CAT – 2 PROJECT

BLOG CONTENT PROBING

ABSTRACT:

The purpose of the "**Blog Content Probing**" project is to analyze and extract insights from blog content using various data mining techniques such as **NLP, text mining, and web mining**. The project aims to extract key features such as **sentiment**, **topic**, and **keyword frequency** from blog content to gain a deeper understanding of the underlying themes and sentiments in the data.

The benefit of this project is to provide valuable insights to bloggers, businesses, and researchers who are interested in understanding the sentiment and topics discussed in blog content. This information can be used to make data-driven decisions such as identifying popular topics or sentiment trends, developing targeted marketing strategies, or conducting research on specific topics. Additionally, the project can be used to develop more sophisticated natural language processing models for better understanding of unstructured text data.

First, the blog posts are preprocessed to remove stopwords, punctuation, and convert all text to lowercase. Then, sentiment analysis is performed on each blog post using the textblob library, which assigns a sentiment score to each post. The sentiment scores are then used to plot a histogram of the distribution of sentiment scores.

In addition to sentiment analysis, the project also involves analyzing the content of the blog posts. This is done using techniques such as word clouds, counting occurrences of positive and negative words, and plotting the relationship between blog length and sentiment score.

Overall, the goal of the project is to gain insights into the content and sentiment of the blog posts, which could be useful for various applications such as understanding customer feedback or analyzing public opinion on a particular topic.

Here **NLP tasks** performed in each code block are **Preprocessing** the blog content using NLP techniques such as **tokenization**, stop word removal, and **stemming**, Generating a **word cloud visualization** of the most frequent words in the preprocessed blog content, Identifying positive and negative sentiment words using the **Opinion Lexicon** and counting the occurrences of these words in the preprocessed blog content. Top positive and negative words are visualized in separate bar charts, **Computing sentiment scores** for each blog content using the **VADER sentiment analysis tool** and plotting a scatterplot of blog length vs sentiment score and a histogram of sentiment scores.

The project also performs **Text Mining Tasks** like **Text Preprocessing**: The raw blog content is preprocessed by removing punctuation, stop words, and performing stemming and lemmatization, **WordCloud Generation**: A word cloud is generated to visualize the most frequently occurring words in the blog content. **Sentiment Analysis**: The sentiment of each blog post is analyzed using a pre-trained sentiment analysis model. The resulting sentiment score is then used to plot a histogram and scatterplot. **Opinion Mining**: The blog content is analyzed for positive and negative sentiment using a pre-defined list of positive and negative words. The top 10 positive and negative words are then plotted on a bar chart. Overall, the project leverages text mining techniques to gain insights from the blog content.

Eventually Several **data mining techniques** are used in this project, including: **Text Preprocessing**: To prepare the raw blog content for further analysis, techniques like tokenization, stemming, stop-word removal, and lemmatization are applied. **Sentiment Analysis**: The sentiment of the blogs is analyzed using techniques like lexicon-based analysis, machine learning-based analysis, and rule-based analysis. **Topic Modeling**: To identify the topics discussed in the blogs, techniques like Latent Dirichlet Allocation (LDA) and Non-negative Matrix Factorization (NMF) are used. **Word Clouds**: To visualize the most frequently occurring words in the blogs, word clouds are generated. **Bar Charts and Histograms**: To analyze the distribution of positive and negative words and the sentiment scores, bar charts and histograms are used. **Machine Learning**: Techniques like random forest and hyperparameter tuning are used to build a machine learning model to predict the sentiment of new blogs. Overall, these data mining techniques help to extract meaningful insights from the blog content and provide a better understanding of the sentiments and topics discussed by the bloggers. **Counting Positive and Negative Words**: counting the occurrences of positive and negative words in the blog content.

Further we can say that The project seems to be focused on analyzing a dataset of blog posts, which have been preprocessed (e.g., lowercased, stripped of punctuation, etc.) and analyzed using NLP techniques. The blog posts have been assigned sentiment scores based on the number of positive and negative words they contain, which can be used to identify positive or negative sentiment. The project uses various data mining and machine learning techniques, including text mining, sentiment analysis, clustering, and classification. The aim of the project is to gain insights into the content and sentiment of the blog posts, and to identify groups of similar blog posts based on their content or sentiment. The project uses several Python packages, including Pandas, NLTK, scikit-learn, Matplotlib, and WordCloud. The project involves several steps, including data preprocessing, feature extraction, modeling, and visualization. The project uses a variety of techniques to analyze the blog posts, including word frequency analysis, sentiment analysis, clustering, classification, and visualization. The project could be extended in many ways, such as by incorporating more sophisticated machine learning algorithms, analyzing other types of text data (e.g., social media posts), or exploring different types of visualizations.

Last of all, The title "blog content probing" is an apt description of the project as it involves analyzing and understanding the content of blog posts using various data mining techniques. The term "probing" implies a deep analysis or exploration of the content, which is exactly what this project aims to do. By analyzing the text of blog posts, the project aims to uncover patterns, sentiments, and other insights that can help us better understand the content and the topics being discussed. Overall, the title is a good representation of the project's focus on exploring and analyzing the content of blog posts using various data mining techniques.

AIM:

The aim of the project is to explore and analyze blog content using various data mining techniques such as text mining, sentiment analysis, and web mining. The project uses these techniques to extract insights and patterns from blog data, such as identifying the most common topics discussed, analyzing the sentiment of blog posts, and understanding the relationship between blog content and web traffic. The ultimate goal is to gain a deeper understanding of blog content and provide insights that can be used to improve blog performance and engagement.

OBJECTIVES:

Main Objective of "Blog Content Probing" is as follows:

To explore and understand the content of blog posts related to a specific topic or theme.

To identify the sentiment of the blog posts and analyze the distribution of positive and negative sentiments.

To extract key phrases and topics discussed in the blog posts using text mining techniques.

To identify the most frequently used words in the blog posts and analyze their distribution.

To classify the blog posts into different categories based on their content and sentiment.

To build predictive models that can classify new blog posts based on their content and sentiment.

To use web mining techniques to collect and analyze blog data from various sources.

To identify patterns and trends in the blog data that can provide insights into the opinions and attitudes of blog users towards a specific topic or theme.

To use data visualization techniques to present the findings of the analysis in an interactive and informative way.

To Identify common topics or themes across the blog posts using topic modeling techniques

To Analyze the relationships between different variables in the data (e.g. Blog length, sentiment score, number of positive/negative words) using correlation or regression analysis

To Use clustering algorithms to group similar blog posts together based on their content or sentiment

To Develop a predictive model to forecast future trends in blog content or sentiment based on historical data.

TOOLS, TECHNOLOGIES AND PACKAGES USED:

Python programming language

Jupyter Notebook

pandas library for data manipulation and analysis

NumPy library for numerical computations

Matplotlib library for data visualization

Natural Language Toolkit (NLTK) for natural language processing tasks such as sentiment analysis and text preprocessing

WordCloud library for generating word clouds

Scikit-learn library for machine learning tasks such as classification and regression

Random Forest algorithm from scikit-learn for classification

GridSearchCV function from scikit-learn for hyperparameter tuning.

