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 = pd.read_csv(os.path.join(comments_directory, "dataset3_comments.csv"))
dataset4_threads = pd.read_csv(os.path.join(threads_directory, "dataset4_threads.csv"))
dataset4_comments_all = pd.read_csv(os.path.join(comments_directory, "dataset4_comments_all.csv"))
dataset4_comments_onlyinfluential = pd.read_csv(os.path.join(comments_directory, "dataset4_comments_onl
influential_authors = pd.read_csv(os.path.join(output_directory, "CSV Files/influential_authors.csv"))
```

Cool Initial Visualizations

Theme Prevelance Hypothesis

LDA Topic Modeling

```
# Download required NLTK data
nltk.download('stopwords', quiet=True)
```

```
## True
```

```
nltk.download('punkt', quiet=True)
```

```
## True
```

```
def preprocess_text(text):
    """Preprocess text data"""
    # Convert to lowercase
    text = str(text).lower()
    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

def create_styled_html(general_results_df):
    """Create a styled HTML table from the results DataFrame"""
    # Create a copy of the DataFrame for styling
    styled_df = general_results_df.copy()

    # Define CSS styles
    styles = [
        dict(selector="caption",
             props=[("caption-side", "top"),
                    ("font-size", "16px"),
                    ("font-weight", "bold"),
                    ("text-align", "center"),
                    ("padding", "10px")]),
        dict(selector="th",
             props=[("font-size", "14px"),
                    ("text-align", "center"),
                    ("background-color", "#f0f0f0"),
                    ("padding", "8px")]),
        dict(selector="td",
             props=[("padding", "8px"),
                    ("text-align", "left")]),
        dict(selector="tr:nth-child(even)",
```

```
            props=[("background-color", "#f9f9f9")]),
        dict(selector="table",
            props=[("border-collapse", "collapse"),
                ("width", "100%"),
                ("margin", "20px 0"),
                ("font-family", "Arial, sans-serif")]),
        dict(selector="",
            props=[("border", "1px solid #ddd")])
    ]

    # Apply styling
    styled_table = (styled_df.style
                    .set_table_styles(styles)
                    .set_caption("LDA Topic Analysis Results")
                    .format(precision=4)
                    .background_gradient(subset=[col for col in styled_df.columns if 'betas' in col],
                                         cmap='Blues')
                    .hide(axis='index'))

    # Add custom CSS for alternating topic columns
    for i in range(1, 7):
        styled_table = styled_table.set_properties(**{
            f'Topic_{i}_terms': {
                'background-color': f'rgba(240, 240, 240, {0.1 * i})',
                'border-right': '2px solid #ddd'
            },
            f'Topic_{i}_betas': {
                'background-color': f'rgba(240, 240, 240, {0.1 * i})',
                'border-right': '2px solid #ddd'
            }
        })

    return styled_table

def analyze_topics(filtered_comments, n_topics=6, n_top_words=20):
    """Perform topic analysis on the comments with adjusted parameters"""
    print("Starting topic analysis...")

    # Convert dates to datetime and sort
    filtered_comments['Date'] = pd.to_datetime(filtered_comments['Date'])
    filtered_comments = filtered_comments.sort_values('Date')

    # Check temporal distribution
    print("\nTemporal distribution of documents:")
    print(filtered_comments['Date'].dt.year.value_counts().sort_index())
    print("\nMonthly document counts:")
    print(filtered_comments.groupby(pd.Grouper(key='Date', freq='M')).size().sort_index())

    # Preprocess texts
    texts = [preprocess_text(text) for text in filtered_comments['Full.Text']]

    # Create document-term matrix with adjusted parameters
    print("Creating document-term matrix...")
```

```
vectorizer = CountVectorizer(
    max_df=0.8,
    min_df=3,
    stop_words='english',
    max_features=5000
)
doc_term_matrix = vectorizer.fit_transform(texts)

# Train LDA model with adjusted parameters
print("Training LDA model...")
lda = LatentDirichletAllocation(
    n_components=n_topics,
    random_state=123,
    max_iter=50,
    learning_decay=0.7,
    batch_size=128,
    evaluate_every=5
)

