twitter-bot-da

March 30, 2024

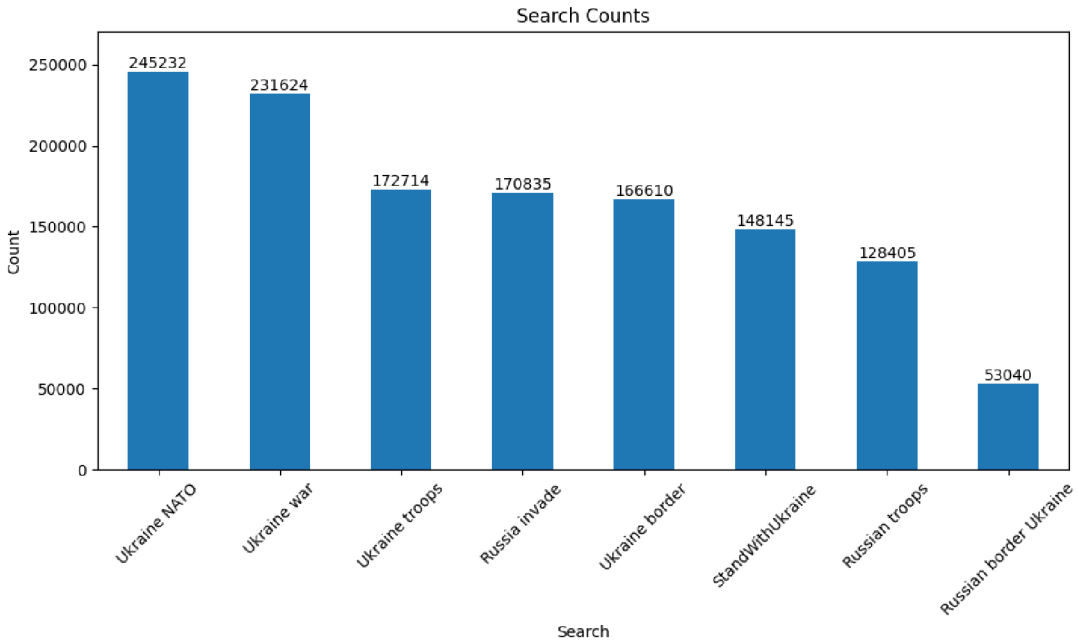
[ ]:
```
import os
import pandas as pd
import numpy as np
import glob
from langdetect import detect
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

[ ]:
```
## First we are merging all the datasets as they are in the same structure.

file_paths = glob.glob('C:\\Users\\dadashza\\Downloads\\archive\\*.csv')

dataframes = []

for file in file_paths:
    df = pd.read_csv(file)
    dataframes.append(df)

merged_df = pd.concat(dataframes)
merged_df = merged_df.reset_index(drop=True)
```

[ ]:
```
import ast

df_english['verification_status'] = df_english['user'].apply(lambda x:␣
 ↪'verified' if isinstance(x, str) else 'non-verified')
df_english['verification_status'] = df_english['verification_status'].
 ↪where(df_english['verification_status'] == 'non-verified',␣
 ↪df_english['user'].apply(lambda x: 'verified' if ast.
 ↪literal_eval(x)['verified'] else 'non-verified'))
```

[4]:
```
# Convert the string values in the 'user' column to dictionaries
df_english['user'] = df_english['user'].apply(lambda x: ast.literal_eval(x))
# Create a new column with the number of followers from the dictionary
df_english['followers_count'] = df_english['user'].apply(lambda x: x.
 ↪get('followersCount'))
```

[6]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Calculate value counts
search_counts = merged_df['Searh'].value_counts()

# Create a bar chart
fig, ax = plt.subplots(figsize=(10, 6))
bar_plot = search_counts.plot(kind='bar')

# Add count numbers on top of each bar
for i, value in enumerate(search_counts):
    bar_plot.text(i, value, str(value), ha='center', va='bottom')

plt.title('Search Counts')
plt.xlabel('Search')
plt.ylabel('Count')
plt.xticks(rotation=45)

# Set the x-axis limits
plt.xlim(-0.5, len(search_counts) - 0.5)
plt.ylim(0, 270000)  # Set the maximum value of the y-axis

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



[9]:
```
# Find date period
print(merged_df['date'].min())
print(merged_df['date'].max())
```

```
2021-12-31 00:00:30+00:00
2022-03-05 23:59:59+00:00
```

[10]:
```
# Function to detect language
def detect_language(text):
    try:
        lang = detect(text)
        return lang
    except:
        return 'unknown'

# Apply language detection to each content and create a new column
merged_df['language'] = merged_df['content'].apply(detect_language)
```

[ ]:
```
# Convert the string values in the 'user' column to dictionaries
merged_df['user'] = merged_df['user'].apply(lambda x: ast.literal_eval(x))
# Create a new column with the number of followers from the dictionary
merged_df['followers_count'] = merged_df['user'].apply(lambda x: x.
 ↪get('followersCount'))
```

[23]:
```
import pandas as pd

# Define the follower count ranges
bins = [0, 50, 100, 200, float('inf')]
labels = ['0-50', '51-100', '101-200', '200+']

# Create a new column with the follower count ranges
merged_df['follower_range'] = pd.cut(merged_df['followers_count'], bins=bins,␣
 ↪labels=labels, right=False)

# Create the frequency table
frequency_table = merged_df['follower_range'].value_counts().reset_index()
frequency_table.columns = ['follower_range', 'count']

# Calculate the percentage column
total_accounts = len(merged_df)
frequency_table['percentage'] = (100 * frequency_table['count'] /␣
 ↪total_accounts).round(2)

# Calculate the cumulative percentage column
frequency_table['cumulative_percentage'] = frequency_table['percentage'].
 ↪cumsum().round(2)
```

```
# Calculate the total number of accounts
total_accounts_row = pd.DataFrame({'follower_range': ['Total'], 'count':␣
↪[total_accounts],
                                   'percentage': [100], 'cumulative_percentage':␣
↪[100]})

# Concatenate the total row to the frequency table
frequency_table = pd.concat([frequency_table, total_accounts_row],␣
↪ignore_index=True)