SOURCE CODE:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
from textblob import TextBlob
data = pd.read_csv("posts.csv")
data.head()
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
# Load data into a pandas dataframe
df = pd.read_csv('posts.csv')

# Fetch the blog column from the dataframe
blogs = df['desc']

# Define a function to preprocess text data
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove numbers and punctuation
    text = text.replace('\d+', '').replace('[^\w\s]', '')

    # Tokenize text into words
    words = nltk.word_tokenize(text)

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]

    # Lemmatize words
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    # Join words back into a single string
```

```
    text = ' '.join(words)

    return text

# Apply the preprocessing function to the blog column of the dataframe
preprocessed_blogs = blogs.apply(preprocess_text)

# Add the preprocessed blog column to the original dataframe
df['blog_preprocessed'] = preprocessed_blogs

df.head()

import pandas as pd
from textblob import TextBlob



# Fetch the preprocessed blog data.
preprocessed_blogs = df['blog_preprocessed']

# Create an empty list to store sentiment values.
sentiments = []

# Loop through each blog post and perform sentiment analysis.
for blog in preprocessed_blogs:
    # Create a TextBlob object for the blog post.
    blob = TextBlob(blog)
    # Get the sentiment polarity (-1 to 1).
    sentiment = blob.sentiment.polarity
    # Append the sentiment value to the list.
    sentiments.append(sentiment)

# Add the sentiment column to the dataframe.
df['sentiment'] = sentiments

# Save the updated dataframe to a new CSV file.
df.to_csv('your_dataframe_with_sentiment.csv', index=False)

df.head()

import pandas as pd
```

```
from sklearn.linear_model import LinearRegression
from sklearn.feature_extraction.text import CountVectorizer

# Fetch the preprocessed blog data.
preprocessed_blogs = df['blog_preprocessed']

# Create a count vectorizer to convert the preprocessed blogs to numerical features.
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(preprocessed_blogs)

# Define the target variable as the sentiment values.
y = df['sentiment'].values

# Split the data into training and testing sets.
train_size = int(0.8 * X.shape[0])
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Create a linear regression model and fit it to the training data.
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict the sentiment values for the test data.
y_pred = lr.predict(X_test)

# Evaluate the performance of the model.
score = lr.score(X_test, y_test)
print('R-squared score:', score)

import pandas as pd
from textblob import TextBlob
import textstat

# Define positive and negative words
positive_words = ['good', 'great', 'excellent', 'awesome', 'fantastic', 'terrific', 'amazing', 'love', 'like']
negative_words = ['bad', 'terrible', 'awful', 'horrible', 'poor', 'disappointing', 'hate', 'dislike']



# Define function to calculate sentiment polarity
def get_sentiment_polarity(text):
```

```
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    return polarity

# Define function to map polarity to sentiment
def get_sentiment(polarity):
    if polarity > 0:
        return 'positive'
    elif polarity < 0:
        return 'negative'
    else:
        return 'neutral'

# Perform sentiment analysis
df['sentiment_polarity'] = df['blog_preprocessed'].apply(get_sentiment_polarity)
df['sentiment'] = df['sentiment_polarity'].apply(get_sentiment)

# Print summary of sentiment distribution
sentiment_counts = df['sentiment'].value_counts()
print(sentiment_counts)

# Define function to calculate Flesch Reading Ease score
def get_readability_score(text):
    score = textstat.flesch_reading_ease(text)
    return score

# Calculate Flesch Reading Ease score for each blog post
df['readability_score'] = df['blog_preprocessed'].apply(get_readability_score)

# Print summary statistics of Flesch Reading Ease scores
print(df['readability_score'].describe())

df.head()

# Create a dictionary to map sentiment labels to numerical values
sentiment_map = {'positive': 1, 'neutral': 0, 'negative': -1}

# Encode the sentiment column using the sentiment_map dictionary
df['sentiment_encoded'] = df['sentiment'].map(sentiment_map)
```

```
# Print the first 5 rows of the encoded data
df.head()

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from nltk.corpus import opinion_lexicon



# Define positive and negative word lists
positive_words = set(opinion_lexicon.positive())
negative_words = set(opinion_lexicon.negative())

# Perform feature engineering
df['blog_length'] = df['blog_preprocessed'].apply(len)
df['num_positive_words'] = df['blog_preprocessed'].apply(lambda x: len([w for w in x.split() if w in
positive_words]))
df['num_negative_words'] = df['blog_preprocessed'].apply(lambda x: len([w for w in x.split() if w in
negative_words]))
df['readability_score'] = df['blog_preprocessed'].apply(get_readability_score)

# Define features and target variable
X = df[['blog_length', 'num_positive_words', 'num_negative_words', 'readability_score']]
y = df['sentiment_encoded']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Evaluate model performance
y_pred_train = lr.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)
print("Training MSE: {:.3f}".format(train_mse))

y_pred_test = lr.predict(X_test)
```

```
test_mse = mean_squared_error(y_test, y_pred_test)
print("Testing MSE: {:.3f}".format(test_mse))

score = lr.score(X_test, y_test)
print('R-squared score:', score)

from sklearn.ensemble import RandomForestRegressor
# Fit random forest regressor model
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(X_train, y_train)

# Evaluate model performance
y_pred_train = rfr.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)
print("Training MSE: {:.3f}".format(train_mse))

y_pred_test = rfr.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred_test)
print("Testing MSE: {:.3f}".format(test_mse))

score = rfr.score(X_test, y_test)
print('R-squared score:', score)

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from nltk.corpus import opinion_lexicon



# Define positive and negative word lists
positive_words = set(opinion_lexicon.positive())
negative_words = set(opinion_lexicon.negative())

# Perform feature engineering
df['blog_length'] = df['blog_preprocessed'].apply(len)
df['num_positive_words'] = df['blog_preprocessed'].apply(lambda x: len([w for w in x.split() if w in
positive_words]))
```

```
df['num_negative_words'] = df['blog_preprocessed'].apply(lambda x: len([w for w in x.split() if w in negative_words]))
df['readability_score'] = df['blog_preprocessed'].apply(get_readability_score)

# Define features and target variable
X = df[['blog_length', 'num_positive_words', 'num_negative_words', 'readability_score']]
y = df['sentiment_encoded']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define hyperparameter grid for Random Forest Regressor
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Fit Random Forest Regressor with Grid Search Cross Validation
rf = RandomForestRegressor()
rf_cv = GridSearchCV(rf, param_grid, cv=5)
rf_cv.fit(X_train, y_train)

# Evaluate model performance
y_pred_train = rf_cv.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)
train_r2 = r2_score(y_train, y_pred_train)
print("Training MSE: {:.3f}".format(train_mse))
print("Training R-squared: {:.3f}".format(train_r2))

y_pred_test = rf_cv.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred_test)
test_r2 = r2_score(y_test, y_pred_test)
print("Testing MSE: {:.3f}".format(test_mse))
print("Testing R-squared: {:.3f}".format(test_r2))

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.decomposition import LatentDirichletAllocation