# Fit the model
lda.fit(doc_term_matrix)

# Get feature names
feature_names = vectorizer.get_feature_names_out()

# Create top terms dataframe
print("Extracting top terms...")
top_terms_dict = {}

for topic_idx, topic in enumerate(lda.components_):
    top_indices = topic.argsort()[:-n_top_words-1:-1]
    top_terms = [feature_names[i] for i in top_indices]
    top_betas = [topic[i] for i in top_indices]

    top_terms_dict[f'Topic_{topic_idx+1}_terms'] = top_terms
    top_terms_dict[f'Topic_{topic_idx+1}_betas'] = [round(beta, 4) for beta in top_betas]

# Create DataFrame
general_results_df = pd.DataFrame(top_terms_dict)

# Add temporal information
doc_topics = lda.transform(doc_term_matrix)
document_results = pd.DataFrame(doc_topics)
document_results.columns = [f'Topic_{i+1}' for i in range(n_topics)]
document_results['Date'] = filtered_comments['Date']

# Save table to Tables directory
tables_path = os.path.join(tables_dir, "lda_analysis_dataset3.html")

# Create HTML content
html_content = f"""
<html>
<head>
```

```
        <style>
            table {{
                border-collapse: collapse;
                width: 100%;
                margin: 20px 0;
                font-family: Arial, sans-serif;
            }}
            th, td {{
                border: 1px solid #ddd;
                padding: 8px;
                text-align: left;
            }}
            th {{
                background-color: #f5f5f5;
            }}
            tr:nth-child(even) {{
                background-color: #f9f9f9;
            }}
            caption {{
                font-size: 1.2em;
                margin-bottom: 10px;
                font-weight: bold;
            }}
        </style>
    </head>
    <body>
        <table>
            <caption>Dataset 3 LDA Topic Analysis Results</caption>
            {general_results_df.to_html(index=False)}
        </table>
    </body>
    </html>
    """

    with open(tables_path, 'w', encoding='utf-8') as f:
        f.write(html_content)

    # Create visualization
    print("Creating visualization...")
    plt.figure(figsize=(15, 10))
    for topic_idx in range(n_topics):
        plt.subplot(2, 3, topic_idx + 1)
        top_terms = top_terms_dict[f'Topic_{topic_idx+1}_terms'][:10]
        top_betas = top_terms_dict[f'Topic_{topic_idx+1}_betas'][:10]
        plt.barh(range(len(top_terms)), top_betas)
        plt.yticks(range(len(top_terms)), top_terms)
        plt.title(f'Topic {topic_idx + 1}')

    plt.tight_layout()

    # Save plot to Images directory
    plot_path = os.path.join(images_dir, "lda_visualization_dataset3.png")
    plt.savefig(plot_path, dpi=300, bbox_inches='tight')
```

```
    plt.close()

    print("Analysis complete! Results saved to:")
    print(f"- Table: {tables_path}")
    print(f"- Plot: {plot_path}")

    return general_results_df, lda, vectorizer, doc_term_matrix, document_results

# Run the analysis
try:
    print("\nStarting topic analysis...")
    general_results_df, lda_model, vectorizer, doc_term_matrix, document_results = analyze_topics(datas
    print("\nAnalysis completed successfully!")

    # Display top terms for each topic
    print("\nTop terms for each topic:")
    print(general_results_df.head())

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

```
##
## Starting topic analysis...
## Starting topic analysis...
##
## Temporal distribution of documents:
## Date
## 1982      5
## 1983     10
## 1984     28
## 1985    192
## 1986     53
## Name: count, dtype: int64
##
## Monthly document counts:
## <string>:14: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
## Date
## 1982-12-31     5
## 1983-01-31     4
## 1983-02-28     0
## 1983-03-31     0
## 1983-04-30     0
## 1983-05-31     0
## 1983-06-30     0
## 1983-07-31     0
## 1983-08-31     0
## 1983-09-30     0
## 1983-10-31     4
## 1983-11-30     2
```

```
## 1983-12-31     0
## 1984-01-31     2
## 1984-02-29     1
## 1984-03-31     0
## 1984-04-30     4
## 1984-05-31     0
## 1984-06-30     1
## 1984-07-31     0
## 1984-08-31     7
## 1984-09-30     1
## 1984-10-31     7
## 1984-11-30     3
## 1984-12-31     2
## 1985-01-31    22
## 1985-02-28     5
## 1985-03-31     3
## 1985-04-30     5
## 1985-05-31     1
## 1985-06-30     6
## 1985-07-31    16
## 1985-08-31    21
## 1985-09-30    29
## 1985-10-31    48
## 1985-11-30    24
## 1985-12-31    12
## 1986-01-31     9
## 1986-02-28    19
## 1986-03-31    19
## 1986-04-30     0
## 1986-05-31     0
## 1986-06-30     0
## 1986-07-31     2
## 1986-08-31     0
## 1986-09-30     4
## Freq: ME, dtype: int64
## Creating document-term matrix...
## Training LDA model...
## Extracting top terms...
## Creating visualization...
## Analysis complete! Results saved to:
## - Table: /Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_dataset3.html
## - Plot: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/lda_visualization_dataset3.png
##
## Analysis completed successfully!
##
## Top terms for each topic:
##   Topic_1_terms  Topic_1_betas  ... Topic_6_terms  Topic_6_betas
## 0           gay       122.8323  ...          aids        16.9779
## 1          dont       102.9667  ...          dont         8.7032
## 2        people        96.9549  ...          test         8.1690
## 3           sex        82.1969  ...          quit         8.1179
## 4          like        75.3091  ...           ill         8.0927
##
## [5 rows x 12 columns]
```