# Sort the frequency table by 'follower_range' in ascending order
frequency_table = frequency_table.sort_values('follower_range')
```

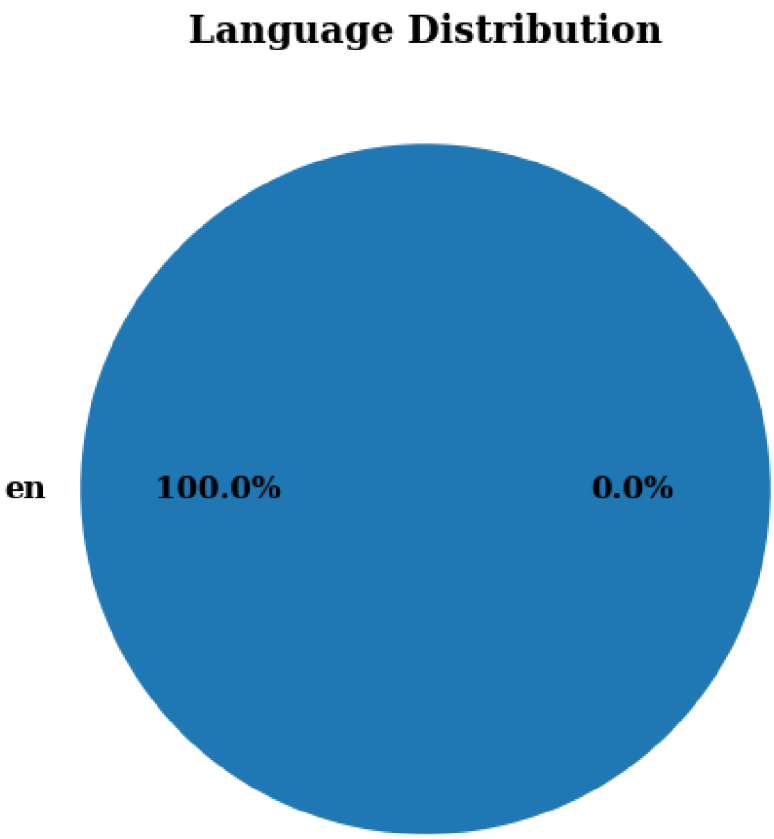
	follower_range	count	percentage	cumulative_percentage
1	0-50	297041	22.56	81.74
2	101-200	131227	9.97	91.71
0	200+	779140	59.18	59.18
3	51-100	109197	8.29	100.00
4	Total	1316605	100.00	100.00

[27]:
```
# Sort the frequency table by 'percentage' in ascending order
frequency_table = frequency_table.sort_values('percentage', ascending=True)
```

[30]:
```
# Save the frequency table as a CSV file
frequency_table.to_csv('frequency_table_1.csv', index=False)
```

[5]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Get the value counts of the 'language' column

language_counts = df_english['language'].value_counts()

# Calculate the percentage of each language
language_percentages = language_counts / language_counts.sum() * 100

# Select the top 3 languages
top_languages = language_percentages.nlargest(3)

# Combine the remaining languages as "Other Languages"
other_languages = language_percentages[~language_percentages.index.
↪isin(top_languages.index)]
other_languages_total = other_languages.sum()

# Create a new series with the top 3 languages and "Other Languages"
final_language_counts = pd.concat([top_languages, pd.
↪Series([other_languages_total], index=["Other Languages"])])
```

```
# Plot the pie chart with custom fonts
plt.figure(figsize=(8, 6))
plt.rcParams['font.family'] = 'serif'  # Set font family
plt.rcParams['font.size'] = 12  # Set font size
plt.rcParams['font.weight'] = 'bold'  # Set font weight
plt.rcParams['text.color'] = 'black'  # Set text color
plt.rcParams['axes.labelcolor'] = 'black'  # Set axes label color
plt.rcParams['axes.titlecolor'] = 'black'  # Set axes title color
final_language_counts.plot.pie(autopct='%1.1f%%')
plt.title("Language Distribution", fontweight='bold')  # Set title font weight
plt.ylabel("")
plt.show()
```



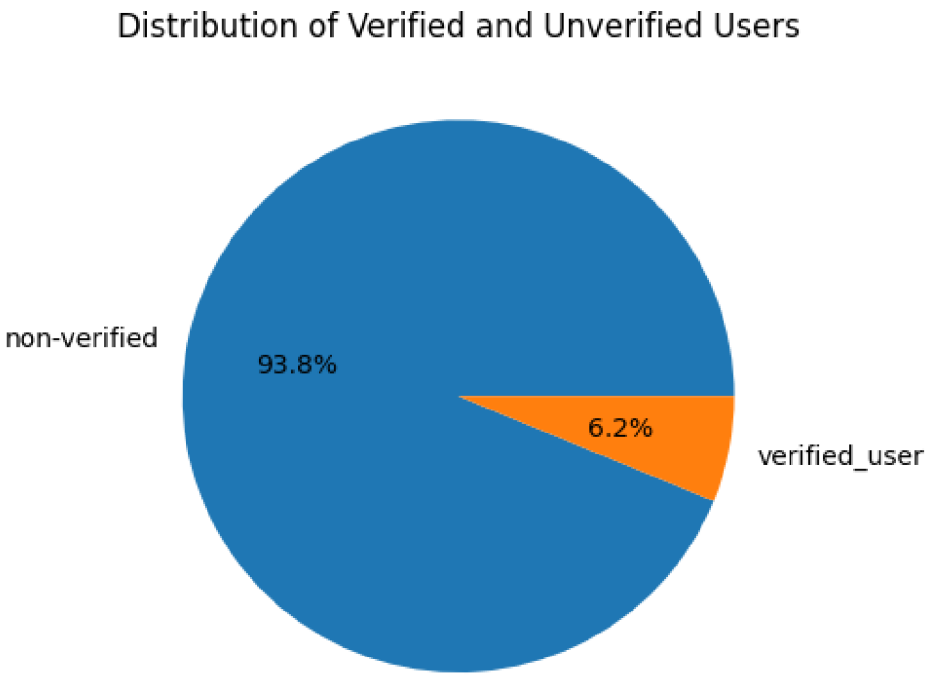
[116]:
```
import ast
```

```
df_english['verification_status'] = df_english['user'].apply(lambda x:␣
 ↪'verified' if isinstance(x, str) else 'non-verified')
df_english['verification_status'] = df_english['verification_status'].
 ↪where(df_english['verification_status'] == 'non-verified',␣
 ↪df_english['user'].apply(lambda x: 'verified' if ast.
 ↪literal_eval(x)['verified'] else 'non-verified'))
```

[117]:
```
df_english['verification_status'].value_counts()
```

[117]:
```
verification_status
non-verified    1143745
verified          77280
Name: count, dtype: int64
```

[38]:
```
import matplotlib.pyplot as plt

# Count the number of verified and unverified users
verification_counts = merged_df['verification_status'].value_counts()

# Create a pie chart
plt.pie(verification_counts.values, labels=verification_counts.index,␣
 ↪autopct='%1.1f%%')
plt.title('Distribution of Verified and Unverified Users')
plt.show()
```