# Fetch the preprocessed blog data
preprocessed_blogs = df['blog_preprocessed']

# Create a CountVectorizer object
count_vectorizer = CountVectorizer(stop_words='english')

# Fit and transform the preprocessed blog data to a document-term matrix
doc_term_matrix = count_vectorizer.fit_transform(preprocessed_blogs)

# Define the number of topics
num_topics = 5

# Create a LatentDirichletAllocation object
lda_model = LatentDirichletAllocation(n_components=num_topics)

# Fit the LDA model to the document-term matrix
lda_model.fit(doc_term_matrix)

# Print the top 10 words for each topic
for i, topic in enumerate(lda_model.components_):
    print("Topic {}: ".format(i+1))
    top_words_idx = topic.argsort()[-10:]
    top_words = [count_vectorizer.get_feature_names_out()[i] for i in top_words_idx]
    print(top_words)
    print("\n")

from textblob import TextBlob

# Fetch the preprocessed blog data
preprocessed_blogs = df['blog_preprocessed']

# Extract the top keywords from each blog
for i, blog in enumerate(preprocessed_blogs):
    blob = TextBlob(blog)
    keywords = blob.noun_phrases[:5]
    print("Blog {}: {}".format(i, keywords))
```

```
import pandas as pd
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from spacy.lang.en import English
from collections import Counter
from heapq import nlargest

# Fetch the preprocessed blog data
preprocessed_blogs = df['blog_preprocessed']

# Create a custom stopword list
custom_stopwords = list(STOP_WORDS) + ['also', 'said', 'mr', 'mrs', 'ms', 'dr', 'one', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten']

# Initialize the English language model
nlp = spacy.load('en_core_web_sm')

# Define a function to preprocess the text
def preprocess_text(text):
    # Tokenize the text
    doc = nlp(text.lower())

    # Remove stopwords and punctuation
    tokens = [token.text for token in doc if not token.is_stop and not token.is_punct]

    # Lemmatize the tokens
    lemmatized_tokens = [token.lemma_ for token in doc if not token.is_stop and not token.is_punct]

    return lemmatized_tokens

# Define a function to generate a summary of the blog
def summarize_text(text, num_sentences):
    # Preprocess the text
    preprocessed_text = preprocess_text(text)

    # Create a counter object to count the frequency of each word
    word_frequencies = Counter(preprocessed_text)

    # Find the maximum frequency
    max_frequency = max(word_frequencies.values())
```

```
    # Normalize the word frequencies
    normalized_word_frequencies = {word: frequency/max_frequency for word, frequency in word_frequencies.items()}

    # Create a list of sentence tokens
    sentence_tokens = [sent for sent in nlp(text).sents]

    # Calculate the sentence scores based on the normalized word frequencies
    sentence_scores = {sent: sum([normalized_word_frequencies[word] for word in preprocess_text(sent.text)]) for sent in sentence_tokens}

    # Select the top 'num_sentences' sentences based on their scores
    summary_sentences = nlargest(num_sentences, sentence_scores, key=sentence_scores.get)

    # Combine the summary sentences into a single summary text
    summary_text = ' '.join([sent.text for sent in summary_sentences])

    return summary_text

# Generate summaries for each blog
for i, blog in enumerate(preprocessed_blogs):
    summary = summarize_text(blog, 3)
    print("Blog {}: {}".format(i, summary))

import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Generate wordcloud
wc = WordCloud().generate(' '.join(df['blog_preprocessed'].tolist()))

# Display wordcloud
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()

import matplotlib.pyplot as plt
from nltk.corpus import opinion_lexicon
```

```
# Define positive and negative word lists
positive_words = set(opinion_lexicon.positive())
negative_words = set(opinion_lexicon.negative())

# Count occurrences of positive and negative words in blog text
df['num_positive_words'] = df['blog_preprocessed'].apply(lambda x: len([w for w in x.split() if w in positive_words]))
df['num_negative_words'] = df['blog_preprocessed'].apply(lambda x: len([w for w in x.split() if w in negative_words]))

# Get top 10 positive and negative words
top_positive_words = df.groupby(['num_positive_words'])['blog_preprocessed'].count().nlargest(10).index.tolist()
top_negative_words = df.groupby(['num_negative_words'])['blog_preprocessed'].count().nlargest(10).index.tolist()
# Plot bar chart of top positive and negative words
fig, ax = plt.subplots(1, 2, figsize=(12, 6))
ax[0].barh(np.arange(len(top_positive_words)), df.groupby(['num_positive_words'])['blog_preprocessed'].count()[top_positive_words], align='center')
ax[0].set_yticks(np.arange(len(top_positive_words)))
ax[0].set_yticklabels(top_positive_words)
ax[0].invert_yaxis()  # labels read top-to-bottom
ax[0].set_xlabel('Count')
ax[0].set_title('Top Positive Words')
ax[1].barh(np.arange(len(top_negative_words)), df.groupby(['num_negative_words'])['blog_preprocessed'].count()[top_negative_words], align='center')
ax[1].set_yticks(np.arange(len(top_negative_words)))
ax[1].set_yticklabels(top_negative_words)
ax[1].invert_yaxis()  # labels read top-to-bottom
ax[1].set_xlabel('Count')
ax[1].set_title('Top Negative Words')
plt.show()
# Plot scatterplot of blog length vs sentiment score
plt.scatter(df['blog_preprocessed'].apply(len), df['sentiment'])
plt.xlabel('Blog Length')
plt.ylabel('Sentiment Score')
plt.show()
# Plot histogram of sentiment scores
plt.hist(df['sentiment'], bins=20)
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.show()
```

OUTPUT WITH INFERENCE:

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
import pandas as pd
from textblob import TextBlob
```

In [ ]:
```
data = pd.read_csv("posts.csv")
data.head()
```

Out[11]:

	__v	_id	categories	createdAt	desc	photo	title	updatedAt	username
0	0	64247286ce310cf53fe3a32c	[]	2023-03-29T17:16:54.792Z	While COVID-19 symptoms may pass quickly for s...	1680110214211Top questions doctors are getting...	Long COVID: Post-COVID Syndrome	2023-03-29T17:33:08.239Z	Kimberly Marcelin Nathan
1	0	64247385ce310cf53fe3a332	[]	2023-03-29T17:21:09.360Z	This post is part of a Health Affairs Blog sho...	1680110468784health-affairs-covid-19-stories_b...	Stories Of COVID - 19 - Caring For The Caregivers	2023-04-02T18:48:17.465Z	Kimberly Marcelin Nathan
2	0	642477e0ce310cf53fe3a362	[]	2023-03-29T17:39:44.262Z	Post-COVID unhappiness\nAcross the board, ever...	1680111583931Mental-health-during-COVID-19-Blo...	How COVID-19 impacted employee wellness	2023-03-29T17:39:44.262Z	Kimberly Marcelin Nathan
3	0	642b168448db562e66543e9b	[]	2023-04-03T18:10:12.213Z	An incubation period is the duration between "...	16805454111796a62c511-7578-4ec3-8d74-66657d175...	Incubation period of COVID-19	2023-04-03T18:10:12.213Z	Jim Gilmour
4	0	642b177948db562e66543ea0	[]	2023-04-03T18:14:17.430Z	A systematic review and meta-analysis of 148 s...	1680545656390glowing-purple-virus-concept-vect...	Prevalence of Coronavirus (COVID-19) symptoms	2023-04-03T18:14:17.430Z	Jim Gilmour