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...")

    # 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...
## Creating static co-occurrence network...
##
## Network visualization completed successfully!
## Check 'topic_network_visualization.png' for the static network visualization.
```

Emotional Tone Hypothesis

```
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(dataset3_comments)
monthly_sentiment_influential = calculate_monthly_sentiment(dataset4_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: 0.741
## Number of months: 59
##
## Dataset Three:
## Average sentiment: -4.296
## Number of months: 46
##
## Influential Authors:
## Average sentiment: -4.681
## Number of months: 33
```

Author Impact Hypothesis

Topic Similarity

```
# Download required NLTK data
nltk.download('stopwords', quiet=True)
```

```
## True
```

```
nltk.download('punkt', quiet=True)
```

```
## True
```

```
def preprocess_text(text):
    """Preprocess text data"""
    # Convert to lowercase
    text = str(text).lower()
    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

def create_styled_html(influential_results_df):
    """Create a styled HTML table from the results DataFrame"""
    # Create a copy of the DataFrame for styling
    styled_df = influential_results_df.copy()

    # Define CSS styles
    styles = [
        dict(selector="caption",
             props=[("caption-side", "top"),
                    ("font-size", "16px"),
                    ("font-weight", "bold"),
                    ("text-align", "center"),
                    ("padding", "10px")]),
        dict(selector="th",
             props=[("font-size", "14px"),
                    ("text-align", "center"),
                    ("background-color", "#f0f0f0"),
                    ("padding", "8px")]),
        dict(selector="td",
             props=[("padding", "8px"),
                    ("text-align", "left")]),
        dict(selector="tr:nth-child(even)",
             props=[("background-color", "#f9f9f9")]),
        dict(selector="table",
             props=[("border-collapse", "collapse"),
```

```
                    ("width", "100%"),
                    ("margin", "20px 0"),
                    ("font-family", "Arial, sans-serif")]),
        dict(selector="",
             props=[("border", "1px solid #ddd")])
    ]

    # Apply styling
    styled_table = (styled_df.style
                    .set_table_styles(styles)
                    .set_caption("LDA Topic Analysis Results")
                    .format(precision=4)
                    .background_gradient(subset=[col for col in styled_df.columns if 'betas' in col],
                                         cmap='Blues')
                    .hide(axis='index'))

    # Add custom CSS for alternating topic columns
    for i in range(1, 7):
        styled_table = styled_table.set_properties(**{
            f'Topic_{i}_terms': {
                'background-color': f'rgba(240, 240, 240, {0.1 * i})',
                'border-right': '2px solid #ddd'
            },
            f'Topic_{i}_betas': {
                'background-color': f'rgba(240, 240, 240, {0.1 * i})',
                'border-right': '2px solid #ddd'
            }
        })

    return styled_table

def analyze_topics(filtered_comments, n_topics=6, n_top_words=20):
    """Perform topic analysis on the comments with adjusted parameters"""
    print("Starting topic analysis...")

