
```

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

```
comparison_data['General_Topic'] = comparison_data['General_Topic'].astype('category')
comparison_data['Influential_Topic'] = comparison_data['Influential_Topic'].astype('category')

X = pd.get_dummies(comparison_data[['General_Topic', 'Influential_Topic']], drop_first=True)

X = sm.add_constant(X)

X = X.applymap(lambda value: int(value) if isinstance(value, bool) else value)
X = X.apply(pd.to_numeric, errors='coerce')
```

```
y = pd.to_numeric(comparison_data['Similarity'], errors='coerce')

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]

print("\nCleaned Predictors (X):")
```

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

```
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
```

```
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:                        23:19:17   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.
```

```
print("\nPerforming pairwise T-tests between General Topics...")
```

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

```
grouped_data = comparison_data.groupby('General_Topic')['Similarity']

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

t_test_results_df = pd.DataFrame(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
```

```
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_visualizati
    """Create static co-occurrence network visualization with topic regions."""
    print("Creating static co-occurrence network...")

    dtm_array = doc_term_matrix.toarray()
    co_occurrence = np.dot(dtm_array.T, dtm_array)

    term_frequency = np.sum(dtm_array, axis=0)

    feature_names = vectorizer.get_feature_names_out()

    top_terms = []
    for topic_idx, topic in enumerate(lda_model.components_):
        top_terms.extend([feature_names[i] for i in topic.argsort()[-15:]])
    top_terms = list(set(top_terms))

    filtered_indices = [i for i, term in enumerate(feature_names) if term in top_terms]
    filtered_co_occurrence = co_occurrence[filtered_indices][:, filtered_indices]
    filtered_terms = [feature_names[i] for i in filtered_indices]

    G = nx.Graph()

    for i in range(len(filtered_terms)):
        for j in range(i + 1, len(filtered_terms)):
            weight = filtered_co_occurrence[i, j]
            if weight > 0:
                G.add_edge(filtered_terms[i], filtered_terms[j], weight=weight)
```

```
    plt.figure(figsize=(20, 20), facecolor='white')

    pos = nx.spring_layout(G, seed=42)

    node_sizes = [np.log1p(term_frequency[filtered_indices][i]) * 500 for i in range(len(filtered_terms)

    term_topic_assignment = []
    for term in filtered_terms:
        term_topics = []
        for topic_idx, topic in enumerate(lda_model.components_):
            if term in [feature_names[i] for i in topic.argsort()[-15:]]:
                term_topics.append((topic_idx, topic[feature_names.tolist().index(term)]))
        if term_topics:
            term_topic_assignment.append(max(term_topics, key=lambda x: x[1])[0])
        else:
            term_topic_assignment.append(0)

    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=0.2, width=edge_widths, edge_color='gray')

    colors = sns.color_palette("Set2", len(lda_model.components_))
    for topic in range(len(lda_model.components_)):
        topic_nodes = [filtered_terms[i] for i, t in enumerate(term_topic_assignment) if t == topic]
        nx.draw_networkx_nodes(
            G, pos, nodelist=topic_nodes,
            node_color=[colors[topic]] * len(topic_nodes),
            node_size=[node_sizes[i] for i, t in enumerate(term_topic_assignment) if t == topic],
            alpha=0.8
        )

    nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold', alpha=0.9)

    legend_elements = [
        plt.Line2D([0], [0], marker='o', color='w', label=f'Topic {i+1}',
                   markerfacecolor=colors[i], markersize=15)
        for i in range(len(lda_model.components_))
    ]
    plt.legend(handles=legend_elements, loc='upper right', title='Topics')

    plt.title("Topic Co-occurrence Network", fontsize=16)
    plt.axis('off')
    plt.savefig(output_path, dpi=300, bbox_inches='tight', facecolor='white')
    plt.close()
    print(f"Co-occurrence network saved to {output_path}.")

    return G

output_path = os.path.join(output_directory, "Images and Tables/Images/cooccurrence_network.png")
try:
    print("Generating the co-occurrence network...")
```

```
    G = create_static_network(
        doc_term_matrix=dataset2_matrix,
        lda_model=lda_dataset2,
        vectorizer=dataset2_vectorizer,
        output_path=output_path
    )
    print("Co-occurrence network generation complete!")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
## Generating the co-occurrence network...
## Creating static co-occurrence network...
## Co-occurrence network saved to /Users/emerson/Github/usenet_webpage/Images and Tables/Images/cooccur
## Co-occurrence network generation complete!
```

Keyword Adoption

```
def preprocess_text_for_matching(text):
    """
    Preprocess text for keyword matching: tokenize, lowercase, and clean.
    """
    if pd.isna(text) or not isinstance(text, str):
        return []
    text = re.sub(r"\W+", " ", text)
    tokens = word_tokenize(text.lower())
    return tokens


def extract_lda_keywords_with_betas(lda_model, vectorizer, n_top_words=20):
    """
    Extract top keywords and their beta values from LDA model topics.
    """
    custom_stop_words = {"iii"}

    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 not in custom_stop_words:
                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.
```

```
    """
    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.")

