Exploring_Social_Media_Reactions_to_Disasters_Using_Text_Mining_and Analysis

November 17, 2024

[1]:
```python
# Importing required libraries for the project
import requests
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
import xml.etree.ElementTree
import pandas as pd
import numpy as np
import re
from datetime import datetime
import string
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob
import spacy
import en_core_web_sm
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from collections import Counter
from sklearn.decomposition import PCA
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from PIL import Image
from io import BytesIO
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
```

```
import networkx as nx
from nrclex import NRCLex
import warnings
warnings.filterwarnings("ignore")
```

1 Data Collection

1.1 1. Guardian News API Data

[2]:
```
# Extracting data from guardian news API and converting to dataframe
data = requests.get("https://content.guardianapis.com/search?
 ↪q=turkey%20syria%20earthquake&from-date=2023-02-06&page-size=100&show-fields=bodyText&api-k
df_news = pd.DataFrame(data.json()['response']['results'])
df_news['content'] = df_news['fields'].apply(lambda x: x['bodyText'])
df_news.head()
```

[2]:
```
                                                  id      type  \
0  world/2024/oct/24/turkey-airstrikes-syria-iraq…   article
1  music/article/2024/jul/19/beyza-yazgan-human-c…   article
2  football/live/2024/nov/16/turkey-wales-uefa-na…  liveblog
3  world/article/2024/jul/17/archive-1974-turkey-…   article
4  global-development/2024/feb/04/year-aftermath-…   article

            sectionId         sectionName    webPublicationDate  \
0               world          World news  2024-10-24T10:38:08Z
1               music               Music  2024-07-19T07:30:34Z
2            football            Football  2024-11-16T19:31:46Z
3               world          World news  2024-07-17T10:24:27Z
4  global-development  Global development  2024-02-04T09:00:40Z

                                            webTitle  \
0  Turkey strikes in Syria and Iraq after attack …
1  Beyza Yazgan: Human Cocoon review – from Middl…
2  Turkey 0-0 Wales: Nations League – as it happened
3              Archive, 1974: Turkey invades Cyprus
4  A year in the aftermath of Turkey’s earthquake…

                                              webUrl  \
0  https://www.theguardian.com/world/2024/oct/24/…
1  https://www.theguardian.com/music/article/2024…
2  https://www.theguardian.com/football/live/2024…
3  https://www.theguardian.com/world/article/2024…
4  https://www.theguardian.com/global-development…

                                              apiUrl  \
0  https://content.guardianapis.com/world/2024/oc…
```

```
1  https://content.guardianapis.com/music/article…
2  https://content.guardianapis.com/football/live…
3  https://content.guardianapis.com/world/article…
4  https://content.guardianapis.com/global-develo…

                                              fields  isHosted      pillarId  \
0  {'bodyText': 'Turkey has launched airstrikes a…     False   pillar/news
1  {'bodyText': 'Born in Busan, Turkey, trained i…     False   pillar/arts
2  {'bodyText': 'Jamie Jackson's report has lande…     False  pillar/sport
3  {'bodyText': 'Invading Turks claim capture of …     False   pillar/news
4  {'bodyText': 'The living and the dead will soo…     False   pillar/news

  pillarName                                            content
0       News  Turkey has launched airstrikes against suspect…
1       Arts  Born in Busan, Turkey, trained in Warsaw and, …
2      Sport  Jamie Jackson's report has landed. Here it is…
3       News  Invading Turks claim capture of Nicosia The Ob…
4       News  The living and the dead will soon be side by s…
```

[3]:
```python
# Saving the Guardian News data to an Excel file
guardian_output_file = "df_news.xlsx"
df_news.to_excel(guardian_output_file, sheet_name="Guardian_News", index=False)

print(f"Guardian News dataset saved to {guardian_output_file}")
```

```
Guardian News dataset saved to df_news.xlsx
```

1.2 2. World News API Data

[4]:
```python
# Extracting data from world news API and converting to dataframe
api = "66780804a1b44b4f963d1b130827154f"
url = "https://api.worldnewsapi.com/search-news"
params = {"api-key":api,
          "text":"earthquake",
          "number":100}
data = requests.get(url,params=params)

df2 = pd.DataFrame(data.json()['news'])
df2.rename(columns={'text':'content','publish_date':'webPublicationDate','url':
 ↪'webUrl'},inplace=True)
df2.head()
```

[4]:
```
          id                                              title  \
0  261882492           Expert reveals cause of Sinai's earthquake
1  264488116  Strong earthquake hits Pakistan, tremors from …
2  261177000  A mega-earthquake could strike the Pacific Nor…
3  260593560  Nanded, Maharashtra trembled due to earthquake…
```

```
4  260840558  Newly discovered fault line explains why NYC s…

                                             content  \
0  Egypt's National Research Institute of Astrono…
1  Pakistan. Earthquake tremors were felt in many…
2  Your phone blares, "Earthquake!" The voice ins…
3  Maharashtra. There is panic spread in Maharash…
4  A newly discovered fault line in the Northeast…

                                             summary  \
0  Egypt&#8217;s National Research Institute of A…
1  Pakistan. Earthquake tremors were felt in many…
2  A huge earthquake and tsunami will someday hit…
3  Maharashtra. There is panic spread in Maharash…
4  A &quot;previously unmapped&quot; fault line c…

                                              webUrl  \
0  https://www.egyptindependent.com/expert-reveal…
1  https://thenewsglory.com/strong-earthquake-hit…
2  https://www.businessinsider.com/big-one-earthq…
3  https://thenewsglory.com/nanded-maharashtra-tr…
4  https://gothamist.com/news/newly-discovered-fa…

                                               image video  \
0  https://amayei.nyc3.digitaloceanspaces.com/202…  None
1  https://nextindiatimes.com/wp-content/uploads/…  None
2  https://i.insider.com/6706de84198738e3a70f7732…  None
3  https://nextindiatimes.com/wp-content/uploads/…  None
4  https://api-prod.gothamist.com/images/347813/f…  None

   webPublicationDate                author                 authors  \
0  2024-11-01 04:08:14       Al-Masry Al-Youm       [Al-Masry Al-Youm]
1  2024-11-13 08:57:00         The News Glory         [The News Glory]
2  2024-10-26 10:33:02  Morgan McFall-Johnsen  [Morgan McFall-Johnsen]
3  2024-10-22 05:05:00         The News Glory         [The News Glory]
4  2024-10-23 19:39:38       Rosemary Misdary       [Rosemary Misdary]

  language source_country  sentiment  category
0       en             eg     -0.201       NaN
1       en             in     -0.710       NaN
2       en             US     -0.227  politics
3       en             in     -0.223       NaN
4       en             us     -0.461       NaN
```

[5]:
```
# Saving the World News data to an Excel file
world_news_output_file = "world_news.xlsx"
df2.to_excel(world_news_output_file, sheet_name="World_News", index=False)
```

```
print(f"World News dataset saved to {world_news_output_file}")
```

World News dataset saved to world_news.xlsx

1.3 3. Web Scraping 2 Datasets from Kaggle

[6]:
```
!pip install -q kaggle
```

[8]:
```
from ipywidgets import FileUpload
from IPython.display import display

# Creating and displaying the file upload widget
upload_widget = FileUpload(accept='', multiple=True)
display(upload_widget)
```

FileUpload(value=(), description='Upload', multiple=True)

[9]:
```
# Function to save uploaded files
def save_uploaded_files(upload_widget):
    if upload_widget.value:
        for file_info in upload_widget.value:  # Directly iterate over the␣
↪uploaded files
            filename = file_info["name"]
            content = file_info["content"]

            # Ensuring filenames do not have invalid characters
            safe_filename = filename.replace(" ", "_")

            # Saving the file
            with open(safe_filename, 'wb') as f:
                f.write(content)

        print(f"Files saved successfully: {[file['name'] for file in␣
↪upload_widget.value]}")
    else:
        print("No files uploaded.")

# Calling this function after uploading files
save_uploaded_files(upload_widget)
```

Files saved successfully: ['kaggle.json']

[10]:
```
# Making directory and coping the json file in it.
! mkdir ~/.kaggle/
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
mkdir: cannot create directory '/home/jovyan/.kaggle/': File exists
```

[11]:
```
# Fetching the required datasets
!kaggle competitions download -c nlp-getting-started
!kaggle datasets download vstepanenko/disaster-tweets
```

```
nlp-getting-started.zip: Skipping, found more recently modified local copy (use
--force to force download)
Dataset URL: https://www.kaggle.com/datasets/vstepanenko/disaster-tweets
License(s): CC0-1.0
disaster-tweets.zip: Skipping, found more recently modified local copy (use
--force to force download)
```

[11]:
```
# Unzipping the zip file gather from website.
! unzip nlp-getting-started.zip
! unzip disaster-tweets.zip
```

Dataset 1 (tweets.csv)

[12]:
```
# Converting the csv file to dataframe
df = pd.read_csv('tweets.csv')
print(df)
```

```
          id  keyword                 location  \
0          0   ablaze                      NaN
1          1   ablaze            New York City
2          2   ablaze           Morgantown, WV
3          3   ablaze                      NaN
4          4   ablaze                      NaN
…          …      …                      …
11365  11365  wrecked  Blue State in a red sea
11366  11366  wrecked               arohaonces
11367  11367  wrecked
11368  11368  wrecked           auroraborealis
11369  11369  wrecked                      NaN

                                                    text  target
0      Communal violence in Bhainsa, Telangana. "Ston…       1
1      Telangana: Section 144 has been imposed in Bha…       1
2      Arsonist sets cars ablaze at dealership https:…       1
3      Arsonist sets cars ablaze at dealership https:…       1
4      "Lord Jesus, your love brings freedom and pard…       0
…                                                    …      …
11365  Media should have warned us well in advance. T…       0
11366  i feel directly attacked   i consider moonbin …      0
11367  i feel directly attacked   i consider moonbin …      0
11368  ok who remember "outcast" nd the "dora" au?? T…       0
11369     Jake Corway wrecked while running 14th at IRP.       1
```

```
[11370 rows x 5 columns]
```

[13]:
```python
# Renaming the column text to content for easy analysing.
df.rename(columns={'text': 'content'}, inplace=True)
df.to_csv('disaster_data.csv', index=False)
print(df.head())
```

```
   id keyword         location  \
0   0  ablaze              NaN
1   1  ablaze              NaN
2   2  ablaze    New York City
3   3  ablaze  Morgantown, WV
4   4  ablaze              NaN

                                             content  target
0  Communal violence in Bhainsa, Telangana. "Ston…       1
1  Telangana: Section 144 has been imposed in Bha…       1
2  Arsonist sets cars ablaze at dealership https:…       1
3  Arsonist sets cars ablaze at dealership https:…       1
4  "Lord Jesus, your love brings freedom and pard…       0
```

Dataset 2 (train.csv and test.csv)

[14]:
```python
# Loading both CSV files and combining them
df1 = pd.read_csv('test.csv')
df2 = pd.read_csv('train.csv')
combined_df = pd.concat([df1, df2])

# Saving the result to a new CSV file
combined_df.to_csv('combined_rows.csv', index=False)
print(combined_df)
```

```
         id keyword location  \
0         0     NaN      NaN
1         2     NaN      NaN
2         3     NaN      NaN
3         9     NaN      NaN
4        11     NaN      NaN
…       …     …       …
7608  10869     NaN      NaN
7609  10870     NaN      NaN
7610  10871     NaN      NaN
7611  10872     NaN      NaN
7612  10873     NaN      NaN

                                                   text  target
0                     Just happened a terrible car crash     NaN
```

```
1     Heard about #earthquake is different cities, s…     NaN
2     there is a forest fire at spot pond, geese are…     NaN
3              Apocalypse lighting. #Spokane #wildfires     NaN
4          Typhoon Soudelor kills 28 in China and Taiwan     NaN
…                                                   …      …
7608  Two giant cranes holding a bridge collapse int…     1.0
7609  @aria_ahrary @TheTawniest The out of control w…     1.0
7610  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt…     1.0
7611  Police investigating after an e-bike collided …     1.0
7612  The Latest: More Homes Razed by Northern Calif…     1.0

[10876 rows x 5 columns]
```

[15]:
```python
# Renaming the column text to content for easy analysing.
combined_df = pd.read_csv('combined_rows.csv')

combined_df.rename(columns={'text': 'content'}, inplace=True)

combined_df.to_csv('disaster_data.csv', index=False)
print(combined_df.head())
```

```
   id keyword location                                            content  \
0   0     NaN      NaN                  Just happened a terrible car crash
1   2     NaN      NaN  Heard about #earthquake is different cities, s…
2   3     NaN      NaN  there is a forest fire at spot pond, geese are…
3   9     NaN      NaN           Apocalypse lighting. #Spokane #wildfires
4  11     NaN      NaN      Typhoon Soudelor kills 28 in China and Taiwan

   target
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
```

[16]:
```python
# Combining all four datasets
df_news = pd.concat([df_news, df2, combined_df, df])
```

2 Data Cleaning and Visualization

[17]:
```python
# Checking for the missing values and if found filling it with their mean for␣
↪Numerical and mode for Categorical Values.
def check_and_handle_missing_values(df_news):
    print("Checking for missing values...\n")

    # Checking for missing values
```

```
    missing_summary = df_news.isnull().sum()
    total_missing = missing_summary.sum()
    df_news['fields'].fillna("Unknown", inplace=True)

    if total_missing > 0:
        print(f"Missing Values Found:\n{missing_summary[missing_summary > 0]}")
        print(f"\nTotal Missing Values: {total_missing}")

        # Handling missing values
        for column in df_news.columns:
            if df_news[column].isnull().any():
                if df_news[column].dtype in ['float64', 'int64']:
                    mean_value = df_news[column].mean()
                    df_news[column].fillna(mean_value, inplace=True)
                    print(f"Filled missing values in '{column}' with mean:␣
↪{mean_value:.2f}")
                else:
                    mode_value = df_news[column].mode()[0]
                    df_news[column].fillna(mode_value, inplace=True)
                    print(f"Filled missing values in '{column}' with mode:␣
↪{mode_value}")
    else:
        print("No missing values found.\n")

    return df_news
print(df_news)
```

```
                                                     id       type  \
0      world/2024/oct/24/turkey-airstrikes-syria-iraq…    article
1      music/article/2024/jul/19/beyza-yazgan-human-c…    article
2      football/live/2024/nov/16/turkey-wales-uefa-na…  liveblog
3      world/article/2024/jul/17/archive-1974-turkey-…    article
4      global-development/2024/feb/04/year-aftermath-…    article
…                                                   …         …
11365                                            11365        NaN
11366                                            11366        NaN
11367                                            11367        NaN
11368                                            11368        NaN
11369                                            11369        NaN

                sectionId         sectionName    webPublicationDate  \
0                   world          World news  2024-10-24T10:38:08Z
1                   music               Music  2024-07-19T07:30:34Z
2                football            Football  2024-11-16T19:31:46Z
3                   world          World news  2024-07-17T10:24:27Z
4      global-development  Global development  2024-02-04T09:00:40Z
…                       …                   …                     …
```

```
11365                NaN                 NaN                      NaN
11366                NaN                 NaN                      NaN
11367                NaN                 NaN                      NaN
11368                NaN                 NaN                      NaN
11369                NaN                 NaN                      NaN

                                                webTitle  \
0      Turkey strikes in Syria and Iraq after attack …
1      Beyza Yazgan: Human Cocoon review – from Middl…
2      Turkey 0-0 Wales: Nations League – as it happened
3                   Archive, 1974: Turkey invades Cyprus
4      A year in the aftermath of Turkey’s earthquake…
…                                                    …
11365                                                NaN
11366                                                NaN
11367                                                NaN
11368                                                NaN
11369                                                NaN

                                                  webUrl  \
0      https://www.theguardian.com/world/2024/oct/24/…
1      https://www.theguardian.com/music/article/2024…
2      https://www.theguardian.com/football/live/2024…
3      https://www.theguardian.com/world/article/2024…
4      https://www.theguardian.com/global-development…
…                                                    …
11365                                                NaN
11366                                                NaN
11367                                                NaN
11368                                                NaN
11369                                                NaN

                                                  apiUrl  \
0      https://content.guardianapis.com/world/2024/oc…
1      https://content.guardianapis.com/music/article…
2      https://content.guardianapis.com/football/live…
3      https://content.guardianapis.com/world/article…
4      https://content.guardianapis.com/global-develo…
…                                                    …
11365                                                NaN
11366                                                NaN
11367                                                NaN
11368                                                NaN
11369                                                NaN

                                                  fields isHosted  \
0      {'bodyText': 'Turkey has launched airstrikes a…    False
1      {'bodyText': 'Born in Busan, Turkey, trained i…    False
```

```
2      {'bodyText': 'Jamie Jackson’s report has lande…    False
3      {'bodyText': 'Invading Turks claim capture of …    False
4      {'bodyText': 'The living and the dead will soo…    False
…                                                   …      …
11365                                                 NaN      NaN
11366                                                 NaN      NaN
11367                                                 NaN      NaN
11368                                                 NaN      NaN
11369                                                 NaN      NaN

           pillarId pillarName  \
0       pillar/news       News
1       pillar/arts       Arts
2      pillar/sport      Sport
3       pillar/news       News
4       pillar/news       News
…             …        …
11365           NaN        NaN
11366           NaN        NaN
11367           NaN        NaN
11368           NaN        NaN
11369           NaN        NaN

                                                content  keyword  \
0      Turkey has launched airstrikes against suspect…      NaN
1      Born in Busan, Turkey, trained in Warsaw and, …      NaN
2      Jamie Jackson’s report has landed. Here it is…      NaN
3      Invading Turks claim capture of Nicosia The Ob…      NaN
4      The living and the dead will soon be side by s…      NaN
…                                                   …      …
11365  Media should have warned us well in advance. T…  wrecked
11366  i feel directly attacked  i consider moonbin …  wrecked
11367  i feel directly attacked  i consider moonbin …  wrecked
11368  ok who remember "outcast" nd the "dora" au?? T…  wrecked
11369     Jake Corway wrecked while running 14th at IRP.  wrecked

                      location text  target
0                          NaN  NaN     NaN
1                          NaN  NaN     NaN
2                          NaN  NaN     NaN
3                          NaN  NaN     NaN
4                          NaN  NaN     NaN
…                        …   …      …
11365  Blue State in a red sea  NaN     0.0
11366             arohaonces  NaN     0.0
11367                         NaN     0.0
11368         auroraborealis  NaN     0.0
11369                      NaN  NaN     1.0
```

```
[29959 rows x 17 columns]
```

[18]:
```python
# Checking for the count of missing values before and after handling them.
print("Before handling missing values:")
print(df_news.isnull().sum())
print(f"Total missing values: {df_news.isnull().sum().sum()}")

# Handling missing values
df_news = check_and_handle_missing_values(df_news)

print("After handling missing values:")
print(df_news.isnull().sum())
print(f"Total missing values: {df_news.isnull().sum().sum()}")
```

```
Before handling missing values:
id                      0
type                29859
sectionId           29859
sectionName         29859
webPublicationDate  29859
webTitle            29859
webUrl              29859
apiUrl              29859
fields              29859
isHosted            29859
pillarId            29859
pillarName          29859
content              7613
keyword               248
location             9689
text                22346
target               3363
dtype: int64
Total missing values: 371708
Checking for missing values…

Missing Values Found:
type                29859
sectionId           29859
sectionName         29859
webPublicationDate  29859
webTitle            29859
webUrl              29859
apiUrl              29859
fields              29859
isHosted            29859
pillarId            29859
```

```
pillarName           29859
content               7613
keyword                248
location              9689
text                 22346
target                3363
dtype: int64

Total Missing Values: 371708
Filled missing values in 'type' with mode: article
Filled missing values in 'sectionId' with mode: world
Filled missing values in 'sectionName' with mode: World news
Filled missing values in 'webPublicationDate' with mode: 2023-02-06T13:04:44Z
Filled missing values in 'webTitle' with mode: A visual guide to the earthquakes
that hit Turkey and Syria
Filled missing values in 'webUrl' with mode:
https://www.theguardian.com/artanddesign/article/2024/jun/07/turkey-rejects-
claim-lord-elgin-had-permission-to-take-parthenon-marbles
Filled missing values in 'apiUrl' with mode:
https://content.guardianapis.com/artanddesign/article/2024/jun/07/turkey-
rejects-claim-lord-elgin-had-permission-to-take-parthenon-marbles
Filled missing values in 'isHosted' with mode: False
Filled missing values in 'pillarId' with mode: pillar/news
Filled missing values in 'pillarName' with mode: News
Filled missing values in 'content' with mode: 11-Year-Old Boy Charged With
Manslaughter of Toddler: Report: An 11-year-old boy has been charged with
manslaughter over the fatal sh…
Filled missing values in 'keyword' with mode: thunderstorm
Filled missing values in 'location' with mode: USA
Filled missing values in 'text' with mode: 11-Year-Old Boy Charged With
Manslaughter of Toddler: Report: An 11-year-old boy has been charged with
manslaughter over the fatal sh…
Filled missing values in 'target' with mean: 0.33
After handling missing values:
id                    0
type                  0
sectionId             0
sectionName           0
webPublicationDate    0
webTitle              0
webUrl                0
apiUrl                0
fields                0
isHosted              0
pillarId              0
pillarName            0
content               0
keyword               0
```

```
location              0
text                  0
target                0
dtype: int64
Total missing values: 0
```

Here we found that there were Total 371708 missing values in the dataset and after handling them by filling them with their Mean(Numerical Value) and Mode(Categorical Value), now their is no missing value.