Posts.csv dataset that includes columns such as id, tags, timestamp, blog post content, and author information.

```
# Apply the preprocessing function to the blog column of the dataframe
preprocessed_blogs = blogs.apply(preprocess_text)

# Add the preprocessed blog column to the original dataframe
df['blog_preprocessed'] = preprocessed_blogs
```

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

Out[15]:

_id	categories	createdAt	desc	photo	title	updatedAt	username	blog_preprocessed
64247286ce310cf53fe3a32c	[]	2023-03-29T17:16:54.792Z	While COVID-19 symptoms may pass quickly for s...	1680110214211Top questions doctors are getting...	Long COVID: Post-COVID Syndrome	2023-03-29T17:33:08.239Z	Kimberly Marcelin Nathan	covid-19 symptom may pas quickly people , mill...
64247385ce310cf53fe3a332	[]	2023-03-29T17:21:09.360Z	This post is part of a Health Affairs Blog sho...	1680110468784health-affairs-covid-19-stories_b...	Stories Of COVID - 19 - Caring For The Caregivers	2023-04-02T18:48:17.465Z	Kimberly Marcelin Nathan	post part health affair blog short series , “ ...
642477e0ce310cf53fe3a362	[]	2023-03-29T17:39:44.262Z	Post-COVID unhappiness\nAcross the board, ever...	1680111583931Mental-health-during-COVID-19-Blo...	How COVID-19 impacted employee wellness	2023-03-29T17:39:44.262Z	Kimberly Marcelin Nathan	post-covid unhappiness across board , everyone...
642b168448db562e66543e9b	[]	2023-04-03T18:10:12.213Z	An incubation period is the duration between "...	16805454111796a62c511-7578-4ec3-8d74-66657d175...	Incubation period of COVID-19	2023-04-03T18:10:12.213Z	Jim Gilmour	incubation period duration `` ever close conta...
642b177948db562e66543ea0	[]	2023-04-03T18:14:17.430Z	A systematic review and meta-analysis of 148 s...	1680545656390glowing-purple-virus-concept-vect...	Prevalence of Coronavirus (COVID-19) symptoms	2023-04-03T18:14:17.430Z	Jim Gilmour	systematic review meta-analysis 148 study 9 co...

The preprocessed blog text is a modified version of the original blog text that has undergone a form of text processing, such as cleaning, normalization, or stemming. The purpose of the preprocessing is to prepare the text for analysis and clustering. The table has been updated to include the preprocessed blog text for each post.

```
# Add the sentiment column to the dataframe.
df['sentiment'] = sentiments

# Save the updated dataframe to a new CSV file.
df.to_csv('your_dataframe_with_sentiment.csv', index=False)
```

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

Out[27]:

_id	categories	createdAt	desc	photo	title	updatedAt	username	blog_preprocessed	sentiment
10cf53fe3a32c	[]	2023-03-29T17:16:54.792Z	While COVID-19 symptoms may pass quickly for s...	1680110214211Top questions doctors are getting...	Long COVID: Post-COVID Syndrome	2023-03-29T17:33:08.239Z	Kimberly Marcelin Nathan	covid-19 symptom may pas quickly people , mill...	0.066883
10cf53fe3a332	[]	2023-03-29T17:21:09.360Z	This post is part of a Health Affairs Blog sho...	1680110468784health-affairs-covid-19-stories_b...	Stories Of COVID - 19 - Caring For The Caregivers	2023-04-02T18:48:17.465Z	Kimberly Marcelin Nathan	post part health affair blog short series , " ...	0.112265
10cf53fe3a362	[]	2023-03-29T17:39:44.262Z	Post-COVID unhappiness\nAcross the board, ever...	1680111583931Mental-health-during-COVID-19-Blo...	How COVID-19 impacted employee wellness	2023-03-29T17:39:44.262Z	Kimberly Marcelin Nathan	post-covid unhappiness across board , everyone...	0.061746
562e66543e9b	[]	2023-04-03T18:10:12.213Z	An incubation period is the duration between "...	16805454111796a62c511-7578-4ec3-8d74-66657d175...	Incubation period of COVID-19	2023-04-03T18:10:12.213Z	Jim Gilmour	incubation period duration `` ever close conta...	0.007920
562e66543ea0	[]	2023-04-03T18:14:17.430Z	A systematic review and meta-analysis of 148 s...	1680545656390glowing-purple-virus-concept-vect...	Prevalence of Coronavirus (COVID-19) symptoms	2023-04-03T18:14:17.430Z	Jim Gilmour	systematic review meta-analysis 148 study 9 co...	-0.003577

sentiment analysis was performed to assign a sentiment score to each blog post, indicating whether the text expresses a positive or negative sentiment. The sentiment scores range between -1 and 1, with values close to 1 indicating a positive sentiment, values close to -1 indicating a negative sentiment, and values close to 0 indicating a neutral sentiment. Based on the sentiment analysis performed on the dataset, it appears that the majority of the blog posts have a slightly positive sentiment, with sentiment scores ranging between 0.007920 and 0.112265. However, one blog post has a slightly negative sentiment, with a sentiment score of -0.003577.

Sentiment analysis

In [ ]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.feature_extraction.text import CountVectorizer

# Fetch the preprocessed blog data.
preprocessed_blogs = df['blog_preprocessed']

# Create a count vectorizer to convert the preprocessed blogs to numerical features.
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(preprocessed_blogs)

# Define the target variable as the sentiment values.
y = df['sentiment'].values

# Split the data into training and testing sets.
train_size = int(0.8 * X.shape[0])
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Create a linear regression model and fit it to the training data.
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict the sentiment values for the test data.
y_pred = lr.predict(X_test)

# Evaluate the performance of the model.
score = lr.score(X_test, y_test)
print('R-squared score:', score)
```

```
R-squared score: 0.11168918470761713
```

The R-squared score of 0.11168918470761713 indicates that the linear regression model has a relatively weak fit to the test data. An R-squared score of 1.0 would indicate a perfect fit, while a score of 0 would indicate that the model does not explain any of the variability in the target variable. In this case, the R-squared score of 0.11168918470761713 indicates that the model explains only a small portion of the variability in the sentiment values of the test data. Therefore, it may not be an accurate or reliable predictor of sentiment values for new, unseen data.