    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()


try:
    print("\nAnalyzing adoption patterns for top 20 shared LDA keywords...")

    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

    shared_keywords = set(lda_keywords_betas_all_comments.keys()).intersection(
        set(lda_keywords_betas_influential_authors.keys())
    )

    combined_keyword_betas = {
        word: lda_keywords_betas_all_comments[word] + lda_keywords_betas_influential_authors[word]
        for word in shared_keywords
    }

    top_20_keywords = sorted(combined_keyword_betas.items(), key=lambda x: x[1], reverse=True)[:20]
    top_20_keywords = {word for word, beta in top_20_keywords}
    print(f"\nTop 20 Shared Keywords: {top_20_keywords}")

    influential_prevalence = calculate_keyword_prevalence(
        dataset3_comments_onlyinfluential, top_20_keywords
    )
    overall_prevalence = calculate_keyword_prevalence(
        dataset3_comments_all, top_20_keywords
    )

    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: {'positive', 'year', 'net', 'case', 'patient', 'test', 'htlv', 'cell', 'blood
##
## Top 20 shared keyword adoption analysis completed successfully!
## Visualization saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/lda_keyword_ad
```

Influence Propogation Network

```python
def create_influence_network(dataset1_comments, influential_authors, output_path):
    """Create and visualize influence network based on thread interactions."""

    dataset1_comments['Date'] = pd.to_datetime(dataset1_comments['Date'], errors='coerce')
    dataset1_comments = dataset1_comments.dropna(subset=['Date'])

    influential_authors_set = set(influential_authors['Author'])
    dataset1_comments = dataset1_comments[dataset1_comments['Author'].isin(influential_authors_set)]

    interactions = []
    for thread_id, group in dataset1_comments.groupby('TH_ID'):
        group = group.sort_values('Date')
        authors = group['Author'].tolist()
        dates = group['Date'].tolist()

        for i in range(1, len(authors)):
            source = authors[i - 1]
            target = authors[i]
            response_time = (dates[i] - dates[i - 1]).total_seconds()
            interactions.append((source, target, response_time))

    G = nx.DiGraph()

    for author in influential_authors_set:
        comment_count = len(dataset1_comments[dataset1_comments['Author'] == author])
        G.add_node(author, size=np.log2(comment_count + 1) * 100)

    edge_weights = {}
    response_times = {}
    for source, target, response_time in interactions:
        if (source, target) in edge_weights:
            edge_weights[(source, target)] += 1
            response_times[(source, target)].append(response_time)
        else:
            edge_weights[(source, target)] = 1
            response_times[(source, target)] = [response_time]

    for (source, target), weight in edge_weights.items():
        avg_response_time = np.mean(response_times[(source, target)]) / 3600  # Average response time i
        G.add_edge(source, target, weight=weight, response_time=avg_response_time)

    fig, ax = plt.subplots(figsize=(20, 16))
    pos = nx.spring_layout(G, seed=42)  # Use spring layout for positioning nodes
```

```
    edges = G.edges()
    weights = [G[u][v]['weight'] for u, v in edges]
    response_times = [G[u][v]['response_time'] for u, v in edges]

    max_weight = max(weights) if weights else 1
    max_response_time = max(response_times) if response_times else 1
    edge_colors = [plt.cm.viridis(rt / max_response_time) for rt in response_times]
    edge_widths = [3 * w / max_weight for w in weights]

    node_sizes = [G.nodes[node]['size'] for node in G.nodes()]
    nx.draw_networkx_nodes(G, pos, node_size=node_sizes, node_color='lightblue', edgecolors='black', al

    nx.draw_networkx_edges(G, pos, edge_color=edge_colors, width=edge_widths, alpha=0.7, edge_cmap=plt.

    nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold')

    size_labels = [min(node_sizes), np.median(node_sizes), max(node_sizes)]
    legend_labels = [f'{int(2 ** (size / 100) - 1)} comments' for size in size_labels]
    for size, label in zip(size_labels, legend_labels):
        plt.scatter([], [], s=size, c='lightblue', label=label, edgecolors='black', alpha=0.9)
    plt.legend(title='Author Activity', loc='upper right')

    sm = plt.cm.ScalarMappable(cmap=plt.cm.viridis, norm=plt.Normalize(0, max_response_time))
    sm.set_array([])
    plt.colorbar(sm, ax=ax, label='Average Response Time (hours)')

    plt.title('Influence Network of Key Authors\nEdge thickness = interaction frequency, Edge color = r
    plt.axis('off')

    plt.savefig(output_path, dpi=300, bbox_inches='tight')
    plt.close()

    stats = {
        'num_nodes': G.number_of_nodes(),
        'num_edges': G.number_of_edges(),
        'avg_degree': np.mean([d for n, d in G.degree()]),
        'most_connected': sorted(G.degree(weight='weight'), key=lambda x: x[1], reverse=True)[:5]
    }
    return stats


output_path = os.path.join(images_dir, "influence_network.png")
stats = create_influence_network(dataset1_comments, influential_authors, output_path=output_path)

print("\nNetwork Summary:")
```

```
##
## Network Summary:
```

```
print(f"Number of authors: {stats['num_nodes']}")
```

```
## Number of authors: 20
```

```
print(f"Number of interactions: {stats['num_edges']}")
```

```
## Number of interactions: 114
```

```
print(f"Average degree: {stats['avg_degree']:.2f}")
```

```
## Average degree: 11.40
```

```
print("\nMost Connected Authors (by total interactions):")
```

```
##
## Most Connected Authors (by total interactions):
```

```
for author, degree in stats['most_connected']:
    print(f"{author}: {degree} interactions")
```

```
## Steve Dyer: 320 interactions
## Ron Rizzo: 236 interactions
## Craig Werner: 204 interactions
## Rob Bernardo: 91 interactions
## Bill Stoll: 44 interactions
```

Tone

##Sentiment Descriptive Statistics

```
def calculate_descriptive_stats(df, column_name):
    """Calculate descriptive statistics for a given dataframe and column."""
    min_value = df[column_name].min()
    max_value = df[column_name].max()
    range_value = max_value - min_value
    mean_value = df[column_name].mean()
    median_value = df[column_name].median()
    sd_value = df[column_name].std()
    skewness_value = skew(df[column_name], nan_policy='omit')
    kurtosis_value = kurtosis(df[column_name], nan_policy='omit')

    return {
        'Min': min_value,
        'Max': max_value,
        'Range': range_value,
        'Mean': mean_value,
        'Median': median_value,
        'SD': sd_value,
        'Skewness': skewness_value,
        'Kurtosis': kurtosis_value
    }

datasets = {
```

```
    "Dataset One (All Comments)": dataset1_comments,
    "Dataset Two (Relevant Comments)": dataset2_comments,
    "Dataset Three (Influential Authors)": dataset3_comments_onlyinfluential
}

results = {}
for dataset_name, df in datasets.items():
    stats = calculate_descriptive_stats(df, column_name="SentimentScore")  # Replace 'SentimentScore' w
    results[dataset_name] = stats

stats_df = pd.DataFrame(results).T
stats_df = stats_df.reset_index().rename(columns={'index': 'Dataset'})

output_path = '/Users/emerson/Github/usenet_webpage/Images and Tables/Tables/sentiment_descriptive_stats
stats_df.to_html(output_path, index=False, float_format="%.2f", border=1)
print(f"HTML table saved to: {output_path}")
```

```
## HTML table saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Tables/sentiment_descrip
```

```
from IPython.core.display import display, HTML
```

```
## <string>:2: DeprecationWarning:
##
## Importing display from IPython.core.display is deprecated since IPython 7.14, please import from IPy
```

```
display(HTML(stats_df.to_html(index=False, float_format="%.2f", border=1)))
```

```
## <IPython.core.display.HTML object>
```

Sentiment Visualizations

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

    plt.figure(figsize=(15, 8))

    def calculate_monthly_sentiment(df):
        df['Date'] = pd.to_datetime(df['Date'])
        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()

    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)

    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')

    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"
    }

    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)

    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()

    images_dir = os.path.join(output_directory, "Images and Tables/Images")
    os.makedirs(images_dir, exist_ok=True)

    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

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

```
    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

```
dates = pd.date_range(start='1982-01-01', end='1987-01-01', freq='M')

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))
})

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

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

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)

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)

anova_results = perform_anova(combined_sentiment_data)
pairwise_results = perform_pairwise_ttests(combined_sentiment_data)

print("\nANOVA Results:")
```

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

```
print(anova_results)
```

```
##              sum_sq     df         F    PR(>F)
## C(Group)   0.526494    2.0  0.766966  0.465957
## Residual  60.752015  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     1.081762  0.281570
## 1    Dataset One  Influential Authors    -0.029957  0.976152
## 2  Dataset Three  Influential Authors    -1.089032  0.278378
```