[19]:
```
# Keeping only the required column of dataset for making further analysis
df_news = df_news[['content']]
```

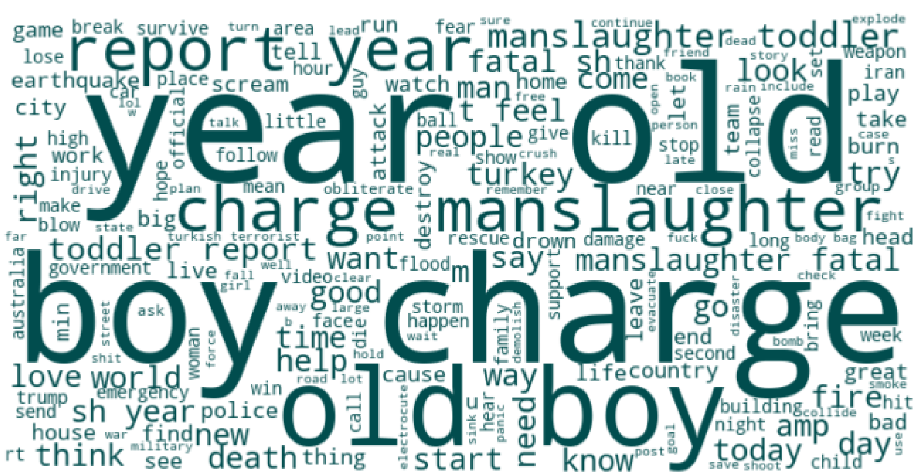
[20]:
```
# Preprocessing the data by lowercasing, removing stop words and punctuations␣
↪and lemmetizing the text.
nlp = spacy.load('en_core_web_sm')

# Ensure all entries in 'content' are strings
df_news['content'] = df_news['content'].fillna('').astype(str)

# Applying preprocessing
df_news['tokens'] = df_news['content'].apply(lambda x: nlp(x.lower()))
df_news['tokens'] = df_news['tokens'].apply(
    lambda x: [w.lemma_ for w in x if not w.is_stop and not w.is_punct and w.
↪lemma_ != ' ']
)
df_news['tokens_joined'] = df_news['tokens'].apply(lambda x: ' '.join(i for i␣
↪in x if i.isalnum() or i.isnumeric()))
df_news['tokens'] = df_news['tokens_joined'].apply(lambda x: nlp(x.lower()))

print(df_news.head())
```

```
                                             content  \
0  Turkey has launched airstrikes against suspect…
1  Born in Busan, Turkey, trained in Warsaw and, …
2  Jamie Jackson's report has landed. Here it is…
3  Invading Turks claim capture of Nicosia The Ob…
4  The living and the dead will soon be side by s…

                                              tokens  \
0  (turkey, launch, airstrike, suspect, kurdish, …
1  (bear, busan, turkey, train, warsaw, 2016, bas…
2  (jamie, jackson, report, land, thank, read, mb…
3  (invade, turk, claim, capture, nicosia, observ…
4  (living, dead, soon, outskirt, antakya, new, g…

                                       tokens_joined
0  turkey launch airstrike suspect kurdish milita…
```

```
1    bear busan turkey train warsaw 2016 base new y…
2    jamie jackson report land thank read mbm night…
3    invade turk claim capture nicosia observer 21 …
4    living dead soon outskirt antakya new governme…
```

[21]:

```python
# Wordcloud in the shape of  map
image = Image.open("tse.png")
mask = np.array(image)

x = " ".join(i for i in df_news.tokens_joined)
wc = WordCloud(background_color = "white",colormap="PuBu", repeat = True, mask␣
 ↪= mask).generate(x)

plt.figure(figsize=(10,8 ))
dark_teal_color = (1, 77, 78)
plt.imshow(wc.recolor(color_func=lambda *args, **kwargs: dark_teal_color),␣
 ↪interpolation='bilinear')
plt.axis("off")
plt.show()
```



[22]:

```python
# Overall sentiments classification in the dataset using TextBlob

def sentiment_score(text):
    analysis = TextBlob(text)
    sentiment = analysis.sentiment.polarity
    return sentiment
```

```
def sentiment_classification(score):
    if score > 0.1:
        sentiment_label = "positive"
    elif score < -0.1:
        sentiment_label = "negative"
    else:
        sentiment_label = "neutral"
    return sentiment_label

df_news['sentiment score'] = df_news['tokens_joined'].apply(sentiment_score)
df_news['sentiment polarity'] = df_news['sentiment score'].
 ↪apply(sentiment_classification)
df_news.head()
```

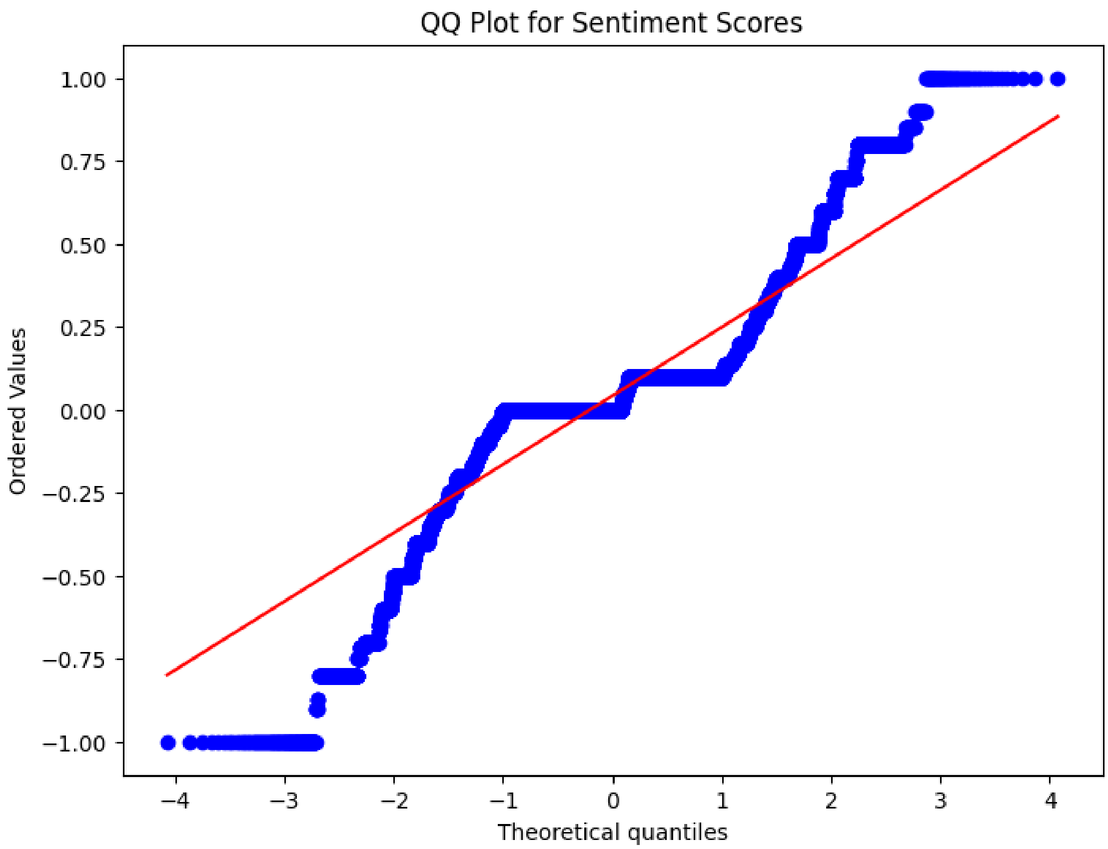
[22]:

```
                                             content  \
0  Turkey has launched airstrikes against suspect…
1  Born in Busan, Turkey, trained in Warsaw and, …
2  Jamie Jackson’s report has landed. Here it is…
3  Invading Turks claim capture of Nicosia The Ob…
4  The living and the dead will soon be side by s…

                                              tokens  \
0  (turkey, launch, airstrike, suspect, kurdish, …
1  (bear, busan, turkey, train, warsaw, 2016, bas…
2  (jamie, jackson, report, land, thank, read, mb…
3  (invade, turk, claim, capture, nicosia, observ…
4  (living, dead, soon, outskirt, antakya, new, g…

                                       tokens_joined  sentiment score  \
0  turkey launch airstrike suspect kurdish milita…        -0.029901
1  bear busan turkey train warsaw 2016 base new y…        -0.065027
2  jamie jackson report land thank read mbm night…         0.108458
3  invade turk claim capture nicosia observer 21 …        -0.008499
4  living dead soon outskirt antakya new governme…         0.018424

  sentiment polarity
0            neutral
1            neutral
2           positive
3            neutral
4            neutral
```

[23]:

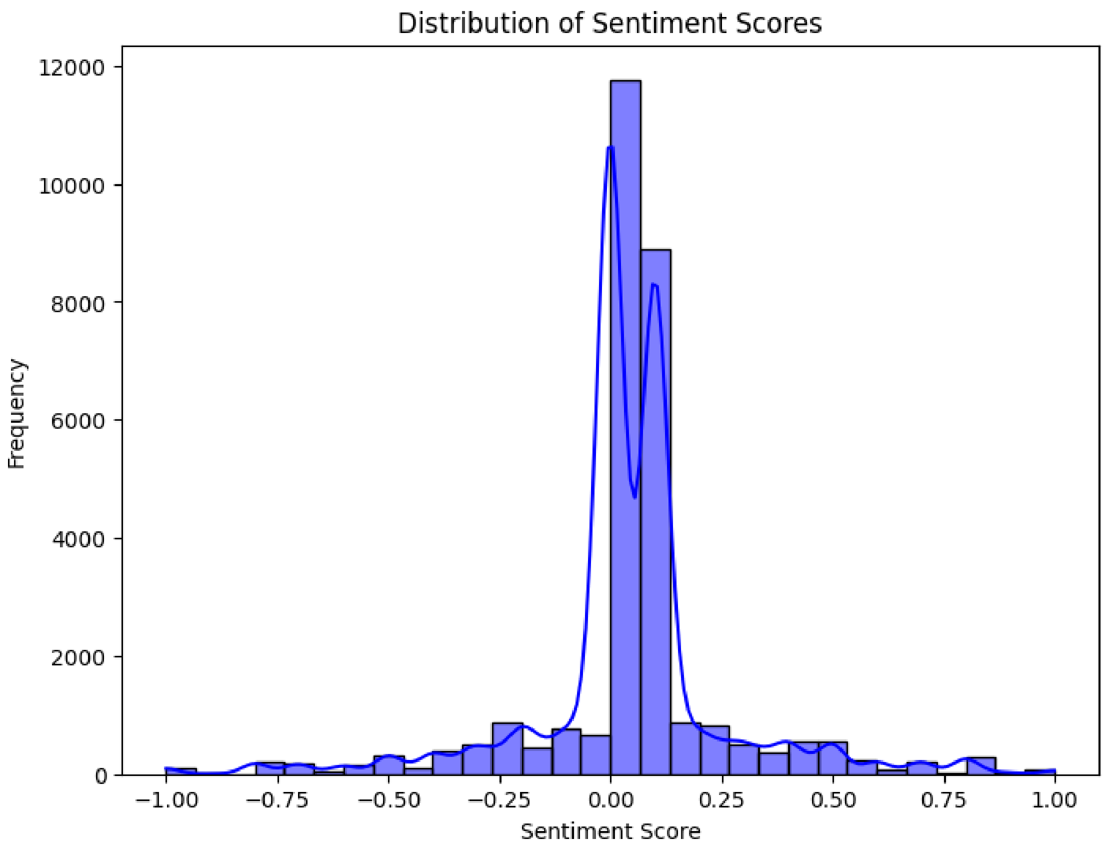
```
# QQ Plot for Sentiment Scores
import scipy.stats as stats

plt.figure(figsize=(8, 6))
stats.probplot(df_news['sentiment score'], dist="norm", plot=plt)
```

```
plt.title("QQ Plot for Sentiment Scores")
plt.show()
```

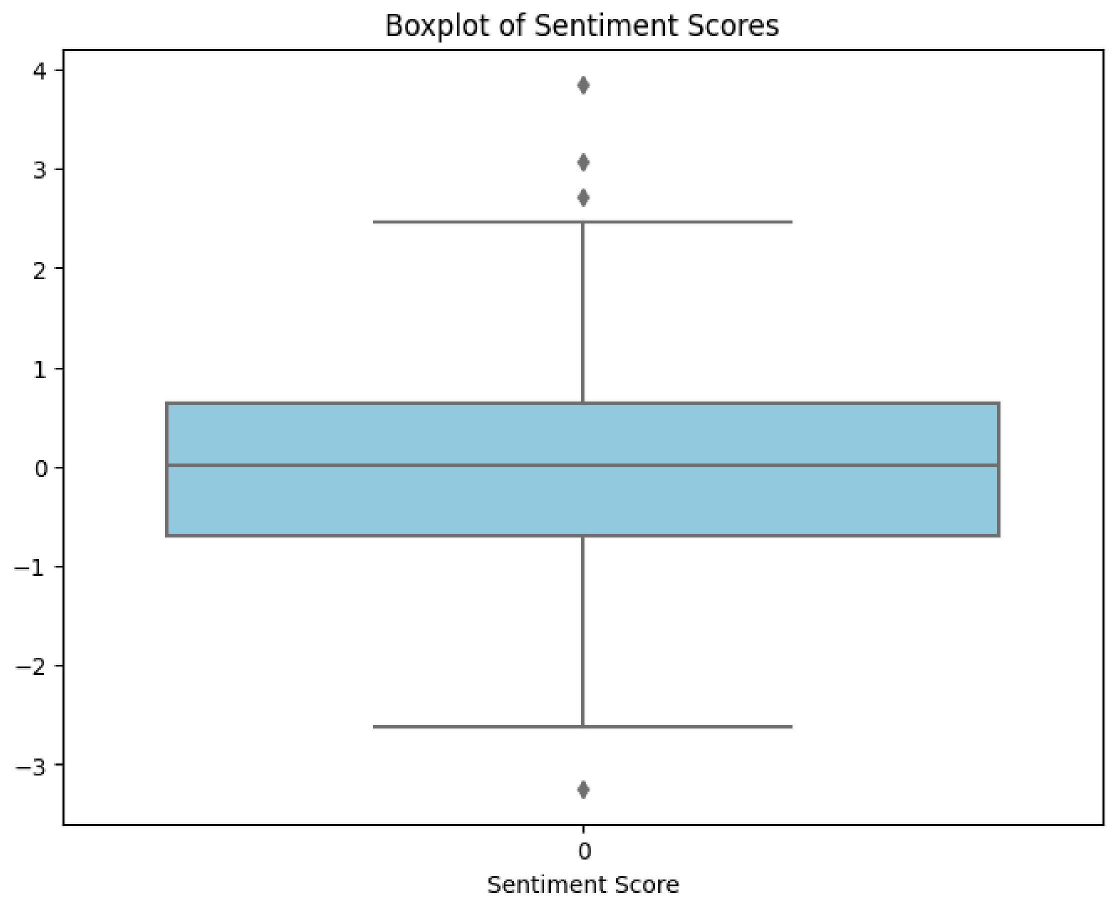


[24]:

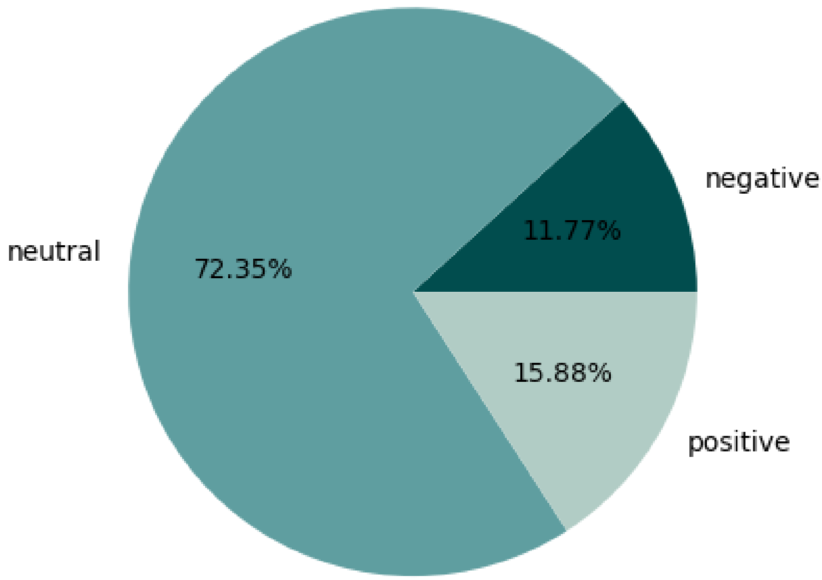
```
# Histogram plot for Sentiment Score Distribution
plt.figure(figsize=(8, 6))
sns.histplot(df_news['sentiment score'], kde=True, bins=30, color="blue")
plt.title("Distribution of Sentiment Scores")
plt.xlabel("Sentiment Score")
plt.ylabel("Frequency")
plt.show()
```



[25]:
```
# Boxplot for Outlier Detection
# Generating a small sample dataset for testing the boxplot visualization as␣
↪kernel restart for large sample
np.random.seed(42)
test_data = {
    "sentiment score": np.random.normal(loc=0, scale=1, size=500)  # 500␣
↪samples with a normal distribution
}
df_sample = pd.DataFrame(test_data)

# Plotting the boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(df_sample['sentiment score'], color="skyblue")
plt.title("Boxplot of Sentiment Scores")
plt.xlabel("Sentiment Score")
plt.show()
```



[26]:
```
# Sentiment polarity distribution i.e.(Neutral, Positive, Negative)
df_news.groupby('sentiment polarity').size().plot(kind='pie', autopct='%.
 ↪2f%%',colors = ["#014d4e","#5F9EA0","#b1ccc5"])
plt.figure(figsize=(15,15))
plt.show()
```




```
<Figure size 1500x1500 with 0 Axes>
```

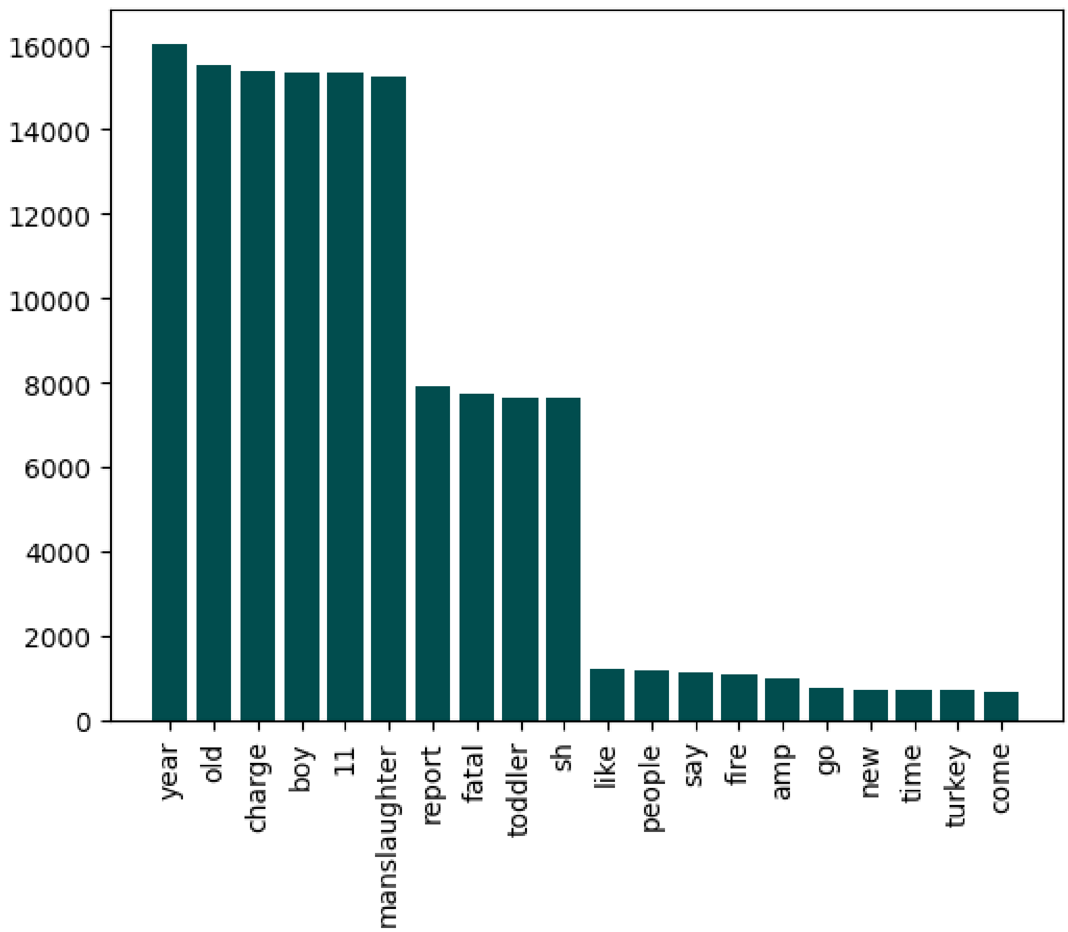
[27]:

```
# Calculating the top 20 frequent words in dataset
df_freq = pd.DataFrame.from_dict(dict(Counter(' '.
 ↪join(df_news['tokens_joined']).split(" "))),orient='index').reset_index()
df_freq.rename(columns={0:'counts','index':'words'},inplace=True)
df_freq.sort_values(by='counts',ascending=False,inplace=True)
df_freq = df_freq.head(20)
print(df_freq)
```

```
              words  counts
342            year   16042
225             old   15531
2582         charge   15367
1866            boy   15359
528              11   15352
9047   manslaughter   15256
35           report    7923
9048          fatal    7726
5949        toddler    7645
9049             sh    7638
256            like    1217
23           people    1159
26              say    1122
```

```
111           fire    1059
9073           amp    1001
257             go     758
61             new     716
121           time     703
0           turkey     702
458           come     653
```

[28]:
```
# Bar plot of word frequencies
plt.bar(df_freq['words'],df_freq['counts'],color = "#014d4e")
plt.xticks(rotation=90)
plt.grid(False)
plt.show()
```



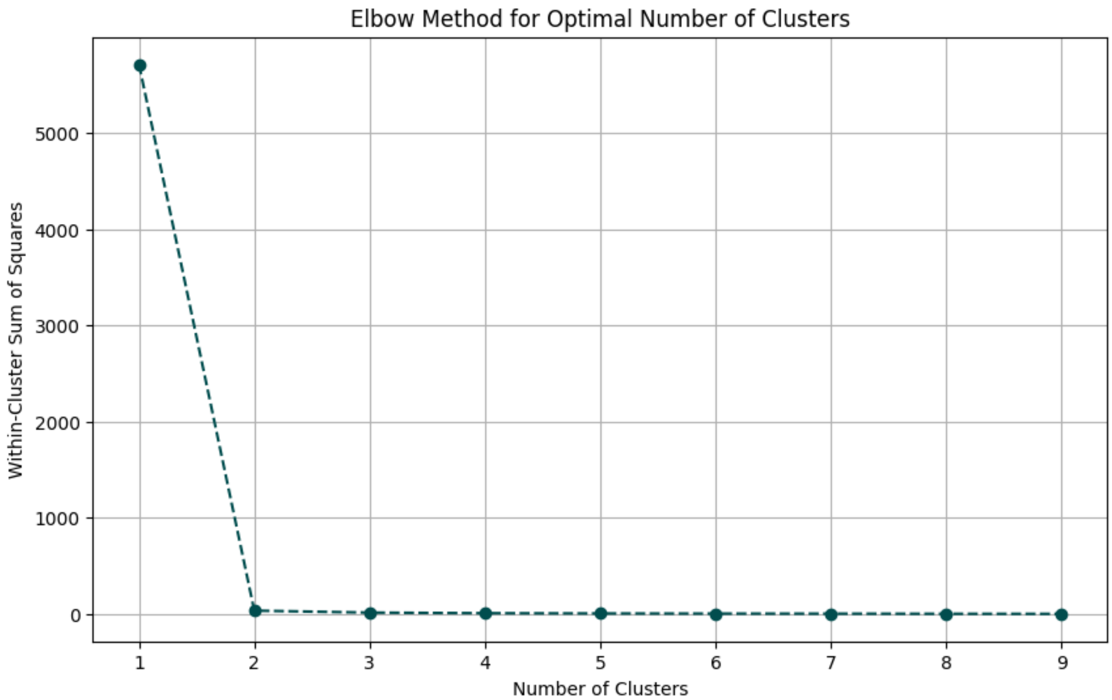
[29]:
```
# Vectorizing the words in the dataset
vectorizer = TfidfVectorizer(stop_words="english")
tokens_vectorized = vectorizer.fit_transform(df_news['tokens_joined'])
tokens_vectorized.shape
```

[29]: (29959, 28979)