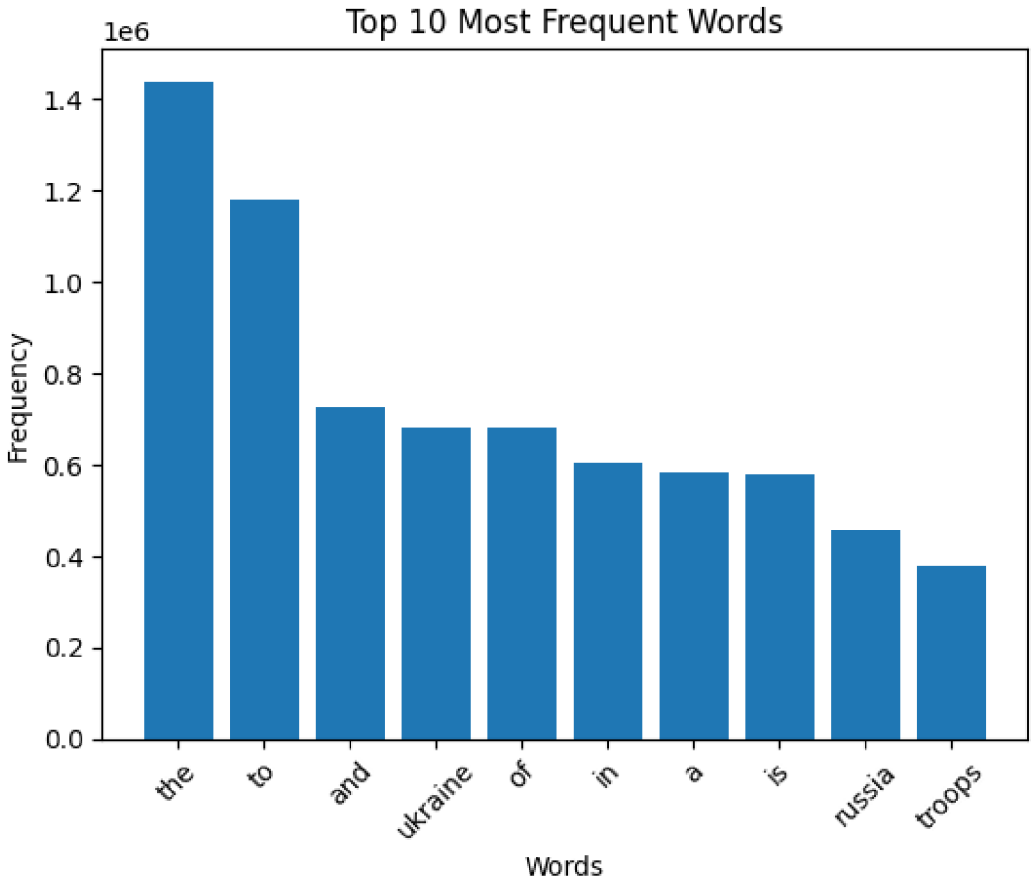


[58]:
```
import pandas as pd

# Calculate language counts
language_counts = merged_df['language'].value_counts()

# Create a sorted table
language_table = pd.DataFrame({'Language': language_counts.index, 'Number of␣
↪Occurrences': language_counts}).sort_values(by='Number of Occurrences',␣
↪ascending=False)

# Display the table
print(language_table.to_string(index=False))
```

```
Language  Number of Occurrences
      en                1221025
      de                  51531
      et                   4040
      pl                   3988
      ja                   3543
      pt                   3391
      fr                   3051
      it                   2794
      af                   2349
      uk                   2263
      es                   1970
      tr                   1644
      ru                   1549
      id                   1404
      da                   1340
      fi                   1312
      nl                   1164
      no                   1124
      ar                    652
      sv                    643
      cs                    570
      hr                    483
      hi                    457
      bg                    399
      sw                    359
      lt                    332
      ca                    292
      vi                    285
      tl                    272
      fa                    260
      lv                    255
```

```
el          231
sl          209
so          164
ta          156
ro          147
th          133
sk          123
sq          111
ko           83
unknown      82
zh-cn        52
mk           51
he           41
ml           38
hu           38
cy           33
te           32
bn           32
ur           28
kn           21
gu           20
pa           15
zh-tw        10
mr            9
ne            5
```

[43]:
```python
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt

# Tokenize the text and count word frequencies
words = [word.lower() for text in df_english['content'] for word in text.
 ↪split()]
word_freq = Counter(words)

# Get the most common words and their frequencies
top_words = word_freq.most_common(10)
top_words, freq = zip(*top_words)

# Create a bar chart of the word frequencies
plt.bar(top_words, freq)
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 10 Most Frequent Words')
plt.xticks(rotation=45)
plt.show()
```



[5]:
```
df_english = merged_df[merged_df['language'] == 'en']
```

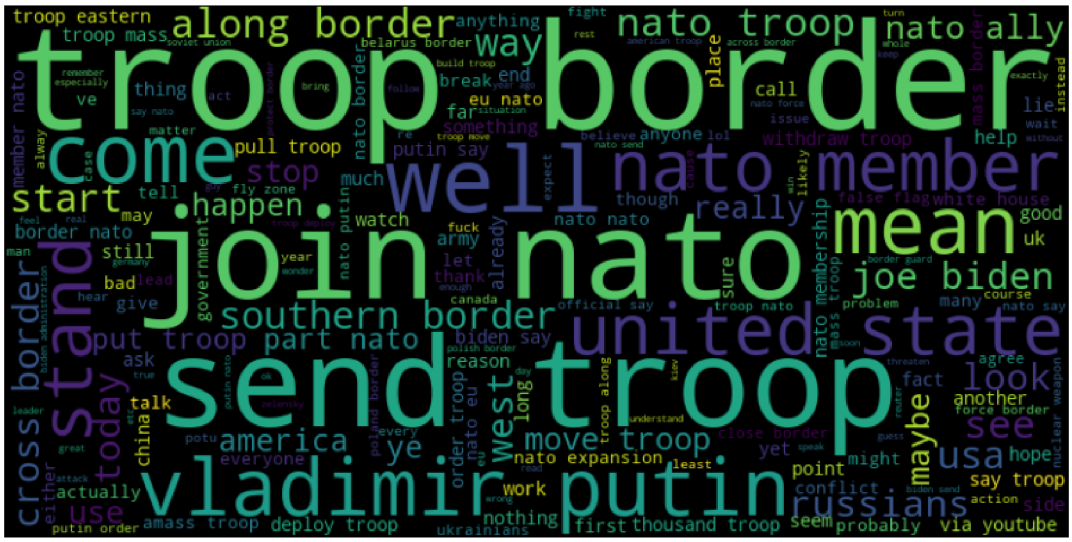
[ ]:
```
df_english.head()
```

[ ]:
```
import spacy
from spacy.lang.en import English
from nltk.corpus import stopwords
import re
import pandas as pd

# Load the English language model in spaCy
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

# Download stopwords (optional)
stop_words = set(stopwords.words('english'))

# Custom words to remove
```

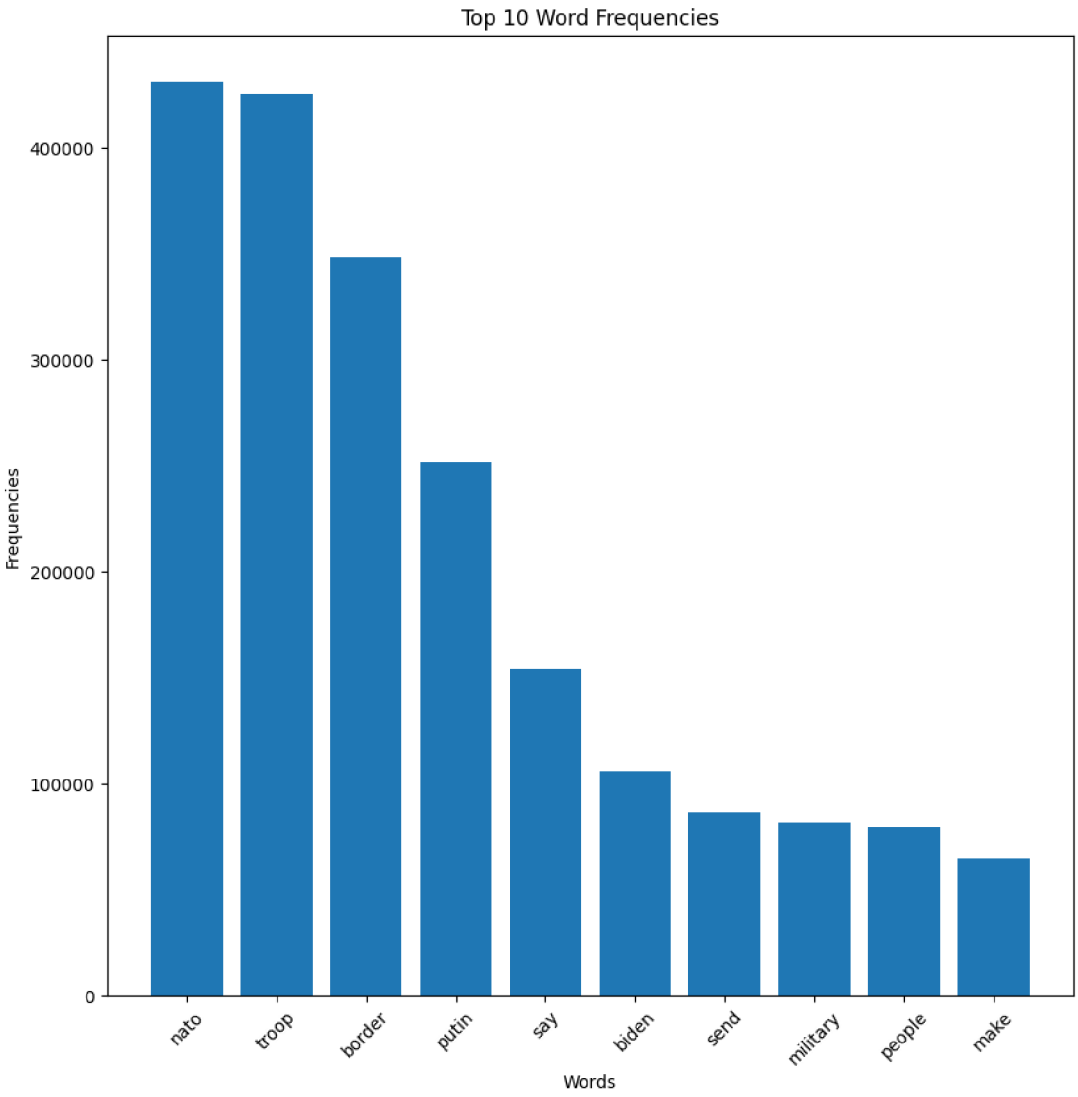
```
custom_words = ['Ukraine', 'Russia', 'Russian', 'war', 'US', 'invade', 'would',␣
 ↪'amp', 'StandWithUkraine', 'invasion',
                'country', 'Ukrainian', 'like', 'want', 'going', 'Europe',␣
 ↪'think', 'world', 'says', 'said', 'one',
                'countries', 'get', 'right', 'back', 'know', 'go', 'time',␣
 ↪'President', 'near', 'support', 'need', 'even', 'take']

# Function to remove stopwords, special characters, and lemmatization using␣
 ↪spaCy
def preprocess_text(text):
    # Tokenize the text
    doc = nlp(text)

    # Remove stopwords, special characters, and custom words
    filtered_tokens = [re.sub(r'[^\w\s]', '', token.lemma_.lower()) for token␣
 ↪in doc if token.lemma_.lower() not in stop_words.union(custom_words)]
    filtered_tokens = [token for token in filtered_tokens if len(token) > 1]  #␣
 ↪Remove individual characters

    return ' '.join(filtered_tokens)

# Apply the preprocess_text function to the 'content_cleaned' column
df_english['content_preprocessed'] = df_english['content_cleaned'].
 ↪apply(preprocess_text)

# Print the updated DataFrame
print(df_english)
```