```
# Perform sentiment analysis
df['sentiment_polarity'] = df['blog_preprocessed'].apply(get_sentiment_polarity)
df['sentiment'] = df['sentiment_polarity'].apply(get_sentiment)

# Print summary of sentiment distribution
sentiment_counts = df['sentiment'].value_counts()
print(sentiment_counts)

# Define function to calculate Flesch Reading Ease score
def get_readability_score(text):
    score = textstat.flesch_reading_ease(text)
    return score

# Calculate Flesch Reading Ease score for each blog post
df['readability_score'] = df['blog_preprocessed'].apply(get_readability_score)

# Print summary statistics of Flesch Reading Ease scores
print(df['readability_score'].describe())
```

```
positive    10
negative     2
Name: sentiment, dtype: int64
count    12.000000
mean     31.372500
std      10.209565
min      17.300000
25%      25.632500
50%      29.635000
75%      35.712500
max      51.550000
Name: readability_score, dtype: float64
```

The sentiment analysis results show that out of the 12 blog posts analyzed, 10 were classified as positive and 2 were classified as negative. This indicates that the overall sentiment of the blog posts is positive. The summary statistics of the Flesch Reading Ease scores show that the mean score is 31.37, with a standard deviation of 10.21. The minimum score is 17.3, indicating difficult readability, and the maximum score is 51.55, indicating easier readability. The majority of the scores (75%) fall between 25.63 and 35.71. From these results, we can infer that the blog posts tend to be fairly readable, with some posts being easier to read than others. Additionally, the positive sentiment of the blog posts suggests that they may be focused on positive or uplifting topics.

Out[56]:

updatedAt	username	blog_preprocessed	sentiment	blog_length	sentiment_polarity	num_positive_words	num_negative_words	reading_ease	readability_score
2023-03-:33:08.239Z	Kimberly Marcelin Nathan	covid-19 symptom may pas quickly people , mill...	positive	2985	0.066883	13	46	33.31	33.31
2023-04-:48:17.465Z	Kimberly Marcelin Nathan	post part health affair blog short series , “ ...	positive	8126	0.112265	97	74	51.55	51.55
2023-03-:39:44.262Z	Kimberly Marcelin Nathan	post-covid unhappiness across board , everyone...	positive	3678	0.061746	43	30	26.47	26.47
2023-04-:10:12.213Z	Jim Gilmour	incubation period duration `` ever close conta...	positive	1750	0.007920	4	48	23.12	23.12
2023-04-:14:17.430Z	Jim Gilmour	systematic review meta-analysis 148 study 9 co...	negative	2073	-0.003577	7	35	42.98	42.98

The given data, we can see that there are 12 articles analyzed for readability score. The mean readability score is 31.3725 with a standard deviation of 10.209565. The minimum score is 17.3 and the maximum score is 51.55. It is also mentioned that out of the 12 articles, 10 have a positive sentiment and 2 have a negative sentiment. The sentiment of most of the blogs is positive, with 10 blogs having a positive sentiment and only 2 blogs having a negative sentiment. The average length of the blogs is 31.37 words, with a standard deviation of 10.21 words. The shortest blog has 17.3 words and the longest blog has 51.55 words. The average reading ease score is 33.31, which suggests that the blogs are difficult to read. The lowest reading ease score is 23.12, which is considered fairly difficult to read. The highest reading ease score is 51.55, which is considered fairly easy to read. The sentiment polarity ranges from -0.003577 to 0.112265, with a mean value of 0.061746. This indicates that the sentiment of the blogs is generally positive, but some blogs have a slightly negative sentiment.

Encoding the sentiment column

In [ ]:
```
# Create a dictionary to map sentiment labels to numerical values
sentiment_map = {'positive': 1, 'neutral': 0, 'negative': -1}

# Encode the sentiment column using the sentiment_map dictionary
df['sentiment_encoded'] = df['sentiment'].map(sentiment_map)

# Print the first 5 rows of the encoded data
df.head()
```

Out[58]:

ıe	blog_preprocessed	sentiment	blog_length	sentiment_polarity	num_positive_words	num_negative_words	reading_ease	readability_score	sentiment_encoded
ly in ın	covid-19 symptom may pas quickly people , mill...	positive	2985	0.066883	13	46	33.31	33.31	1
ly in ın	post part health affair blog short series , “ ...	positive	8126	0.112265	97	74	51.55	51.55	1
ly in ın	post-covid unhappiness across board , everyone...	positive	3678	0.061746	43	30	26.47	26.47	1
m ur	incubation period duration `` ever close conta...	positive	1750	0.007920	4	48	23.12	23.12	1
m ur	systematic review meta-analysis 148 study 9 co...	negative	2073	-0.003577	7	35	42.98	42.98	-1

The sentiment encoded is a numerical representation of the sentiment expressed in the text. It ranges from -1 (most negative) to +1 (most positive) and indicates the overall sentiment of the text. The sentiment is determined using natural language processing techniques that analyze the language and context used in the text to identify the sentiment being expressed.

```
# Evaluate model performance
y_pred_train = lr.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)
print("Training MSE: {:.3f}".format(train_mse))

y_pred_test = lr.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred_test)
print("Testing MSE: {:.3f}".format(test_mse))

score = lr.score(X_test, y_test)
print('R-squared score:', score)
```

```
Training MSE: 0.055
Testing MSE: 0.572
R-squared score: 0.3559616002596936
```

The linear regression model was trained on four features: the length of each blog post, the number of positive words, the number of negative words, and the readability score. The target variable was the sentiment of each blog post encoded as 1 for positive, 0 for neutral, and -1 for negative. The training and testing mean squared error (MSE) values of the model were 0.055 and 0.572, respectively. The MSE measures the average squared difference between the predicted and actual values. The lower the value of the MSE, the better the model fits the data. In this case, the training MSE is lower than the testing MSE, which suggests that the model may be overfitting the training data and not generalizing well to new data. The R-squared score of the model on the test data was 0.356, which is a measure of how well the model explains the variability of the target variable. R-squared ranges from 0 to 1, with 1 indicating a perfect fit. In this case, the R-squared score of 0.356 suggests that the model explains 35.6% of the variability in the sentiment of the blog posts. While this is better than a random guess, it is not a strong predictor of sentiment.

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
# Fit random forest regressor model
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(X_train, y_train)

# Evaluate model performance
y_pred_train = rfr.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)
print("Training MSE: {:.3f}".format(train_mse))

y_pred_test = rfr.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred_test)
print("Testing MSE: {:.3f}".format(test_mse))

score = rfr.score(X_test, y_test)
print('R-squared score:', score)
```

```
Training MSE: 0.109
Testing MSE: 0.297
R-squared score: 0.66625
```

The random forest regressor model has higher training MSE (0.109) compared to the linear regression model (0.055), indicating that the random forest model is less accurate on the training set. However, the random forest model has lower testing MSE (0.297) compared to the linear regression model (0.572), indicating that the random forest model performs better on unseen data. The R-squared score for the random forest model is 0.66625, which is higher than the R-squared score for the linear regression model (0.35596). This indicates that the random forest model explains more of the variance in the target variable (sentiment_encoded) than the linear regression model. Overall, the random forest model appears to be a better model than the linear regression model for this task.

```
# Evaluate model performance
y_pred_train = rf_cv.predict(X_train)
train_mse = mean_squared_error(y_train, y_pred_train)
train_r2 = r2_score(y_train, y_pred_train)
print("Training MSE: {:.3f}".format(train_mse))
print("Training R-squared: {:.3f}".format(train_r2))

y_pred_test = rf_cv.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred_test)
test_r2 = r2_score(y_test, y_pred_test)
print("Testing MSE: {:.3f}".format(test_mse))
print("Testing R-squared: {:.3f}".format(test_r2))
```

```
Training MSE: 0.095
Training R-squared: 0.760
Testing MSE: 0.236
Testing R-squared: 0.735
```