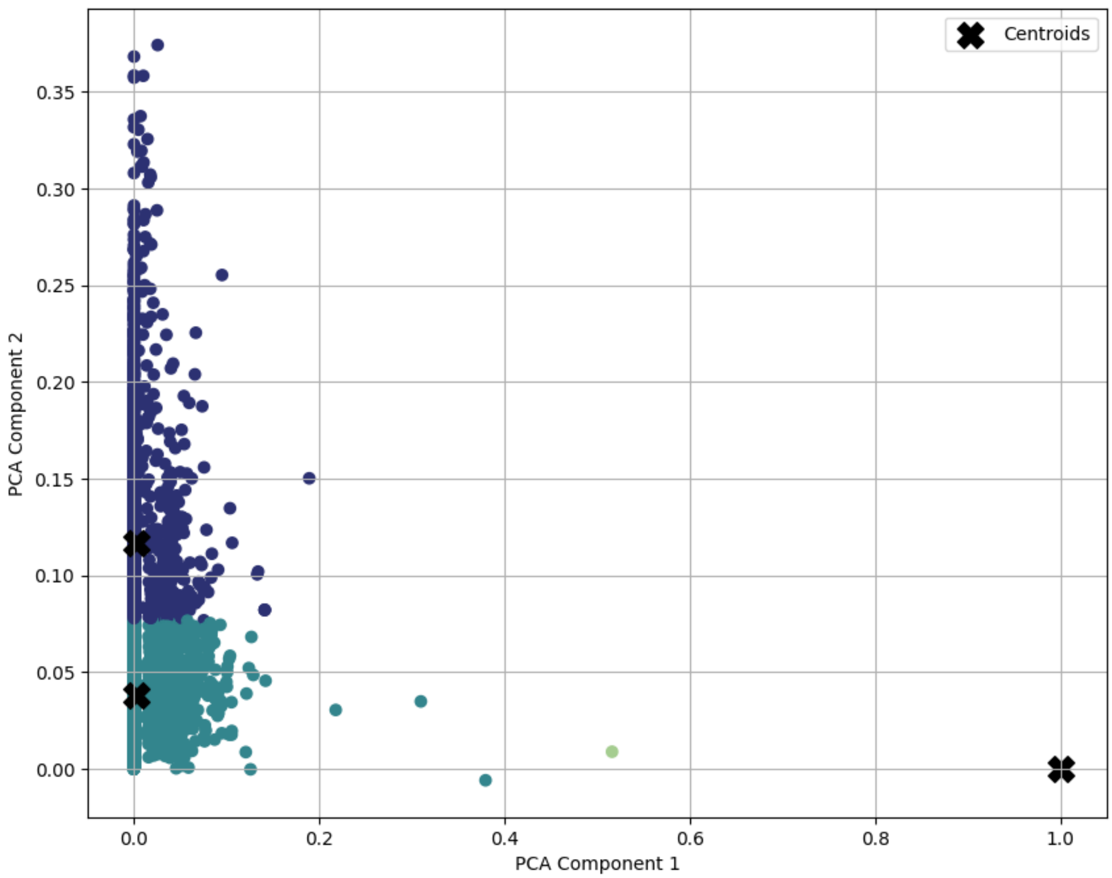
[30]:
```
# Using TruncatedSVD to reduce dimensions to 2 instead of PCA beacuse it crash␣
↪the kernel
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=2)
reduced_tfidf = svd.fit_transform(tokens_vectorized)

# Checking the shape of the reduced data
print("Shape of reduced data:", reduced_tfidf.shape)
```

Shape of reduced data: (29959, 2)

[31]:
```
# Determining the optimal number of clusters using the Elbow method
wcss = []
cluster_range = range(1, 10)

for k in cluster_range:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(reduced_tfidf)
    wcss.append(kmeans.inertia_)

# Plot the Elbow method
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, wcss, marker='o', linestyle='--',color = "#014d4e")
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.grid(True)
plt.show()
```



[32]:
```
# Performing KMeans clustering with 3 clusters as elbow point was observed at 3␣
↪in above plot
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(reduced_tfidf)

# Plot the clusters
plt.figure(figsize=(10, 8))
plt.scatter(reduced_tfidf[:, 0], reduced_tfidf[:, 1], c=clusters, cmap='crest')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
↪s=200, c='black', marker='X', label='Centroids')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

plt.legend()
plt.grid(True)
plt.show()
```



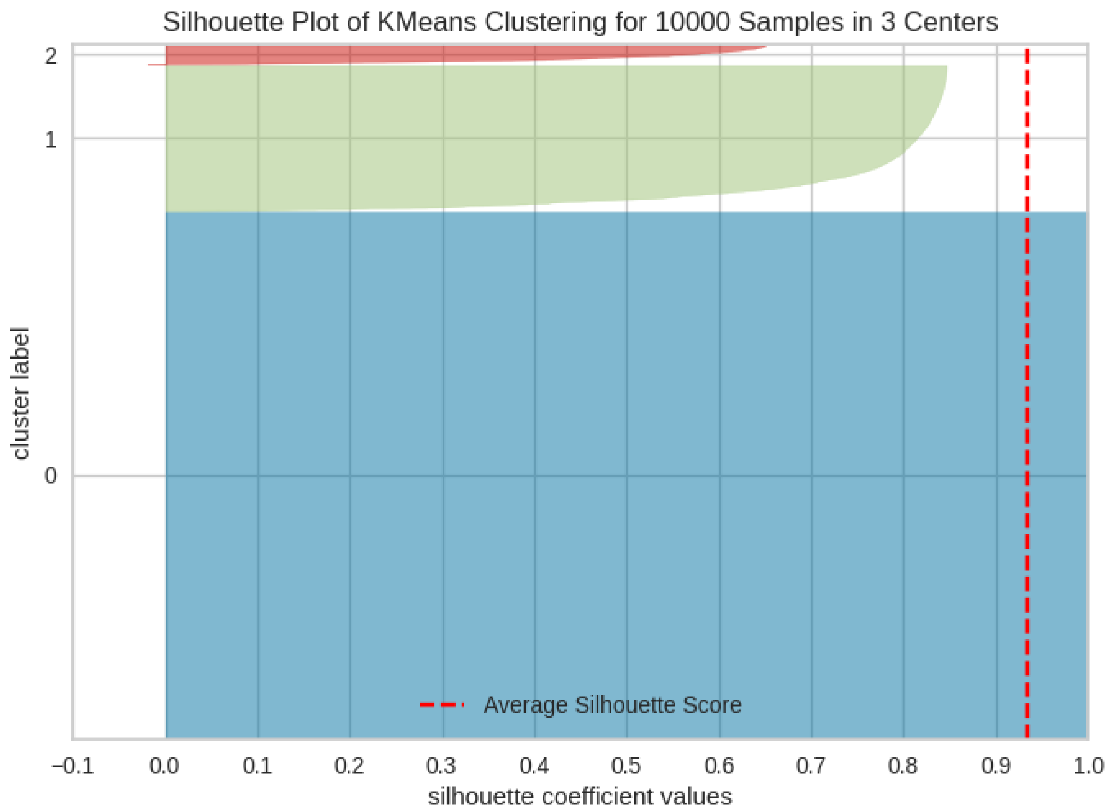
[33]:
```
# Adding a column of cluster labels to the dataframe
df_news['clusters'] = clusters
```

[34]:
```
# Getting top 20 words from each cluster
def get_top_keywords_sparse(data, clusters, labels, n_terms):
    # Creating a DataFrame with the mean feature values per cluster
    cluster_means = []
    for cluster_id in np.unique(clusters):
        cluster_indices = np.where(clusters == cluster_id)[0]  # Getting rows␣
↪belonging to the cluster
        cluster_mean = data[cluster_indices].mean(axis=0)  # Sparsing matrix␣
↪operation
        cluster_means.append(cluster_mean)

    # Converting cluster means to a dense array for processing
    cluster_means = np.vstack(cluster_means)

    key = []
    value = []
```

```
    for i, row in enumerate(cluster_means):
        # Getting the indices of the top n_terms features
        top_indices = np.argsort(row.A1)[-n_terms:]  # `A1` for sparse matrix␣
 ↪to dense array
        key.append(f'Cluster {i}')
        value.append(','.join([labels[t] for t in top_indices]))
        print(f'\nCluster {i}')
        print(','.join([labels[t] for t in top_indices]))

    return pd.DataFrame({'Cluster': key, 'Top Keywords': value})

df_keywords = get_top_keywords_sparse(tokens_vectorized, clusters, vectorizer.
 ↪get_feature_names_out(), 20)
print(df_keywords)
```

```
Cluster 0
fiine,figure,figment,fighters,fightextremism,fightin,fightâêterrorism,fighting,f
ightthefire,shooting,report,fatal,toddler,sh,year,old,boy,charge,11,manslaughter

Cluster 1
want,collapse,bomb,drown,good,news,cause,crash,day,attack,come,love,need,time,di
saster,death,emergency,storm,amp,new

Cluster 2
bag,drown,building,good,need,day,come,feel,time,look,body,scream,say,burn,kill,t
hink,know,amp,people,like
     Cluster                                       Top Keywords
0   Cluster 0  fiine,figure,figment,fighters,fightextremism,f…
1   Cluster 1  want,collapse,bomb,drown,good,news,cause,crash…
2   Cluster 2  bag,drown,building,good,need,day,come,feel,tim…
```

[35]:
```
# Plotting silhoutte score for 3 clusters
from yellowbrick.cluster import SilhouetteVisualizer

# Instantiating KMeans with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)

# Creating a silhouette visualizer
visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick')

# Using a subset if necessary to avoid memory issues
subset = reduced_tfidf[:10000]  # Adjust based on dataset size

visualizer.fit(subset)
visualizer.show()
print("Silhouette Score:", visualizer.silhouette_score_)
```

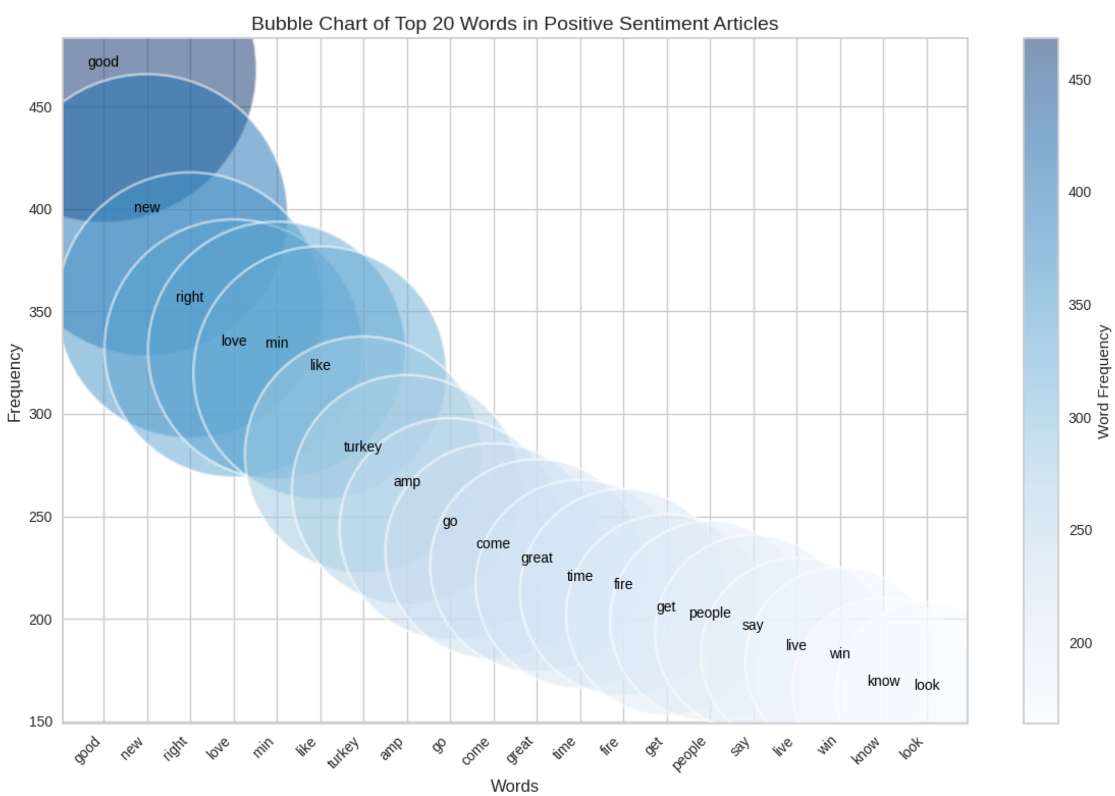



```
Silhouette Score: 0.9336966329741258
```

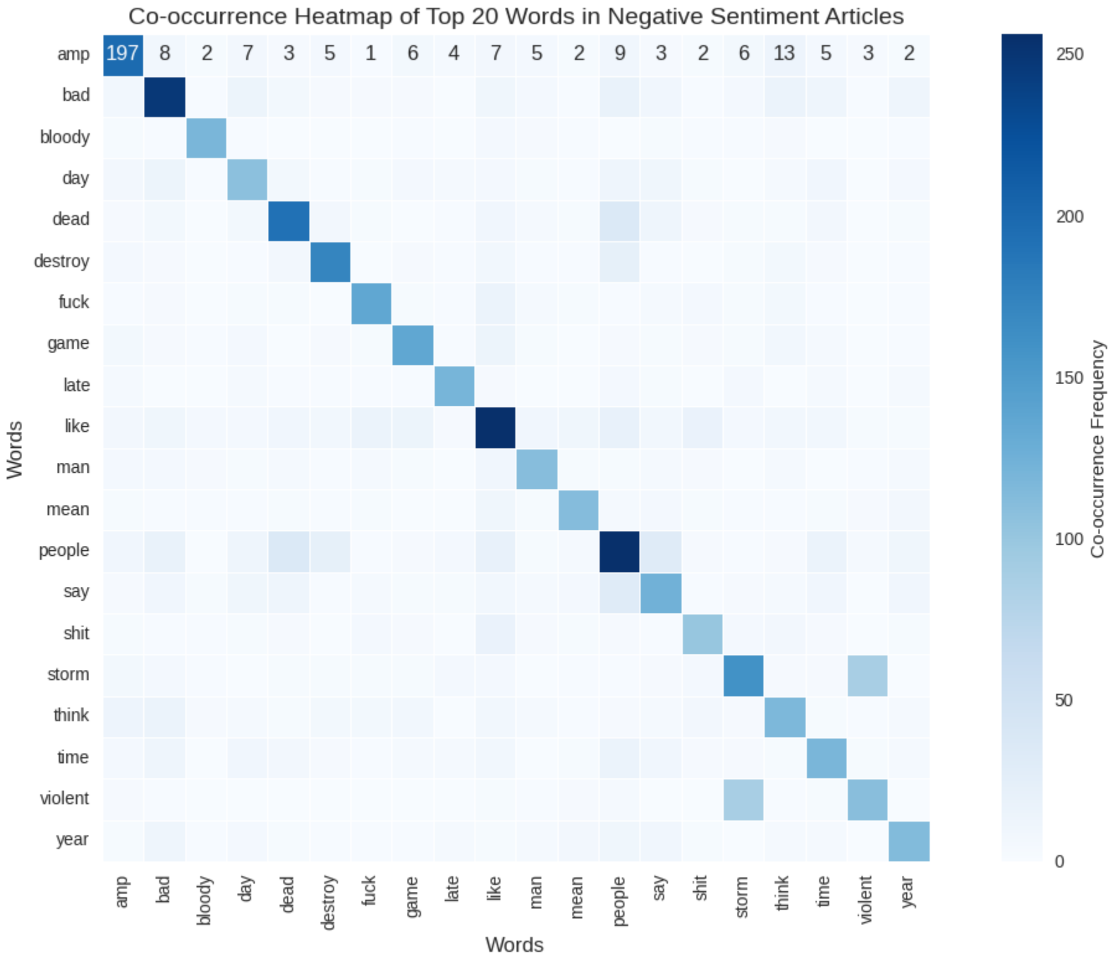
[36]:

```
# Bubble plot for the positive sentiment

# Aggregating positive sentiment tokens
positive_tokens = df_news.loc[df_news['sentiment polarity'] == "positive",␣
 ↪'tokens_joined'].str.split().sum()
word_freq = Counter(positive_tokens)
top_words = word_freq.most_common(20)

# Converting to DataFrame for easier handling
df_words = pd.DataFrame(top_words, columns=['words', 'counts'])

# Creating the bubble chart
plt.figure(figsize=(12, 8))
bubble_sizes = df_words['counts'] * 100  # Scaling bubble sizes for better␣
 ↪visualization
scatter = plt.scatter(
    range(len(df_words)),
    df_words['counts'],
    s=bubble_sizes,
```

```
    alpha=0.5,
    c=df_words['counts'],
    cmap='Blues',
    edgecolors="w",
    linewidth=2
)

# Customizing plot aesthetics
plt.xticks(range(len(df_words)), df_words['words'], rotation=45, ha='right',␣
 ↪fontsize=10)
plt.title("Bubble Chart of Top 20 Words in Positive Sentiment Articles",␣
 ↪fontsize=14)
plt.xlabel("Words", fontsize=12)
plt.ylabel("Frequency", fontsize=12)

# Annotating each bubble with the corresponding word
for i, (word, count) in enumerate(zip(df_words['words'], df_words['counts'])):
    plt.text(i, count, word, ha='center', va='bottom', fontsize=10,␣
 ↪color='black')

plt.colorbar(scatter, label='Word Frequency')  # Add a colorbar for frequency␣
 ↪values
plt.tight_layout()
plt.show()
```



[37]:
```python
# Co-occurrence heatmap for the positive sentiment

# Extracting the tokens for negative sentiment
negative_tokens = df_news.loc[df_news['sentiment polarity'] == "negative",␣
↪'tokens_joined']

# Use CountVectorizer to get the co-occurrence matrix of top 20 words
vectorizer = CountVectorizer(max_features=20, stop_words="english")  # Top 20␣
↪most frequent words
X = vectorizer.fit_transform(negative_tokens)

# Computing the co-occurrence matrix
cooccurrence_matrix = (X.T * X).toarray()

# Getting the top 20 words
words = vectorizer.get_feature_names_out()

# Creating a DataFrame for the co-occurrence matrix
df_cooccurrence = pd.DataFrame(cooccurrence_matrix, index=words, columns=words)

# Plotting the co-occurrence heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(
    df_cooccurrence,
    annot=True,
    cmap="Blues",
    fmt="d",
    linewidths=0.5,
    square=True,
    cbar_kws={'label': 'Co-occurrence Frequency'}
)
plt.title("Co-occurrence Heatmap of Top 20 Words in Negative Sentiment␣
↪Articles", fontsize=14)
plt.xlabel("Words", fontsize=12)
plt.ylabel("Words", fontsize=12)
plt.tight_layout()
plt.show()
```



[38]:
```
# Preparing data for apriori
from scipy.sparse import csr_matrix

# Taking a sample of 9000 rows from the dataset as it restarts the kernel again␣
↪and again
df_sample = df_news.sample(n=9000, random_state=42)
data = list(df_sample['tokens_joined'].apply(lambda x: x.split(" ")))

# Using TransactionEncoder for transformation
a = TransactionEncoder()

# Transforming data with sparse matrix option
a_data = a.fit(data).transform(data, sparse=True)  # Ensure sparse␣
↪transformation
sparse_data = csr_matrix(a_data)   # Convert to a CSR sparse matrix

# Converting sparse matrix to a DataFrame
```

```python
df2 = pd.DataFrame.sparse.from_spmatrix(sparse_data, columns=a.columns_)
print(df2.head())
```

```
      0  01  011220  02  03  034  04  05  06  …                   \
0  0  0   0       0   0   0    0   0   0   0  …    0     0  0     0    0
1  0  0   0       0   0   0    0   0   0   0  …    0     0  0     0    0
2  0  0   0       0   0   0    0   0   0   0  …    0     0  0     0    0
3  0  0   0       0   0   0    0   0   0   0  …    0     0  0     0    0
4  0  0   0       0   0   0    0   0   0   0  …    0     0  0     0    0


0      0     0       0    0         0
1      0     0       0    0         0
2      0     0       0    0         0
3      0     0       0    0         0
4      0     0       0    0         0

[5 rows x 14436 columns]
```

[39]:
```python
# Implementing Apriori Algorithm
from mlxtend.frequent_patterns import apriori, association_rules

# Implementing Apriori Algorithm
df3 = apriori(df2, min_support=0.2, use_colnames=True, verbose=1)

# Generate association rules
rules = association_rules(df3, metric="lift", min_threshold=1,␣
 ↪num_itemsets=None)

# Display the first 10 rules
print(rules.head(10))
```

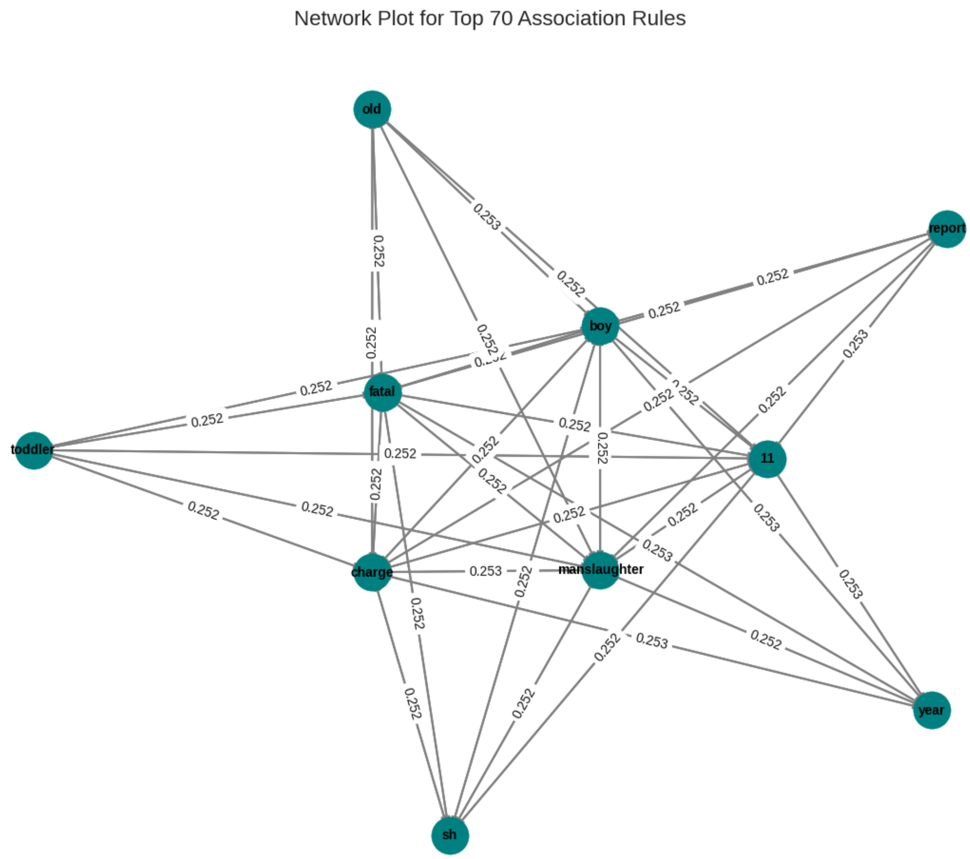
```
Processing 10 combinations | Sampling itemset size 106
      antecedents     consequents  antecedent support  consequent support  \
0           (boy)            (11)            0.256111            0.254778
1            (11)           (boy)            0.254778            0.256111
2        (charge)            (11)            0.256222            0.254778
3            (11)        (charge)            0.254778            0.256222
4         (fatal)            (11)            0.255667            0.254778
5            (11)         (fatal)            0.254778            0.255667
6            (11)  (manslaughter)            0.254778            0.252556
7  (manslaughter)            (11)            0.252556            0.254778
8           (old)            (11)            0.259333            0.254778
9            (11)           (old)            0.254778            0.259333

    support  confidence      lift  representativity  leverage  conviction  \
0  0.252444    0.985683  3.868796               1.0  0.187193   52.052643
```

```
1  0.252444    0.990842  3.868796              1.0  0.187193   81.225582
2  0.252333    0.984822  3.865416              1.0  0.187054   49.099498
3  0.252333    0.990406  3.865416              1.0  0.187054   77.521929
4  0.252111    0.986093  3.870404              1.0  0.186973   53.586135
5  0.252111    0.989533  3.870404              1.0  0.186973   71.114847
6  0.252333    0.990406  3.921536              1.0  0.187988   77.904096
7  0.252333    0.999120  3.921536              1.0  0.187988  846.945056
8  0.252444    0.973436  3.820726              1.0  0.186372   28.054011
9  0.252444    0.990842  3.820726              1.0  0.186372   80.873746

   zhangs_metric   jaccard  certainty  kulczynski
0       0.996818  0.976784   0.980789    0.988262
1       0.995034  0.976784   0.987689    0.988262
2       0.996663  0.975515   0.979633    0.987614
3       0.994731  0.975515   0.987100    0.987614
4       0.996367  0.975914   0.981338    0.987813
5       0.995178  0.975914   0.985938    0.987813
6       0.999699  0.989542   0.987164    0.994763
7       0.996727  0.989542   0.998819    0.994763
8       0.996764  0.964756   0.964354    0.982139
9       0.990670  0.964756   0.987635    0.982139
```