[ ]:
```
df_english['content_preprocessed'] = df_english['content_preprocessed'].
 ↪apply(lambda x: ' '.join(word for word in x.split() if word.lower() not in␣
 ↪[custom_word.lower() for custom_word in custom_words]))
```

[16]:
```
## WORDCLOUD

text = ' '.join(df_english['content_preprocessed'].tolist())

wordcloud = WordCloud(width=800, height=400, max_font_size=150,␣
 ↪random_state=42).generate(text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

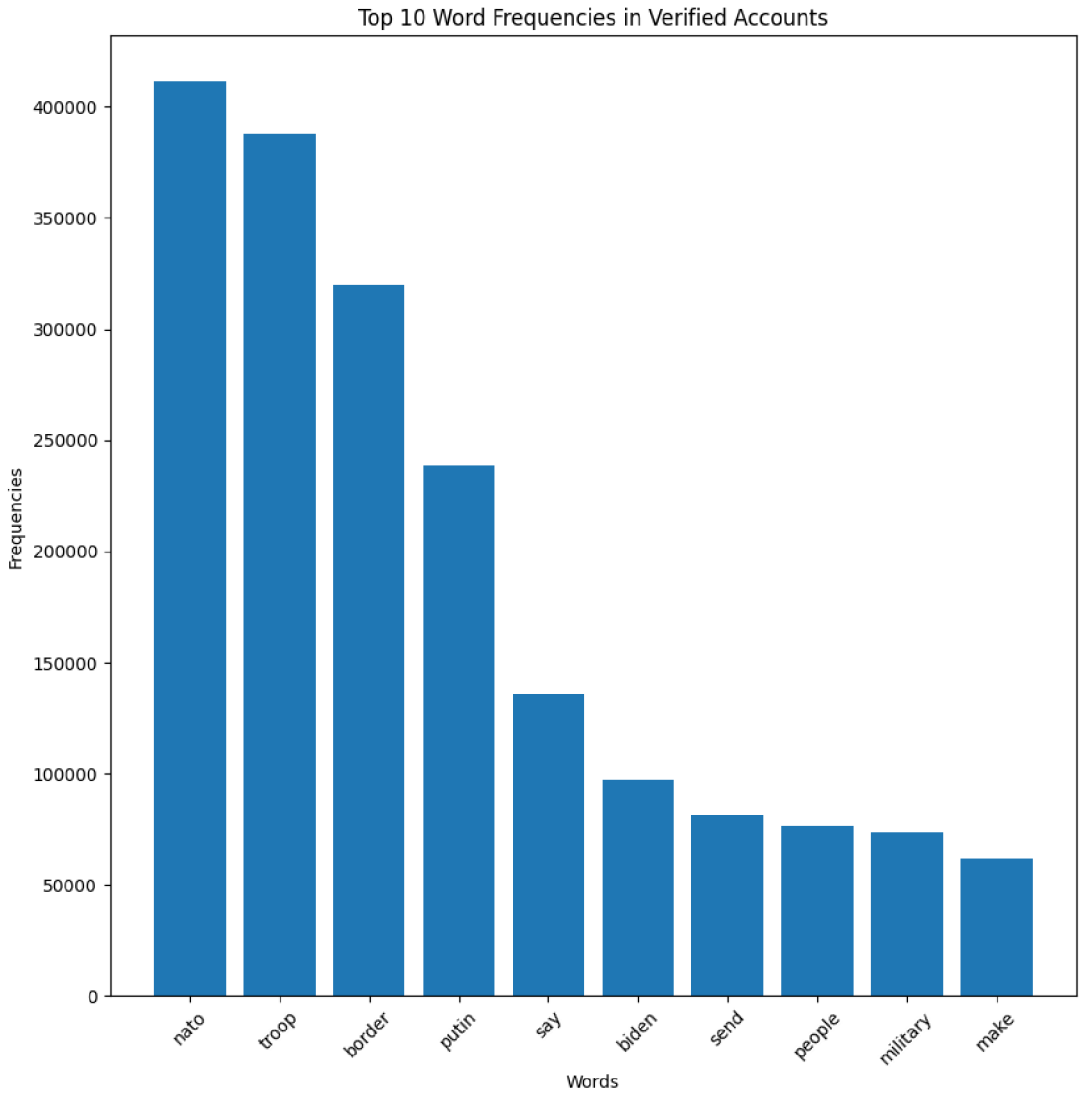


[17]:
```
from collections import Counter

text = ' '.join(df_english['content_preprocessed'].tolist())

# Tokenize the text into individual words
words = text.split()

# Count the frequencies of each word
word_freq = Counter(words)

# Get the top 10 words and their frequencies
top_words = [word for word, freq in word_freq.most_common(10)]
top_freqs = [freq for word, freq in word_freq.most_common(10)]

# Create the bar chart
plt.figure(figsize=(10, 10))
plt.bar(top_words, top_freqs)
plt.xlabel('Words')
plt.ylabel('Frequencies')
plt.title('Top 10 Word Frequencies')
plt.xticks(rotation=45)
plt.show()
```



[51]:
```
from collections import Counter

# Filter the dataframe for rows with verified accounts
verified_df = df_english[df_english['verification_status'] == 'non-verified']