    # Convert dates to datetime and sort
    filtered_comments['Date'] = pd.to_datetime(filtered_comments['Date'])
    filtered_comments = filtered_comments.sort_values('Date')

    # Check temporal distribution
    print("\nTemporal distribution of documents:")
    print(filtered_comments['Date'].dt.year.value_counts().sort_index())
    print("\nMonthly document counts:")
    print(filtered_comments.groupby(pd.Grouper(key='Date', freq='M')).size().sort_index())

    # Preprocess texts
    texts = [preprocess_text(text) for text in filtered_comments['Full.Text']]

    # Create document-term matrix with adjusted parameters
    print("Creating document-term matrix...")
    vectorizer = CountVectorizer(
        max_df=0.8,
        min_df=3,
```

```
    stop_words='english',
    max_features=5000
)
doc_term_matrix = vectorizer.fit_transform(texts)

# Train LDA model with adjusted parameters
print("Training LDA model...")
lda = LatentDirichletAllocation(
    n_components=n_topics,
    random_state=123,
    max_iter=50,
    learning_decay=0.7,
    batch_size=128,
    evaluate_every=5
)

# Fit the model
lda.fit(doc_term_matrix)

# Get feature names
feature_names = vectorizer.get_feature_names_out()

# Create top terms dataframe
print("Extracting top terms...")
top_terms_dict = {}

for topic_idx, topic in enumerate(lda.components_):
    top_indices = topic.argsort()[:-n_top_words-1:-1]
    top_terms = [feature_names[i] for i in top_indices]
    top_betas = [topic[i] for i in top_indices]

    top_terms_dict[f'Topic_{topic_idx+1}_terms'] = top_terms
    top_terms_dict[f'Topic_{topic_idx+1}_betas'] = [round(beta, 4) for beta in top_betas]

# Create DataFrame
influential_results_df = pd.DataFrame(top_terms_dict)

# Add temporal information
doc_topics = lda.transform(doc_term_matrix)
document_results = pd.DataFrame(doc_topics)
document_results.columns = [f'Topic_{i+1}' for i in range(n_topics)]
document_results['Date'] = filtered_comments['Date']

# Save table to Tables directory
tables_path = os.path.join(tables_dir, "lda_analysis_dataset4.html")

# Create HTML content
html_content = f"""
<html>
<head>
    <style>
        table {{
            border-collapse: collapse;
```

```
            width: 100%;
            margin: 20px 0;
            font-family: Arial, sans-serif;
        }}
        th, td {{
            border: 1px solid #ddd;
            padding: 8px;
            text-align: left;
        }}
        th {{
            background-color: #f5f5f5;
        }}
        tr:nth-child(even) {{
            background-color: #f9f9f9;
        }}
        caption {{
            font-size: 1.2em;
            margin-bottom: 10px;
            font-weight: bold;
        }}
    </style>
</head>
<body>
    <table>
        <caption>Dataset 4 LDA Topic Analysis Results</caption>
        {influential_results_df.to_html(index=False)}
    </table>
</body>
</html>
"""

with open(tables_path, 'w', encoding='utf-8') as f:
    f.write(html_content)

# Create visualization
print("Creating visualization...")
plt.figure(figsize=(15, 10))
for topic_idx in range(n_topics):
    plt.subplot(2, 3, topic_idx + 1)
    top_terms = top_terms_dict[f'Topic_{topic_idx+1}_terms'][:10]
    top_betas = top_terms_dict[f'Topic_{topic_idx+1}_betas'][:10]
    plt.barh(range(len(top_terms)), top_betas)
    plt.yticks(range(len(top_terms)), top_terms)
    plt.title(f'Topic {topic_idx + 1}')

plt.tight_layout()

# Save plot to Images directory
plot_path = os.path.join(images_dir, "lda_visualization_dataset4.png")
plt.savefig(plot_path, dpi=300, bbox_inches='tight')
plt.close()

print("Analysis complete! Results saved to:")
```

```
    print(f"- Table: {tables_path}")
    print(f"- Plot: {plot_path}")

    return influential_results_df, lda, vectorizer, doc_term_matrix, document_results

# Run the analysis
try:
    print("\nStarting topic analysis...")
    influential_results_df, lda_model, vectorizer, doc_term_matrix, document_results = analyze_topics(da
    print("\nAnalysis completed successfully!")