[40]:

```python
# Network plot for top 70 rules

# Creating a directed graph
G = nx.DiGraph()

# Adding edges with weights for the top 70 rules
for index, rule in rules.head(70).iterrows():
    antecedents = ', '.join(list(rule['antecedents']))  # Joining antecedents
    consequents = ', '.join(list(rule['consequents']))  # Joining consequents
    support = round(rule['support'], 3)
    confidence = round(rule['confidence'], 2)

    # Adding edge with attributes
    G.add_edge(antecedents, consequents, weight=support, confidence=confidence)

plt.figure(figsize=(15, 12))

# Using spring layout for better positioning
pos = nx.spring_layout(G, seed=42)

# Extracting edge labels
edge_labels = nx.get_edge_attributes(G, 'weight')

# Drawing nodes, edges, and labels
nx.draw_networkx_nodes(G, pos, node_size=800, node_color='teal')
```

```
nx.draw_networkx_edges(G, pos, width=1.5, edge_color='gray')
nx.draw_networkx_labels(G, pos, font_size=10, font_color='black',␣
 ↪font_weight='bold')

# Adding edge labels (e.g., support values)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

plt.title("Network Plot for Top 70 Association Rules", fontsize=16)
plt.axis('off')
plt.show()
```



[41]:
```
# Implementing Emotion detection using NRCLex library
from nrclex import NRCLex
import nltk

# Function for extracting top emotions
def extract_emotions(text):
    """Returns a dictionary of the emotions detected in the text."""
```

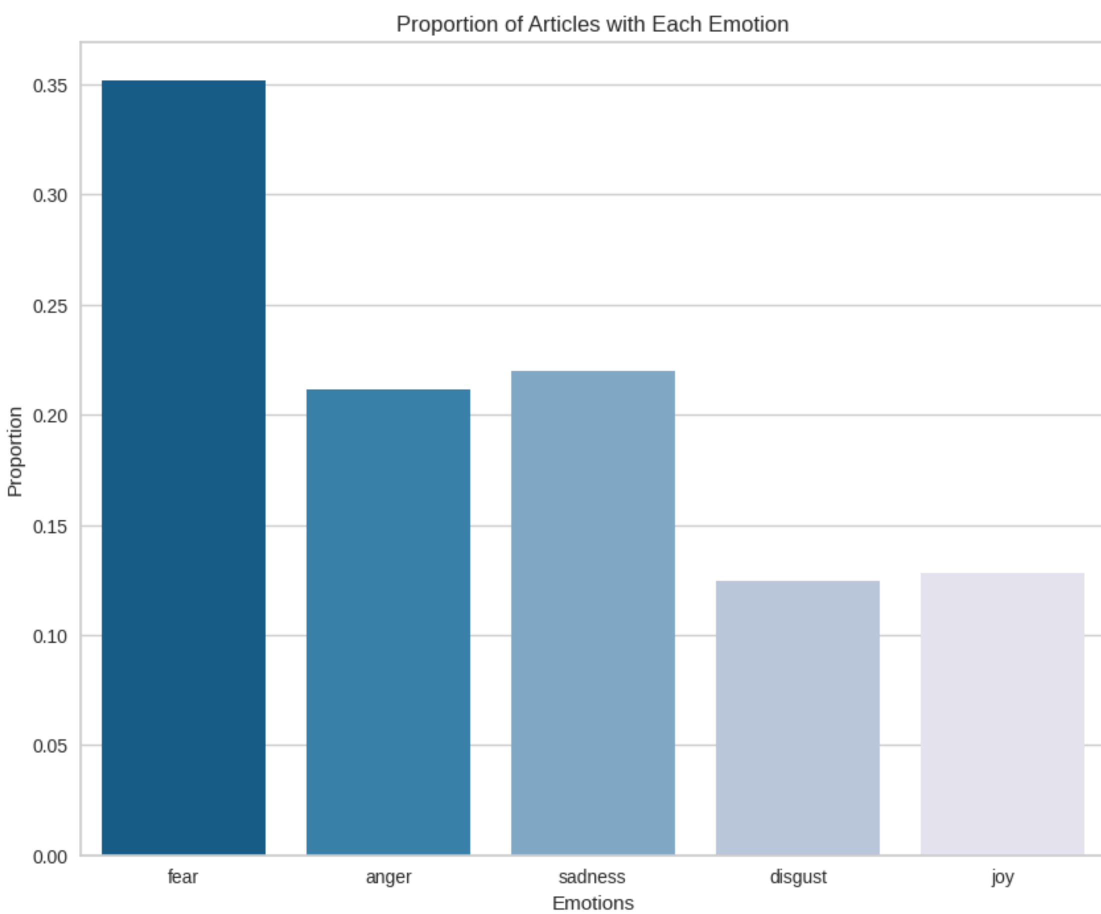
```
    emotion_data = NRCLex(text).top_emotions
    return {emotion: True for emotion, _ in emotion_data}

# Function to classify specific emotions
def classify_emotions(text, target_emotions=["fear", "anger", "sadness",␣
 ↪"disgust", "joy"]):
    """
    Detects specific emotions in text and returns a dictionary of presence (1␣
 ↪or 0).
    """
    detected_emotions = extract_emotions(text)
    return {emotion: int(detected_emotions.get(emotion, False)) for emotion in␣
 ↪target_emotions}

# Applying the classifier to the dataset
emotions = ["fear", "anger", "sadness", "disgust", "joy"]

# Extracting and expand results into columns
df_emotions = df_news['tokens_joined'].apply(lambda s: classify_emotions(s,␣
 ↪emotions)).apply(pd.Series)

# Merging the results with the original DataFrame
df_news = pd.concat([df_news, df_emotions], axis=1)
print(df_news.head())
```

```
                                             content  \
0  Turkey has launched airstrikes against suspect…
1  Born in Busan, Turkey, trained in Warsaw and, …
2  Jamie Jackson’s report has landed. Here it is…
3  Invading Turks claim capture of Nicosia The Ob…
4  The living and the dead will soon be side by s…

                                              tokens  \
0  (turkey, launch, airstrike, suspect, kurdish, …
1  (bear, busan, turkey, train, warsaw, 2016, bas…
2  (jamie, jackson, report, land, thank, read, mb…
3  (invade, turk, claim, capture, nicosia, observ…
4  (living, dead, soon, outskirt, antakya, new, g…

                                       tokens_joined  sentiment score  \
0  turkey launch airstrike suspect kurdish milita…        -0.029901
1  bear busan turkey train warsaw 2016 base new y…        -0.065027
2  jamie jackson report land thank read mbm night…         0.108458
3  invade turk claim capture nicosia observer 21 …        -0.008499
4  living dead soon outskirt antakya new governme…         0.018424

  sentiment polarity  clusters  fear  anger  sadness  disgust  joy
```

0	neutral	2	0	0	0	0	0
1	neutral	2	0	0	0	0	0
2	positive	2	0	0	0	0	0
3	neutral	2	0	0	0	0	0
4	neutral	2	0	0	0	0	0

[42]:

```
# Barplots for proportion of articles having each emotion

# Calculating proportions for each emotion
proportions = df_emotions.mean()

# Creating a barplot to visualize the proportions
plt.figure(figsize=(10,8))
sns.barplot(x=proportions.index, y=proportions.values, palette="PuBu_r")

plt.xlabel('Emotions')
plt.ylabel('Proportion')
plt.title('Proportion of Articles with Each Emotion')
plt.show()
```



3 Model Implementation

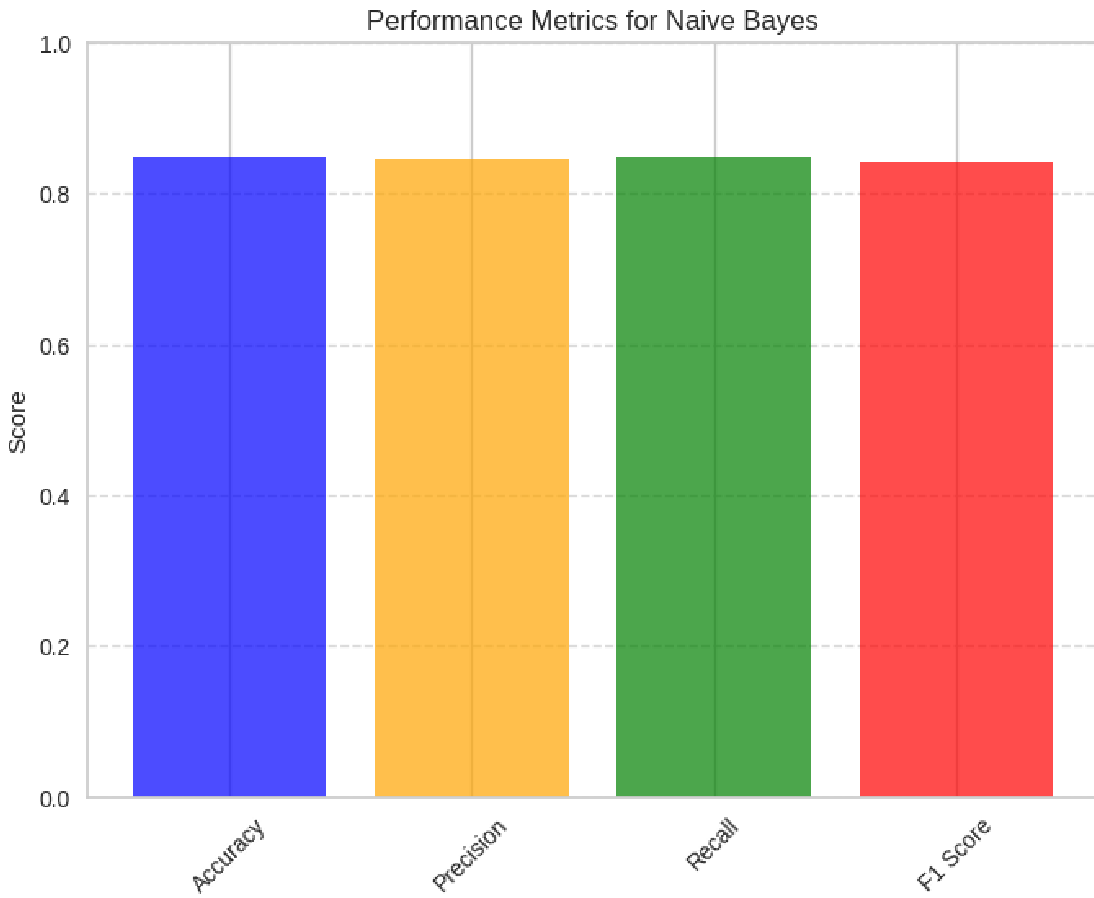
[43]:
```python
# Ensuring preprocessing is complete, including TF-IDF vectorization and␣
↪train-test splitting.

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# Assuming df_news['tokens_joined'] contains the preprocessed text
text_data = df_news['tokens_joined']  # Update column name as needed
labels = df_news['sentiment polarity']  # Sentiment classification labels

# Converting text data to numerical features
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X = tfidf_vectorizer.fit_transform(text_data)

# Train-test split for supervised models
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2,␣
↪random_state=42)
```

3.1 1.Naive Bayes

[44]:
```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.metrics import confusion_matrix

nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

print("Naive Bayes Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Precision:", precision_score(y_test, y_pred_nb, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_nb, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_nb, average='weighted'))

# Metrics for Naive Bayes
naive_bayes_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_nb),
    "Precision": precision_score(y_test, y_pred_nb, average='weighted'),
    "Recall": recall_score(y_test, y_pred_nb, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_nb, average='weighted')
}

# Plotting Naive Bayes metrics
plt.figure(figsize=(8, 6))
```

```
plt.bar(naive_bayes_metrics.keys(), naive_bayes_metrics.values(),␣
 ↪color=['blue', 'orange', 'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Naive Bayes")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Naive Bayes Results:
Accuracy: 0.8484646194926568
Precision: 0.8450326559648338
Recall: 0.8484646194926568
F1 Score: 0.8419170237509482
```



3.2 2.Logistic Regression

[45]:
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)

print("Logistic Regression Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Precision:", precision_score(y_test, y_pred_lr, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_lr, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_lr, average='weighted'))

# Metrics for Logistic Regression
logistic_regression_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_lr),
    "Precision": precision_score(y_test, y_pred_lr, average='weighted'),
    "Recall": recall_score(y_test, y_pred_lr, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_lr, average='weighted')
}

# Plotting Logistic Regression metrics
plt.figure(figsize=(8, 6))
plt.bar(logistic_regression_metrics.keys(), logistic_regression_metrics.
 ↪values(), color=['blue', 'orange', 'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Logistic Regression")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

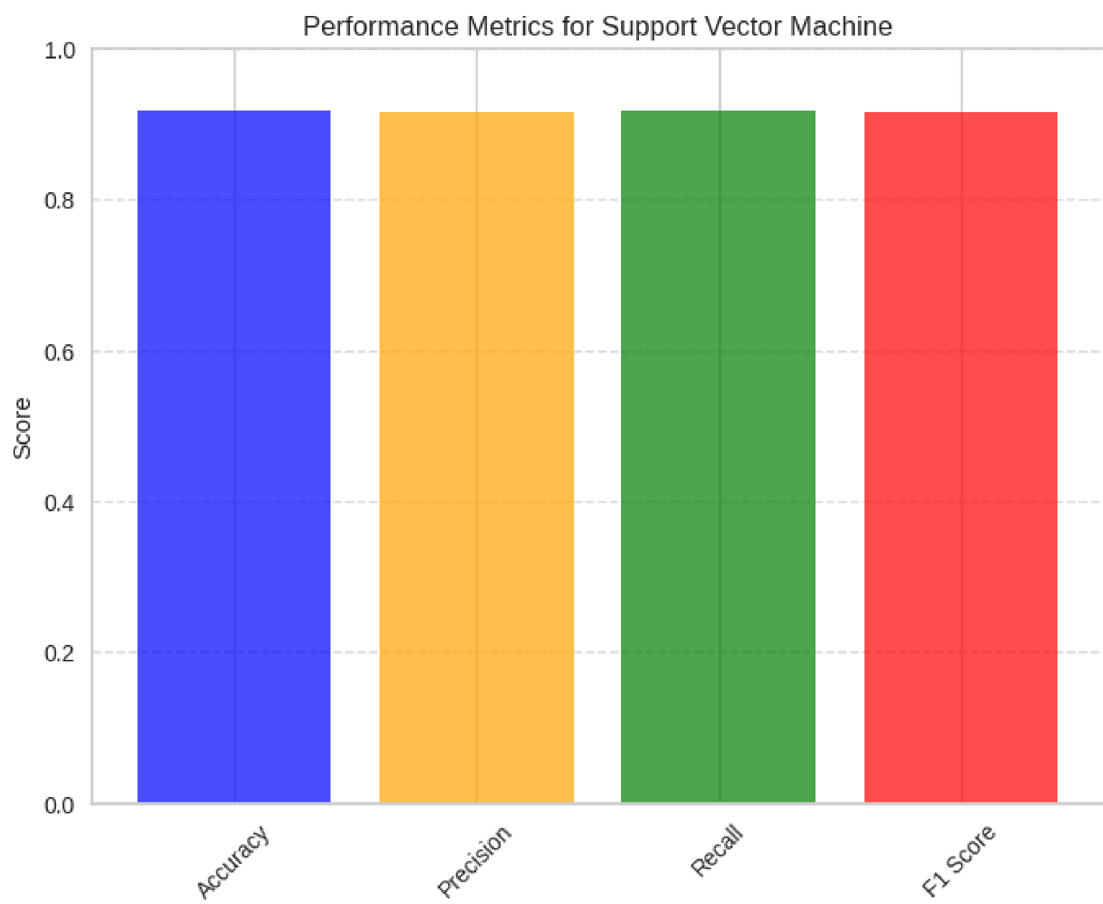
```
Logistic Regression Results:
Accuracy: 0.8768357810413885
Precision: 0.8756392961032513
Recall: 0.8768357810413885
F1 Score: 0.8700515238145119
```

3.3 3. Support Vector Machine (SVM)

[46]:
```python
from sklearn.svm import SVC

svm_model = SVC(kernel='linear', C=1)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)

print("SVM Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_svm, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_svm, average='weighted'))

# Metrics for SVM
svm_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_svm),
    "Precision": precision_score(y_test, y_pred_svm, average='weighted'),
```

```
    "Recall": recall_score(y_test, y_pred_svm, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_svm, average='weighted')
}

# Plotting SVM metrics
plt.figure(figsize=(8, 6))
plt.bar(svm_metrics.keys(), svm_metrics.values(), color=['blue', 'orange',␣
 ↪'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Support Vector Machine")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
SVM Results:
Accuracy: 0.9168891855807744
Precision: 0.9155117946888653
Recall: 0.9168891855807744
F1 Score: 0.9157172918986746
```



3.4 4. Decision Tree

[47]:
```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(max_depth=10, random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

print("Decision Tree Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_dt, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_dt, average='weighted'))

# Metrics for Decision Tree
decision_tree_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_dt),
    "Precision": precision_score(y_test, y_pred_dt, average='weighted'),
    "Recall": recall_score(y_test, y_pred_dt, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_dt, average='weighted')
}

# Plotting Decision Tree metrics
plt.figure(figsize=(8, 6))
plt.bar(decision_tree_metrics.keys(), decision_tree_metrics.values(),␣
 ↪color=['blue', 'orange', 'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Decision Tree")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

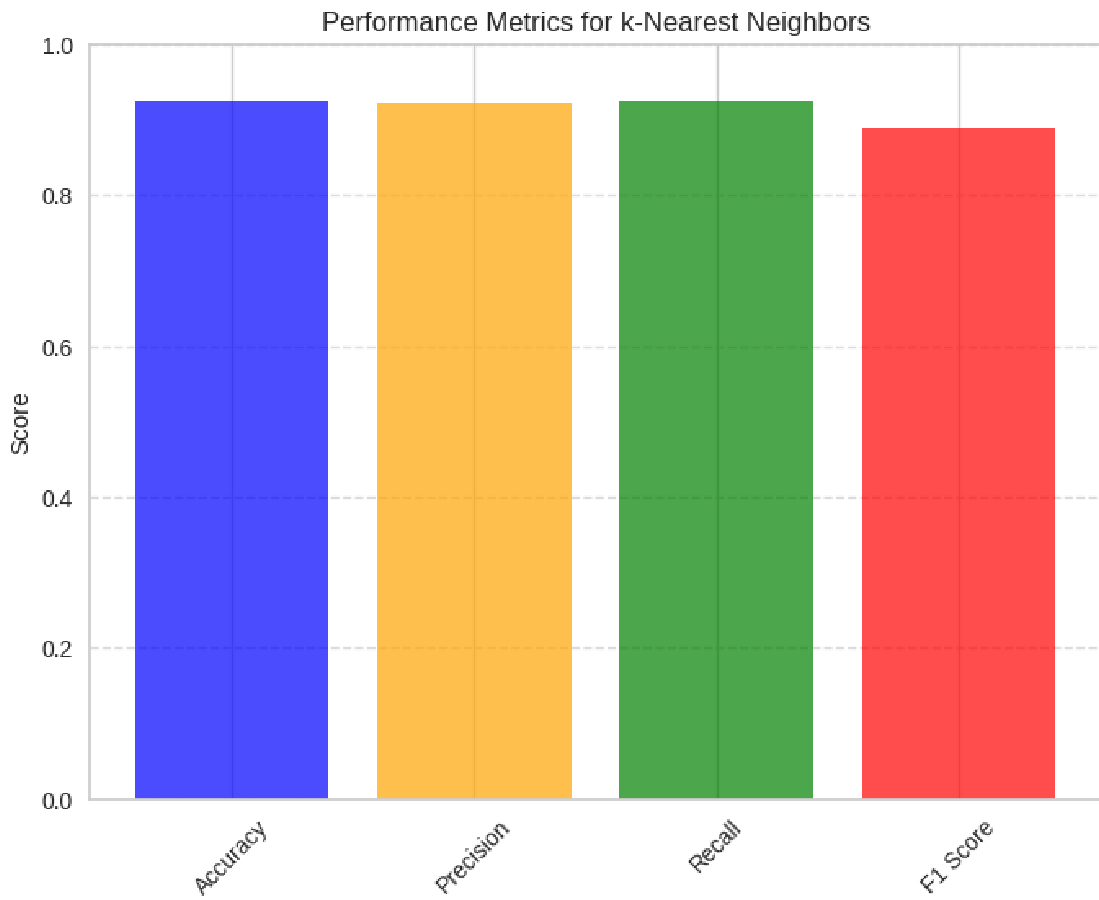
```
Decision Tree Results:
Accuracy: 0.7818758344459279
Precision: 0.7868058225993019
Recall: 0.7818758344459279
F1 Score: 0.7327650317591157
```

3.5 5. k-Nearest Neighbors (kNN)

[48]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
import matplotlib.pyplot as plt

# Sample data size = 10,000 rows as the kernel restarts again and again for␣
↪more than 10000
X_sample = tokens_vectorized[:10000]
y_sample = df_news['sentiment polarity'][:10000]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample,␣
↪test_size=0.2, random_state=42)

# Initializing and train kNN
```

```python
knn_model = KNeighborsClassifier(n_neighbors=5, algorithm='auto')
knn_model.fit(X_train, y_train)

# Predicting on the test set
y_pred_knn = knn_model.predict(X_test)

# Evaluating and printing metrics
print("k-Nearest Neighbors Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_knn, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_knn, average='weighted'))

# Collecting metrics for visualization
knn_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_knn),
    "Precision": precision_score(y_test, y_pred_knn, average='weighted'),
    "Recall": recall_score(y_test, y_pred_knn, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_knn, average='weighted')
}

# Plotting metrics
plt.figure(figsize=(8, 6))
plt.bar(knn_metrics.keys(), knn_metrics.values(), color=['blue', 'orange',␣
 ↪'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for k-Nearest Neighbors")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
k-Nearest Neighbors Results:
Accuracy: 0.9235
Precision: 0.92095815353 73809
Recall: 0.9235
F1 Score: 0.8899854532624898
```