# Join the preprocessed text from the filtered rows
text = ' '.join(verified_df['content_preprocessed'].tolist())

# Tokenize the text into individual words
words = text.split()

# Count the frequencies of each word
```

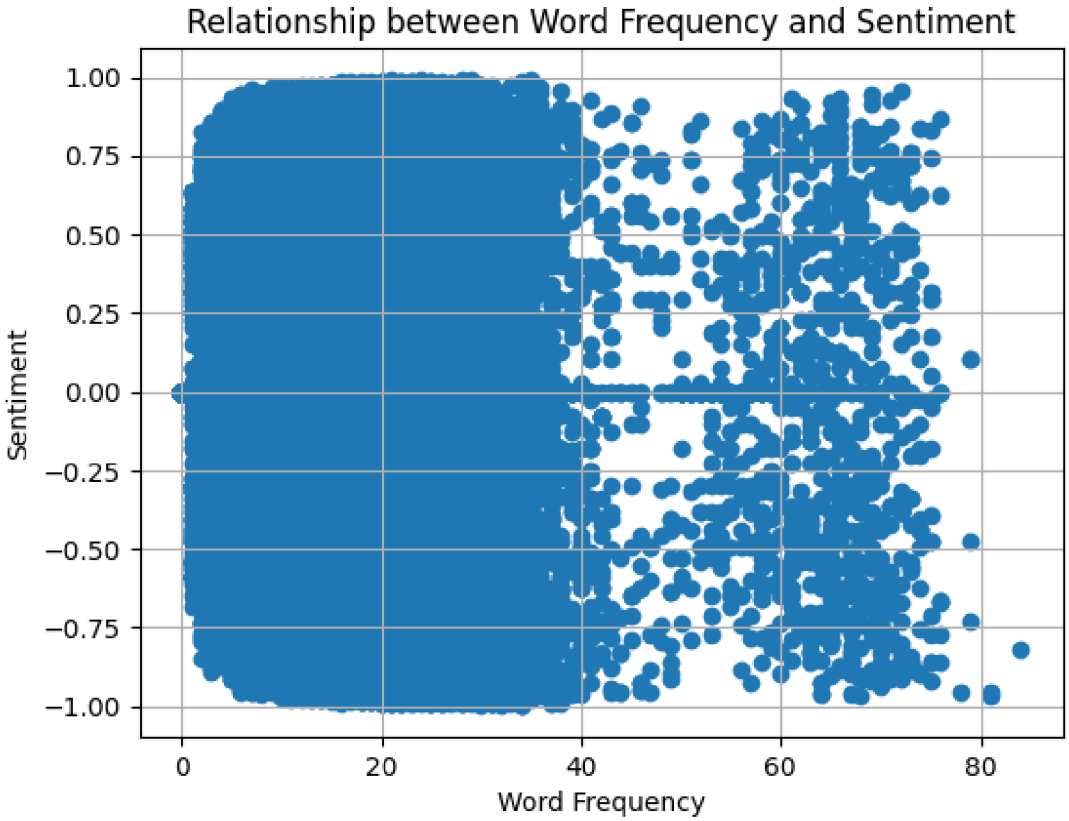
```
word_freq = Counter(words)

# Get the top 10 words and their frequencies
top_words = [word for word, freq in word_freq.most_common(10)]
top_freqs = [freq for word, freq in word_freq.most_common(10)]

# Create the bar chart
plt.figure(figsize=(10, 10))
plt.bar(top_words, top_freqs)
plt.xlabel('Words')
plt.ylabel('Frequencies')
plt.title('Top 10 Word Frequencies in NonVerified Accounts')
plt.xticks(rotation=45)
plt.show()
```



[ ]:
```
df_english['verification_status'] = df_english['user'].apply(lambda x:␣
↪'verified' if isinstance(x, dict) and x.get('verified') else 'non-verified')
```

[36]:
```
import pandas as pd
import matplotlib.pyplot as plt
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')

# Instantiate the VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Calculate sentiment scores for each text in df['content']
sentiment_scores = df_english['content_preprocessed'].apply(lambda x: sia.
↪polarity_scores(x)['compound'])

# Calculate word frequency
word_freq = df_english['content_preprocessed'].str.split().apply(len)

# Create the scatter plot
plt.scatter(word_freq, sentiment_scores)

# Customize the scatter plot
plt.title("Relationship between Word Frequency and Sentiment")
plt.xlabel("Word Frequency")
plt.ylabel("Sentiment")
plt.grid(True)

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

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\dadashza\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```



[119]:
```
df_english['verification_status'].value_counts()
```

[119]:
```
verification_status
non-verified    1143745
verified          77280
Name: count, dtype: int64
```

[5]:
```
import pandas as pd

# Randomly select 16355 rows from the "verified" users
df_verified = df_english[df_english['verification_status'] == "verified"].
 ↪sample(n=818)
# Randomly select 16355 rows from the "non-verified" users
df_non_verified = df_english[df_english['verification_status'] ==␣
 ↪"non-verified"].sample(n=15537)

# Concatenate the two dataframes to form the final dataframe with the desired␣
 ↪proportion
df = pd.concat([df_verified, df_non_verified])
```

```
# Shuffle the rows of the final dataframe
df = df.sample(frac=1).reset_index(drop=True)
```

[ ]:
```
df.head()
```

0.1 Machine Learning Algorithms to Detect Bots

[ ]:
```
from sklearn import svm
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Convert preprocessed text data into numerical feature vectors
tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(df['content_preprocessed'])

# Train the One-Class SVM model
ocsvm_model = svm.OneClassSVM()
ocsvm_model.fit(X)

# Make predictions on the same feature vectors
predictions = ocsvm_model.predict(X)

# Map the numerical predictions to labels
labels = ['Normal Instance (Unverified User)' if p == 1 else 'Anomaly Detected'␣
 ↪for p in predictions]

# Update the labels based on the verification status
for index, row in df.iterrows():
    if row['verification_status'] == 'verified':
        labels[index] = 'Normal Instance (Verified User)'

# Create a new column 'anomaly_label' in the DataFrame with the predicted␣
 ↪anomaly labels
df['anomaly_label'] = labels

# Print the instances and their anomaly labels
for index, row in df.iterrows():
    print(f"Instance {index + 1}: {row['content_preprocessed']} | Anomaly Label:
 ↪ {row['anomaly_label']}")
```

[21]:
```
df.anomaly_label.value_counts()
```

[21]:
```
anomaly_label
Anomaly Detected                      7903
Normal Instance (Unverified User)     7634
Normal Instance (Verified User)        818
```

```
Name: count, dtype: int64
```

[27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16355 entries, 0 to 16354
Data columns (total 40 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   _type                          16355 non-null  object
 1   url                            16355 non-null  object
 2   date                           16355 non-null  object
 3   content                        16355 non-null  object
 4   renderedContent                16355 non-null  object
 5   id                             16355 non-null  int64
 6   user                           16355 non-null  object
 7   replyCount                     16355 non-null  int64
 8   retweetCount                   16355 non-null  int64
 9   likeCount                      16355 non-null  int64
 10  quoteCount                     16355 non-null  int64
 11  conversationId                 16355 non-null  int64
 12  lang                           16355 non-null  object
 13  source                         16355 non-null  object
 14  sourceUrl                      16355 non-null  object
 15  sourceLabel                    16355 non-null  object
 16  outlinks                       5298 non-null   object
 17  tcooutlinks                    5298 non-null   object
 18  media                          1758 non-null   object
 19  retweetedTweet                 0 non-null      float64
 20  quotedTweet                    1559 non-null   object
 21  inReplyToTweetId               7649 non-null   float64
 22  inReplyToUser                  7649 non-null   object
 23  mentionedUsers                 8677 non-null   object
 24  coordinates                    199 non-null    object
 25  place                          199 non-null    object
 26  hashtags                       3776 non-null   object
 27  cashtags                       46 non-null     object
 28  Searh                          16355 non-null  object
 29  language                       16355 non-null  object
 30  content_cleaned                16355 non-null  object
 31  content_preprocessed           16355 non-null  object
 32  verification_status            16355 non-null  object
 33  followers_count                16355 non-null  int64
 34  verification_status_encoded    16355 non-null  int32
 35  date_numeric                   16355 non-null  int32
 36  kmeans_cluster_label           16355 non-null  int32
 37  anomaly_label                  16355 non-null  object
```