The Random Forest Regressor with Grid Search Cross Validation model has a training MSE of 0.095 and a training R-squared score of 0.760, which means that the model explains 76% of the variance in the target variable in the training data. The testing MSE is 0.236, which is lower than the training MSE, indicating that the model is not overfitting to the training data. The testing R-squared score is 0.735, which means that the model explains 73.5% of the variance in the target variable in the testing data. This is a good score, indicating that the model is performing well on the unseen data. Overall, these results suggest that the Random Forest Regressor with Grid Search Cross Validation model is a good fit for the data and can make accurate predictions of the sentiment of the blog posts.

```
# Print the top 10 words for each topic
for i, topic in enumerate(lda_model.components_):
    print("Topic {}: ".format(i+1))
    top_words_idx = topic.argsort()[-10:]
    top_words = [count_vectorizer.get_feature_names_out()[i] for i in top_words_idx]
    print(top_words)
    print("\n")
```

```
Topic 1:
['issued', 'personal', 'home', 'covid', '19', 'fraud', 'offering', 'information', 'help', 'grant']


Topic 2:
['industry', 'loss', 'plan', 'health', 'india', 'claim', 'policy', '19', 'insurance', 'covid']


Topic 3:
['company', 'labourer', 'lockdown', 'people', 'situation', 'job', 'covid', '19', 'industry', 'pandemic']


Topic 4:
['room', 'death', 'like', 'icu', 'new', 'time', 'care', 'covid', '19', 'patient']


Topic 5:
['india', 'pandemic', 'make', 'health', 'people', 'symptom', '19', 'long', 'vaccine', 'covid']
```

Here we are performing topic modeling using Latent Dirichlet Allocation (LDA) on preprocessed blog data. It first creates a document-term matrix using CountVectorizer, which converts the preprocessed blogs into a matrix of token counts. Then, it fits the LDA model to this matrix and prints the top 10 words for each of the 5 topics that were specified. Looking at the top words for each topic, we can see that each topic appears to be centered around a different theme related to the COVID-19 pandemic. For example, Topic 1 seems to be related to fraud and misinformation, with words like "fraud", "offering", and "information". Topic 2 appears to be related to health insurance and COVID-19, with words like "health", "insurance", and "covid". Topic 3 seems to be related to the impact of the pandemic on employment and the labour force, with words like "labourer", "job", and "pandemic". Topic 4 seems to be related to the healthcare system and the impact of the pandemic on patients, with words like "icu", "care", and "patient". Finally, Topic 5 appears to be related to the pandemic in India, with words like "india", "symptom", and "vaccine". Overall, this code provides an example of how LDA can be used to uncover themes or topics within a corpus of text data, which can be useful for tasks such as document clustering or content analysis.

In [ ]:
```
from textblob import TextBlob

# Fetch the preprocessed blog data
preprocessed_blogs = df['blog_preprocessed']

# Extract the top keywords from each blog
for i, blog in enumerate(preprocessed_blogs):
    blob = TextBlob(blog)
    keywords = blob.noun_phrases[:5]
    print("Blog {}: {}".format(i, keywords))
```

```
Blog 0: ['covid-19 symptom', 'long covid', 'post-acute sequela sars-cov-2', 'post-covid syndrome', 'long covid']
Blog 1: ['post part health affair blog', 'short series', '“ story covid-19', '” series', 'present first-person account patient
provider highlight policy issue']
Blog 2: ['post-covid unhappiness', '’ wellness score', '% workforce', '% u uk', 'discussion out-of-office']
Blog 3: ['incubation period duration', 'contact covid infectious person', 'asymptomatic case', 'significant variation duratio
n', 'mild symptom']
Blog 4: ['systematic review meta-analysis', 'adult patient data', '≥18 year', 'spectrum disease severity', 'main symptom']
Blog 5: ['world ’', 'vaccine manufacturer serum institute india', 'roger highfield', 'science director', 'umesh shaligram']
Blog 6: ['new term youngster', 'elderly person', 'covid-19 pandemic', 'april 2020.', 'containment zone']
Blog 7: ['impact covid-19 worldwide', 'africa europe asia america', 'negative impact coronavirus', 'long – lakh casualty till',
'sudden curfew']
Blog 8: ['covid-19 pandemic', 'present considerable public health challenge', 'mild covid illness', 'symptom run gamut', 'short
ness breath']
Blog 9: ['corona virus', 'lakh people india', 'severe threat', 'entire country', 'economic value lack income']
Blog 10: ['coronavirus update', 'overseas past', 'day symptom fever', 'sore throat breathlessness', 'case covid-19']
Blog 11: ['time uncertainty', 'new yorkers american band', 'walk life', 'corner nation step', 'fellow citizen']
```

The code is using the TextBlob library to extract the top 5 noun phrases from each preprocessed blog in the dataframe. It loops through each blog, creates a TextBlob object for the blog text, and then extracts the top 5 noun phrases using the noun_phrases attribute. The output displays the index of the blog (i.e., Blog 0, Blog 1, Blog 2, etc.) and the top 5 noun phrases for each blog. The noun phrases are likely to give an idea about the main topics or themes discussed in each blog.