3.6 6. Apriori Algorithm

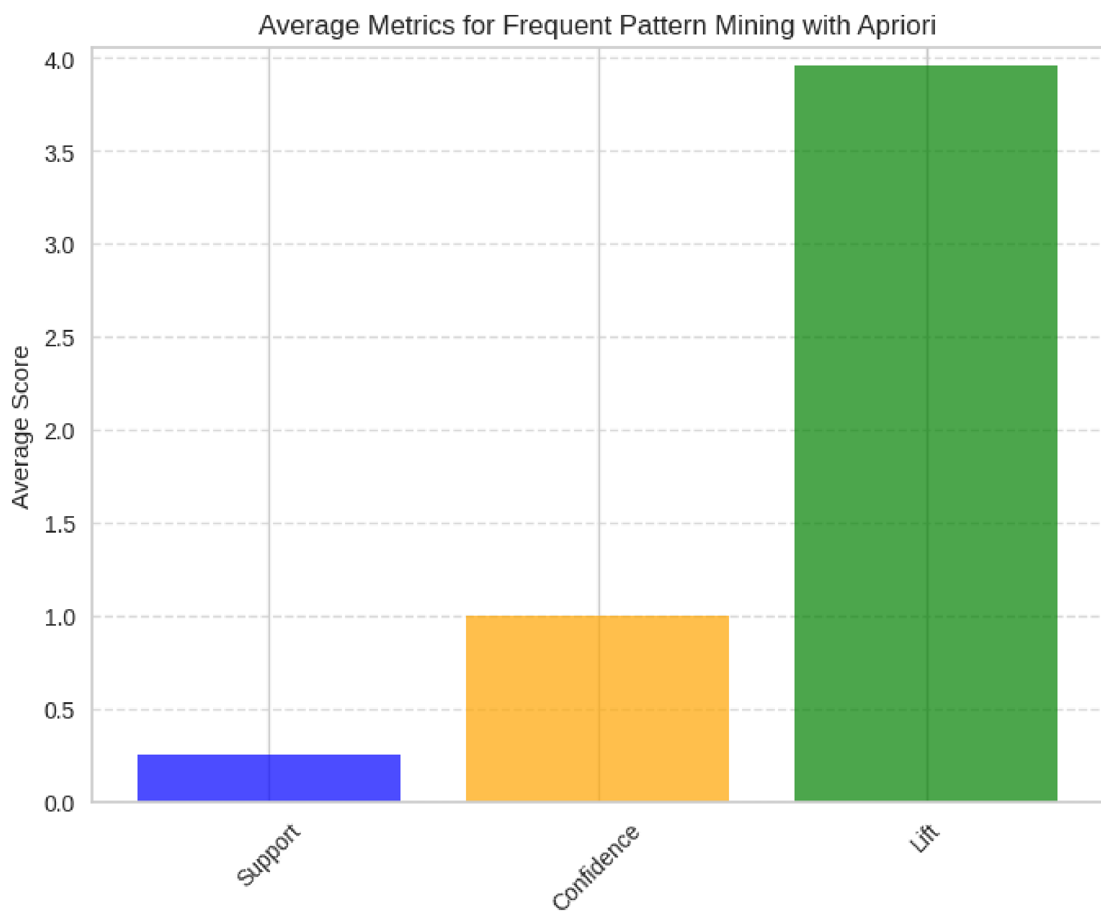
[49]:
```
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd
import matplotlib.pyplot as plt

# Sample data size = 10,000 rows as the kernel restarts again and again for␣
↪more than 10000
df_sample = df_news.sample(n=10000, random_state=42)

# Preparing the data for Apriori
transactions = df_sample['tokens_joined'].apply(lambda x: x.split())   #␣
↪Tokenized text
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_trans = pd.DataFrame(te_array, columns=te.columns_)
```

```python
# Applying Apriori
frequent_itemsets = apriori(df_trans, min_support=0.02, use_colnames=True)

# Generating Association Rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0,␣
↪num_itemsets=None)

# Displaying the top rules
print("Top 10 Association Rules:")
print(rules.head(10))

# Plotting support, confidence, and lift
rules_metrics = {
    "Support": rules['support'].mean(),
    "Confidence": rules['confidence'].mean(),
    "Lift": rules['lift'].mean()
}

plt.figure(figsize=(8, 6))
plt.bar(rules_metrics.keys(), rules_metrics.values(), color=['blue', 'orange',␣
↪'green'], alpha=0.7)
plt.title("Average Metrics for Frequent Pattern Mining with Apriori")
plt.ylabel("Average Score")
plt.ylim(0, max(rules_metrics.values()) + 0.1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Top 10 Association Rules:
       antecedents     consequents  antecedent support  consequent support  \
0            (boy)            (11)              0.2552              0.2544
1             (11)           (boy)              0.2544              0.2552
2         (charge)            (11)              0.2554              0.2544
3             (11)        (charge)              0.2544              0.2554
4          (fatal)            (11)              0.2551              0.2544
5             (11)         (fatal)              0.2544              0.2551
6             (11)  (manslaughter)              0.2544              0.2519
7   (manslaughter)            (11)              0.2519              0.2544
8            (old)            (11)              0.2587              0.2544
9             (11)           (old)              0.2544              0.2587

   support  confidence      lift  representativity  leverage  conviction  \
0   0.2518    0.986677  3.878448               1.0  0.186877   55.963859
1   0.2518    0.989780  3.878448               1.0  0.186877   72.875815
2   0.2517    0.985513  3.873872               1.0  0.186726   51.466551
3   0.2517    0.989387  3.873872               1.0  0.186726   70.157867
4   0.2515    0.985888  3.875345               1.0  0.186603   52.834044
```

5	0.2515	0.988601	3.875345	1.0	0.186603	65.345710
6	0.2517	0.989387	3.927697	1.0	0.187617	70.487644
7	0.2517	0.999206	3.927697	1.0	0.187617	939.083200
8	0.2519	0.973715	3.827495	1.0	0.186087	28.365694
9	0.2519	0.990173	3.827495	1.0	0.186087	75.434688

	zhangs_metric	jaccard	certainty	kulczynski
0	0.996462	0.976726	0.982131	0.988228
1	0.995393	0.976726	0.986278	0.988228
2	0.996321	0.975203	0.980570	0.987450
3	0.994984	0.975203	0.985746	0.987450
4	0.996051	0.974806	0.981073	0.987244
5	0.995116	0.974806	0.984697	0.987244
6	0.999729	0.988610	0.985813	0.994296
7	0.996388	0.988610	0.998935	0.994296
8	0.996537	0.964395	0.964746	0.981944
9	0.990789	0.964395	0.986743	0.981944

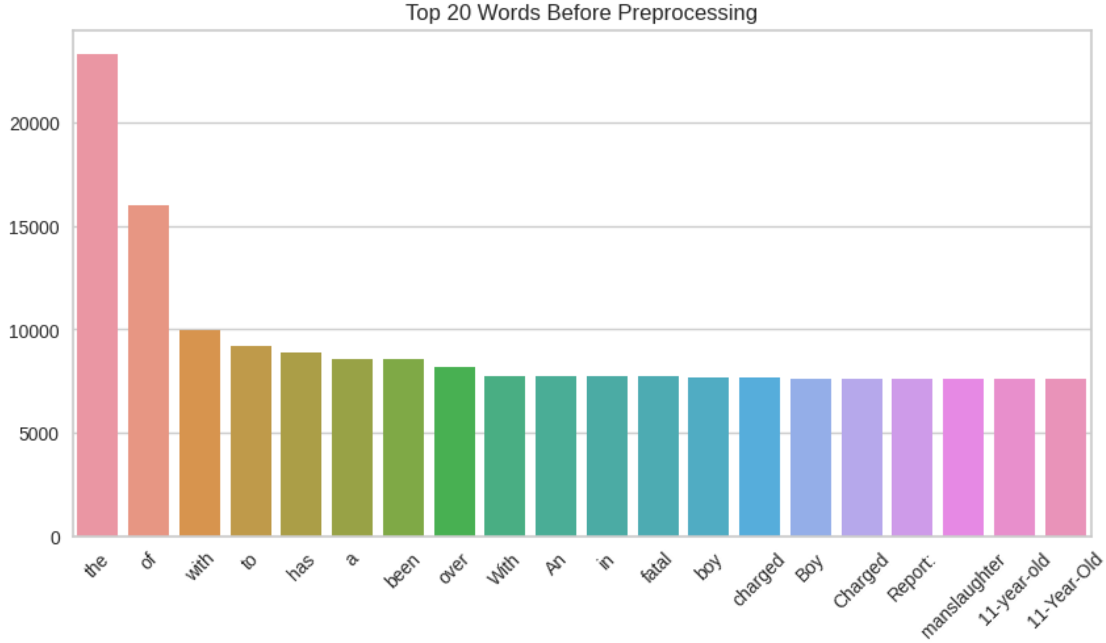


3.6.1 Data Before and After Preprocessing

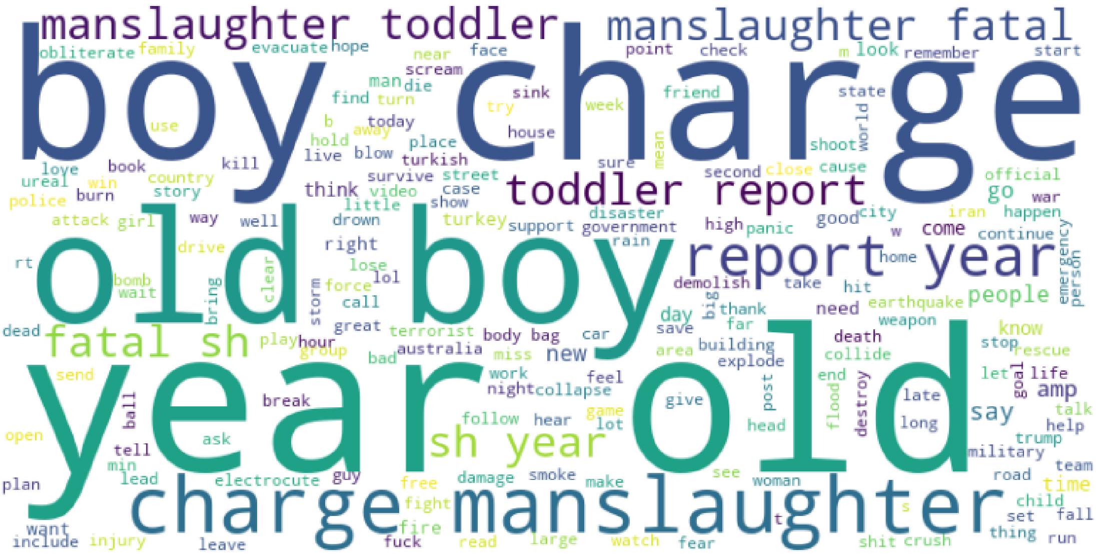
[50]:
```python
def plot_wordcloud(text_data, title):
    wordcloud = WordCloud(width=800, height=400, background_color="white").
 ↪generate(" ".join(text_data))
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.title(title)
    plt.show()

# Function to plot top word frequencies
def plot_top_words(text_data, title):
    all_words = " ".join(text_data).split()
    word_freq = Counter(all_words)
    top_words = word_freq.most_common(20)

    words = [w[0] for w in top_words]
    counts = [w[1] for w in top_words]

    plt.figure(figsize=(10, 5))
    sns.barplot(x=words, y=counts)
    plt.xticks(rotation=45)
    plt.title(title)
    plt.show()

### Before Preprocessing ###

# Plotting word cloud before preprocessing
raw_text_data = df_news['content'].astype(str).tolist()
plot_wordcloud(raw_text_data, "Word Cloud Before Preprocessing")

# Plotting top word frequencies before preprocessing
plot_top_words(raw_text_data, "Top 20 Words Before Preprocessing")

### After Preprocessing ###

# Plotting word cloud after preprocessing
cleaned_text_data = df_news['tokens_joined'].tolist()
plot_wordcloud(cleaned_text_data, "Word Cloud After Preprocessing")

# Plotting top word frequencies after preprocessing
plot_top_words(cleaned_text_data, "Top 20 Words After Preprocessing")
```

Word Cloud Before Preprocessing

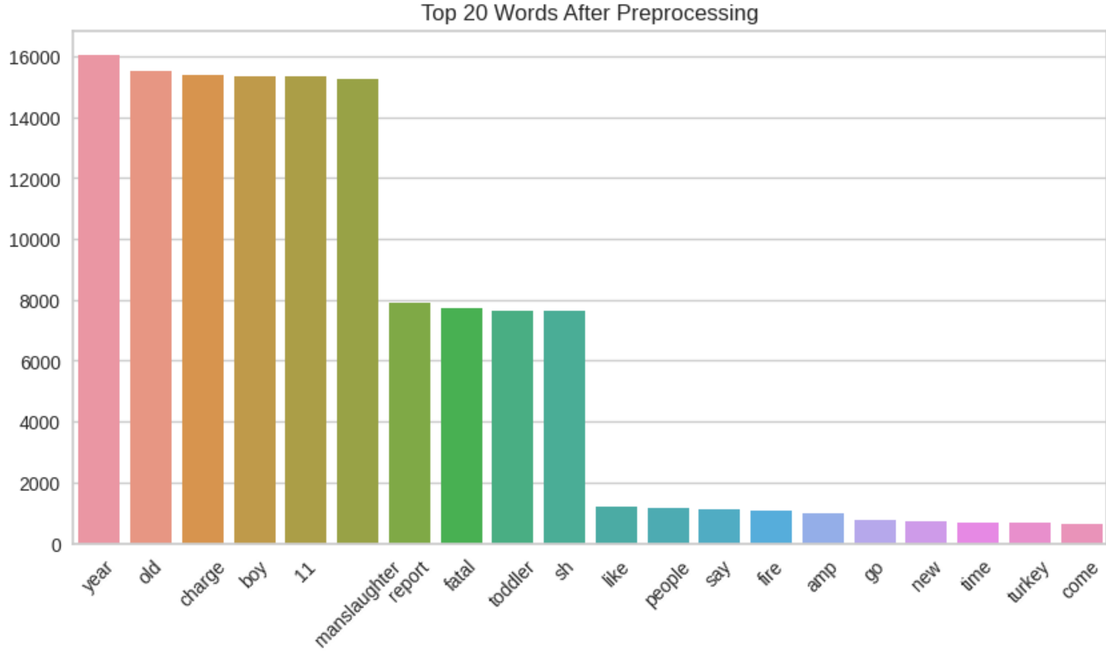
Top 20 Words Before Preprocessing



Word Cloud After Preprocessing



Top 20 Words After Preprocessing



4 YouTube Comments Analysis

4.1 Data Collection

[51]:
```
# Extracting comments from video ID : 0jZE4p1SOvg
data = requests.get("https://www.googleapis.com/youtube/v3/commentThreads?
 ↪key=AIzaSyAkFGtjiXLXY-q_x-qYA9pfjSBVH8Rqnow&textFormat=plainText&part=snippet&videoId=0jZE4
next = data.json()['nextPageToken']
next
data1 = requests.get(f"https://www.googleapis.com/youtube/v3/commentThreads?
 ↪key=AIzaSyAkFGtjiXLXY-q_x-qYA9pfjSBVH8Rqnow&textFormat=plainText&part=snippet&videoId=0jZE4
comm = []
for i,v in pd.DataFrame(data.json()['items']).iterrows():
    comm.append(v['snippet']['topLevelComment']['snippet']['textOriginal'])
comm1 = []
for i,v in pd.DataFrame(data1.json()['items']).iterrows():
    comm1.append(v['snippet']['topLevelComment']['snippet']['textOriginal'])

# Extracting comments from video ID: _TzKuBi1gjw
data_2 = requests.get("https://www.googleapis.com/youtube/v3/commentThreads?
 ↪key=AIzaSyAkFGtjiXLXY-q_x-qYA9pfjSBVH8Rqnow&textFormat=plainText&part=snippet&videoId=_TzKu
next_2 = data_2.json()['nextPageToken']
next_2
data1_2 = requests.get(f"https://www.googleapis.com/youtube/v3/commentThreads?
 ↪key=AIzaSyAkFGtjiXLXY-q_x-qYA9pfjSBVH8Rqnow&textFormat=plainText&part=snippet&videoId=_TzKu
comm2 = []
for i,v in pd.DataFrame(data_2.json()['items']).iterrows():
    comm2.append(v['snippet']['topLevelComment']['snippet']['textOriginal'])
comm1_2= []
for i,v in pd.DataFrame(data1_2.json()['items']).iterrows():
    comm1_2.append(v['snippet']['topLevelComment']['snippet']['textOriginal'])
```

[52]:
```
# Converting to DataFrame

ids = []
dates = []
comments = []

for i, comment in enumerate(comm):
    ids.append(i + 1)
    dates.append(datetime.now().strftime("%Y-%m-%d"))
    comments.append(comment)

for i, comment in enumerate(comm1):
    ids.append(len(comm) + i + 1)
    dates.append(datetime.now().strftime("%Y-%m-%d"))
    comments.append(comment)
```

```
# Creating DataFrame
df1 = pd.DataFrame({"id": ids, "date": dates, "comment": comments})

# Convert the DataFrame to Excel
df1.to_excel("comments_data.xlsx", index=False)


ids2 = []
dates2 = []
comments2 = []


for i, comment in enumerate(comm2):
    ids2.append(i+1)
    dates2.append(datetime.now().strftime("%Y-%m-%d"))
    comments2.append(comment)


for i, comment in enumerate(comm1_2):
    ids2.append(len(comm2)+i+1)
    dates2.append(datetime.now().strftime("%Y-%m-%d"))
    comments2.append(comment)

# Creating DataFrame
df2 = pd.DataFrame({"id": ids2, "date": dates2, "comment": comments2})

# Convert the DataFrame to Excel
df2.to_excel("comments_data2.xlsx", index=False)

df1.head(),df2.head()
```

[52]:
```
(   id        date                                            comment
 0   1  2024-11-17  Toa predmet prirodni nauki za prirodni katastr…
 1   2  2024-11-17  Da toa treba da ucat u skoloto za prirodni kat…
 2   3  2024-11-17  Why aren't they built as per Japanese standards?.
 3   4  2024-11-17                                            Weather
 4   5  2024-11-17                                             Today,
    id        date                                            comment
 0   1  2024-11-17                         Allah Hu Akbar God bhagwan
 1   2  2024-11-17                    This kind of reminds me of 9/11
 2   3  2024-11-17                   Rest in peace Turkey and Syria
 3   4  2024-11-17                                  why'd that happen
 4   5  2024-11-17  It just happened again today while sleeping no…)
```

[53]:
```
# Combining comments from both videos
df3 = pd.concat([df1, df2], axis=0)
```

```
print(df3)
```

```
      id        date                                            comment
0      1  2024-11-17  Toa predmet prirodni nauki za prirodni katastr…
1      2  2024-11-17  Da toa treba da ucat u skoloto za prirodni kat…
2      3  2024-11-17  Why aren't they built as per Japanese standards?.
3      4  2024-11-17                                            Weather
4      5  2024-11-17                                              Today
..   …         …                                                …
195  196  2024-11-17                    It's terrible the earthquake.
196  197  2024-11-17                                     Yaa Allah
197  198  2024-11-17
198  199  2024-11-17
199  200  2024-11-17                                  God Please save us

[400 rows x 3 columns]
```

4.2 Data Cleaning and Visualization

[54]:

```
# Checking missing value and filling them out.

missing_values = df3.isnull().sum()
print("Missing values before handling:")
print(missing_values)

# Filling missing values with the mean of each column
df3_filled = df3.fillna(df3.mean())

# Check if there are any missing values left after handling
missing_values_after = df3_filled.isnull().sum()
print("\nMissing values after handling:")
print(missing_values_after)

print("\nFirst few rows of the cleaned dataframe:")
print(df3_filled.head())
```

```
Missing values before handling:
id         0
date       0
comment    0
dtype: int64

Missing values after handling:
id         0
date       0
comment    0
dtype: int64
```

```
First few rows of the cleaned dataframe:
   id        date                                            comment
0   1  2024-11-17  Toa predmet prirodni nauki za prirodni katastr…
1   2  2024-11-17  Da toa treba da ucat u skoloto za prirodni kat…
2   3  2024-11-17  Why aren't they built as per Japanese standards?.
3   4  2024-11-17                                            Weather
4   5  2024-11-17                                              Today
```

[55]:
```python
# Keeping only the required column for our analysis
df3 = df3[['comment']]

#removing out the emoji's

df3 = df3.astype(str).apply(lambda x: x.str.encode('ascii', 'ignore').str.
 ↪decode('ascii'))

#drop missing values from comments

df3['comment'].replace('', np.nan, inplace=True)
df3.dropna(subset=['comment'], inplace=True)

#removing punctuation:

def remove_punctuation(txt):
    txt_nopunct = "".join([c for c in txt if c not in string.punctuation])
    return txt_nopunct

df3['comment'] = df3['comment'].apply(lambda x: remove_punctuation(x))
df3.head()
```

[55]:

	comment
0	Toa predmet prirodni nauki za prirodni katastr…
1	Da toa treba da ucat u skoloto za prirodni kat…
2	Why arent they built as per Japanese standards
3	Weather
4	Today

[56]:
```python
# Removing the stopwords also tokenizing and stemming of the text data
def tokenize(txt):
    tokens = re.split('\W+', txt)
    return tokens

df3['comments_tokenized'] = df3['comment'].apply(lambda x: tokenize(x.lower()))
df3.head()

stop_words = set(stopwords.words('english'))
```

```
def remove_stopwords(txt):
    txt_clean = [word for word in txt if word not in stop_words]
    return txt_clean

df3['comments_without_stopwords'] = df3['comments_tokenized'].apply(lambda x:␣
↪remove_stopwords(x))
stemmer = PorterStemmer()

df3['comments_without_stopwords_joined'] = df3['comments_without_stopwords'].
↪apply(lambda x: ' '.join(x))
df3.head()
```

[56]:

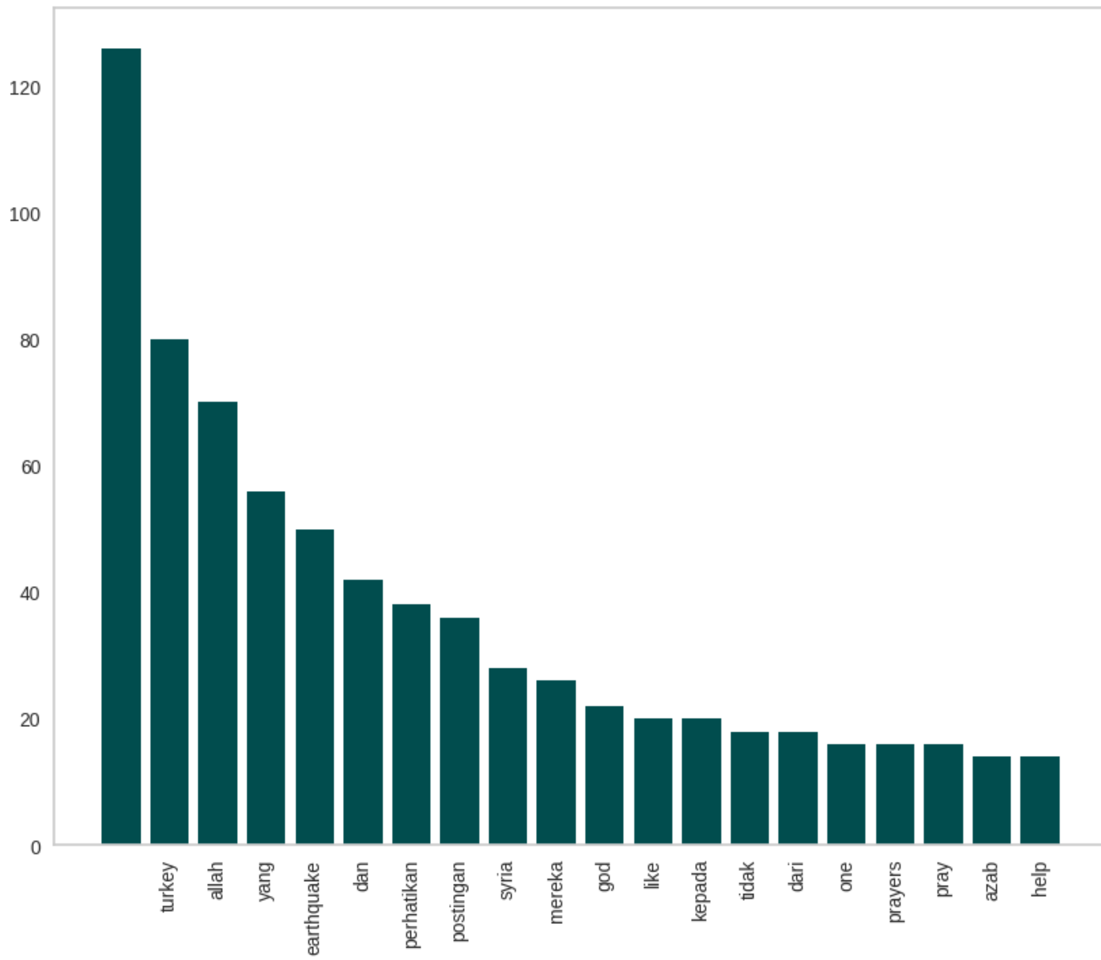
```
                                         comment  \
0  Toa predmet prirodni nauki za prirodni katastr…
1  Da toa treba da ucat u skoloto za prirodni kat…
2     Why arent they built as per Japanese standards
3                                         Weather
4                                           Today

                              comments_tokenized  \
0  [toa, predmet, prirodni, nauki, za, prirodni, …
1  [da, toa, treba, da, ucat, u, skoloto, za, pri…
2  [why, arent, they, built, as, per, japanese, s…
3                                      [weather]
4                                        [today]

                      comments_without_stopwords  \
0  [toa, predmet, prirodni, nauki, za, prirodni, …
1  [da, toa, treba, da, ucat, u, skoloto, za, pri…
2           [arent, built, per, japanese, standards]
3                                      [weather]
4                                        [today]

               comments_without_stopwords_joined
0  toa predmet prirodni nauki za prirodni katastr…
1  da toa treba da ucat u skoloto za prirodni kat…
2               arent built per japanese standards
3                                         weather
4                                           today
```