```
 38  anomaly_score               16355 non-null  float64
 39  anomaly_features            16355 non-null  object
dtypes: float64(3), int32(3), int64(7), object(27)
memory usage: 4.8+ MB
```

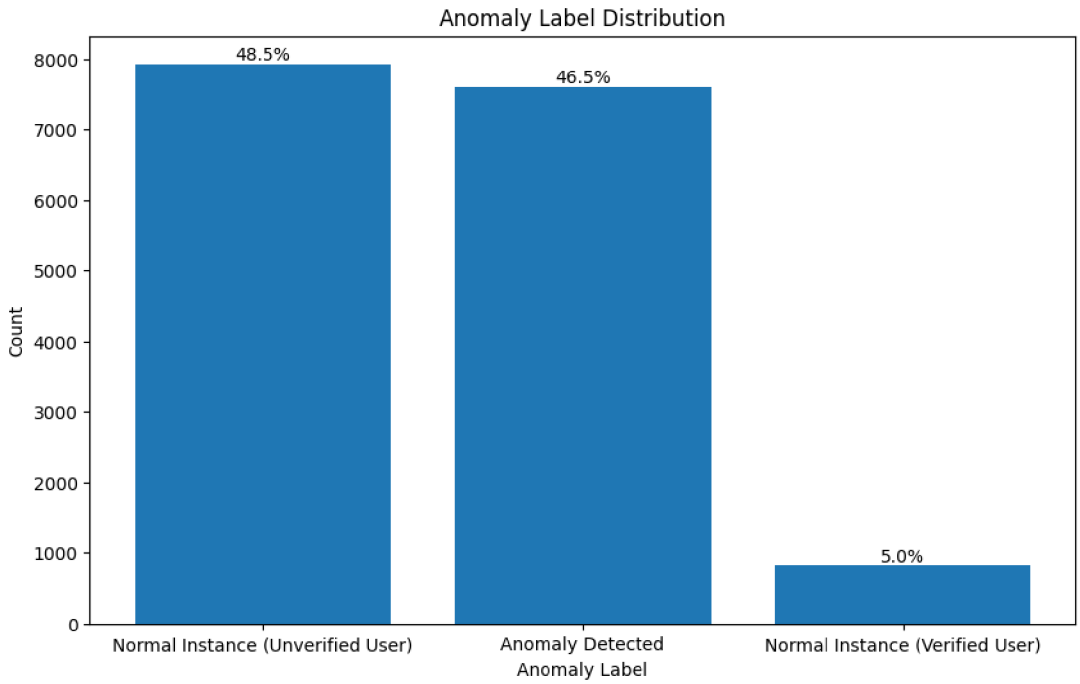
[ ]:
```python
from sklearn import svm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from scipy.sparse import hstack
import pandas as pd

# Convert preprocessed text data into numerical feature vectors
tfidf_vectorizer = TfidfVectorizer()
X_text = tfidf_vectorizer.fit_transform(df['content_preprocessed'])

# Encode the 'verification_status' column as numerical labels
label_encoder = LabelEncoder()
df['verification_status_encoded'] = label_encoder.
 ↪fit_transform(df['verification_status'])

# Convert the 'date' column to numeric representation
df['date_numeric'] = pd.to_datetime(df['date']).astype('int64')

# Select the 'followers_count', 'verification_status_encoded', and␣
 ↪'date_numeric' columns as additional features
X_additional = df[['followers_count', 'verification_status_encoded']].values

# Concatenate the TF-IDF matrix and the additional features
X_combined = hstack([X_text, X_additional])

# Train the One-Class SVM model
ocsvm_model = svm.OneClassSVM()
ocsvm_model.fit(X_combined)

# Make predictions on the same feature vectors
predictions = ocsvm_model.predict(X_combined)

# Map the numerical predictions to labels
labels = ['Normal Instance (Unverified User)' if p == 1 else 'Anomaly Detected'␣
 ↪for p in predictions]

# Update the labels based on the verification status
for index, row in df.iterrows():
    if row['verification_status'] == 'verified':
        labels[index] = 'Normal Instance (Verified User)'
```

```
# Create a new column 'anomaly_label' in the DataFrame with the predicted␣
 ↪anomaly labels
df['anomaly_label'] = labels

# Print the instances and their anomaly labels
for index, row in df.iterrows():
    print(f"Instance {index + 1}: {row['content_preprocessed']} | Anomaly Label:
 ↪ {row['anomaly_label']}")
```

[118]:
```
df.anomaly_label.value_counts()
```

[118]:
```
anomaly_label
Normal Instance (Unverified User)    7929
Anomaly Detected                     7608
Normal Instance (Verified User)       818
Name: count, dtype: int64
```

[119]:
```
import matplotlib.pyplot as plt

# Calculate the value counts of 'anomaly_label' column
label_counts = df['anomaly_label'].value_counts()

# Calculate the percentages
percentages = label_counts / len(df) * 100

# Create a wider figure
plt.figure(figsize=(10, 6))  # Adjust the width (first value) as desired

# Plot the value counts as a bar chart
bars = plt.bar(label_counts.index, label_counts)
plt.xlabel('Anomaly Label')
plt.ylabel('Count')
plt.title('Anomaly Label Distribution')

# Add percentages to the bars
for bar, percentage in zip(bars, percentages):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height,
             f'{percentage:.1f}%', ha='center', va='bottom')

plt.show()
```



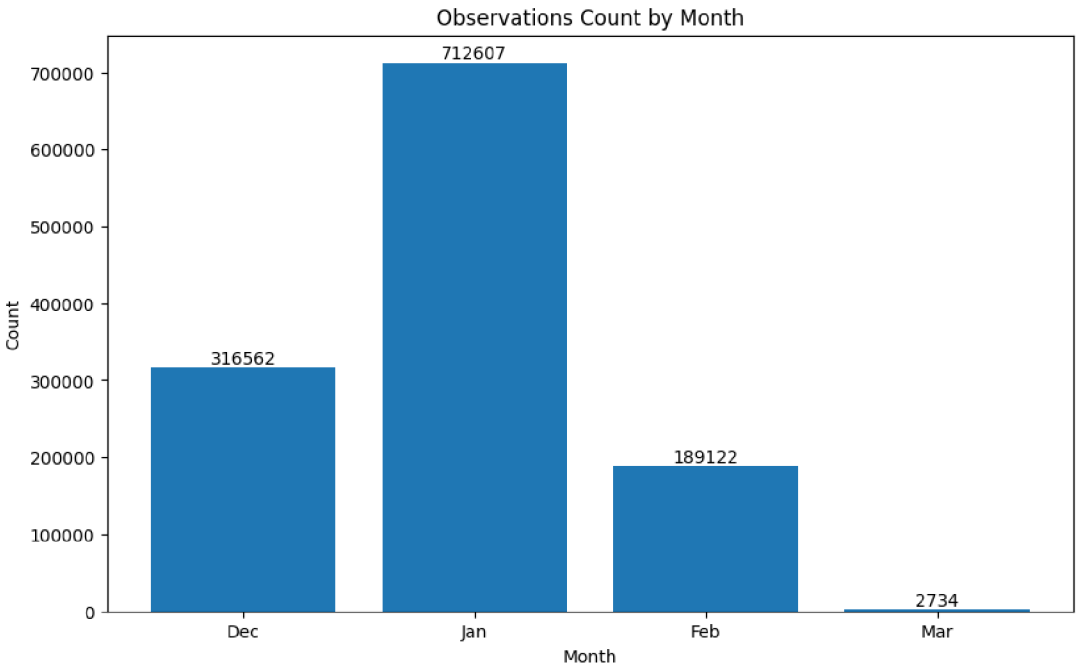
[90]:
```
df.date
```

[90]:
```
0        2022-02-13 12:31:31+00:00
1        2022-02-07 00:30:03+00:00
2        2022-03-02 21:18:54+00:00
3        2022-02-28 18:09:38+00:00
4        2022-02-10 12:19:05+00:00
                    ...
16350    2022-01-26 18:21:51+00:00
16351    2022-01-15 17:05:30+00:00
16352    2022-03-05 21:42:42+00:00
16353    2022-02-26 21:30:54+00:00
16354    2022-02-04 15:20:56+00:00
Name: date, Length: 16355, dtype: object
```

[ ]:
```
import pandas as pd