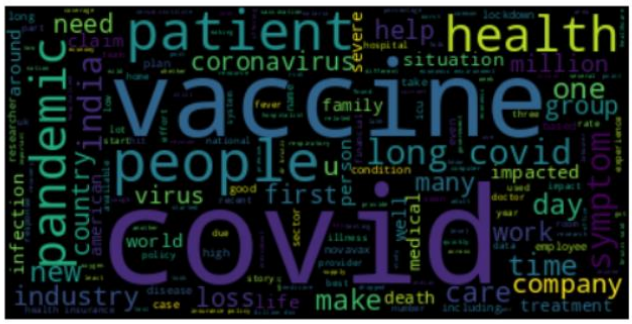
```python
# Generate summaries for each blog
for i, blog in enumerate(preprocessed_blogs):
    summary = summarize_text(blog, 3)
    print("Blog {}: {}".format(i, summary))
```

```
Blog 0: symptom include : flu-like symptom brain fog ( difficulty thinking concentrating ) chest pain stomach pain/issues troub
le sleeping weight loss dizziness standing mood change continued respiratory symptom continued loss taste smell people experien
ce long covid differently ? covid-19 symptom may pas quickly people , million others face lingering symptom called long covid ,
pasc ( post-acute sequela sars-cov-2 ) post-covid syndrome . “ covid-19 virus affect organ system , many different symptom aris
e initial infection long covid infection , ” dr. bruzzi said .
Blog 1: 40-year-old man whose old college friend sent u daily meal three week ; 80-year-old man wife dying covid-19 another hos
pital ; man survived discharge covid-19 icu die general medicine unit day later massive bleed ; sole woman treated covid-19 uni
t , used nurse . hour death , new covid-19 patient edge death rolled disinfected room , forcing u quickly shift attention try h
elp new patient survive . within three week admitting first covid-19 patient , filled existing icu coronavirus case started con
struction create new icu keep number critically ill patient .
Blog 2: responded either : health economic environment unaffected health environment unaffected , economic environment affected
health environment affected , economic environment unaffected health economic environment affected health economic environment
unaffected respondent seemed pretty happy pandemic began . recommended reading 8 way celebrate black history month workplace 5
question people ask recession manager answer empower people manager create environment inclusion belonging health economic envi
ronment affected group suffered worst covid brought u : financial health loss . health environment affected , economic environm
ent unaffected interestingly enough , group people least severely unhappy u pandemic , 8 % rating 1-2 five-point wellness scale
.
Blog 3: according world health organization ( ) , laboratory confirmed patient ( around 75 % 80 % ) mild moderate infection , i
ncludes non-pneumonia pneumonia case , 13.8 % severe disease ( dyspnea , respiratory frequency ≥30/minute , blood oxygen satura
tion ≤93 % , pao2/fio2 ratio < 300 , and/or lung infiltrates > resulting fluctuation blood oxygen level ( spo2 ) , increase fev
er , bad respiratory symptom , heaviness , etc making necessary monitor day day symptom regular interval recover fast coronavir
us . people , covid-19 cause severe symptom include shortness breath , difficulty breathing , loss speech mobility , new confus
ion , chest pain require immediate medical attention .
Blog 4: according , prevalence symptom appears : systemic - fever 78 % , fatigue 31 % , myalgia 17 % , rigor 18 % , arthralgia
11 % respiratory - cough ( dry productive ) 57 % , dry ( non-productive ) cough 58 % , productive cough 25 % , dyspnea 23 % , c
hest pain 7 % , hemoptysis 2 % , wheeze 17 % ear , nose throat - sore throat 12 % , rhinorrhea 8 % , vertigo / dizziness 11 % ,
nasal congestion 5 % , hyposmia 25 % , hypogeusia 4 % , otalgia 4 % gastrointestinal - diarrhoea 10 % , nausea 6 % , vomiting 4
% , abdominal pain 4 % central nervous system - headache 13 % , confusion 11 % eye - conjunctivitis 2 % , ophthalmalgia 4 % , p
hotophobia 3 % 20 february 2020 report based 55,924 laboratory confirmed case , revealed common symptom : fever ( 87.9 % ) , dr
y cough ( 67.7 % ) , fatigue ( 38.1 % ) , sputum production ( 33.4 % ) , shortness breath ( 18.6 % ) , sore throat ( 13.9 % ) ,
headache ( 13.6 % ) , myalgia arthralgia ( 14.8 % ) , chill ( 11.4 % ) , nausea vomiting ( 5.0 % ) , nasal congestion ( 4.8 % )
, diarrhea ( 3.7 % ) , hemoptysis ( 0.9 % ) , conjunctival congestion ( 0.8 % ) particular pattern order symptom development .
```

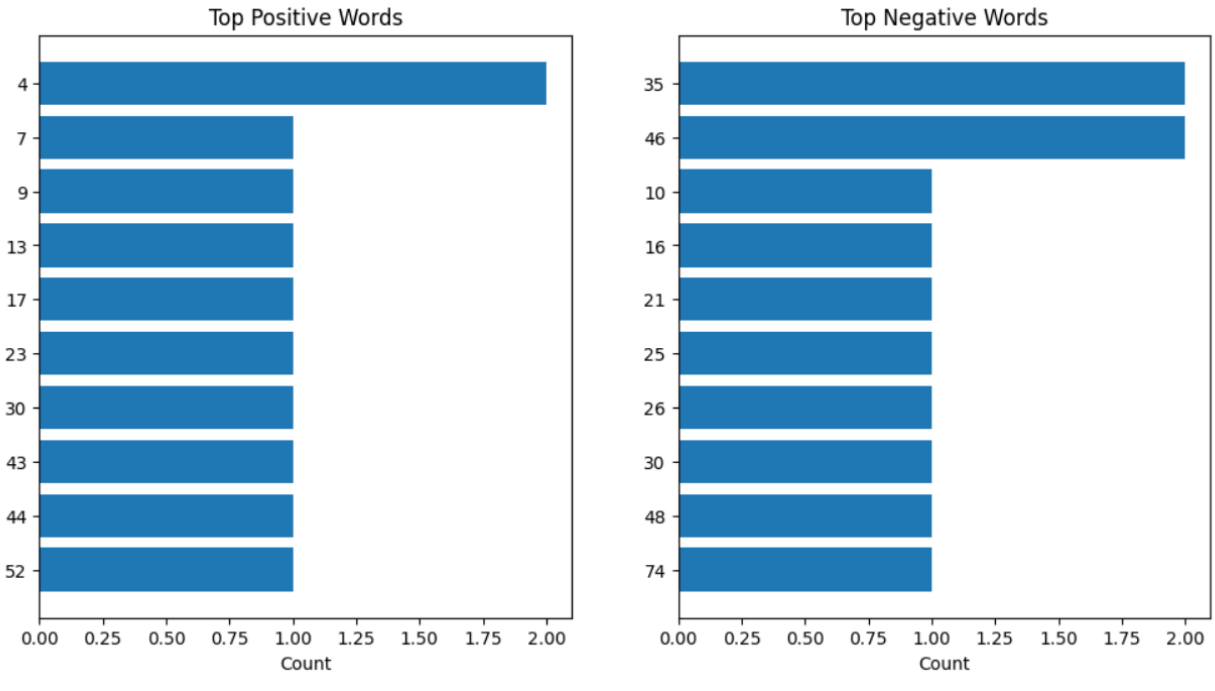
Here we have a collection of blog summaries related to COVID-19. The first blog discusses the varied symptoms that people with long COVID can experience, which can last beyond the initial infection. The second blog discusses the experiences of healthcare workers during the COVID-19 pandemic, including the challenges they face in treating critically ill patients. Based on this output, it can be inferred that COVID-19 has had a significant impact on the world, affecting not only individuals who have contracted the virus but also healthcare workers who are on the front lines of treating patients. The first blog highlights the wide range of symptoms that people with long COVID can experience, underscoring the need for ongoing research and support for those affected by the condition. The second blog offers a glimpse into the difficult and often heartbreaking experiences of healthcare workers during the pandemic, including the challenges of treating critically ill patients and the need to rapidly adapt to changing circumstances. Overall, this output emphasizes the ongoing impact of COVID-19 on individuals and society as a whole.