[57]:

```
# Wordlcoud in Turkey's map
image = Image.open("tse.png")
mask = np.array(image)

x = " ".join(i for i in df3.comments_without_stopwords_joined)
```

```
wc = WordCloud(background_color = "white",colormap="PuBu", repeat = True, mask = mask).generate(x)

plt.figure(figsize=(10,8 ))
dark_teal_color = (1, 77, 78)
plt.imshow(wc.recolor(color_func=lambda *args, **kwargs: dark_teal_color), interpolation='bilinear')
plt.axis("off")
plt.show()
```

[58]:
```
# 20 most frequent words in comments and their frequency in bar plot
df_freq = pd.DataFrame.from_dict(dict(Counter(' '.join(df3['comments_without_stopwords_joined']).split(" "))),orient='index').reset_index()
df_freq.rename(columns={0:'counts','index':'words'},inplace=True)
df_freq.sort_values(by='counts',ascending=False,inplace=True)
df_freq = df_freq.head(20)

plt.figure(figsize=(10,8))
plt.grid(False)
plt.bar(df_freq['words'],df_freq['counts'],color = "#014d4e")
plt.xticks(rotation=90)
plt.show()
```



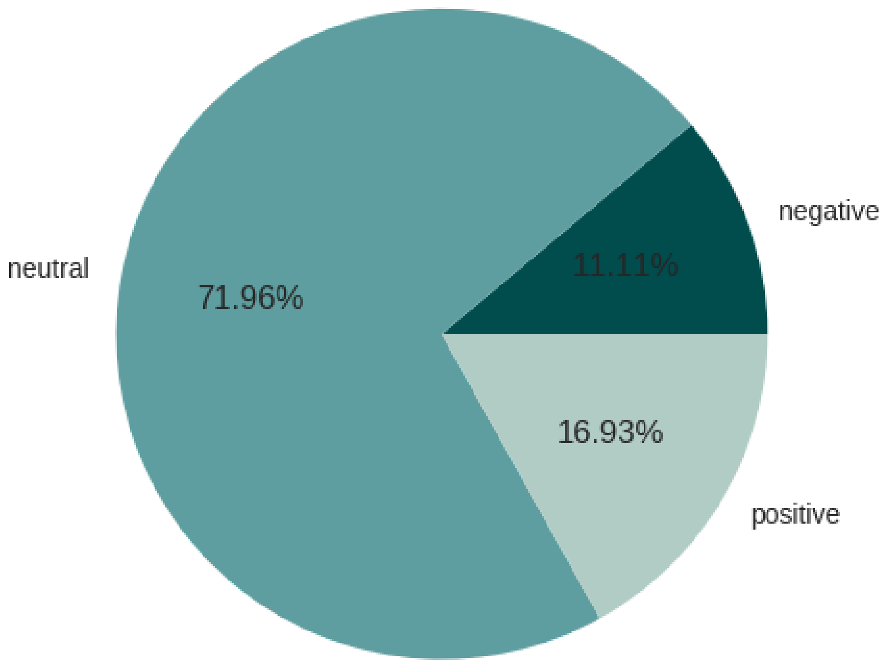
[59]:
```
# Getting sentiment polarity using TextBlob
df3['polarity'] = df3['comment'].apply(lambda x: TextBlob(x).sentiment.polarity)

df3['sentiment'] = df3['polarity'].apply(lambda x: 'positive' if x>0 else␣
↪('negative' if x<0 else 'neutral'))

df3.sentiment.value_counts()
df3.head()
```

[59]:
```
                                             comment  \
0  Toa predmet prirodni nauki za prirodni katastr…
1  Da toa treba da ucat u skoloto za prirodni kat…
2     Why arent they built as per Japanese standards
3                                            Weather
4                                              Today

                                 comments_tokenized  \
```

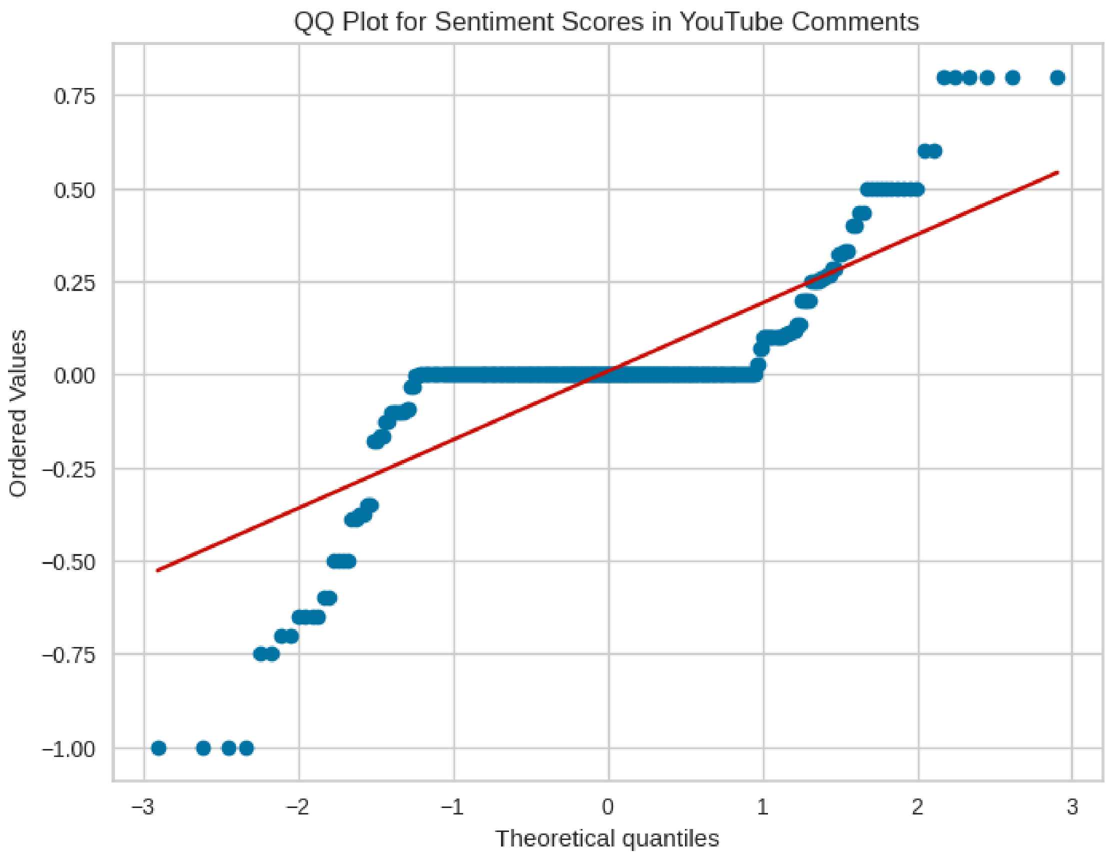
```
0  [toa, predmet, prirodni, nauki, za, prirodni, …
1  [da, toa, treba, da, ucat, u, skoloto, za, pri…
2  [why, arent, they, built, as, per, japanese, s…
3                                        [weather]
4                                          [today]

                        comments_without_stopwords  \
0  [toa, predmet, prirodni, nauki, za, prirodni, …
1  [da, toa, treba, da, ucat, u, skoloto, za, pri…
2           [arent, built, per, japanese, standards]
3                                        [weather]
4                                          [today]

                 comments_without_stopwords_joined  polarity sentiment
0  toa predmet prirodni nauki za prirodni katastr…       0.0   neutral
1  da toa treba da ucat u skoloto za prirodni kat…       0.0   neutral
2                  arent built per japanese standards      0.0   neutral
3                                          weather       0.0   neutral
4                                            today       0.0   neutral
```

[60]:
```python
# Pie chart of sentiment classification
df3.groupby('sentiment').size().plot(kind='pie', autopct='%.2f%%',colors =␣
 ↪["#014d4e","#5F9EA0","#b1ccc5"])
```

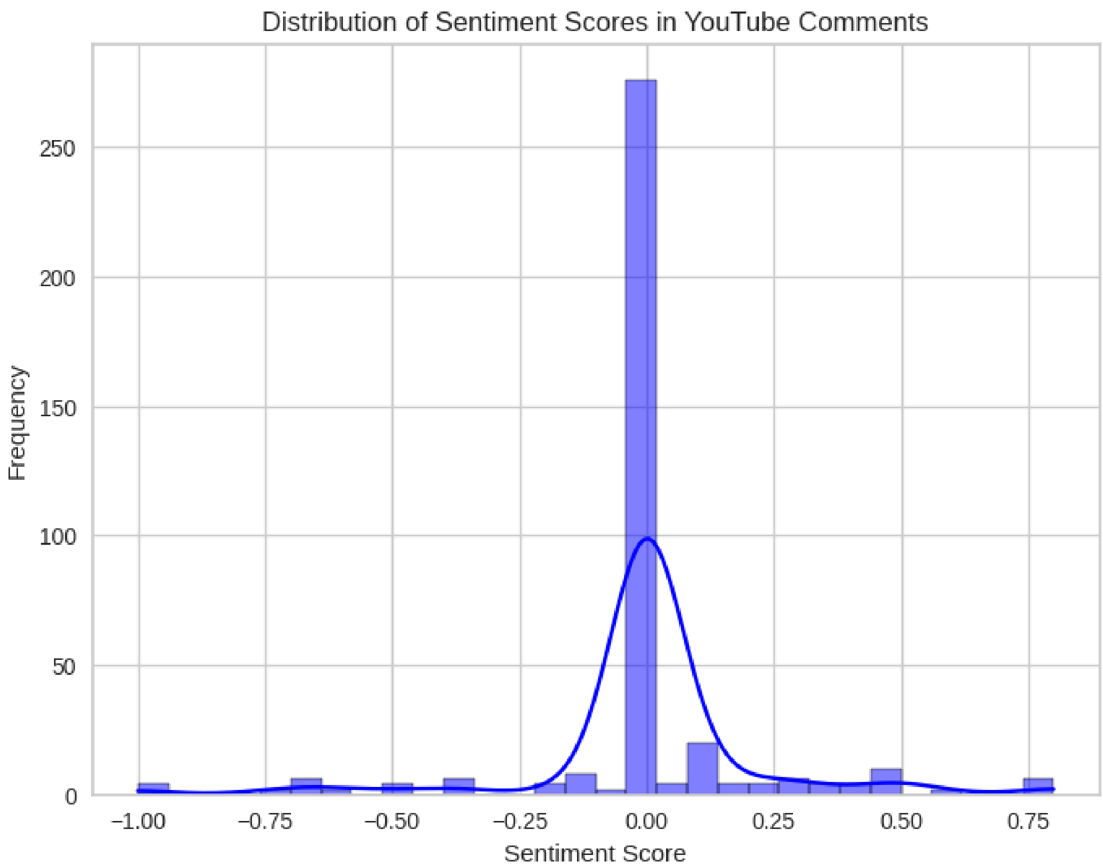
[60]: <Axes: >



[88]:
```
# QQ Plot for Sentiment Scores
import scipy.stats as stats

plt.figure(figsize=(8, 6))
stats.probplot(df3['polarity'], dist="norm", plot=plt)
plt.title("QQ Plot for Sentiment Scores in YouTube Comments")
plt.show()
```



[86]:
```
# Histogram plot for Sentiment Score Distribution
plt.figure(figsize=(8, 6))
sns.histplot(df3['polarity'], kde=True, bins=30, color="blue")  # Replace
 ↪column name
plt.title("Distribution of Sentiment Scores in YouTube Comments")
plt.xlabel("Sentiment Score")
plt.ylabel("Frequency")
plt.show()
```



[61]:
```
# Vectorize comments for clustering

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df3['comments_without_stopwords_joined'])
nlp = spacy.load('en_core_web_sm')
df3['tokens_joined'] = df3['comments_tokenized'].apply(lambda x: ' '.join(i for␣
 ↪i in x if i.isnumeric()))
df3['comments_tokenized'] = df3['tokens_joined'].apply(lambda x: nlp(x.lower()))
X.shape
```

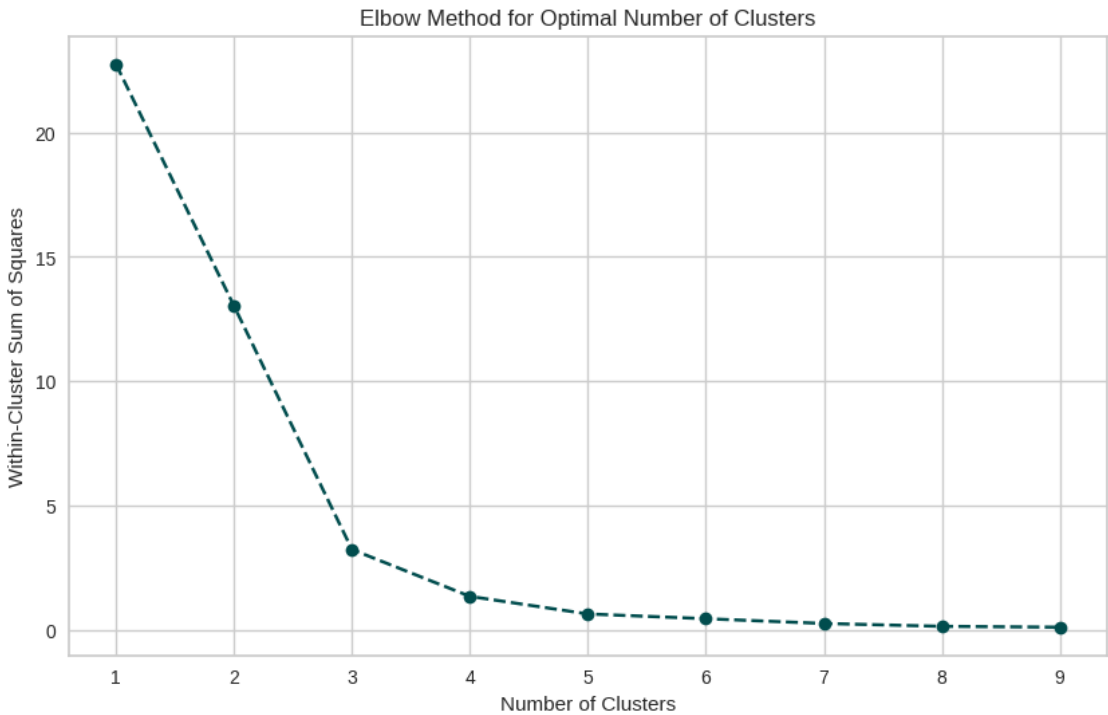
[61]: (378, 1057)

[62]:
```
# Implementing PCA to reduce data dimension to 2
pca = PCA(n_components=2)
reduced_tfidf = pca.fit_transform(X.toarray())

reduced_tfidf.shape
```

[62]: (378, 2)

[63]:

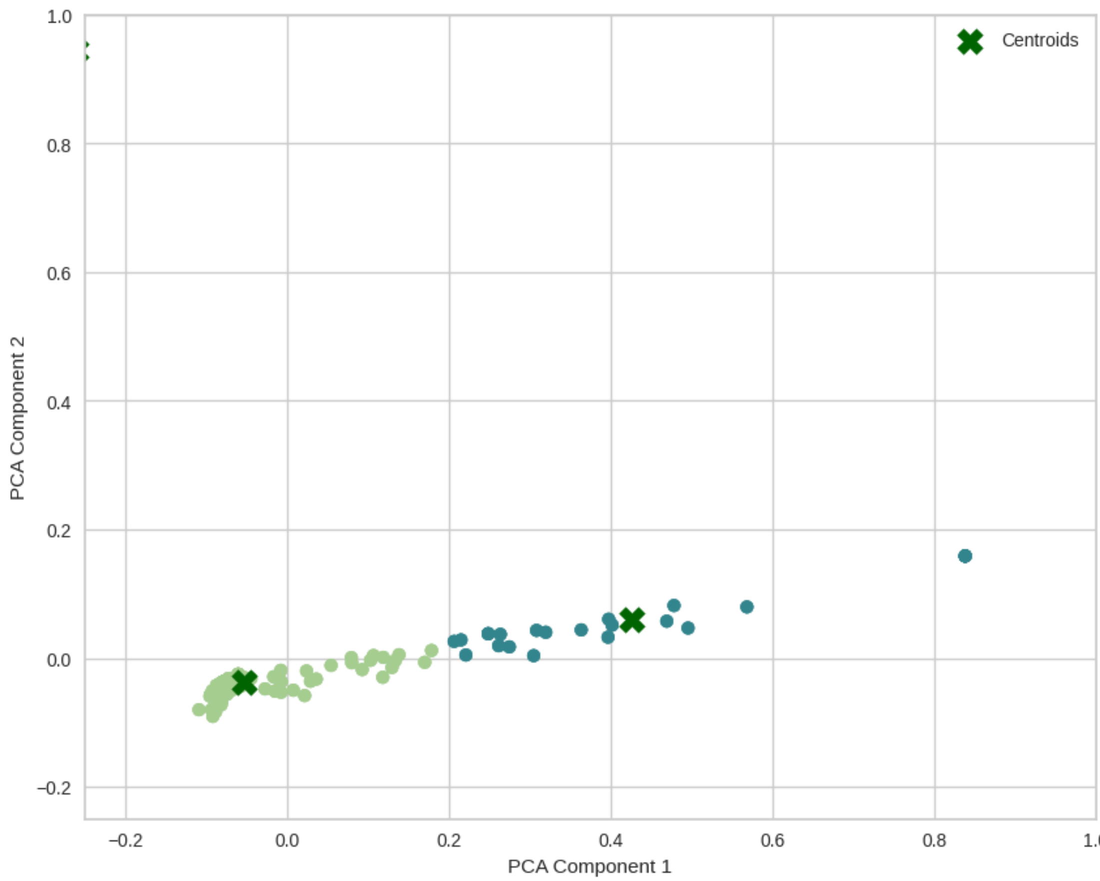
```
# Elbow plot to determing optimal number of clusters
wcss = []
cluster_range = range(1, 10)

for k in cluster_range:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(reduced_tfidf)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(cluster_range, wcss, marker='o', linestyle='--',color = "#014d4e")
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.grid(True)
plt.show()
```



[64]:

```
# KMeans clustering with 3 clusters
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(reduced_tfidf)

# Scatterplot of clusters
plt.figure(figsize=(10, 8))
plt.scatter(reduced_tfidf[:, 0], reduced_tfidf[:, 1], c=clusters, cmap='crest')
```

```
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
↪s=200, c='darkgreen', marker='X', label='Centroids')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

plt.legend()
plt.xlim(-0.25,1)
plt.ylim(-0.25,1)
plt.grid(True)
plt.show()
```



[65]:
```
# Adding cluster label column to the original dataframe
df3['clusters'] = clusters
```

[66]:
```
# Getting most frequent 20 words from each cluster
def get_top_keywords(data, clusters, labels, n_terms):
    # Converting the data to a dense DataFrame and compute the mean per cluster
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()
```

```
    print(df)
    print(df.sum(axis=1))

    for i, r in df.iterrows():
        print('\nCluster {}'.format(i))

        # Sorting the terms by importance
        top_indices = np.argsort(r)[-n_terms:]  # Get the top 'n_terms' indices
        print("Top Indices:", top_indices)

        # Ensuring the indices are within the range of available labels
        top_indices = top_indices[top_indices < len(labels)]
        print("Filtered Top Indices:", top_indices)

        # Printing the most frequent terms in the cluster
        print(', '.join([labels[t] for t in top_indices]))

get_top_keywords(X, clusters, vectorizer.get_feature_names_out(), 20)
```

```
       0         1         2         3         4         5         6     \
0  0.006211  0.003173  0.003586  0.001552  0.001212  0.000843  0.002995
1  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
2  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

       7         8         9     …     1047      1048      1049      1050  \
0  0.002627  0.003809  0.003809  …  0.003959  0.001648  0.002676  0.002143
1  0.000000  0.000000  0.000000  …  0.000000  0.000000  0.000000  0.000000
2  0.000000  0.000000  0.000000  …  0.000000  0.000000  0.000000  0.000000

       1051      1052      1053    1054      1055      1056
0  0.005422  0.000112  0.000421  0.0013  0.001879  0.001602
1  0.000000  0.000000  0.000000  0.0000  0.000000  0.000000
2  0.000000  0.000000  0.000000  0.0000  0.000000  0.000000

[3 rows x 1057 columns]
0    2.144605
1    1.839661
2    1.414214
dtype: float64

Cluster 0
Top Indices: 1037    430
1038     75
1039    135
1040     19
1041    662
1042    535
```