# Set the maximum column width to None for displaying entire content
pd.set_option('display.max_colwidth', None)

# Assuming 'df' is your dataframe
anomaly_detected_content = df[df['anomaly_label'] == 'Normal Instance (Verified␣
↪User)']['content'].head(50)
```

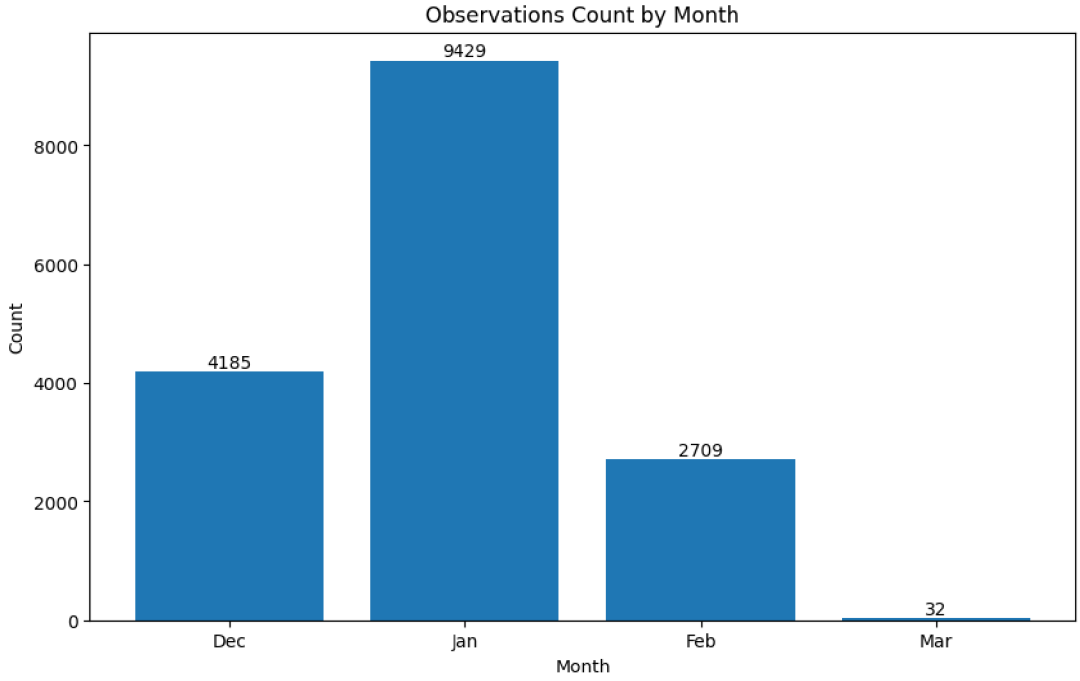
```
# Display the sentences with the entire content
print(anomaly_detected_content)
```

[111]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Convert 'date' column to datetime format
df_english['date'] = pd.to_datetime(df_english['date'])

# Group the DataFrame by month and count the observations
monthly_counts = df_english.groupby(df_english['date'].dt.month)['date'].count()

# Create a wider figure
plt.figure(figsize=(10, 6))  # Adjust the width (first value) as desired

# Define the numerical x-axis values as an array of [0, 1, 2, 3]
x = np.arange(len(monthly_counts))

# Plot the monthly counts as a bar chart using the numerical x-axis values
bars = plt.bar(x, monthly_counts)
plt.xticks(x, ['Dec', 'Jan', 'Feb', 'Mar'])  # Customize the x-axis labels
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Observations Count by Month')

# Add the count values above each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height,
             f'{int(height)}', ha='center', va='bottom')

plt.show()
```



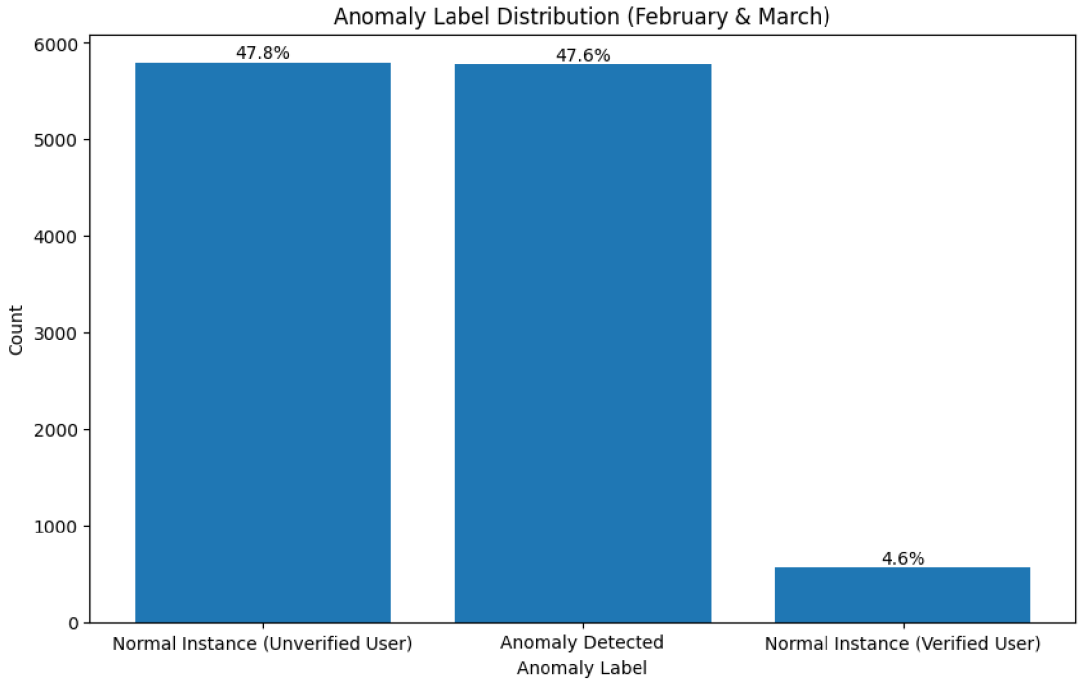
[99]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Convert 'date' column to datetime format
df['date'] = pd.to_datetime(df['date'])

# Group the DataFrame by month and count the observations
monthly_counts = df.groupby(df['date'].dt.month)['date'].count()

# Create a wider figure
plt.figure(figsize=(10, 6))  # Adjust the width (first value) as desired

# Define the numerical x-axis values as an array of [0, 1, 2, 3]
x = np.arange(len(monthly_counts))

# Plot the monthly counts as a bar chart using the numerical x-axis values
bars = plt.bar(x, monthly_counts)
plt.xticks(x, ['Dec', 'Jan', 'Feb', 'Mar'])  # Customize the x-axis labels
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Observations Count by Month')