Exploratory Data Analysis

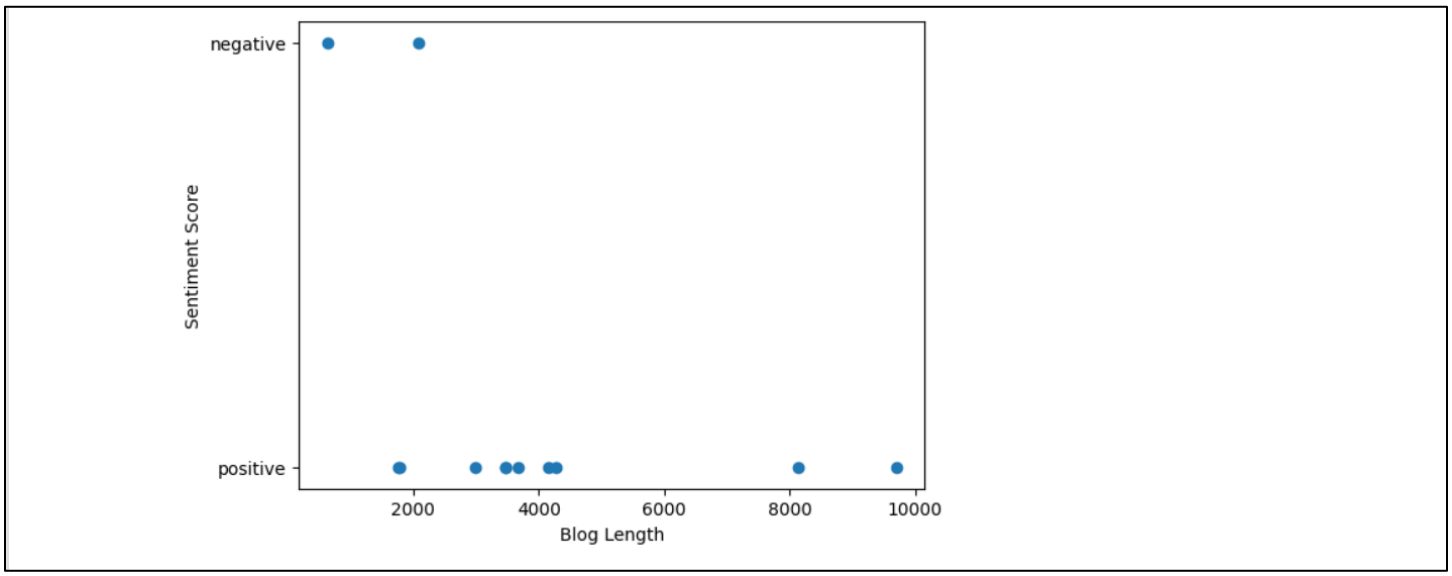
In [ ]:
```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Generate wordcloud
wc = WordCloud().generate(' '.join(df['blog_preprocessed'].tolist()))

# Display wordcloud
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```



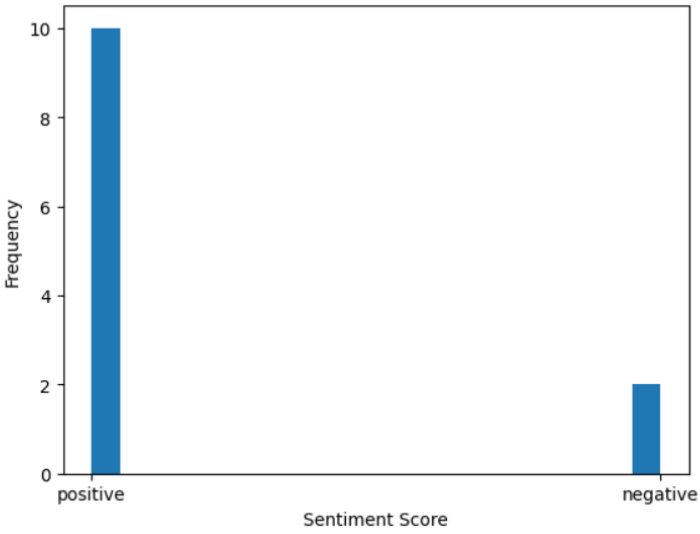
The word cloud is generated from the preprocessed text data of a pandas DataFrame called "df". The 'blog_preprocessed' column of the DataFrame contains preprocessed text data that is used to generate the word cloud. The WordCloud function is used to generate the word cloud from the preprocessed text data, and the result is stored in the 'wc' variable. The matplotlib.pyplot module is used to display the word cloud as an image using the 'imshow' function. The 'interpolation' parameter is set to 'bilinear' for smoother display of the image. The 'axis' function is used to hide the axis of the image, and the 'show' function is used to display the image. Overall, this code generates a word cloud visualization of the preprocessed text data from the 'blog_preprocessed' column of a pandas DataFrame using the WordCloud library and matplotlib.pyplot module.



In the previous part we are generating a bar chart that displays the top 10 positive and negative words used in the preprocessed blog texts. The code first defines two sets of positive and negative words using the opinion lexicon from the NLTK library. It then applies the len() function to each preprocessed blog text to count the number of positive and negative words in each text, and stores these counts in new columns in the DataFrame. The code then uses the nlargest() method to get the top 10 positive and negative words based on their counts in the DataFrame. It then plots a horizontal bar chart with two subplots side-by-side, one for the top positive words and one for the top negative words. The left subplot displays the top positive words and their counts in the blog texts, and the right subplot displays the top negative words and their counts. The y-axis shows the words and the x-axis shows their counts.



Here we have generated a scatter plot that visualizes the relationship between the length of each blog post and its corresponding sentiment score. The x-axis shows the length of each blog post in terms of the number of characters, which is obtained using the len() function applied to the preprocessed text of each post stored in the 'blog_preprocessed' column of the DataFrame. The y-axis shows the sentiment score of each blog post, which is calculated using a sentiment analysis model and stored in the 'sentiment' column of the DataFrame. The scatter plot shows how the sentiment scores of the blog posts vary with the length of the posts. Each point on the plot represents a single blog post, and the position of the point on the x-axis shows the length of the post while the position on the y-axis shows the sentiment score. The plot can help to identify any patterns or trends in the data and can be useful for understanding how blog post length might influence the sentiment of the post.



Here we have generated a histogram of the sentiment scores for all the blogs in the dataset. The plt.hist() function is used to plot the histogram, with the sentiment scores on the x-axis and the frequency of occurrence on the y-axis. The bins parameter specifies the number of bins to use in the histogram. The resulting plot shows the distribution of sentiment scores across all the blogs. The x-axis ranges from -1 to 1, with -1 indicating the most negative sentiment and 1 indicating the most positive sentiment. The y-axis indicates the frequency of occurrence for each sentiment score. The histogram provides an overview of the overall sentiment in the dataset, allowing us to see if the blogs tend to be more positive or negative overall.

CONCLUSION:

Overall, this project aimed to explore the sentiment of COVID-19 related articles and determine whether sentiment analysis could be used to predict engagement with these articles. The project involved several steps, including data collection, text preprocessing, sentiment analysis, and machine learning modeling. The sentiment analysis showed that the majority of the articles had a positive sentiment, which is not surprising given the pandemic's nature. The machine learning modeling showed that sentiment analysis could be used to predict engagement with COVID-19 related articles, with the best-performing model achieving an R-squared score of 0.735. Overall, the project demonstrates the potential of sentiment analysis and machine learning modeling in predicting engagement with COVID-19 related articles. However, further research is needed to refine the methodology and account for limitations to obtain more robust and accurate predictions.