    # Display top terms for each topic
    print("\nTop terms for each topic:")
    print(influential_results_df.head())

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

```
##
## Starting topic analysis...
## Starting topic analysis...
##
## Temporal distribution of documents:
## Date
## 1984     5
## 1985    92
## 1986    15
## Name: count, dtype: int64
##
## Monthly document counts:
## <string>:14: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
## Date
## 1984-01-31     1
## 1984-02-29     0
## 1984-03-31     0
## 1984-04-30     0
## 1984-05-31     0
## 1984-06-30     1
## 1984-07-31     0
## 1984-08-31     2
## 1984-09-30     0
## 1984-10-31     0
## 1984-11-30     0
## 1984-12-31     1
## 1985-01-31     9
## 1985-02-28     3
## 1985-03-31     2
## 1985-04-30     3
## 1985-05-31     1
```

```
## 1985-06-30     3
## 1985-07-31     6
## 1985-08-31    11
## 1985-09-30    18
## 1985-10-31    26
## 1985-11-30     8
## 1985-12-31     2
## 1986-01-31     5
## 1986-02-28     4
## 1986-03-31     5
## 1986-04-30     0
## 1986-05-31     0
## 1986-06-30     0
## 1986-07-31     0
## 1986-08-31     0
## 1986-09-30     1
## Freq: ME, dtype: int64
## Creating document-term matrix...
## Training LDA model...
## Extracting top terms...
## Creating visualization...
## Analysis complete! Results saved to:
## - Table: /Users/emerson/Github/usenet_webpage/Images and Tables/Tables/lda_analysis_dataset4.html
## - Plot: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/lda_visualization_dataset4.png
##
## Analysis completed successfully!
##
## Top terms for each topic:
##   Topic_1_terms  Topic_1_betas  ... Topic_6_terms  Topic_6_betas
## 0         virus        52.8712  ...         virus        19.9342
## 1        immune        43.9322  ...       article        14.3091
## 2         cells        41.5976  ...          like        14.2630
## 3        cancer        21.0905  ...     practices        14.1039
## 4         blood        20.1301  ...         steve        12.2899
##
## [5 rows x 12 columns]
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import os

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

    # List to store topic similarity data
    topic_similarities = []

    # Calculate Jaccard similarity for all topic pairs
    for i in range(1, len(general_results.columns) // 2 + 1):  # General topics
```

```
        general_terms = set(general_results[f'Topic_{i}_terms'])
        for j in range(1, len(influential_results.columns) // 2 + 1):  # Influential topics
            influential_terms = set(influential_results[f'Topic_{j}_terms'])
            similarity = len(general_terms.intersection(influential_terms)) / len(general_terms.union(i
            topic_similarities.append({
                'General_Topic': i,
                'Influential_Topic': j,
                'Similarity': similarity
            })

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

    # 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 Authors 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 Authors 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}")

    # 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
try:
    comparison_df, best_matches_df = compare_topics(
        general_results_df,
        influential_results_df,
        output_dir=os.path.join(output_directory, "Images and Tables/Images")
    )
    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
##
## Topic Similarity Statistics:
## Average similarity: 0.122
## Maximum similarity: 0.429
## Minimum similarity: 0.000
##
## Jaccard Similarity Analysis Complete.
##
## Best Matches DataFrame:
##    General_Topic  Influential_Topic  Similarity
## 0              1                  3    0.176471
## 1              2                  5    0.428571
## 2              3                  1    0.142857
## 3              4                  4    0.250000
## 4              5                  6    0.250000
## 5              6                  2    0.052632
```