# Add the count values above each bar
```

```
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height,
             f'{int(height)}', ha='center', va='bottom')

plt.show()
```



[133]:
```
import matplotlib.pyplot as plt

# Filter the DataFrame for December and January
df_filtered = df[(df['date'].dt.month == 2) | (df['date'].dt.month == 3)]

# Calculate the value counts of 'anomaly_label' column in the filtered DataFrame
label_counts = df_filtered['anomaly_label'].value_counts()

# Calculate the percentages
percentages = label_counts / len(df_filtered) * 100

# Create a wider figure
plt.figure(figsize=(10, 6))  # Adjust the width (first value) as desired

# Plot the value counts as a bar chart
bars = plt.bar(label_counts.index, label_counts)
plt.xlabel('Anomaly Label')
```

```
plt.ylabel('Count')
plt.title('Anomaly Label Distribution (February & March)')

# Add percentages to the bars
for bar, percentage in zip(bars, percentages):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height,
             f'{percentage:.1f}%', ha='center', va='bottom')

plt.show()
```



[ ]:
```
os.getcwd()
```

0.2 ISOLATION FORREST

[ ]:
```
from sklearn.ensemble import IsolationForest
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Convert preprocessed text data into numerical feature vectors
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['content_preprocessed'])

# Train the Isolation Forest model
```

```
isolation_forest = IsolationForest()
isolation_forest.fit(X)

# Predict the anomaly scores
anomaly_scores = isolation_forest.decision_function(X)

# Set a threshold to classify instances as bot or non-bot
threshold = -0.5  # Adjust the threshold based on your data and requirements

# Create a new column 'bot_label' in the DataFrame with the predicted labels
df['bot_label'] = ['Bot' if score < threshold else 'Non-Bot' for score in␣
 ↪anomaly_scores]

# Print the instances and their bot labels
for index, row in df.iterrows():
    print(f"Instance {index + 1}: {row['content_preprocessed']} | Bot Label:␣
 ↪{row['bot_label']}")
```

[34]:
```
df.bot_label.value_counts()
```

[34]:
```
bot_label
Non-Bot    16355
Name: count, dtype: int64
```

0.3 KMEANS

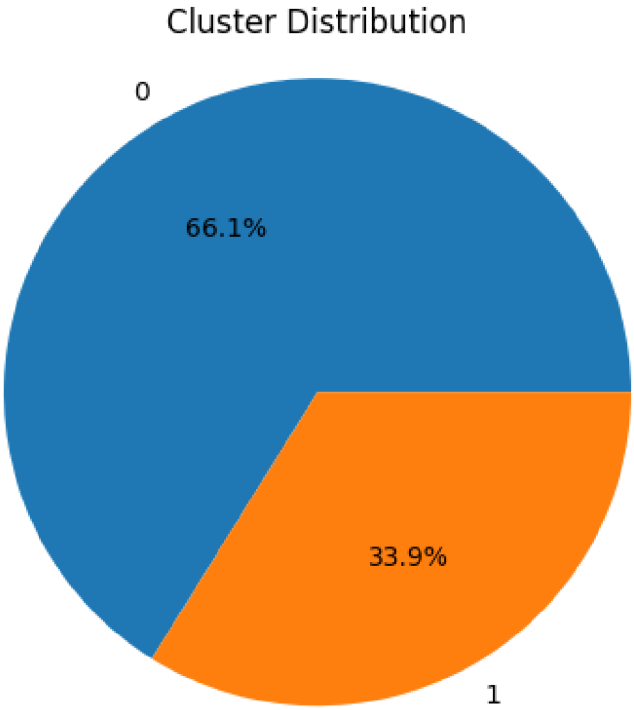
[ ]:
```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# Assuming your dataframe is named 'df' and contains the columns␣
 ↪'content_preprocessed', 'date', and 'verification_status'

# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Apply TF-IDF vectorization on 'content_preprocessed' column
X_text = vectorizer.fit_transform(df['content_preprocessed'])

# Initialize and train the K-Means clustering model
model = KMeans(n_clusters=2)  # Adjust the number of clusters based on your␣
 ↪dataset
model.fit(X_text)

# Get the cluster labels
predicted_labels = model.labels_
```

```
# Create a new column 'cluster_label' in the dataframe with the predicted␣
 ↪cluster labels
df['kmeans_cluster_label'] = predicted_labels

# Print the instances and their cluster labels
for data, label in zip(df['content_preprocessed'], predicted_labels):
    print(f"Cluster {label}: {data}")
```

[66]:
```
df.kmeans_cluster_label.value_counts()
```

[66]:
```
kmeans_cluster_label
0    10818
1     5537
Name: count, dtype: int64
```

[67]:
```
import matplotlib.pyplot as plt

# Calculate the value counts of 'kmeans_cluster_label' column
cluster_counts = df['kmeans_cluster_label'].value_counts()

# Plot the value counts as a pie chart
plt.pie(cluster_counts, labels=cluster_counts.index, autopct='%1.1f%%')
plt.title('Cluster Distribution')
plt.axis('equal')
plt.show()
```



[ ]:
```
import pandas as pd

# Set the maximum column width to None for displaying entire content
pd.set_option('display.max_colwidth', None)

# Assuming 'df' is your dataframe
df[df['kmeans_cluster_label'] == 1]['content'].head(50)
```

[ ]:
```
df.head()
```

[ ]:
```
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from scipy.sparse import hstack
import pandas as pd

# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Apply TF-IDF vectorization on 'content_preprocessed' column
X_text = vectorizer.fit_transform(df['content_preprocessed'])
```

```
# Encode the 'verification_status' column as numerical labels
label_encoder = LabelEncoder()
df['verification_status_encoded'] = label_encoder.
 ↪fit_transform(df['verification_status'])

# Select the 'date_numeric' and 'verification_status_encoded' columns
X_additional = df[['verification_status_encoded', "date_numeric",␣
 ↪"followers_count"]].values

# Combine the TF-IDF matrix and the additional columns
X_combined = hstack([X_text, X_additional])

# Initialize and train the K-Means clustering model
model = KMeans(n_clusters=2)  # Adjust the number of clusters based on your␣
 ↪dataset
model.fit(X_combined)

# Get the cluster labels
predicted_labels = model.labels_

# Create a new column 'cluster_label' in the dataframe with the predicted␣
 ↪cluster labels
df['kmeans_feature'] = predicted_labels

# Print the instances and their cluster labels
for data, label in zip(df['content_preprocessed'], predicted_labels):
    print(f"Cluster {label}: {data}")
```

[65]:
```
df['kmeans_feature'].value_counts()
```

[65]:
```
kmeans_feature
0    16336
1       19
Name: count, dtype: int64
```

[ ]:
```
df[df['kmeans_cluster_label'] == 1]['content_preprocessed']
```

[78]:
```
df['kmeans_cluster_label'].value_counts()
```

[78]:
```
kmeans_cluster_label
0    10818
1     5537
Name: count, dtype: int64
```

[ ]:
```
df['classification_svm'].value_counts()
```

[68]:
```
# Filter the data based on conditions
filtered_data = df[(df['classification_svm'] == 'Anomaly Detected')]
```

[ ]:
```
filtered_data.content.head(30)
```

[83]:
```
# Save the DataFrame as a CSV file
df.to_csv('data_processed.csv', index=False)
```

[88]:
```
# Find date period
print(df['date'].min())
print(df['date'].max())
```

```
2021-12-31 00:16:53+00:00
2022-03-05 23:59:19+00:00
```

[ ]:
```
df.head()
```