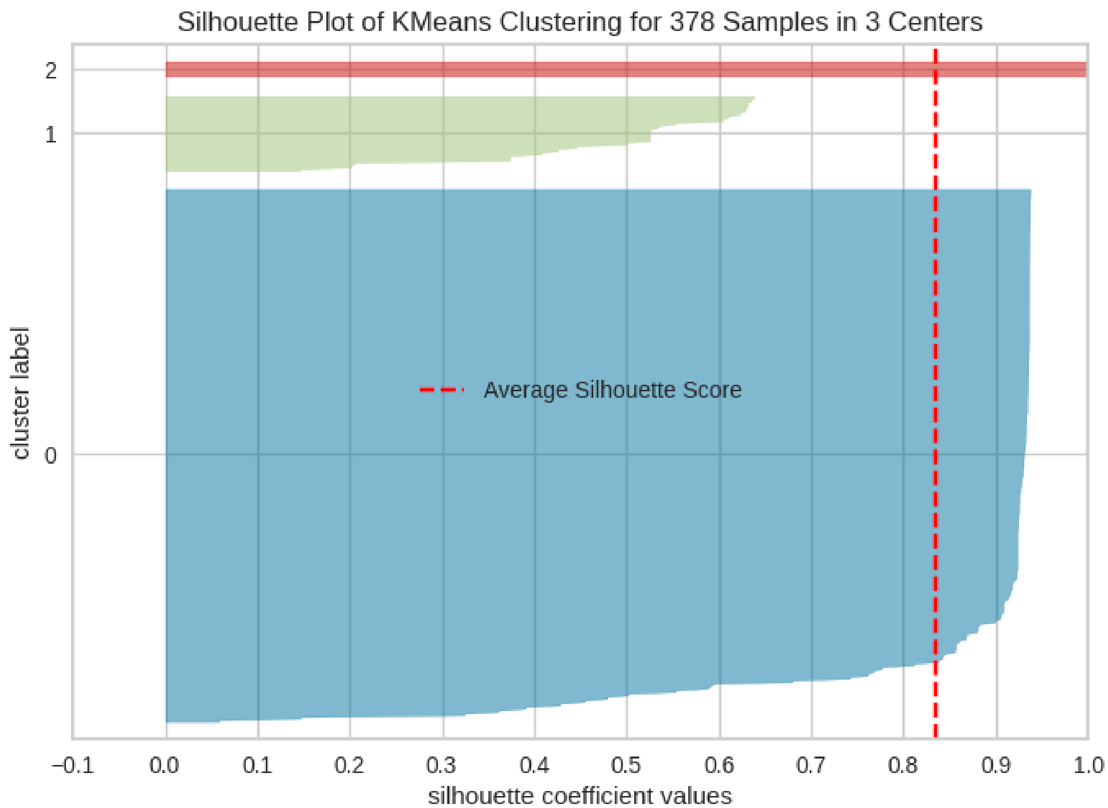
```
1043    656
1044    414
1045     43
1046    973
1047     35
1048    896
1049    383
1050    655
1051    370
1052    664
1053    345
1054    730
1055    271
1056     42
Name: 0, dtype: int64
Filtered Top Indices: 1037    430
1038     75
1039    135
1040     19
1041    662
1042    535
1043    656
1044    414
1045     43
1046    973
1047     35
1048    896
1049    383
1050    655
1051    370
1052    664
1053    345
1054    730
1055    271
1056     42
Name: 0, dtype: int64
israel, atsu, buildings, 911, omg, like, ok, im, allahuakbar, turkey, akbar,
syria, help, oh, happen, one, god, prayers, earthquake, allah

Cluster 1
Top Indices: 1037    631
1038    931
1039    127
1040     57
1041    521
1042    386
1043    682
1044    651
```

```
1045    330
1046    384
1047     72
1048    720
1049    178
1050    691
1051    776
1052    730
1053    781
1054    728
1055    896
1056    973
Name: 1, dtype: int64
Filtered Top Indices: 1037    631
1038    931
1039    127
1040     57
1041    521
1042    386
1043    682
1044    651
1045    330
1046    384
1047     72
1048    720
1049    178
1050    691
1051    776
1052    730
1053    781
1054    728
1055    896
1056    973
Name: 1, dtype: int64
nato, thanks, bro, answer, language, helps, palestine, notice, footage, helped,
astu, pls, country, peace, rest, prayers, rip, pray, syria, turkey

Cluster 2
Top Indices: 1037    345
1038    346
1039    347
1040    355
1041    362
1042    361
1043    360
1044    359
1045    358
1046    357
```

```
1047    356
1048    332
1049    354
1050    353
1051    352
1052    351
1053    350
1054    349
1055    170
1056    317
Name: 2, dtype: int64
Filtered Top Indices: 1037    345
1038    346
1039    347
1040    355
1041    362
1042    361
1043    360
1044    359
1045    358
1046    357
1047    356
1048    332
1049    354
1050    353
1051    352
1052    351
1053    350
1054    349
1055    170
1056    317
Name: 2, dtype: int64
god, going, golongan, grandma, ha, gy, guys, guy, gurun, greece, grandmother,
free, grandad, governmet, gov, got, gospel, google, condolence, family
```

[67]:

```python
# Plotting silhoutte score for 3 clusters
from yellowbrick.cluster import SilhouetteVisualizer
visualizer = SilhouetteVisualizer(kmeans, colors='yellowbrick')
visualizer.fit(reduced_tfidf)
visualizer.show()
print(visualizer.silhouette_score_)
```




```
0.8347531504009202
```

[68]:
```python
# Preparing data for apriori

data = list(df3['comments_without_stopwords'].apply(lambda x:list(x)))
a = TransactionEncoder()
a_data = a.fit(data).transform(data)
df2 = pd.DataFrame(a_data,columns=a.columns_)
df2 = df2.replace(False,0)
df2 = df2.replace(True,1)
df2.head()
```

[68]:
```
      009  1  100000  117  123  14  1518  2  2015  …  yo  yodo  yox  ysir  \
0  0    0  0       0    0    0   0     0  0     0  …   0     0    0     0
1  0    0  0       0    0    0   0     0  0     0  …   0     0    0     0
2  0    0  0       0    0    0   0     0  0     0  …   0     0    0     0
3  0    0  0       0    0    0   0     0  0     0  …   0     0    0     0
4  0    0  0       0    0    0   0     0  0     0  …   0     0    0     0

   za  zabur  zaman  zemjotresi  zilzalaha  zone
0   1      0      0           1          0     0
```

```
1  1         0        0              0             0       0
2  0         0        0              0             0       0
3  0         0        0              0             0       0
4  0         0        0              0             0       0

[5 rows x 1068 columns]
```

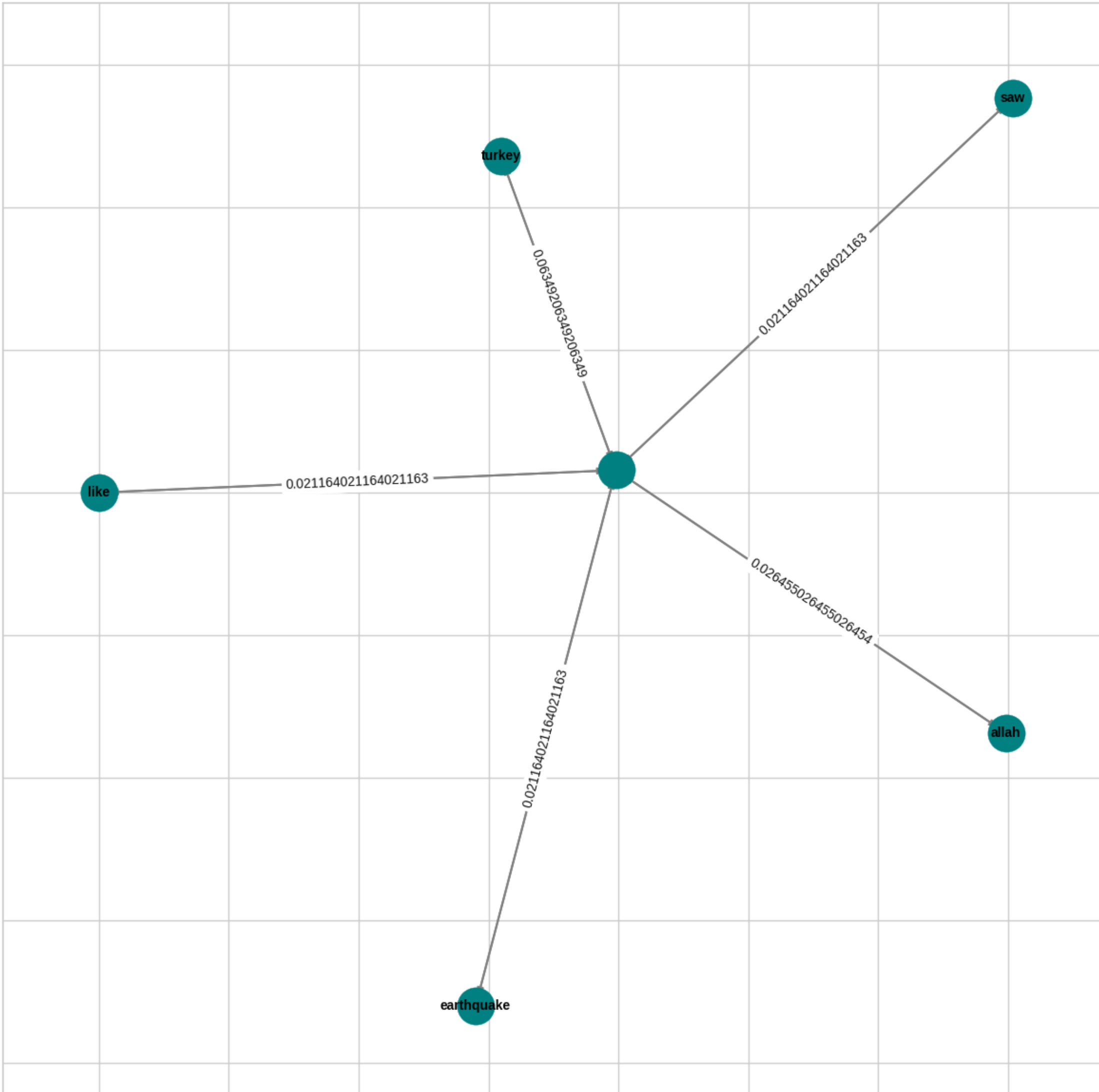
[69]:
```python
# Implementing apriori algorithm to find association rules
from mlxtend.frequent_patterns import apriori, association_rules
df4 = apriori(df2, min_support = 0.02, use_colnames = True, verbose = 1)
rules = association_rules(df4, metric = "lift", min_threshold = 0.5,
    num_itemsets=None)
print(rules.head(10))
```

```
Processing 123 combinations | Sampling itemset size 32
    antecedents   consequents  antecedent support  consequent support  \
0            ()       (allah)            0.275132            0.068783
1       (allah)            ()            0.068783            0.275132
2            ()  (earthquake)            0.275132            0.089947
3  (earthquake)            ()            0.089947            0.275132
4            ()        (like)            0.275132            0.047619
5        (like)            ()            0.047619            0.275132
6            ()         (saw)            0.275132            0.021164
7         (saw)            ()            0.021164            0.275132
8            ()      (turkey)            0.275132            0.190476
9      (turkey)            ()            0.190476            0.275132

    support  confidence      lift  representativity  leverage  conviction  \
0  0.026455    0.096154  1.397929               1.0  0.007531    1.030283
1  0.026455    0.384615  1.397929               1.0  0.007531    1.177910
2  0.021164    0.076923  0.855204               1.0 -0.003583    0.985891
3  0.021164    0.235294  0.855204               1.0 -0.003583    0.947904
4  0.021164    0.076923  1.615385               1.0  0.008062    1.031746
5  0.021164    0.444444  1.615385               1.0  0.008062    1.304762
6  0.021164    0.076923  3.634615               1.0  0.015341    1.060406
7  0.021164    1.000000  3.634615               1.0  0.015341         inf
8  0.063492    0.230769  1.211538               1.0  0.011086    1.052381
9  0.063492    0.333333  1.211538               1.0  0.011086    1.087302

   zhangs_metric   jaccard  certainty  kulczynski
0       0.392701  0.083333   0.029392    0.240385
1       0.305682  0.083333   0.151039    0.240385
2      -0.189349  0.061538  -0.014311    0.156109
3      -0.156863  0.061538  -0.054959    0.156109
4       0.525547  0.070175   0.030769    0.260684
5       0.400000  0.070175   0.233577    0.260684
6       1.000000  0.076923   0.056965    0.538462
```

```
7       0.740541  0.076923  1.000000   0.538462
8       0.240876  0.157895  0.049774   0.282051
9       0.215686  0.157895  0.080292   0.282051
```

[70]:

```python
# Network graph to show association between words used in comments
G = nx.DiGraph()

for index, rule in rules.head(10).iterrows():
    antecedents = ' '.join(rule['antecedents'])
    consequents = ' '.join(rule['consequents'])
    support = rule['support']
    confidence = rule['confidence']

    G.add_edge(antecedents, consequents, weight=support)


plt.figure(figsize=(15,15))
pos = nx.spring_layout(G, seed=42)
edge_labels = nx.get_edge_attributes(G, 'weight')

nx.draw_networkx_nodes(G, pos, node_size=800, node_color='teal')
nx.draw_networkx_edges(G, pos, width=1.5, edge_color='gray')
nx.draw_networkx_labels(G, pos, font_size=10, font_color='black',␣
 ↪font_weight='bold')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

plt.show()
```



[71]:
```
# Emotion detection using NRCLex

# function for extracting the emotions
def top_emotion_extractor(top_emotion):
    emotions=[]
    for i in top_emotion:
        emotions.append(str(i[0]))
    return(emotions)

def return_bin(emotion,list_of_emo):
    if emotion in list_of_emo:
        boo=1
    else:
```

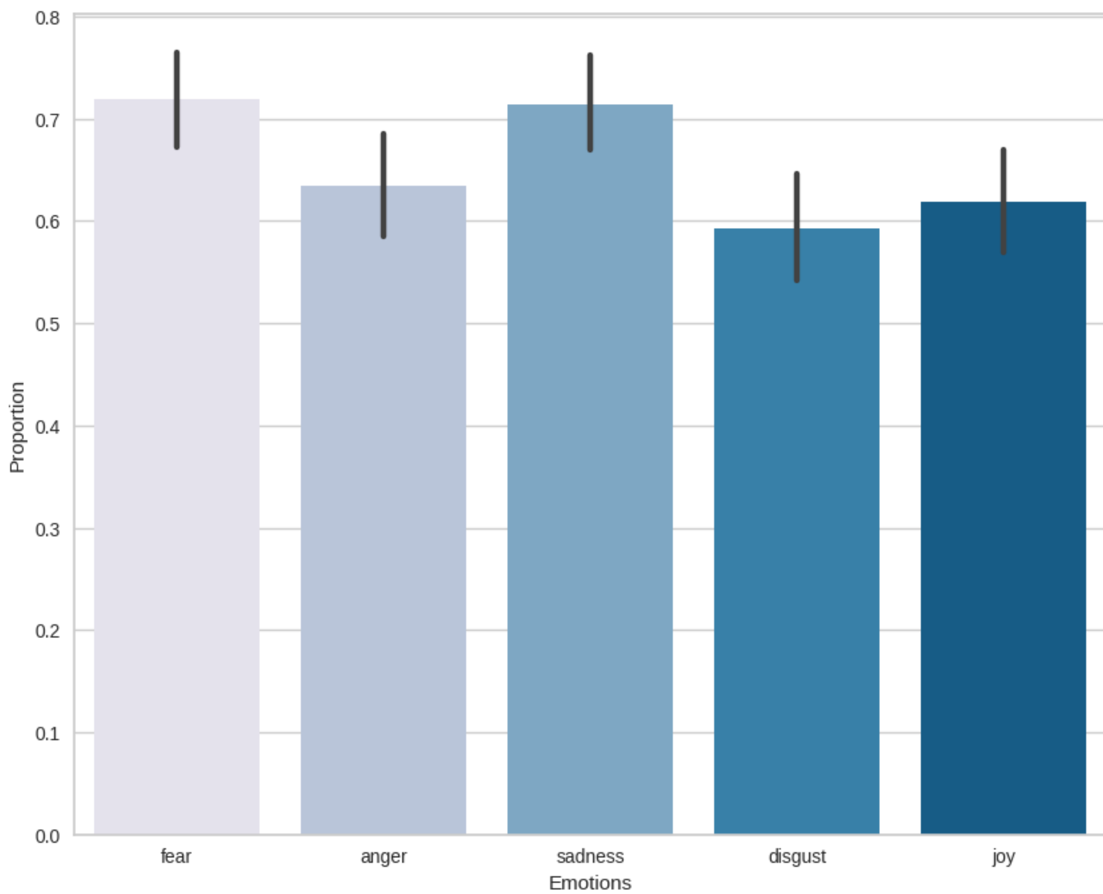
```
        boo=0
    return(boo)

# function for classifying emotions of "fear", "anger", "sadness", "joy" and␣
↪"disgust"
def emotion_classifier(text):
    emotions=top_emotion_extractor(NRCLex(text).top_emotions)
    fear=return_bin("fear",emotions)
    anger=return_bin("anger",emotions)
    sadness=return_bin("sadness",emotions)
    disgust=return_bin("disgust",emotions)
    joy=return_bin("joy",emotions)
    return (fear,anger,sadness,disgust,joy)

# Getting emotion for each comment
df3['fear'] = df3['comments_without_stopwords_joined'].apply(lambda s:␣
↪emotion_classifier(s)[0])
df3['anger'] = df3['comments_without_stopwords_joined'].apply(lambda s:␣
↪emotion_classifier(s)[1])
df3['sadness'] = df3['comments_without_stopwords_joined'].apply(lambda s:␣
↪emotion_classifier(s)[2])
df3['disgust'] = df3['comments_without_stopwords_joined'].apply(lambda s:␣
↪emotion_classifier(s)[3])
df3['joy'] = df3['comments_without_stopwords_joined'].apply(lambda s:␣
↪emotion_classifier(s)[4])
df3.head()
```

[71]:

```
                                             comment comments_tokenized  \
0  Toa predmet prirodni nauki za prirodni katastr…                ()
1  Da toa treba da ucat u skoloto za prirodni kat…                ()
2     Why arent they built as per Japanese standards                ()
3                                            Weather                ()
4                                              Today                ()

                         comments_without_stopwords  \
0  [toa, predmet, prirodni, nauki, za, prirodni, …
1  [da, toa, treba, da, ucat, u, skoloto, za, pri…
2           [arent, built, per, japanese, standards]
3                                          [weather]
4                                            [today]

                   comments_without_stopwords_joined  polarity sentiment  \
0  toa predmet prirodni nauki za prirodni katastr…       0.0   neutral
1  da toa treba da ucat u skoloto za prirodni kat…       0.0   neutral
2                 arent built per japanese standards       0.0   neutral
3                                            weather       0.0   neutral
```

```
4                                              today       0.0    neutral

  tokens_joined  clusters  fear  anger  sadness  disgust  joy
0                       0     1      1        1        1    1
1                       0     1      1        1        1    1
2                       0     1      1        1        1    1
3                       0     1      1        1        1    1
4                       0     1      1        1        1    1
```

[72]:
```python
# Bar plot for proportion of every emotion in the comments
emotions_df = df3[['fear', 'anger', 'sadness', 'disgust', 'joy']]
plt.figure(figsize=(10,8))
sns.barplot(emotions_df,palette="PuBu")

plt.xlabel('Emotions')
plt.ylabel('Proportion')
plt.show()
```



5 Model Implementation

[73]:
```
from sklearn.model_selection import train_test_split

# Assuming 'sentiment' is the target column with labels (e.g., positive,␣
↪neutral, negative)
y = df3['sentiment']  # Target variable

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Printing sizes of the splits
print("Training Set Size:", X_train.shape)
print("Testing Set Size:", X_test.shape)
```

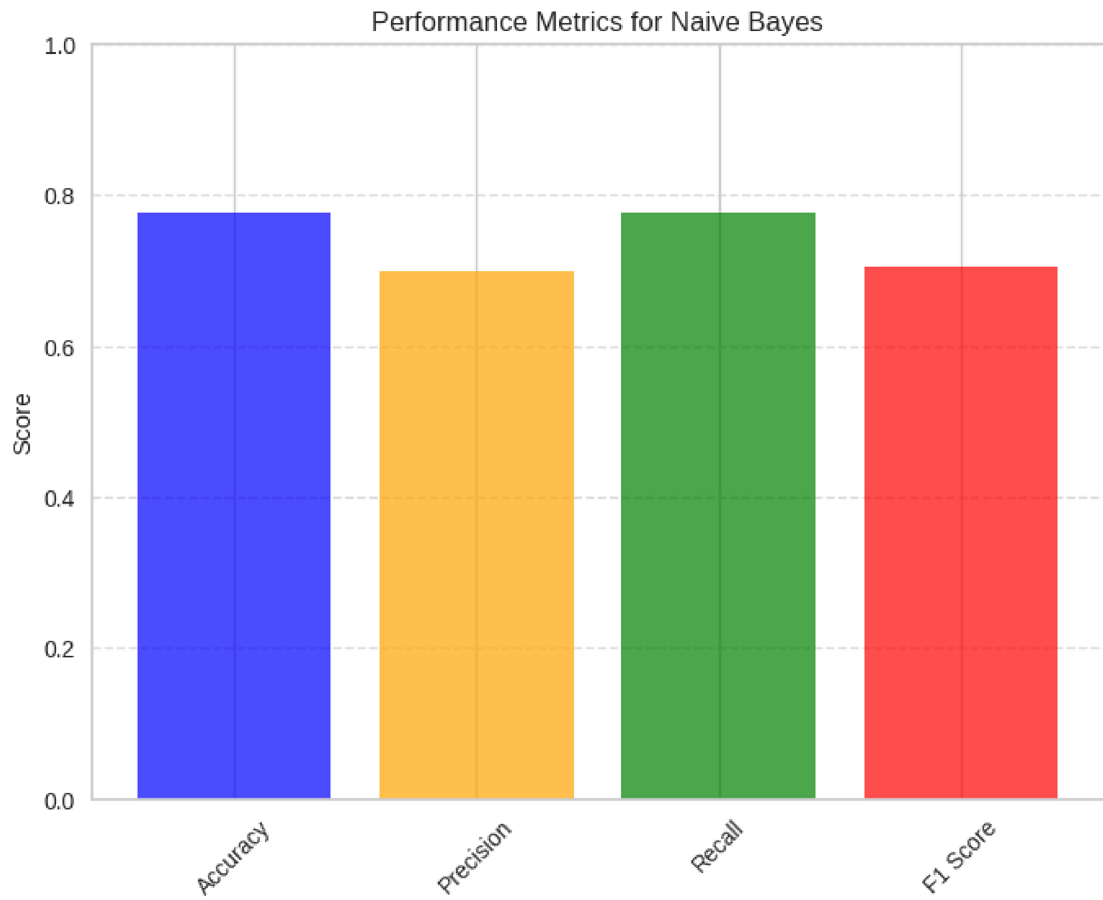
```
Training Set Size: (302, 1057)
Testing Set Size: (76, 1057)
```

5.1 1. Naive Bayes

[74]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

# Vectorizing the comments
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(df3['comments_without_stopwords_joined'])
y = df3['sentiment']  # Replace with the actual target column if different

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Modelling and evaluation
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

print("Naive Bayes Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Precision:", precision_score(y_test, y_pred_nb, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_nb, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_nb, average='weighted'))
```

```python
# Metrics for Naive Bayes
naive_bayes_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_nb),
    "Precision": precision_score(y_test, y_pred_nb, average='weighted'),
    "Recall": recall_score(y_test, y_pred_nb, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_nb, average='weighted')
}

# Plotting Naive Bayes metrics
plt.figure(figsize=(8, 6))
plt.bar(naive_bayes_metrics.keys(), naive_bayes_metrics.values(),␣
 ↪color=['blue', 'orange', 'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Naive Bayes")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Naive Bayes Results:
Accuracy: 0.7763157894736842
Precision: 0.6975511695906432
Recall: 0.7763157894736842
F1 Score: 0.7040060781875951
```



5.2 2. Logistic Regression

[75]:

```python
from sklearn.linear_model import LogisticRegression

# Modelling and evaluation
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)
y_pred_logreg = logreg_model.predict(X_test)

print("Logistic Regression Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Precision:", precision_score(y_test, y_pred_logreg, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_logreg, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_logreg, average='weighted'))