Keyword Adoption

```
# Define selected AIDS-related keywords
aids_related_keywords = [
    "aids", "virus", "disease", "immune", "blood", "gay", "homosexuality",
    "hiv", "sexual", "cells", "medical", "drug", "patients", "test", "health",
    "public", "rights", "positive", "infection", "htlv", "homosexual"
]

def preprocess_text(text):
    """Preprocess text by cleaning and tokenizing."""
    # Convert to lowercase and remove special characters
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words and len(word) > 2]
    return tokens

def calculate_keyword_prevalence_aids(df, keywords, text_column='Full.Text'):
    """Calculate keyword prevalence for selected AIDS-related terms."""
    # Check for existing 'Date' column ambiguity and rename conflicting columns
    if 'Date' in df.columns and df.columns.duplicated().any():
        print("Found duplicate columns. Renaming to avoid conflicts...")
        df = df.loc[:, ~df.columns.duplicated()]
```

```
    # Ensure 'Date' column is datetime
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

    # Drop rows with invalid dates
    df = df.dropna(subset=['Date'])

    # Process texts and count keywords by month
    monthly_counts = []
    for idx, row in df.iterrows():
        tokens = preprocess_text(row[text_column])
        # Count keywords in this document
        keyword_counts = Counter(word for word in tokens if word in keywords)
        # Add date information
        for word, count in keyword_counts.items():
            monthly_counts.append({
                'Date': row['Date'],
                'word': word,
                'count': count
            })

    # Convert to DataFrame
    counts_df = pd.DataFrame(monthly_counts)

    # Check if counts_df is empty
    if counts_df.empty:
        raise ValueError("No keywords found in the dataset after processing.")

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

    # Calculate prevalence
    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):
    """Create faceted plot of keyword adoption patterns."""
    # Combine the data
    influential_prevalence['Group'] = 'Influential Authors'
    overall_prevalence['Group'] = 'Overall Discussion'
    combined_data = pd.concat([influential_prevalence, overall_prevalence])

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

    # Calculate grid dimensions
    n_cols = 4
```

```
    n_rows = int(np.ceil(n_keywords / n_cols))

    # Create figure
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(16, n_rows * 2.5))
    fig.suptitle('Keyword Adoption Over Time: Influential Authors vs Overall Discussion')

    # Flatten axes array for easier iteration
    axes = axes.flatten()

    # Plot each keyword
    for idx, keyword in enumerate(sorted(keywords)):
        keyword_data = combined_data[combined_data['word'] == keyword]

        # Plot lines for both groups
        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)

        # Customize subplot
        axes[idx].set_title(keyword)
        axes[idx].tick_params(axis='x', rotation=45)
        axes[idx].grid(True, alpha=0.3)

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

    # Add legend
    handles, labels = axes[0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, .96),
               ncol=2, frameon=False)

    # Adjust layout
    plt.tight_layout()
    plt.subplots_adjust(top=.9)

    # Save plot
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.close()

# Run Keyword Adoption Analysis
if __name__ == "__main__":
    try:
        print("\nAnalyzing adoption patterns for AIDS-related keywords...")

        # Calculate keyword prevalence for AIDS-related terms
        influential_prevalence = calculate_keyword_prevalence_aids(
            dataset4_comments_onlyinfluential, aids_related_keywords
        )
        overall_prevalence = calculate_keyword_prevalence_aids(
            dataset3_comments, aids_related_keywords
```

```
        )

        # Create visualization
        output_path = os.path.join(images_dir, "aids_keyword_adoption_over_time.png")
        plot_keyword_adoption(influential_prevalence, overall_prevalence, output_path)

        print("\nAnalysis completed successfully!")
        print(f"Visualization saved to: {output_path}")

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

```
##
## Analyzing adoption patterns for AIDS-related keywords...
## <string>:38: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
## <string>:38: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
##
##
## Analysis completed successfully!
## Visualization saved to: /Users/emerson/Github/usenet_webpage/Images and Tables/Images/aids_keyword_a
```

Statistical Analysis

Emotional Tone

```
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')
```

```
## <string>:3: FutureWarning:
##
## 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
```

```
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)   0.263799    2.0  0.365854  0.694127
## Residual  63.812960  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    -0.547565  0.585025
## 1    Dataset One  Influential Authors    -0.835713  0.405026
## 2  Dataset Three  Influential Authors    -0.312314  0.755358
```

Thematic Prevelance and Jaccard Similarity

```
from scipy.stats import ttest_rel, ttest_ind

# Prepare data for statistical analysis
def prepare_similarity_data(comparison_df):
    """
    Extract general and influential topic similarities for analysis.
    """
    similarities = comparison_df['Similarity']
    return similarities

# Perform t-test on Jaccard similarities
def perform_similarity_ttest(general_similarities, influential_similarities):
    """
    Perform paired t-test on Jaccard similarities between general and influential topics.
    """
    t_stat, p_value = ttest_rel(general_similarities, influential_similarities)
    return t_stat, p_value

# Extract similarities for analysis
general_similarities = comparison_df.pivot(index='General_Topic', columns='Influential_Topic', values='
influential_similarities = comparison_df.pivot(index='General_Topic', columns='Influential_Topic', valu

# Perform t-test
t_stat, p_value = perform_similarity_ttest(general_similarities, influential_similarities)

# Display results
print("\nJaccard Similarity Paired t-Test Results:")
```

```
##
## Jaccard Similarity Paired t-Test Results:
```

```
print(f"t-statistic: {t_stat:.3f}, p-value: {p_value:.3f}")
```

```
## t-statistic: 0.000, p-value: 1.000
```

```
# Determine significance
if p_value < 0.05:
    print("The differences in Jaccard similarities are statistically significant.")
else:
    print("The differences in Jaccard similarities are not statistically significant.")
```

```
## The differences in Jaccard similarities are not statistically significant.
```