# Metrics for Logistic Regression
logistic_regression_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_logreg),
```

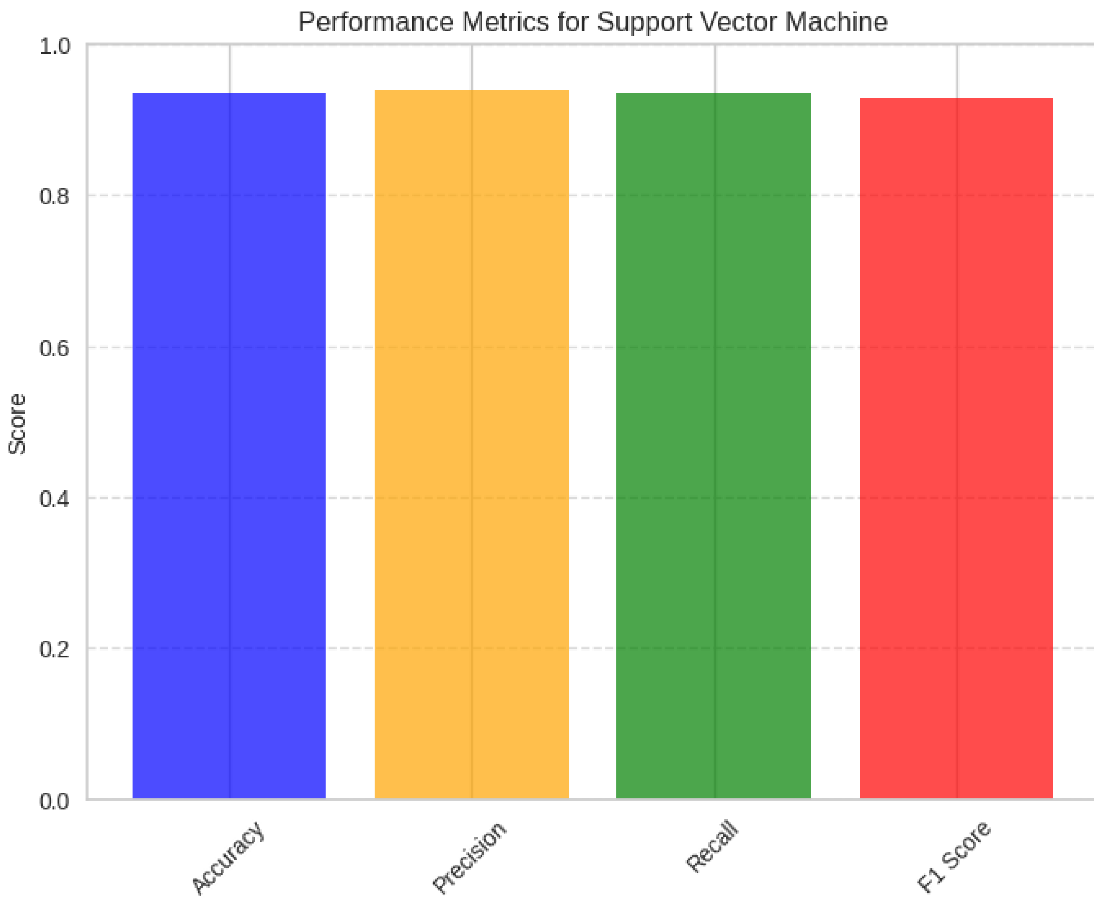
```
    "Precision": precision_score(y_test, y_pred_logreg, average='weighted'),
    "Recall": recall_score(y_test, y_pred_logreg, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_logreg, average='weighted')
}

# Plotting Logistic Regression metrics
plt.figure(figsize=(8, 6))
plt.bar(logistic_regression_metrics.keys(), logistic_regression_metrics.
 ↪values(), color=['blue', 'orange', 'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Logistic Regression")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Logistic Regression Results:
Accuracy: 0.7368421052631579
Precision: 0.6754385964912281
Recall: 0.7368421052631579
F1 Score: 0.6364709851551957
```

5.3 3. Support Vector Machine (SVM)

[76]:
```
from sklearn.svm import SVC

# Modelling and evaluation
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)

print("Support Vector Machine Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_svm, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_svm, average='weighted'))

# Metrics for SVM
svm_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_svm),
    "Precision": precision_score(y_test, y_pred_svm, average='weighted'),
    "Recall": recall_score(y_test, y_pred_svm, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_svm, average='weighted')
}

# Plotting SVM metrics
plt.figure(figsize=(8, 6))
plt.bar(svm_metrics.keys(), svm_metrics.values(), color=['blue', 'orange',␣
↪'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Support Vector Machine")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Support Vector Machine Results:
Accuracy: 0.9342105263157895
Precision: 0.9396929824561402
Recall: 0.9342105263157895
F1 Score: 0.9287485016890051
```



5.4 4. Decision Tree

[77]:
```
from sklearn.tree import DecisionTreeClassifier

# Modelling and evaluation
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

print("Decision Tree Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_dt, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_dt, average='weighted'))

# Metrics for Decision Tree
decision_tree_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_dt),
```

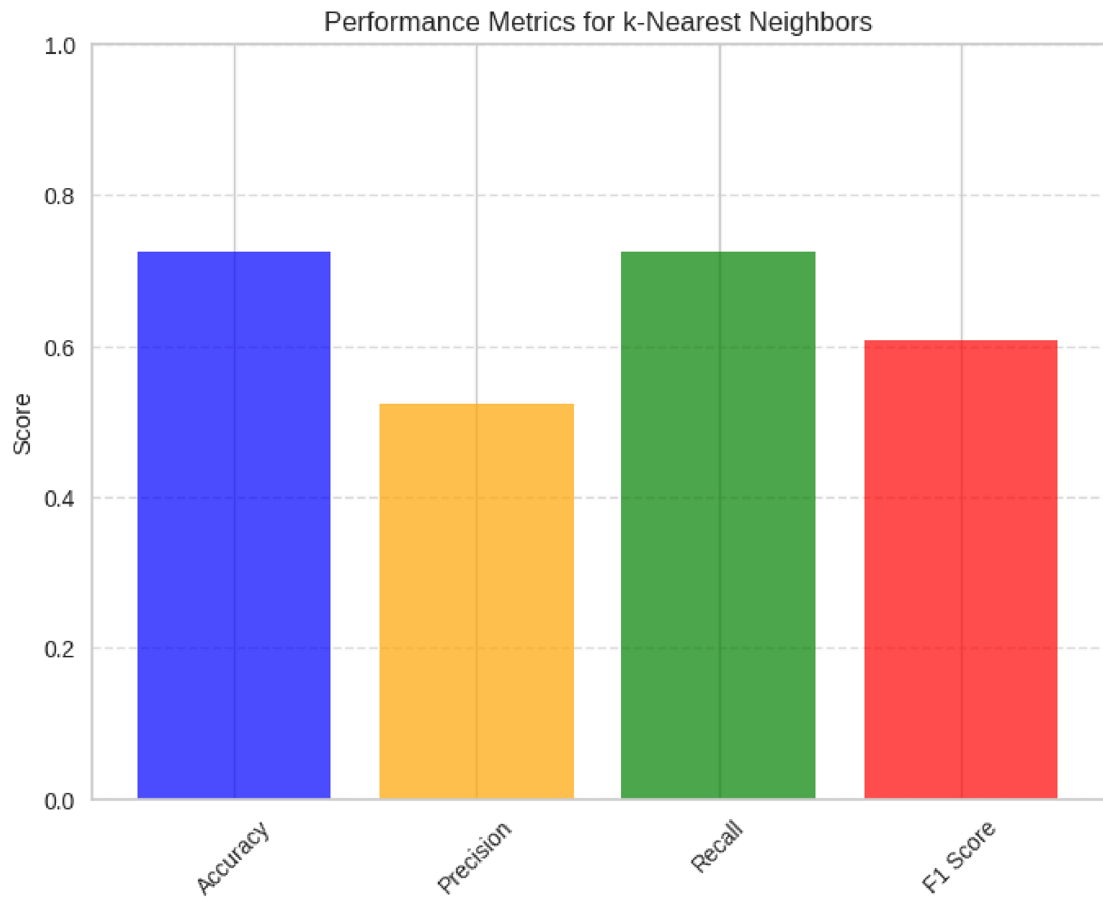
```
    "Precision": precision_score(y_test, y_pred_dt, average='weighted'),
    "Recall": recall_score(y_test, y_pred_dt, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_dt, average='weighted')
}

# Plotting Decision Tree metrics
plt.figure(figsize=(8, 6))
plt.bar(decision_tree_metrics.keys(), decision_tree_metrics.values(),␣
 ↪color=['blue', 'orange', 'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for Decision Tree")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Decision Tree Results:
Accuracy: 0.7105263157894737
Precision: 0.6954233409610984
Recall: 0.7105263157894737
F1 Score: 0.6572403791737408
```

5.5 5. k-Nearest Neighbors (kNN)

[78]:
```python
from sklearn.neighbors import KNeighborsClassifier

# Modelling and evaluation
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)

print("k-Nearest Neighbors Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_knn, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred_knn, average='weighted'))

# Collecting metrics for visualization
knn_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_knn),
    "Precision": precision_score(y_test, y_pred_knn, average='weighted'),
    "Recall": recall_score(y_test, y_pred_knn, average='weighted'),
    "F1 Score": f1_score(y_test, y_pred_knn, average='weighted')
}

# Plotting metrics
plt.figure(figsize=(8, 6))
plt.bar(knn_metrics.keys(), knn_metrics.values(), color=['blue', 'orange',␣
↪'green', 'red'], alpha=0.7)
plt.title("Performance Metrics for k-Nearest Neighbors")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
k-Nearest Neighbors Results:
Accuracy: 0.7236842105263158
Precision: 0.523718836565097
Recall: 0.7236842105263158
F1 Score: 0.6076737645640818
```



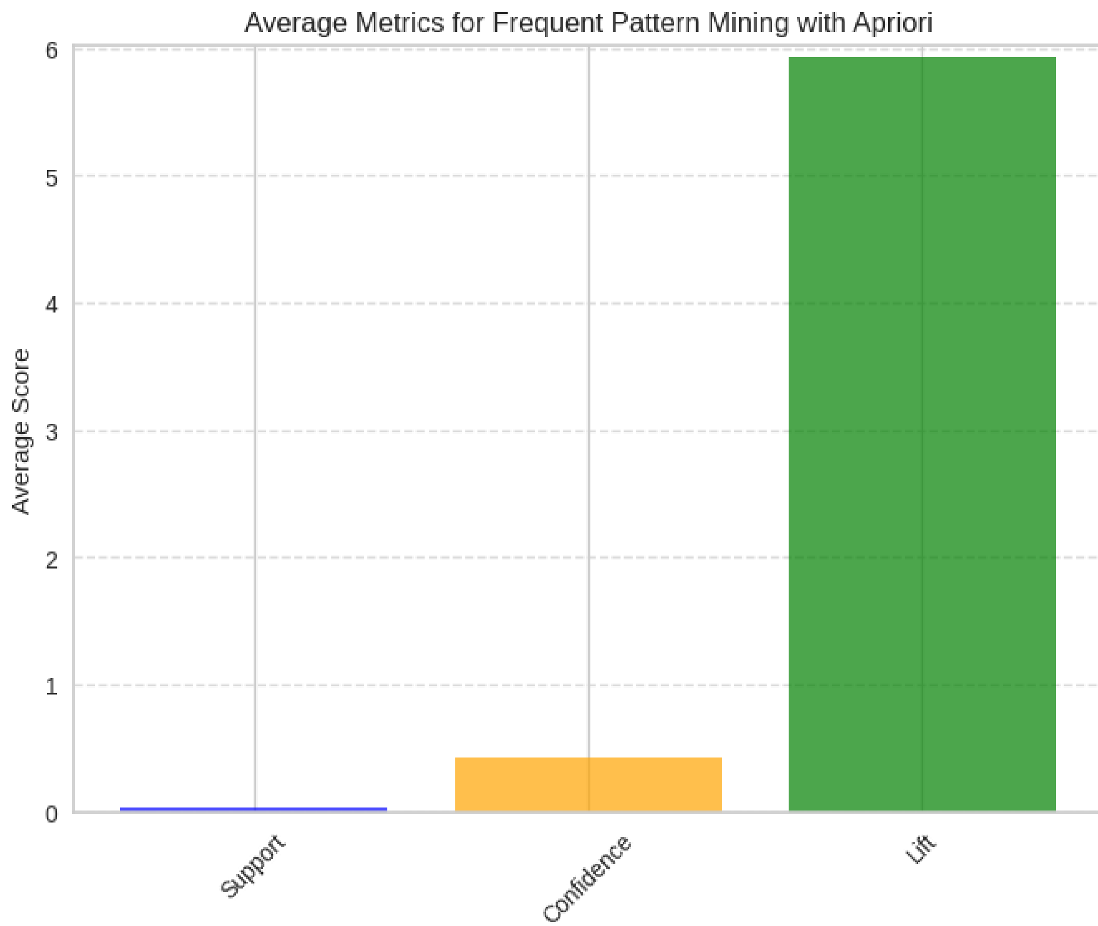
5.6 6. Apriori Algorithm

[79]:
```
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

# Preparing transactions from comments
transactions = df3['comments_without_stopwords'].tolist()
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df_trans = pd.DataFrame(te_array, columns=te.columns_)

# Applying Apriori
frequent_itemsets = apriori(df_trans, min_support=0.02, use_colnames=True)

# Generating Association Rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0,␣
↪num_itemsets=None)
```

```python
print("Top 10 Association Rules:")
print(rules.head(10))

# Plotting support, confidence, and lift
rules_metrics = {
    "Support": rules['support'].mean(),
    "Confidence": rules['confidence'].mean(),
    "Lift": rules['lift'].mean()
}

plt.figure(figsize=(8, 6))
plt.bar(rules_metrics.keys(), rules_metrics.values(), color=['blue', 'orange',␣
 ↪'green'], alpha=0.7)
plt.title("Average Metrics for Frequent Pattern Mining with Apriori")
plt.ylabel("Average Score")
plt.ylim(0, max(rules_metrics.values()) + 0.1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Top 10 Association Rules:
  antecedents consequents  antecedent support  consequent support   support  \
0          ()     (allah)            0.275132            0.068783  0.026455
1     (allah)          ()            0.068783            0.275132  0.026455
2          ()      (like)            0.275132            0.047619  0.021164
3      (like)          ()            0.047619            0.275132  0.021164
4          ()       (saw)            0.275132            0.021164  0.021164
5       (saw)          ()            0.021164            0.275132  0.021164
6          ()    (turkey)            0.275132            0.190476  0.063492
7    (turkey)          ()            0.190476            0.275132  0.063492
8          ()     (would)            0.275132            0.026455  0.021164
9     (would)          ()            0.026455            0.275132  0.021164

   confidence      lift  representativity  leverage  conviction  \
0    0.096154  1.397929               1.0  0.007531    1.030283
1    0.384615  1.397929               1.0  0.007531    1.177910
2    0.076923  1.615385               1.0  0.008062    1.031746
3    0.444444  1.615385               1.0  0.008062    1.304762
4    0.076923  3.634615               1.0  0.015341    1.060406
5    1.000000  3.634615               1.0  0.015341         inf
6    0.230769  1.211538               1.0  0.011086    1.052381
7    0.333333  1.211538               1.0  0.011086    1.087302
8    0.076923  2.907692               1.0  0.013885    1.054674
9    0.800000  2.907692               1.0  0.013885    3.624339

   zhangs_metric   jaccard  certainty  kulczynski
0       0.392701  0.083333   0.029392    0.240385
```

1	0.305682	0.083333	0.151039	0.240385
2	0.525547	0.070175	0.030769	0.260684
3	0.400000	0.070175	0.233577	0.260684
4	1.000000	0.076923	0.056965	0.538462
5	0.740541	0.076923	1.000000	0.538462
6	0.240876	0.157895	0.049774	0.282051
7	0.215686	0.157895	0.080292	0.282051
8	0.905109	0.075472	0.051839	0.438462
9	0.673913	0.075472	0.724088	0.438462

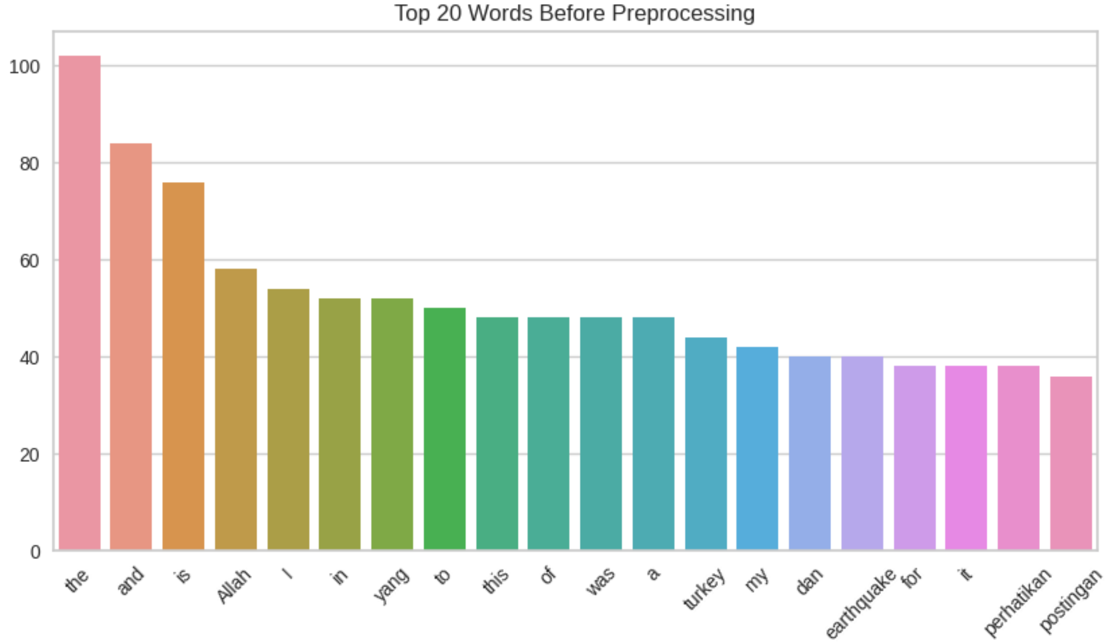


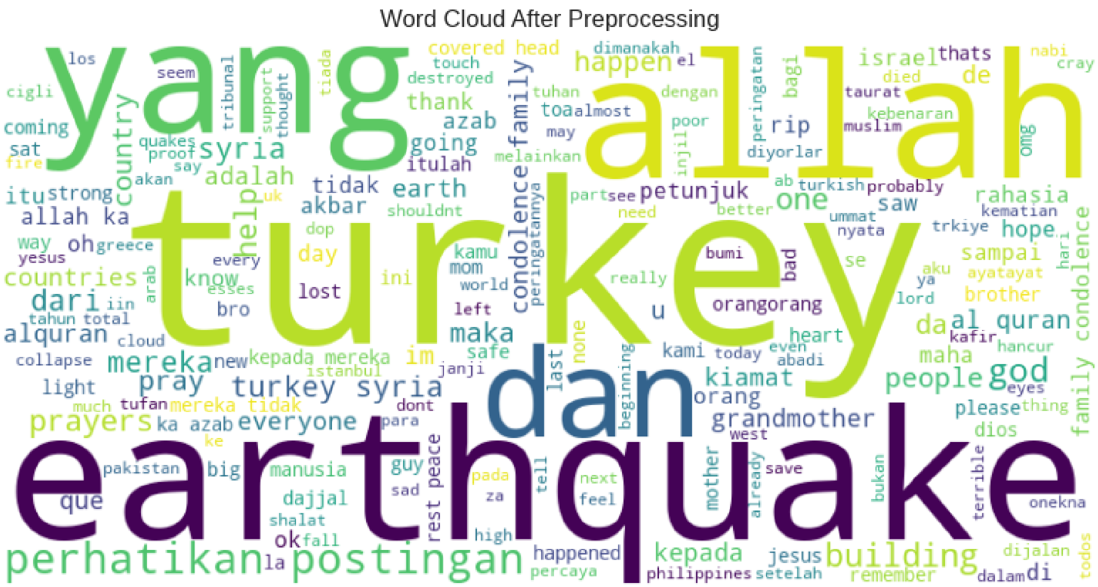
[80]:
```
def plot_wordcloud(text_data, title):
    wordcloud = WordCloud(width=800, height=400, background_color="white").
 ↪generate(" ".join(text_data))
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.title(title)
    plt.show()
```

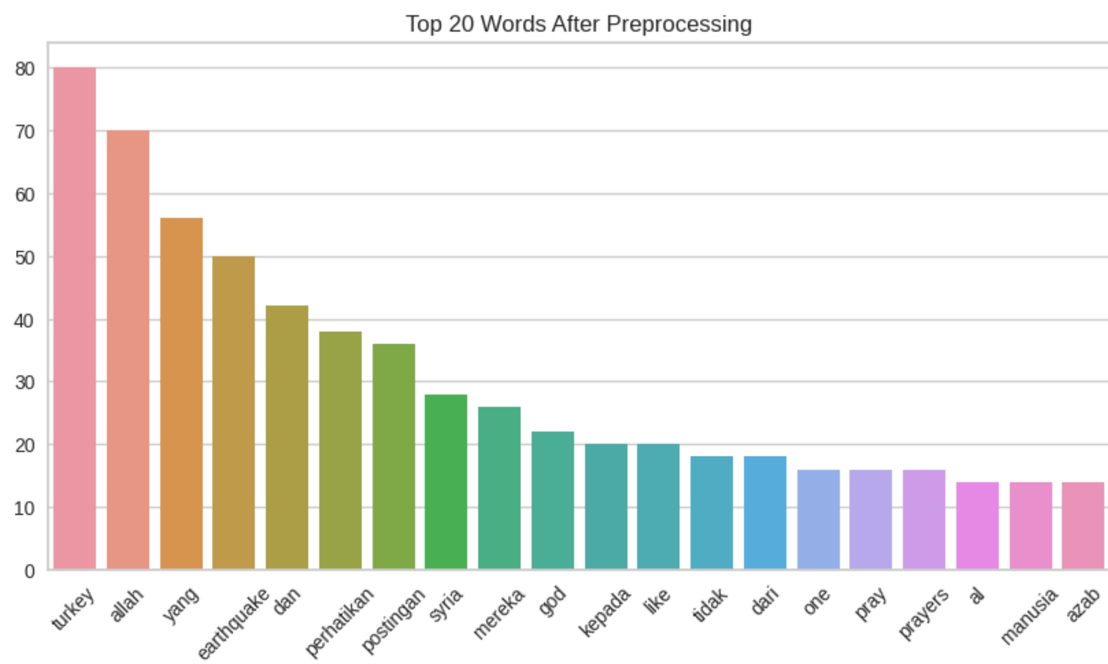
```python
# Function to plot top word frequencies
def plot_top_words(text_data, title):
    all_words = " ".join(text_data).split()
    word_freq = Counter(all_words)
    top_words = word_freq.most_common(20)

    words = [w[0] for w in top_words]
    counts = [w[1] for w in top_words]

    plt.figure(figsize=(10, 5))
    sns.barplot(x=words, y=counts)
    plt.xticks(rotation=45)
    plt.title(title)
    plt.show()

### Before Preprocessing ###

# Plotting word cloud before preprocessing
raw_text_data = df3['comment'].astype(str).tolist()
plot_wordcloud(raw_text_data, "Word Cloud Before Preprocessing")

# Plotting top word frequencies before preprocessing
plot_top_words(raw_text_data, "Top 20 Words Before Preprocessing")

### After Preprocessing ###

# Plotting word cloud after preprocessing
cleaned_text_data = df3['comments_without_stopwords_joined'].tolist()
plot_wordcloud(cleaned_text_data, "Word Cloud After Preprocessing")

# Plotting top word frequencies after preprocessing
plot_top_words(cleaned_text_data, "Top 20 Words After Preprocessing")
```

Word Cloud Before Preprocessing

Top 20 Words Before Preprocessing




Word Cloud After Preprocessing
turkey
earthquake
allah
yang
dan
perhatikan
postingan
turkey syria
building
prayers
mereka
syria
kiamat
grandmother
people
god
al quran
alqurran
dari
pray
everyone


Top 20 Words After Preprocessing
0
10
20
30
40
50
60
70
80
turkey
allah
yang
earthquake
dan
perhatikan
postingan
syria
mereka
god
kepada
like
tidak
dari
one
pray
prayers
al
manusia
azab